UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

ESTHEVÃO BACHETTI VERVLOET

APLICAÇÃO DE *STACKED SPARSE AUTOENCODERS* AO PROBLEMA DE RECONHECIMENTO DE NÚMEROS EM IMAGENS NATURAIS

VITÓRIA
2019

ESTHEVÃO BACHETTI VERVLOET

APLICAÇÃO DE *STACKED SPARSE AUTOENCODERS* AO PROBLEMA DE RECONHECIMENTO DE NÚMEROS EM IMAGENS NATURAIS

Parte manuscrita do Projeto de Graduação do aluno **Esthevão Bachetti Vervloet**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Jorge Leonid Aching Samatelo

VITÓRIA
2019

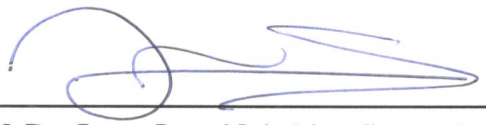
ESTHEVÃO BACHETTI VERVLOET

APLICAÇÃO DE *STACKED SPARSE AUTOENCODERS* AO PROBLEMA DE RECONHECIMENTO DE NÚMEROS EM IMAGENS NATURAIS

Parte manuscrita do Projeto de Graduação do aluno **Esthevão Bachetti Vervloet**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

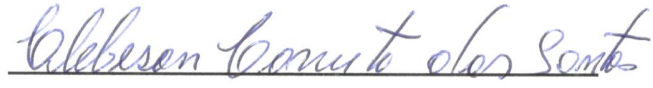
Aprovado em 03 de Dezembro de 2019.

COMISSÃO EXAMINADORA:



Prof. Dr. Jorge Leonid Aching Samatelo
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Patrick Marques Ciarelli
Universidade Federal do Espírito Santo
Examinador



Msc. Clebeson Canuto dos Santos
Universidade Federal do Espírito Santo
Examinador

Aos meus pais, amigos e família, por todo o amor e suporte durante toda a minha jornada, sempre estando ao meu lado nos momentos mais desafiadores. Ao meu orientador por ter sido meu guia e mentor durante todo esse trabalho, além de ser o professor que despertou meu interesse nessa área tão fascinante. À banca examinadora pela aceitação do convite e pelo tempo investido para leitura e avaliação desse trabalho.

RESUMO

Há muito tempo, a área de reconhecimento de texto em imagens vem sendo profundamente estudada e novas e melhores ferramentas são propostas para tentar solucionar tarefas relacionadas. Em especial, a classificação de algarismos em fotos vem auxiliando a automatização de diversos processos, como sistemas de navegação e geração de mapas. Cada vez mais a ciência se aproxima de soluções ideais, todavia isso é apenas possível com a criação de novas técnicas e otimização de técnicas já conhecidas. Além disso, devido ao aumento no interesse no reconhecimento e classificação de imagens, problemáticas mais complexas e com mais variáveis estão surgindo, necessitando de algoritmos mais completos e potentes para sua solução. É ocorrente que metodologias anteriores se tornem obsoletas com os novos avanços na área, ou múltiplas técnicas não serem compatíveis e benéficas quando aplicadas em conjunto, fomentando a necessidade de testes para obter o quanto cada uma contribui para o sistema. O objetivo deste trabalho é classificar, por meio da técnica de Aprendizado Profundo aplicada a uma rede neural *Stacked Sparse Autoencoder*, imagens de algarismos em ambientes naturais com diferentes tipos de fonte, iluminação e qualidade. Em adição, faz-se uma análise do uso das técnicas de pré-treinamento *Greedy Layer-Wise Training* e de regularização aplicadas em redes neurais. Foram realizados diversos testes em diferentes configurações e arquiteturas a fim de encontrar parâmetros mais otimizados para a tarefa vigente, obtendo assim uma acurácia média entre classes de $91,71\%$ na classificação do banco de dados de teste da base de dados SVHN.

Palavras-chave: *Deep-learning*. Redes neurais. *Autoencoders*. *Greedy Layer-Wise Pre-train*. Regularizadores. *Street View House Numbers*.

ABSTRACT

For a long time, the area of optical character recognition have been deeply studied and new and better tools are being proposed to try to solve related tasks. In particular, classification of numbers in photos has been helping to automate various processes such as navigation and map generation. Increasingly, science is approaching ideal solutions, but this is only possible with the development of new techniques and optimization of known ones. Furthermore, due to the increased interest in image recognition and classification, problems more complex and with many more uncertainties are being brought up, requiring more complete and powerful algorithms for their solution. Methodologies often become obsolete with new advances in the field, or multiple techniques are not compatible and beneficial when applied together, fomenting the need of tests to obtain how each one contributes with the sistem. The objective of this work is to, through the use of the Deep Learning technique applied to a Stacked Sparse Autoencoder neural network, classify images of numbers in natural environments with different type of sources, lighting and quality. In addition, we analize the use of the Greedy Layer-Wise Training technique and the use of regularization methods applied in neural networks. Several tests were performed in different configurations and architectures in order to find optimized parameters for the current task, thus obtaining an accuracy of $91,71\%$ for classification on the SVHN dataset.

Keywords: Deep-learning. Neural Networks. Autoencoders. Greedy Layer-Wise Training. Regularizers. *Street View House Numbers*.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DL	*Deep Learning*
DBN	*Deep Belief Networks*
LASSO	*Least Absolute Shrinkage and Selection Operator*
LFM	*Linear Factor Models*
ML	*Machine Learning*
MSE	*Mean Squared Error*
PCA	*Principal Component Analysis*
ReLU	*Rectified Linear Units*
RNA	*Redes Neurais Artificiais*
ROI	*Region of Interest*
SMC	*Softmax Classifier*
SSAE	*Stacked Sparse Autoencoder*
SVHN	*Street View House Number*
UFES	Universidade Federal do Espírito Santo

SUMÁRIO

1 INTRODUÇÃO

1.1 Apresentação e Objeto da Pesquisa

A capacidade de aprendizado de um ser humano ainda fascina a comunidade científica que estuda a área de Aprendizado de Máquina (ML, do inglês *Machine Learning*). Uma criança pode aprender sobre animais, letras e cores e saber distinguir entre elas com apenas poucos exemplos, sem a necessidade de imenso número de interações. Tal capacidade de aprendizagem é algo que ainda não é conseguido replicar perfeitamente de forma computacional. O reconhecimento de objetos a partir de experiências prévias é uma atividade que os seres humanos exercem de forma inata sempre que enfrentam novas situações.

O reconhecimento automático de objetos a partir de imagens é uma importante tarefa na área de visão computacional. Este apresenta diversas aplicações no cotidiano, como: a tradução instantânea de textos presentes em fotos (WANG et al., 2012); reconhecimento biométrico (CONNOR; ROSS, 2018); desenvolvimento de veículos autônomos (SHAH et al., 2018) e identificação de números de residências (GOODFELLOW et al., 2013). Este último é de extrema importância na criação de mapas mais precisos e permite que os sistemas de navegação possam ser melhorados. Entre os diferentes desafios, o reconhecimento de texto a partir de imagens é um problema ainda não solucionado, mas que vem sendo abordado por um número cada vez maior de pesquisas como é relatado em Ye e Doermann (2015).

