Leitura de planilhas de xadrez manuscritas usando redes neurais com mecanismos de atenção

Sergio Yuji Hayashi

Dissertação apresentada ao
Instituto de Matemática e Estatística
da Universidade de São Paulo
para obtenção do título de
Mestre em Ciências

Programa: Ciência da Computação
Orientadora: Profª. Drª. Nina S. T. Hirata

São Paulo
20 de Dezembro de 2021

Leitura de planilhas de xadrez manuscritas usando redes neurais com mecanismos de atenção

Sergio Yuji Hayashi

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 20 de Dezembro de 2021.

Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profª. Drª. Nina S. T. Hirata (orientadora) - IME-USP
- Prof. Dr. Roberto de Alencar Lotufo - UNICAMP
- Prof. Dr. Byron Leite Dantas Bezerra - UPE

Agradecimentos

> *I have always been convinced that the only way to get artificial intelligence to work is to do the computation in a way similar to the human brain. That is the goal I have been pursuing. We are making progress, though we still have lots to learn about how the brain actually works.*
>
> — Geoffrey Hinton

Ao Departamento de Ciência da Computação do IME/USP e todo o corpo docente. À orientadora Nina S. T. Hirata por todo o apoio durante o período do programa de mestrado, pelas sábias opiniões e pelas intermináveis revisões.

À Opus Software, em especial ao diretor Francisco Elias Barguil, pelo apoio e compreensão sem os quais não seria possível conciliar o trabalho com a pesquisa.

Ao MF Adriano Caldeira e André Salama pelo fornecimento das planilhas de xadrez.

Aos meus pais Paulo e Olga pela educação que me deram. À minha esposa Yuki e meus filhos Kaito e Kouki, porque foi na companhia deles em casa que passei quase todos os feriados e finais de semana nestes últimos 3 anos me dedicando ao mestrado, pelo qual demonstraram total compreensão, interesse e apoio.

Resumo

Sergio Yuji Hayashi. **Leitura de planilhas de xadrez manuscritas usando redes neurais com mecanismos de atenção**. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

O reconhecimento de texto manuscrito continua sendo um problema em aberto, objeto de intensa pesquisa na área de aprendizado de máquina. Neste projeto focamos numa categoria específica de problema nesta área, a leitura automática de planilhas de xadrez. Planilhas de xadrez contém anotações de lances de jogos escritos à mão pelos próprios jogadores num formato chamado de notação algébrica. Em comparação com um texto tradicional em linguagem natural, planilhas de xadrez são formulários de formato fixo, seu conteúdo textual é restrito a um vocabulário reduzido e a escrita em geral não é totalmente cursiva. Mesmo assim, elas ainda apresentam uma alta variabilidade de estilos de escrita à mão, tornando a sua leitura um problema suficientemente complexo. O objetivo deste trabalho é o treinamento ponta a ponta de uma rede neural para a leitura destas planilhas, em cenários com uma quantidade limitada de dados. A rede neural deverá receber a imagem de uma planilha e produzir em sua saída a sequência de lances que estão escritos na planilha. Além do reconhecimento da escrita propriamente, a rede deverá aprender a ordem correta de leitura. Por se tratar de um problema para o qual não encontramos trabalhos na literatura da área, o método utilizado consistiu na criação de um conjunto de dados e uma ampla investigação experimental utilizando uma rede neural recorrente com mecanismo de atenção. Identificamos três subtarefas subjacentes ao problema: (1) o aprendizado do modelo de linguagem, relacionado com a previsibilidade dos lances, (2) o alinhamento entre a entrada e a saída, e (3) o reconhecimento da escrita propriamente. Constatamos que essas tarefas possuem distintos graus de dificuldade e que existem alguns fatores que são críticos no aprendizado delas. Mais do que isso, constatamos também que uma combinação adequada desses fatores é fundamental para um treinamento ponta a ponta bem sucedido. Um modelo básico foi avaliado quanto ao reconhecimento dos 16 primeiros lances e alcançou acurácia de 65,78% em termos de lances corretamente reconhecidos.

Palavras-chave: Reconhecimento de texto escrito a mão, Rede neural, Rede convolucional, Rede neural recorrente, Mecanismo de atenção.

Abstract

Sergio Yuji Hayashi. **Reading Handwritten Chess Score Sheets with Attention Networks**. Thesis (Masters). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

Handwriting recognition remains an open problem, a subject of intense research in the area of machine learning. In this project we focus on a specific category of problem in this area, the automatic reading of chess score sheets. Chess score sheets contain notation of game moves handwritten by the players themselves in a format called algebraic notation. Compared to traditional natural language text, chess score sheets are fixed-format forms, their textual content is restricted to a reduced vocabulary, and writing in general is not entirely cursive. Even so, they still present a high variability of handwriting styles, making their reading a sufficiently complex problem. The objective of this work is the end-to-end training of a neural network for reading these score sheets, in scenarios with a limited amount of data. The network should receive an image of a score sheet and produce as output the sequence of moves that are written in the score sheet. Besides recognizing the handwriting, the network must be able to learn the correct reading order. As we have found no records about this problem in the literature, the adopted method consisted of creating a dataset and an extensive experimental investigation using a recurrent neural network with attention mechanism. We have identified three underlying subtasks of the problem: (1) the learning of the language model, related to the predictability of the moves, (2) the alignment between input and output, and (3) the recognition properly said. We found out that these tasks have distinct levels of difficulty and that there are critical factors for learning them. More than that, we also found out that an adequate combination of these factors is fundamental for a successful end-to-end training. A basic model was evaluated regarding the recognition of the first sixteen moves and it achieved an accuracy of 65.78% in terms of correctly recognized moves.

Keywords: Handwritten text recognition, Neural network, Convolutional neural network, Recurrent neural network, Attention mechanism.

Lista de Abreviaturas

BLSTM	Bidirectional LSTM
CER	Character error rate
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
FIDE	International Chess Federation
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HTR	Handwritten Text Recognition
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LSTM	Long Short-Term Memory
MDLSTM	Multi Dimensional LSTM
MNIST database	Modified National Institute of Standards and Technology database
NLP	Natural Language Processing
OCR	Optical Character Recognition
PGN	Portable Game Notation
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
SAN	Standard Algebraic Notation
SVM	Support Vector Machine
VGG16	Very Deep Concolutional Networks for Large-scale Image Recognition
WER	Word error rate

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Notação de xadrez

O xadrez é um jogo de tabuleiro criado há mais de 1500 anos[1] e continua sendo jogado no mundo inteiro por uma grande população de jogadores ativos. Ao redor do mundo inteiro há, por exemplo, 352.234 jogadores com ranking no FIDE[2] e 50.334.080 membros cadastrados no site `chess.com`[3], em números de dezembro de 2020.

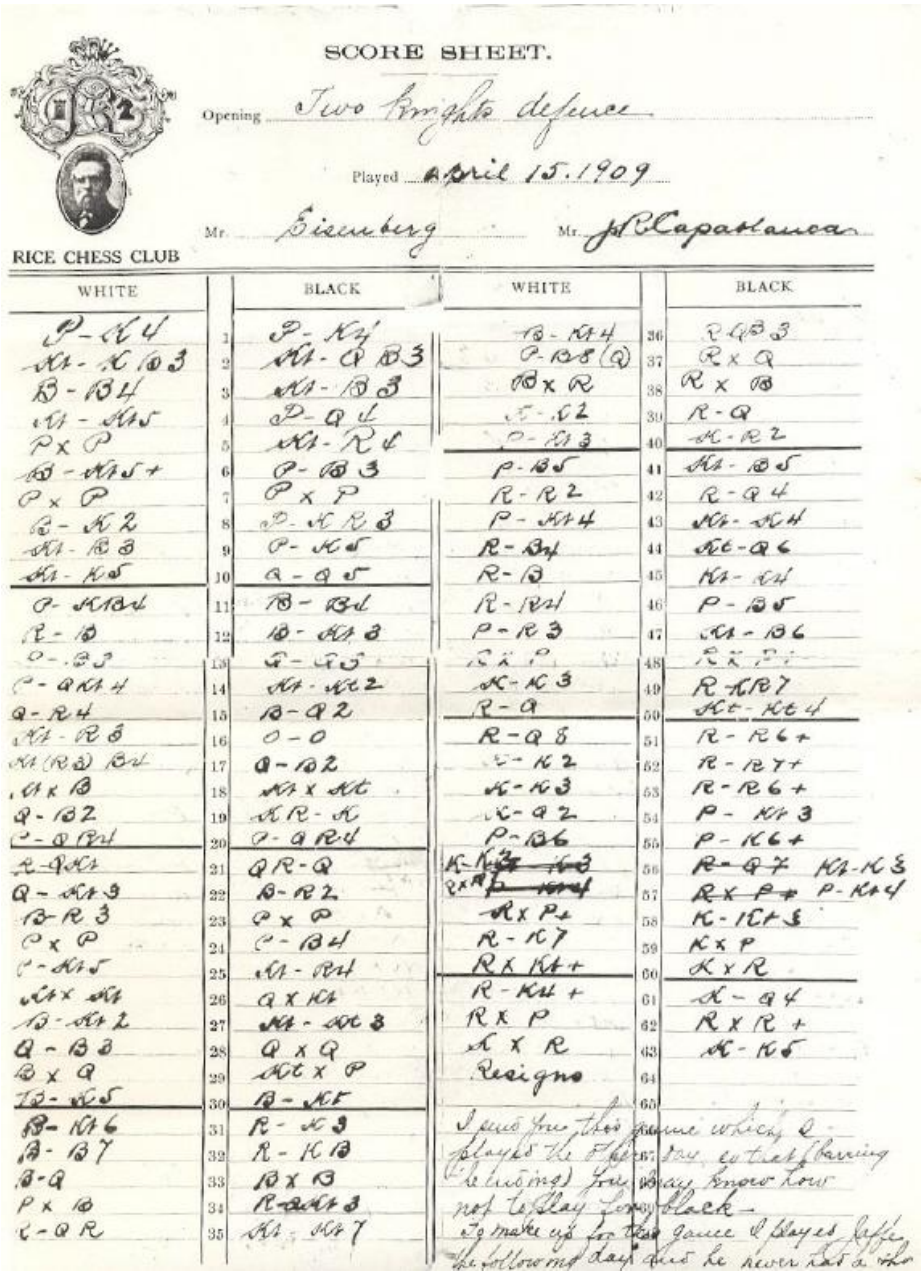
A evolução de uma partida de xadrez é comumente anotada manualmente em um formulário específico pelos próprios jogadores durante a partida. No formulário são anotados todos os lances da partida, lance a lance, de ambos os jogadores, e por ambos os jogadores. A anotação, cujo formato é regulamentado pela FIDE (*International Chess Federation* [4]), é obrigatória em partidas oficiais. O formulário com as anotações dos lances é denominado **planilha** (*score sheet* em inglês). Exemplos de planilhas são mostrados nas figuras 1.1 e 1.2.

As planilhas de xadrez podem ser usadas em uma análise pós-jogo. Por exemplo, baseado nas anotações, o professor pode simular o jogo sobre o tabuleiro, repassando cada lance do jogo junto com o aluno e fornecendo-lhe orientações. Ela também serve para a resolução de conflitos durante uma partida. Caso surja algum questionamento, o árbitro pode consultar a planilha para entender o andamento do jogo até aquele momento.

Os jogos de xadrez são em geral catalogados em banco de dados para diversas finalidades. Existem bases online disponíveis como *ChessBase*[5] e estes em geral oferecem também análises dos jogos. A análise automática de jogos pode ser realizada por ferramentas denominadas *chess engine*, como o *Stockfish*[6], que avaliam se um lance foi bom ou ruim, exibem alternativas de lances melhores, ou realizam simulações.

Para inserção no banco de dados e para a análise por software, os lances anotados na

[1] https://en.wikipedia.org/wiki/History_of_chess
[2] https://www.fide.com/news/288
[3] https://www.chess.com/members
[4] https://www.fide.com/
[5] https://en.chessbase.com/
[6] https://stockfishchess.org/

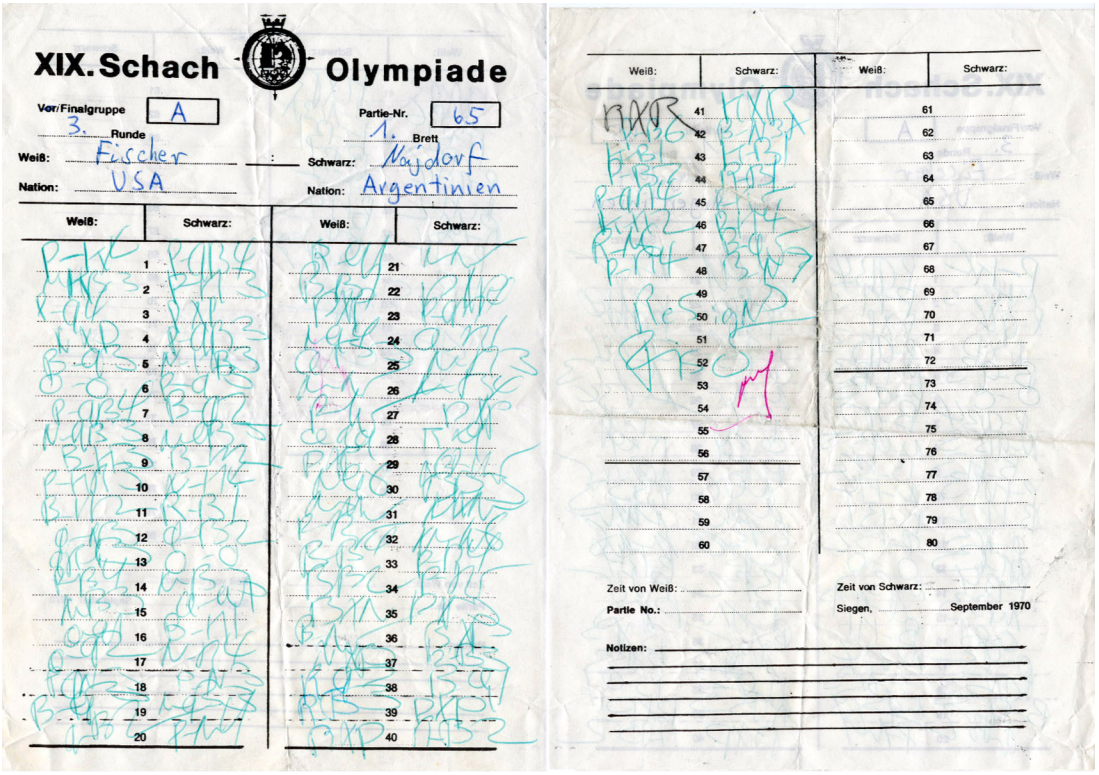


fonte: https://en.wikipedia.org/wiki/Glossary_of_chess#Score_sheet

Figura 1.1: *Planilha da partida entre Capablanca e Eisenberg em 1909.*

planilha precisam ser lidos e digitados no computador. Esse processo é realizado manualmente. Em geral, uma pessoa proficiente em xadrez lê a planilha e insere os dados no sistema, simulando os movimentos em um tabuleiro virtual exibido na tela.

A leitura da planilha nem sempre é fácil (veja o exemplo da figura 1.2). Na prática observa-se que pessoas experientes, no nível de Mestre Internacional ou Grande Mestre, conseguem ler as planilhas com muito mais facilidade do que pessoas com menos experiência, tanto em termos de interpretação correta quanto de velocidade de leitura. Portanto, apesar de a notação incluir apenas dígitos e caracteres do alfabeto convencional, a capacidade de uma pessoa entender o contexto e inferir a jogada mais provável naquele



fonte: https://en.wikipedia.org/wiki/Bobby_Fischer

Figura 1.2: *Exemplo de planilha. Partida entre Bobby Fisher e Miguel Najdorf nas olimpíadas em Siegen na Alemanha em 1970*

contexto de jogo faz a diferença na capacidade de leitura. Ou seja, a correta leitura da planilha é bastante facilitada se, além da capacidade de reconhecer letras e números, há um bom conhecimento sobre os padrões de jogos.

1.2 Reconhecimento de escrita por computador

A escrita, introduzida há mais de 2 mil anos, é um recurso amplamente utilizado para registro e transmissão de informação de diferentes tipos. Não é difícil encontrarmos vários exemplos no dia-a-dia: livros, jornais impressos, documentos diversos, entre outros.

Com o avanço dos sistemas computacionais, muitos dos textos gerados atualmente são produzidos diretamente no formato digital. O armazenamento de textos em formato digital facilita uma série de atividades relacionadas com a organização, busca e análise de informação. No entanto, ainda são produzidos muitos textos manuscritos e existem muitos manuscritos e impressos que não estão digitalizados.

Pode-se também argumentar que, devido à facilidade de utilização de computadores e de entrada de textos neles por meio do teclado ou outros dispositivos, a escrita manuscrita vem perdendo espaço. No entanto, a preferência pela escrita manual ainda prevalece em vários cenários. Muitos formulários ainda são preenchidos em papel nos dias de hoje. Os dispositivos como *tablets*, por outro lado, permitem que a escrita manual possa ser realizada diretamente no dispositivo. No entanto, nesses casos captura-se uma imagem ou

a trajetória da escrita, mas não se tem o reconhecimento do que foi escrito. Além disso, muitos desses dispositivos com telas sensíveis a toque (como os celulares), possuem uma área pequena, limitando o espaço para a escrita. Portanto a leitura automática de diferentes tipos de documentos, a partir da imagem deles, é de bastante interesse tanto na academia como na indústria.

A digitalização de documentos em geral envolve uma etapa de captura de imagem, por exemplo por meio de escâner ou câmera, seguida de uma etapa de reconhecimento do texto. Os sistemas utilizados para o reconhecimento de texto em imagens de documentos são comumente conhecidos como OCR (do inglês *Optical Character Recognition*). A capacidade dos OCRs atuais em reconhecer textos impressos é bastante boa, principalmente quando a imagem apresenta uma boa qualidade (bom contraste e caracteres aparecendo suficientemente destacados).

Porém, OCRs atuais ainda apresentam índices de reconhecimento bastante baixos para textos manuscritos. O reconhecimento de escrita manuscrita (HTR do inglês *Handwritten Text Recognition*) é muito mais desafiador do que o reconhecimento de texto impresso por conta do estilo cursivo (no qual os caracteres aparecem conectados, sem a separação clara observada em textos impressos) e também devido às variações no estilo de escrita entre indivíduos. Os exemplos das figuras 1.1 e 1.2 ilustram bem essas características desafiadoras. Na figura 1.3 mostramos o resultado gerado por dois OCRs comerciais (`Textract` e `PenToPrint`) quando aplicados sobre um recorte de uma planilha de xadrez. Pode-se observar que a acurácia é baixa. Em termos de caracteres é abaixo de 50%, e de palavras não chega a 20%, conforme pode ser visto na tabela abaixo, na qual os erros de reconhecimento estão destacados em vermelho.

Correto	e4	c5	Nc3	e6	d3	d6	g3	a6	Be3	b5	Qd7	Bb7	Bg2	Qc7	Nge2	Nf6
Textract	04	ck	Ne3	R6	d3	d6	g3	0	ge3	br	0d+	Bh?	Bg}	Qek	Aff Neh	Nf{
PenToPrint	04	05	No3	eb	d3						Qd7	367	Bg2	Q7		

Os OCRs modernos são baseados em técnicas de aprendizado de máquina. As técnicas de aprendizado de máquina são uma abordagem flexível para lidar com a variabilidade pois são capazes de, a partir de exemplos entrada-saída de um processamento desejado, gerar um modelo de predição que aproxima a relação entre a entrada e a saída. Isto é, OCRs podem ser "treinados" para reconhecer os diferentes caracteres, em diferentes fontes, estilos e tamanhos. Uma vez treinado, ele será capaz de reconhecer com boa acurácia outras instâncias de caracteres similares aos usados no treinamento. Essas técnicas baseadas em aprendizado de máquina são atualmente referenciadas como orientadas a dados, em contraste às técnicas anteriores que eram predominantemente baseadas em heurísticas ou conjunto de regras fixas. A grande vantagem das técnicas orientadas a dados é o fato de não haver necessidade de se reprogramar o sistema computacional quando os dados de entrada apresentam características distintas daquelas para as quais o sistema foi preparado; basta "retreinar" a máquina com os dados novos.

No caso de reconhecimento de manuscritos, em 1989, Yan LeCun e outros (LeCun, Boser *et al.*, 1989) desenvolveram com sucesso um modelo de redes neurais treinado via *backpropagation* para reconhecimento de dígitos manuscritos de código postal. Este feito representou um grande avanço no uso da máquina para esta categoria de problemas.

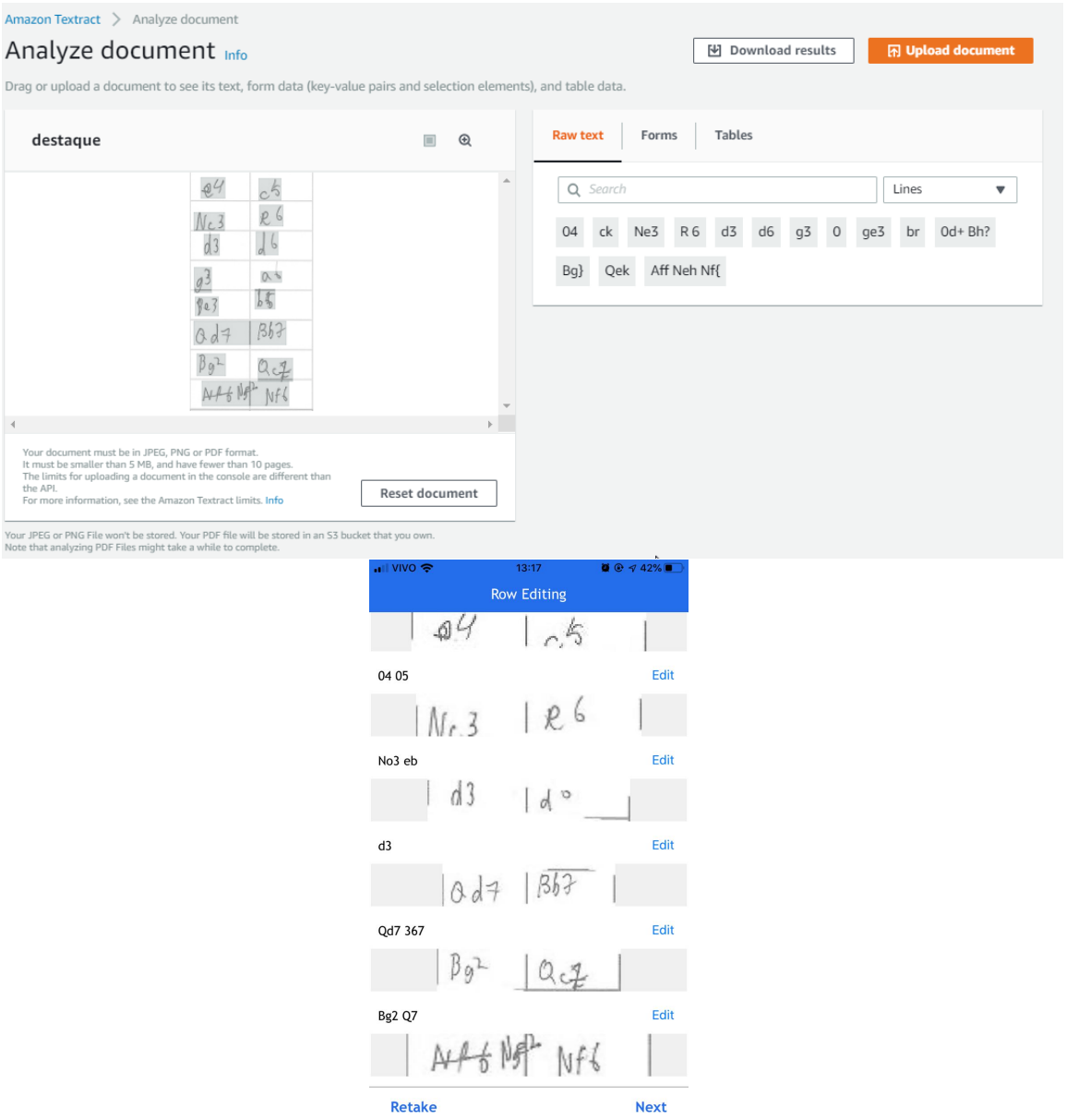


Figura 1.3: *Soluções de OCR aplicadas ao problema de leitura de formulários de xadrez. Na parte superior, a transcrição gerada pelo AWS Textract (https://aws.amazon.com/pt/textract/). Na parte inferior, a transcrição gerada pelo Pen to Print (https://play.google.com/store/apps/details?id=p2p.serendi.me.p2p). Ambos resultaram em taxa de acerto abaixo de 50%.*

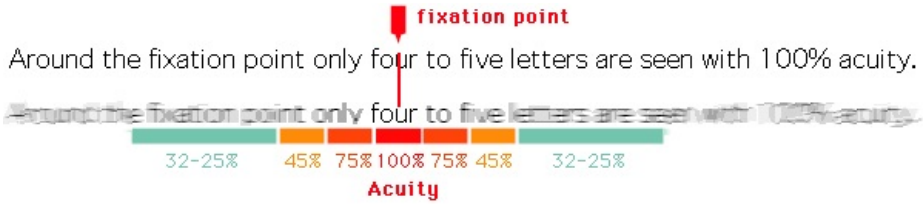
Um dos resultados da pesquisa de LeCun é a bem conhecida base de dados MNIST[7] de reconhecimento de dígitos manuscritos. Essa base acabou tornando-se um dos principais "*benchmarks*" para a avaliação de técnicas na área de aprendizado de máquina. No entanto, no contexto de HTR, as soluções desenvolvidas naquela ocasião eram restritas a problemas de reconhecimento de caracteres individuais. O reconhecimento de uma linha de texto manuscrito, na qual os caracteres não estão claramente segmentados ou apresentam sobreposição, além de aparecerem em uma diversidade de formas, ainda é um grande

[7] http://yann.lecun.com/exdb/mnist/

desafio, fazendo o problema de HTR como um todo um campo de pesquisa que tem atraído cada vez mais atenção.

Mais recentemente, avanços na área de aprendizado de máquina permitiram a expansão do modelo de LeCun de 1989 para problemas mais complexos. As técnicas atuais, denominadas *deep learning*, estão sendo aplicadas com sucesso em vários problemas que envolvem imagens. Dentre eles, problemas similares ao tratado neste trabalho, nos quais a entrada é uma imagem e a saída é um texto digital, são o *captioning* de imagens (You *et al.*, 2016) e o reconhecimento de expressões matemáticas manuscritas (J. Zhang *et al.*, 2017). Em HTR, alguns trabalhos que utilizam *deep learning* estão descritos em Bluche *et al.*, 2017; Kang, Toledo *et al.*, 2018.

No processo de leitura de texto, o foco do sistema visual humano "passeia" sobre o texto seguindo a direção de leitura do texto (figura 1.4). Pesquisas recentes em aprendizado de máquina têm introduzido técnicas que são inspiradas na capacidade do sistema visual humano em focar a atenção visual em partes específicas da cena de seu campo de visão. Esses mecanismos são chamados de "atenção" (*attention*) (Bahdanau *et al.*, 2015) e têm sido amplamente utilizados em redes neurais profundas, para uma diversidade de tarefas que envolvem, por exemplo, processamento de linguagem natural.



fonte: https://en.wikipedia.org/wiki/Reading

Figura 1.4: *Controle da atenção visual pelo ser humano durante leitura num estudo realizado por Hans Hermer Hunziker, publicado no livro "In the eye of the reader: foveal and peripheral perception - from letter recognition to the joy of reading"(em alemão) em 2006*

1.3 Objetivos

Considerando-se o cenário apresentado acima, o principal objetivo deste trabalho é o desenvolvimento de uma solução ponta a ponta (do inglês *end-to-end*) para a leitura automática de planilhas de xadrez. Por solução ponta a ponta referimo-nos a uma rede neural que seja capaz de processar diretamente uma imagem de uma planilha de xadrez, como a mostrada na figura 1.1, para gerar uma sequência na saída com a identificação dos lances anotados na planilha. O treinamento da rede deve ser baseado em exemplos consistindo de imagens de planilha e as respectivas sequências-alvo. Para fazer o mapeamento entrada-saída de forma correta, o modelo deve lidar concomitantemente com múltiplas subtarefas implícitas, sem nenhuma informação adicional. As principais subtarefas são identificar o início da sequência na imagem, determinar a direção de leitura (da esquerda para a direita, e de cima para baixo), determinar a quebra de linha, e fazer o reconhecimento da escrita propriamente.

Em uma investigação inicial, não encontramos publicações nem conjuntos de dados

relacionados a este problema. Desta forma, delineamos os seguintes objetivos específicos:

1. Criar um *dataset* para o problema. O *dataset* deve ser adequado à solução ponta a ponta, isto é, conter pares formados por uma imagem de planilha e o respectivo rótulo (sequência com a identidade dos lances escritos na planilha).
2. Estudar e aplicar, no problema abordado neste trabalho, algum modelo recente de rede neural profunda que esteja sendo utilizado em problemas similares.
3. Identificar os pontos de dificuldade de treinamento da rede. Entender os comportamentos associados e os fatores que afetam esses comportamentos, e propor estratégias para contornar ou solucionar os pontos de dificuldade.

1.4 Contribuições

Uma primeira família de trabalhos que poderiam servir como referência para o problema de leitura de planilhas de xadrez são aqueles relacionados à leitura de tabelas. No entanto, avaliamos que tratar esse problema como um problema de leitura de tabelas seria uma empreitada para além da necessária, uma vez que as planilhas de xadrez possuem uma estrutura bastante regular enquanto as tabelas podem ser muito mais complexas. Qualquer solução desenvolvida não traria benefícios para a tarefa de leitura de tabelas.

Outra família de problemas relacionados é o de reconhecimento de textos manuscritos (HTR). Esse problema, na literatura, é tratado em diferentes níveis de granularidade tais como reconhecimento de caracteres, reconhecimento de palavras, ou reconhecimento de linhas de textos. Uma grande maioria dos trabalhos em HTR se restringem ao reconhecimento de uma linha de texto; portanto, assumem uma única sequência linear na imagem. Em contraste, o problema tratado nesta dissertação apresenta o desafio de identificar o final de linha ou quebra de linha, além obviamente de fazer a leitura na ordem correta em cada linha (da esquerda para a direita). Uma terceira família de problemas que nos pareceram relacionados são aqueles que tratam problemas do tipo imagem para sequência, como por exemplo o de *image captioning* Xu *et al.*, 2015. Tanto nos problemas de HTR como nos de *image captioning*, notamos que as soluções predominantes utilizam alguma estratégia para extração de características da imagem, seguida de uma rede recorrente com mecanismo de atenção. Desta forma, decidimos explorar esse tipo de arquitetura em nosso problema. Também decidimos tratar o reconhecimento em nível de palavras (lances individuais) e não de caracteres individuais.

O desenvolvimento do trabalho foi fortemente baseado em investigações experimentais. Inicialmente foram realizados experimentos exploratórios visando-se familiarização com os dados, o ambiente de treinamento e aspectos operacionais do processo de treinamento em si. Nesses experimentos, foi inicialmente constatado que modelos baseados na arquitetura de rede considerada possuem capacidade para processar os dados, porém têm tendência a "decorar" as sequências e também a apresentar forte *overfitting*.

Após outros experimentos exploratórios, alcançamos uma configuração adequada da arquitetura de rede e demais parâmetros que possibilitaram o treinamento ponta a ponta da rede – um modelo referência. Também identificamos três subtarefas subjacentes ao

problema e indicações de fatores que influenciam e possibilitam o aprendizado de cada uma das tarefas:

1. alinhamento – diz respeito à identificação da região na imagem da planilha que corresponde ao lance sendo decodificado num certo instante;
2. reconhecimento do lance – diz respeito ao reconhecimento da escrita em si (que deve ser um dos lances no vocabulário considerado);
3. modelo de linguagem – diz respeito aos padrões de jogo e, portanto, à previsibilidade do próximo lance, que pode ajudar na predição correta do lance.

Em uma segunda etapa de investigação, realizamos experimentos controlados para estudar o efeito de diferentes fatores quanto ao aprendizado dessas subtarefas, tomando como base o modelo referência.

As principais contribuições deste trabalho podem ser sumarizadas da seguinte forma:

- A criação de um *dataset* com imagens de planilhas de xadrez, devidamente anotadas com o *ground-truth*;
- A identificação de subtarefas implícitas que devem ser aprendidas concomitantemente pelo modelo em um treinamento ponta a ponta, os fatores que influenciam o aprendizado de cada uma delas, e os requisitos para o aprendizado conjunto delas.
- Um modelo que consiste de uma rede neural composta por um módulo convolucional seguido por uma rede recorrente com uma estrutura *encoder-decoder* com camada de atenção, obtido por treinamento ponta a ponta, e que poderá ser usado como modelo de referência em trabalhos futuros.

1.5 Organização do texto

O restante deste texto está organizado da seguinte forma.

No capítulo 2 apresentamos algumas terminologias e conceitos básicos, no contexto de jogos de xadrez e também relacionados às redes neurais e, em especial, às redes convolucionais, às redes recorrentes e ao mecanismo de atenção, que são utilizadas na solução desenvolvida.

No capítulo 3 discorremos brevemente sobre trabalhos encontrados na literatura que tratam problemas nos quais a entrada são imagens e a saída são textos e revisamos alguns métodos baseados em *deep learning* utilizados na área de HTR.

Dos capítulos 4 ao 6 apresentamos o trabalho desenvolvido. Este é dividido em: métodos (capítulo 4) com a descrição dos elementos e procedimentos usados, investigação experimental (capítulo 5) com a descrição dos experimentos controlados feitos para investigar o comportamento do modelo referência com respeito a cada uma das subtarefas que compõem o aprendizado ponta a ponta, e os resultados (capítulo 6).

No capítulo 7 apresentamos as conclusões e algumas considerações sobre trabalhos futuros.

O apêndice ao final contém conteúdo relacionado ao trabalho que julgamos não essencial para a leitura do texto, mas que podem complementar a leitura daqueles que se interessarem por mais detalhes. Cada um deles está apontado ao longo do texto, nas partes pertinentes.

