UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

MODELO DE UMA REDE NEURAL PROFUNDA PARA RECONHECIMENTO DE TEXTO MANUSCRITO *OFFLINE*

CAÍQUE REINHOLD

BLUMENAU
2016

CAÍQUE REINHOLD

MODELO DE UMA REDE NEURAL PROFUNDA PARA RECONHECIMENTO DE TEXTO MANUSCRITO *OFFLINE*

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof.ª. Luciana Pereira de Araújo, Mestra - Orientadora

BLUMENAU
2016

MODELO DE UMA REDE NEURAL PROFUNDA PARA RECONHECIMENTO DE TEXTO MANUSCRITO *OFFLINE*

Por

CAÍQUE REINHOLD

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof.ª. Luciana Pereira de Araújo, Mestra – Orientadora, FURB
Membro:	Prof. Daniel Theisges dos Santos, Mestre – FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB

Blumenau, 05 de Julho de 2016

Dedico este trabalho à minha família e aos meus amigos próximos pelo apoio e motivação oferecidos para a conclusão deste trabalho.

“We juggle imaginary data with algorithms in our head with the intent of making computers dance”

K Lars Lohn

RESUMO

Este trabalho apresenta a descrição de um modelo para reconhecimento de texto manuscrito *off-line* assim como a implementação de um protótipo para validação deste modelo. O reconhecimento de texto manuscrito off-line é uma técnica útil que pode ser aplicada em diferentes tarefas, como a digitalização de documentos históricos. Entretanto, o reconhecimento de texto manuscrito sem restrições é ainda hoje um problema bastante desafiador. O modelo apresentado utiliza um híbrido de redes neurais convolucionais e recorrentes que classificam texto manuscrito utilizando os pixels da imagem como entrada através de técnicas do aprendizado profundo. Para o treinamento do modelo é utilizada uma camada de *Connectionist Temporal Classification*, que emprega uma função de custo que é capaz de alinhar uma sequência de caracteres a sequência de entrada. Um protótipo do modelo é implementado e treinado utilizando a base de dados de imagens de texto manuscrito IAM. Os resultados obtidos foram satisfatórios e demonstram que o modelo é uma alternativa válida para o reconhecimento de texto manuscrito *off-line*, se mostrando robusto o suficiente para aprender a classificar um padrão com uma grande variabilidade como o texto manuscrito.

Palavras-chave: Aprendizado profundo. Redes neurais convolucionais. Redes neurais Recorrentes. Reconhecimento de texto manuscrito.

ABSTRACT

This work presents a model for off-line handwritten text recognition and the implementation of a prototype that evaluates this model. Recognition of offline handwritten text is a useful technique that can be applied to different tasks, such as digitalization of historical documents. However, the unconstrained handwritten text recognition is still considered a very challenging problem. The model presented is composed by an hybrid of convolutional and recurrent neural networks that recognize handwritten text using the image pixels as input features based on deep learning techniques. To train the model a Connectionist Temporal Classification layer was used, wich employs a cost function to align the sequence of label characters to the input sequence. A prototype of this model is implemented and trained using the IAM handwritten text database. The results obtained were satisfactory and show that the model is a valid choice for recognition of off-line handwritten text, beeing robust enough to be able to classify a pattern with so many variability such as handwritten text.

Key-words: Deep learning. Convolutional neural networks. Recurrent neural networks. Handwritten text recognition.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASCII – *American Standard Code for Information Interchange*

BLSTM – *Biderectional Long Short-Term Memory*

BPTT – *Backpropagation Trhough Time*

CER – *Character Error Rate*

CTC – *Connectionist Temporal Classification*

GPU – *Graphical Processing Unit*

HMM – *Hidden Markov Models*

HOG – *Histogram of Oriented Gradient*

LSTM – *Long Short-Term Memory*

MLP – *Multilayer Perceptron*

PNG – *Portable Network Graphics*

RF – Requisitos Funcionais

RNA – Rede Neural Artificial

RNC – Redes Neurais Convolucionais

RNF – Requisitos Não Funcionais

RNR – Redes Neurais Recorrentes

RNRMD – Redes Neurais Recorrentes Multidimensionais

UC – Caso de Uso

UML – *Unified Modeling Language*

WER – *Word Error Rate*

SUMÁRIO

1 INTRODUÇÃO

Apesar da era digital oferecer diversos métodos eficazes para se armazenar informação, a escrita manual ainda é utilizada devido à conveniência do papel e caneta (PLAMONDON; SRIHARI, 2000). Entretanto, como mostra Hughes (2004), o armazenamento de informações em meio digital traz diversos benefícios. Segundo ele, alguns dos benefícios são: a maior durabilidade da informação, o menor espaço para armazenamento, agilidade na busca, criação de cópias sem perda de informação, entre outros. Desta forma, o estudo de meios para o reconhecimento de texto manuscrito torna-se atrativo.

O reconhecimento de texto manuscrito é a transição para o formato digital de informações manuscritas (VINCIARELLI, 2002). Este se divide em duas áreas, *online* e *off-line*, onde na primeira está disponível a trajetória da caneta na superfície e na segunda área estão disponíveis imagens do texto manuscrito. Este assunto vem sendo estudado desde a década de 70 e já possui ferramentas sendo utilizadas em aplicações comerciais (VINCIARELLI, 2002). Entretanto, o desenvolvimento de um sistema confiável e de propósito geral ainda é considerado um problema em aberto (BEZERRA; ZANCHETTIN; ANDRADE, 2012).

A complexidade do reconhecimento de texto manuscrito em imagens vem da variação entre os padrões de caracteres, visto que cada pessoa possui um estilo de escrita diferente. Este estilo pessoal pode ainda sofrer alterações devido ao material utilizado para escrita, o estado emocional da pessoa, assim como o espaço e o tempo disponíveis para a escrita (BEZERRA; ZANCHETTIN; ANDRADE, 2012). Outras dificuldades provem da dificuldade em segmentar as linhas de texto devido às variações no espaço entre linhas, linhas de base inconsistentes ou sobreposição de traços de linhas diferentes (ALAEI; PAL; NAGABHUSHAN, 2011).

Recentes estudos de neurociência mostram que o neocortex, que é associado a várias habilidades cognitivas, não explicitamente pré-processa os sinais sensoriais. Ele na verdade deixa que se propaguem por uma hierarquia complexa de módulos que ao longo do tempo aprendem a representar observações baseadas nas regularidades que estas apresentam. Esta descoberta foi um dos grandes motivadores da emergência do Aprendizado Profundo (AREL; ROSE; KARNOWSKI, 2010).

O Aprendizado Profundo, do inglês *Deep Learning*, foca em modelos computacionais para representação de informações que exibam comportamentos semelhantes ao do nosso neocortex (AREL; ROSE; KARNOWKI, 2010). Ou seja, modelos que são capazes de

aprender a representar informações de alto nível, como classes de objetos, a partir de dados sensoriais, como os pixels de uma imagem.

Diante do exposto, este trabalho apresenta um modelo de rede neural profunda para o reconhecimento de texto manuscrito *off-line.* Além disso, é apresentada a implementação de um protótipo baseado neste modelo para transcrição de imagens de texto manuscrito em texto digital.

1.1 OBJETIVOS

O objetivo deste trabalho é especificar um modelo de rede neural capaz de reconhecer texto manuscrito em imagens digitais.

Os objetivos específicos do trabalho são:

a) reconhecer texto manuscrito em imagens digitais;
b) utilizar aprendizado profundo para reconhecimento de padrões visuais;
c) desenvolver um protótipo para validar o modelo apresentado.

1.2 ESTRUTURA

A estrutura deste trabalho é dividida em quatro capítulos, sendo que no primeiro foi apresentada uma introdução ao tema do trabalho, os objetivos do mesmo e sua estrutura.

O segundo capítulo contém a fundamentação teórica envolvida no reconhecimento de texto manuscrito e Aprendizado Profundo, que são de suma importância para o entendimento do trabalho. O capítulo apresenta também alguns trabalhos correlatos.

O terceiro capítulo traz o desenvolvimento do trabalho. Nele são expostos o modelo proposto, sua implementação, assim como técnicas e ferramentas utilizadas. No capítulo ainda são discutidos detalhes relacionados à base de dados utilizada para o treinamento do modelo e, por fim, são apresentadas as discussões e resultados obtidos até o termino do trabalho.

O quarto e último capítulo aborda as principais conclusões e contribuições, assim como sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada a definição de reconhecimento de texto seguida de uma breve revisão histórica da sua evolução na seção 2.1. Em seguida, na seção 2.2, são apresentados os conceitos de redes neurais, assim como a estrutura de redes convolucionais na seção 2.2.1, e as redes recorrentes na seção 2.2.2. O conceito de Aprendizado Profundo é discutido na seção 2.3. Na seção 2.4 é feita uma introdução sobre classificação de sequencias. Por fim na seção 2.5 são apresentados os trabalhos correlatos.

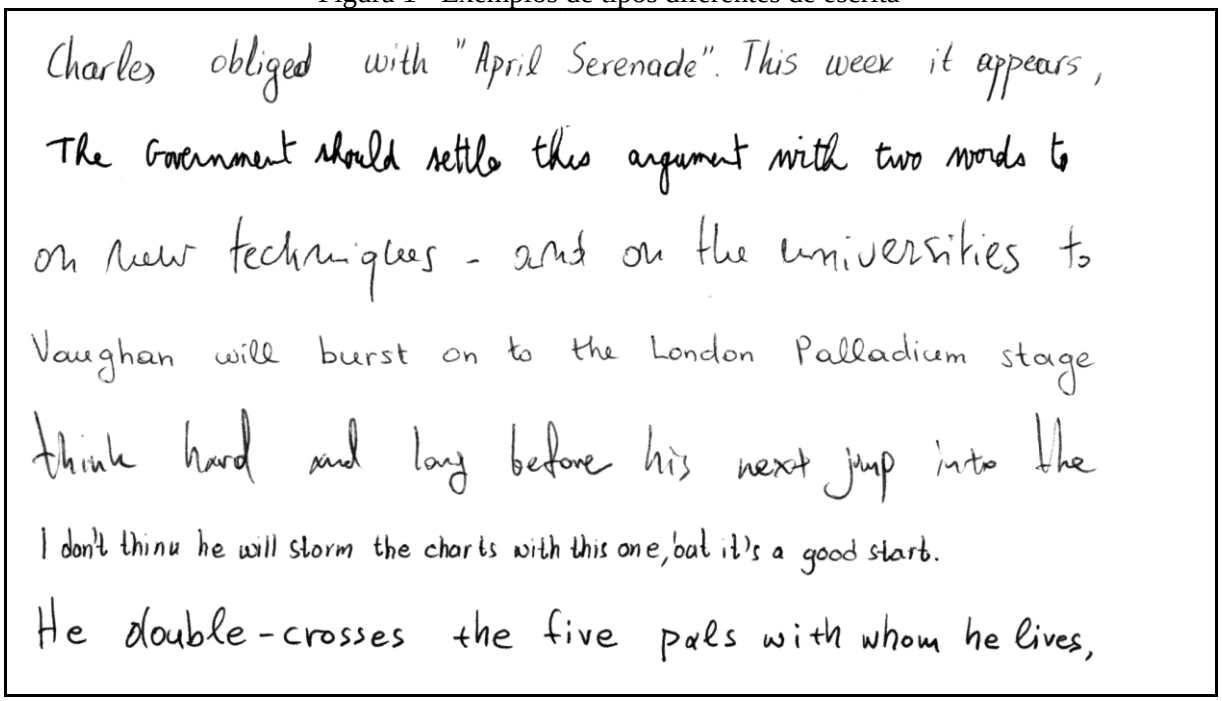
2.1 RECONHECIMENTO DE TEXTO MANUSCRITO

A escrita manual consiste de marcas gráficas feitas à mão em uma superfície com o propósito de se comunicar. Este propósito é alcançado devido à relação convencionada da marca com a linguagem falada (PLAMODON; SRIHARI, 2000). O reconhecimento de texto manuscrito é a tarefa de transformar a linguagem representada de forma gráfica em uma forma simbólica, como a representação *American Standard Code for Information Interchange* (ASCII) (PLAMODON; SRIHARI, 2000).

O reconhecimento de texto manuscrito se divide em duas áreas: *online* e *off-line*. No *online* uma série temporal de coordenadas da trajetória da ponta da caneta é usada para o reconhecimento dos dígitos. O caso *online* é normalmente utilizado em superfícies digitais. Enquanto que no caso *off-line* apenas a imagem do texto está disponível, como no caso de documentos digitalizados (SEILER; SCHENKEL; EGGIMANNN, 1996). As taxas de reconhecimento no caso *off-line* tendem a ser menores, devido a maior dificuldade em se extrair suas características relevantes (PLAMONDON; SRIHARI, 2000).