Porém, enquanto problemas de reconhecimento de caracteres altamente restritos, como por exemplo, reconhecimento ótico de caracteres (OCR) em documentos impressos ou reconhecimento de dígitos manuscritos já são praticamente resolvidos, o reconhecimento de caracteres em cenas naturais é um problema em aberto, já que estes são corrompidos por fatores naturais, como embaçamento da imagem, distorção, efeitos de iluminação e variação de estilos e fontes utilizadas na escrita dos números (NETZER et al., 2011). Neste trabalho, descreve-se imagens naturais como imagens do mundo real produzidas por meio de fotografias, logo, não criadas computacionalmente.

O reconhecimento deste tipo de imagem é um desafio tão grande para um sistema computacional que é comumente utilizado em sistemas de diferenciação entre humano e máquina, como os

CAPTCHAs. O CAPTCHA é um teste de Turing inverso completamente automatizado utilizado para diferenciar entre computador e humano. Seu uso tem como função impedir que *softwares* automatizados façam acesso a serviços que são indicados apenas a seres humanos, como o preenchimento de formulários e criação de usuários. Porém, com o avanço de pesquisas na área de segmentação e reconhecimento de imagens utilizando métodos de aprendizado profundo, sistemas modernos já são capazes de identificar essas imagens com considerável taxa de acerto (GOODFELLOW et al., 2013); (NETZER et al., 2011).

O foco deste trabalho é abordar o problema de reconhecimento de números com múltiplos dígitos obtidos do conjunto de dados *Street View House Numbers* (SVHN). Desta forma, a fim de fazer o reconhecimento dos caracteres presentes nesse conjunto, será utilizada uma rede *Autoencoder* Esparso Empilhado (SSAE, do inglês *Stacked Sparse Autoencoders*) que opera diretamente nos píxeis da imagem. Este modelo será configurado utilizando *sparse autoencoders* em sequência para obtenção de correlações entre os dados de entrada, em conjunto com uma camada de classificação *Softmax*.

Em adição, são feitas análises da influência do uso da metodologia *Greedy Layer-wise Training* (HINTON; OSINDERO; TEH, 2006) e de regularizadores de ativação no comportamento de um *Autoencoder* durante seu treino. Tal comportamento será avaliado a partir das taxas de acurácia na tarefa de classificação aqui abordada.

1.2 Trabalhos Relacionados

A aprendizagem profunda se mostrou efetiva para resolver alguns dos mais desafiadores problemas na área de visão computacional desde que a primeira rede de *autoencoder* profunda foi proposta em Hinton e Zemel (1994).

Em Netzer e outros (2011), os autores fazem o uso do método de aprendizagem de características SSAE, para auxiliar camadas de classificação *softmax* na categorização de imagens do banco de dados SVHN, no formato de dígitos de 32×32 píxeis, alcançando taxa de acerto acima de $89,7\%$. Comparado com técnicas de construção manual de caraterísticas, como Histogramas de Gradientes Orientados (HOG, do inglês *Histograms of Oriented Gradients*), os métodos baseados no aprendizado destas, SSAE, apresentaram um desempenho muito maior na extração

de especificidades dos dados (NETZER et al., 2011).

Contudo, como observado em Netzer e outros (2011), uma estimativa da acurácia da performance humana nesse banco de dados é de $98,0\%$. Para o desenvolvimento de um aplicativo que utilize o sistema proposto, como a construção de um mapa, é necessário pelo menos ter uma acurácia de nível humano, logo os $89,7\%$ obtidos não são suficientes para a validação deste sistema (GOODFELLOW et al., 2013). Desta forma, análises para a melhoria do sistema de *stacked sparse autoencoders* podem aumentar a acurácia na resolução do problema proposto.

O estado da arte do banco de dados SVHN utilizado no presente trabalho foi apresentado em Cubuk e outros (2019), tal que o mesmo é de $99,0\%$ de acurácia. Para alcançar esse resultado, os autores utilizaram de um tipo de Rede Neural Convolucional, uma *Wide Res-Net*, aplicada à metodologia de *AutoAugment*.

1.3 Objetivos

1.3.1 Objetivo Geral

Este trabalho tem como objetivo geral o estudo e implementação de uma rede *Stacked Sparse Autoencoder*, proposta em Xu e outros (2015), utilizada em conjunto à uma camada de classificação *Softmax*, apresentada em Netzer e outros (2011), para ser empregada no problema de classificação de números em imagens de cenas naturais. Com a alteração de parâmetros da rede implementada, será feita uma análise sobre a variação do desempenho da rede utilizando o método de pré-treinamento *Greedy Layer-Wise Training* (BENGIO, 2009), e de uma mudança das funções de ativação utilizadas, a fim de maximizar o desempenho da rede neural.

1.3.2 Objetivos Específicos

Os objetivos específicos para esse trabalho são: (*i*) Realizar uma revisão bibliográfica de trabalhos que estudam o problema de detecção e classificação de números em imagens baseados em arquiteturas de *autoencoders*; (*ii*) Obter o banco de imagens para estudo e análise do problema; (*iii*) Estudar e implementar as arquiteturas *stacked sparse autoencoders* com uma camada *softmax* e a integração dos mesmos; (*iv*) Fazer a variação de função de ativação e do uso do método

de pré-treino e avaliar as alterações causadas no desempenho; (*v*) Validar e testar o sistema implementado utilizando a arquitetura de maior desempenho.

1.4 Estrutura do Texto

O presente trabalho está estruturado de forma que inicia-se com a introdução, tal que este capítulo inicial tem como objetivo contextualizar a problemática abordada por esse trabalho, apresentando o problema, indicando suas aplicações práticas no cotidiano e ideias iniciais sobre a solução deste. Em adição, são expostos os objetivos da realização deste projeto de graduação. Em seguida, é apresentado o referencial teórico, sendo que neste capítulo são apresentados os conceitos fundamentais necessários para o entendimento do sistema proposto, tais quais: Redes Neurais Artificiais, Paradigmas de Aprendizagem, *Autoencoders*, *Sparce Autoencoders*, a metodologia *Greedy Layer-wise training*, dentre outros.

Posteriormente, tem-se o capítulo dedicado à solução proposta, dado que neste serão descritas todas as etapas da técnica proposta para a resolução do problema abordado e, em seguida, é apresentado o capítulo de resultados, onde são descritos os resultados obtidos para cada análise realizada. Por fim, no capítulo final deste trabalho, são indicadas todas as conclusões relativas aos resultados obtidos, além de apresentar propostas de melhorias para estudos futuros.

2 EMBASAMENTO TEÓRICO

2.1 Introdução

Este capítulo tem como objetivo expor os conceitos teóricos básicos necessários para o entendimento deste trabalho. Portanto, inicialmente, é definido o conceito de Redes Neurais Artificiais, sequenciado por dois paradigmas de aprendizado, sendo esses o aprendizado supervisionado e o não-supervisionado. Em seguida, o conceito de uma arquitetura *Autoencoder* é descrito, iniciando pela exposição de sua estrutura básica e, logo após, um resumo histórico. Posteriormente, são analisadas configurações específicas de *autoencoders*, abordando os conceitos *undercomplete* e *overcomplete* e explicitando sobre regularizadores. Logo após, é descrito o *Sparce Autoencoder*, iniciando com uma explicação geral sobre sua estrutura e finalizando com uma representação do conceito de penalidade inserido. Finalmente, discorre-se sobre um método de treinamento de *Autoencoders* multicamada denominado *Greedy Layer-wise Training* e uma breve descrição da metodologia de classificação de números em imagens naturais.