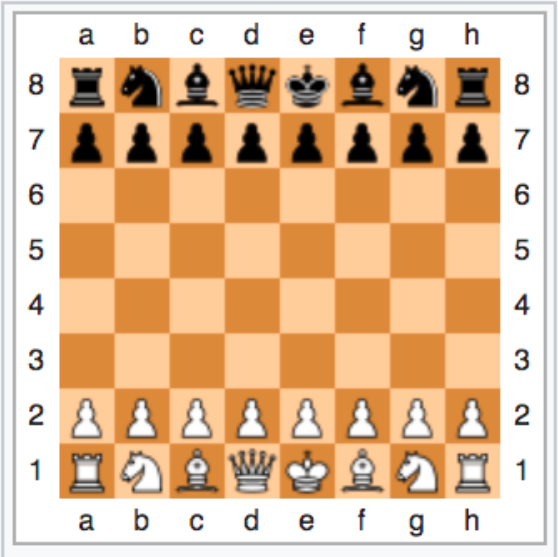
Capítulo 2

Preliminares

Neste capítulo apresentamos conceitos e terminologias relacionadas a notações de xadrez e também a redes neurais, destacando nesse caso as arquiteturas recorrentes, redes do tipo *encoder-decoder*, e mecanismo de atenção.

2.1 Padrão SAN de notação de xadrez

O tabuleiro de xadrez é composto de 8 linhas, chamadas de *rank*, e 8 colunas chamadas de *file*, conforme mostrado na figura 2.1. As linhas são designadas por números de *1* a *8* e as colunas por letras de *a* a *h*. No início do jogo, as peças brancas ocupam as linhas *1* e *2*, enquanto as pretas ocupam as linhas *7* e *8*. Cada posição do tabuleiro é designada pela junção de uma letra e um número. Por exemplo, *a3* é a posição correspondente à coluna *a*, na linha *3*.



fonte: https://en.wikipedia.org/wiki/Chess

Figura 2.1: *Configuração inicial do tabuleiro e designação das linhas e colunas.*

Em uma partida de xadrez, os lances são comumente anotados à mão pelo próprio jogador em um formulário próprio, denominado planilha (*score sheet* em inglês) como as mostradas no capítulo anterior, nas figuras 1.1 e 1.2. O padrão de notação mais utilizado atualmente é o chamado *Standard algebraic notation* (SAN) ou notação algébrica.

Dependendo da língua, a letra usada para identificação das peças pode variar. A língua oficial é a inglesa. A correspondência entre as notações em inglês e português utilizadas para a identificação de peças estão listadas abaixo:

Português		Inglês	
Nome	Letra	Nome	Letra
Rei	*R*	*King*	*K*
Dama	*D*	*Queen*	*Q*
Torre	*T*	*Rook*	*R*
Bispo	*B*	*Bishop*	*B*
Cavalo	*C*	*Knight*	*N*
Peão	(sem letra)	*pawn*	(sem letra)

A notação de um movimento de peça no padrão SAN[1] consiste da concatenação da letra que indica a peça sendo movimentada (omitida no caso de peão) e a posição para a qual a peça está sendo movimentada. Por exemplo, a notação $Nc6$ indica a movimentação de uma peça cavalo (N) para a posição $c6$ do tabuleiro. A notação consistindo somente de $c6$ indica a movimentação de um peão para a posição $c6$.

A título de curiosidade, a notação utilizada originalmente no passado tinha um formato mais descritivo, porém com o passar do tempo ela foi evoluindo até atingir a forma atual bastante concisa, conforme mostrado na figura 2.2.

```
1614:  The white king commands his owne knight into the third house before
       his owne bishop.
1750:  K. knight to His Bishop's 3d.
1837:  K.Kt. to B.third sq.
1848:  K.Kt. to B's 3rd.
1859:  K. Kt. to B. 3d.
1874:  K Kt to B3
1889:  KKt-B3
1904:  Kt-KB3
1946:  N-KB3
Modern:  Nf3
```

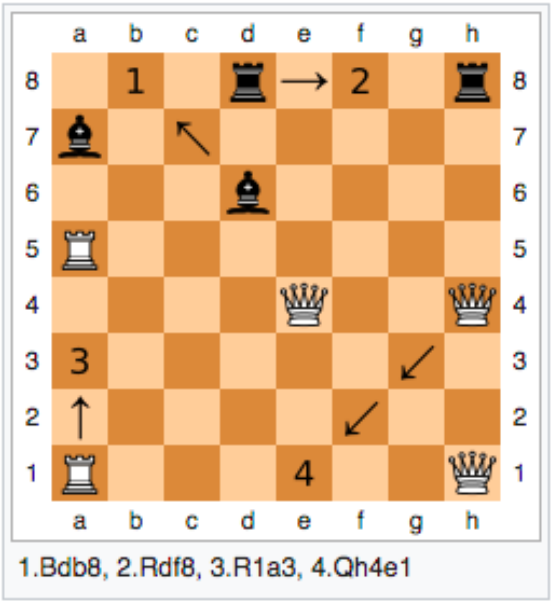
fonte: https://en.wikipedia.org/wiki/Chess_notation

Figura 2.2: *Evolução do padrão de notação de jogos de xadrez ao longo dos anos.*

Quando o movimento realizado resulta na captura de uma outra peça, coloca-se um x entre a letra que indica a peça sendo movimentada e a posição para a qual ela está sendo movimentada. Por exemplo, a notação $Nxc6$ indica que uma peça cavalo está sendo movimentada para a posição $c6$, e que tomará a peça que encontra-se naquela posição. Note que não há informação explícita sobre qual peça será tomada. Quando mais de uma

[1] https://opensource.apple.com/source/Chess/Chess-110.0.6/Documentation/PGN-Standard.txt

peça do mesmo tipo pode atacar uma dada posição, há ambiguidade na notação acima (exemplos são ilustrados na figura 2.3).



fonte: https://en.wikipedia.org/wiki/Algebraic_notation_(chess)

Figura 2.3: *Exemplos de casos de ambiguidade que requerem a notação da posição de origem. 1- Dois bispos podem se mover para a posição b8. 2- Duas torres na mesma linha podem se mover para a posição f8. 3- Duas torres na mesma coluna podem se mover para a posição a3. 4- Três damas podem se mover para a posição e1.*

Para remover ambiguidades, a notação segue as regras conforme a sequência abaixo:

1. Se apenas a identificação da coluna é suficiente, indica-se a letra da coluna após a letra da peça. Por exemplo, escreve-se $Bdb8$ no caso 1 da figura 2.3, para indicar que o bispo que está se movendo para a posição $b8$ é aquele que estava na coluna d .

2. Se as peças ocupam uma mesma coluna, porém a linha é suficiente para identificar a peça, então indica-se o número da linha após a letra da peça. Por exemplo, no caso 3 da figura 2.3 a notação $R1a3$ indica que a torre (R) que está se movendo para a posição $a3$ é aquela que se encontrava na linha 1.

3. Se somente a identificação da coluna ou somente da linha não é suficiente, então indica-se a posição atual da peça (coluna e linha) após a letra que identifica a peça. Por exemplo, no caso 4 da figura 2.3, a notação $Qh4e1$ indica que é a dama da posição $h4$ que está sendo movimentada para a posição $e1$. Caso haja tomada de peça, a anotação incluiria um x, isto é, $Qh4xe1$. Este tipo de ambiguidade raramente acontece. Apesar do peão ser a única com mais de 2 peças no início de jogo, no máximo 2 peões conseguem atacar uma mesma posição. Assim, a situação só pode acontecer com outras peças, desde que tenha havido uma promoção (lance em que um peão alcança a última fila e pode ser trocado por uma outra peça, como por exemplo a dama). Nestes casos, pode acontecer de 3 peças do mesmo tipo ficarem

posicionadas com possibilidade de atacar uma mesma posição, como no caso 4 da figura 2.3 no qual 3 damas podem atacar a posição $e4$.

Além disso, alguns tipos de lances específicos tem notações especiais:

- *Promoção* – a letra indicando a peça para a qual foi promovida é informada após a posição, que pode ser precedida ou não do sinal de =. Por exemplo, $e8 = Q$ or $e8Q$.
- *Roque* (lance em que se troca a posição do rei ou dama com a torre) – é indicado como $O - O$ (lado do rei) e $O - O - O$ (lado da dama).
- *Xeque* (lance em que é feito a ameaça direta ao rei, ou seja, pode ser feito o xeque mate no lance seguinte caso o adversário não faça a defesa) – pode ser indicado com o sinal + após a jogada.
- *Xeque duplo* (além do xeque, há um ataque ao mesmo tempo a uma peça de maior valor como, por exemplo, a dama) – indica-se com sinal ++ após a jogada.
- *Xeque mate* (lance após o qual o rei não tem como escapar, portanto indica também o final de jogo) – indica-se com o sinal # após o lance.

Ao final do jogo, 1 – 0 indica vitória das brancas, 0 – 1 das pretas, e 1/2 – 1/2 (ou 0.5 – 0.5) indica empate.

Em 1993 foi criada uma padronização para a representação da notação do xadrez em formato digital chamado de *Portable Game Notation* (PGN[2]). Este é um formato texto, no qual um movimento de uma peça é descrito usando-se a notação algébrica para facilitar e padronizar o processamento pela máquina, podendo conter informações adicionais relativas ao jogo. O PGN de *import* é um formato propositalmente mais relaxado e tem por objetivo facilitar a entrada de dados por um ser humano. Por outro lado, o PGN de *export* é gerado pelo computador, com convenções mais rígidas.

Um arquivo PGN é constituído por um conjunto de *tags* e seus respectivos valores, seguido pelo texto que descreve a sequência de lances em formato SAN. O valor de cada *tag* é especificado em formato texto e entre aspas duplas. São 7 *tags* obrigatórias indicando o evento, a localização, a data, o número da partida no evento, o nome dos jogadores e o resultado final. Existem também *tags* opcionais. Um exemplo de PGN de *export*, de uma partida entre Fischer e Spassky ocorrida em 1992 e com uma sequência de 85 lances, é mostrado na figura 2.4.

[2] https://en.wikipedia.org/wiki/Portable_Game_Notation

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 {This opening is called the Ruy Lopez.}
4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5
Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6
23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5
hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6
Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

fonte: https://en.wikipedia.org/wiki/Portable_Game_Notation

Figura 2.4: *Exemplo de um arquivo no formato PGN de* export.

2.2 Redes Neurais

Os algoritmos de *Machine Learning* que mais têm sido utilizados atualmente em problemas que envolvem processamento de imagens ou de linguagem natural são as redes neurais.

Redes Neurais são modelos computacionais (estrutura de dados mais algoritmo) inspirados no cérebro humano. Elas contém um conjunto de nós individuais, chamados de neurônios (ou *nós* ou *unidades*), conectados uns aos outros, e são tais que os sinais de entrada são propagados entre os nós ao longo da rede inteira de forma paralela para gerar uma saída representando o resultado de um cálculo, comumente chamado de predição.

Cada nó, no seu lado de entrada, é conectado a vários outros nós a partir dos quais recebe diversos estímulos (análogo aos dendritos). Do seu lado de saída, cada nó gera um único estímulo (análogo ao axônio), resultante do processamento realizado no nó a partir dos dados de entrada. Essa saída é propagada para os demais nós subsequentes. Desta forma, esses neurônios computacionais se assemelham estruturalmente a neurônios do cérebro.

Os parâmetros que influenciam no processamento do sinal em cada nó e na propagação do sinal são ajustados por técnicas de *machine learning*.

2.2.1 Neurônio artificial

O primeiro modelo computacional de um neurônio artificial foi proposto por Warren MuCulloch e Walter Pitts em 1943 (Mcculloch e Pitts, 1943), como parte de um estudo para entendimento do funcionamento do cérebro na tentativa de produzir padrões complexos usando nós (ou neurônios artificiais) básicos conectados entre si. Neste modelo, chamado de *Binary Threshold Neuron*, foi introduzindo o conceito de *threshold*. Cada

neurônio artificial calcula a soma dos sinais de entrada ponderado pelo peso da conexão de entrada e propaga o sinal para a saída, com valor 1 se a soma é maior do que o *threshold* ou 0 caso contrário. O valor de saída era interpretado como sendo o valor $True$ de uma determinada proposição lógica.

O cérebro humano possui em torno de 86 bilhões de neurônios[3], que podem cada um se conectar a até 10.000 outros neurônios, e chegam a ter até 100 trilhões de conexões de sinapse trafegando sinais, que segundo estudos é equivalente a um computador com capacidade de processamento de 1 trilhão de bits por segundo. Os neurônios são células que diferente de outros órgãos do corpo são concebidos para que as mesmas células durem a vida inteira.

Por outro lado, a maior rede neural publicada até hoje tem em torno de 160 bilhões[4] de parâmetros (equivalente a sinapse), o que representa 0.16% do número de sinapses do cérebro. Portanto, podemos dizer que uma rede neural ainda é muito menor se comparada com o cérebro humano.

2.2.2 *Perceptron*

Em 1958, Rosemblatt publicou o *Perceptron* (Rosenblatt, 1958), o primeiro algoritmo de aprendizagem de máquina para redes neurais. O aprendizado consistia em ajustar os pesos de cada conexão de entrada usando exemplos (pares entrada-saída). O algoritmo era restrito à rede neural de uma camada constituída de nós de entrada e um nó de decisão na saída, com *threshold* binário, conforme mostrado na figura 2.5. Como pode ser visto, no exemplo a camada de entrada consiste de três nós (x_1, x_2, x_3) e um neurônio artificial que realiza a combinação linear (isto é, soma das entradas ponderada pelos respectivos pesos indicados por w_1, w_2, w_3 na figura) e aplica a decisão definida pela função de ativação. No *Perceptron* somente a última camada era passível de treinamento. Caso existissem, as demais camadas correspondiam à extração de *features*, e seus pesos eram previamente definidos e fixos na rede.

O *Percetron* foi um grande sucesso por um certo tempo, mas estudos publicados por Minsky e Papert, 1969 demonstraram que ele não era capaz de resolver problemas não lineares, evidenciando a necessidade de incorporar mais camadas intermediárias e não linearidade no modelo. No entanto, o Perceptron, enquanto algoritmo de treinamento, não era útil para treinar redes com mais de uma camada.

2.2.3 *Backpropagation*

A partir dos anos 80 foram sendo desenvolvidos algoritmos de aprendizado baseado no procedimento de *backpropagation* (David E. Rumelhart *et al.*, 1986; David E Rumelhart *et al.*, 1988), como uma forma eficiente de aprendizado em redes neurais de mais de uma camada, com é mostrada na figura 2.6. A figura mostra uma rede com uma camada de entrada com n nós, duas camadas ocultas, e uma camada de saída com um único nó.

O treinamento de uma rede neural consiste na atualização dos pesos associados às

[3] https://human-memory.net/brain-neurons-synapses/

[4] https://spectrum.ieee.org/tech-talk/computing/software/biggest-neural-network-ever-pushes-ai-deep-learning

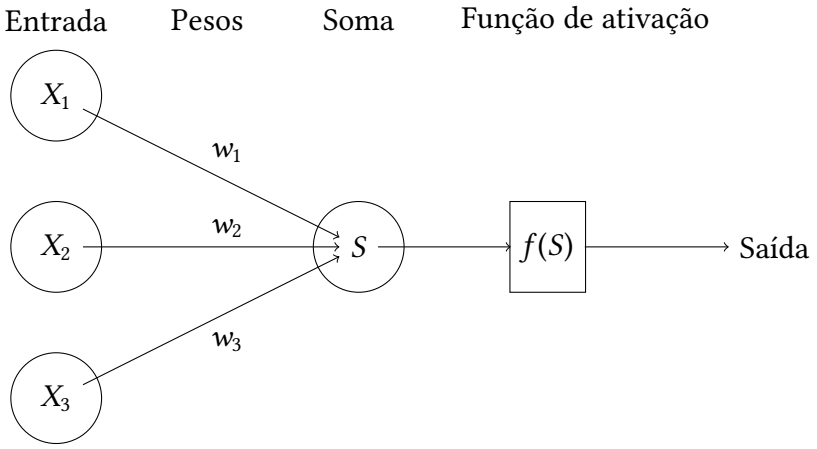


Figura 2.5: *Arquitetura de uma rede que pode ser treinada via algoritmo* perceptron *criada por Rosenblatt em 1958. Compõe-se de uma camada de entrada e um único nó de saída com* threshold *binário. Os pesos das conexões de entrada para o nó de saída são ajustados automaticamente pelo algoritmo.*

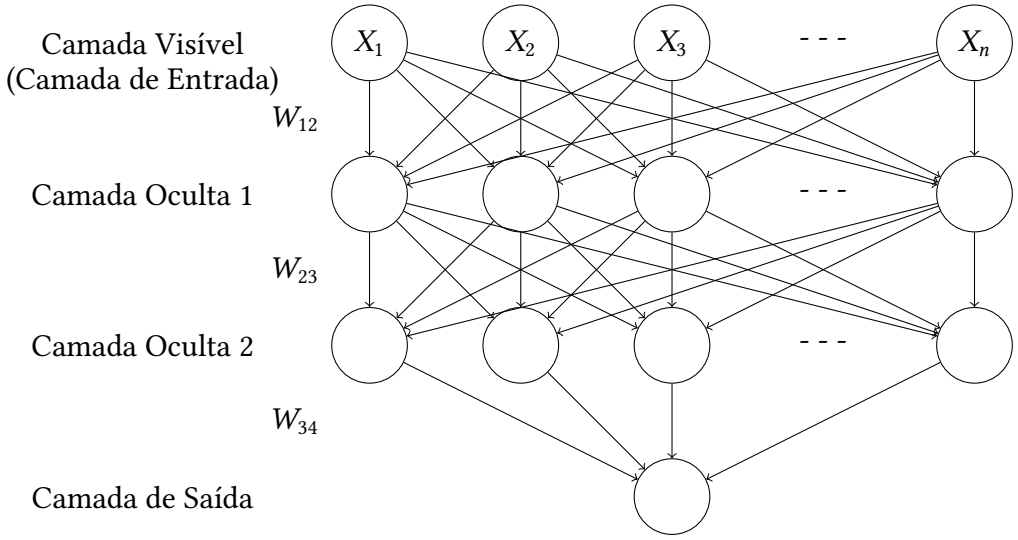


Figura 2.6: *Uma rede neural com 4 camadas. A primeira é a camada dos dados de entrada. As 2 camadas intermediárias são chamadas de camadas ocultas. A última é a camada de saída. Redes assim podem ser treinadas via* backpropagation.

conexões da rede de forma a minimizar o valor de uma função custo, por exemplo o quadrado da diferença entre o valor esperado e o valor predito pela rede. Essa minimização pode ser calculada iterativamente alterando-se os pesos no sentido inverso ao do vetor gradiente da função custo calculada no ponto correspondente ao peso atual. O algoritmo *Backpropagation* é um procedimento eficiente para realizar essa atualização de peso. Uma vez que a função custo é uma composição de funções menores, os componentes (derivadas parciais) do vetor gradiente podem ser calculados de forma recursiva a partir da última camada até a primeira camada. O procedimento tem a mesma complexidade do cálculo da predição em si, portanto muito eficiente, e pode ser realizado de forma paralela ao longo da rede.

No entanto, o procedimento por si só não foi suficiente para o treinamento de redes muito grandes e, por muito tempo, redes neurais com múltiplas camadas eram vistos como modelos difíceis de serem treinados e altamente propensos a *overfitting* (isto é, com boa capacidade de mapear as entradas usadas no treinamento às respectivas saídas, mas com desempenho inferior para realizar predições corretas em entradas novas não usadas no treinamento). Nesse período surgiram outras técnicas como SVM (*Support Vector Machine*) e *Random Forests* que passaram a ser preferidas.

2.3 Redes Neurais Convolucionais

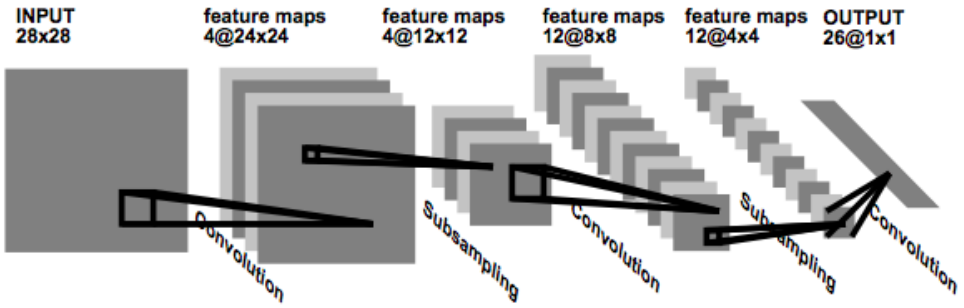
As pesquisas continuaram e ao longo das duas décadas seguintes foram desenvolvidas técnicas que finalmente possibilitaram o treinamento eficiente de redes com múltiplas camadas, que passaram a ser chamadas de *Deep Learning* (Hinton *et al.*, 2006). Observou-se que através do treinamento camada a camada pelo método não supervisionado tinha-se pesos iniciais melhores, que faziam os pesos da rede convergirem mais rapidamente quando a rede inteira era treinada pelo método supervisionado usando o algoritmo *backpropagation*. Também contribuiu para o sucesso o uso de GPU's que se mostraram 70 vezes mais rápidos para este tipo de processamento.

A disponibilidade de mais recursos computacionais e *datasets* maiores, nesta era de *big data* e *cloud computing*, e o desenvolvimento de diversas técnicas e novas arquiteturas de redes neurais possibilitaram a resolução de diversos problemas considerados complexos. Atualmente as técnicas de *deep learning*, que consistem basicamente de técnicas que utilizam-se de redes neurais profundas, constituem o estado-da-arte em áreas como Processamento de Linguagem Natural e Visão Computacional (A. Zhang *et al.*, 2020).

Dentre as redes que se destacaram está a *Convolutional Neural Network* (CNN), ou Rede neural convolucional, comumente utilizada em problemas de classificação de imagens. CNN foi historicamente inspirado na estrutura do córtex visual, parte do cérebro onde são processadas as imagens captadas pelos olhos, baseado nos trabalhos de Hubel e Wiesel, 1959 e Fukushima, 1980. Uma característica importante de uma CNN é que ela é baseada em operações matemáticas de convolução sobre imagens. Uma operação de convolução é aplicada localmente, e sobre cada ponto da imagem de entrada. Os pesos nas camadas convolucionais podem ser vistos como o *kernel* da operação matemática de convolução, e portanto como extratores de características das imagens. Esse tipo de rede é muito mais eficiente e facilmente treinada do que uma rede neural convencional totalmente conectada (*fully connected*) pois os pesos são compartilhados (i.e., o mesmo *kernel* é aplicado sobre todos os pontos), resultando portanto em um número bem mais reduzido de parâmetros a serem aprendidos.

Em 1989, Yan Lecun e outros (LeCun, Boser *et al.*, 1989) aplicaram com sucesso uma rede convolucional *feed forward*, que chamaram de *LeNet*, treinado via *backpropagation*, para o reconhecimento de imagens de dígitos manuscritos de código postal em um projeto com a empresa de correio americano. Trata-se de um sistema completo ponta a ponta cujos dados de entrada são as imagens (o conjunto de *pixels*) de códigos postais, sem segmentação prévia, e a saída são os códigos postais. Como pode ser visto na figura 2.7, a rede alterna camadas de convolução e de amostragem (*subsampling*), sendo esta segunda

para redução do tamanho dos mapas gerados após uma camada de convolução. A parte final da rede consiste de camadas totalmente conectadas.



fonte: LeCun e Bengio, 1998

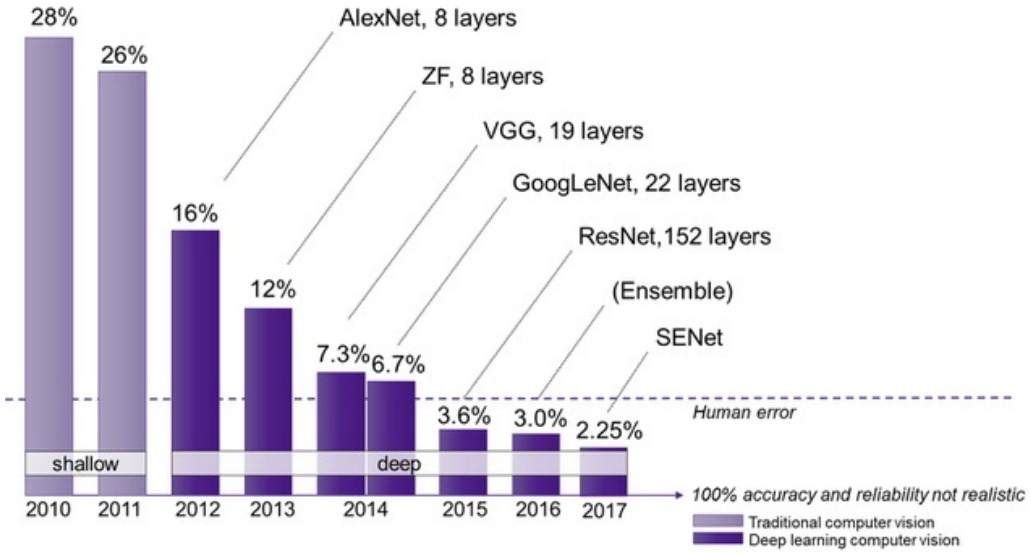
Figura 2.7: *Arquitetura do* LeNet*, de 1998. Uma das primeiras redes colocadas em produção, usando redes neurais convolucionais e treinado via* backpropagation*, para reconhecimento de código postal manuscrito.*

O treinamento de uma rede convolucional também pode ser feito pelo algoritmo de *backpropagation*. Aplica-se a mesma ideia de propagação do gradiente calculado no sentido inverso à propagação de sinal feita na predição, mas com a diferença de que os pesos são compartilhados ao longo dos pontos do mapa de entrada. Para isso, calcula-se o gradiente pela derivada parcial em cada um dos pontos de forma análoga à uma rede totalmente conectada; porém, nas camadas convolucionais o gradiente a ser utilizado para ajuste do parâmetro é a soma dos gradientes sobre toda a extensão da imagem de entrada.

Em 2012 os vencedores da competição ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) para classificação de imagens de uma base de imagens conhecida por *ImageNet* utilizaram uma rede convolucional (Krizhevsky *et al.*, 2012). O desempenho da rede proposta melhorou o resultado dos anos anteriores, que usavam abordagens baseadas em SVM, em mais de 10 pontos percentuais. As abordagens anteriores consistiam de *pipelines* heurísticos de algoritmos de visão computacional. Por outro lado, a rede convolucional vencedora foi treinada de forma ponta a ponta, sem etapas explícitas de extração de *features*. A partir de então, redes deste tipo passaram a predominar na solução de problemas tais como classificação de imagens, reconhecimento de objetos em imagens e segmentação semântica de imagens.

Redes neurais profundas cada vez maiores foram sendo propostos, reduzindo-se o índice de erro na classificação do *ImageNet* para valores na casa dos 2%, equiparáveis aos obtidos pelos seres humanos, conforme mostrado na figura 2.8. As camadas de amostragem são atualmente chamadas de camadas de *pooling*. Novas funções de ativação foram propostas, sendo uma das mais usadas a ReLU (*Rectified Linear Unit*, que é igual a 0 para valores negativos e é a identidade para os não-negativos).

O sucesso do uso das redes neurais nesta competição representou o marco para difusão do termo *Deep Learning* e adoção de *Deep Neural Networks* e desencadeou um grande boom na tecnologia de *Machine Learning* em diversas áreas. Na área de visão computacional as redes convolucionais passaram a fazer parte de soluções como a parte responsável pela extração de características.



fonte: https://semiengineering.com/new-vision-technologies-for-real-world-applications/

Figura 2.8: *Resultado da competição* ILSVRC - Imagenet Large Scale Visual Recognition Challenge *(http:// image-net.org/). Em 2012, AlexNet, baseada numa arquitetura de redes neurais convolucionais baixou em 10 pontos percentuais o melhor resultado da competição em edições anteriores.*

2.4 Redes Recorrentes

Algumas categorias de problemas como tradução ou predição do próximo elemento de uma sequência temporal requerem que sejam considerados os resultados das predições até aquele momento. Para essa categoria de problemas foi desenvolvida a rede neural recorrente, chamada em inglês de *Recurrent Neural Network* (RNN).

Uma RNN pode ser modelada como uma rede *feed forward* de múltiplas camadas. Um exemplo é mostrado na figura 2.9. O eixo vertical representa as camadas usuais da rede: uma entrada x_t passa por três camadas h^1, h^2, h^3, e é produzida uma saída y_t. No eixo horizontal está representada a dimensão temporal sendo x_{t-1}, x_t e x_{t+1} três elementos consecutivos de uma sequência de entrada. Na RNN, a rede que processa um elemento é replicado para os demais elementos da sequência e os pesos de um certo nó são compartilhados com suas réplicas ao longo da dimensão temporal (este mecanismo muitas vezes é chamado de "desenrolar" a rede). Desta forma, um nó recebe não apenas as informações provenientes dos nós na camada anterior (conexões verticais na figura), mas também do estado do nó no instante anterior (conexões horizontais na figura).

O treinamento de uma RNN pode ser feito também por *backpropagation*. Similarmente à rede convolucional, o gradiente resultante é a soma dos gradientes dos nós que compartilham pesos (por exemplo, h^3_{t-1}, h^3_t, h^3_{t+1} terão o mesmo gradiente).

Um dos problemas relacionados ao treinamento de RNNs é o chamado *vanishing gradient*. Para sequências muito longas, o gradiente aproxima-se rapidamente de zero e tende a "desaparecer" à medida que se avançam os cálculos para camadas anteriores no processo de *backpropagation*.

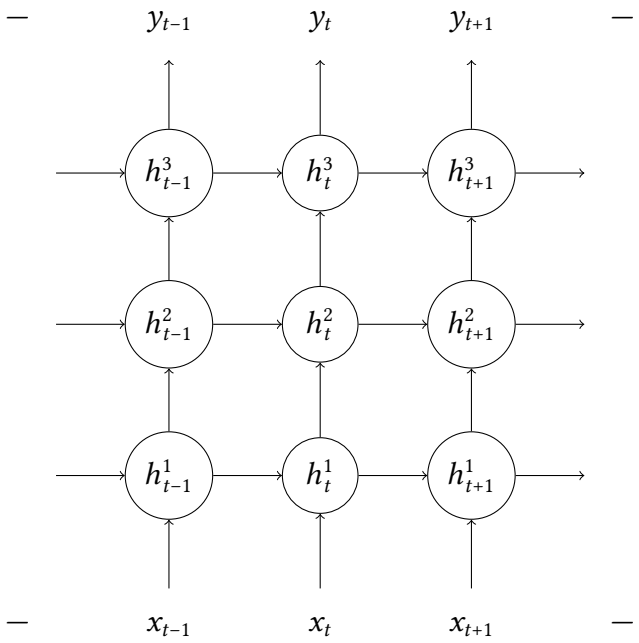
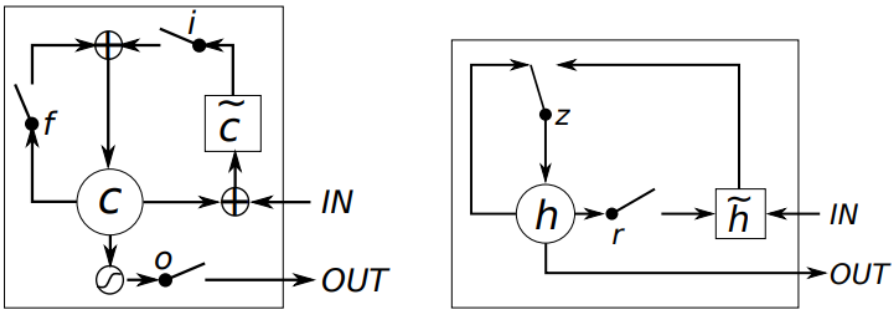


Figura 2.9: *Exemplo de uma rede recorrente. O índice t representa a dimensão do tempo. Em cada instante da dimensão do tempo é consumida uma entrada e gerada uma saída, ao mesmo tempo em que a representação interna é também por sua vez armazenada e propagada para o instante seguinte.*

Para atenuar o problema do *vanishing gradient* foram criados nós estruturalmente modificados para guardar informações a longo prazo, controlados por mecanismo conhecidos por *gates*. Os tipos de nós mais conhecidos são o LSTM e o GRU (figura 2.10).



fonte: Chung *et al.*, 2014

Figura 2.10: *Ilustração do LSTM (esquerda) e GRU (direita). No LSTM (à esquerda),* `i`*,* `f` *e* `o` *são respectivamente o* input*,* forget *e* output gates*;* `c` *e* $\tilde{c}$ *denotam o estado (memória) e o estado atualizado temporário. No GRU (à direita),* `r` *e* `z` *são respectivamente o* reset *e* update gates*;* `h` *e* $\tilde{h}$ *são o estado e o estado atualizado temporário.*

LSTM: *Long short-term memory*, proposto inicialmente em Hochreiter e Schmidhuber, 1997, é ilustrado na figura 2.10 (esquerda). Esta ilustração segue a implementação usada em Graves, 2013. Os *gates* são funções parametrizadas por pesos aprendidos em conjunto no modelo, e controlam o quanto de informação é retida e propagada. Na figura, *i*

é o *input gate*, f é o *forget gate*, o é o *output gate*. A entrada *IN* é combinada com o estado (memória) para gerar o estado atualizado temporário $\tilde{C}$. O estado é atualizado incorporando parte de $\tilde{C}$ por meio do *input gate* e "esquecendo" parte de C por meio da atuação do *forget gate*. Matematicamente, o funcionamento do nó LSTM pode ser expresso pelas seguintes equações, nas quais j indica o índice de um nó LSTM, e t o instante de tempo.

$$h_t^j = o_t^j \tanh\left(c_j^t\right) \tag{2.1}$$
$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j \tag{2.2}$$
$$\tilde{c}_t^j = \tanh\left(W_c x_t + U_c h_{t-1}\right)^j \tag{2.3}$$
$$o_t^j = \sigma\left(W_o x_t + U_o h_{t-1} + V_o c_t\right)^j \tag{2.4}$$
$$f_t^j = \sigma\left(W_f x_t + U_f h_{t-1} + V_f c_{t-1}\right)^j \tag{2.5}$$
$$i_t^j = \sigma\left(W_i x_t + U_i h_{t-1} + V_i c_{t-1}\right)^j \tag{2.6}$$

fonte: Chung *et al.*, 2014

Nessas equações,

- h_t^j é a saída ou ativação do nó recorrente no instante de tempo t (indicada como *OUT* na figura); ela é uma fração, indicada pelo *output gate* o_t^j, de um valor calculado em função do estado (ou memória) c_t^j;
- O estado c_t^j é atualizado "esquecendo-se" parcialmente o estado anterior c_{t-1}^j e adicionando-se parte do novo estado temporário $\tilde{c}_t^j$; o *forget gate* f_t^j e o *input gate* i_t^j determinam o quanto do estado anterior é "esquecido" e o quanto de informação nova é incorporada.
- Os *gates* o_t^j, f_t^j e i_t^j são expressos como uma combinação linear da entrada atual x_t, da saída anterior h_{t-1} e do estado anterior c_{t-1}; os parâmetros W, U e V são aprendidos durante o treinamento; σ é a função logística .