Apesar de mais de 30 anos de pesquisas sobre o assunto, o reconhecimento de texto manuscrito continua ainda hoje sendo um problema em aberto (GRAVES et al., 2009). Os motivos são os diversos desafios que se precisa superar neste tipo de problema, como a grande variedade de estilos de escrita, visto que cada indivíduo possui um estilo único de escrita. Outro problema se deve a dificuldade na segmentação, uma vez que traços de escrita podem cruzar outras palavras e espaços entre os componentes nem sempre segmentam palavras ou caracteres de maneira correta. Como palavras inteiras devem ser reconhecidas pela dificuldade de segmentação, o vocabulário extenso de palavras que devem ser reconhecidas apresenta mais uma dificuldade (BERTOLAMI; BUNKE, 2008). Exemplos das dificuldades descritas podem ser vistos na Figura 1, como estilos diferentes de escrita e traços sobrepostos.

Figura 1 - Exemplos de tipos diferentes de escrita



Fonte: Bertolami e Bunke (2008, p. 10).

Apesar de existirem aplicações comerciais que utilizam o reconhecimento de texto manuscrito *off-line* com taxas de acertos suficientemente boas, ele é usado apenas em aplicações muito restritas ou específicas. Alguns exemplos de aplicações comerciais que fazem uso deste tipo de técnica são o reconhecimento automático de endereços em correspondências e leitura automática de cheques bancários (PLAMONDON; SRIHARI, 2000).

2.2 REDES NEURAIS

Redes neurais são descritas por McCulloch e Pitts (1943) desde os primórdios da sua concepção como modelos matemáticos da capacidade de processamento de informações do cérebro. A estrutura básica de uma Rede Neural Artificial (RNA) consiste de várias unidades de processamento ligadas entre si por conexões valoradas, estes são análogos aos neurônios e sinapses. O funcionamento da rede se dá passando um vetor de valores aos neurônios de entrada, que espalham suas ativações através das conexões valoradas (SCHMIDHUBER, 2015).

O objetivo de RNAs é modelar funções matemáticas complexas, de forma que para cada entrada se obtenha um resultado esperado (GRAVES, 2008). O uso mais comum para RNAs é a reconhecimento de padrões. O reconhecimento de padrões consiste em classificar a representação de um objeto em um conjunto de categorias (LECUN; BENGIO, 1995).

Para seu uso em classificação precisam passar por uma etapa de treinamento (FAUSETT, 1994). No caso de redes neurais treinadas de forma supervisionada, este treinamento é realizado passando a rede exemplos de entradas e seus respectivos resultados esperados. O treinamento de uma RNA é realizado otimizando seus pesos com os gradientes de erro de alguma função de custo. Para o cálculo destes gradientes de erro se utiliza o algoritmo de *Backpropagation*, que calcula os gradientes partindo da última para a primeira camada, levando em consideração os pesos das ligações para propagar os erros (FAUSETT, 1994). As RNAs podem ser treinadas também de forma não supervisionada. Neste caso são apresentados a rede apenas os exemplos de entrada, sem qualquer informação sobre o resultado esperado, fazendo com a rede encontre os padrões que forem relevantes para os dados de entrada (BENGIO, 2009).

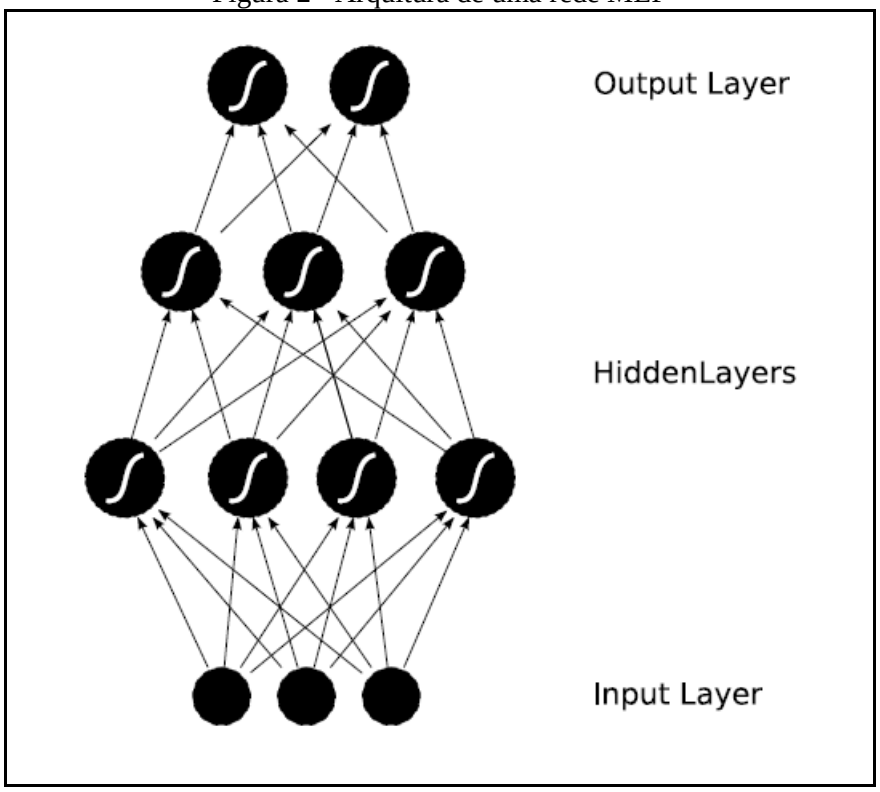
Vários tipos de redes de redes neurais foram propostos ao longo dos anos, com diversas propriedades e aplicações diferentes. Uma distinção importante é em relação às RNAs cujas conexões formam ciclos e as que são acíclicas. RNAs que formam ciclos são normalmente chamadas de Redes Neurais Recorrentes (RNR), já as que não formam ciclos são conhecidas como redes *feedforward* (GRAVES, 2008). Neste trabalho são destacados três tipos de redes neurais, as Redes Neurais *Multilayer Perceptron* (MLP), Redes Neurais Convolucionais (RNC) e as RNRs, que são apresentadas, respectivamente, nas seções 2.2.1, 2.2.2 e 2.2.3.

2.2.1 Redes Neurais *Multilayer Perceptron*

As redes MLP são o tipo mais difundido e utilizado de rede neural. Isto se deve por serem um modelo relativamente simples e que é capaz de modelar qualquer tipo de problema. Uma rede MLP com uma camada oculta e um número suficiente de neurônios é considerada um aproximador universal, pois é capaz de modelar qualquer função com um nível arbitrário de aproximação (GOODFELLOW et al., 2013).

Uma MLP é dividida em camadas que recebem entradas de uma camada anterior, calculam sua ativação e propagam para a próxima camada. Os neurônios de cada camada são ligados a todos os neurônios da camada seguinte através de conexões valoradas. Cada neurônio soma os valores recebidos dos neurônios da camada anterior. Esta soma é transformada utilizando uma função de ativação e essa ativação é propagada para os neurônios da próxima camada, até que a última camada seja atingida (GRAVES, 2008). A arquitetura de uma MLP pode ser vista na Figura 2.

Figura 2 - Arquitura de uma rede MLP



Fonte: Graves (2008).

2.2.2 Redes Neurais Convolucionais

Em seu trabalho sobre o córtex visual de gatos, Hubel e Wiesel (1968) mostram que o córtex visual é uma disposição complexa de células que são sensíveis a pequenas sub-regiões do campo visual, chamadas campos receptivos. Estas células agem como filtros sobre as entradas dos campos receptivos e são adaptadas para explorar a forte correlação espacial local presente em imagens naturais. Baseado nisso foram criadas as Redes Neurais Convolucionais, que são um tipo de RNA *feedforward* inspirada no córtex visual dos animais.

As RNCs combinam três ideias principais em sua arquitetura que garantem que sejam muito mais robustas do que outros tipos de RNAs em relação a deslocamento, escala e distorção dos dados de entrada. Essas ideias são: campos receptivos locais, parâmetros compartilhados e amostragem (LECUN et al., 1998).

Campos receptivos locais consistem em fazer com que os neurônios de uma camada da RNC sejam ativados apenas por um pequeno conjunto de ativações da camada anterior. Isso faz com que seja possível aos neurônios reconhecer características visuais básicas, como linhas ou cantos, que são combinadas em camadas subsequentes para reconhecer características de mais alto nível (LECUN et al., 1998).

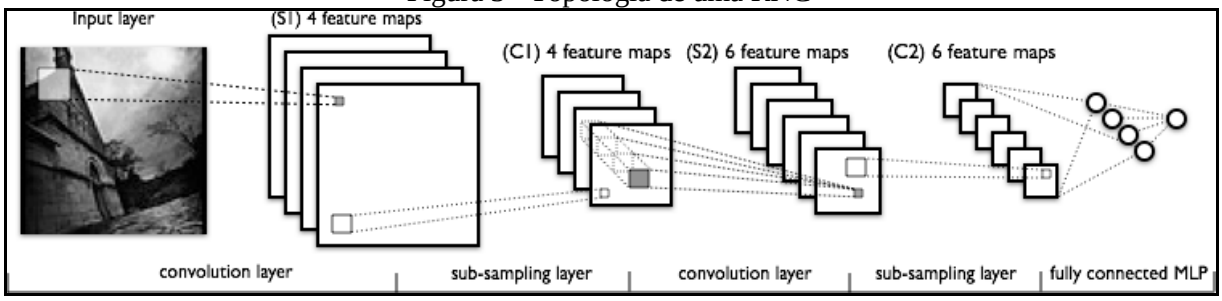
As ativações dos campos receptivos locais se dão através de filtros de convolução, cujos parâmetros são aprendidos pela rede. Estes filtros são replicados por toda a imagem

fazendo com que todos os campos receptivos compartilhem os mesmos parâmetros. Esse compartilhamento de parâmetros faz que com que as características visuais sejam detectadas independentemente da sua posição na imagem. Além disso, o compartilhamento faz com que o número de parâmetros livres que a rede deve aprender seja muito menor se comparado a RNAs comuns, o que torna o aprendizado mais eficiente (LISA LAB, 2016).

A operação de amostragem consiste em particionar as ativações da imagem em pequenos retângulos e criar uma nova representação com a maior ativação de cada retângulo. A amostragem tem como objetivo criar uma representação mais compacta das características mantendo as informações mais relevantes e descartando as irrelevantes. Como apenas as características relevantes são mantidas a representação se torna menos suscetível a transformações ou distorções (BOUREAU; PONCE; LECUN, 2010).

A estrutura padrão de uma RNC, como pode ser vista na Figura 3, é composta por camadas de convolução, com vários filtros de convolução para que várias características sejam extraídas em campo receptivo, intercaladas com camadas de amostragem. Como o tamanho da imagem diminui a cada camada devido ao processo de amostragem, o número de filtros convolucionais geralmente aumenta a cada camada para que não se percam características importantes da imagem (LISA Lab., 2016).

Figura 3 - Topologia de uma RNC



Fonte: LISA Lab. (2016)

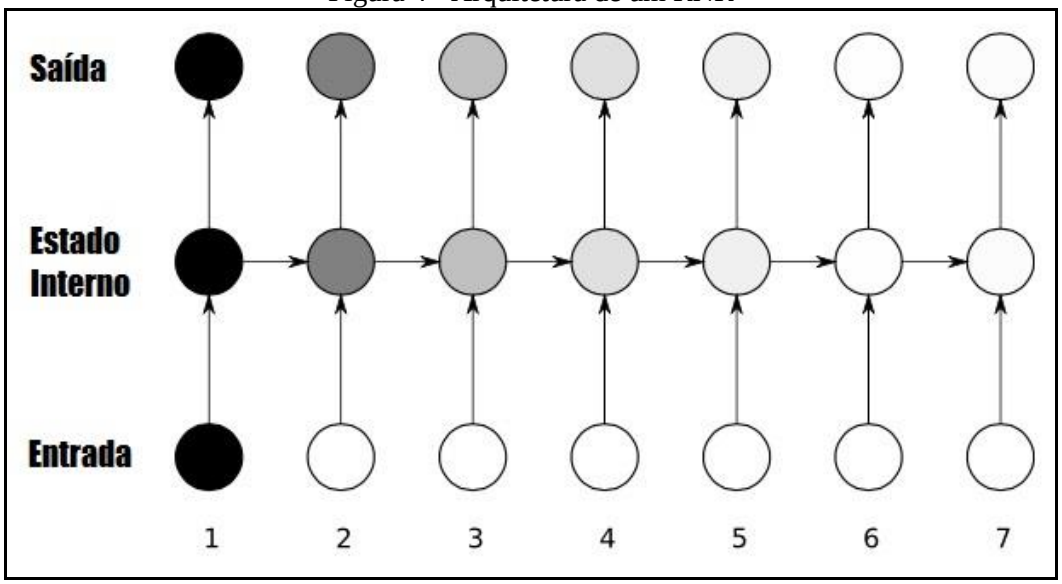
2.2.3 Redes Neurais Recorrentes

Como visto em Graves (2008), RNRs possibilitam que as ligações em seus neurônios formem ciclos, conhecidos como conexões recorrentes. Estas conexões recorrentes permitem que o estado interno da rede crie uma espécie de “memória” de entradas anteriores, que pode ser usada para influenciar no resultado da rede.