2.2 Redes Neurais Artificiais

Redes Neurais Artificiais (RNA) são modelos definidos como combinações de neurônios artificiais. O neurônio artificial utilizado na composição de uma RNA é baseado em um modelo que tenta simular o funcionamento das células do sistema nervoso. Uma vez que o estímulo de saída corresponde à combinação linear das entradas do neurônio passada por uma função não linear denominada como função de ativação. Cada entrada está associada a um peso correspondente à sua importância em relação à saída. Tal modelo, mostrado na Figura 1, foi inicialmente apresentado em McCulloch e Pitts (1943) usando uma função degrau como função de ativação.

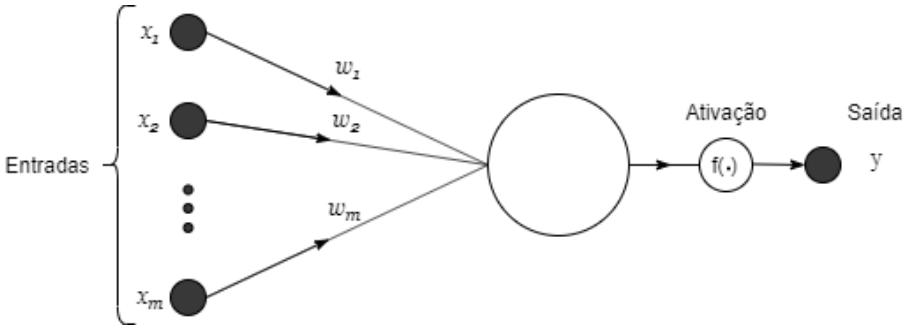
Na Figura 1, observa-se que: os pesos sinápticos são representados por $w_1, w_2, \cdots, w_m$; as entradas são denotadas por $x_1, x_2, \cdots, x_m$; o viés como b; a função de ativação como f. Desta forma, a saída no neurônio artificial é descrito pela equação (2.1).

$$y = f(\sum_{i=1}^{m} w_i x_i + b) \tag{2.1}$$

Toda RNA apresenta duas facetas fundamentais: (i) a arquitetura da RNA, como os neurônios

artificiais estão organizados; (*ii*) o paradigma de aprendizado, como é feito o ajuste dos pesos associados a cada neurônio artificial.

Figura 1 – Modelo matemático figurado de um neurônio



Fonte: Produção do próprio autor.

2.3 Arquitetura de uma RNA

Normalmente, os neurônios são organizados em camadas. Diferentes camadas podem aplicar diferentes funções de ativação em suas entradas. As camadas formam uma RNA e são organizadas da seguinte forma: (*i*) uma camada de neurônios de entrada, que não é composta de neurônios propriamente; (*ii*) uma ou mais camadas intermediárias ou ocultas de neurônios interligados, formando a estrutura central de processamento da RNA; (*iii*) uma camada de neurônios de saída, conectada aos neurônios da última camada oculta, gerando a resposta final da RNA. O sentido do fluxo de informação é da primeira (entrada), para a última camada (saída).

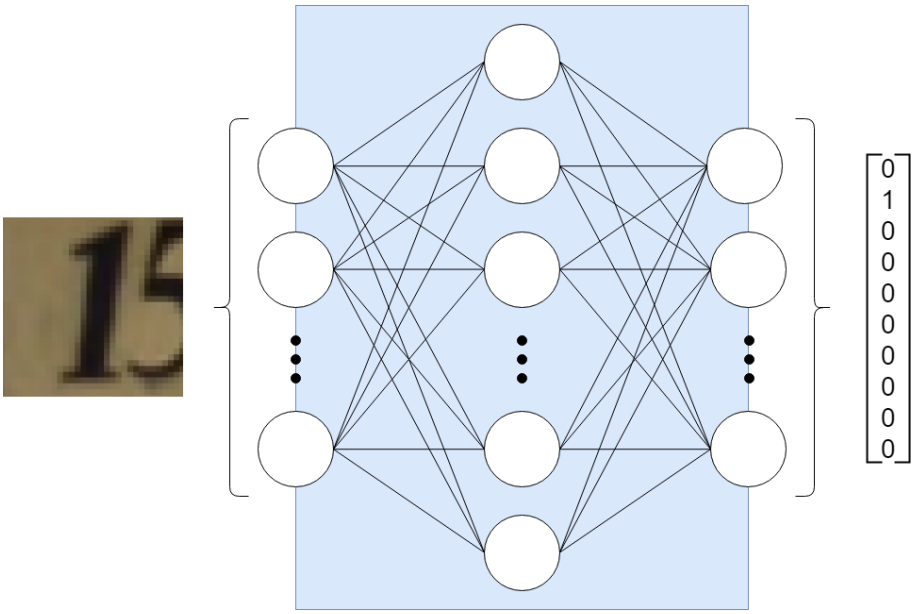
2.4 Paradigmas de Aprendizado

Os paradigmas de aprendizado são divididos em duas formas básicas, sendo essas o aprendizado supervisionado e aprendizado não-supervisionado.

- **Aprendizado Supervisionado**. No aprendizado supervisionado, toda amostra é composta pelo par: valor de entrada e valor desejado de saída. Dessa forma, dado um conjunto de treino $\mathscr{T}$, suas n amostras são dadas pelos pares (x_p, y_p), ou seja: $\mathscr{T} = \{(x_p, y_p)\}_{p=1}^{n}$. Por conseguinte, o desempenho da rede é calculado a partir da comparação entre a saída esperada e a saída obtida para cada valor de entrada, e a diferença entre essas é denominada de erro, o qual é informado à rede para o ajuste dos pesos a fim de melhorar suas futuras respostas. Neste paradigma, faz-se necessário ter conhecimento prévio do comportamento

desejado da rede quanto aos valores de entrada usados no treinamento (FERNEDA, 2006). Um classificador de imagens de algarismos faz uso direto do paradigma de aprendizado supervisionado para seu treinamento, tendo como entrada uma imagem e como saída qual algarismo está contido na imagem de entrada, como observado na Figura 2;

Figura 2 – Modelo figurativo de um classificador de imagens de algarismos com saída codificada em *One-Hot-Encoding*



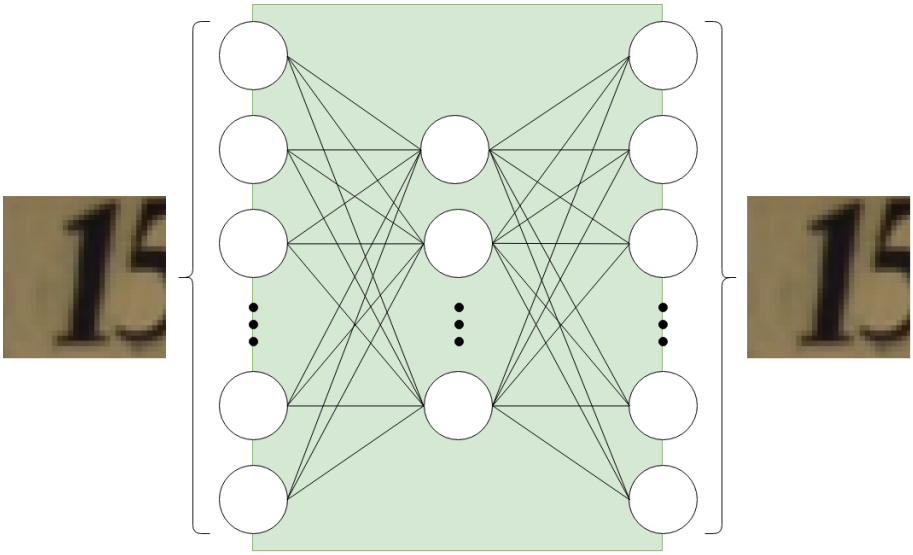
Fonte: Produção do próprio autor.