GRU: *Gated recurrent unit*, proposto inicialmente em Cho, Van Merriënboer *et al.*, 2014, é ilustrado na figura 2.10 (direita). Analogamente ao LSTM, ele controla o fluxo por *gates*, mas possui um mecanismo mais simplificado para gerenciar o estado. O seu funcionamento pode ser expresso pelas seguintes equações:

$$h_t^j = \left(1 - z_t^j\right) h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{2.7}$$
$$\tilde{h}_t^j = \tanh\left(W x_t + U\left(r_t \odot h_{t-1}\right)\right)^j \tag{2.8}$$
$$z_t^j = \sigma\left(W_z x_t + U_z h_{t-1}\right)^j \tag{2.9}$$
$$r_t^j = \sigma\left(W_r x_t + U_r h_{t-1}\right)^j \tag{2.10}$$

fonte: Chung *et al.*, 2014

Nessas equações,

- A saída do nó é o próprio estado, denotado por h_t^j. Diferente do LSTM, o GRU expõe o próprio estado interno. Essa ativação é definida por uma combinação linear entre o estado anterior h_{t-1} e o estado atual temporário $\tilde{h}_t$, parametrizados pelo *update gate* z_t^j, que regula quanto do estado anterior é retido e quanto do estado atual temporário é incorporado;
- O estado atual temporário $\tilde{h}_t$ é calculado combinando-se a entrada atual x_t e parte do estado anterior; o *reset gate* r_t^j determina o quanto do estado anterior é considerado. Em particular quando este valor é 0, a rede comporta-se como se estivesse fazendo *reset* (descartando toda a memória). O símbolo $\odot$ é a multiplicação de matrizes elemento a elemento, também conhecido por produto de *Hadamard*;
- Os gates z_t^j e r_t^j são expressos como uma combinação linear da entrada atual x_t e do estado anterior h_{t-1}. Os parâmetros W e U são aprendidos durante o treinamento.

As principais diferenças entre o LSTM e o GRU estão no número de *gates* (o GRU possui um *gate* a menos) e o fato do GRU expor diretamente o estado interno em sua saída. Isso faz com que o GRU seja mais simples.

Segundo Chung *et al.*, 2014, os modelos apresentam comportamentos diferentes e o desempenho depende muito do problema e do *dataset* em questão.

GRU/LSTM Bidirecional: Tanto para GRUs quanto para LSTMs, as unidades (nós) podem ser organizadas na dimensão temporal de tal forma que dependam de estados anteriores considerando-se a sequência em sentidos opostos. Isto é, cada nó enxergaria o estado anterior de quando a sequência é percorrida no sentido normal e também o estado anterior de quando a mesma sequência é percorrida de trás para a frente. Esta modificação é conhecida por GRU Bidirecional ou LSTM Bidirecional (BLSTM). Esse tipo de camada tem a vantagem de codificar dependências em percorrimentos da sequência em ambos os sentidos.

2.5 *Encoder-Decoder*

No contexto de redes recorrentes, *encoder-decoder* (Sutskever *et al.*, 2014; Cho, Courville *et al.*, 2015) é uma arquitetura formada por dois módulos de redes recorrentes, projetada para tratar o problema de alinhamento em casos nos quais o tamanho da entrada e da saída são diferentes. Um esquema é ilustrado na figura 2.11. A RNN que compõe a parte de *encoding* realiza uma codificação da sequência de entrada inteira (indicada como *feature vector* na figura) e em seguida a RNN que compõe a parte de *decoding* realiza a decodificação para a sequência de saída.

Uma das dificuldades enfrentadas por este modelo é o tratamento de sequências muito longas uma vez que toda a entrada é codificada numa representação interna antes de se iniciar a geração da saída. Isso faz com que dependências em sequências muito longas se tornem difíceis de serem descobertas, além de aumentar a suscetibilidade ao problema do *vanishing gradient*.

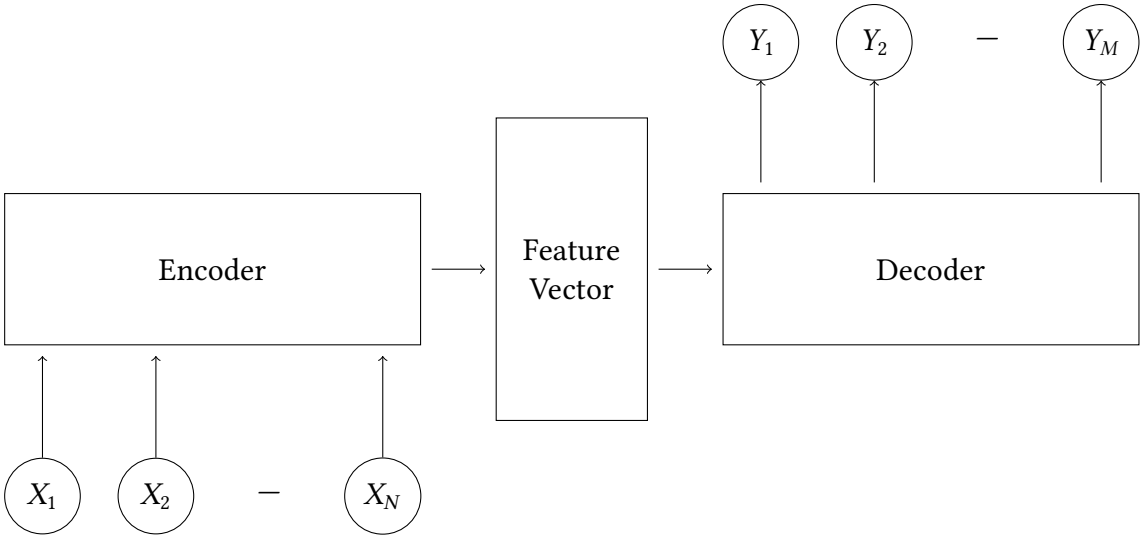


Figura 2.11: *Estrutura geral de uma rede recorrente do tipo* encoder-decoder*; permitem trabalhar problemas com sequencias de entrada e saída de tamanho distintos.*

2.6 Mecanismo de atenção

As redes recorrentes possibilitam que dependências ao longo de uma sequência ou entre pontos diferentes da sequência possam ser devidamente codificadas e consideradas na inferência da rede. No entanto, como tudo deve ser "carregado" na rede ao longo da sequência, disponível como o estado final do *encoder* ao final do processamento da sequência de entrada, isso torna inviável o estabelecimento de correlações muito complexas ou entre pontos muito distantes. Uma possibilidade para contornar essa questão seria a utilização, no *decoder*, de uma rede totalmente conectada a cada um dos nós de saída do *encoder*. Porém isso aumentaria bastante o custo computacional, principalmente se a sequência de entrada for longa. O mecanismo de atenção (Bahdanau *et al.*, 2015) provê uma forma mais performática de extrair estas correlações.

2.6.1 Mecanismo de atenção no modelo *encoder-decoder*

O *encoder*, que é uma rede recorrente, processa cada elemento da sequência de entrada, e nesse processamento, além de propagar o estado interno (memória) de uma posição para outra, também pode gerar uma saída para cada elemento. Assim, além do estado final (o estado ao final da sequência), no *encoder* podemos considerar também uma sequência de saída com o mesmo tamanho de sua sequência de entrada.

No *decoder*, também uma rede recorrente, tipicamente o estado final do *encoder* e mais um elemento indicando "vazio" ou "início" são usados para alimentar a sua primeira posição. As demais posições são alimentadas pela saída e estado da posição anterior.

No caso do *encoder-decoder* com mecanismo de atenção, o *decoder* recebe adicionalmente um vetor de contexto (*context vector*) que é a informação calculada pela camada de atenção a partir da sequência de saída do *encoder*. Na figura 2.12 mostramos um exemplo de uma rede *RNN encoder-decoder* com mecanismo de atenção.

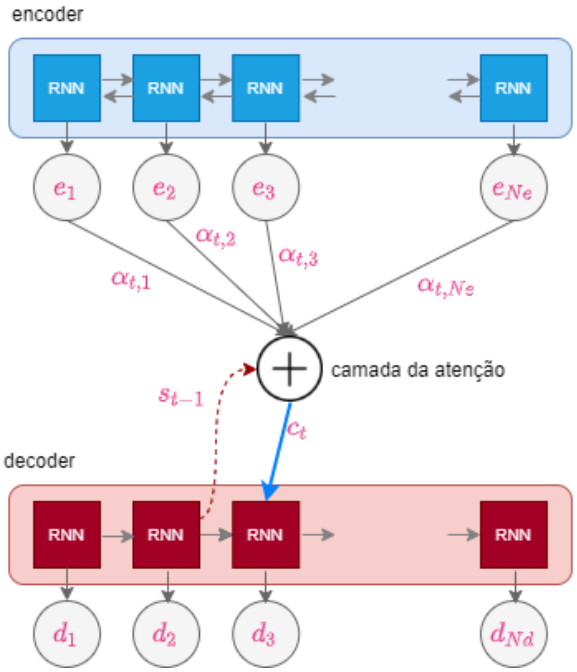


Figura 2.12: *Exemplo de uma rede do tipo* encoder-decoder *com mecanismo de atenção. O* encoder *é formado por uma camada recorrente bidirecional, e o* decoder *também uma camada recorrente. No processamento do* decoder*, para cada posição "t" do* decoder*, a sequência inteira de saída do* encoder $e_1, e_1, \ldots e_{Ne}$ *é referenciada através de uma camada de atenção, na qual o peso para cada posição é representado por "α" e o* context vector*, saída da camada de atenção, representada por "c_t".*

Seja t uma posição qualquer no *decoder*. A ideia do vetor de contexto (c_t) é capturar em quais posições no *encoder* encontram-se as informações relevantes para que seja gerada uma saída correta na posição t do *decoder*.

O vetor de contexto c_t é calculado em cada posição do *decoder* e trata-se da seguinte combinação convexa[5]:

$$c_t = \sum_{i=1}^{N} \alpha_{t,i} e_i \tag{2.11}$$

na qual $e_1, e_2, \ldots . e_N$ denotam os elementos (vetores) na sequência de saída do *encoder* e os pesos $\alpha_{t,i}$ (escalares) são chamados de alinhamento entre as posições t do *decoder* e i do *encoder*. Quanto mais "alinhadas" as posições, maior será o peso $\alpha_{t,i}$ associado ao elemento e_i que será considerado na posição t do *decoder*.

O alinhamento $\alpha_{t,i}$ é baseado em um escore e dado por:

$$\alpha_{t,i} = softmax_i(\ score_{t,i}\ ) \tag{2.12}$$

A função $softmax$ é utilizada para garantir que o vetor de contexto (equação 2.11) é uma combinação convexa.

Para o cálculo do escore que define o peso de alinhamento (equação 2.12) existem

[5]Dados N pontos $x_1, x_2, \ldots, x_N \in \mathbb{R}^d$, uma combinação convexa desses pontos é um ponto $x \in \mathbb{R}^d$ que satisfaz $x = \sum_{i=1}^{N} \alpha_i x_i$, na qual $\alpha_i \geq 0$ e $\sum_{i=1}^{N} \alpha_i = 1$.

algumas variações. Bahdanau *et al.*, 2015 propuseram o *additive attention* que é calculada da seguinte forma:

$$score_{t,i} = V(tanh(W_1(e_i) + W_2(s_{t-1}))) \tag{2.13}$$

Trata-se da adição dos vetores e_i e s_{t-1}, ponderados respectivamente por W_1 e W_2, ao qual aplica-se a função $tanh$ e em seguida uma combinação linear ponderada por V. Esta equação pode ser implementada como uma rede neural simples, na qual os pesos V, W_1 e W_2 independem de t e são ajustados em conjunto com o restante da rede durante o treinamento.

Outras formas para o cálculo do escore são o *content-based* ($score_{t,i} = cosine[s_{t-1}, e_i]$ (Graves, Wayne *et al.*, 2014)), o *location-based* que não usa o conteúdo do *encoder* para o cálculo do escore (Luong *et al.*, 2015) e o *scaled dot-product* ($score_{t,i} = \frac{s_{t-1}^{\mathsf{T}} e_i}{\sqrt{n}}$) utilizado em *Transformers* (Vaswani *et al.*, 2017).

O mecanismo de atenção ajuda, portanto, a recuperar a dependência entre posições distantes através de referência a toda a sequência de entrada durante a geração de cada elemento da saída, sem onerar a complexidade do cálculo. Esse mecanismo já foi aplicado com sucesso em diversos problemas como reconhecimento de fala (Chorowski, Bahdanau, Cho *et al.*, 2014), geração de *caption* para imagens (Xu *et al.*, 2015), tradução (Bahdanau *et al.*, 2015) e leitura (Cheng *et al.*, 2016).

Capítulo 3

Trabalhos relacionados

Neste capítulo discorremos inicialmente de forma bastante breve sobre problemas do tipo *image-to-sequence*, no qual enquadra-se o problema de reconhecimento de escrita manuscrita (HTR, do inglês *Handwritten Text Recognition*). Em seguida, descrevemos algumas soluções recentes para o problema de HTR encontrados na literatura da área.

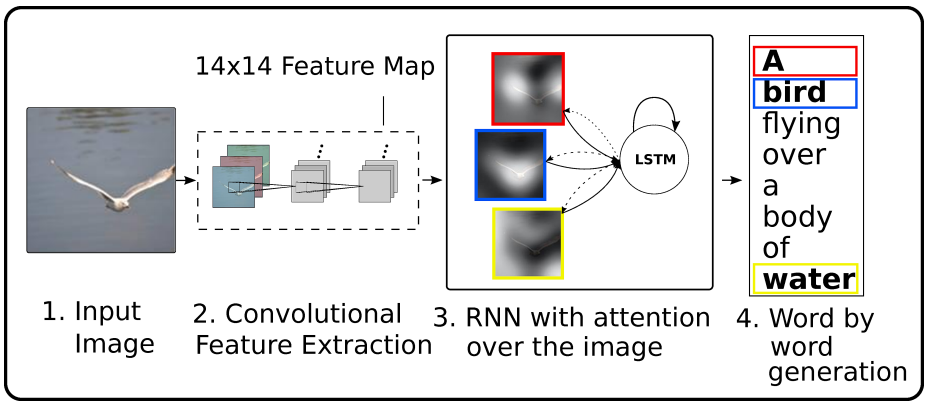
3.1 Problemas com saída do tipo sequência

O problema tratado neste projeto, o de leitura de notações de xadrez a partir de uma imagem de planilha, consiste em transformar o conteúdo textual de uma imagem para o formato texto digital. Neste processo, uma importante tarefa é o reconhecimento do texto manuscrito (ou HTR).

A produção de uma saída sequencial é comum em problemas do tipo *image-to-text* e *sequence-to-sequence*. São exemplos de problemas do tipo *image-to-text* a predição do próximo elemento da sequência no contexto de escrita online (Graves, 2013), a interpretação de fórmulas matemáticas manuscritas (J. Zhang *et al.*, 2017) e o *image captioning* (You *et al.*, 2016). São exemplos de problemas *sequence-to-sequence* a tradução de linguagem natural (Bahdanau *et al.*, 2015) e o reconhecimento da fala a partir de sinais de áudio (Graves, Mohamed *et al.*, 2013).

Dentre esses problemas, destacamos o problema de *image captioning* por ser um dos primeiros exemplos de aplicação do mecanismo de atenção para imagens e que estruturalmente é similar ao problema tratado neste trabalho.

O problema de *captioning* de imagens consiste na geração de um texto descritivo do conteúdo de uma imagem. Um modelo proposto para esse problema por Xu *et al.*, 2015 é ilustrado na figura 3.1. Pode-se ver que uma rede convolucional é usada para extração de *features* a partir da imagem de entrada. As *features* são passadas para uma rede recorrente com mecanismo de atenção, a qual gera um texto descritivo (uma sequência de palavras) de acordo com os pontos de atenção aprendidos.



fonte: Xu *et al.*, 2015

Figura 3.1: *Diagrama em alto nível da arquitetura de uma solução para o problema de* Image Captioning. *O modelo usa uma rede convolucional para extração de* features *e uma rede recorrente com mecanismo de atenção para a geração do texto de saída.*

3.2 Reconhecimento de escrita manuscrita

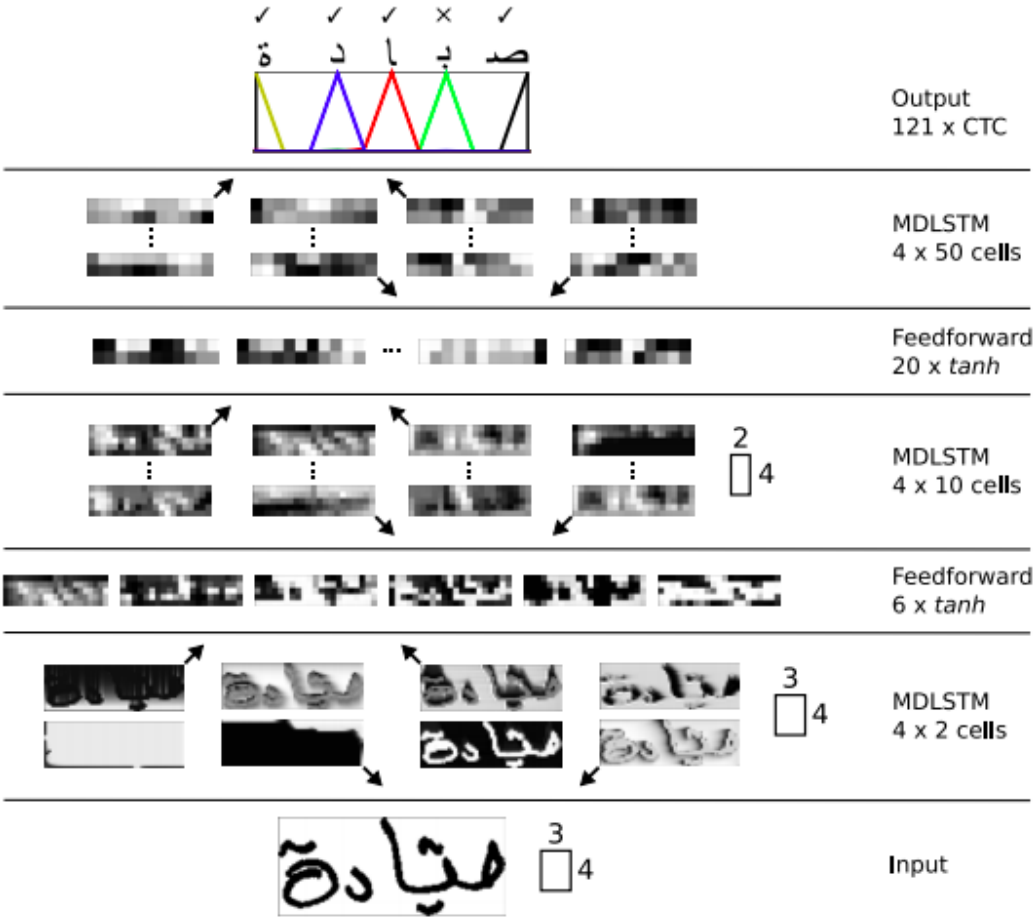
Nesta seção descrevemos três soluções baseadas em redes neurais profundas encontradas na literatura para o problema de HTR. Elas são amostras representativas das soluções existentes no início deste trabalho e são apresentadas em ordem cronológica, refletindo a evolução ao longo dos anos. Mais recentemente, um método que não utiliza recorrência foi proposta em Kang, Riba *et al.*, 2020, porém não incluímos aqui por ela utilizar um mecanismo distinto conhecido por *transformers* (Vaswani *et al.*, 2017).

Avaliação dos métodos HTR: O desempenho dos métodos HTR é geralmente descrito em termos de duas métricas: *character error rate* (CER) e *word error rate* (WER) (Frinken e Bunke, 2014). Essas métricas capturam a taxa de reconhecimento na granularidade de caracteres e palavras, respectivamente, e consideram o número de substituições, inserções e remoções necessárias (no contexto de comparação de *strings* é conhecida por *edit distance*) para transformar uma sequência predita para a correspondente sequência-alvo (*ground-truth*). Muitos dos trabalhos utilizam o *IAM Handwriting Database* [1], ou simplesmente IAM, um *dataset* público contendo amostras de textos manuscritos no estilo cursivo, devidamente rotulados.

3.2.1 Soluções baseadas apenas em redes recorrentes

Graves e Schmidhuber, 2009 usaram uma rede recorrente na *ICDAR 2009 Arabic handwriting recognition competition*, voltado para o reconhecimento de palavras manuscritas em árabe. Obtiveram uma acurácia de 91,4%, usando a arquitetura mostrada na figura 3.2. A entrada é a imagem de uma palavra e a saída esperada é a sequência de caracteres da palavra. A arquitetura emprega camadas MDLSTM (*Multi-directional LSTM*, uma extensão das camadas LSTM) e CTC (*Connectionist Temporal Classification*, que mapeia a saída da rede para uma sequência de caracteres de forma ótima), explicados brevemente a seguir.

[1] https://fki.tic.heia-fr.ch/databases/iam-handwriting-database



fonte: Graves e Schmidhuber, 2009

Figura 3.2: *Arquitetura do modelo para reconhecimento de texto manuscrito baseado em redes recorrentes usando MDLSTM. As imagens de entrada são inicialmente coletadas em janelas de 4x3* pixels, *que por sua vez são processadas por 4 camadas MDLSTM, uma para cada direção. Na figura a ativação das células da camada LSTM são apresentados separadamente, e as setas nos cantos das figuras indicam a direção da varredura. Em seguida o resultado da camada MDSLTM é agrupado novamente em janelas de tamanho 4x3, e encaminhado para uma camada* feed forward *com* tanh *como função de ativação. Este processo é repetido por mais 2 vezes até que o resultado da última camada MDSLTM seja uma sequência de 1 dimensão, que é por sua vez transcrito para o texto final através de uma camada CTC. Neste exemplo da figura todos os caracteres são corretamente mapeados exceto o penúltimo.*

MDLSTM (Graves e Schmidhuber, 2009): O LSTM, conforme explicado na seção 2.4, foi concebido originalmente para processar sequências de uma dimensão e em um determinado sentido. O nó LSTM mantém um estado ou memória ao longo do processamento da sequência, permitindo desta forma que se relacionem características de elementos distintos da sequência.

A ideia do MDLSTM é estender esta funcionalidade para n direções. Isso é feito através do uso de n conexões recorrentes (cada uma relacionada à posição anterior da sequência em cada uma das direções). Nós deste tipo são úteis para processar, por exemplo, o conjunto

de pixels de uma imagem.

CTC (Graves, Fernández *et al.*, 2006): O CTC (*Connectionist temporal classification*) é um algoritmo utlizado comumente com redes recorrentes, em situações quando a sequência de entrada possui tamanho distinto da sequência-alvo. No caso de reconhecimento de uma imagem de palavra, *features* extraídos da imagem são organizados de forma sequencial (por exemplo, cada coluna do mapa de *features* poderia ser um elemento na sequência) formando a entrada de uma rede recorrente. Já a saída-alvo para essa entrada é a sequência de caracteres da palavra presente na imagem. A rede recorrente gera, por construção, uma sequência de mesmo tamanho da sequência de entrada. Este tamanho pode ser diferente to tamanho da sequência-alvo. O CTC então trata de alinhar a sequência de saída com a sequência-alvo, de forma ótima. Como o tamanho da sequência de saída é tipicamente maior que o número de caracteres na palavra-alvo, é natural que múltiplos elementos da sequência de saída sejam alinhados com um mesmo caractere da palavra-alvo.

A seguir, a noção de alinhamento é detalhado um pouco mais por meio de um exemplo. Supondo que a imagem de entrada contém a palavra “Hello”, e a saída da rede contém 14 posições, podemos imaginar uma divisão da imagem em 14 fatias verticais. Cada uma das posições na sequência de saída corresponde à identificação do caractere na respectiva fatia. Supondo que a sequência de saída seja “HHHHeeellllooo”, ela precisa ainda ser alinhada com a sequência correta “Hello”. Porém, analisando-se apenas a sequência “HHHHeeellllooo” não é possível saber que “llll” corresponde a “ll” e não a “l”. Para resolver esse problema, o CTC permite que a codificação gerada pela rede contenha um caractere especial separador “-”. Assim, no exemplo anterior, para representar dois “l” consecutivos em 4 posições na saída, em vez de “llll” deve ser gerada “l-ll”, “ll-l”, “-l-l” ou “l-l-”. Para decodificar, reduz-se cada subsequência de caracteres repetidos para apenas um caractere, e em seguida elimina-se os caracteres separadores. Isto significa que para uma dada saída esperada, podem existir múltiplas codificações "corretas". Por exemplo, se a saída esperada é “ab”, e a rede gera uma codificação de tamanho 3, então as possibilidades para codificação dessa entrada são “aab”, “abb”, “a-b”, “-ab”, “ab-”, nas quais “-” indica o caractere especial separador. O CTC cuida de levar isso em consideração quando calcula o escore de alinhamento entre a saída gerada pela rede e a sequência-alvo. O *backpropagation* leva em consideração esse escore de forma a ajustar os pesos da rede para que ela gere saídas que melhor se alinhem à sequência-alvo nas próximas iterações.

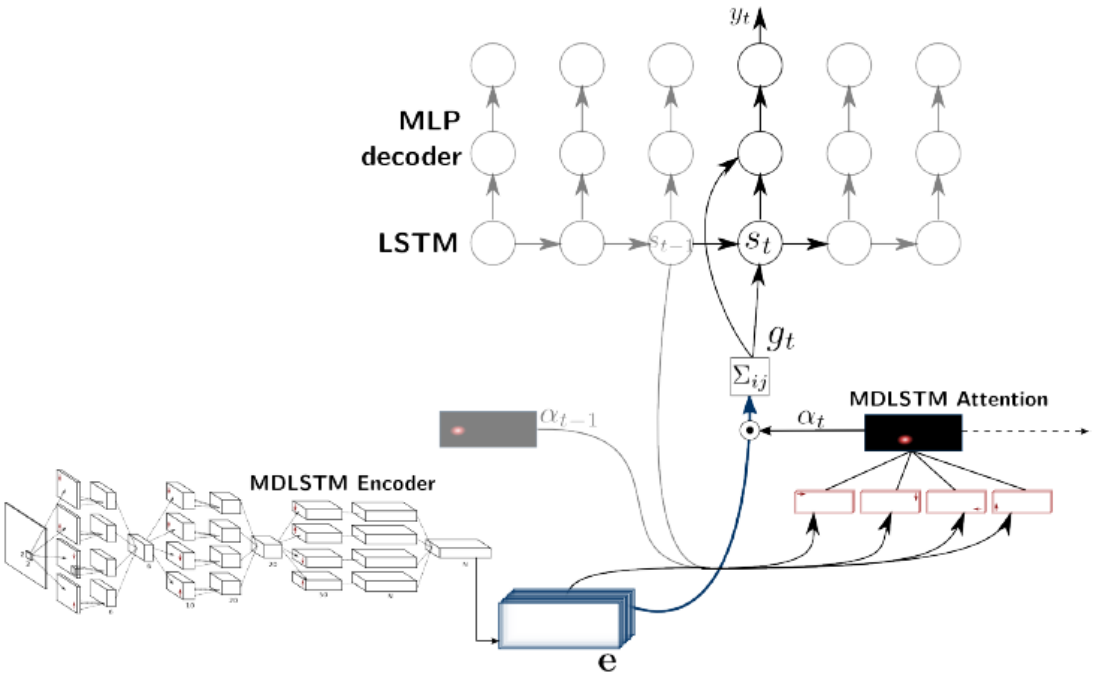
Uma vez que a entrada é transformada em uma representação sequencial considerando-se fatias verticais do mapa de *features*, a solução acima é apropriada para textos que ocupam apenas uma linha.

3.2.2 Soluções com recorrência e mecanismo de atenção

Modelos do estado da arte recentes para problemas de HTR e outros problemas análogos usam mecanismos de atenção (seção 2.6). Além de recuperar informações de sequências longas, o mecanismo de atenção permite ao modelo colocar o foco em pontos relevantes da entrada de acordo com a sequência-alvo da saída, e no caso de HTR pode ajudar na detecção da direção de leitura e na identificação da quebra de linha. A visualização do foco de atenção permite analisar quais elementos da entrada podem estar afetando o

comportamento da rede.

Bluche *et al.*, 2017: Esses autores propuseram um modelo que alterna camadas de MDLSTM e convolução com *subsampling* seguidos por uma camada recorrente com mecanismo de atenção (figura 3.3) para textos com múltiplas linhas.

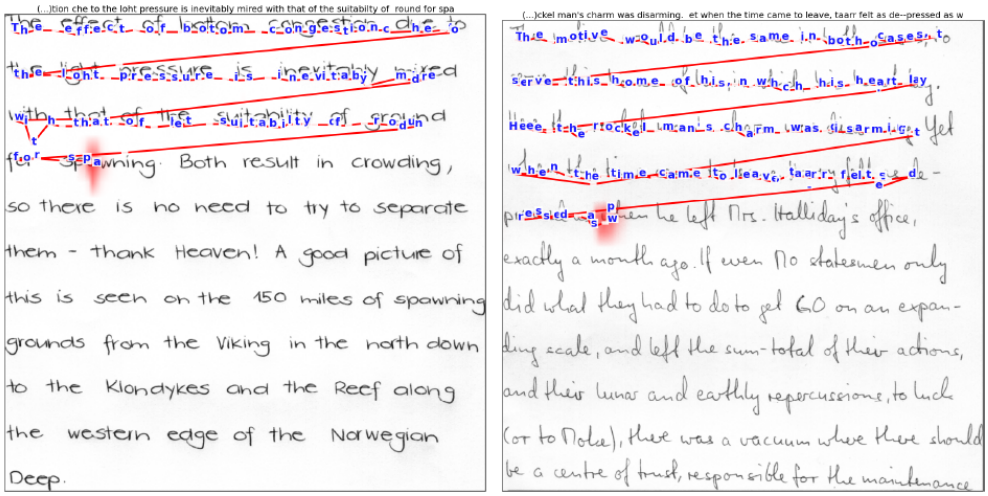


fonte: Bluche *et al.*, 2017

Figura 3.3: *Exemplo de rede combinando camadas convolucionais e módulo recorrente do tipo* encoder-decoder *com MDLSTM.*

De forma geral, a arquitetura é do tipo *encoder-decoder* com camada de atenção. O *encoder* consiste de um MDLSTM que gera uma sequência e sobre o qual atua a camada de atenção para gerar o vetor de contexto, de forma similar ao explicado na seção 2.6. O *decoder* deste modelo não usa, porém, o resultado da posição anterior y_{t-1} para a predição na posição seguinte. Em vez disso, a camada de atenção utiliza a informação do alinhamento anterior α_{t-1}. Além disso, o módulo *MDLSTM Encoder* precisa passar por um pré-treinamento para reconhecimento de palavras e de linhas, antes de a rede ser treinada sobre a base de parágrafos para o aprendizado da quebra de linhas. Neste sentido, o treinamento não é totalmente ponta a ponta.

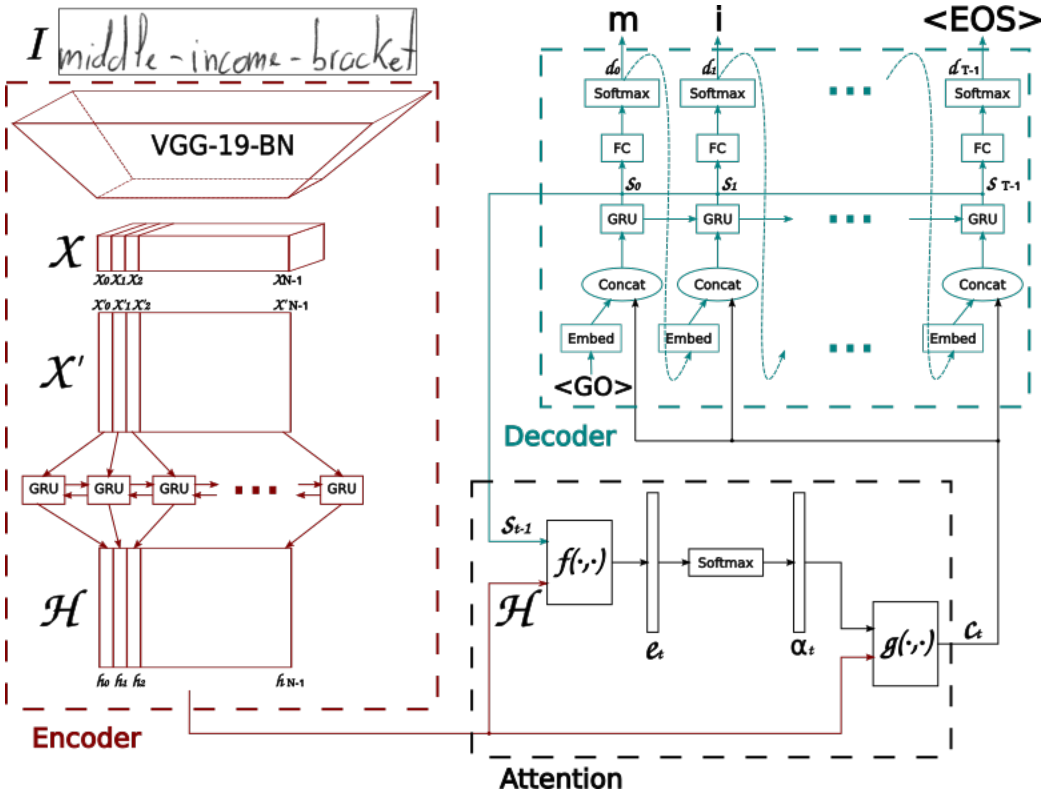
O resultado obtido sobre a base de parágrafos do IAM foi CER de 16,2%, demonstrando a capacidade do modelo fazer a leitura de um parágrafo inteiro sem segmentação explícita de linhas ou palavras. A figura 3.4 mostra o foco de atenção deslizando sobre a imagem na direção correta de leitura e com a correta identificação da quebra de linhas.



fonte: Bluche *et al.*, 2017

Figura 3.4: *Visualização do mecanismo de atenção em ação na leitura de uma página inteira, com destaque para a correta detecção da direção de leitura e da quebra de linhas.*

Kang, Toledo *et al.*, 2018: Esses autores utilizaram um modelo baseado em mecanismo de atenção sobre o *dataset* IAM em nível de palavras. A arquitetura utilizada é ilustrada na figura 3.5. O modelo processa imagens com uma linha de texto. Comparado com o trabalho de Bluche *et al.*, 2017, este modelo não requer o pré-treinamento por CTC e portanto o treinamento é realizado ponta a ponta.



fonte: Kang, Toledo *et al.*, 2018

Figura 3.5: *Modelo* encoder-decoder *com mecanismo de atenção para reconhecimento de palavras.*

O modelo é composto por um *encoder* que consiste de um módulo de CNN e uma camada GRU bidirecional, seguido por uma camada de atenção, e por um *decoder* que é também uma rede recorrente, porém com camada GRU unidirecional.