A arquitetura de uma RNR, como visto na Figura 4, é formada por uma camada de entrada, uma camada para o estado interno e uma camada de saída. A cada ponto de uma sequência, as ativações da camada de entrada são somadas as ativações do ponto anterior da sequência que estão armazenados no estado interno da rede, formando um novo conjunto de

ativações. O estado interno é então atualizado com esse novo conjunto de ativações e essas ativações são repassadas a camada de saída (TOMÁ et al., 2010).

Figura 4 - Arquitetura de um RNR

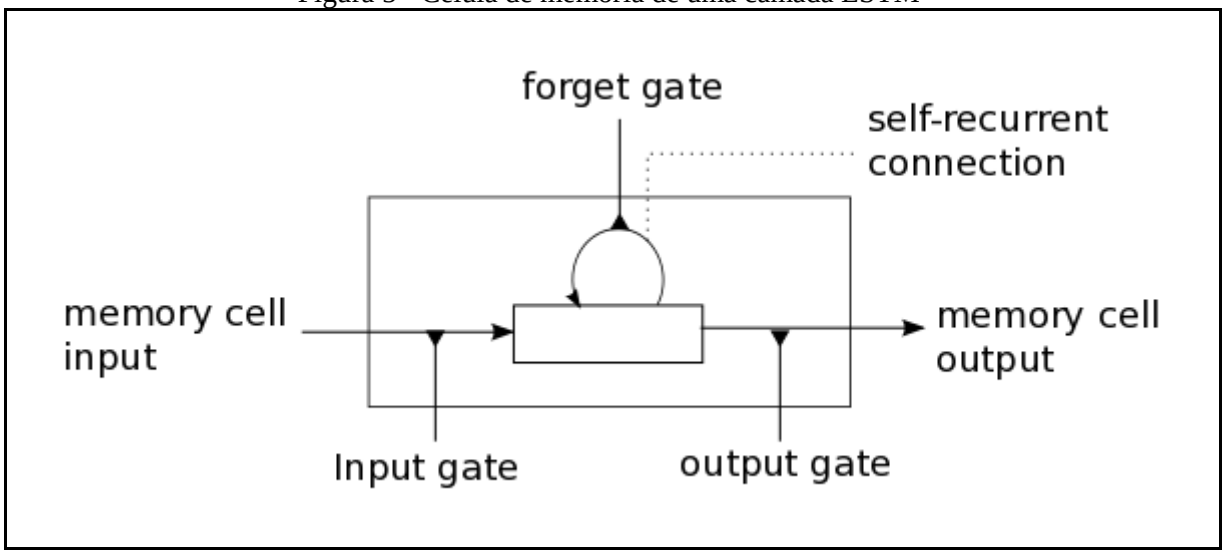


Fonte: adaptado de Graves (2008).

Uma limitação de RNRs comuns, como mostra Hochreiter et al. (2001), é que o aprendizado de dependências de curto prazo tende a dominar os valores das conexões da rede, fazendo com que a rede esqueça dependências de longo prazo. O resultado é que a rede é capaz de guardar apenas uma curta janela de contexto das entradas.

Uma solução para o problema da falta de dependências de longo prazo foi proposta por Hochreiter e Schmidhuber (1997) que consiste em uma nova arquitetura de RNR denominada *Long Short-Term Memory* (LSTM). A arquitetura de uma rede LSTM se assemelha a de uma RNR comum, mas no caso da LSTM o estado interno da rede é trocado por uma estrutura chamada células de memória. Estas células de memória são controladas por três portões de ativação: o portão de entrada; o portão de saída e o portão de esquecimento. Que podem ser vistos na Figura 5. Os portões de entrada e saída filtram valores irrelevantes na entrada e saída das células de memória, enquanto o portão de esquecimento filtra valores irrelevantes no próprio estado da célula de memória (HOCHREITER; SCHMIDHUBER, 1997). Estes mecanismos possibilitam que as redes LSTM possam guardar e acessar informações de contexto durante longos períodos (GRAVES, 2008).

Figura 5 - Célula de memória de uma camada LSTM



Fonte: LISA Lab. (2016).

2.3 APRENDIZADO PROFUNDO

A abordagem mais comum em aprendizado de máquina é a de criar manualmente algoritmos para extrair características dos dados e então utilizar um algoritmo de classificação treinável para classificar as características extraídas (BENGIO; LECUN, 2007). Uma grande desvantagem deste método é que o desenvolvimento de técnicas manuais de extração de características geralmente são complexas e demandam conhecimento específico sobre o problema (BENGIO; LECUN, 2007). O motivo para isso, conforme visto em Bengio (2009), é que funções que mapeiam dados brutos, como os pixels de uma imagem, possuem muita variação e são, portanto, mais difíceis de modelar, precisando de classificadores mais profundos. Já funções que mapeiam características mais abstratas, como no caso das características extraídas manualmente, sofrem menos variação e são mais facilmente assimiladas por arquiteturas comuns de classificadores.

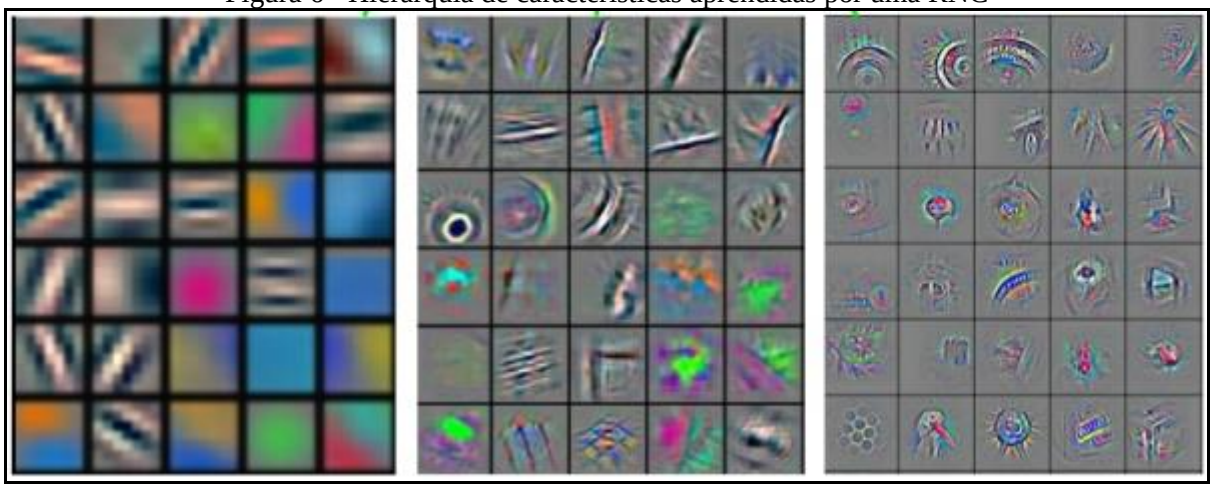
Apesar de modelos de Redes Neurais com várias camadas serem estudados desde os anos 60, o Aprendizado Profundo só obteve um avanço significativo em 2006, quando um novo método para treinar redes neurais profundas utilizando aprendizado não supervisionado foi proposto (SCHMIDHUBER, 2015). Desde então, pesquisas tanto em métodos supervisionados como não supervisionados, assim como avanços em *hardware* e o uso de processamento em *Graphical Processing Unit* (GPU), tornaram o Aprendizado Profundo viável (SCHMIDHUBER, 2015).

A dificuldade em se treinar redes neurais profundas vem de um problema conhecido como *Vanishing Gradients Problem*. Este problema faz com que o erro calculado por *Backpropagation* diminua rapidamente à medida que é propagado aos níveis mais profundos

da rede, sendo exponencial em relação ao número de camadas (HOCHREITER et al., 2001). O trabalho de Ciresan et al. (2010) mostra que, apesar de não resolver o problema de *Vanishing Gradients*, devido ao seu grande poder de processamento paralelo, o uso de GPUs torna o treinamento de redes neurais profundas viável. Através de um treinamento mais longo, acelerado pelo processamento paralelo de GPUs, é possível que os pequenos erros propagados às camadas mais profundas encontrem os parâmetros ótimos (CIRESAN et al., 2010).

O objetivo do Aprendizado Profundo pode ser definido como aprender uma hierarquia de características, no qual as características de cada nível são formadas pela composição de características de níveis inferiores (BENGIO, 2009). Utilizando algoritmos que possam aprender essas características em todos os níveis da hierarquia, possibilita um sistema que possa aprender funções complexas desde os dados brutos, sem a necessidade de extrair estas características manualmente (BENGIO, 2009). A Figura 6 mostra características aprendidas por três camadas de uma RNC, um dos tipos de arquiteturas profundas. Nela é possível ver como a cada camada as características tem um nível de abstração cada vez maior.

Figura 6 - Hierarquia de características aprendidas por uma RNC



Fonte: adaptado de Zeiler e Fergus (2014).

2.4 CLASSIFICAÇÃO SEQUÊNCIAL

No aprendizado de máquina, classificação sequencial engloba todas as tarefas na qual uma sequência de dados é transcrita em uma sequência de classes (GRAVES, 2008). Exemplos conhecidos são o reconhecimento de fala ou de texto manuscrito. O que difere os problemas de classificação sequencial dos problemas de classificação comuns é que os dados e suas respectivas classes não são independentes e igualmente distribuídos. Isso significa que os dados e as classes são fortemente correlacionados, fazendo com que os algoritmos, em alguns casos, tenham que determinar, além da classe, a localização desta classe na sequência de dados (GRAVES, 2008).

Estes problemas de classificação sequencial podem ainda ser divididos em três categorias, dependendo do tipo de sequencias de classes resultantes, que são: classificação de sequência, classificação de segmento e classificação temporal. Sendo que cada categoria afeta a escolha de técnicas e algoritmos utilizados para resolver o problema (GRAVES, 2008).

Na classificação de sequência, toda a sequência de dados deve receber apenas uma classe. Um exemplo seria o reconhecimento de um caractere manuscrito individual. Caso as sequências de entrada possuam um tamanho fixo, ou possam ser preenchidas para terem um tamanho fixo, a sequência pode ser concatenada em uma única entrada, tornando o problema uma classificação comum de padrão (GRAVES, 2008).

Um problema de classificação sequencial é considerado da categoria de classificação de segmento quando várias classes são necessárias para uma única sequência de dados e o alinhamento entre a sequência de entrada e a sequência de classes são conhecidos de antemão. Um exemplo seria o reconhecimento de uma palavra falada que esteja separada por fonemas. Um dos recursos principais para o um bom resultado nesse tipo de problema é o uso de contexto, ou seja, levar em consideração toda ou parte da série de entradas na classificação de cada segmento (GRAVES, 2008).

A classificação temporal é o caso mais geral das três categorias, no qual nada pode ser presumido sobre a sequência de classes, a não ser que ela tem tamanho igual ou menor do que a sequência de entradas. Neste caso, o alinhamento entre as entradas e as classes é desconhecido, fazendo com que o algoritmo deva decidir a posição da classe na sequência de entrada. Um exemplo de classificação temporal é o reconhecimento de linhas de texto manuscrito (GRAVES, 2008).

Como visto em Graves et al. (2006), a capacidade de incorporar contexto de uma sequência de entradas das RNRs a torna uma escolha atrativa para problemas de classificação sequencial. Entretanto, RNRs só podem ser treinadas para fazer uma série de classificações independentes. Com isso, o uso de RNRs por si só não pode ser aplicável a problemas de classificação temporal, visto que seria necessário que os dados de treinamento fossem pré-segmentados. Além disso, seria necessário também uma etapa de processamento adicional no resultado da rede para formar a sequência de classes final.