- **Aprendizado Não-supervisionado**. Diferentemente do paradigma supervisionado, este baseia seu conjunto de treinamento apenas nos valores de entrada (x_p), tal que a tarefa de aprendizado é descobrir correlações entre os exemplos de treino apresentados baseados em suas características e distribuições. Assim, $\mathscr{T} = \{(x_p)\}_{p=1}^{n}$. Em decorrência de não ter determinado o comportamento esperado para o modelo, o número de classes, ou categorias, para classificação não está definido a priori. Mediante a isso, torna-se necessário que a rede encontre atributos estatísticos relevantes, criando representações próprias dos estímulos que circulam na rede, agrupando-os (RAUBER, 1998).

A utilização de uma etapa de aprendizado não-supervisionado, antes da etapa supervisionada, para inicialização dos pesos da RNA é capaz de melhorar significantemente o desempenho da rede (HINTON; OSINDERO; TEH, 2006). Do mesmo modo, algumas vantagens do uso de etapas de pré-treinamento não-supervisionado em conjunto com arquiteturas multicamadas foram mostradas nos trabalhos propostos por Bengio e outros (2007) e Poultney e outros (2007), na qual focavam em estratégias para a construção de arquiteturas profundas.

Etapas de pré-treinamento de RNA atuam como espécies de regularizadores. Para modelos suficientemente grandes, essas são capazes de de diminuir a média e variância dos erros de teste, tal que esses efeitos são mais pronunciados quanto mais profunda a rede for. Em adição, essa prática acrescenta tanto robustez, quanto melhora na generalização para a base de dados. Todavia, pré-treinamento pode piorar o desempenho de arquiteturas com poucas camadas ocultas e reduzido número de neurônios (ERHAN et al., 2009).

Figura 3 – Modelo figurativo de um compressor e decompressor de imagens



Fonte: Produção do próprio autor.

Um *autoencoder*, explicado em detalhe na seção 2.5, normalmente utiliza do paradigma de aprendizado não-supervisionado para seu treinamento, por exemplo, recebendo imagens compostas por uma sequência de píxeis na entrada e fornecendo como saída uma reconstrução da entrada após realizar a compressão desta, como observado na Figura 3. A partir da imagem reconstruída, é feita uma comparação com a imagem de entrada, obtendo assim o erro de reconstrução da rede.

2.5 *Autoencoders*

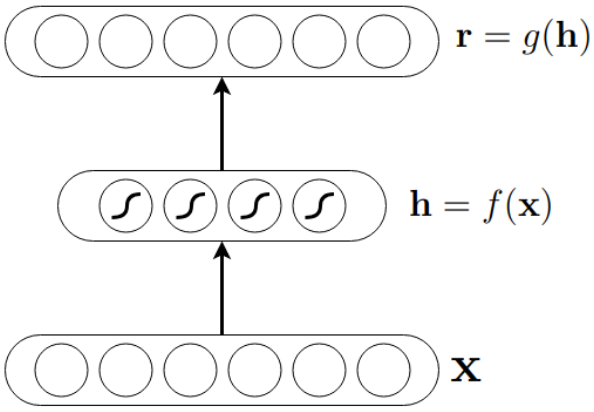
Um *autoencoder* é definido como uma rede neural que é treinada especificamente para reproduzir sua entrada na saída e frequentemente tem como objetivo obter uma redução dimensional da informação inserida. A arquitetura de um *autoencoder* é tipicamente decomposta nas seguintes partes, ilustrada na Figura 4:

- uma entrada, $\mathbf{x}$;
- uma função codificadora, f;

- o “código”, ou representação interna, $\mathbf{h} = f(\mathbf{x})$;
- uma função decodificadora, g;
- uma saída, também denominada “reconstrução”, $\mathbf{r} = g(\mathbf{h}) = g(f(\mathbf{x}))$;
- uma função custo $L(\mathbf{r}, \mathbf{x})$ que mede o quão precisa é a reconstrução $\mathbf{r}$ dado uma entrada $\mathbf{x}$.

O objetivo é minimizar o valor esperado da função de custo a partir de um conjunto de treinamento $\mathscr{T}$ não-supervisionado.

Figura 4 – Esquema de *autoencoder* generalizado



Fonte: Produção do próprio autor.

Porém, há peculiaridades no aprendizado de um *autoencoder*, tal que, ao aprender a copiar perfeitamente toda entrada para sua saída, apenas transmitindo os valores de uma camada para a outra, este não extrai os aspectos mais importantes dos dados inseridos em sua camada de entrada. Adicionar restrições na arquitetura de um *autoencoder* faz com que o mesmo não consiga apenas transferir os dados de entrada para sua saída, obrigando que este aprenda propriedades importantes dos dados para saber como reconstruí-los.

Essas restrições podem ser feitas por penalidades ou estabelecendo limites na arquitetura da rede. Ao limitar o número de unidades das camadas ocultas da rede neural como sendo inferior ao tamanho da entrada, o sistema é forçado a aprender uma versão compacta desta. Neste caso, a rede é denominada *Undercomplete Autoencoder*.

Quando há alguma correlação entre as entradas, esse modelo pode ser capaz de identificar essa relação. *Autoencoders* simples frequentemente acabam por aprender uma representação de dimensão reduzida bem próxima à representação gerada por *PCA* (PEARSON, 1901). Alguns novos modelos de *autoencoders* utilizam funções de natureza estocástica para representar as funções do codificador e decodificador, ao invés de funções determinísticas. Há mais de três

décadas que vem se fazendo estudos com redes neurais utilizando a ideia de *autoencoders*. Assim, tem-se:

- Em LeCun e outros (1989), é indicado sobre a utilização de camadas ocultas com restrições para obter melhores resultados em reconhecimento de números em imagens;
- Em Hinton e Zemel (1994), é feito o treinamento de *autoencoders* utilizando o princípio de Tamanho de Descrição Mínima (MDL, do inglês *Minimum Description Length*);
- Em Xu e outros (2015), é utilizado de *Autoencoders* esparsos estacados para a detecção de câncer de mama;
- Em Sønderby e outros (2016), é usado um *Variational Autoencoders*, fazendo uso de gradientes descendentes estocásticos, para realizar a classificação de algarismos em imagens.

2.5.1 *Undercomplete Autoencoders*

A capacidade de um *autoencoder* de copiar a entrada na saída não apresenta significativa utilização, já que, espera-se que ao treinar essa rede neural focando na reconstrução da entrada na saída, os valores de **h**, camada oculta da Figura 4, representarão relações importantes dos dados de entrada. Uma forma de obter essas relações é através do uso de *Undercomplete Autoencoders*. Estes são *autoencoders* nos quais a dimensão de sua camada oculta **h** é inferior à dimensão da entrada **x**. Ao aprender uma representação comprimida da informação, ocasionada pela diferença de tamanho entre **h** e **x**, a rede é forçada a adquirir as características mais significativas dos dados de treinamento.