Os autores testaram dois mecanismos de atenção, o *content based* e o *location based*. O *location based* é basicamente igual ao *content-based*, mas com uma codificação explícita da informação posicional no *encoder*, adicionada na equação da atenção. De acordo com os autores, o *location-based attention* (Chorowski, Bahdanau, Serdyuk *et al.*, 2015) pouco contribui, levando-os à conclusão de que o GRU bidirecional por si só já codifica alguma informação posicional no mapa de *features* resultante do *encoder*.

O resultado obtido foi CER de 6,88% e WER de 17,45% sobre a base de palavras do IAM *dataset*.

Dentre os três trabalhos acima, a solução de Kang, Toledo *et al.*, 2018 é a que mais se assemelha com a desenvolvida nesta dissertação. Porém um diferença fundamental é que Kang, Toledo *et al.*, 2018 tratam textos com apenas uma linha, enquanto o problema tratado neste trabalho é de múltiplas linhas.

Capítulo 4

Método

Neste e nos próximos capítulos descrevemos o desenvolvimento realizado para resolver o problema de leitura de planilhas de xadrez. Conforme já mencionado anteriormente, não encontramos na literatura trabalhos publicados sobre esse problema em específico. Isto significa que no momento inicial do desenvolvimento não havia uma solução que pudesse ser utilizada como referência ou ponto de partida. Também não encontramos *datasets* de treinamento para este problema.

Diante desta situação, decidimos por criar um *dataset* minimamente utilizável e explorar arquiteturas compostas por um módulo convolucional e um módulo recorrente *encoder-decoder* com mecanismo de atenção, baseada em uma solução para o problema de *image captioning* proposta em Xu *et al.*, 2015 e brevemente descrita na seção 3.1. A justificativa para a escolha dessa arquitetura de rede deve-se à semelhança estrutural entre o problema tratado em *image captioning* e o problema tratado neste trabalho, quando consideramos a entrada e a saída. Em outras palavras, ambos os problemas têm como entrada uma imagem e como saída um texto (sequência de palavras), embora a natureza das mesmas seja completamente distinta de um problema para o outro.

O desenvolvimento do trabalho envolveu uma ampla investigação experimental. Neste capítulo descrevemos o que pode ser visto como o aspecto “materiais e métodos” deste trabalho. Primeiramente detalhamos o *dataset* (imagens de planilhas reais) e outras imagens geradas a partir dele, utilizadas nos experimentos. Em seguida, descrevemos a organização dos experimentos realizados, destacando os objetivos e eventualmente antecipando parte dos resultados. Por último, apresentamos a configuração experimental básica, uma configuração de arquitetura, dados de entrada e de procedimentos e parâmetros de treinamento identificados por meio dos primeiros experimentos e que culminou em um "modelo referência", que pode ser treinado ponta a ponta. Este modelo serviu de referência para o restante dos experimentos. Os experimentos em si são descritos detalhadamente no capítulo seguinte.

4.1 Dataset

4.1.1 Imagens reais de torneios

Realizamos a primeira coleta de planilhas de torneios durante o carnaval de 2020. Devido à pandemia de COVID-19, os torneios presenciais foram logo suspensos. Por conta disso e pelo fato de as planilhas de torneios serem descartadas após cadastradas no sistema computacional (base de jogos), foi possível realizar apenas uma coleta. As planilhas coletadas contém anotações no padrão SAN, em língua portuguesa.

Limpeza de dados

Planilhas de torneio podem conter uma diversidade de erros ou anotações inadequadas que podem dificultar o treinamento de algoritmos de *machine learning*. Por exemplo, elas podem conter anotações simplificadas que não seguem estritamente o padrão SAN, sendo portanto ambíguos, ou anotações com rasuras, ou anotações em outras línguas como o Inglês ou Espanhol, distintas do Português (adotado neste trabalho). Na figura 4.1 mostramos alguns exemplos de casos considerados inadequados.

Na inserção dos lances na base de jogos, o leitor humano é muitas vezes capaz de fazer a leitura correta apesar da presença dos erros de anotação. Para tanto, a pessoa pode contar com sua capacidade de inferência baseada na situação do jogo naquele ponto, ou em sua experiência na leitura de planilhas ou até mesmo perguntando aos próprios jogadores. Adicionalmente, uma vez que em cada partida há as anotações de ambos os jogadores, pode-se também fazer um cruzamento entre as respectivas planilhas para elucidação de dúvidas. As partes da planilha que são indecifráveis são simplesmente cadastradas como ilegíveis, resultando em cadastros parciais (apenas até onde pode ser lido) ou em casos extremos a planilha pode até ser ignorada totalmente. Isso, no entanto, não invalida o resultado do jogo em si. É importante mencionar que a sequência de lances cadastrada no sistema é a sequência de fato ocorrida no jogo, mesmo que a anotação contenha erros.

A filtragem de planilhas problemáticas foi um dos primeiros passos na construção do *dataset*. Para cada planilha, foi recuperado o registro no formato PGN do site *chess-results.com*, onde ficam cadastrados todos os jogos. Esse processo de cadastro ocorre geralmente logo após uma partida. Num primeiro momento, planilhas com erros aparentes de anotação tais como anotações que omitiam um ou mais lances ou anotações que não correspondiam "exatamente" à sequência constante no arquivo PGN foram removidas.

Uma das razões da não correspondência exata entre a escrita e o PGN são os casos ambíguos. Exemplos de casos ambíguos relacionam-se com alguns símbolos que são opcionais na anotação manuscrita, tais como o "`x`", o "`!`", o "`#`". Por exemplo, na anotação "`Bxd3`", movimento do Bispo com tomada de peça, o "x" é opcional. Portanto, neste caso o jogador poderia igualmente anotar "`Bd3`". Rigorosamente, este último corresponde ao mesmo lance, porém sem a tomada de peça. Quando há tomada de peça e o jogador anota "`Bd3`", pode-se inferir facilmente pelo contexto do jogo que a anotação rigorosa deveria ser "`Bxd3`". No cadastro do jogo, insere-se a anotação rigorosa. Desta forma, podem ocorrer casos nos quais na imagem da planilha vê-se "`Bd3`", porém no arquivo PGN (*ground-truth*) a mesma é identificada como "`Bxd3`" (Figura 4.2). Isso criaria uma situação de rótulos

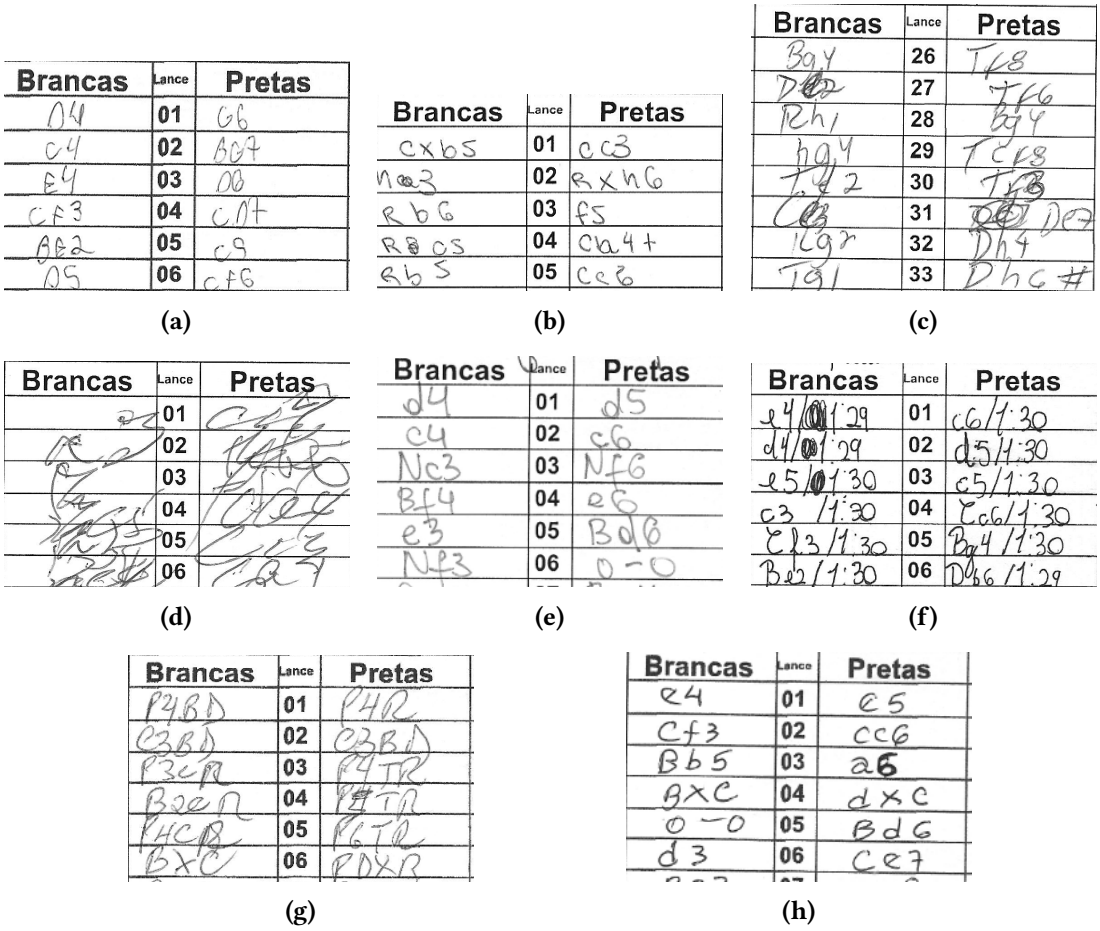


Figura 4.1: *Exemplos de anotações reais, mas inadequadas para o* dataset *de treinamento: (a) e (b) não fazem distinção entre maiúsculo e minúsculo (por exemplo, C de cavalo e c de peão); (c) contém rasuras que tornam a escrita ilegível em determinados pontos; (d) é totalmente ilegível; (e) está em inglês (cavalo anotado como N e não C); (f) contém anotações adicionais, no caso a anotação de tempo restante de jogo; (g) não adere ao padrão SAN ("P4"); (h) a notação está simplificada (por exemplo, BxC omite a posição da peça).*

conflitantes no *dataset*.

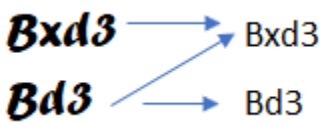


Figura 4.2: *Exemplo de caso de ambiguidade em que uma mesma imagem de lance pode ser mapeada para 2 lances diferentes.*

Rotulação e Recortes

Após a remoção de planilhas problemáticas, elas foram escaneadas (a 300 dpi) e rotuladas a partir dos respectivos arquivos PGN obtidos do cadastro. O rótulo, *ground-truth*, é uma sequência de códigos numéricos, correspondendo aos lances da partida.

As imagens do *dataset* consistem de recortes das imagens escaneadas. Esses recortes correspondem à região na imagem da planilha que estão efetivamente preenchidos com os lances. Decidimos não utilizar imagens de planilhas inteiras pois elas contêm outras informações tais como nome do jogador, o que tornaria a tarefa de leitura ainda mais complexa. Para possibilitar a exploração de diferentes condições de entrada, foram gerados recortes com diferentes números de lances.

Tipo de recorte	Tamanho da sequência	Total escaneados	Qtd após *cleanup* inicial	Qtd final experimentos
1 lance	1		10944	
1 linha	2		5480	
2 linhas	4		4788	492 (train: 378, test:114)
4 linhas	8		684	
8 linhas	16		684	492 (train: 378, test:114)
Página cheia	variável	853	684	

Tabela 4.1: Dataset *de recortes reais gerados a partir de 853 planilhas de torneio. A quantidade de recortes de um lance, de uma linha e de duas linhas é maior do que o número de planilhas porque foram feitos recortes iniciados em lances arbitrários da partida (janelas móveis), não necessariamente no lance inicial. A "Qtd final experimentos" indica o número de imagens de torneio usados nos experimentos.*

Das planilhas coletadas resultaram 684 planilhas após a filtragem das problemáticas. A tabela 4.1 mostra a constituição do *dataset*, indicando quais tipos de recortes e quantos de cada tipo foram gerados. Denominaremos esses recortes como **recortes reais**, para diferenciá-los dos recortes artificiais descritos adiante. Os recortes foram realizados a partir do primeiro lance do jogo, exceto nos casos de recortes de um lance, de uma linha (2 lances) e de duas linhas (4 lances). Nestes três casos, foram realizados recortes de regiões com lances que não necessariamente correspondem ao início da partida (e esta é a razão de haver mais amostras para esse tipo de recortes).

4.1.2 Imagens artificiais

Recortes transcritos

Os **recortes transcritos** foram extraídos de planilhas de xadrez preenchidas por voluntários que transcreveram sequências de lances de diversos jogos. O número de voluntários foram 5, mas houve predominância maior de 2 voluntários em particular. As sequências transcritas foram extraídas de bases de dados de jogos reais tais como o `Kaggle.com - 35 Million Chess Games`[1].

Planilhas preenchidas desta forma tendem a ser mais legíveis e a conterem menos erros e ambiguidades pois os voluntários realizaram a tarefa concentrados em somente

[1] https://www.kaggle.com/milesh1/35-million-chess-games/version/1

copiar a sequência, sem necessidade de se concentrarem no jogo em si, como ocorre com os jogadores durante as partidas. Foram geradas um total de 2010 planilhas transcritas com sequências de partidas distintas.

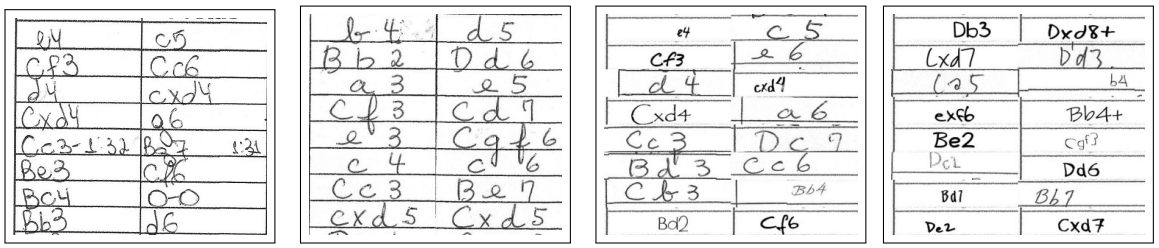
Recortes mosaico

Os **recortes mosaico** foram gerados artificialmente por composição de imagens de lances. Para facilitar a criação desse tipo de recorte, criamos uma **biblioteca de imagens de lances** da seguinte forma. Para cada lance (vocábulo) foram reunidas 250 imagens, dando-se prioridade para as imagens manuscritas (extraídas de imagens reais, de torneios e as transcritas). Quando não havia um número suficiente desses, foram geradas imagens com fontes *trueType*. Essa biblioteca de lances foi cuidadosamente revisada, imagem por imagem, para garantir que todas as anotações estejam legíveis e com o rótulo correto.

Dois tipos de recortes mosaico foram utilizados neste trabalho. Um contendo sequências de jogos reais extraídos da base de jogos cadastrados, e outro contendo lances em ordem randômica.

Exemplos dos quatro tipos de recortes são mostrados na figura 4.3.

A quantidade de imagens obtidas desta forma é limitada pelo tamanho da base de dados de jogos, que contém milhões de jogos. Portanto, podemos considerar que essa quantidade é quase ilimitada. O mesmo podemos dizer para os recortes randômicos.



(a) (b) (c) (d)

Figura 4.3: *Exemplos de recortes. Da esquerda para a direita: (a) recorte real de torneio, (b) recorte transcrito, (c) recorte mosaico com sequencia real, (d) recorte mosaico com sequência aleatória.*

4.2 Investigação experimental

A investigação experimental consistiu de duas principais partes, explicadas a seguir.

4.2.1 Experimentos exploratórios

Inicialmente fizemos experimentos exploratórios, com o objetivo de validar a arquitetura, investigar a complexidade do problema, e também a consistência dos dados.

Ambiente: Os treinamentos exploratórios foram realizados na plataforma *Google Colab* em *notebooks* usando-se o *Tensorflow* versão 2.3.0, com máquinas Linux com GPU Nvidia

Tesla K80/T4/P100-PCIE-16GB, processador Intel(R) Xeon(R) 2.20GHz, memória RAM de 13G e disco de 37G. O *dataset* foi armazenado no *Google Drive*, assim podendo ser acessado a partir do ambiente do *Google Colab*.

Dados de treinamento: Na fase exploratória foram inicialmente utilizados recortes reais (provenientes das planilhas reais de torneio; ver Tabela 4.1), de tamanhos distintos, divididos em treinamento e validação. Posteriormente, utilizamos também os recortes transcritos e os recortes mosaico.

Os experimentos exploratórios foram importantes para constatarmos que as redes convolucionais são suficientes para o reconhecimento de palavras individuais, e que a camada de atenção é imprescindível para o alinhamento correto. Concluímos que um modelo completo consistindo de um módulo convolucional, uma rede recorrente *encoder-decoder* e com mecanismo de atenção, possui capacidade para processar de forma ponta a ponta as imagens de planilha de xadrez (especificamente, os recortes).

Um dos principais resultados desses experimentos exploratórios foi a identificação de uma configuração de arquitetura, assim como dos dados de entrada e de parâmetros de treinamento, que chamamos de **configuração básica**, que se mostrou adequada para treinamentos ponta a ponta bem sucedidos.

Além disso, constatamos que o problema como um todo constitui-se de subtarefas com características distintas e que, para um treinamento ponta a ponta bem sucedido, as condições de treinamento devem ser tais que essas subtarefas sejam aprendidas conjuntamente. Em condições não adequadas, há tendência do treinamento favorecer o aprendizado de uma subtarefa em detrimento de outra, comprometendo o desempenho final do modelo.

Uma vez que esses experimentos foram realizados sem planejamento, eles não consideram situações diretamente comparáveis entre si. Desta forma não incluímos a descrição deles no corpo desta dissertação e acreditamos que isso não traz prejuízo à compreensão do texto. Parte desses experimentos, porém, estão descritos detalhadamente no apêndice C, para uma possível leitura complementar.

4.2.2 Experimentos relacionados ao aprendizado de subtarefas

Na segunda etapa das investigações, o foco foi um estudo sobre o aprendizado de cada uma das subtarefas anteriormente identificadas, individualmente. Para cada subtarefa, investigamos como certos fatores afetam o aprendizado.

Todos os experimentos nessa segunda etapa de investigação foram realizados de forma melhor controlada, baseados na configuração básica mencionada anteriormente e descrita na próxima seção.

Os experimentos em si são descritos no próximo capítulo (capítulo 5).

4.3 Modelo referência

Descrevemos aqui a **configuração experimental básica** que foi adotada a partir dos experimentos relacionados ao aprendizado das subtarefas. Denominamos **modelo referência** o modelo obtido a partir do treinamento de acordo com essa configuração básica.

Ambiente: Os treinamentos a partir dos experimentos controlados e voltados ao estudo sobre aprendizado de cada uma das subtarefas foram realizados em ambiente local, como aplicação Python via linha de comandos, e não mais em notebooks. Todos os fontes encontram-se disponíveis em https://github.com/sergiohayashi/chess-attention.thesisHayashi2021. A máquina local utilizada possui um processador Intel i7-97F00F com 8 cores, sistema operacional windows 10 Enterprise, GPU RTX-2060 de 6Gb de memória dedicada, 32Gb de memória convencional, 1TB de disco do tipo NVMe. O software utilizado foi o Python 3.9.6 e o Tensorflow 2.5.0 com CUDA 11.2.

Adotamos como base do código uma implementação da arquitetura proposta em Xu *et al.*, 2015, disponível no tutorial do Tensorflow (https://www.tensorflow.org/tutorials/text/image_captioning). A partir daí foram feitos ajustes trazendo elementos da arquitetura usada em HTR e descrita em Kang, Toledo *et al.*, 2018, além de outros ajustes finos de acordo com os experimentos.

Predição em nível de palavras: O reconhecimento de escrita pode ser modelado tanto como um problema de predição sequencial de caracteres individuais quanto de palavras individuais. Neste trabalho optamos por por predição de palavras e não de letras individuais. Uma primeira razão para isso é o fato de, comparando com uma linguagem natural, o problema tratado aqui possuir um vocabulário bem mais reduzido. Outra razão é que desta forma todas as saídas geradas serão necessariamente vocábulos válidos no domínio do xadrez (o que não poderia ser garantido caso a predição fosse em nível de caracteres). Além disso, ao considerarmos palavras como unidades a serem preditas, permitimos ao modelo abstrair padrões de jogos implícitos nas sequências de lances.

Tamanho do recorte: Levando-se em consideração os requisitos de recursos computacionais, resolvemos por focar em subproblemas de recortes de 8 linhas. Os recortes de oito linhas, que correspondem aos dezesseis lances iniciais, contemplam as principais estruturas de interesse: sentido da leitura (da esquerda para a direita e de cima para baixo), quebra de linha, e reconhecimento dos lances propriamente. Consequentemente, o *dataset* com o qual trabalhamos são imagens de recortes de 8 linhas, e não de página cheia. Todos os recortes têm exatamente 16 lances, portanto não estamos considerando os casos de sequências menores ou maiores do que 16 lances.

Conjunto de teste e avaliação: O conjunto de teste é formado por recortes de 114 imagens de torneio, disjunto dos dados de treinamento. A avaliação é feita em termos de acurácia (lances corretamente identificados sobre o total de lances processados), e em algumas situações em termos de CER (de fato, calcula-se o CER com respeito a cada palavra na sequência-alvo e em seguida toma-se a média). O CER é a métrica comumente

conhecida como *edit distance* (ver também seção 3.2). Note que o modelo faz a predição em nível de palavras (ou lances), portanto a acurácia nos dá o índice de acerto em termos de lances. Por outro lado, o CER que é calculado em nível de caracteres fornece uma noção complementar sobre o desempenho do modelo. Por exemplo, quando o modelo troca "Bd3" por "Bb3", isto é contabilizado como erro no cálculo da acurácia, enquanto o CER leva em consideração a similaridade entre as palavras. Uma observação quanto a outra métrica, WER, comumente utilizada em reconhecimento de escrita: o WER é o equivalente ao CER, porém calculado em nível de palavras. Enquanto em reconhecimento de texto a métrica faz sentido (uma palavra ausente ou reconhecida como sendo duas palavras podem impactar bastante a acurácia, mas pouco o WER), no caso das planilhas de xadrez as palavras estão naturalmente bem separadas uma das outras, fazendo com que a acurácia seja suficiente.

Dados de treinamento: 5000 recortes, extraídas de todas as imagens de torneio (exceto as 114 de teste) e de todas as planilhas transcritas manualmente, e o restante consistindo de recortes mosaico (ver Seção 4.1.2) com sequências de jogos reais.

Vocabulário: Como estamos lidando com um subproblema de sequências com tamanho $M = 16$, rigorosamente o vocabulário deveria conter todos os lances distintos que podem ocorrer nos 16 primeiros movimentos de qualquer jogo. Foi considerado, no entanto, a variedade de lances que ocorrem nas oito primeiras linhas das 114 planilhas no conjunto de teste e das 2010 planilhas transcritas, totalizando 175 lances distintos. Juntamente com os quatro vocábulos especiais[2] (`unk`, `pad`, `start` e `end`), o vocabulário considerado consiste de um total de 179 possíveis classes em cada posição predita. Os eventuais lances que ocorrem no conjunto de treinamento e não fazem parte desse vocabulário de 175 lances foram tratados como `unk`.

Arquitetura da rede: A arquitetura e as dimensões dos dados que se mostraram adequadas para os recortes de 8 linhas, são mostradas na figura 4.4.

Como módulo convolucional foi usado a rede VGG16 pré-treinada com ImageNet, portanto com pesos fixos (que não são treinados com o restante da rede), sem suas camadas finais totalmente conectadas de classificação. Do módulo convolucional, foi usada a penúltima camada para a extração de *features*, para termos uma granularidade maior para o restante da rede poder atuar. O tamanho da entrada, na primeira camada da convolução, em função do subproblema de 8 linhas, foi fixada em (800, 862, 3), resultando portanto em uma mapa de *features* de dimensão de (50, 53, 512) para ser usado pelo módulo recorrente.

O *encoder* consiste de uma rede recorrente bidirecional com nós do tipo GRU para codificação da informação posicional, seguida de uma rede totalmente conectada aplicada em cada posição da sequência, e uma camada de ativação *ReLU*. Entre o módulo convolucional e a rede recorrente os dados são vetorizados de um formato (h, w, d) para (h × w, d), sendo h a altura, w a largura e d a profundidade do mapa de *features*.

[2] Os `unk`, `pad`, `start` e `end` são usados comumente em redes para indicar, respectivamente, uma palavra desconhecida (fora do vocabulário adotado), preencher uma posição em caso de dado inexistente, início da sequência e final da sequência.

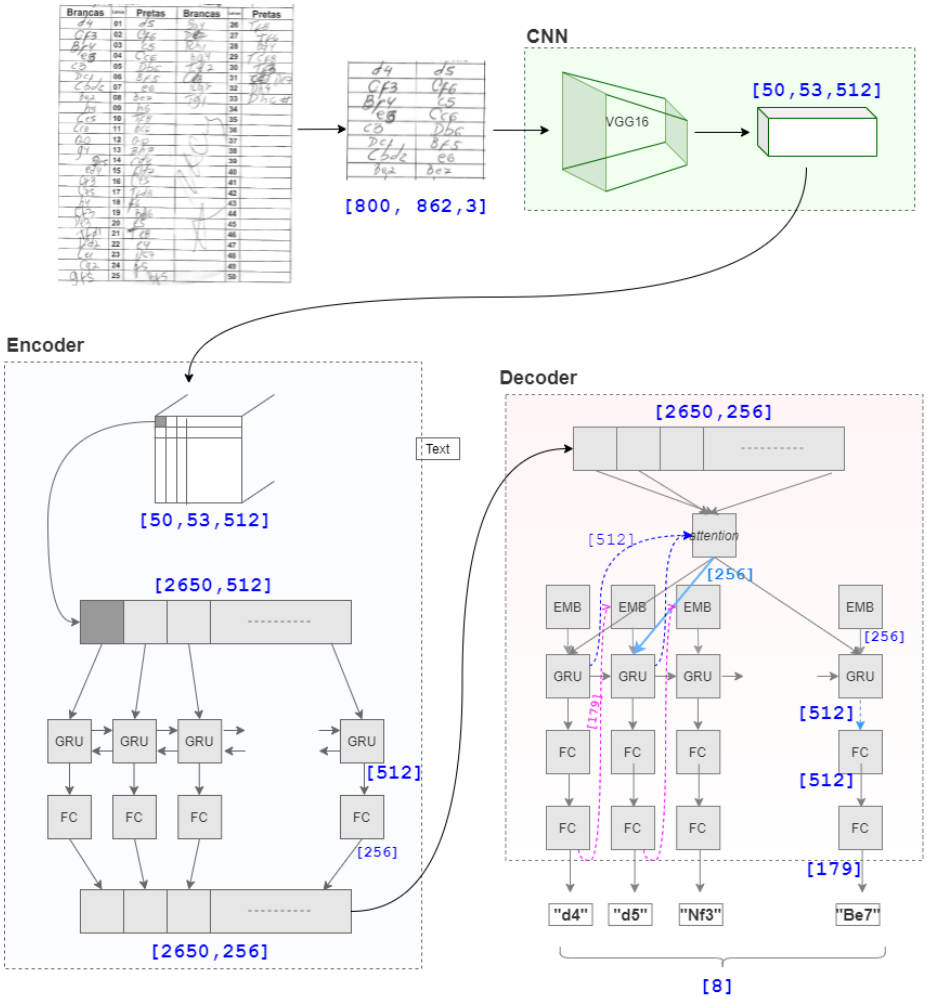


Figura 4.4: *Arquitetura do modelo referência.*

O *decoder* é constituído por uma camada de *embedding*[3], seguida por uma rede recorrente unidirecional com nós do tipo GRU, e duas camadas totalmente conectadas. A camada recorrente no *decoder* recebe o vetor de contexto gerado pela camada da atenção, a partir do estado oculto da posição anterior no *decoder* e o escore do tipo aditivo (Seção 2.6), relativo a cada elemento na sequência de saída do *encoder*.

A representação interna dos nós nas camadas recorrentes, bem como as camadas totalmente conectadas internas do mecanismo de atenção tem dimensão 512. A camada de *embedding* no *decoder* tem dimensão 256. Tanto no *encoder* como no *decoder* foram inseridas camadas de *dropout* com taxa 0.2. No apêndice B.1 é anexado o código fonte referente à implementação da rede, para o leitor interessado.

Treinamento:

- ***Learning rate*:** O *learning rate* foi fixado em 0.0005, e o *batch size*, em 16.

[3] A camada de *embedding* trata de mapear a identidade de uma palavra para a correspondente representação interna da rede.

- **Função de custo:** Uma vez que modelamos a saída como uma sequência de palavras em um dicionário restrito (vocabulário), é esperado que a cada posição t da sequência o modelo identifique a t-ésima palavra na imagem de entrada. Desta forma a função de custo utilizada é o *categorical cross-entropy* com logits, implementado pela função `tf.keras.losses.SparseCategoricalCrossentropy`. Esta função aceita como parâmetros de entrada a predição e o valor esperado. O valor esperado é passado como um número inteiro representando a posição no vocabulário. A predição é passada como um vetor de mesmo tamanho do vocabulário, no qual cada posição representa a probabilidade do resultado ser a palavra daquela posição no formato de valor *logit* da probabilidade. O cálculo desta função é explicado em mais detalhes no apêndice A.

- **Algoritmo de otimização e critério de convergência:** Para a otimização da função de custo foi utilizado o Adam (Kingma e Ba, 2014) , implementado em `tf.keras.optimizers.Adam`.

  O critério de parada de treinamento, na parte experimental, foi estabelecido como sendo *loss* de 0.25 ou acurácia em treinamento de 0.90.

- **Teacher forcing:** O mecanismo de *teacher forcing* foi mantido habilitado.

  Alguns modelos de redes recorrentes utilizam, entre outras informações, o resultado de predição na posição $t - 1$ para computar a predição na posição t. No processo de treinamento, a rede precisaria então aprender a predizer o próximo elemento a partir de predições anteriores, que podem estar com erros grandes, principalmente no início do treinamento. Isto pode resultar em dificuldade de convergência, instabilidade e desempenho baixo.

  A técnica denominada *teacher forcing* (Lamb *et al.*, 2016) foi proposta justamente para melhorar esses aspectos. Nesta técnica, durante o treinamento, para calcular no *decoder* a predição em uma dada posição t, passa-se a saída correta (*ground-truth*), ao invés da saída da posição anterior $t - 1$ calculada pelo modelo.

  O efeito esperado é que em cada posição da sequência a rede aprenda mais rapidamente uma vez que não precisa lidar com o erro acumulado ao longo da sequência. Esta técnica é aplicada principalmente em problemas como a construção de um modelo de linguagem ou predições de séries temporais nos quais a predição em uma posição depende unicamente dos elementos anteriores.

  Na aplicação da rede assim treinada, uma desvantagem é o fato de a resposta correta da posição anterior não estar disponível. Assim, o desempenho de teste tende a cair em geral.

- **Pré-processamento da imagem e cache em disco:** O processamento das imagens de entrada pela rede convolucional é realizada antes do treinamento e então o resultado é armazenado em disco para efeito de cache. Isso é feito para amenizar o requisito de memória necessária para esta camada da rede. Não havendo restrição de memória, esse processamento pode ser feito durante o treinamento.

- **Laço de treinamento:** No modelo *encoder-decoder*, conforme explicado na seção 2.5, uma sequência inteira é passada ao *encoder* e ele produz uma sequência de saída

e um estado final. Já na parte do *decoder*, muitas vezes é conveniente que um laço externo cuide passo a passo do processamento, pois o cálculo de sua saída num dado instante t pode depender cálculos realizados de forma externa à rede recorrente (como a atenção), o que impede que a rede *decoder* seja alimentada diretamente com uma sequência inteira.

No laço, o erro de predição é calculado para cada posição da sequência, porém o *back propagation* é aplicado, a partir dos erros acumulados, apenas ao final de uma iteração do laço, após a sequência inteira ter sido processada.

Uma flexibilidade decorrente do laço de treinamento é o treinamento incremental no tamanho da sequência. Em vez de se realizar o treinamento usando sequências longas inteiras logo de início, pode-se iniciar o treinamento com subsequências menores, que são gradativamente aumentados a cada vez que a rede atinge convergência para um determinado tamanho. Isso foi usado em alguns casos exploratórios para sequências mais longas, mas não nos experimentos comparativos.

No *Tensorflow* a aplicação do gradiente descendente e o laço de treinamento podem ser feitos automaticamente através da função `fit`, mas neste projeto o laço de treinamento é controlado na aplicação. Neste caso é utilizado a função `GradientTape` que coleta os valores passados pela rede no cálculo da inferência e posteriormente através da função `apply_gradientes` do *optimizer* aplica-se explicitamente o gradiente às variáveis treináveis da rede. No apêndice B.2 é anexado o trecho do código fonte relacionado ao laço de treinamento, para o leitor interessado em detalhes de implementação.

Capítulo 5

Subtarefas e fatores que afetam o aprendizado ponta a ponta

Neste capítulo descrevemos os experimentos realizados para investigar o comportamento do modelo referência em relação ao aprendizado de subtarefas subjacentes ao problema de leitura de planilhas de xadrez. Essas subtarefas foram identificadas durante os experimentos exploratórios e são três: (1) a predição do lance baseado na previsibilidade, (2) o alinhamento entre a atenção visual e a sequência-alvo, (3) o reconhecimento do lance baseado na informação visual.

Em uma análise a posteriori, essas subtarefas são óbvias, porém não estavam claras no início. É natural que o modelo necessite, ao longo da leitura, ajustar sucessivamente o foco de atenção para o local correto na imagem. Esse comportamento pode ser entendido como a habilidade do modelo seguir a ordem correta de leitura, isto é, identificar a direção de leitura (esquerda para a direita), reconhecer uma mudança de linha (de cima para baixo) e delimitar corretamente na imagem a região que contém a escrita do lance a ser reconhecido. Uma vez o foco estando no local correto, resta decodificar a informação visual para se fazer o reconhecimento da escrita propriamente dita. Apenas essas duas subtarefas, se aprendidas com sucesso, são suficientes para a leitura automática das planilhas de xadrez. Além disso, também é natural que os padrões de jogo impliquem um certo grau de previsibilidade (após uma determinada sequência de lances, certos lances são mais prováveis do que outros na próxima jogada). Essa previsibilidade, por sua vez, pode ajudar a leitura. O resultado do aprendizado da previsibilidade pode ser pensado como um modelo de linguagem. No caso de leitura feita por humanos, o conhecimento sobre a língua ajuda a antecipar a próxima palavra, a partir do que já foi lido até aquele ponto. Tal capacidade requer, em geral, um conhecimento contextual.