Para tornar o treinamento de RNRs viável para problemas de classificação temporal, Graves et al. (2006) apresenta a *Connectionist Temporal Classification* (CTC), uma camada acoplada à saída de RNRs que faz com que seu uso em problemas de classificação temporal seja possível. A CTC, resumidamente, transforma os resultados da RNR em uma distribuição

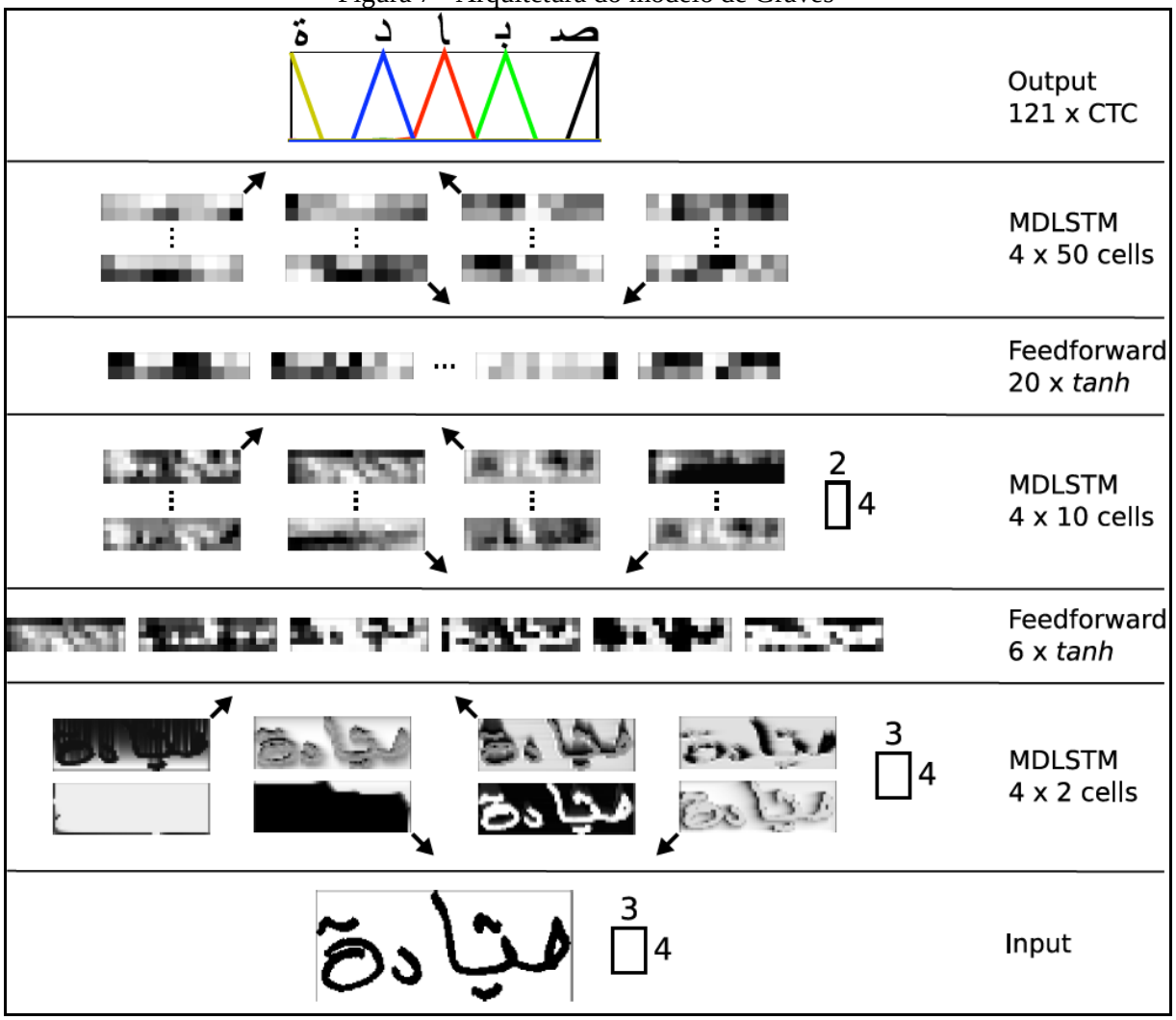
da probabilidade de cada classe aparecer na sequência de entrada. A RNR pode então ser usada para classificação selecionando a sequência de classes mais provável para cada entrada.

2.5 TRABALHOS CORRELATOS

Nesta seção são apresentados alguns trabalhos voltados ao reconhecimento de texto manuscrito *off-line*. Foram selecionados três trabalhos que propõem diferentes modelos de classificadores para texto manuscrito baseados em redes neurais. Sendo esses os trabalhos de Graves (2012), Su et al (2015) e Chen, Yan e Huo (2015).

Em seu trabalho, Graves (2012) desenvolve um protótipo para o reconhecimento de texto manuscrito arábico. O modelo utiliza várias camadas de Redes Neurais Recorrentes Multidimensionais (RNRMD). RNRMDs são uma especificação de RNRs que guardam contexto da sequência de entrada em mais de uma dimensão. No caso do trabalho é utilizado contexto em duas dimensões, tornando possível seu uso com imagens. Estas camadas são intercaladas por camadas *Multilayer Perceptron* a fim de reduzir sua dimensão vertical, de modo que na camada final a matriz de entrada seja transformada em vetor para ser processado pela camada CTC. O modelo pode ser visualizado na Figura 7.

Figura 7 - Arquitetura do modelo de Graves



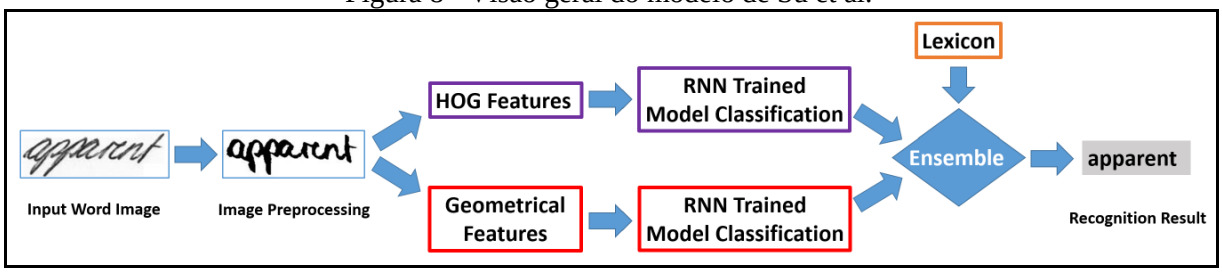
Fonte: Graves (2008).

Inicialmente a imagem é dividida em blocos de 4x3 pixels, então quatro RNRMDs na primeira camada escaneiam os blocos um a um. Cada RNRMD escaneia os blocos em uma direção, de cima para baixo e da esquerda para a direita, de baixo para cima e da esquerda para a direita e assim por diante. Este método é utilizado para que a camada tenha informações sobre o contexto da imagem em todas as direções. A saída desta camada passa então por uma camada *Multilayer Perceptron* para que sua dimensão vertical seja reduzida mantendo apenas as características mais importantes. Os resultados desta primeira camada são passados para mais duas camadas semelhantes, até que sua dimensão vertical seja reduzida a um para poder então ser processada pela camada CTC. A camada CTC no final da rede rotula as características recebidas em caracteres, e a sequência de caracteres reconhecidos é então pesquisada em um dicionário para o reconhecimento da palavra.

No trabalho de Su et al. (2015), é apresentado um modelo que utiliza extração manual de características e usa RNRs como classificador. Cada imagem é previamente pré-processada

para correção de inclinações verticais ou horizontais e normalizada. Depois de pré-processada as características são extraídas de uma pequena janela da imagem, que vai deslizando da esquerda para a direita. Dois grupos de características são extraídos, o primeiro é um conjunto de nove características geométricas extraídas dos pixels, o segundo é extraído através do algoritmo *Histogram of Oriented Gradient* (HOG). O resultado desta etapa são duas sequências de vetores de características. Estas duas sequências de vetores são então passadas para duas RNRs baseadas no modelo *Biderectional Long Short-Term Memory* (BLSTM). BLSTMs são tipos especiais de LSTM onde que utiliza tanto o contexto anterior quanto o posterior do item da sequência sendo visto no momento. O resultado é dado pela camada CTC de cada RNR. Os resultados das duas RNRs são finalmente combinados formando uma sequência de probabilidades do aparecimento de um caractere. Por fim, um algoritmo procura em um dicionário a palavra mais provável para esta sequência. Uma visão geral do modelo de Su et al (2015) é mostrada na Figura 8.

Figura 8 - Visão geral do modelo de Su et al.



Fonte: Su et al. (2015).

Chen, Yan e Huo (2015) apresentam em seu trabalho um modelo hibrido que mistura RNRs com *Hidden Markov Models* (HMM). As imagens passam por etapa de pré-processamento onde tem sua altura normalizada para 60 pixels. A imagem é dividida em janelas que deslizam da esquerda para a direita a cada 3 pixels. Cada janela, tem sua dimensionalidade reduzida a vetores com apenas 50 características utilizando o algoritmo de *Principal Component Analysis*. Cada sequência de vetores é então passada ao modelo composto por 5 camadas de RNRs do tipo *Long Short-Term Memory* (LSTM). Devido à complexidade computacional do treinamento de 5 camadas de RNR, Chen, Yan e Huo (2015) propõem uma versão aperfeiçoada do algoritmo de *Backpropagation Through Time* (BPTT), utilizado para o treinamento de RNRs. O algoritmo proposto, chamado de *Context Sensitive Chunk based* BPTT, torna o treinamento mais eficiente em GPUs dividindo a sequência de entrada em pequenos pedaços concatenados com informações do contexto. Por fim, as ativações das camadas LSTM passam por um modelo HMM que as classifica em uma das possíveis classes de caracteres.

3 DESENVOLVIMENTO

Neste capítulo são abordadas as etapas do desenvolvimento do modelo proposto, assim como do protótipo desenvolvido. Na seção 3.1 são enumerados os requisitos principais do protótipo. A seção 3.2 apresenta a especificação do modelo e do protótipo desenvolvido. A seção 3.3 detalha a implementação das técnicas e algoritmos utilizados para o desenvolvimento do protótipo. Por fim, na seção 3.4 são apresentados os testes de validação do protótipo e seus resultados e discussões.

3.1 REQUISITOS

Os requisitos do protótipo desenvolvido são:

a) transcrever texto manuscrito em imagens digitais (Requisito Funcional - RF);
b) possibilitar o treinamento da rede para reconhecimento de outros dicionários de palavras (RF);
c) utilizar a linguagem Python para desenvolver o protótipo (Requisito Não Funcional - RNF);
d) utilizar a biblioteca de matemática simbólica Theano para implementação e treinamento da rede (RNF).

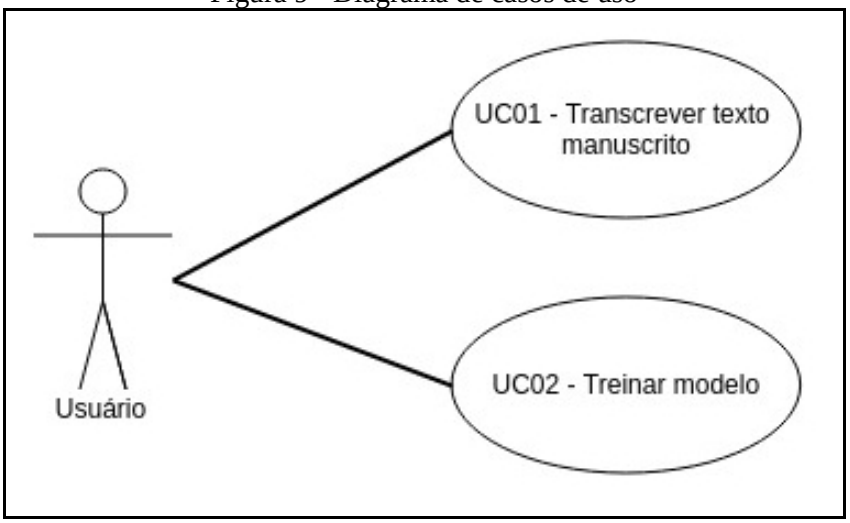
3.2 ESPECIFICAÇÃO

Nesta seção é apresentada a especificação do modelo proposto e do protótipo desenvolvido. Na seção 3.2.1 são detalhados os casos de uso (UC) utilizando a notação *Unified Modeling Language* (UML). Em seguida é apresentado a especificação do modelo proposto para reconhecimento de texto manuscrito. Por fim é apresentado um diagrama de classes em notação UML do protótipo desenvolvido.

3.2.1 Casos de uso

O modelo proposto atenderá a dois casos de uso, `transcrever texto manuscrito` e `treinar modelo`, conforme mostrado na Figura 9.

Figura 9 - Diagrama de casos de uso



O principal caso do uso, `transcrever texto manuscrito`, é detalhado no Quadro 1. Em seguida, no Quadro 2 é detalhado o caso de uso `treinar modelo`.

Quadro 1 - Detalhamento do UC01

Caso de uso	Transcrever texto manuscrito
Descrição	Este caso de uso descreve os procedimentos para a transcrição de texto manuscrito em uma imagem.
Pré-condição	1. Imagem com texto manuscrito pré-processada. 2. Arquivo com parâmetros aprendidos no treinamento.
Pós-condição	Nenhuma.
Cenário principal	1. O usuário inicia o modelo com o arquivo de pesos ótimos. 2. O usuário passa o caminho da imagem a ser transcrita. 3. O modelo exibe o texto contido na imagem.