Undercomplete autoencoders mais eficientes podem ser obtidos ao utilizar funções não lineares no codificador e decodificador. Porém, se este aumento de não linearidades for em demasia, pode ocorrer do *autoencoder* aprender a copiar os dados sem extrair informação útil sobre a distribuição destes.

2.5.2 *Autoencoders* Regularizados

Um problema semelhante na utilização de *undercomplete autoencoders* que utilizam de não linearidades pode ocorrer no caso da construção de um *autoencoder* cujas camadas ocultas tem dimensões igual ou maior que a entrada, conhecido como *overcomplete*, pela possibilidades

desses apenas repassarem a entrada para a saída sem analisar suas relações.

Para evitar esse problema, *autoencoders* regularizados utilizam funções de custo que encorajam o modelo a obter outras propriedades, além de copiar a entrada para a saída, não necessitando assim de restringir a capacidade nem tamanho das estruturas internas da rede. Essas outras propriedades incluem esparsidade da representação, redução da derivada da representação e robustez a ruído ou a entradas incompletas.

Autoencoders regularizados podem ser obtidos de diversas formas, entre as de maior relevância tem-se: (*i*) *Sparse autoencoder* (penaliza ativações de $\mathbf{h}$) (XU et al., 2015); (*ii*) *Bottleneck autoencoder* (limitando a dimensionalidade de $\mathbf{h}$) (BOURLARD; KAMP, 1988); (*iii*) *Denoising autoencoder* (insere ruído na entrada $\mathbf{x}$) (VINCENT et al., 2010); (*iv*) *Contractive autoencoder* (penaliza o Jacobiano de f) (RIFAI et al., 2011); (*v*) *Variational autoencoder* (por modelagem probabilística) (PU et al., 2016). Neste trabalho é abordado sobre os *Autoencoders* regularizados utilizando esparsidade da ativação dos neurônios, através do uso de penalização.

Um *Sparse Autoencoder* é definido como um *autoencoder* regularizado que adiciona uma penalidade $\Omega(\mathbf{h})$, denominada penalidade por esparsidade (do inglês *sparsity penalty*), ao critério de treinamento do modelo em $\mathbf{h}$, e tem seu erro de reconstrução dado pela equação (2.2).

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h}) \tag{2.2}$$

Devido à capacidade destes para obter características dos dados de entrada, *sparse autoencoder* são frequentemente utilizados por tarefas que necessitam de formas de correlacionar conjuntos de dados, como nos classificadores.

A penalidade $\Omega(\mathbf{h})$ aumenta o valor de erro da função de custo proporcionalmente ao quão esparsa são as ativações da rede. Desta forma, a fim de minimizar a função de custo, essa é forçada a manter suas ativações próximas de zero, ativando apenas os neurônios necessários. Em vista disso, a utilização da penalidade $\Omega(\mathbf{h})$ ocasiona esparsidade da ativação dos neurônios presentes na camada oculta da rede, fazendo com que a rede selecione apenas características úteis para representar a entrada.

2.5.3 *Layer-Wise Training*

O método Treinamento Guloso por Camada (do inglês *Greedy Layer-Wise Training*), proposto em Hinton, Osindero e Teh (2006), foi introduzido para o treinamento de Redes de Crença Profunda (DBN, do inglês *Deep Belief Networks*) uma camada de cada vez. O princípio deste se baseia no treinamento de cada camada a partir da saída da camada anterior, tal que esta já tenha sido treinada. Tal método foi introduzido utilizando uma Máquina Restrita de Boltzmann (RBM, do inglês *Restricted Boltzmann Machines*) como base de cada camada. Tal método de inicialização de parâmetros por camada mostrou-se eficiente e promissor em uma série de outros trabalhos (BENGIO et al., 2007; RANZATO; BOUREAU; LECUN, 2007; LAROCHELLE et al., 2007).

Foi proposto em Bengio e outros (2009) uma simplificação de tal método que fosse mais adequado para *Stacked Autoassociators*, outra denominação de *Stacked Autoencoders*. Essa metodologia de treinamento proposta pode ser utilizada para fazer a inicialização não supervisionada das camadas da rede a fim de colocar os parâmetros dessas em uma região na qual um mínimo local ótimo possivelmente possa ser alcançado a partir do gradiente descendente.

O procedimento de treinamento *layer-wise* de um *Stacked Autoencoder* consiste em cinco etapas:

(i) Treinamento não supervisionado da primeira camada como *autoencoder* para minimizar o erro de reconstrução dos dados de entrada não alterado, observado na Figura 5 (a);

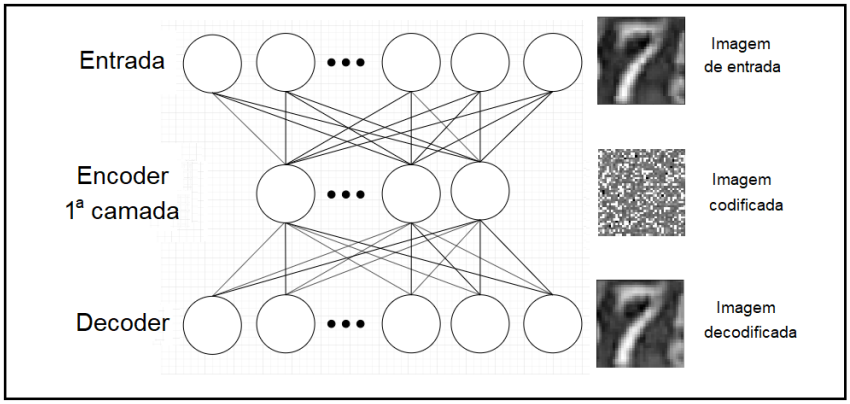
(ii) As saídas de cada neurônio da camada oculta já treinada é agora utilizada como entrada para a próxima camada em seu treinamento, também treinada sem supervisão para ser um *autoencoder*. Nessa etapa, os pesos das camadas já treinadas são mantidos fixos. Uma representação na primeira iteração é vista na Figura 5 (b);

(iii) Itera-se como na etapa (ii) para adicionar o número desejado de camadas;

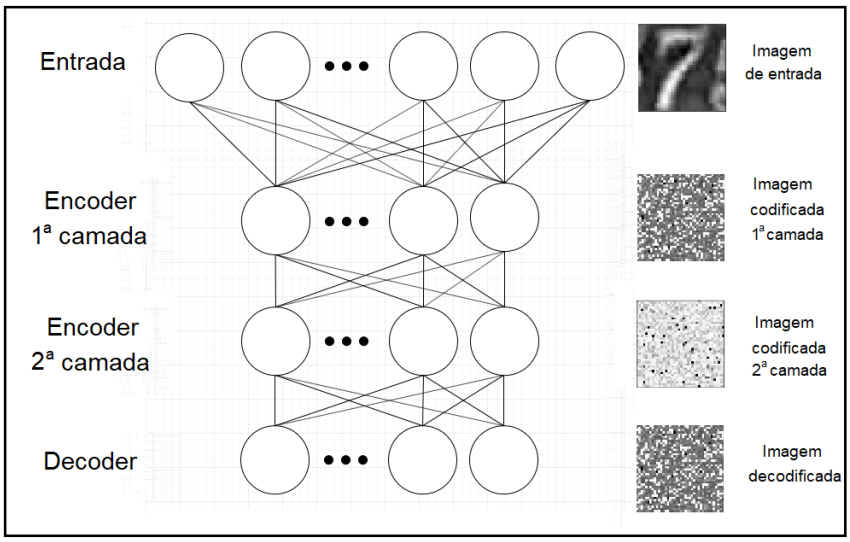
(iv) Utiliza-se a saída da última camada oculta como entrada para uma camada classificadora e inicializam-se seus parâmetros (aleatoriamente ou por treinamento supervisionado, mantendo o resto da rede com pesos fixos). A representação dessa etapa é dada pela Figura 5 (c), tendo os pesos das camadas 1 e 2 congelados;

(v) Faz-se o *fine-tuning* dos parâmetros com um treinamento supervisionado completo utilizando o erro de reconstrução global, a representação dessa etapa também é dada pela Figura 5 (c).