Baseado nos experimentos anteriores, observamos que algumas dessas subtarefas se mostraram mais fáceis de serem aprendidas do que as outras. Portanto, os experimentos deste capítulo buscam caracterizar quais são os fatores que afetam o aprendizado dessas subtarefas, a quais partes do modelo cada subtarefa está mais relacionada, e como o aprendizado de uma subtarefa afeta o aprendizado de outra.

Dentre as 3 subtarefas, tivemos fortes indícios de que a mais fácil é o aprendizado da

previsibilidade da sequência, depois o alinhamento, e por fim o reconhecimento dos lances individuais. Portanto, as investigações referentes às subtarefas são apresentadas seguindo essa ordem de dificuldade de aprendizado.

5.1 Comportamento do modelo referência

Nos experimentos descritos neste capítulo, realizamos comparações entre o modelo referência (arquitetura da figura 4.4, treinada de acordo com a configuração básica descrita na seção 4.3) e alguns modelos obtidos a partir de ligeiras modificações. Os modelos modificados foram todos gerados a partir de treinamento seguindo a mesma configuração básica, exceto a alteração de algum parâmetro indicado explicitamente em cada experimento.

Para comparar a diferença entre modelos, a visualização de atenção é uma ferramenta bastante útil. Portanto explicamos brevemente sobre ela.

Mapa de atenção: A camada de atenção gera um vetor de contexto para cada uma das posições do *decoder* a partir de uma soma ponderada dos elementos da sequência do *encoder* (ver seção 2.6). Os pesos associados a cada elemento do *encoder* são aprendidos pela rede. Cada um desses elementos do *encoder* está associado a um ponto no mapa de *features* extraído da imagem de entrada. Um ponto no mapa de *features*, por sua vez, está associado a uma região na imagem de entrada (campo receptivo). Desta forma, é possível relacionar cada posição do *encoder* a uma região específica na imagem de entrada.

A **visualização da camada de atenção** consiste em destacar as regiões da imagem de entrada sobre as quais o modelo está focando em um dado instante. Os pontos de foco de uma posição t do *decoder* são precisamente aquelas regiões correspondentes às posições i do *encoder* que melhor se alinham com a posição t do *decoder*, determinado pelo peso de alinhamento (ver equação 2.12).

Na visualização, as regiões da imagem com pesos maiores aparecem com tons mais claros evidenciando dessa forma onde o modelo foca naquele instante t da predição. Observe que para cada instante t temos um mapa de pesos. Desta forma, ao visualizarmos o foco de atenção a cada posição do *decoder*, podemos verificar qual é a região que mais está afetando a saída naquela posição e podemos ver como o foco de atenção "desliza" sobre a imagem ao longo das iterações.

O esquema de visualização é mostrado na figura 5.1.

Modelo referência: O treinamento do modelo referência (isto é, treinamento de acordo com a configuração básica descrita na seção 4.3) apresentou o comportamento mostrado no gráfico da figura 5.2. Note que a convergência ocorre após 8 épocas de treinamento, com as curvas de *loss* de treinamento e validação bastante próximos e uma acurácia, embora não muito alta, crescente. O mapa de atenção mostra que o modelo aprendeu a ordem de leitura.

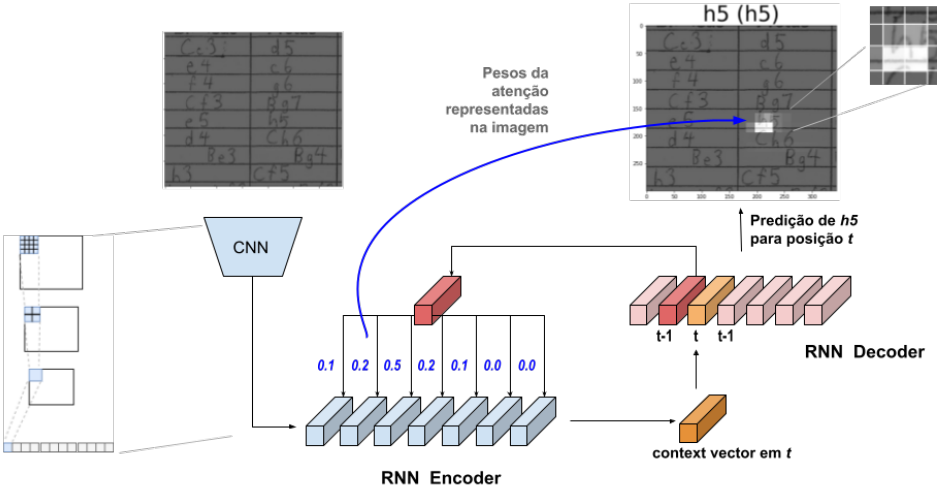


Figura 5.1: *Diagrama mostrando a visualização do mapa de atenção. Cada elemento na sequência do* encoder *corresponde a um quadrante na imagem original. Variando-se a cor de fundo de cada quadrante de acordo com o valor do peso, podemos visualizar onde o modelo foca mais para aquele instante da predição.*

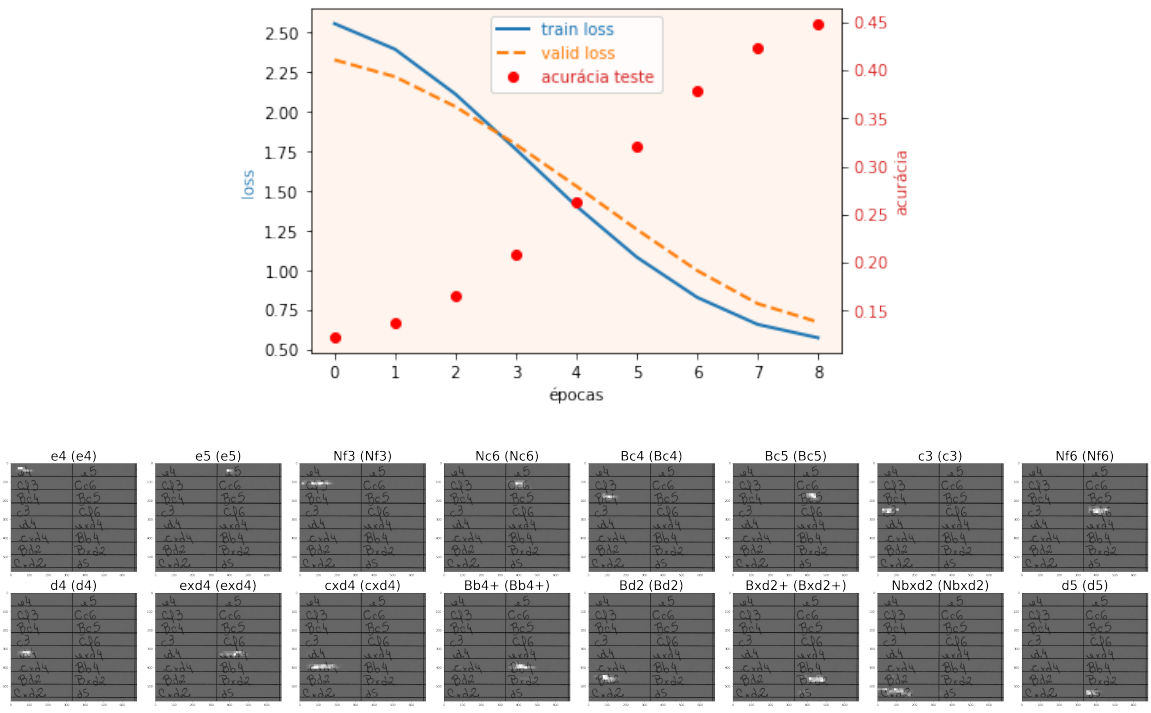


Figura 5.2: *Curva de* loss *e acurácia do modelo referência (parte superior) e sequência de mapas de atenção sobre um recorte de teste (parte inferior).*

5.2 Aprendizado da previsibilidade

A previsibilidade refere-se à habilidade do modelo predizer o próximo elemento, dado o histórico da sequência. Também é conhecido como *language model*.

Nos experimentos exploratórios inicias (apêndice C.2) constatamos que um modelo consegue aprender a previsibilidade somente com a parte do *decoder*. Testes de ablação mostraram que o treinamento converge mesmo sem o uso de qualquer informação visual da entrada. Por outro lado, constatamos também que em situações em que a sequência não apresenta nenhum tipo de padrão, ou seja, a ordem é aleatória, a previsibilidade não pode ser aprendida.

O aprendizado da previsibilidade relaciona-se fortemente, portanto, com a rede recorrente do *decoder* e à existência de padrões de sequência nos dados. Podemos indagar de que forma o mecanismo de *teacher forcing* afeta o aprendizado da previsibilidade.

5.2.1 Influência do *teacher forcing* no aprendizado da previsibilidade

Foi realizado um experimento para avaliar a influência do *teacher forcing*, desligando essa opção do modelo referência.

O resultado da comparação, entre o modelo referência (com *teacher forcing* ligado) e o novo modelo treinado com *teacher forcing* desligado, é apresentado na figura 5.3. Mostramos, à esquerda, a curva de *loss* do modelo referência (com *teacher forcing* ligado); ao centro, a curva de *loss* do modelo com *teacher forcing* desabilitado; à direita, a comparação das acurácias desses dois modelos sobre o conjunto de teste. Observa-se que enquanto o treinamento do modelo referência converge em cerca de 8 épocas, o desligamento do mecanismo de *teacher forcing* faz o treinamento do novo modelo demorar muito mais para convergir (mais de 25 épocas). Por outro lado, ambos os modelos atingem acurácia similar sobre o conjunto de teste. Em outras palavras, *teacher forcing* tem o efeito de acelerar a convergência sem necessariamente prejudicar a acurácia.

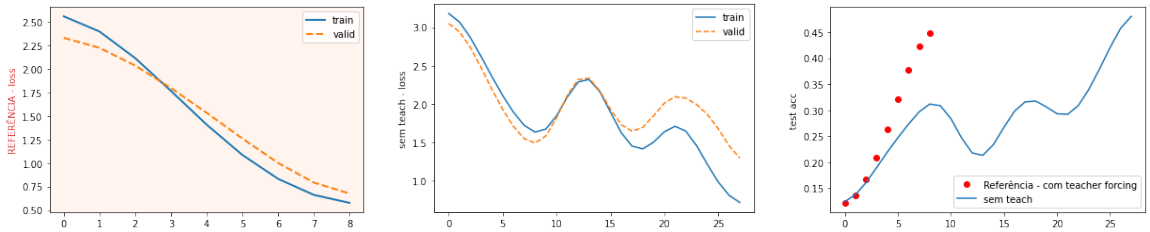


Figura 5.3: *À esquerda, a curva de* loss *do modelo referência. Ao centro, a curva de* loss *do modelo com* teacher forcing *desabilitado. À direita, acurácia de ambos os modelos sobre o conjunto de teste.*

5.2.2 Discussão

Com o *teacher forcing* ligado, o treinamento converge muito mais rápido. Este comportamento pode ser explicado pela forma como o mecanismo de *teacher forcing* opera, isto é, de durante o treinamento informar ao *decoder* o *groud-truth* da posição anterior. Desta

forma, o aprendizado da previsibilidade ao longo da sequência é aparentemente favorecido uma vez que não há acúmulo de erros provenientes de reconhecimento incorreto em posições anteriores. Sem o *teacher forcing*, eventuais erros podem se acumular ao longo da sequência, e nas posições posteriores o modelo não poderá contar com dados confiáveis para a predição do próximo elemento, dificultando assim a convergência.

Por outro lado, podemos também questionar se com o *teacher forcing* ligado a generalização poderia ser comprometida, já que durante o treinamento o modelo não é confrontado com uma predição errada na posição anterior, enquanto erros de reconhecimento podem ocorrer no processamento de uma nova planilha. Os resultados experimentais indicam que não. Tanto com o *teacher forcing* ligado ou deslidado, o modelo converge e atinge acurácias similares. Uma possível explicação quanto a esses resultados é o fato de que a predição feita pelo modelo não ser exclusivamente baseada nas predições das posições anteriores, mas também dos estados ocultos que supostamente codificam tanto a informação visual proveniente da imagem de entrada como a previsibilidade das sequências.

5.3 Aprendizado do alinhamento

O alinhamento refere-se ao correto posicionamento do foco de atenção na imagem, durante a leitura. Em outras palavras, quando o modelo está predizendo o t-ésimo elemento da sequência de saída, é desejável que a camada de atenção esteja atribuindo maior peso às posições no *encoder* que correspondem à região na qual o t-ésimo lance está localizado na imagem. Podemos dizer que um modelo capaz de realizar o alinhamento correto, aprendeu a ordem de leitura.

Naturalmente, o aprendizado do alinhamento está portanto relacionado com a camada da atenção. Informalmente, podemos dizer que essa é a camada que vai "selecionar" o trecho da imagem para o qual o *decoder* vai dar mais "atenção", por meio do escore de alinhamento (seção 2.6), para cada posição na sequência de predição.

Observamos, nos experimentos exploratórios, que entre os fatores que influenciam diretamente o aprendizado de alinhamento estão a quantidade de dados de treinamento e também a baixa previsibilidade nas sequências.

Para validarmos o aprendizado de alinhamento, a ferramenta utilizada é a visualização do mapa de atenção, que destaca os trechos na imagem sobre os quais o modelo deposita maior atenção.

5.3.1 Tamanho do conjunto de treinamento

Em alguns treinamentos realizados nos experimentos exploratórios com recortes reais, o modelo não foi capaz de aprender o alinhamento. Notou-se que aparentemente existe uma relação entre o tamanho da sequência e a quantidade de dados de treinamento.

Para verificar essa questão, foi realizado um experimento utilizando uma quantidade menor de dados. Enquanto o modelo referência foi treinado com 5000 recortes, neste experimento o treinamento foi realizado com um subconjunto de 2000 recortes. O conjunto

de treinamento é constituído unicamente de recortes com sequências reais válidas (recortes reais, recortes transcritos ou recortes-mosaico com sequências reais).

Mostramos, na figura 5.4, similarmente ao caso anterior, o resultado dos treinamentos do modelo referência e do modelo treinado com menos dados (2000 ao invés de 5000), e a comparação da evolução da acurácia sobre o conjunto de teste. Podemos ver pela curva de *loss* entre treinamento/validação que ambos os treinamentos convergem, porém o modelo treinado com 2000 exemplos apresenta forte *overfitting*. Ao analisarmos a acurácia de ambos no conjunto de teste, podemos ver que a acurácia do modelo treinado com menos dados é relativamente baixa, indicando que neste caso o aprendizado não foi satisfatório.

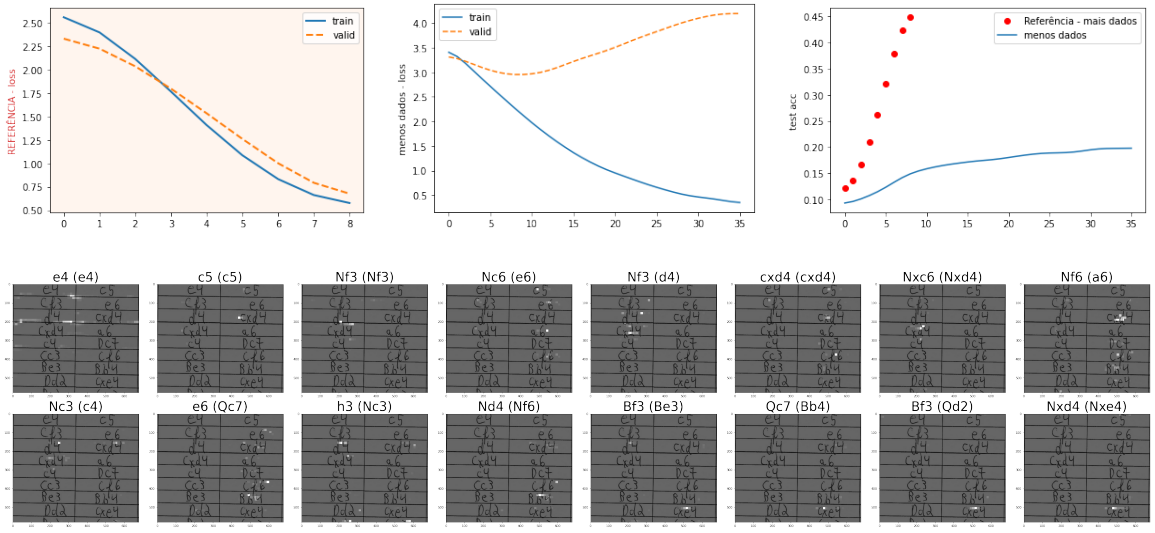


Figura 5.4: *Resultado do experimento com menos dados. O modelo converge com bastante overfitting, e a acurácia na base de testes é bastante baixa.*

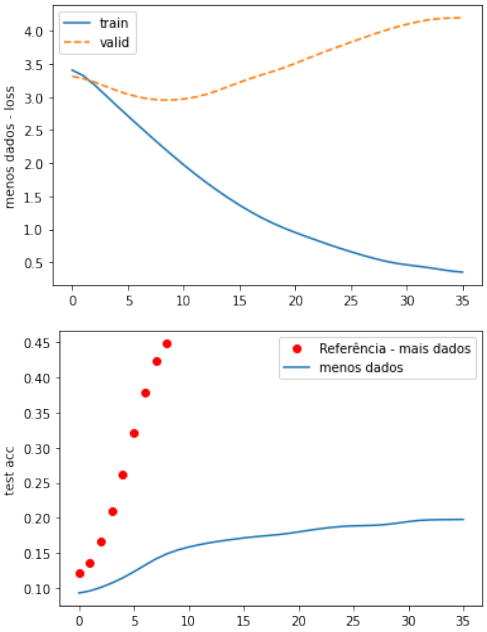
Na mesma figura, pelo mapa de atenção de uma das predições fica claro que o modelo não aprendeu o alinhamento. Ao mesmo tempo, como ocorre convergência do treinamento, pode-se especular que a rede está aprendendo a previsibilidade (ou "decorando" as sequências). Neste sentido, uma menor quantidade de sequências de treinamento (e portanto menor conflito entre prováveis lances para a próxima posição) deve estar favorecendo o fortalecimento do *decoder* quanto ao aprendizado da previsibilidade (modelo de linguagem) em detrimento do treinamento apropriado da camada de atenção.

Esta comparação indica que há uma quantidade mínima necessária de dados de treinamento para que o modelo aprenda o alinhamento. Empiricamente observamos que quanto maior o tamanho da sequência, maior é a quantidade mínima de dados de treinamento necessária. Por exemplo, para sequências de 2 linhas (ou seja, 4 lances), observamos que o alinhamento é aprendido mesmo com quantidade muito menor de dados de treinamento.

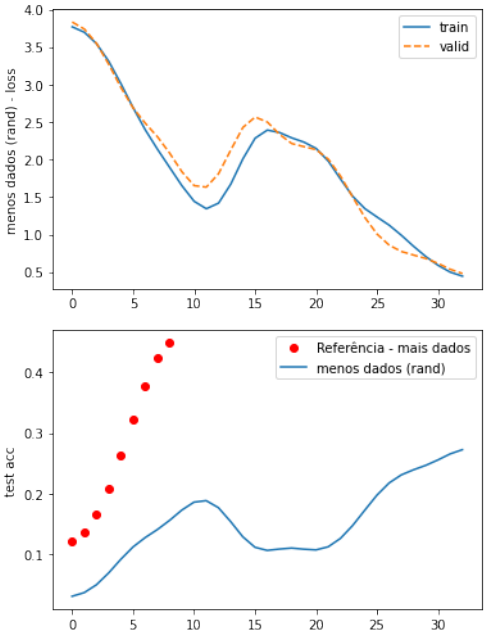
5.3.2 Treinamento com sequências de lances em ordem randômica

O resultado da seção anterior mostrou que há uma quantidade mínima de dados que são necessários para o modelo aprender o alinhamento. Portanto, a quantidade de dados é um fator crítico. Discutimos que uma possível razão desse fato é a menor diversidade de variações de próximo lance quando a quantidade de sequências é menor. Assim, essas sequências acabam se tornando altamente previsíveis, facilitando a rede a aprender a previsibilidade.

Uma pergunta interessante nesta situação é o que aconteceria se quebrássemos a previsibilidade? Podemos quebrar a previsibilidade utilizando sequências de lances em ordem randômica. Para investigar esta possibilidade foi realizado um experimento com uma quantidade reduzida de dados, de 2000 amostras, todos do tipo recorte-mosaico com sequências randômicas. O resultado do experimento é apresentado na figura 5.5.



Experimento anterior, com poucos dados e sequências reais de jogo.



Experimento também com poucos dados, porém com sequências em ordem randômica.

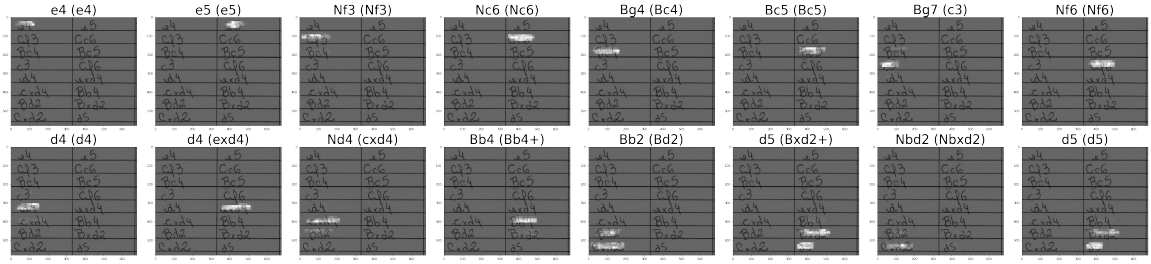


Figura 5.5: *Resultado do experimento com menor quantidade de dados de treinamento, usando dados artificiais com lances em ordem randômica. O treinamento apresenta maior dificuldade para convergência, mas o alinhamento é aprendido. A acurácia é levemente melhor do que no experimento anterior.*

Os gráficos da esquerda são os mesmo do experimento anterior, do modelo treinado com 2000 recortes com sequências reais de jogos, e reproduzido aqui apenas por conveniência. Os gráficos da direita são do modelo também treinado com 2000 dados, porém todas sequências randômicas. Podemos ver que neste segundo treinamento a convergência ocorre com menor *overfitting*, e o modelo apresenta uma acurácia levemente melhor do que a do experimento anterior (no qual o modelo não aprendeu o alinhamento). Pelo mapa de atenção, podemos ver que o alinhamento foi aprendido.

Logo, podemos entender que o uso de sequências randômicas impossibilita o modelo de convergir baseado majoritariamente no aprendizado de previsibilidade pois ela inexiste e força o modelo a aprender o alinhamento, de forma que ele possa utilizar a informação visual para reconhecer corretamente o lance.

5.3.3 A influência do tamanho da imagem

Vimos no experimento descrito acima que a quantidade de imagens de treinamento é crítico para o aprendizado de alinhamento. Quando são utilizadas imagens de sequências reais de jogos, enquanto com 5000 imagens de treinamento há aprendizado de alinhamento, como 2000 não há.

Um outro fator que consideramos interessante investigar é o tamanho da imagem de entrada. Apesar de ser intuitivo que para a leitura da escrita seja razoável que a imagem tenha uma qualidade mínima, não é óbvio que para o alinhamento seja necessária a mesma qualidade. Este questionamento pode ser comparado com o próprio funcionamento do sistema visual humano. Mesmo em condições de qualidade visual baixa, somos capazes de direcionar a atenção para uma região de interesse, apesar de não ser possível reconhecer o que se encontra naquela região.

Motivado pela questão acima, realizamos um treinamento utilizando as 5000 imagens, porém alterando o tamanho das imagens de 800 × 862, utilizado pelo modelo referência, para 400 × 431, a metade do tamanho original. Além disso, desligamos o *teacher forcing* para não favorecer a convergência baseada apenas na previsibilidade das sequências; com isso, de certa forma espera-se que a rede seja forçada a aprender o alinhamento. O restante da configuração básica foi mantido.

O resultado do experimento é mostrado na figura 5.6. À esquerda o treinamento do modelo referência, ao centro este novo treinamento, e à direita a evolução da acurácia sobre o conjunto de teste. A convergência é demorada, ocorrendo em torno de 170 épocas (frente as 8 épocas do modelo referência). Apesar disso, conforme podemos ver nos mapas de atenção, percebe-se que houve aprendizado do alinhamento. Uma consequência disso é que a acurácia tende a ser melhor que os modelos onde não houve aprendizado do alinhamento (ver, por exemplo, a curva de acurácia da figura 5.4).

Este resultado indica que o alinhamento pode ser aprendido mesmo com imagens de tamanho reduzido. Note que a acurácia em relação ao modelo referência é bem inferior, fornecendo indícios de que a redução do tamanho da imagem deve estar reduzindo a capacidade de reconhecimento da escrita. Esta questão é retomada mais adiante, quando abordamos a subtarefa de reconhecimento.

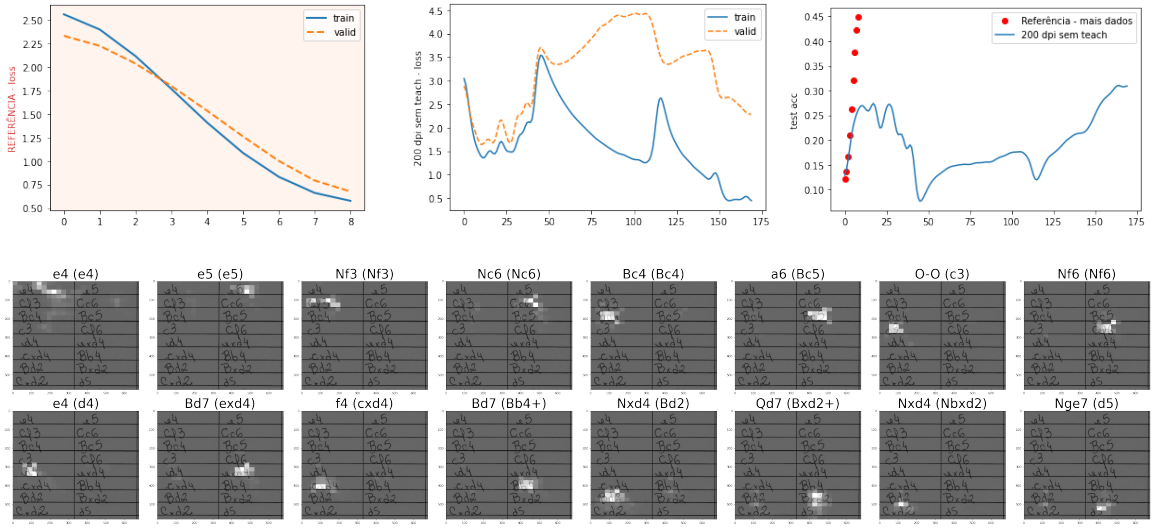


Figura 5.6: *Modelo referência × Treinamento com imagens de tamanho menor e* teacher forcing *desligado. Há demora na convergência, mas o alinhamento é aprendido.*

5.3.4 Alinhamento de sequências longas

Durante os experimentos exploratórios, em um dos testes realizados foi possível fazer um modelo aprender o alinhamento em sequências mais longas.

Nesse caso, o modelo utilizado é semelhante ao modelo referência, com algumas diferenças. Foram usadas imagens de página cheia com até 100 lances. O tamanho da imagem é relativamente menor, de 900×678 (o modelo referência usa imagens que correspondem a aproximadamente 100 pixels por linha, o que no caso de página cheia resultaria em uma imagem com altura de cerca de 2500 pixels).

O tamanho do conjunto de treinamento foi de 4000 amostras, e o modelo foi treinado incrementalmente no tamanho da sequência. Os dados utilizados foram recortes mosaico com sequências de lances aleatórias. A figura 5.7 mostra o mapa de atenção ao longo de uma sequência, para os 50 primeiros lances. Cabe ressaltar que neste caso a convergência foi bastante lenta e a acurácia bem baixa, apesar do alinhamento correto.

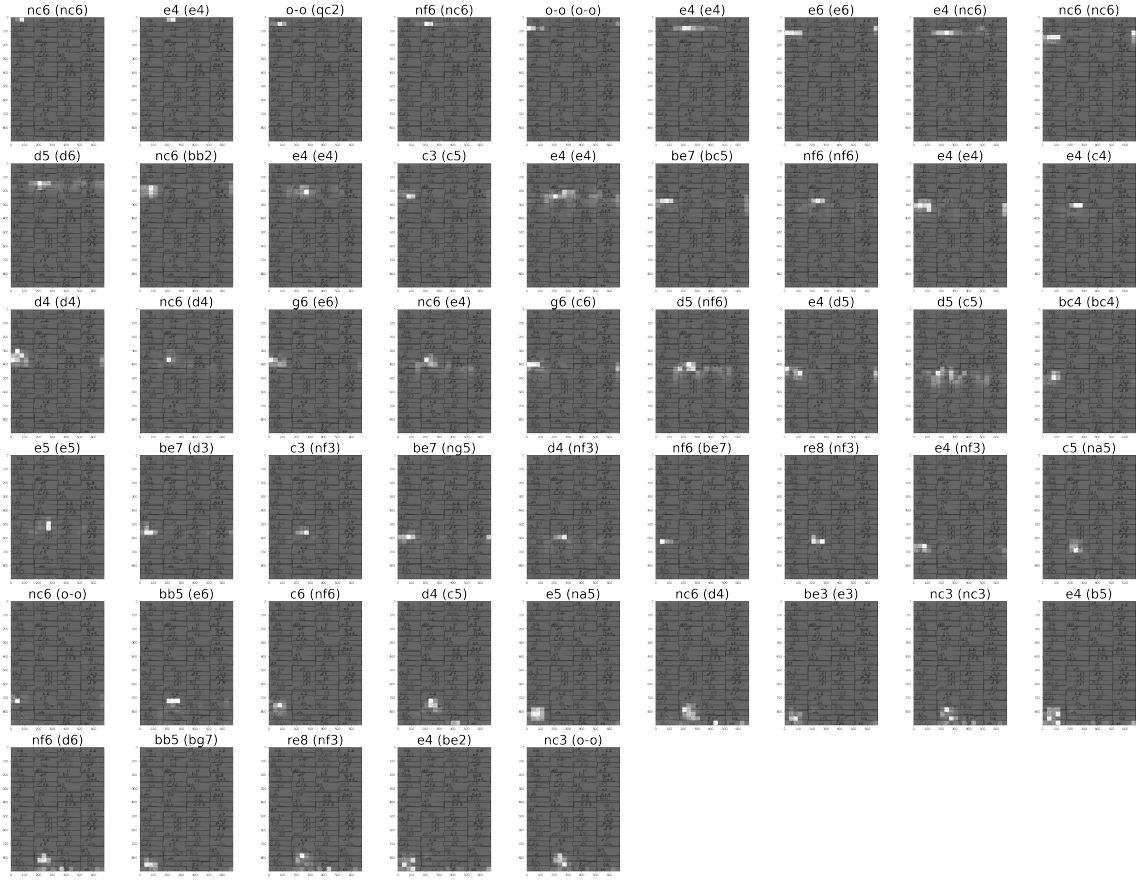
5.3.5 Discussão

Vimos na seção 5.3.1 que o modelo referência treinado com 5000 exemplos de treinamento converge e há indícios de que aprende o alinhamento juntamente com a previsibilidade. Por outro lado, se o treinamento é realizado com uma quantidade reduzida de dados (por exemplo, 2000), apesar de haver convergência, o alinhamento não é aprendido. A convergência parece estar sendo alcançada apenas baseada na previsibilidade das sequências.

A questão da quantidade de dados pode ser em parte amenizada usando-se sequências com lances em ordem aleatória. Especificamente, mostramos que 2000 dados desse tipo são suficientes para o modelo aprender o alinhamento, enquanto a mesma quantidade de sequências reais não são suficientes. Essa diferença reforça a percepção de que ao quebrar a previsibilidade acabamos forçando a rede a aprender o alinhamento. O experimento mostrado na última seção indica que o aprendizado de alinhamento acontece mesmo para sequências mais longas.

A figura 5.8 mostra as acurácias de teste dos distintos modelos investigados no estudo sobre o aprendizado do alinhamento. O modelo referência apresenta a maior acurácia, e convergência rápida. Num cenário de menos dados, a utilização de sequências randômicas no treinamento permite o aprendizado de alinhamento e por conseguinte uma melhor acurácia do que o treinamento que não aprende o alinhamento. A redução no tamanho da imagem parece não ser impeditivo para o aprendizado de alinhamento, embora a convergência ocorra de forma bastante lenta.

Em termos de dificuldade de aprendizado, o alinhamento parece ser uma tarefa mais fácil do que o reconhecimento da escrita, pois o modelo é capaz de aprender o posicionamento correto do foco de atenção, mesmo sem reconhecer corretamente os lances. Podemos dizer também que o modelo sempre irá priorizar a convergência por meio do aprendizado da previsibilidade, a menos que a quantidade de dados seja suficientemente grande a ponto de dificultar a previsibilidade. Com dados em quantidade suficientemente grande, é possível o aprendizado conjunto da previsibilidade e do alinhamento.



Seq. esperada:
nc6 e4 qc2 nc6 o-o e4 e6 nc6 nc6 d6
bb2 e4 c5 e4 bc5 nf6 e4 c4 d4 d4
e6 e4 c6 nf6 d5 c5 bc4 e5 d3 nf3
ng5 nf3 be7 nf3 nf3 na5 o-o e6 nf6 c5
na5 d4 e3 nc3 b5 d6 bg7 nf3 be2 o-o

Seq obtida:
nc6 e4 o-o nf6 o-o e4 e6 e4 nc6 d5
nc6 e4 c3 e4 be7 nf6 e4 e4 d4 nc6
g6 nc6 g6 d5 e4 d5 bc4 e5 be7 c3
be7 d4 nf6 re8 e4 c5 nc6 bb5 c6 d4
e5 nc6 be3 nc3 e4 nf6 bb5 re8 e4 nc3

Figura 5.7: *Resultado da predição dos 50 lances iniciais de uma imagem de página cheia do conjunto de validação. O tamanho da imagem de entrada é de (900, 678) o que dá no final da camada de convolução um mapa de dimensão (28,21). O tamanho do conjunto de treinamento foi de 4000 amostras, e foi treinado incrementalmente no tamanho da sequência, levando bastante tempo para convergir. O modelo aprende o alinhamento mas a acurácia é baixa.* Loss *de treinamento: 0.08,* loss *de validação: 4, CER nesta imagem de validação: 65%.*

Assumindo a previsibilidade como sendo uma tarefa fácil, podemos dizer que relativamente o alinhamento é uma tarefa com nível de dificuldade média.