Quadro 2 - Detalhamento do UC02

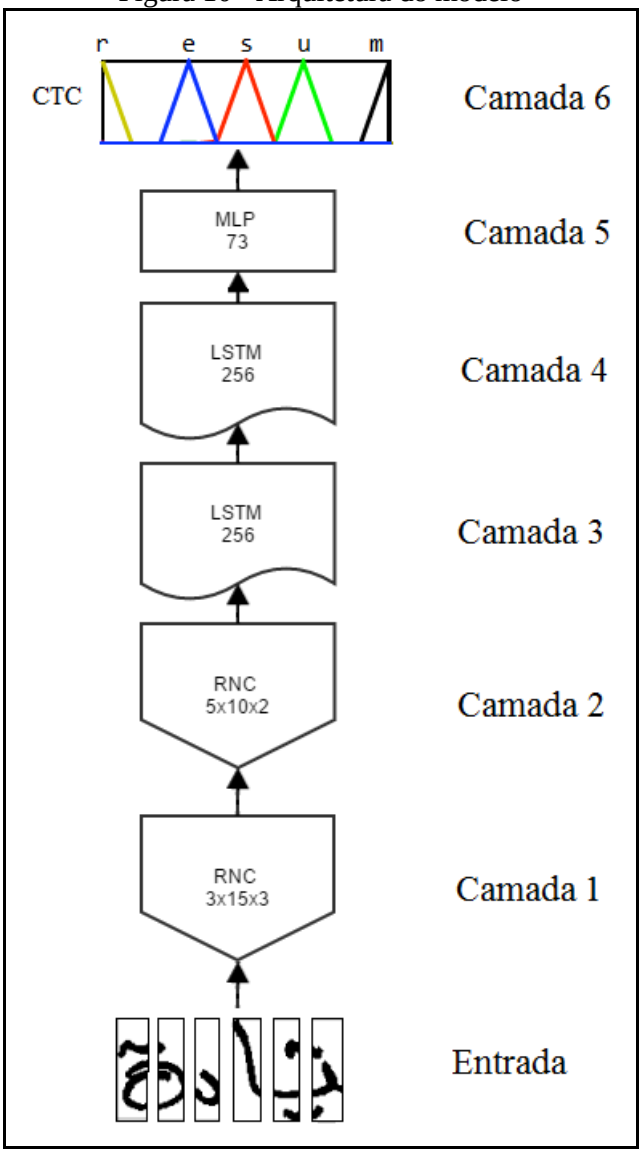
Caso de uso	Treinar modelo
Descrição	Este caso de uso descreve os procedimentos para o treinamento do modelo de reconhecimento de texto manuscrito.
Pré-condição	Base de dados pré-processada.
Pós-condição	Arquivo com parâmetros aprendidos do modelo.
Cenário principal	1. O usuário inicia o modelo. 2. O usuário inicia o treinamento do modelo. 3. A cada iteração do treinamento é exibido o progresso do aprendizado. 4. Ao fim do treinamento os parâmetros aprendidos são armazenados em um arquivo.

3.2.2 Especificação do modelo

O modelo proposto neste trabalho para o reconhecimento de texto manuscrito *off-line* é composto por 6 camadas, sem contar a camada de entrada, sendo que apenas as 5 primeiras possuem parâmetros treináveis. O modelo é composto por um híbrido de RNCs e RNRs. As RNCs são utilizadas para extração de características relevantes das imagens, como traços ou formas. Essas características são passadas as RNRs, que utilizam o contexto de características vistas até o momento para classificar o caractere sendo visto no momento. Além disso, são

utilizadas camadas MLP e CTC. A Figura 10 apresenta um diagrama com a arquitetura de camadas do modelo.

Figura 10 - Arquitetura do modelo



A entrada do modelo é uma imagem com uma altura de 160 pixels e com largura que seja múltiplo de 20. Esta imagem será dividida em sequência de imagens com 160 pixels de altura por 20 pixels de largura, por isso a necessidade da largura ser múltiplo de 20. Os traços são indicados por 1 e a ausência de traços com 0.

A primeira camada do modelo é formada por uma RNC com 3 filtros com 15 pixels de altura por 3 pixels de largura. A altura maior do filtro tem o objetivo de reduzir a altura da imagem. As ativações da convolução passam uma amostragem de tamanho 2x2. A segunda camada, C2 é formada por mais uma RNC, essa com 5 filtros com 10 pixels de altura por 2 de largura e uma amostragem de tamanho 2x2. Ambas as camadas utilizam a tangente hiperbólica como função de ativação.

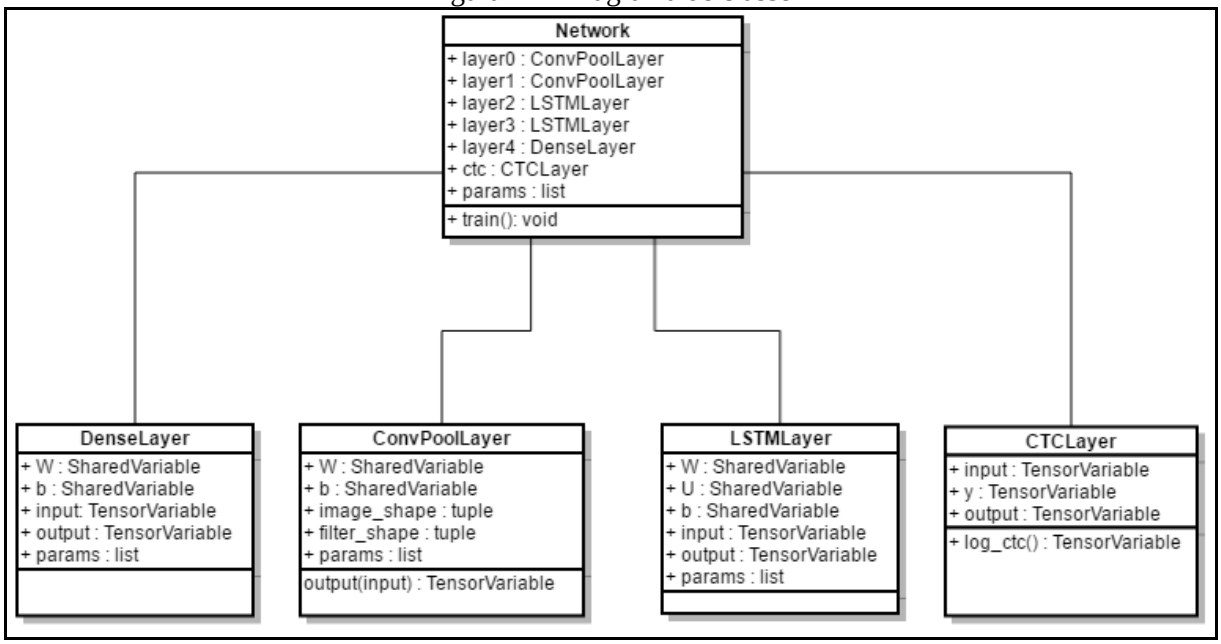
A terceira camada do modelo é formada por uma RNR do tipo LSTM com 256 células de memória e utiliza a função sigmoide para ativação dos portões internos e a tangente hiperbólica para ativação da saída. A quarta camada é idêntica à terceira camada. A escolha de 256 células de memória foi baseada em outros trabalhos utilizando camadas LSTM para reconhecimento de texto.

A quinta camada do modelo é uma camada de rede MLP utilizada para reduzir a dimensionalidade das ativações da quarta camada para 73, representando as 72 classes de caracteres mais uma classe que representa a ausência de qualquer caractere. Por fim, a sexta camada do modelo é composta por uma camada CTC para que a rede possa ser treinada para um problema de classificação temporal, como é o caso do reconhecimento de texto manuscrito.

3.2.3 Diagrama de classes do protótipo

Apesar da implementação do protótipo com a biblioteca Theano seguir o paradigma de programação funcional, a implementação de cada camada é encapsulada em classes para um melhor entendimento do código fonte desenvolvido. As classes desenvolvidas são descritas no diagrama de classes visto na Figura 11.

Figura 11 - Diagrama de classe



A classe `Network` é responsável por representar o modelo. Ela é utilizada para montar a estrutura de camadas do modelo, instanciando e interligando as camadas de redes neurais. Além disso, ela é responsável por implementar a lógica de treinamento do modelo.

As classes `DenseLayer`, `ConvPoolLayer` e `LSTMLayer` representam as camadas MLP, RNC e LSTM respectivamente. Estas classes servem como estruturas que armazenam os valores aprendidos e o grafo de operações que define a saída da camada. A classe `CTCLayer` representa a camada CTC e é responsável por implementar a função de custo que será minimizada no treinamento da rede. Esta é a única camada que não possui valores que devem ser aprendidos.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas e detalhadas as técnicas e ferramentas utilizadas para a implementação do protótipo.

3.3.1 Técnicas e ferramentas utilizadas

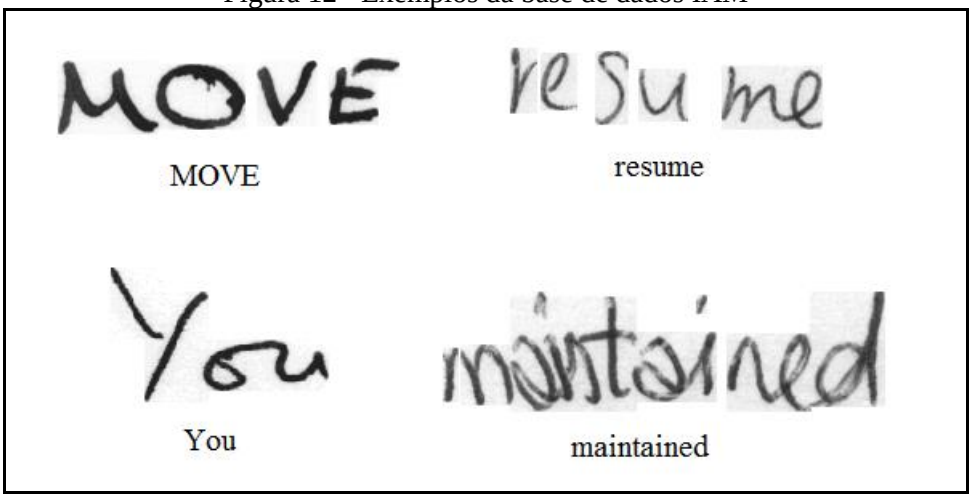
O protótipo do modelo foi desenvolvido utilizando a linguagem de programação Python com o interpretador CPython 2.7.10. Para a implementação das redes neurais foi utilizado a biblioteca de matemática simbólica Theano, que é baseada no paradigma de programação funcional. Para a manipulação dos dados, como as imagens e rótulos do treinamento foi utilizada a biblioteca NumPy que provê métodos eficientes para manipulação de dados multidimensionais. Para o pré-processamento das imagens de treinamento foi utilizada a biblioteca de visão computacional OpenCV que oferece várias rotinas de processamento de imagens. Para o treinamento do modelo foi utilizada a base de dados de imagens de texto manuscrito IAM.

O desenvolvimento do protótipo ocorreu em três etapas. Na primeira etapa foi desenvolvido o modulo responsável pelo pré-processamento das imagens passadas ao modelo. Na segunda etapa foram desenvolvidas as redes neurais utilizadas e o modelo completo. Na terceira etapa o modelo foi treinado para a classificação de texto manuscrito.

3.3.1.1 A base de dados IAM

A base de dados IAM é uma base de imagens de formulários manuscritos em inglês que pode ser utilizada para o treinamento ou teste de performance de sistemas para reconhecimento de texto manuscrito. Sendo uma das mais conhecidas bases de dados de texto manuscrito, ela é usada como *benchmark* para técnicas de reconhecimento de texto manuscrito (MARTI; BUNKE, 2002).

Figura 12 - Exemplos da base de dados IAM



A base contém formulários de texto manuscrito que foram escaneados com uma resolução de 300dpi e salvos no formato PNG. Contribuíram para a base de dados 657 pessoas diferentes, que escreveram 1539 formulários. Destes, foram isolados e rotulados tanto linhas quanto palavras, somando 13353 exemplos de linhas ou 115320 palavras. A Figura 12 mostra alguns exemplos da base de palavras isoladas e seus devidos rótulos (MARTI; BUNKE, 2002).

3.3.1.2 Theano

Theano é uma biblioteca que permite definir e calcular eficientemente operações matemáticas envolvendo dados multidimensionais. As operações matemáticas são definidas através da interface em Python e utilizando o paradigma de programação funcional. A biblioteca representa as operações matemáticas em forma de um grafo, para que possa aplicar simplificações matemáticas e remover computações redundantes. Essas operações são então compiladas em funções pela biblioteca em código C/C++ ou CUDA utilizando bibliotecas otimizadas de álgebra linear para que a execução obtenha melhor performance (AL-RFOU et al., 2016).

Devido ao seu foco em aplicações de Aprendizado Profundo, a biblioteca ainda fornece diversos mecanismos para facilitar a implementação de redes neurais. Uma das mais importantes é a diferenciação automática de funções, livrando o trabalho de implementar a retro propagação da rede. Além disso, a biblioteca provê também algumas funções comuns na implementação de redes neurais como funções de ativação ou a operação de convolução.