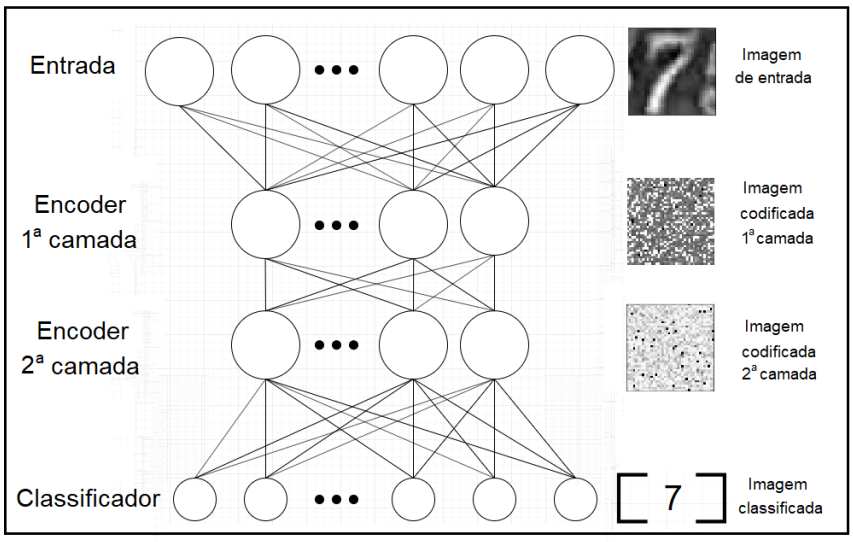
Figura 5 – Modelos utilizados no método de pré-treino *Greedy Layer-wise Training*, sendo (a) na etapa i, (b) nas etapas ii e iii e (c) nas etapas iv e v



(a)



(b)



(c)

Fonte: Produção do próprio autor.

2.6 Classificação de Números a partir de Imagens Naturais

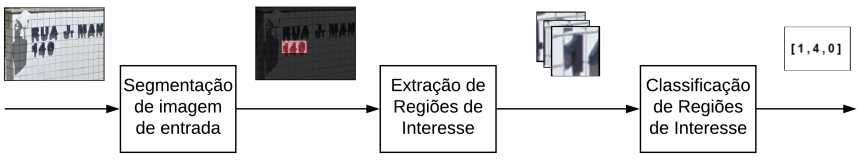
Um sistema para a classificação de números a partir de imagens naturais é, em termos gerais, um sistema de detecção de objetos, onde os objetos de interesse são os números presentes na imagem de entrada. Nesse contexto, tal sistema, mostrado na Figura 6, consiste na execução sequencial de três etapas: (i) segmentação da imagem de entrada; (ii) extração da *Region of interest* - ROI (tradução, região de interesse); (iii) Classificação da ROI. A seguir, uma breve explicação de cada etapa é efetuada.

(i) **Segmentação da imagem de entrada**. A partir da imagem original, determina-se uma imagem binária, na qual um objeto binário será todo conjunto de píxeis que correspondem a uma posição de um número, ou conjunto de números, na imagem original;

(ii) **Extração das ROI**. A partir da imagem binária gerada na etapa anterior são extraídas ROIs, representadas como Caixas Delimitadoras (BB, do inglês *bounding boxes*). Cada BB contém um objeto binário relacionado à posição de um número, ou conjunto de números na imagem original. Após a extração das BB, é feito um redimensionamento dessas, permitindo uma padronização no tamanho;

(iii) **Classificação dos ROI**. Considerando como entrada cada BB redimensionada, é treinado um modelo de aprendizado que tenha a capacidade de extrair características relevantes da ROI, e, usando essas características, classificar a ROI em uma de 10 possíveis classes, na qual cada classe corresponde a um dos dez dígitos do sistema numérico decimal.

Figura 6 – Fluxograma de um sistema para a classificação de números a partir de imagens naturais



Fonte: Produção do próprio autor.

3 PROPOSTA

3.1 Introdução

Nesse projeto é proposto um modelo para a classificação de números em imagens naturais, usando o conjunto de dados SVHN (o qual é explicado em detalhe na seção 4.2). Nesse sentido, as etapas de segmentação da imagem de entrada e extração das ROIs já estão implementadas na conjunto de dados alvo [1]. Logo, se faz necessário apenas desenvolver a etapa de classificação dos ROIs. Tal modelo está baseado na arquitetura *autoencoders* usando uma técnica de treinamento mista.

3.2 Arquitetura do Modelo

Aqui é proposto um modelo de aprendizado composto de duas etapas, uma etapa para extração de características e outra para classificação. Ambas são explicadas a seguir:

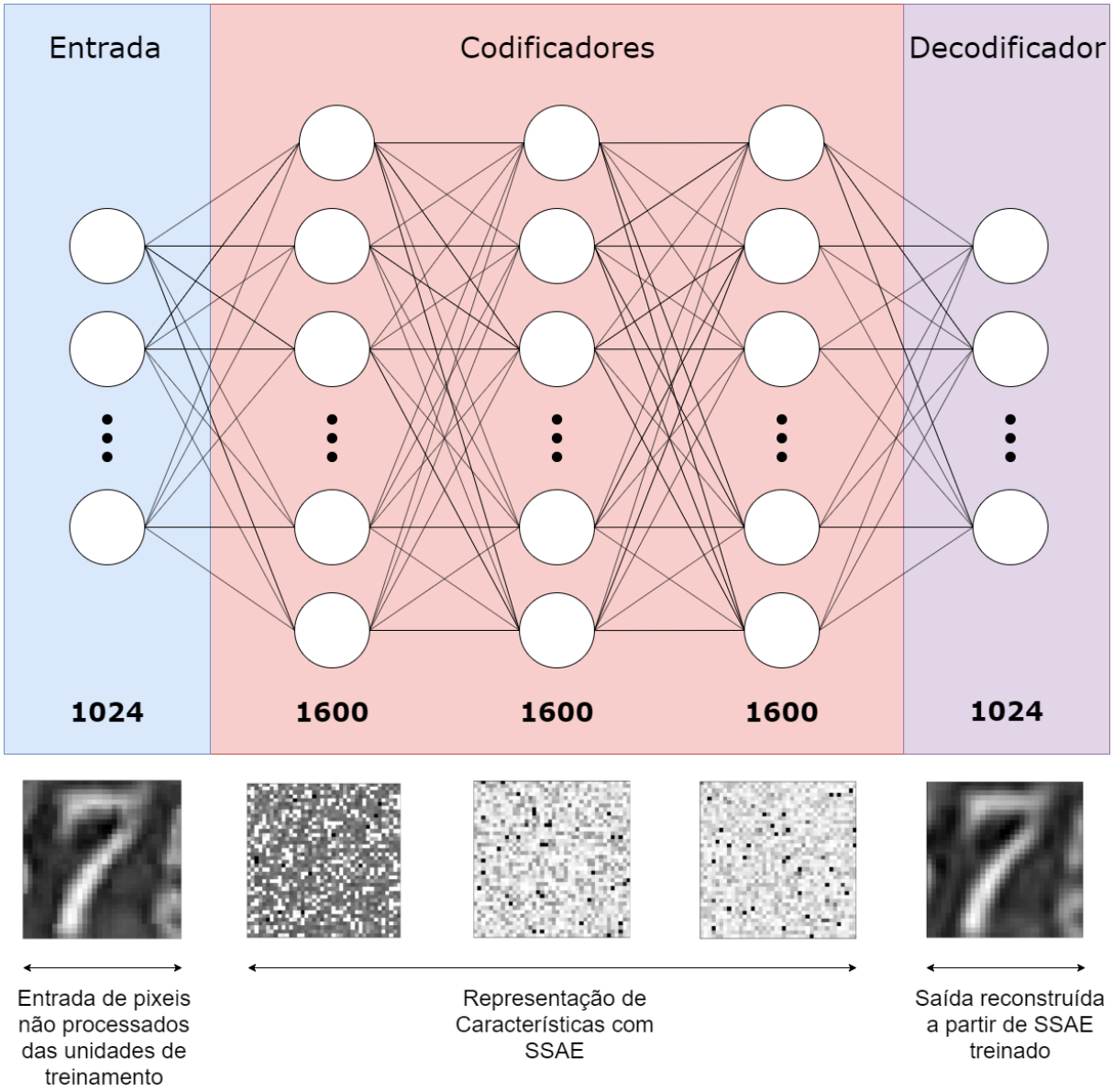
a) **Etapa de extração de características**. Composta por um *Stacked Sparse Autoencoder* para obtenção de uma representação dos dados de entrada do modelo. O treinamento do SSAE permite que esse aprenda características específicas dos dados nele inseridos. Na Figura 7 é ilustrada uma arquitetura construída e as representações dos dados em cada camada da rede;

b) **Etapa de classificação**. Composta por uma camada de classificação *Softmax* (SMC). Esse classificador é um modelo supervisionado que generaliza a regressão logística. O treinamento do classificador é feito utilizando a representação em alto nível gerada pela rede SSAE.

As etapas comentadas são integradas via um *Overcomplete* SSAE, mostrado na Figura 8, que é construído a partir de três tipos diferentes de camadas, sendo essas: camada de entrada (do inglês *input layer*), camada densa (do inglês *dense layer*) e um SMC . Especificamente, a arquitetura utilizada é constituída de um *input layer* de 1024 neurônios, tendo assim um neurônio para cada pixel da imagem, seguido por três *dense layers* empilhadas, finalizado com um SMC como última camada, tal que essa é um *dense layer* de 10 neurônios com função de ativação *Softmax*.

[1] O conjunto de dados SVHN já apresenta as imagens em formato de Dígitos Recortados (do inglês *Cropped Digits*) de 32×32 píxeis

Figura 7 – Ilustração de arquitetura de um SSAE e das múltiplas representações dos dados nas camadas internas



Fonte: Produção do próprio autor.

3.3 Treinamento do Modelo

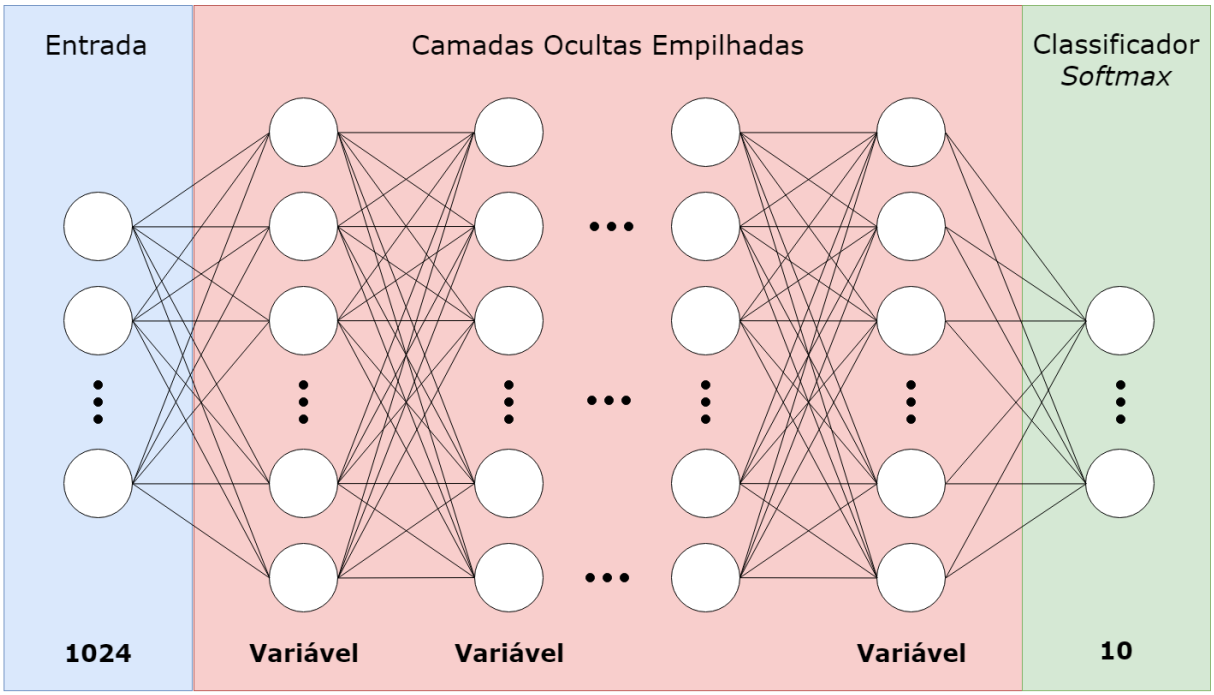
O treinamento do modelo é efetuado em duas etapas:

a) **Etapa de Treinamento não Supervisionado (pré-inicialização):** é implementado o método de treinamento não supervisionado *greedy layer-wise* de um SSAE, a fim de promover uma pré-inicialização adequada dos pesos da rede;

b) **Etapa de Treinamento Supervisionado:** Tendo o SSAE pré-inicializado, faz-se um ajuste fino nos pesos a partir de um treinamento supervisionado. Para tal, é aplicado ao modelo SSAE+SMC as imagens de treino do conjunto de dados alvo e via o algoritmo

Gradiente Descendente Estocástico (SGD, do inglês *Stochastic Gradient Descent*) o valor dos parâmetros do modelo são ajustados. Igualmente, com intuito de fazer uso do critério de regularização por esparsidade no modelo SSAE+SMC, adicionou-se uma penalidade Ω à função de custo do treinamento, sendo escolhida para isso o regularizador L1, também conhecido como regularizador LASSO, ou *Least Absolute Shrinkage and Selection Operator*.

Figura 8 – Ilustração da arquitetura de um *Overcomplete* SSAE



Fonte: Produção do próprio autor.