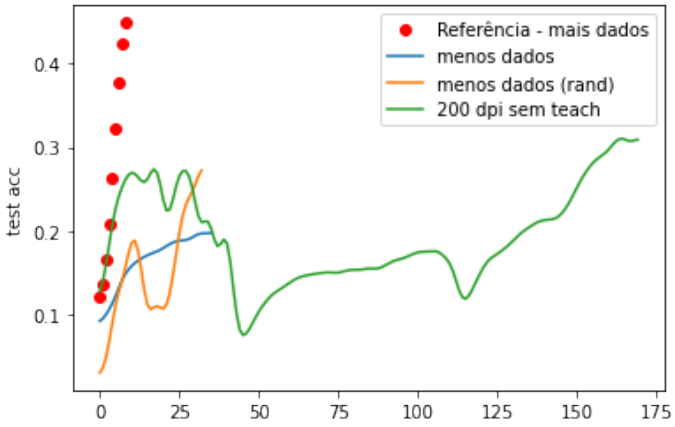


Figura 5.8: *Evolução da acurácia sobre o conjunto de teste ao logo do treinamento dos experimentos descritos nesta seção.*

5.4 Aprendizado do reconhecimento de lances

O reconhecimento, a terceira subtarefa investigada, refere-se à habilidade de predição correta de cada lance. Mesmo o modelo tendo aprendido o alinhamento, e usando-se do conhecimento da previsibilidade, é imprescindível que ele faça a correta leitura de cada lance (informação visual) para apresentar um desempenho satisfatório.

As imagens dos lances podem ter diferenças sutis, que em alguns casos mesmo para o ser humano é difícil distinguir. Por exemplo, as letras “e” e “c” são muito parecidas. No caso das planilhas em português, as letras “C” (cavalo) e “c” (peão) também são bem difíceis de distinguir. As letras “a”, “g” e “d” também são facilmente confundíveis. Estes são alguns exemplos de diferenças sutis que o modelo deve aprender a identificar.

Por meio dos experimentos exploratórios, notamos que um fator crítico no aprendizado do reconhecimento relaciona-se com a dimensão da imagem. Aqui investigamos, portanto, o efeito que imagens de qualidade inferior tem sobre o reconhecimento.

5.4.1 Influência do tamanho da imagem de entrada

Intuitivamente, podemos dizer que a qualidade da informação visual é importante no reconhecimento. Isto é, deve existir uma resolução mínima a partir da qual é possível reconhecer a escrita. Resoluções menores do que esse mínimo poderiam simplesmente impossibilitar a distinção entre uma letra “e” e “c”, por exemplo.

No modelo referência, o tamanho da imagem de entrada utilizado para recortes de 8 linhas (800 × 862), juntamente com o mapa de atenção de dimensão 50 × 53, corresponde a cerca de 6 pontos de atenção por linha de planilha.

Para verificar a importância do tamanho da imagem de entrada, realizamos um treinamento em condições equivalentes ao do modelo referência, exceto pelo tamanho da imagem de entrada. Neste novo treinamento, reduzimos o tamanho da imagem pela metade, o que resultou em cerca de 3 pontos de atenção por linha de planilha.

O resultado é apresentado na figura 5.9. Seguindo o mesmo esquema das figuras anteriores, mostramos o gráfico de treinamento do modelo referência, do modelo com a imagem de entrada de tamanho reduzido e a evolução da acurácia sobre o conjunto de teste de ambos. Podemos ver que, enquanto no modelo referência há convergência sem *overfitting*, neste novo modelo a convergência ocorre, porém de forma mais lenta e com significativo *overfitting*. A acurácia na base de testes também é bastante baixa quando comparado com a do modelo referência.

Quanto ao alinhamento, convém comparar o resultado aqui apresentado com o da figura 5.6. Lá o treinamento foi realizado também com as imagens reduzidas à metade do tamanho original, mas adicionalmente com o *teacher forcing* desligado. Essa comparação permite observar que com o *teacher forcing* ligado, que é o caso deste experimento, a convergência ocorre muito mais rapidamente, porém o alinhamento não é tão claro. Como consequência, a acurácia é menor que a alcançada lá.

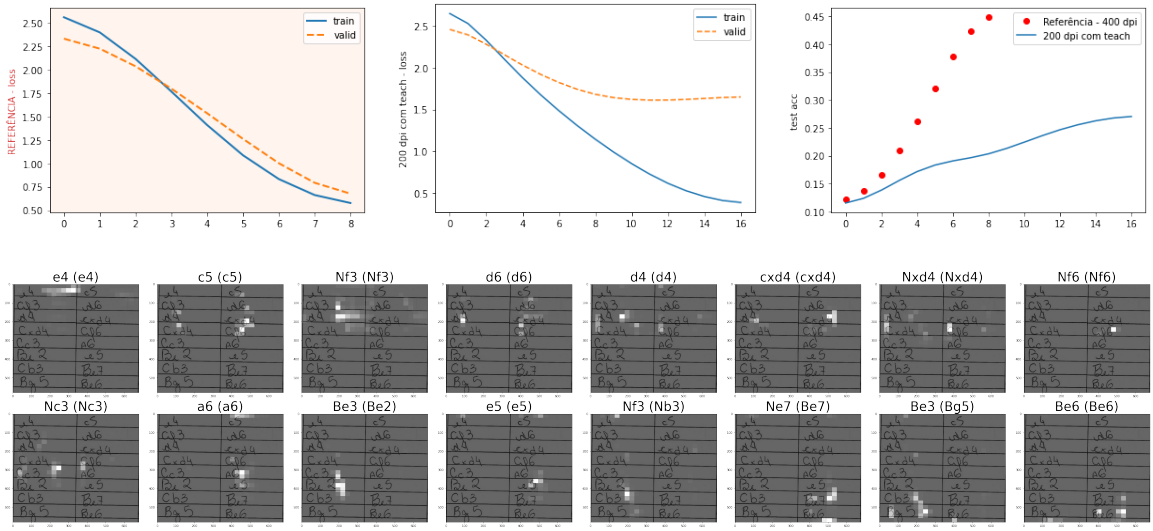


Figura 5.9: *Treinamento com imagens com metade da resolução. O modelo aprende o alinhamento e converge relativamente rápido, mas apresenta um grau de* overfitting *entre treinamento e validação bastante grande. O índice final de acurácia em testes também fica bem abaixo do modelo referência.*

Uma consequência do tamanho menor da imagem de entrada é uma menor quantidade de pontos sobre os quais a camada da atenção pode atuar. Na figura 5.10 mostramos mapas de atenção que contrastam a densidade de pontos de atenção em função do tamanho da imagem de entrada. Para cada linha da planilha no mapa da esquerda há cerca de 6 pontos de atenção, enquanto no mapa da direta há apenas cerca de 3 pontos atenção.

5.4.2 Discussão

A resolução da imagem de entrada determina o número de pontos no mapa de *features* e esta por sua vez corresponde ao tamanho da sequência de entrada do *encoder*. A camada de atenção atua sobre os elementos do *encoder* e, desta forma, indiretamente sobre uma região (ponto de atenção) na imagem de entrada.

Empiricamente, para a arquitetura considerada, observamos que 6 pontos de atenção por linha de planilha é uma quantidade ideal (adotada em nossa configuração básica).

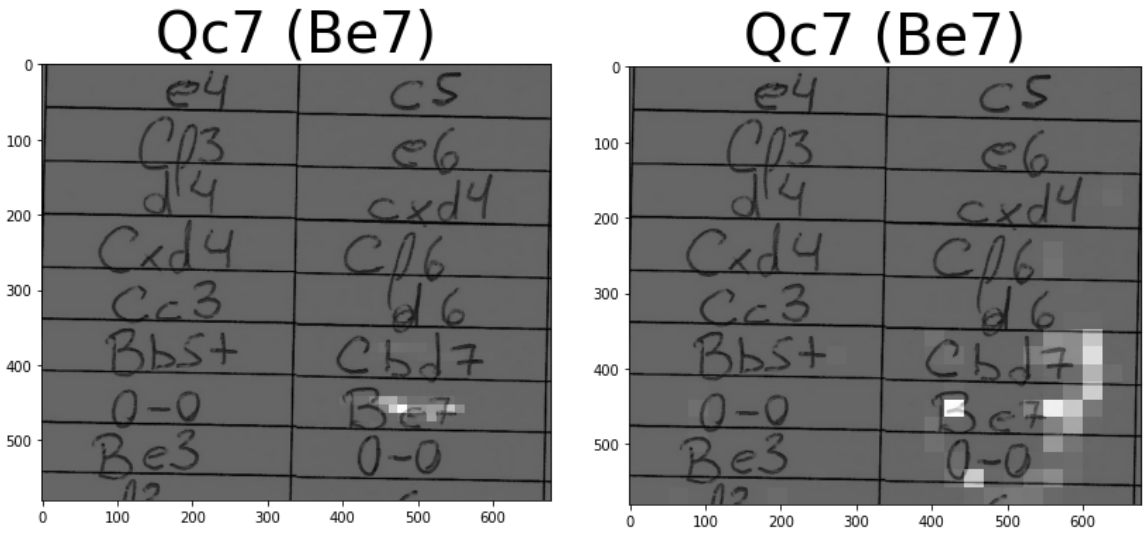


Figura 5.10: *Detalhe de predição de uma posição. À esquerda, o mapa de atenção referente ao modelo referência que usa uma imagem 800×862, e à direita o do modelo que usa uma imagem 400×431. Podemos ver que o modelo referência tem o dobro de pontos de atenção, o que permite que a camada da atenção possa atuar de forma mais precisa.*

Não observamos ganhos ao aumentar essa quantidade (em testes comparativos feitos). Os experimentos indicam que 3 pontos de atenção ainda são suficientes para o aprendizado de alinhamento (embora com maior dificuldade), mas afetam negativamente o reconhecimento.

Uma vez que em uma imagem com 8 linhas de planilha, de dimensão 400×431, os textos são facilmente legíveis a olho nu, podemos concluir que não necessariamente a resolução da imagem de entrada, mas a densidade de pontos de atenção é um fator importante para o bom desempenho do modelo, mais especificamente em relação a sua habilidade de reconhecimento.

Uma consequência desta constatação é que modelos que utilizam arquiteturas similares esbarrarão em limitações computacionais (memória) para o processamento de sequências maiores (recortes com maior número de linhas de planilha). Em parte esta limitação poderia ser mitigada com o uso de um módulo convolucional customizado para as imagens de planilha, ao invés da utilização de um módulo pré-treinado *off the shelf*. Por exemplo, um menor número de camadas convolucionais ou a não utilização de *maxpooling* resultariam em maior número de pontos de atenção. Restaria verificar se as *features* extraídas seriam suficientes para codificar as informações relevantes para o reconhecimento. Neste trabalho não tratamos esta questão.

Capítulo 6

Resultados

Neste capítulo descrevemos os dois principais resultados deste trabalho.

O primeiro resultado refere-se a uma compreensão conceitual sobre diversos aspectos envolvidos no problema de leitura de planilhas de xadrez, e em particular sobre as subtarefas implícitas e como as mesmas se relacionam com o modelo baseado em uma estrutura consistindo de camadas convolucionais e uma rede recorrente *encoder-decoder* com atenção. A descrição apresentada aqui visa fornecer uma perspectiva integrada dos estudos sobre o aprendizado de subtarefas descritos no capítulo anterior.

O segundo resultado é o modelo desenvolvido propriamente, que foi chamado de modelo referência, e aqui apresentamos métricas e exemplos de desempenhos alcançados por ele e uma pequena variação dele.

6.1 Aprendizado integrado das subtarefas

Os treinamentos exploratórios e posterior estudos controlados mostraram que o aprendizado ponta a ponta no problema de leitura de planilhas de xadrez é possível. Também constatamos que o problema compõe-se de subtarefas que devem ser aprendidos de forma conjunta pelo modelo: (1) a predição do lance baseada na previsibilidade, (2) o alinhamento da atenção visual, e (3) o reconhecimento do lance propriamente, baseado na informação visual. Constatamos também que existem fatores específicos que influenciam o aprendizado de cada uma delas. Em particular, alguns desses fatores facilitam e outros fatores são condições necessárias para o aprendizado de uma subtarefa.

Nesta seção sumarizamos os entendimentos alcançados, colocando-os sob uma perspectiva integrada que considera o aprendizado conjunto (e não individual) dessas subtarefas, condição fundamental para que seja possível o aprendizado ponta a ponta. No diagrama da figura 6.1, mostramos as subtarefas identificadas e os fatores que influenciam o aprendizado das mesmas, de acordo a investigação experimental realizada neste trabalho. Apontamos também a parte da arquitetura que está mais relacionada com cada uma das tarefas.

As discussões apresentadas a seguir podem ter sobreposição com discussões já apresentadas no capítulo anterior, porém acreditamos que reuni-las todas aqui de forma resumida

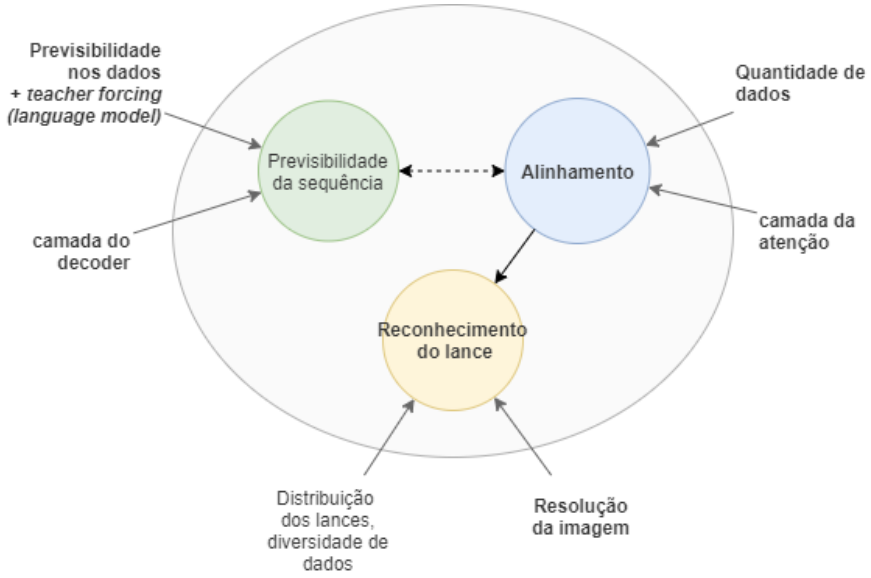


Figura 6.1: *Diagrama ilustrando as subtarefas que compõe o aprendizado ponta a ponta, e os fatores que influenciam o aprendizado de cada subtarefa. As subtarefas são o alinhamento, a previsibilidade da sequência e o reconhecimento do lance, sendo a previsibilidade a mais fácil, seguida pelo alinhamento e o reconhecimento, que é a mais difícil. Dentre os fatores, a quantidade de dados e a resolução da imagem de entrada são os que mais influenciam no aprendizado conjunto dessas três subtarefas.*

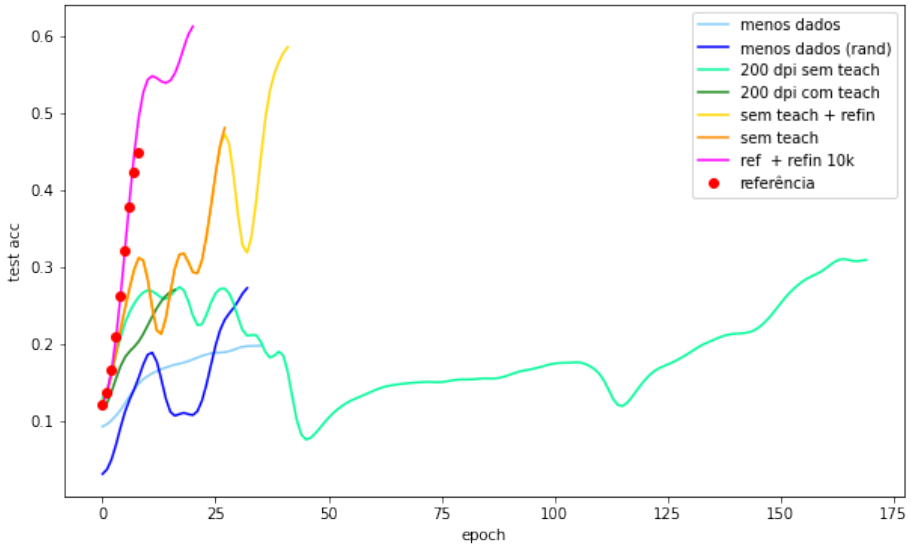
ajuda a ampliar a compreensão.

Na figura 6.2 apresentamos um resumo dos experimentos realizados, por meio de um gráfico de acurácia ao longo das épocas de treinamento no conjunto de teste e uma tabela com os principais fatores estudados no capítulo anterior. Na tabela, cada linha representa um experimento, para os quais são detalhados a configuração dos "fatores", os efeitos positivos (S) ou negativos (N) sobre o tempo de convergência, alinhamento, *overfitting* e acurácia, e na última coluna a acurácia final na base de testes. Além dos experimentos descritos no capítulo anterior, são apresentados também 2 refinamentos (6 e 8) realizados com "targets" menores (critério de convergência mais rígido) para melhoria da acurácia final. O refinamento 8 está detalhado mais adiante.

Quando os treinamentos foram realizados com relativamente poucos dados (por exemplo, figura 5.4 na seção 5.3.1) percebemos que a convergência ocorria em geral baseada apenas na previsibilidade, sem o aprendizado do alinhamento. Ou, quando o alinhamento era satisfatório, em uma situação de resolução menor das imagens de entrada (por exemplo, figura 5.6 na seção 5.3.3), a acurácia de reconhecimento era baixa.

Com base nessas observações e outras discutidas no capítulo anterior, acreditamos que as subtarefas podem ser ordenadas de acordo com o grau de dificuldade, da mais fácil para a mais difícil, da seguinte forma: (1) previsibilidade, (2) alinhamento, e (3) reconhecimento.

Constatamos que uma condição fundamental para que ocorra um aprendizado conjunto satisfatório dessas três subtarefas é uma quantidade suficiente de dados (reais ou artificias, com sequências reais de jogos), e uma resolução mínima da imagem de entrada. Concreta-



	Experimento	Qtd dados	Tam imagem	Tipo de dados	Teacher forcing	Target trein.	Tempo Converg	Alinha-mento	overfit	acc testes	acc testes
1	menos dados	2000	400dpi	seq reais	S	0.25/0.9	S	N			20
2	menos dados (rand)	2000	400dpi	rand	N	0.25/0.9	S	S	S		31.5
3	200 dpi sem teach	5000	200dpi	seq reais	N	0.25/0.9	N	S	S		30.3
4	200 dpi com teach	5000	200dpi	seq reais	S	0.25/0.9	S	S-	S		27.9
5	sem teach	5000	400dpi	seq reais	N	0.25/0.9	S	S	N	S	57.70%
6	sem teach + refin	5000	400dpi	seq reais	N	0.1/0.95	S	S	N	S	62.8
7	**referencia**	**5000**	**400dpi**	**seq reais**	**S**	**0.25/0.9**	**S**	**S**	**N**	**S**	51.53
8	ref + refin 10k	10000	400dpi	seq reais	S	0.07/0.97	S	S	N	S	65.78

Figura 6.2: *Na tabela, o resumo dos experimentos conforme descrito no capítulo anterior e dois refinamentos adicionais (linhas 6 e 8), e no gráfico as respectivas curvas de acurácia sobre a base de testes ao longo das épocas de treinamento. Ver texto.*

mente, para o recorte considerado de 8 linhas (16 lances), constatamos que 5000 imagens e resolução de 100 pixels por linha de planilha são suficientes (modelo referência).

Previsibilidade: Esta é uma tarefa que pode ser aprendida facilmente, principalmente com a opção *teacher forcing* ligada. Apenas o *decoder* sozinho é capaz de aprender a previsibilidade, conforme mostraram os experimentos sem uso da imagem de entrada. De forma geral, quanto maior a previsibilidade nos dados de treinamento, mais fácil é o treinamento desta habilidade. De fato, a previsibilidade é reduzida quanto maior é a quantidade de sequências de treinamento, ou quando sequências randômicas são utilizadas, ou possivelmente quanto mais longas são as sequências, dificultando o aprendizado da previsibilidade. Neste sentido, a quantidade de dados parece atuar como um regularizador que impede o modelo de se especializar apenas na previsibilidade (modelo de linguagem), forçando o aprendizado do alinhamento. Sequências randômicas (figura 5.5 na seção 5.3.2) seriam um caso extremo de regularização, no qual remove-se totalmente a previsibilidade dos dados. A visualização do mapa de atenção foi fundamental para essas análises.

Alinhamento: A camada de atenção é crucial para o aprendizado de alinhamento conforme mostraram os experimentos iniciais com e sem essa camada. O aprendizado do

alinhamento parece ser favorecido quando, além de menor previsibilidade (conforme descrito acima no item sobre previsibilidade), as imagens de entrada possuem melhor qualidade. No entanto, a qualidade da imagem tem efeito muito maior no reconhecimento propriamente dito. No modelo referência, as imagens utilizadas têm cerca de 100 pixels por linha de planilha, quantidade que mostrou-se suficiente para o aprendizado conjunto das três subtarefas. Utilizando-se imagens com metade desse tamanho o reconhecimento foi prejudicado, mas foi ainda possível fazer o modelo aprender o alinhamento (figura 5.6 na seção 5.3.3).

Reconhecimento do lance: Uma condição imprescindível ou requisito para o reconhecimento do lance via informação visual é o alinhamento correto. Porém, apenas o alinhamento correto não é suficiente. A qualidade da imagem (expressa em termos de pixels por linha de planilha) mostrou-se um fator relevante. Argumentamos, no entanto, que mais do que a resolução da imagem de entrada propriamente, um aspecto determinante parece ser a densidade de pontos de atenção sobre a imagem de entrada. Esse número parece ser crucial para uma codificação da informação visual que seja suficiente para o reconhecimento do lance (seção 5.4). Neste sentido, a parte da rede que está mais relacionada com esta sub-tarefa é o módulo convolucional. Nos experimentos, porém, esse módulo foi mantido fixo. Uma possível customização desse módulo é deixada para investigações futuras.

6.2 Desempenho dos modelos

O modelo referência foi gerado a partir da configuração básica definida na seção 4.3, que por sua vez resultou dos experimentos exploratórios. A configuração básica define uma arquitetura, as dimensões das imagens e parâmetros e procedimentos de treinamento que possibilitaram um treinamento ponta a ponta de forma robusta.

6.2.1 Avaliação quantitativa

No conjunto de teste, o modelo referência alcançou 52% de acurácia e índice CER de 36,9%, calculados sobre sequências de 16 lances. As métricas não são tão boas, mas servem como uma referência inicial.

Um experimento adicional foi realizado alterando-se o critério de convergência da configuração básica. No modelo referência, assumiu-se que a convergência ocorre quando o treinamento atinge uma *loss* de 0,25 ou acurácia de treinamento de 0,9. Neste novo treinamento, a partir de mesma configuração básica do modelo referência, alteramos o alvo de *loss* no critério de convergência de 0,25 para 0,07 e o tamanho do conjunto de treinamento de 5000 para 10000 imagens. Este incremento do dataset foi feito exclusivamente com imagens do tipo recorte mosaico com sequências reais, conforme descrito na seção 4.1.2. Chamamos o modelo resultante de **modelo final**.

A figura 6.3 mostra a acurácia (curva vermelha) e o CER (curva azul) do modelo final no conjunto de teste, em função do tamanho da sequência. Aqui, o modelo considerado e a predição são únicos. A avaliação é feita, porém, considerando-se a sequência até cada um dos tamanhos indicados. Assim, para o tamanho da sequência 1 estamos calculando o

índice somente com respeito ao primeiro elemento da sequência, para 2, até o segundo elemento da sequência, e assim por diante.

A acurácia é de 65,78% (posição 16 no eixo horizontal e eixo vertical à esquerda) e o CER é de 24,88% (posição 16 no eixo horizontal e eixo vertical à direita), ambos calculados sobre a sequência completa.

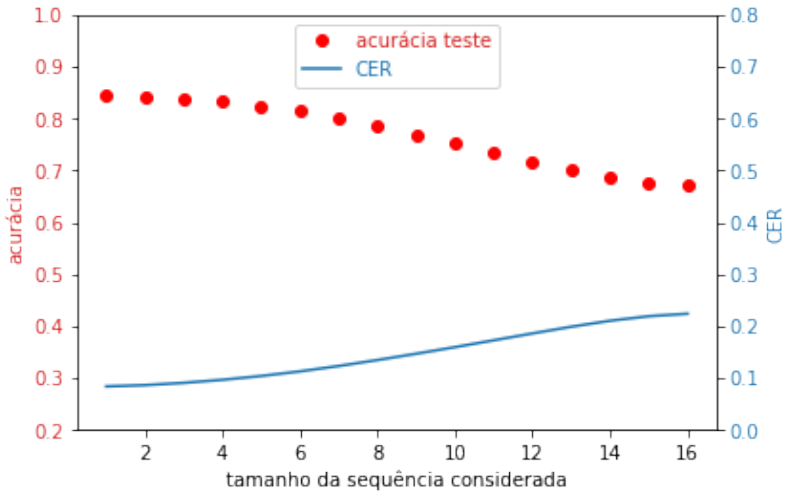


Figura 6.3: *Desempenho do modelo final sobre recortes de 8 linhas (16 lances), do conjunto de teste. Acurácia de 85,08% e CER de 7,45% na primeira posição e acurácia de 65,78% e CER de 24,88% na última posição 16 (erro considerado sobre toda a sequência).*

Pode-se observar que a acurácia decresce quanto mais lances são considerados no cômputo do erro. Esse decréscimo é esperado, devido à propagação de erro no reconhecimento sequencial. Quando o cálculo é restrito ao reconhecimento apenas do primeiro lance, temos acurácia de 85,08% e CER de 7,45%. Apenas para uma comparação relativa, o CER alcançado pelo modelo de Kang, Toledo *et al.*, 2018 sobre o *dataset* IAM foi de 6,88% (quanto menor, melhor é o CER).

Quando comparamos o desempenho do modelo referência com o do modelo final, temos uma diferença na acurácia que varia de 52% para 65,78% e no CER de 36,8% para 24,88%, correspondendo a uma melhora significativa. Essa melhora foi devida à alteração de um parâmetro de treinamento e ao uso de uma quantidade maior de dados de treinamento (note porém que esse aumento foi por meio de recortes mosaico).

O cálculo dessas duas métricas permite uma análise mais detalhada dos erros. A acurácia, que é calculada em termos de "palavras", simplesmente indica se o modelo fez a predição correta do lance, sem levar em consideração as similaridades entre as escritas do lance predito e do lance correto (*ground-truth*). Isto é, uma predição "e7" ao invés de "e4", ou "Bd2" ao invés de "Dd2", ou ainda "Be2" ao invés de "Bg2", são contabilizadas simplesmente como casos de erros da mesma forma que um caso em que "Nf3" é predito como "e4". O CER, por ser calculado em termos de caracteres, consegue diferenciar melhor esses dois casos. Note, no entanto, que o modelo foi treinado para reconhecer lances (palavras) e não os caracteres individuais.

Na figura 6.4 mostramos os resultados correspondentes as 5 melhores predições, e na

figura 6.5 as 5 piores predições, dentre as imagens no conjunto de teste. Nos melhores casos, a quantidade de erros não passa de um entre os 16 lances analisados. Avaliando o mapa de atenção, podemos ver que nos melhores casos os pontos de atenção estão melhor definidos e também localizados nas posições corretas. Já nos piores casos, os pontos de atenção não estão sempre nas posições corretas.

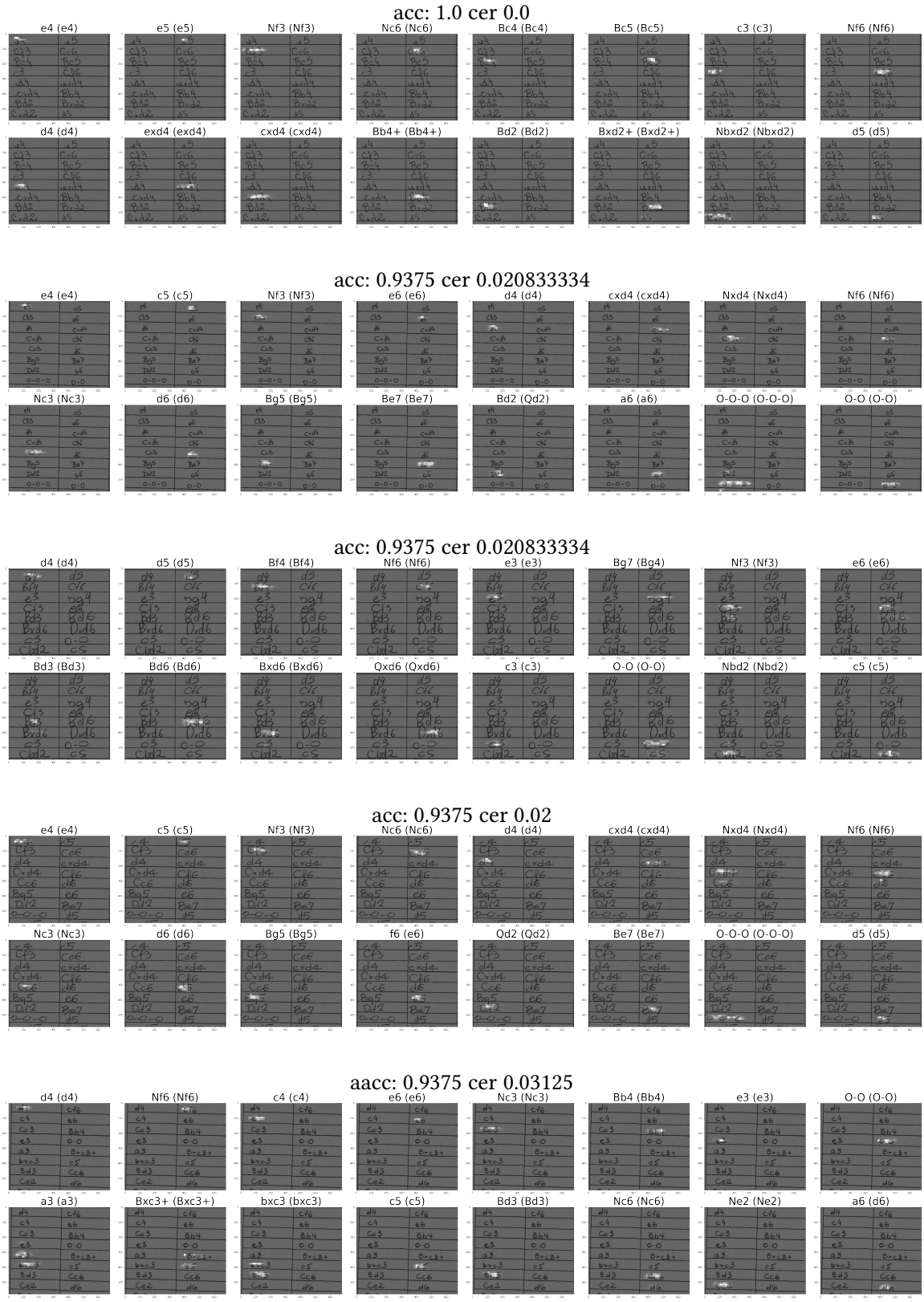


Figura 6.4: *As 5 planilhas de teste com melhores índices de predição.*

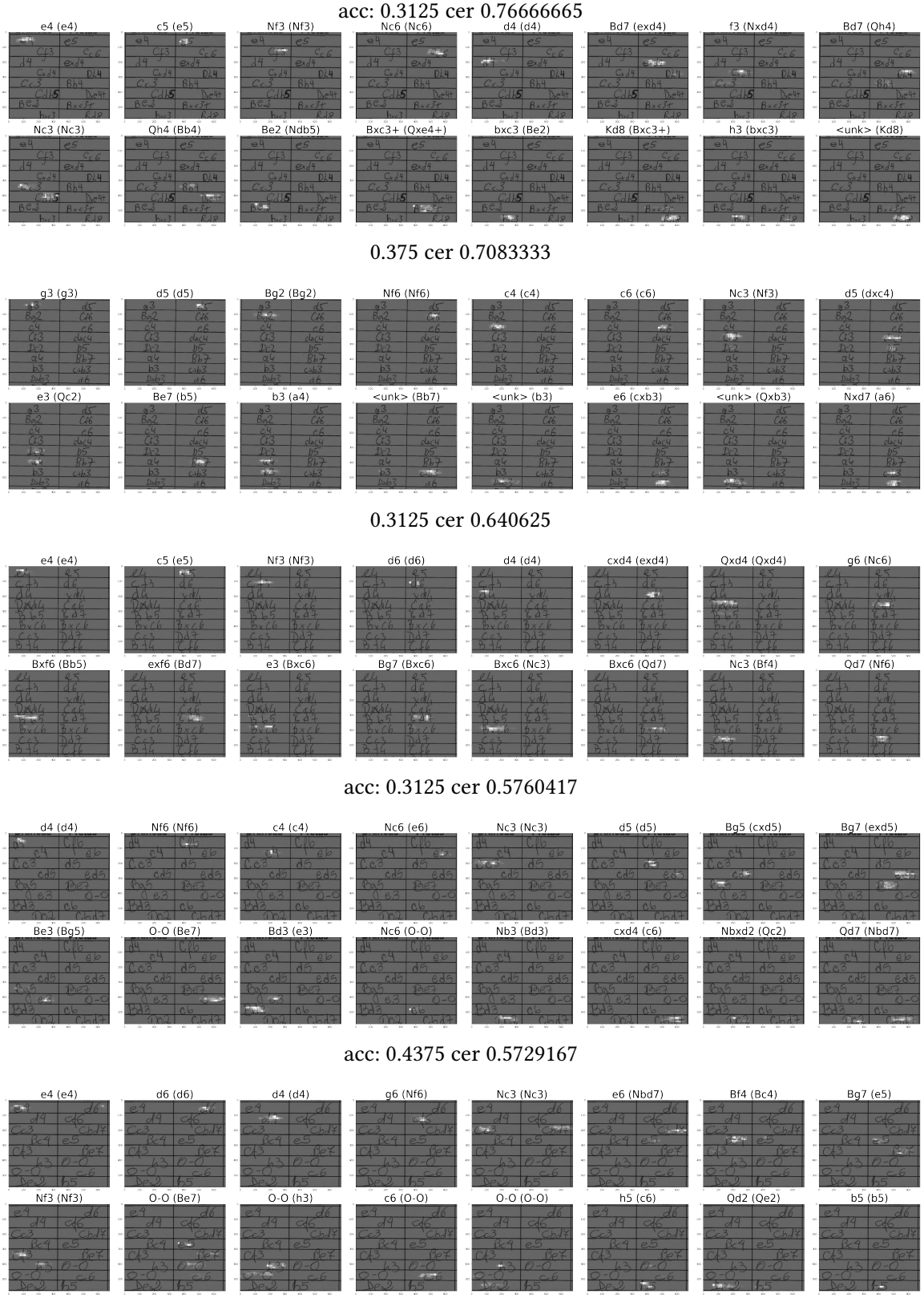


Figura 6.5: *As 5 planilhas de teste com os piores índices de predição.*

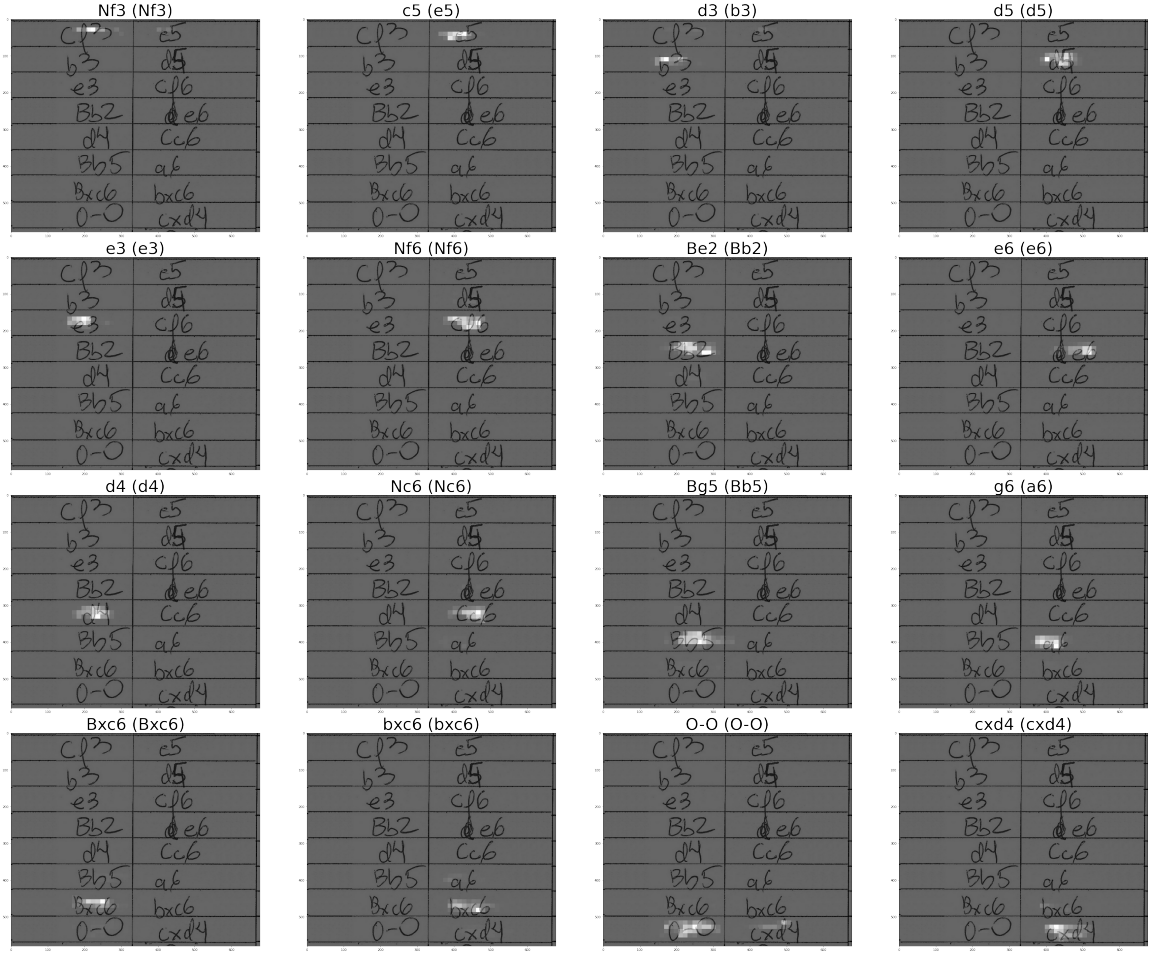
6.2.2 Avaliação qualitativa

Todos os treinamentos e testes relatados anteriormente foram feitos com imagens de planilhas reais obtidas de um único torneio e complementadas com imagens artificias (de planilhas com transcrições ou por imagens do tipo recortes-mosaico). Pelo fato de múltiplas planilhas terem sido preenchidas pelo mesmo jogador as métricas quantitativas calculadas sobre o conjunto de teste podem estar ligeiramente "contaminadas".