3.3.2 Pré-processamento das imagens

Para que as imagens da base de dados atendam ao formato de entrada do modelo é necessário que elas passem por algumas etapas de pré-processamento. O formato de imagens recebido pelo modelo são imagens com 160 pixels de altura e uma largura que seja múltiplo de 20 cujos valores estejam entre 0 e 1. As imagens utilizadas neste protótipo são da versão de palavras isoladas da base de dados IAM. A base de dados IAM contém imagens em escala de cinza e possuem tamanhos variados.

O primeiro passo do pré-processamento é a binarização da imagem para que ruídos do fundo da imagem sejam eliminados. Para tal é utilizado o método `threshold` da biblioteca OpenCV. O valor para o limiar da binarização é fornecido pela base de dados.

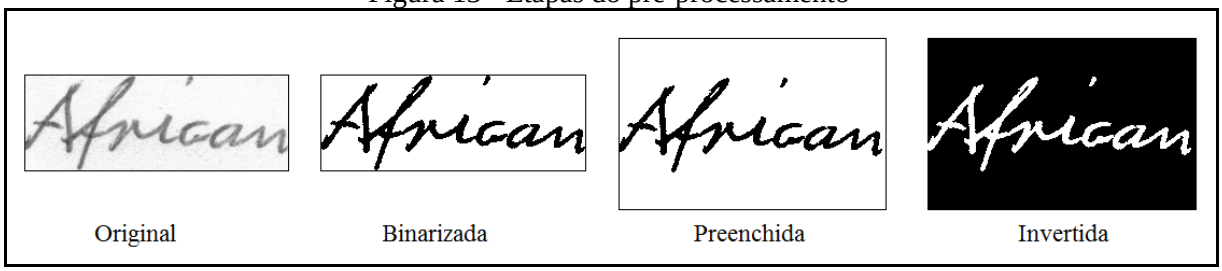
Após a binarização a imagem tem sua altura normalizada para 160 pixels. Nesta etapa, caso a imagem tenha altura menor do que 160 pixels ela é preenchida com espaço em branco igualmente nas duas partes, em cima e embaixo. Caso sua altura seja maior que 160, a imagem é redimensionada para que sua altura seja 160 mantendo a proporção da largura. A implementação do redimensionamento é feita utilizando o método `rezise` da biblioteca OpenCV como pode ser visto na linha 10 do Quadro 3. A imagem é também preenchida com espaço em branco a sua direita para que sua largura seja um múltiplo de 20.

Quadro 3 – Normalização da altura da imagem

```
1.  MAX_HEIGHT = 160
2.  if img.shape[0] < MAX_HEIGHT:
3.      pad = np.zeros(((MAX_HEIGHT - img.shape[0]) / 2, img.shape[1]))
4.      pad.fill(255)
5.      img = np.vstack((pad, img))
6.      pad = np.zeros((MAX_HEIGHT - img.shape[0], img.shape[1]))
7.      pad.fill(255)
8.      img = np.vstack((img, pad))
9.  elif img.shape[0] > MAX_HEIGHT:
10.     img = cv2.resize(img, (MAX_HEIGHT * img.shape[1] / img.shape[0],
11.                            MAX_HEIGHT),
12.                      interpolation=cv2.INTER_AREA)
```

Em seguida os valores dos pixels são normalizados para que seu valor esteja entre 0 e 1, isto é feito dividindo cada pixel da imagem por 255. Como pixels brancos são representados por 255, o resultado desta etapa é uma imagem com valor 1 onde não há nada e valor 0 onde há um traço. A última etapa do pré-processamento é inverter os valores dos pixels, para que a ausência de informação seja representada por 0 e as informações sejam representados por 1. A Figura 13 mostra o resultado de cada etapa do pré-processamento da imagem.

Figura 13 - Etapas do pré-processamento



3.3.3 Implementação do modelo

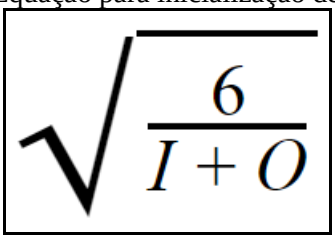
O modelo é formado por 4 tipos diferentes de camadas: RNC, LSTM, MLP e CTC. Apesar de sua implementação ser feita de forma funcional com a biblioteca Theano, a computação de cada uma das camadas foi encapsulada em uma classe para que o código fonte fosse de mais fácil entendimento. Após a implementação de cada camada, o modelo completo foi montado, instanciando e interligando cada camada.

A classes representando as camadas da rede seguem a mesma estrutura. Elas são formadas por uma variável simbólica do Theano representando sua entrada, um grafo de operações representando a computação da saída da camada e suas matrizes de valores e vetor de *bias*. As matrizes de valores e vetor de *bias*, que são os valores aprendidos pela rede, são representadas por variáveis compartilhadas do Theano. Variável compartilhada é um conceito da biblioteca Theano que pode ser usado em expressões simbólicas, mas que possui um valor interno, que será o valor que ela representará quando a expressão for computada.

3.3.3.1 Camada MLP

A classe que representa a camada MLP é formada por uma matriz de valores `W`, um vetor de *bias* `b` e sua computação simbólica representada pelo atributo `output`. Os valores da matriz e do vetor de *bias* são encapsulados em variáveis compartilhadas do Theano, como visto nas linhas 12 e 13 do Quadro 5. Caso não existam valores para a matriz o e vetor, eles devem ser inicializados. O vetor de *bias* é inicializado com zeros. Já a matriz de valores é inicializada utilizando uma técnica proposta por Glorot e Bengio (2010), em que os valores iniciais da matriz são uma distribuição uniforme entre o valor negativo e positivo da função descrita no Quadro 4. Na equação, I é a dimensionalidade da entrada da camada e O é a dimensionalidade da saída. Essa inicialização garante que as informações possam ser facilmente transitadas pela rede desde o início do treinamento, fazendo com que o treinamento possa convergir rapidamente.

Quadro 4 - Equação para inicialização de parâmetros



A entrada da camada, representada pela variável `input` no Quadro 5, é uma variável simbólica do Theano que representa uma matriz de tamanho NxM. Onde N representa o número de exemplos que serão processados, que no caso do modelo proposto é o número de pedaços em que a imagem foi dividida, e M é o número de características que a camada recebe. Para calcular a saída da rede a matriz de entrada é multiplicada pela matriz de valores `W` e os valores do vetor `b` são somados a cada linha da matriz resultante. Por fim, cada elemento da matriz é ativado pela função da tangente hiperbólica. Este cálculo é representado simbolicamente utilizando operações do Theano, conforme visto nas linhas 15 e 16 do Quadro 5.

Quadro 5 - Construtor da camada MLP

```
1.  def __init__(self, input, in_size, out_size, params=None,
2.                activation=tt.tanh):
3.      self.input = input
4.
5.      if params is None:
6.          W_values = self._init((in_size, out_size))
7.          b_values = np.zeros((out_size,), dtype=theano.config.floatX)
8.      else:
9.          W_values = params[0]
10.         b_values = params[1]
11.
12.     self.W = theano.shared(value=W_values, name='W', borrow=True)
13.     self.b = theano.shared(value=b_values, name='b', borrow=True)
14.
15.     lin_output = tt.dot(self.input, self.W) + self.b
16.     self.output = (lin_output if activation == None
17.                    else activation(lin_output))
18.     self.params = [self.W, self.b]
```

3.3.3.2 Camada RNC

A camada convolucional é composta por uma matriz `W` de quatro dimensões representando seus filtros e um vetor de *bias* `b`, que são encapsulados em variáveis compartilhadas do Theano. Caso não sejam passados nenhum valor previamente treinado para a camada, seus valores são inicializados utilizando o mesmo método descrito para as camadas MLP.

Diferente das outras que possuem um atributo `output` com as operações para computar o resultado da camada, na camada Convolucional foi necessário criar uma função que cria um

novo grafo de operações para cada entrada. Isso é necessário, pois como a imagem é dividida em várias partes, cada parte da imagem é uma entrada diferente e deve ser calculada separadamente.

O método `output` recebe como parâmetro uma matriz de 4 dimensões de tamanho BxCxAxL, onde B é sempre 1 no modelo, pois apenas uma imagem é processada por vez, C é o número de canais da imagem, A corresponde a altura e L a largura da imagem. Essa imagem passa pelas operações de convolução e amostragem, implementadas utilizando as funções `conv2d` e `pool_2d` da biblioteca Theano e podem ser vistas nas linhas 2 e 8 do Quadro 6, respectivamente. O resultado destas operações é uma matriz de 4 dimensões representado uma imagem com o mesmo número de canais do que o número de filtros da camada e a altura e largura dependem do tamanho dos filtros e da amostragem. Por fim, os valores do vetor `b` são somados a cada canal da imagem e cada pixel é ativado pela função da tangente hiperbólica.

Quadro 6 - Implementação da camada de convolução

```
1.  def output(self, input):
2.      conv_out = conv.conv2d(
3.          input=input,
4.          filters=self.W,
5.          filter_shape=self.filter_shape,
6.          image_shape=self.image_shape
7.      )
8.      pooled_out = pool_2d(
9.          input=conv_out,
10.         ds=self.poolsize,
11.         ignore_border=True
12.     )
13.
14.     return self.activation(
15.         pooled_out + self.b.dimshuffle('x', 0, 'x', 'x')
16.     )
```

3.3.3.3 Camada LSTM

A camada LSTM possui 8 matrizes de valores, duas para cada um de seus portões de ativação. Entretanto, na implementação da camada LSTM essas matrizes foram concatenadas em apenas duas, as matrizes `W` e `U`. Isto é feito para que os cálculos da ativação de cada portão possam ser executados em paralelo, acelerando a execução da camada. A classe representado a camada LSTM é composta então por duas matrizes de valor `W` e `U`, um vetor de *bias* `b` e a computação simbólica do seu resultado.

As equações descritas na Figura 14, descrevem como calcular um passo da camada LSTM. As equações de um a quatro calculam as ativações dos portões de entrada, memória, esquecimento e saída respectivamente. A variável `x` na equação representa o vetor com as características de entrada e a variável `h` representa um vetor com o resultado. O subscrito `t`

representa que este é um valor deste passo da sequência de entrada da camada, já o subscrito `t-1` representa que o valor é derivado do passo anterior da camada. Após as ativações dos portões é calculado na equação 5 o novo valor da memória da camada, e em seguida, na equação 6 o valor da saída da camada. A implementação do passo da camada LSTM pode ser vista no método `step` na Figura 14.

Figura 14 - Equações para cálculo de um passo da LSTM

$$G_i = \sigma(W_i * x_t + U_i * h_{t-1} + b_i) \quad (1)$$

$$G_c = tanh(W_c * x_t + U_c * h_{t-1} + b_c) \quad (2)$$

$$G_f = \sigma(W_f * x_t + U_f * h_{t-1} + b_f) \quad (3)$$

$$G_o = \sigma(W_o * x_t + U_o * h_{t-1} + b_o) \quad (4)$$

$$C_t = G_i * G_c + G_f * C_{t-1} \quad (5)$$

$$h_t = G_o * tanh(C_t) \quad (6)$$

A sequência de entrada da camada LSTM é representada por uma matriz de tamanho NxM onde N é o número de elementos da sequência e M o número de características de entrada da camada. O resultado da saída é obtido iterando sobre sequência de entrada e executando o passo da camada para cada elemento da sequência. Iterações sobre sequências são representadas pela operação `scan` na biblioteca Theano. O Quadro 7 mostra a implementação da saída da camada LSTM.

Quadro 7 - Implementação da computação dos passos da camada LSTM

```
1.  def slice(x, n, dim):
2.      return x[n * dim:(n + 1) * dim]
3.
4.  def step(x, h_, c_):
5.      preact = tt.dot(h_, self.U)
6.      preact += x
7.
8.      i = tt.nnet.sigmoid(slice(preact, 0, out_size))
9.      f = tt.nnet.sigmoid(slice(preact, 1, out_size))
10.     o = tt.nnet.sigmoid(slice(preact, 2, out_size))
11.     c = tt.tanh(slice(preact, 3, out_size))
12.     c = f * c_ + i * c
13.     h = o * tt.tanh(c)
14.
15.     return h, c
16.
17. tmp = tt.dot(self.input, self.W) + self.b
18. vals, _ = theano.scan(step, sequences=[tmp], outputs_info=[
19.     np.zeros((out_size,), dtype=theano.config.floatX),
20.     np.zeros((out_size,), dtype=theano.config.floatX),
21. ])
```

3.3.3.4 Camada CTC

A camada CTC, diferente das outras camadas, não possui valores treináveis. Sua função é prover a função de custo para que a rede possa ser treinada para um problema de classificação sequencial, como é o caso do reconhecimento de texto manuscrito.

A função de custo da camada CTC é calculada somando as probabilidades de todos os possíveis alinhamentos entre a sequência de entrada e a de saída. Entretanto, o cálculo das probabilidades dos alinhamentos é feito de forma recursiva, dificultando sua implementação na biblioteca Theano. Esta recursividade foi definida na forma de uma matriz para que pudesse ser calculada iterativamente, como pode ser visto no Quadro 8. Após calculadas as probabilidades, elas são somadas e é retornado o valor negativo do seu logaritmo natural.