Como não conseguimos preparar um segundo conjunto de teste que seja 100% independente dos dados usados no treinamento (em termos de jogadores ou imagens de lances individuais), para termos uma noção da capacidade de generalização do modelo final, apresentamos algumas avaliações isoladas do modelo final sobre planilhas externas ao *dataset* usado no treinamento e teste.

Inicialmente mostramos o desempenho do modelo final sobre duas planilhas de um outro torneio (que chamaremos de torneio 2) ocorrido mais recentemente, e preenchidos por jogadores que não participaram do primeiro torneio (que chamaremos torneio 1).

O resultado sobre uma primeira planilha do torneio 2 é mostrado na figura 6.6. O CER é de 13,54%, um valor melhor que a média de 24,88% obtido com relação ao conjunto de teste (torneio 1). Pelo mapa da atenção podemos ver que o modelo foca a atenção corretamente sobre cada lance na imagem. Notamos também que o modelo acerta todas as ocorrências de dígitos, cometendo alguns erros no reconhecimento de letras. É interessante notar que os erros estão relacionados com letras visualmente semelhantes tais como a troca de “e” com “c”, de “b” por “e”, e de “b” ou “a” por "g". Podemos ver também que o modelo lê corretamente lances que estão rasurados, que são os casos dos lances “d5” e “e6”.

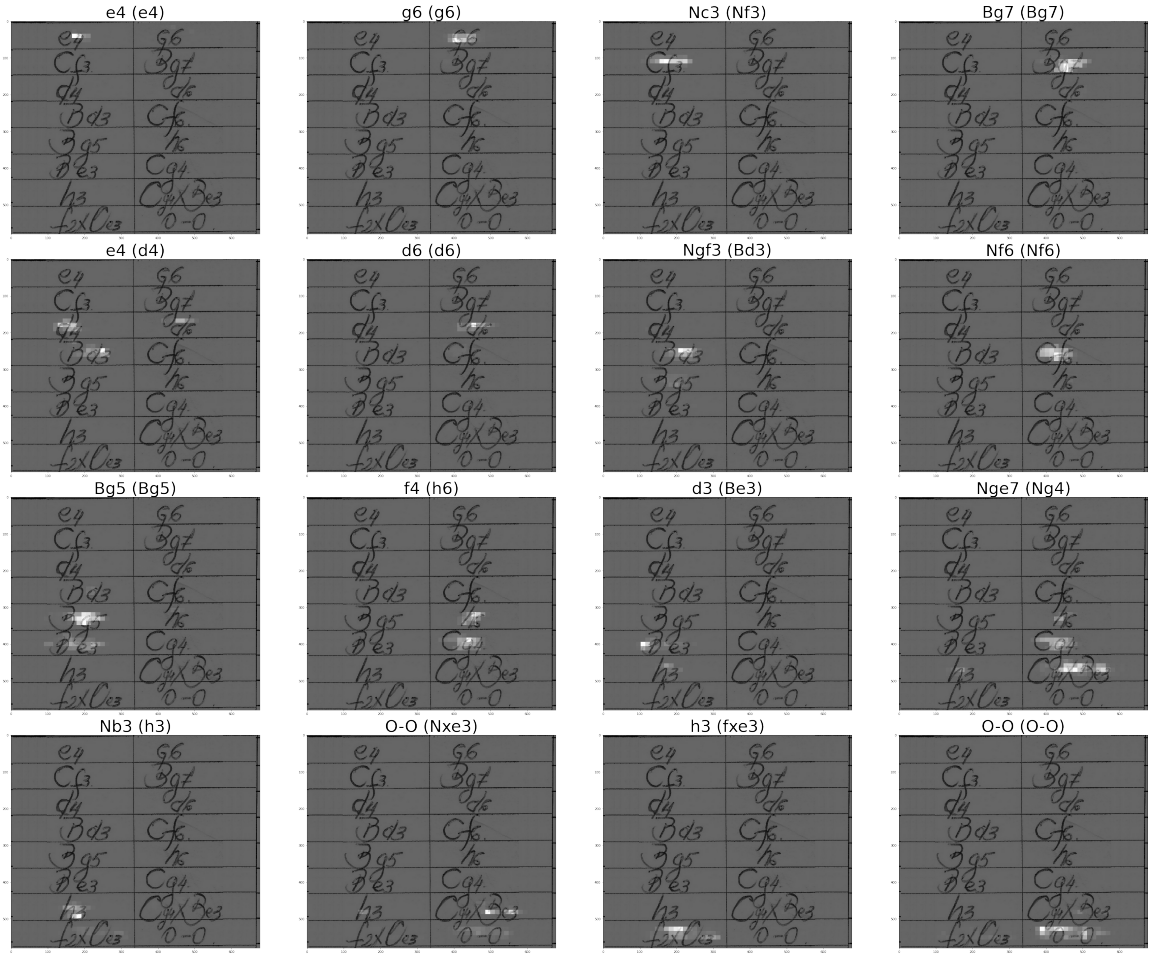


acc: 0.6875 cer 0.13541667

Seq. esperada:	**Seq obtida:**
Nf3 e5 b3 d5	Nf3 c5 d3 d5
e3 Nf6 Bb2 e6	e3 Nf6 Be2 e6
d4 Nc6 Bb5 a6	d4 Nc6 Bg5 g6
Bxc6 bxc6 O-O cxd4	Bxc6 bxc6 O-O cxd4

Figura 6.6: *Primeiro exemplo de predição para uma imagem de um segundo torneio, não utilizada no treinamento ou teste do modelo final.*

Os resultados sobre uma segunda planilha do torneio 2 estão na figura 6.7. Neste exemplo, o alinhamento não é tão claro nas posições intermediárias, indicando maior dificuldade do modelo para fazer a leitura. Visualmente podemos constatar também diferenças na característica da escrita. Em comparação às planilhas usadas no treinamento, os caracteres e também as linhas não estão claramente separadas entre si, dando margem para o modelo confundir a delimitação da região na imagem correspondente a cada lance, o que pode explicar a imprecisão no alinhamento. Esta planilha contém inclusive um erro de anotação na décima quarta posição, escrita como ”Cg4xBe3” quando a correta seria “Cxe3”, e o modelo erra ao predizer “O-O”, o que pode ter sido devido a parte do foco estar na linha abaixo.



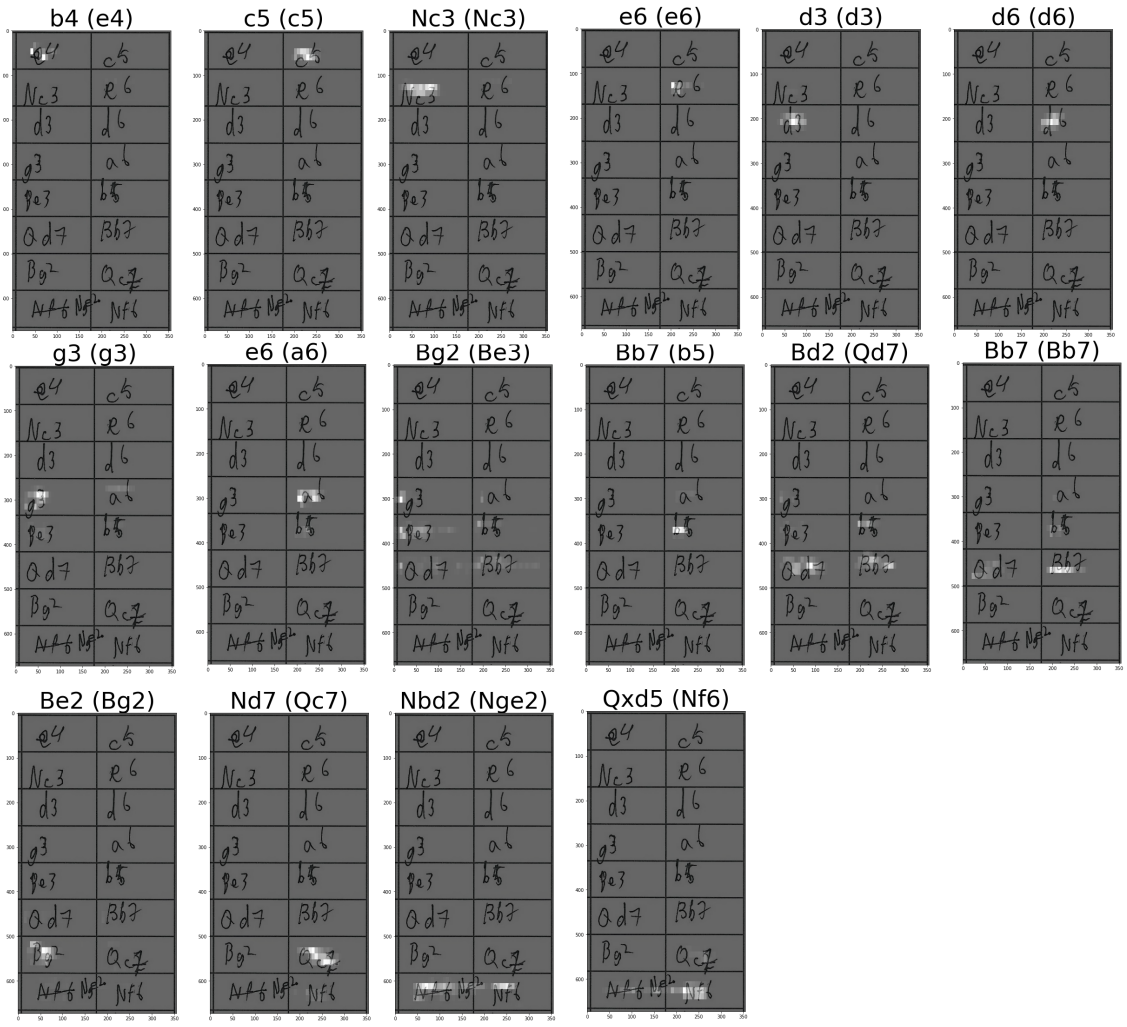
acc: 0.4375 cer 0.4322917

Seq. esperada:	**Seq obtida:**
e4 g6 Nf3 Bg7	e4 g6 Nc3 Bg7
d4 d6 Bd3 Nf6	e4 d6 Ngf3 Nf6
Bg5 h6 Be3 Ng4	Bg5 f4 d3 Nge7
h3 Nxe3 fxe3 O-O	Nb3 O-O h3 O-O

Figura 6.7: *Segundo exemplo de predição para uma imagem de um segundo torneio, não utilizada no treinamento ou teste do modelo final.*

Os dois exemplos acima indicam que o modelo generaliza relativamente bem. No primeiro exemplo, com características de escrita similares as das planilhas usadas no treinamento, o desempenho é muito bom. Já no segundo exemplo, no qual as características de escrita são bem distintas (por exemplo letras em estilo mais cursivas e caracteres em linhas distintas se tocando), o desempenho foi bem inferior. Porém é um comportamento esperado, dada a limitação de variedades e de quantidades de planilhas reais utilizadas no treinamento.

Por último, retomamos o exemplo que foi utilizado na introdução deste texto (seção 1.2) para ilustrar o desempenho obtido com softwares OCR comerciais. Vale notar que aquele recorte é de uma planilha anterior e não faz parte do conjunto de planilhas dos dois torneios. Utilizamos o nosso modelo final para processar o mesmo recorte daquele exemplo. O resultado é mostrado na figura 6.8.



acc: 0.4375 cer 0.3854167

Seq. esperada:	**Seq obtida:**
e4 c5 Nc3 e6 d3 d6	b4 c5 Nc3 e6 d3 d6
g3 a6 Be3 b5 Qd7 Bb7	g3 e6 Bg2 Bb7 Bd2 Bb7
Bg2 Qc7 Nge2 Nf6	Be2 Nd7 Nbd2 Qxd5

Figura 6.8: *Predição de uma imagem utilizada na avaliação de softwares comerciais citados na introdução.*

Nota-se que que nas posições intermediárias os pontos de atenção não estão muito bem definidos, porém melhoram nas posições finais. Apesar de errar o primeiro lance, o modelo acerta os lances iniciais e mais ao final o desempenho piora. Outro detalhe a ser notado é o fato de que a planilha está em inglês (“N” para Cavalo). Mesmo assim o modelo acerta a peça, apesar de o *dataset* ter sido construído com planilhas em português.

Embora tenha sido feita uma verificação para filtrar as planilhas que não estavam em inglês, eventualmente uma ou outra podem estar no *dataset* por engano. Esta poderia ser uma explicação para o modelo ter reconhecido a notação em inglês, mas também é plausível que o modelo tenha se baseado na previsibilidade da sequência, principalmente nas posições iniciais, que são padrões de abertura bastante utilizados.

Comparando com os softwares comerciais testados, o modelo teve um desempenho melhor. O CER para `PenToPrint` foi de 0.61, e para `Textract` 0.48, enquanto para o modelo final foi de 0.38. A tabela 6.1 é uma reprodução da mesma tabela mostrada na seção 1.2, que destaca os acertos e erros de cada software, porém complementados com o resultado do modelo.

Correto	e4	c5	Nc3	e6	d3	d6	g3	a6	Be3	b5	Qd7	Bb7	Bg2	Qc7	Nge2	Nf6
`Textract`	04	ck	Ne3	R6	d3	d6	g3	0	ge3	br	0d+	Bh?	Bg}	Qek	Aff Neh	Nf{
`PenToPrint`	04	05	No3	eb	d3						Qd7	367	Bg2	Q7		
Nosso	b4	c5	Nc3	e6	d3	d6	g3	e6	Bg2	Bb7	Bd2	Bb7	Be2	Nd7	Nbd2	Qxd5

Tabela 6.1: *Desempenho de dois softwares comerciais, `Textract` e `PenToPrint`, e do modelo final na leitura de um recorte com 16 lances, de uma planilha externa ao* dataset.

É importante destacar que essa comparação é apenas relativa uma vez que os OCRs comerciais são projetados para processar documentos genéricos enquanto o nosso modelo foi treinado especificamente para a leitura de planilhas de xadrez. Neste sentido, o resultado superior é esperado e mostra que incluir as características e propriedades dos dados do domínio de interesse no algoritmo ajuda no desempenho (comparado a uma solução mais genérica). Por outro lado, o treinamento de nosso modelo utilizou uma variedade bastante restrita de estilos de escrita. Não sabemos a quantidade de dados utilizadas no treinamento desses softwares, mas supomos que deve ser muito maior que a quantidade utilizada em nosso caso. Desta forma, temos indícios de que o desempenho de nosso modelo pode ser melhorado bastante fazendo-se o seu treinamento com uma quantidade maior de variedades de escrita.

Capítulo 7

Conclusão

Neste trabalho nos propusemos a desenvolver uma solução baseada em redes neurais profundas que permitisse um treinamento ponta a ponta para o problema de leitura de planilhas manuscritas de xadrez. Os principais desafios nesse problema são o posicionamento correto do foco de atenção na leitura e o reconhecimento da escrita em si. Na fase de pesquisa bibliográfica não foram encontrados na literatura nenhum trabalho que trate especificamente esse problema. Além disso, soluções comerciais de OCR para manuscritos, no que tange à tarefa de reconhecimento, também mostraram-se limitados.

Para viabilizar o desenvolvimento da solução, uma das primeiras providências foi a preparação de um conjunto de dados; imagens de planilhas preenchidas e respectiva rotulação associando uma sequência com a identificação dos lances. O desenvolvimento da solução teve algumas decisões importantes. Uma das principais foi a ideia de trabalhar com recortes das imagens de planilha. Isto é, ao invés de trabalhar com a imagem fiel da planilha, recortamos regiões das imagens correspondentes aos primeiros M lances do jogo. Esta decisão visou focar o esforço nos desafios de leitura dos lances, sem nos preocuparmos com outras informações como nome do jogador, data do jogo entre outros que podem estar na planilha. Outra vantagem de usar recortes é a possibilidade de trabalhar com sequências menores para investigar de forma mais eficiente (principalmente do ponto de vista computacional) questões gerais relacionadas ao processo de treinamento e comportamento do modelo. Também, baseado em trabalhos que tratam problemas similares, decidimos investigar no contexto do nosso problema uma estrutura de arquitetura composta por camadas convolucionais para a extração de *features* mais uma rede recorrente do tipo *encoder-decoder* com atenção. Com isso, o problema foi tratado como o de reconhecimento de uma sequência de lances, sem o reconhecimento individual dos caracteres.

O processo de desenvolvimento da solução foi, de fato, bastante exploratório e experimental. No início um período foi dedicado à familiarização com as ferramentas (Google colab, biblioteca TensorFlow), a operacionalização de processos de treinamento, e o funcionamento da arquitetura utilizada. Em seguida, experimentos mais específicos e controlados foram realizados.

Nesses experimentos, foi constatado inicialmente que modelos baseados na arquitetura de rede considerada possuem capacidade para processar os dados, porém têm tendência a

"decorar" as sequências (não sendo, portanto, capazes de aprender o alinhamento correto). Posteriormente, ficou evidente que a tarefa de leitura inclui três subtarefas que precisam ser aprendidas concomitantemente para que um treinamento ponta a ponta seja realizada com sucesso. Essas subtarefas são o modelo de linguagem (previsibilidade), alinhamento entre entrada e saída, e o reconhecimento do lance propriamente. Identificamos alguns fatores que são críticos para promover ou perturbar o equilíbrio do aprendizado dessas três subtarefas. De forma geral, constatamos que uma quantidade mínima de dados de treinamento e igualmente um tamanho mínimo da imagem de entrada são fundamentais. Um resumo desses achados está apresentado na seção 6.1.

7.1 Contribuições

Uma primeira contribuição desta dissertação é a descrição de uma configuração de arquitetura, das imagens de entrada e dos parâmetros de treinamento que possibilitam o treinamento ponta a ponta para o problema considerado. Um modelo melhorado gerado a partir de um treinamento baseado nessa configuração obteve acurácia de teste de 65.78% e CER de 24,88% para sequências de tamanho 16. Como produtos desta dissertação resultaram também um pequeno *dataset* com imagens de planilhas de torneio e outro de planilhas preenchidas por voluntários, além do código desenvolvido que está disponível em https://github.com/sergiohayashi/chess-attention.thesisHayashi2021

Outra importante contribuição foi a identificação das subtarefas e os fatores que influenciam o aprendizado concomitante delas, possibilitando assim um treinamento ponta a ponta efetivo.

Acreditamos que esse tipo de estudo contribui para um melhor entendimento da solução como um todo, dada a complexidade das modernas redes neurais. Também acreditamos que as investigações experimentais relatadas podem ser úteis na solução de outros problemas para os quais não existe uma solução que possa ser usada como referência, principalmente nos casos em que a quantidade de dados disponível é pequena.

Em particular, destacamos a relevância do aumento de dados de treinamento por meio da criação artificial de imagens. Diferentemente do bem conhecido método de "*data augmentation*", os aumentos utilizados neste trabalho foram planejados em vista de habilidades desejadas no modelo (por exemplo, o uso de sequências randômicas para fortalecer a habilidade de alinhamento).

7.2 Limitações e trabalhos futuros

Uma das limitações do estudo realizado foi a quantidade reduzida de planilhas reais de jogos para serem usados no treinamento e teste. Essa limitação foi em parte devida à situação de isolamento imposta pela pandemia do COVID-19. Como consequência, os conjuntos de teste e de treinamento podem conter planilhas escritas por um mesmo jogador. Além disso, a biblioteca com os recortes de um lance utilizada na construção dos recortes mosaico contém os recortes de lances extraídos das imagens de planilhas reais, incluindo aquelas no conjunto de teste. Por conta disso, é possível que os resultados sobre o conjunto

de testes sejam ligeiramente otimistas.

Outra limitação está relacionada com a disponibilidade de recursos computacionais. Os experimentos controlados foram todos realizados com recortes de 8 linhas (16 lances) devido à limitações de memória. Portanto, não é possível afirmar se outros fatores que afetam o treinamento poderiam ser detectados caso experimentos com sequências maiores fossem realizados.

Uma terceira limitação diz respeito à configuração da arquitetura utilizada assim como os parâmetros de treinamento utilizados. Não foi realizado nenhum experimento para a otimização desses elementos.

As limitações listadas acima são, na verdade, oportunidades para trabalhos futuros. Há certamente várias melhorias que podem ser feitas em torno da arquitetura de rede adotada. Por exemplo, podemos investigar uma dimensão adequada do módulo convolucional tal que o mesmo promova o balanceamento ótimo entre o tamanho da imagem de entrada e o tamanho do mapa de *features*, ou a dimensão ótima para representação de dados interna à rede, os tipos de nó na rede recorrente, entre outras.

Em particular, o ajuste do módulo convolucional pode resultar em uma rede menor, e desta forma mitigar em parte a questão do custo computacional em termos de utilização de memória, além de implicar também na redução do tempo de processamento. Neste caso, teríamos um módulo convolucional customizado ao invés de módulos de CNNs pré-treinadas sobre o ImageNet, que poderiam ser pré-treinados com dados do contexto de HTR ou mesmo serem treinados em conjunto com a rede completa.

Outra sugestão de trabalho futuro é a realização de uma avaliação mais ampla e sistemática, considerando-se sequências maiores e dados de diferentes torneios, para testar os limites da solução.

Finalmente, no presente trabalho o foco foi a realização de um treinamento ponta a ponta e, uma vez que encontramos as condições suficientes, o próximo passo natural é buscar a melhoria do desempenho. Acreditamos que um caminho promissor para melhorar o desempenho do modelo são as abordagens centradas em dados, tais como o *curriculum learning* (Bengio *et al.*, 2009). Nesta abordagem, emprega-se um princípio similar ao usado no ensino escolar: ensina-se inicialmente um conteúdo simplificado e gradativamente vai-se aumentando a complexidade ao longo dos anos. No contexto de aprendizado de máquina, a arquitetura e os parâmetros ficam fixos (como se fosse um estudante) e trabalha-se a distribuição dos dados de treinamento de forma a favorecer o treinamento de alguma habilidade específica (um nível de conteúdo). Na prática, em linhas gerais, o processo pode ser implementando como um processo iterativo. Uma vez terminada uma iteração, faz-se uma análise de erros para identificar erros comuns. Então prepara-se uma distribuição de dados que visa reduzir esse tipo de erro e treina-se o modelo por mais algumas épocas. Um exemplo que foi testado preliminarmente mas não está relatado nesta dissertação é o tratamento do erro devido à confusão entre as letras "e" e "c". Utilizando-se uma porcentagem de recortes mosaico consistindo apenas de lances que envolvem uma dessas letras, observamos uma melhora significativa na acurácia de teste. Uma abordagem assim busca promover a melhoria incremental dos modelos e pode ser adequada para adaptar o modelo a novos cenários de uso (por exemplo, um novo torneio, com jogadores distintos).

Apêndice A

Função de custo

Para a função de custo é calculado o *cross entropy loss* interpretando a saída do decoder como a probabilidade em *logits* de cada uma das words do vocabulário.

A função de custo é calculado sobre cada elemento da sequência individualmente através da função `tf.keras.losses.SparseCategoricalCrossentropy` passando como parâmetro `from_logits= True`.

Esta função aceita como parâmetro de entrada a predição e o valor esperado. O parâmetro do valor esperado é passado como um número inteiro representando a posição no vocabulário.

O parâmetro da predição é passado como um vetor do tamanho do vocabulário, onde cada posição representa a probabilidade do resultado ser a palavra daquela posição no formato de valor *logit* da probabilidade.

A função *logit* é o inverso da sigmoide, e mapeia a probabilidade de $(0, 1)$ para $(-\infty, +\infty)$ de forma não linear e é definido como:

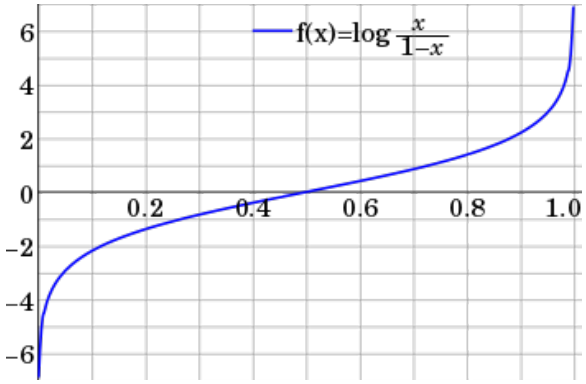
$$logit(p) = log\frac{p}{1-p} \tag{A.1}$$

Na figura A.1 a curva do *logit*. Pode se observar que desta forma o *range* de valores representados no vetor de predição torna-se ilimitado, e não mais entre $(0, 1)$ facilitando desta forma a operação na rede.

A função $\frac{p}{1-p}$ é também chamado de $odds$ e $logit(p)$ também é chamado de $log - odds$.

O inverso do cálculo do *logit* é dado pela seguinte fórmula, que é a função sigmoide.

$$p = \frac{e^{logit(p)}}{1 + e^{logit(p)}} \tag{A.2}$$



fonte: https://en.wikipedia.org/wiki/Logit

Figura A.1: *Curva da função* logit *entre 0 e 1*

Sobre esta distribuição linear de probabilidade é aplicada a fórmula do cálculo da distância da entropia cruzada ou *cross entropy loss* cuja formula é:

$$H(p, q) = -\sum_{x} p_x log(\, q_x \,) \tag{A.3}$$

Onde p é o *ground truth* e q a distribuição de probabilidade da predição.

Exemplo de cálculo do custo Considere um vocabulário de tamanho 3, e uma predição em um determinado ponto da sequência tal que a resposta correta é palavra na posição 1, e o vetor *logits* dado pela rede como predição é `[1.0, 5.0, 3.0]`. Vamos calcular a função de custo desta predição.

A chamada à função de erro no *Tensorflow* fica como `tf.keras.losses .SparseCategoricalCrossentropy( from_logits=True )( y_true=1, y_pred=[1.0, 5.0, 3.0]).numpy()` e dá um resultado de `0.1429317`. Abaixo o cálculo passo a passo até chegar neste valor.

A predição que é interpretada como *log probability* não normalizada (*logit*) portanto é inicialmente transformada em distribuição de probabilidade linear (necessária para a função de *cross entropy loss*). Isso é implementado na função *softmax* do Tensorflow:

Na implementação do *Tensorflow*, o *softmax* faz o cálculo em 3 passos. (1) Normaliza a distribuição subtraindo pelo máximo e em seguida (2) transforma em probabilidade linear pela função exponencial e (3) calcula o *softmax* tal que o total dê igual a 1. A fórmula é conforme abaixo:

$$a_i = \frac{e^{z_i - \max(z)}}{\sum_{\forall j} e^{z_j - \max(z)}} \tag{A.4}$$

Aplicando a fórmula temos:

$$[1.0, 5.0, 3.0] \tag{A.5}$$
$$= [[-4., 0., -2.]] \quad \text{subtraindo pelo max} \tag{A.6}$$
$$= [0.01831564, 1, 0.13533528] \quad (e^x) \tag{A.7}$$
$$= [0.01587624, 0.86681333, 0.11731043] \quad \text{softmax} \tag{A.8}$$

Temos então a distribuição de probabilidade linear. Para o cálculo do erro, como neste caso a resposta esperada $p(x)$ tem o valor 1 para o índice da palavra esperada e 0 no restante prevalece o valor de $log(q_x)$ da posição 1.

Temos então,

$$H(p, q) = -\sum_x p_x log(\ q_x\ ) \tag{A.9}$$
$$= -\ (0 * log(0.01587624) + 1 * log(0.86681333) + 0 * log(0.11731043)) \tag{A.10}$$
$$= -log(0.8668133) = -(-0.14293167) = 0.14293167 \tag{A.11}$$

Portanto, o valor final para a função de *loss* para predição `[1.0, 5.0, 3.0]` sendo o valor esperado a palavra no índice `1` do vocabulário é `0.14293167`.

O *loss* é calculado desta forma individualmente para cada elemento da sequência e aplicado ao final para o cálculo do gradiente descendente.