Quadro 8 - Implementação da função de custo da camada CTC

```
1.  def plain_ctc(self):
2.      l = tt.concatenate((self.y, [self.blank, self.blank]))
3.      sec_diag = tt.neq(l[:-2], l[2:]) * tt.eq(l[1:-1], self.blank)
4.
5.      recurrence_relation = (
6.          tt.eye(self.y.shape[0]) +
7.          tt.eye(self.y.shape[0], k=1) +
8.          tt.eye(self.y.shape[0], k=2) *
9.          sec_diag.dimshuffle((0, 'x'))
10.     )
11.
12.     pred_y = self.input[:, self.y]
13.     probs, _ = theano.scan(
14.         lambda curr, accum: curr * tt.dot(accum, recurrence_relation),
15.         sequences=[pred_y],
16.         outputs_info=[tt.eye(self.y.shape[0])[0]]
17.     )
18.
19.     l_probs = tt.sum(probs[-1, -2:])
20.     return -tt.log(l_probs)
```

3.3.3.5 O modelo

O modelo é representado pela classe `Network`, que encapsula a computação de todas as camadas e o treinamento da rede. Em seu construtor as camadas são instanciadas passando seus parâmetros como número de neurônios ou formato dos filtros, conforme visto na especificação. As camadas são então interligadas para que as ativações sejam propagadas da primeira à última camada.

Primeiramente são instanciadas as camadas RNC. Então, utilizando a operação `scan`, é criada uma iteração sobre a imagem de entrada que a divide em janelas de 20 pixels e alimenta a primeira camada RNC. O resultado da primeira camada é utilizado para alimentar a segunda camada RNC. A matriz de 4 dimensões resultante, representando as características extraídas pelas camadas RNC é então achatada para uma dimensão utilizando o `flatten` da biblioteca Theano como pode ser visto no Quadro 9, se tornando um vetor. O resultado desta operação de iteração é uma matriz onde cada linha representa um pedaço da imagem e cada coluna uma característica extraída daquele pedaço pelas camadas RNC.

Quadro 9 - Iteração das camadas RNC sobre os pedaços da imagem

```
1.  def conv(index, X):
2.      slice_x = X[:, index * w:(index + 1) * w]
3.      return self.layer1.output(self.layer0.output(
4.          slice_x.dimshuffle('x', 'x', 0, 1)
5.      )).flatten(1)
6.
7.  conv_out, _ = theano.scan(
8.      fn=conv, sequences=[tt.arange(self.input.shape[1] / w)],
9.      non_sequences=self.input, outputs_info=None
10. )
```

Em seguida são instanciadas as camadas LSTM. Para isso são calculados a quantidade de características de entrada da primeira camada LSTM a partir do tamanho dos filtros da segunda camada RNC, como visto na linha 3 do Quadro 10. A primeira camada LSTM recebe como entrada o resultado da iteração das camadas RNC. O resultado da primeira camada LSTM é passado para a segunda. Então a camada MLP é instanciada e recebe o resultado da segunda camada LSTM como entrada. Por fim, a camada CTC é instanciada e recebe o resultado da camada MLP como entrada. A implementação da interligação dessas camadas pode ser vista no Quadro 10.

Quadro 10 - Interligação das camadas

```
1.  self.layer2 = LSTMLayer(
2.      input=conv_out,
3.      in_size=((h2 - fh2 + 1) / 2) * ((w2 - fw2 + 1) / 2) * fn2,
4.      out_size=256
5.  )
6.
7.  self.layer3 = LSTMLayer(
8.      input=self.layer2.output,
9.      in_size=256,
10.     out_size=256
11. )
12.
13. self.layer4 = DenseLayer(
14.     input=self.layer4.output,
15.     in_size=256,
16.     out_size=len(CLASSES) + 1,
17.     activation=None
18. )
19.
20. self.ctc = CTCLayer(
21.     input=self.layer3.output,
22.     y=self.y
23. )
```

3.3.3.6 Treinamento

Devido ao mecanismo de diferenciação automática da biblioteca Theano a implementação do treinamento da rede neural torna-se bastante simples. A função grad do Theano calcula os gradientes de erro a partir da função de custo definida pela camada CTC, que são utilizados para atualizar as matrizes de valores das camadas. Os valores das matrizes

são atualizados a partir dos gradientes utilizando uma taxa de aprendizado. A taxa de aprendizado utilizada no treinamento é de 0,001 e diminui para 0,0001 a partir da época 20. As atualizações das matrizes foram compiladas em uma função Theano para que possam ser calculadas. A função train_model, como visto na linha 8 do Quadro 11, recebe como parâmetros uma imagem e seu rótulo e atualiza os valores das matrizes.

Quadro 11 - Implementação da atualização dos pesos da rede

```
1.  cost = self.ctc.log_ctc()
2.  grads = tt.grad(cost, self.params)
3.  updates = [
4.      (param_i, param_i - learning_rate * grad_i)
5.      for param_i, grad_i in zip(self.params, grads)
6.  ]
7.
8.  train_model = theano.function(
9.      inputs=[self.input, self.y],
10.     outputs=cost,
11.     updates=updates
12. )
```

Para o treinamento foram utilizados 82452 dos 115320 exemplos da base de palavras isoladas IAM. Foram removidos exemplos que possuíam erros na segmentação da palavra e os exemplos que foram usados para realizar o teste do modelo. Devido a limitação de memória RAM do computador utilizado para realizar o treinamento da rede, os exemplos de treinamento foram separados em 3 grupos de 27484 exemplos, para que pudessem ser carregados para memória separadamente.

Cada época do treinamento possui três iterações. Em cada iteração um grupo de exemplos é carregado para a memória e o método `train_model` é executado para cada exemplo. Ao final da iteração é calculada a média de erros para o grupo de exemplos e caso o erro seja o menor visto até o momento as matrizes de valores são salvas em um arquivo e o menor erro é atualizado. O treinamento continua por 50 épocas ou até ser parado pelo usuário.

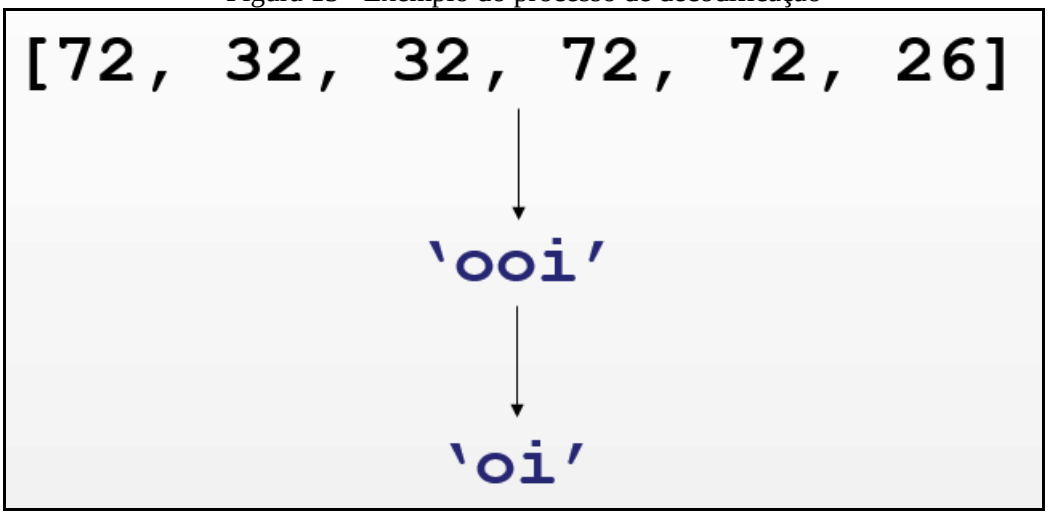
3.3.4 Decodificação

O resultado final da rede, devido ao treinamento com a função de custo CTC, será a probabilidade de cada caractere para cada elemento da sequência de entrada. Para transformar este resultado em uma palavra é necessária uma etapa conhecida como decodificação.

O algoritmo escolhido para a decodificação do resultado é um dos algoritmos mais simples para decodificação do resultado de uma camada CTC. Ele consiste basicamente em encontrar o caractere mais provável a cada elemento da sequência de saída e remover os que se repetem. Um exemplo deste processo pode ser visualizado na Figura 15. No exemplo o resultado da rede é um vetor com as classes 72, 32 e 26, que representam os caracteres `null`,

`o` e `i` respectivamente. Este vetor é representado como uma `string` e a partir desta representação os caracteres repetidos são removidos.

Figura 15 - Exemplo do processo de decodificação



No Quadro 12 é possível ver a implementação da decodificação da rede. Na linha 2 a rede faz a previsão e retorna a uma lista com o índice da classe mais provável para cada elemento da sequência. O método `stringfy`, na linha 3, mapeia estes índices para seus caracteres correspondentes. Por fim, utilizando o método `groupby` da biblioteca padrão linguagem Python, são removidas as repetições de caracteres.

Quadro 12 - Implementação da decodificação do resultado

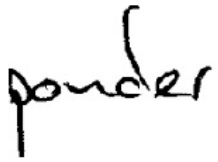
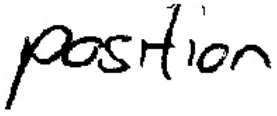
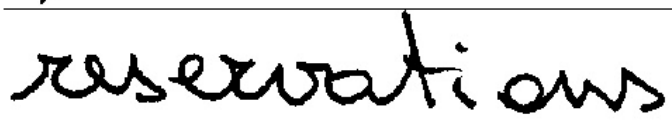
```
1.  def predict(self, img):
2.      net_out = self.pred(img)
3.      net_out = stringfy(net_out)
4.      return ''.join(ch for ch, _ in itertools.groupby(net_out))
```

3.4 RESULTADOS E DISCUSSÕES

Após a implementação do protótipo foi iniciado seu treinamento. O treinamento do protótipo foi realizado em um computador com um processador *Core* I7-3612QM com *clock* de 2,10GHz e 8 GB de memória RAM e sistema operacional Ubuntu 15.10. O treinamento da rede durou 40 horas, quando foi interrompido devido a limitações de tempo para implementação do protótipo.

Após o treinamento o protótipo foi utilizado para reconhecer alguns dos exemplos do conjunto de exemplos de teste a fim de avaliar o seu funcionamento. Na Figura 16 são mostradas as imagens utilizadas nesta validação inicial, assim como seus rótulos e a previsão do protótipo para a imagem.

Figura 16 - Resultados da validação inicial do protótipo

Imagem	Rótulo	Previsão
	ponder	poder
	position	postion
	atomic	atomic
	to	to
	reservations	heoservations
	fortnight	tofortuight

A partir desta validação inicial é possível verificar que o protótipo mostra uma boa taxa de reconhecimento para palavras pequenas. Entretanto, o reconhecimento de palavras maiores só se mostra eficiente nos últimos caracteres da palavra. Isto se deve, pois, a rede não foi capaz de propagar os erros do treinamento relativos a dependências mais antigas da sequência de imagens. Foi possível aprender apenas as dependências mais recentes, que neste caso são os últimos caracteres de cada imagem. O motivo para este problema foi o tempo de treinamento do protótipo que, levando em consideração a quantidade de parâmetros que devem ser aprendidos pela rede, foi curto.

Em seguida, foi realizado um teste para avaliar a performance do protótipo. Para isto foram calculadas as previsões do protótipo para todas as 2040 imagens do conjunto de exemplos de teste. Em seguida a performance foi medida utilizando duas medidas comumente utilizadas para o reconhecimento de texto manuscrito: a taxa de erro de palavra (WER) e a taxa de erro de caractere (CER).

As taxas WER e CER medem a proporção de erros da previsão e são definidas pela equação vista no Quadro 13, onde S representa o número de substituições, D o de deleções, I o de inserções e N o número total de palavras ou caracteres. Substituições, deleções e inserções representam as operações necessárias para transformar a previsão do modelo no

rótulo do exemplo. Como o protótipo desenvolvido reconhece palavras isoladas, a taxa WER é calculada como o número de previsões erradas dividido pelo total de previsões realizadas.

Quadro 13 - Equação do cálculo das taxas WER e CER

$$ERROR = \frac{S + I + D}{N}$$

Após calculadas as previsões do protótipo para o conjunto de teste foram calculadas as taxas WER e CER para todos os exemplos do conjunto. Foram calculados também as taxas WER e CER apenas de imagens que continham palavras com menos de 4 caracteres, a fim de mostrar que a taxa de erro acima do esperado para os testes se deve à falta de treinamento de dependências antigas da sequência de entrada. Os resultados destes testes podem ser visualizados na Tabela 1. Nela é possível perceber que as taxas de erro para palavras com menos de 4 caracteres é sensivelmente menor, provando a hipótese da falta de treinamento do protótipo.

Tabela 1 – Resultado dos testes de performance do protótipo

	WER	CER
Total	71,4	48,38
Menos de 4	58,62	48,83

Outro motivo para taxas de erro tão acima do esperado vem da escolha do algoritmo para decodificação dos resultados do modelo. O método se baseia apenas na escolha do caractere mais provável para cada elemento da sequência de entrada, não levando em consideração ativações fracas da rede. Isso faz com que um caractere possa ser escolhido para representar o resultado da previsão mesmo que a probabilidade de que o caractere faça parte do resultado seja muito baixa.

O Quadro 14 resume as características e resultados finais do trabalho desenvolvido em comparação aos trabalhos correlatos.

Quadro 14 - Comparação dos resultados com trabalhos correlatos

Características / trabalhos correlatos	Graves (2012)	Su et al. (2015)	Chen, Yan e Huo (2015)	Caíque Reinhold (2016)
Técnicas de pré-processamento	Binarização	Correção das inclinações vertical e horizontal e binarização	Normallização da altura e binarização	Normalização da altura e binarização
Mecanismo de classificação	RNRMD	BLSTM	LSTM + HMM	RNC + LSTM
Características utilizadas pelo classificador	Pixels da imagem	Características geométricas e HOG	*Principal Component Analysis*	Pixels da imagem
Tipo de texto reconhecido	Códigos postais	Formulários	Sem restrição	Sem restrição
WER	10,58	6,7	29,03	71,4
CER	Não informado	Não informado	15,16	48,83

É importante ressaltar que os trabalhos desenvolvidos por Graves (2012) e Su et al. (2015) apresentam uma discrepância na taxa de acerto em relação aos demais trabalhos devido às restrições impostas no texto que pode ser reconhecido. Em ambos os casos o dicionário de palavras que podem ser reconhecidas é restrito à sua aplicação específica, possuindo menos variação que o reconhecimento de texto sem restrições.

Uma vantagem do modelo apresentado neste trabalho em relação ao trabalho de Su et al. (2015) é que, assim como os trabalhos de Graves (2012) e Chen, Yan e Huo (2015) ele pode ser utilizado para reconhecimento de outros alfabetos. Com mínimas ou nenhuma mudança, o modelo pode ser treinado para aprender a reconhecer texto em outras línguas ou alfabetos. Isto é possível pois a extração de características da imagem é aprendida pelo modelo e o pré-processamento necessário não modifica o formato do conteúdo da imagem. Já no trabalho de Su et al. (2015) a correção das inclinações no pré-processamento da imagem e a extração das características geométricas fazem que seja necessário um conhecimento prévio do alfabeto ou língua reconhecido pelo modelo.

Outras configurações para o modelo foram testadas até chegar a configuração apresentada. A primeira delas consistia de uma camada de RNC e duas LSTM e se mostrou ineficiente. Testes com duas camadas RNC apresentaram resultados melhores, mostrando que uma única camada RNC não era suficiente para extrair informações visuais relevantes das imagens. Entretanto, a convergência do modelo com duas camadas RNC e duas camadas LSTM se mostrou muito lenta. A solução foi utilizar uma camada MLP no final da rede para redução da dimensionalidade das características de saída, resultando no modelo apresentado.

Outras dificuldades encontradas no desenvolvimento do trabalho foram encontradas na utilização da função de custo CTC. Em sua implementação inicial, o arredondamento de valores muito baixos fazia com que o resultado da função fosse infinito, impossibilitando o treinamento da rede. A solução para este problema foi implementar a função de custo de um modo que os cálculos fossem feitos no espaço logarítmico.

4 CONCLUSÕES

Este trabalho apresentou a especificação de um modelo de rede neural profunda utilizando RNCs e RNRs para o reconhecimento de texto manuscrito *off-line,* assim como a implementação de um protótipo para a validação do modelo. O protótipo foi implementado na linguagem de programação Python utilizando a biblioteca Theano e treinado com a base de dados IAM de imagens de texto manuscrito.

Os resultados obtidos com o protótipo se mostraram abaixo do esperado, com uma taxa CER de 48,83% e WER de 71,4%. Apesar disso, o modelo apresentado foi validado e se mostra uma alternativa válida para o reconhecimento de texto manuscrito *off-line*. Os motivos para o desempenho do protótipo abaixo do esperado foram elencados e possíveis soluções foram propostas.

O modelo apresentado é capaz de aprender características de ponta a ponta, ou seja, aprende a classificar classes de um alto nível de abstração a partir dos dados brutos, neste caso a imagem. Ele se mostra também bastante robusto na classificação de um padrão com uma variabilidade tão alta como o texto manuscrito em imagens digitais.

4.1 EXTENSÕES

Algumas possíveis extensões para o trabalho são:

a) utilizar BLSTM como camadas intermediárias, pois como visto nos trabalhos correlatos, um conhecimento do contexto da entrada nos dois sentidos (do começo ao fim e do fim para o começo) tende a mostrar melhores resultados;
b) utilizar um algoritmo para decodificação mais robusto, que leve ativações fracas da rede em consideração;
c) utilizar técnicas de regularização para evitar o *overfitting* da rede, aumentando seu poder de generalização para exemplos não vistos;
d) realizar o treinamento do protótipo utilizando GPU para reduzir o tempo de treinamento;
e) desenvolver um sistema de reconhecimento de texto manuscrito completo a partir do modelo apresentado;
f) criar uma base de dados de textos manuscritos em português e treinar o modelo para o reconhecimento de texto manuscrito na língua portuguesa.

REFERÊNCIAS

AL-RFOU, Rami et al. Theano: A Python framework for fast computation of mathematical expressions. **arXiv: v. 1, n.** 1605.02688, jul. 2016.

ALAEI, Alireza; PAL, Umapada; NAGABHUSHAN, P.. A new scheme for unconstrained handwritten text-line segmentation. **Pattern Recognition**, [s.l.], v. 44, n. 4, p.917-928, abr. 2011.

AREL, I; ROSE, D C; KARNOWSKI, T P. Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]. **Ieee Computacional Intelligence Magazine**, [s.l.], v. 5, n. 4, p.13-18, nov. 2010.

BENGIO, Y.. Learning Deep Architectures for AI. **Foundations And Trends In Machine Learning**, [s.l.], v. 2, n. 1, p.1-127, 2009.

BENGIO, Yoshua; COURVILLE, Aaron; VINCENT, Pascal. Representation Learning: A Review and New Perspectives. **Ieee Transactions On Pattern Analysis And Machine Intelligence**, [s.l.], v. 35, n. 8, p.1798-1828, ago. 2013.

BENGIO, Yoshua; LECUN, Yann. Scaling Learning Algorithms toward AI. In: BOTTOU, Léon et al. **Large-Scale Kernel Machines**. Cambridge: Mit Press, 2007. p. 321-359.

BERTOLAMI, Roman; BUNKE, H.. **Ensemble Methods for Offline Handwritten Text Line Recognition**. 2008. 142 f. Tese (Doutorado) - Curso de Ciência da Computação, Universidade de Berna, Berna, 2008.

BEZERRA, Byron Leite Dantas; ZANCHETTIN, Cleber; ANDRADE, Vinícius Braga de. A Hybrid RNN Model for Cursive Offline Handwriting Recognition. In: BRAZILIAN SYMPOSIUM ON NEURAL NETWORKS, 13., 2012, Curitiba. **Proceedings of the 13th Brazilian Symposium on Neural Networks**. Curitiba: Ieee, 2012. p. 113 - 118.

BOUREAU, Y-lan; PONCE, Jean; LECUN, Yann. A Theoretical Analysis of Feature Pooling in Visual Recognition. In: ICML, 27., 2010, Haifa. **Proceedings of the 27th Internacional Conference on Machine Learning**. Madison: Icml, 2010. p. 111 - 118.

CHEN, Kai; YAN, Zhi-jie; HUO, Qiang. A context-sensitive-chunk BPTT approach to training deep LSTM/BLSTM recurrent neural networks for offline handwriting recognition. In: ICDAR, 13., 2015, Nancy. **Proceedings of the 13th International Conference on Document Analysis and Recognition**. [s.l.]: Ieee, 2015. p. 411 - 415.

CIRESAN, Dan Claudiu et al. Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. **Neural Computation**, [s.l.], v. 22, n. 12, p.3207-3220, dez. 2010.

FAUSETT, Laurene. **Fundamentals of neural networks: architectures, algorithms, and applications**. [s.l.]: Prentice-hall, 1994.

GLOROT, Xavier; BENGIO, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In: AISTATS-10, 13., 2010, Sardinia. **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. [s.l.]: Jmlr, 2010. p. 249 - 256.

GOODFELLOW, Ian J. et al. Maxout Networks. In: ICML, 30., 2013, Atlanta. **Proceedings of The 30th International Conference on Machine Learning**. [s.l.]: Jmlr, 2013. p. 1319 - 1327.

GRAVES, A. et al. A Novel Connectionist System for Unconstrained Handwriting Recognition. **Ieee Transactions On Pattern Analysis And Machine Intelligence**, [s.l.], v. 31, n. 5, p.855-868, maio 2009. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/tpami.2008.137.

GRAVES, Alex et al. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In: ICML, 23., 2006, Pittsburgh. **Proceedings of the International Conference on Machine Learning**. Pittsburgh: Acm, 2006. p. 369 - 376.

GRAVES, Alex. **Supervised Sequence Labelling with Recurrent Neural Networks**. 2008. 114 f. Tese (Doutorado) - Curso de Ciência da Computação, Universidade Técnica de Munique, Munique, 2008.

_____. Offline Arabic Handwriting Recognition with Multidimensional Recurrent Neural Networks. In: MÄRGNER, Volker; ABED, Haikal El. **Guide to OCR for Arabic Scripts**. Londres: Springer, 2012. p. 297-313.

HOCHREITER, Sepp et al. Gradient Flow in Recurrent Nets: the Diculty of Learning Long-Term Dependencies. In: KOLEN, John F.; KREMER, Stefan C.. **A Field Guide to Dynamical Recurrent Networks**. [s.i]: Ieee Press, 2001. p. 237-245.

HOCHREITER, Sepp; SCHMIDHUBER, Jurgen. LONG SHORT-TERM MEMORY. **Neural Computation**. [s.i], p. 1711-1733. nov. 1997.

HUBEL, D.; WIESEL, T.. Receptive fields and functional architecture of monkey striate cortex. **Journal Of Physiology**. Londres, p. 215-243. mar. 1968.

HUGHES, Lorna M.. **Digitizing collections : strategic issues for the information manager**. Londres: Facet, 2004. 327 p.

LECUN, Yann; BENGIO, Yoshua. Pattern Recognition and Neural Networks. In: ARBIB, Michael A.. **Handbook of Brain Theory and Neural Networks**. Cambridge: Mit Press, 1995. p. 430-452.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings Of The Ieee**, [s.l.], v. 86, n. 11, p.2278-2324, nov. 1998.

LISA LAB (Canadá). **Convolutional Neural Networks**. 2016. Disponível em: <http://deeplearning.net/tutorial/lenet.html>. Acesso em: 11 jun. 2016.

MARTI, U. V.; BUNKE, H.. The IAM-database: an English sentence database for offline handwriting recognition. **International Journal On Document Analysis And Recognition**, [s.l.], v. 5, n. 1, p.39-46, 1 nov. 2002.

MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The Bulletin Of Mathematical Biophysics**. [s.l.], p. 115-133. dez. 1943.

PLAMONDON, R.; SRIHARI, S.n.. Online and off-line handwriting recognition: a comprehensive survey**. Ieee Transactions On Pattern Analysis And Machine Intelligence**, [s.l.], v. 22, n. 1, p.63-84, 2000. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/34.824821.

SCHMIDHUBER, Juergen. Deep Learning in Neural Networks: An Overview. **Neural Networks**. [s.i], p. 85-117. jan. 2015.

SEILER, R.; SCHENKEL, M.; EGGIMANNN, F.. Off-line cursive handwriting recognition compared with on-line recognition. In: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 13., 1996, Vienna. **Proceedings of 13th International Conference on Pattern Recognition**. Vienna: Ieee, 1996. p. 505 - 509.

SU, Bolan et al. Segmented handwritten text recognition with recurrent neural network classifiers. In: ICDAR, 13., 2015, Nancy. **Proceedings of the 13th International Conference on Document Analysis and Recognition**. [s.l.]: Ieee, 2015. p. 286 - 390.

TOMÁ, Mikolov et al. Recurrent neural network based language model. In: INTERSPEECH 2010, 11., 2010, Chiba**. Proceedings of the 11th Annual Conference of the International Speech Communication Association**. Chiba: Isca, 2010. p. 1045 - 1048.

VINCIARELLI, Alessandro. A survey on off-line Cursive Word Recognition. **Pattern Recognition**, [s.i.], v. 35, n. 7, p.1433-1446, jul. 2002.

ZEILER, Matthew D.; FERGUS, Rob. Visualizing and Understanding Convolutional Networks. In: EUROPEAN CONFERENCE ON COMPUTER VISION, 13., 2014, Zurique. **Computer Vision – ECCV 2014**. Zurique: Springer, 2014. p. 818 - 833.