Apêndice B

Trechos de códigos

B.1 Trecho de código do modelo

```
class BahdanauAttention(tf.keras.Model):

    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, features, hidden):
        # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)


        # hidden shape == (batch_size, hidden_size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # attention_hidden_layer shape == (batch_size, 64, units)
        # score shape == (batch_size, 64, 1)
        # This gives you an unnormalized score for each image feature.
        score = tf.nn.tanh(self.W1(features)
                           + self.W2(hidden_with_time_axis))

        # attention_weights shape == (batch_size, 64, 1)
        # you get 1 at the last axis because you are applying score to self.V
        attention_weights = tf.nn.softmax(self.V(score), axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return (context_vector, attention_weights)


class CNN_Encoder(tf.keras.Model):

```

```
36      def __init__(self, embedding_dim, units):
37         super(CNN_Encoder, self).__init__()
38         self.units = units
39         self.bgru = \
40            tf.keras.layers.Bidirectional(tf.keras.layers.GRU(self.units,
41               dropout=0.2, return_sequences=True))
42         self.drop = tf.keras.layers.Dropout(0.2)
43         self.fc = tf.keras.layers.Dense(embedding_dim)
44
45      def call(self, x):
46         x = self.bgru(x)
47         x = self.drop(x)
48         x = self.fc(x)
49         x = tf.nn.relu(x)
50         return x
51
52
53   class RNN_Decoder(tf.keras.Model):
54
55      def __init__(self,embedding_dim,units,vocab_size):
56         super(RNN_Decoder, self).__init__()
57         self.units = units
58
59         self.embedding = tf.keras.layers.Embedding(vocab_size,
60               embedding_dim)
61         self.gru = tf.keras.layers.GRU(self.units, dropout=0.2,
62               return_sequences=True, return_state=True,
63               recurrent_initializer='glorot_uniform')
64         self.fc1 = tf.keras.layers.Dense(self.units)
65         self.fcF = tf.keras.layers.Dense(vocab_size)
66         self.drop = tf.keras.layers.Dropout(0.2)
67         self.attention = BahdanauAttention(self.units)
68
69      def call(self,x,features,hidden):
70
71         # defining attention as a separate model
72         (context_vector, attention_weights) = self.attention(features,
73               hidden)
74
75         # x shape after passing through embedding == (batch_size, 1,
              embedding_dim)
76         x = self.embedding(x)
77
78         # x shape after concatenation == (batch_size, 1, embedding_dim +
              hidden_size)
79         x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
80         (output, state) = self.gru(x)
81
82         # shape == (batch_size, max_length, hidden_size)
83         x = self.fc1(output)
84         x = self.drop(x)
85
86         # x shape == (batch_size * max_length, hidden_size)
87         x = tf.reshape(x, (-1, x.shape[2]))
88
89         # output shape == (batch_size * max_length, vocab)
```

```
90          x = self.fcF(x)
91          return (x, state, attention_weights)
92
93       def reset_state(self, batch_size):
94          return tf.zeros((batch_size, self.units))
```

B.2 Trecho de código do laço de treinamento

```
1
2    encoder = CNN_Encoder(EMBEDDING_DIM, UNITS)
3    if FREEZE_ENCODER:
4       encoder.trainable = False
5
6    decoder = RNN_Decoder(EMBEDDING_DIM, UNITS, VOCAB_SIZE)
7
8    optimizer = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)
9    loss_object = \
10      tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
11         reduction='none')
12
13   def loss_function(real, pred):
14      mask = tf.math.logical_not(tf.math.equal(real, 0))
15      loss_ = loss_object(real, pred)
16
17      mask = tf.cast(mask, dtype=loss_.dtype)
18      loss_ *= mask
19      return tf.reduce_mean(loss_)
20
21   @tf.function
22   def train_step(img_tensor, target, train_length):
23      loss = 0
24      zeros = np.zeros(target.shape[0]).astype(int)
25
26      hidden = decoder.reset_state(batch_size=target.shape[0])
27
28      dec_input = \
29         tf.expand_dims((zeros if NO_TEACH else [tokenizer.word_index['<start>'
30                     ]] * target.shape[0]), 1)
31
32      with tf.GradientTape() as tape:
33         features = encoder(img_tensor)
34
35         for i in range(1, train_length + 1):
36
37            # passing the features through the decoder
38            (predictions, hidden, _) = decoder(dec_input, features,
39                  hidden)
40
41            loss += loss_function(target[:, i], predictions)
42            train_acc_metric.update_state(target[:, i], predictions)
43
44            # using teacher forcing, or not (parameter)
45            dec_input = tf.expand_dims((zeros if NO_TEACH else target[:
46                  , i]), 1)
47
```

```
48        total_loss = loss / int(train_length)
49
50        trainable_variables = encoder.trainable_variables \
51           + decoder.trainable_variables
52        gradients = tape.gradient(loss, trainable_variables)
53        optimizer.apply_gradients(zip(gradients, trainable_variables))
54
55        return (loss, total_loss)
56
57
58    @tf.function
59    def test_step(img_tensor, target, train_length):
60        loss = 0
61        zeros = np.zeros(target.shape[0]).astype(int)
62
63        hidden = decoder.reset_state(batch_size=target.shape[0])
64        dec_input = \
65           tf.expand_dims((zeros if NO_TEACH else [tokenizer.word_index['<start>'
66                       ]] * target.shape[0]), 1)
67        features = encoder(img_tensor)
68
69        for i in range(1, train_length + 1):
70           (predictions, hidden, _) = decoder(dec_input, features, hidden)
71
72           loss += loss_function(target[:, i], predictions)
73           valid_acc_metric.update_state(target[:, i], predictions)
74
75           dec_input = tf.expand_dims((zeros if NO_TEACH else target[:,
76                                i]), 1)
77
78        total_loss = loss / int(train_length)
79        return (loss, total_loss)
```

Apêndice C

Experimentos exploratórios

Este apêndice detalha alguns dos experimentos exploratórios realizados em uma etapa inicial. A configuração utilizada nos experimentos exploratórios variou bastante. Um mínimo de informação, tais como o ambiente para a execução dos experimentos e os dados usados, está disponível na seção 4.2.1. Aqui descrevemos as principais explorações realizadas.

C.1 Fase exploratória 1: Convolução

Nesta fase treinamos um modelo baseado em uma arquitetura composta basicamente de camadas de convolução (ver diagrama na figura C.1). A arquitetura consiste de 3 camadas convolucionais, cada uma seguida de *max pooling*, e com número de filtros 64, 128 e 256, respectivamente, seguidas por 3 camadas totalmente conectadas, com 512, 256 e 261 nós, respectivamente.

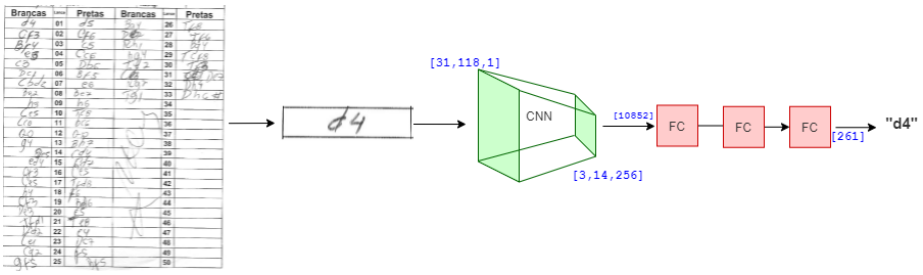


Figura C.1: *Diagrama do modelo explorado na fase 1. A entrada usada pela rede são recortes de um lance, extraídos da imagem de uma planilha.*

A entrada da rede consiste de imagens de recorte contendo um único lance, com dimensão 31×118. O número de nós da última camada (a camada de saída) é 261, correspondendo ao tamanho do vocabulário considerado. Foram utilizados todos os 10.944 recortes reais de um lance (ver tabela 4.1), sendo 2000 para validação e o restante para treinamento. Um exemplo de recorte é mostrado na figura C.2.

Neste experimento, o treinamento convergiu para uma acurácia de 90% após 25 épocas, porém a acurácia na base de validação foi de 54%, correspondendo a um forte *overfitting*.

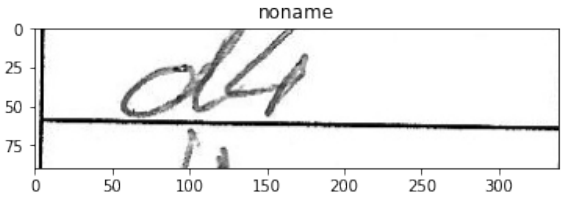


Figura C.2: *Exemplo de um recorte real contendo um único lance.*

Não obstante, diante desses resultados, concluímos que uma rede convolucional simples é capaz de extrair das imagens informações suficientes para o reconhecimento de lances individuais, e nesta fase não nos preocupamos com o *overfitting*.

C.2 Fase exploratória 2: Convolução + recorrência

Um modelo baseado em uma arquitetura com somente camadas de convolução não é suficiente para predição de uma sequência. Portanto, para recortes com mais de um lance foi considerada uma arquitetura composta de um módulo CNN e outro RNN. Conforme já discutido, o tamanho da saída da camada convolucional em geral não é igual ao tamanho da sequência-alvo. Assim, uma estrutura do tipo *encoder-decoder* é adequada para o módulo RNN. A figura C.3 mostra a arquitetura utilizada nos experimentos desta fase.

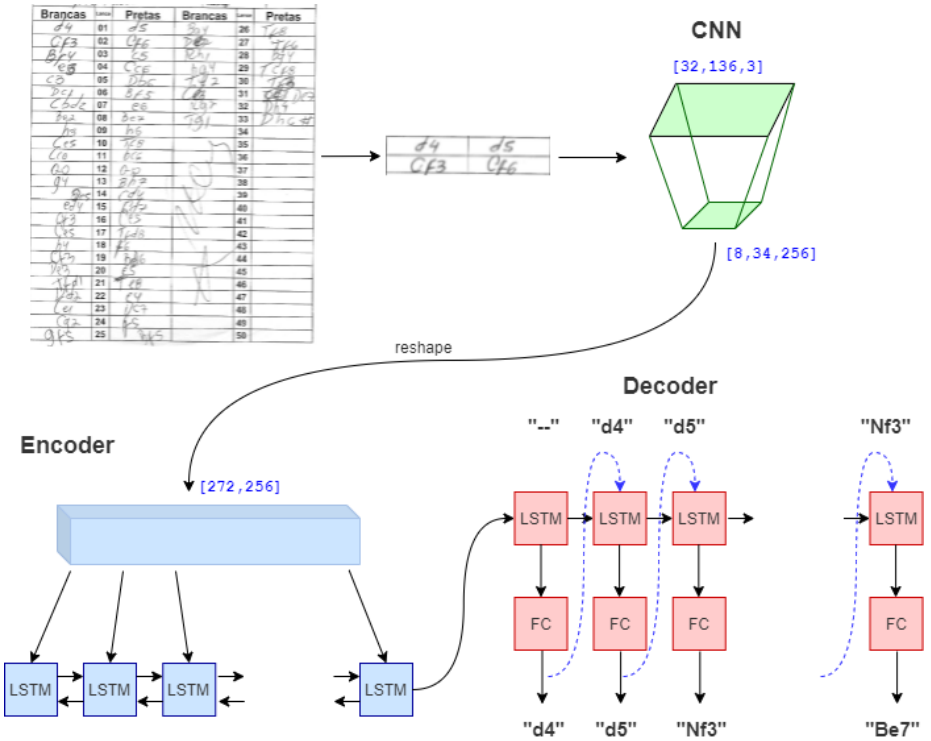


Figura C.3: *Diagrama do modelo explorado na fase 2.*

A arquitetura consiste de 3 camadas de convolução, cada uma seguida por *max pooling*, como no caso do modelo anterior. Em seguida, segue uma camada recorrente formada por nós BLSTM, que corresponde ao *encoder*. A sequência de entrada do *encoder* é o mapa de

features, vetorizado. O estado final do *encoder* alimenta uma segunda camada recorrente, do tipo LSTM, que corresponde ao *decoder*.

Note que a primeira posição do *decoder* recebe o estado final do *encoder* mais o código da *string vazia*. Os demais nós recebem o estado e a saída do nó anterior. Durante o treinamento, usamos a estratégia *Teacher Forcing*, ou seja, ao invés da saída do nó da posição anterior, passamos o *ground-truth*. Por exemplo, se o rótulo correto é a sequência "c3 Nf6 e5 Nd5", então a rede recorrente, na parte do *decoder* recebe como sequência de entrada "_, c3, Nf6, e5". Isto é, para inferir o primeiro elemento "c3", a entrada do primeiro nó é vazia (_). Já para inferir "Nf6" o segundo nó recebe "c3", e para inferir o "e5" o terceiro nó recebe "Nf6", e assim por diante.

C.2.1 Experimento básico

O primeiro experimento com a arquitetura da figura C.3 consistiu em variar o tamanho da sequência de entrada. Os resultados estão descritos a seguir.

- **2 linhas (4 lances):** foram usados 4.788 recortes reais (tabela 4.1), com tamanho das imagens igual a 32 × 136, separados em conjunto de treinamento e validação. O modelo demonstrou uma grande capacidade de acerto, com 99,9% de acurácia na base de treinamento, mas apresentou um *overfitting* bastante grande, com acurácia de 16% na base de validação. Este resultado forneceu-nos indícios de que o modelo possivelmente poderia estar decorando as amostras de treinamento, ao invés de estar aprendendo características relevantes ao reconhecimento.
- **8 linhas:** Resultado análogo ao experimento com duas linhas foi observado também para recortes reais de 8 linhas (16 lances).
- **página cheia:** Neste caso foram apenas 684 recortes reais, de tamanho 360×318. Neste teste também foi observado que o comportamento do modelo com relação à acurácia de treinamento continuou tão alta quanto a observada com recortes menores, ou seja, uma acurácia bastante alta no treinamento, com rápida convergência, e desempenho sofrível no conjunto de validação.

C.2.2 Experimentos de ablação

Para entender melhor o comportamento descrito acima, no qual o modelo aparentemente atinge um bom desempenho facilmente, foram feitos dois experimentos de ablação. Nesses experimentos foram utilizados recortes reais de página cheia.

1. No primeiro experimento de ablação foi utilizada como entrada uma mesma imagem fixa, e como saída cada uma das sequências no conjunto de treinamento. Aqui a intenção é testar a hipótese de que a rede não está fazendo uso da informação proveniente da imagem de entrada.
2. No segundo experimento de ablação, consideramos uma situação ainda mais extrema, no qual usamos somente o *decoder*. Isto é, foi usado um modelo sem o CNN e sem o *encoder*. A entrada do *decoder* é simplesmente a sequência final esperada (*ground-truth*) deslocada de uma posição. Este experimento visou entender a capacidade do

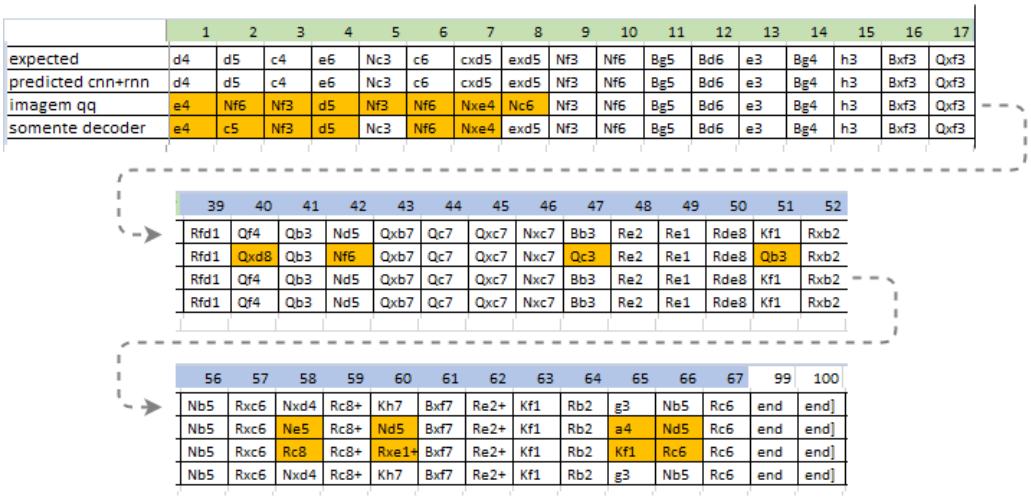
decoder, sozinho, de gerar a saída esperada. É importante notar que o treinamento usa a estratégia de *teacher forcing*.

Os resultados obtidos estão mostrados na tabela C.1. Podemos ver que, paradoxalmente, a acurácia de treinamento aumenta à medida que menos informação da imagem de entrada é fornecida à rede.

Tipo de ablação	Acurácia de treinamento	Épocas	Validação
Sem ablação	77%	210	forte *overfitting*
Imagem de entrada fixa	81%	210	forte *overfitting*
Somente *decoder*	95%	250	forte *overfitting*

Tabela C.1: *Testes de ablação para verificar a relevância da imagem de entrada para a predição.*

Na figura C.4 mostramos um exemplo de predição. São mostrados três blocos. Em cada bloco, a primeira linha indica a posição na sequência, a segunda linha os respectivos *ground-truth* e as três últimas linhas as predições geradas nos três casos testados: (1) normalmente, com a imagem de entrada correta; (2) com a imagem de entrada fixa e variando-se a sequência-alvo; (3) sem imagem de entrada, usando apenas o módulo *decoder*.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
expected	d4	d5	c4	e6	Nc3	c6	cxd5	exd5	Nf3	Nf6	Bg5	Bd6	e3	Bg4	h3	Bxf3	Qxf3
predicted cnn+rnn	d4	d5	c4	e6	Nc3	c6	cxd5	exd5	Nf3	Nf6	Bg5	Bd6	e3	Bg4	h3	Bxf3	Qxf3
imagem qq	e4	Nf6	Nf3	d5	Nf3	Nf6	Nxe4	Nc6	Nf3	Nf6	Bg5	Bd6	e3	Bg4	h3	Bxf3	Qxf3
somente decoder	e4	c5	Nf3	d5	Nc3	Nf6	Nxe4	exd5	Nf3	Nf6	Bg5	Bd6	e3	Bg4	h3	Bxf3	Qxf3

	39	40	41	42	43	44	45	46	47	48	49	50	51	52
	Rfd1	Qf4	Qb3	Nd5	Qxb7	Qc7	Qxc7	Nxc7	Bb3	Re2	Re1	Rde8	Kf1	Rxb2
	Rfd1	Qxd8	Qb3	Nf6	Qxb7	Qc7	Qxc7	Nxc7	Qc3	Re2	Re1	Rde8	Qb3	Rxb2
	Rfd1	Qf4	Qb3	Nd5	Qxb7	Qc7	Qxc7	Nxc7	Bb3	Re2	Re1	Rde8	Kf1	Rxb2
	Rfd1	Qf4	Qb3	Nd5	Qxb7	Qc7	Qxc7	Nxc7	Bb3	Re2	Re1	Rde8	Kf1	Rxb2

	56	57	58	59	60	61	62	63	64	65	66	67	99	100
	Nb5	Rxc6	Nxd4	Rc8+	Kh7	Bxf7	Re2+	Kf1	Rb2	g3	Nb5	Rc6	end	end]
	Nb5	Rxc6	Ne5	Rc8+	Nd5	Bxf7	Re2+	Kf1	Rb2	a4	Nd5	Rc6	end	end]
	Nb5	Rxc6	Rc8	Rc8+	Rxe1+	Bxf7	Re2+	Kf1	Rb2	Kf1	Rc6	Rc6	end	end]
	Nb5	Rxc6	Nxd4	Rc8+	Kh7	Bxf7	Re2+	Kf1	Rb2	g3	Nb5	Rc6	end	end]

Figura C.4: *Exemplo de predição resultante do estudo de ablação para uma imagem de página cheia, feita com um modelo CNN+RNN. A primeira linha indica a posição na sequência.* **expected***: é o* ground truth*,* **predicted cnn+rnn***: é a predição do modelo treinado normalmente,* ***imagen qq****: é a predição usando-se uma mesma imagem para todo os exemplos de treinamento,* ***somente*** **decoder***: é a predição feita pelo modelo composto somente com a parte do* decoder*. Observa-se que sem a imagem a predição é tão boa quanto com a imagem na entrada.*

Na parte inicial da sequência o modelo com a imagem correta acerta mais. Por outro lado, o modelo com a imagem de entrada fixa – e portanto sem correspondência com a sequência de saída que está se tentando predizer (3ª linha no bloco) e o modelo somente com o *decoder* (4ª linha) chutam sempre o "e4" e erram a predição dos lances iniciais. Isto faz sentido pois estes dois últimos modelos não tem acesso à imagem de entrada para fazer o chute inicial. No entanto, nas posições mais à frente, o modelo com a imagem correta na entrada apresenta comportamento parecido ao do modelo que usa uma imagem fixa na

entrada e o modelo com apenas o *decoder* apresenta um desempenho ligeiramente superior aos dois primeiros.

A partir da observação acima, entendemos que em posições não iniciais da sequência, a informação da imagem não é utilizada pela rede; o *decoder* por si só é suficiente para fazer uma boa inferência. Em outras palavras, podemos dizer que a RNN por si só exibe capacidade de "aprender" o modelo de linguagem, que diz respeito à previsibilidade do próximo lance baseado nos anteriores.

Logo, o problema da inferência da sequência de lances do xadrez parece ser tal que um modelo recorrente consegue facilmente memorizar a sequência. Isto não é surpreendente se aceitarmos que o número de possíveis lances, dada uma certa configuração de jogo, é limitado e que existem padrões de jogo preferenciais, mais frequentemente empregados.

O fato da informação da imagem ser relevante apenas nas posições iniciais da sequência, e a rede se apoiar somente na previsibilidade da sequência indica que, apesar de apresentar um desempenho médio bom, redes do tipo experimentados nesta fase são incapazes de generalizar.

C.3 Fase exploratória 3: Convolução + recorrência + atenção

Até aqui foram feitos testes exploratórios com modelos mais simples. Nesta fase exploramos um modelo baseado em uma arquitetura com todos os módulos (convolucional, *encoder*, *decoder* e atenção), mostrada na figura C.5. A arquitetura foi baseada em uma solução proposta para o problema de *image captioning* em Xu *et al.*, 2015.

Na parte convolucional, usamos as camadas convolucionais pré-treinadas do *InceptionV3* (Szegedy *et al.*, 2016). As imagens de entrada foram ajustadas para o tamanho de 299×299. A saída do módulo convolucional são mapas de *features* com dimensão 8×8 e profundidade 2048. Consequentemente, 8×8 é a menor granularidade possível para a camada da atenção. Note que neste modelo o *encoder* não é uma rede recorrente; ela consiste de uma rede totalmente conectada convencional aplicada igualmente em toda a extensão da sequência, seguida por ativação *ReLU*. O *decoder* é constituído por uma camada recorrente formada por nós do tipo GRU, seguidos por 2 camadas totalmente conectadas.

Usamos recortes reais de duas linhas (4 lances), e posteriormente recortes de quatro linhas, consistindo de 4.788 e 684 amostras, respectivamente (ver tabela 4.1).

Na figura C.6 mostramos exemplos de predição na base de validação, com a visualização dos pontos de atenção. No primeiro exemplo, um recorte real de duas linhas, podemos ver que o modelo aprendeu corretamente o alinhamento (evidenciado pelos pontos de atenção). Porém, no segundo exemplo, um recorte real de quatro linhas, o alinhamento não acontece de forma correta.

Este experimento mostra que para recortes de duas linhas (ou seja, sequências de tamanho quatro) o modelo consegue aprender o alinhamento. Isto é, segue a ordem correta

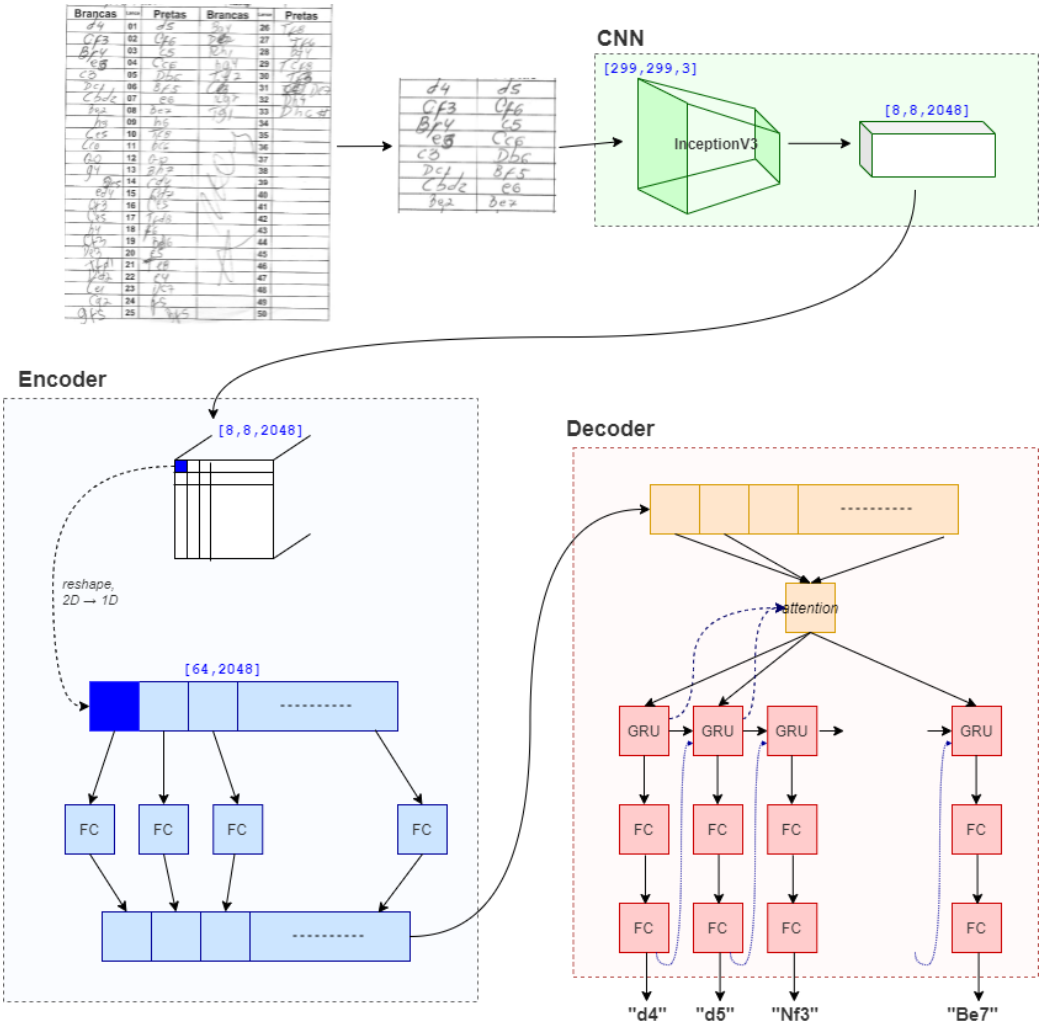
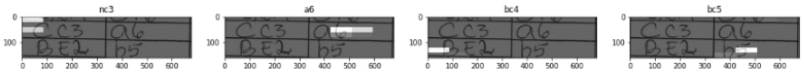
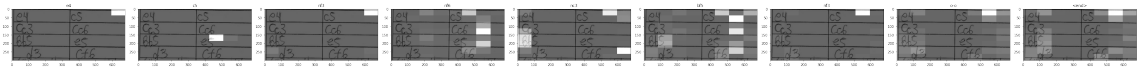


Figura C.5: *Modelo baseado em uma arquitetura completa, formada por camadas de convolução,* encoder *e* decoder*, e com mecanismo de atenção.*

de leitura, identificando o momento de quebra de linha. Também reconhece relativamente bem a sequência de palavras. No entanto, no caso de recortes de quatro linhas pode-se observar que o alinhamento está totalmente errado, indicando a dificuldade da rede em manter o alinhamento correto para sequências longas.



Seq. esperada: Nc3 a6 Be2 b5
Seq. obtida: Nc3 a6 Bc4 Bc5



Seq. esperada: e4 c5 Nc3 Nc6 Bb5 e5 d3 Nf6
Seq. obtida: e4 c5 Nf3 Nf6 Nc3 Bf5 Nf3 0-0

Figura C.6: *Resultado da predição e visualização de alinhamento para recortes reais de duas (na parte superior) e quatro linhas (na parte inferior).*

Referências

[Bahdanau *et al.* 2015] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". Em: *3rd International Conference on Learning Representations (ICLR)*. Ed. por Yoshua Bengio e Yann LeCun. 2015 (citado nas pgs. 6, 24, 26, 27).

[Bengio *et al.* 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert e Jason Weston. "Curriculum learning". Em: *Proceedings of the 26th International Conference on Machine Learning*. 2009, pgs. 41–48 (citado na pg. 77).

[Bluche *et al.* 2017] Théodore Bluche, Jérôome Louradour e Ronaldo Messina. "Scan, attend and read: End-to-end handwritten paragraph recognition with MDLSTM attention". Em: *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 1. IEEE. 2017, pgs. 1050–1055 (citado nas pgs. 6, 31–33).

[Cheng *et al.* 2016] Jianpeng Cheng, Li Dong e Mirella Lapata. "Long short-term memory-networks for machine reading". Em: *arXiv preprint arXiv:1601.06733* (2016) (citado na pg. 26).

[Cho, Courville *et al.* 2015] Kyunghyun Cho, Aaron Courville e Yoshua Bengio. "Describing multimedia content using attention-based encoder-decoder networks". Em: *IEEE Transactions on Multimedia* 17.11 (2015), pgs. 1875–1886 (citado na pg. 23).

[Cho, Van Merriënboer *et al.* 2014] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau e Yoshua Bengio. "On the properties of neural machine translation: encoder-decoder approaches". Em: *arXiv preprint arXiv:1409.1259* (2014) (citado na pg. 22).

[Chorowski, Bahdanau, Cho *et al.* 2014] Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. "End-to-end continuous speech recognition using attention-based recurrent nn: first results". Em: *arXiv preprint arXiv:1412.1602* (2014) (citado na pg. 26).

[Chorowski, Bahdanau, Serdyuk *et al.* 2015] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho e Yoshua Bengio. "Attention-based models for speech recognition". Em: *arXiv preprint arXiv:1506.07503* (2015) (citado na pg. 33).

[Chung *et al.* 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho e Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". Em: *arXiv preprint arXiv:1412.3555* (2014) (citado nas pgs. 21–23).

[Frinken e Bunke 2014] Volkmar Frinken e Horst Bunke. *Continuous Handwritten Script Recognition.* 2014 (citado na pg. 28).

[Fukushima 1980] Kunihiko Fukushima. "Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". Em: *Biological Cybernetics* 36.4 (1980), pgs. 193–202 (citado na pg. 18).

[Graves 2013] Alex Graves. "Generating sequences with recurrent neural networks". Em: *arXiv preprint arXiv:1308.0850* (2013) (citado nas pgs. 21, 27).

[Graves, Fernández *et al.* 2006] Alex Graves, Santiago Fernández, Faustino Gomez e Jürgen Schmidhuber. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks". Em: *Proceedings of the 23rd International Conference on Machine Learning*. ACM. 2006, pgs. 369–376 (citado na pg. 30).

[Graves, Mohamed *et al.* 2013] Alex Graves, Abdel-rahman Mohamed e Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". Em: *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pgs. 6645–6649 (citado na pg. 27).

[Graves e Schmidhuber 2009] Alex Graves e Jürgen Schmidhuber. Em: *Advances in neural information processing systems*. 2009, pgs. 545–552 (citado nas pgs. 28, 29).

[Graves, Wayne *et al.* 2014] Alex Graves, Greg Wayne e Ivo Danihelka. "Neural turing machines". Em: *arXiv preprint arXiv:1410.5401* (2014) (citado na pg. 26).

[Hinton *et al.* 2006] Geoffrey E Hinton, Simon Osindero e Yee-Whye Teh. "A fast learning algorithm for deep belief nets". Em: *Neural computation* 18.7 (2006), pgs. 1527–1554 (citado na pg. 18).

[Hochreiter e Schmidhuber 1997] Sepp Hochreiter e Jürgen Schmidhuber. "Long short-term memory". Em: *Neural computation* 9.8 (1997), pgs. 1735–1780 (citado na pg. 21).

[Hubel e Wiesel 1959] David H Hubel e Torsten N Wiesel. "Receptive fields of single neurons in the cat's striate cortex". Em: *The Journal of Physiology* 148.3 (1959), pgs. 574–591 (citado na pg. 18).

[Kang, Riba *et al.* 2020] Lei Kang, Pau Riba, Marçal Rusiñol, Alicia Fornés e Mauricio Villegas. "Pay attention to what you read: non-recurrent handwritten text-line recognition". Em: *arXiv preprint arXiv:2005.13044* (2020) (citado na pg. 28).

[Kang, Toledo *et al.* 2018] Lei Kang, J Ignacio Toledo *et al.* "Convolve, attend and spell: an attention-based sequence-to-sequence model for handwritten word recognition". Em: *German Conference on Pattern Recognition*. Springer. 2018, pgs. 459–472 (citado nas pgs. 6, 33, 41, 65).

[Kingma e Ba 2014] Diederik P Kingma e Jimmy Ba. "Adam: a method for stochastic optimization". Em: *arXiv preprint arXiv:1412.6980* (2014) (citado na pg. 44).

[Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks". Em: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pgs. 1097–1105 (citado na pg. 19).

[Lamb *et al.* 2016] Alex M Lamb *et al.* "Professor forcing: a new algorithm for training recurrent networks". Em: *Advances in neural information processing systems* 29 (2016), pgs. 4601–4609 (citado na pg. 44).

[LeCun e Bengio 1998] Yann LeCun e Yoshua Bengio. "Convolutional networks for images, speech, and time series". Em: *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, pgs. 255–258. isbn: 0262511029 (citado na pg. 19).

[LeCun, Boser *et al.* 1989] Yann LeCun, Bernhard Boser *et al.* "Backpropagation applied to handwritten zip code recognition". Em: *Neural Computation* 1.4 (1989), pgs. 541–551 (citado nas pgs. 4, 18).

[Luong *et al.* 2015] Minh-Thang Luong, Hieu Pham e Christopher D Manning. "Effective approaches to attention-based neural machine translation". Em: *arXiv preprint arXiv:1508.04025* (2015) (citado na pg. 26).

[Mcculloch e Pitts 1943] Warren S. Mcculloch e Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". Em: *Journal of Symbolic Logic* 9.2 (1943), pgs. 49–50. doi: 10.2307/2268029 (citado na pg. 15).

[Minsky e Papert 1969] Marvin Minsky e Seymour Papert. "Perceptron: an introduction to computational geometry". Em: *The MIT Press, Cambridge, expanded edition* 19.88 (1969), pg. 2 (citado na pg. 16).

[Rosenblatt 1958] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." Em: *Psychological review* 65.6 (1958), pg. 386 (citado na pg. 16).

[David E. Rumelhart *et al.* 1986] David E. Rumelhart, James L. McClelland e CORPORATE PDP Research Group, ed. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986. isbn: 0-262-68053-X (citado na pg. 16).

[David E Rumelhart *et al.* 1988] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams *et al.* "Learning representations by back-propagating errors". Em: *Cognitive Modeling* 5.3 (1988), pg. 1 (citado na pg. 16).

[Sutskever *et al.* 2014] Ilya Sutskever, Oriol Vinyals e Quoc V. Le. "Sequence to sequence learning with neural networks". Em: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pgs. 3104–3112 (citado na pg. 23).

[Szegedy *et al.* 2016] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens e Zbigniew Wojna. "Rethinking the inception architecture for computer vision". Em: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pgs. 2818–2826 (citado na pg. 91).

[Vaswani *et al.* 2017] Ashish Vaswani *et al.* "Attention is all you need". Em: *Advances in neural information processing systems*. 2017, pgs. 5998–6008 (citado nas pgs. 26, 28).

[Xu *et al.* 2015] Kelvin Xu *et al.* "Show, attend and tell: neural image caption generation with visual attention". Em: *International Conference on Machine Learning*. 2015, pgs. 2048–2057 (citado nas pgs. 7, 26–28, 35, 41, 91).

[You *et al.* 2016] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang e Jiebo Luo. "Image captioning with semantic attention". Em: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pgs. 4651–4659 (citado nas pgs. 6, 27).

[A. Zhang *et al.* 2020] Aston Zhang, Zachary C. Lipton, Mu Li e Alexander J. Smola. *Dive into Deep Learning*. https://d2l.ai. 2020 (citado na pg. 18).

[J. Zhang *et al.* 2017] Jianshu Zhang *et al.* "Watch, attend and parse: an end-to-end neural network based approach to handwritten mathematical expression recognition". Em: *Pattern Recognition* 71 (2017), pgs. 196–206 (citado nas pgs. 6, 27).