A etapa de treinamento não-supervisionado *Layer-wise* tem por objetivo fazer a inicialização dos pesos de cada camada da rede neural empilhada, ao invés de iniciar tais utilizando pesos aleatórios. Tem-se que o número de neurônios treináveis para uma arquitetura *Overcomplete* SSAE aumenta consideravelmente com o aumento do tamanho do vetor de entrada da rede. A rede SSAE apresentada na Figura 7 contém mais de 8 milhões de pesos treináveis e a inicialização destes influencia diretamente no tempo de convergência do treinamento.

A inicialização de tais parâmetros antes da etapa de treinamento supervisionado pode ser extremamente benéfico para a tarefa de classificação, definindo o ponto de partida do segundo treinamento a partir de uma rede neural na qual foram extraídas características significativas do conjunto de dados. Porém, não se tem bem definido quantitativamente a melhoria que o

pré-treinamento provê para esse tipo de rede em comparação a uma inicialização utilizando uma amostragem aleatória retirada de uma distribuição de probabilidade, tal como é feito via o método de inicialização de Glorot[2].

[2] Descrito em Glorot e Bengio (2010), o método de inicialização de Glorot, também conhecida como inicialização de Xavier, faz uso de uma distribuição de média zero e uma desvio padrão específico, σ, que depende dos números de unidades de entrada (f_{in}) e dos números de unidades de saída (f_{out}) do neurônio a ser inicializado, como observado na seguinte equação, $\sigma = \sqrt{\frac{2}{f_{in}+f_{out}}}$.

4 RESULTADOS

4.1 Introdução

Neste capítulo serão apresentados os resultados obtidos pela metodologia proposta. Dessa maneira, inicia-se como uma breve descrição do conjunto de dados utilizado, elucidando suas características, tamanho e divisão. Ademais, retratar-se-ão as métricas utilizadas para avaliar o desempenho das arquiteturas da rede neural proposta. Por fim, será feita a apresentação dos resultados práticos e uma análise de eficácia do modelo proposto nesse trabalho.

4.2 Conjunto de Dados SVHN

O conjunto de dados *Street View House Number* (SVHN) foi proposto em Netzer e outros (2011). De acordo com os criadores, para a construção deste conjunto foi utilizado o aplicativo *Google Street View* e o *framework Amazon Mechanical Turk*. No total, esse conjunto reúne mais de 600.000 caracteres rotulados que foram divididos em três subconjuntos menores, um de treino, com 73.257 dígitos, um de teste, com 26.032 dígitos, e um terceiro extra, também utilizado para treino, com 531.131 dígitos. A SVHN é fornecida em dois formatos distintos:

a) **Números Inteiros** – a versão original, com resoluções variáveis, diretamente de como estão aparecendo nas imagens. Cada imagem inclui um rótulo dos dígitos detectados e os BB para cada dígito do número;
b) **Dígitos Recortados** – todos os dígitos foram recortados e redimensionados para caber em uma resolução fixa de 32×32 píxeis. Esse redimensionamento foi feito de forma a não introduzir deformações nos dígitos.

Amostras do formato de dígitos recortados, que será utilizado nesse trabalho, são apresentadas na Figura 9. Também é visto que na abordagem de dígitos recortados são introduzidos dígitos de distração nos lados do dígito de interesse, o dígito central, aumentando o nível de dificuldade na classificação desse.

Foi necessário realizar uma etapa de pré-processamento nas imagens do conjunto de dados devido ao fato de todas as imagens serem coloridas, logo, fez-se uma conversão de RGB para escala de cinza, obtendo a média dos valores de cada camada. Além disso, subtraiu-se das

imagens uma média de todos os valores do conjunto de dados a fim de centralizar os dados e gerar gradientes menos oscilantes.

Figura 9 – Amostras do banco de dados SVHN

Fonte: Netzer e outros (2011b).

Nesse trabalho foi utilizada apenas a base extra para ambos treino e validação, e a base de teste foi usada sem alterações. Desta forma, a seguinte divisão do conjunto de dados foi feita: (*i*) o conjunto de treino foi conformado por 80% de conjunto treino extra; (*ii*) o conjunto de validação foi conformado por 20% de conjunto treino extra; (*iii*) o conjunto de teste foi o conjunto teste original.

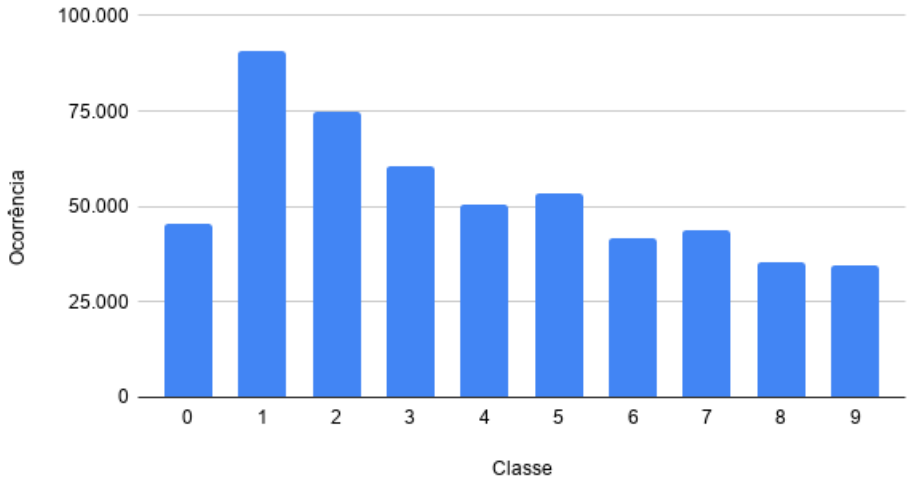
O conjunto de dados extra da base SVHN não é balanceado, isso é, algumas classes apresentam maiores ocorrências que outras. O número de amostras por classe pode ser observado no Gráfico 1.

4.3 Recursos Computacionais

4.3.1 Recursos de *Software*

Para a estruturação e treinamento da rede neural proposta foi utilizado o *framework Tensorflow*, o qual é um projeto de código aberto desenvolvido pela *Google Brain Team* (GOOGLE, 2011), que utiliza programação baseada em fluxo de dados em grafos (ABADI et al., 2016).

Gráfico 1 – Gráfico de ocorrência de classes no conjunto de dados Extra da base SVHN



Fonte: Produção do próprio autor.

Devido a sua estrutura básica e flexibilidade, *Tensorflow* é comumente utilizado em problemas de *Machine Learning*, mais especificamente, para aplicações de *Deep Learning*. Esse fato se dá por ele ser código aberto (do inglês *open source*), ter um grande suporte de desenvolvimento, possuir compatibilidade com utilização de GPU e ter sua programação baseada na linguagem Python. Além disso, toda a implementação das estruturas propostas foram feitas utilizando Keras, uma API de alto nível, programada em *Python*, que utiliza *Tensorflow* como *backend*, simplificando as operações necessárias para a criação, treinamento e análise das arquiteturas.

Ademais, para fazer o gerenciamento das dependências e para a construção de um ambiente de desenvolvimento virtual no sistema operacional Windows, utilizou-se a plataforma *open-source Anaconda Distribution*. Dentro desse ambiente virtual, utilizou-se de *notebooks* Jupyter para construção do código e execução dos treinamentos.

4.3.2 Recursos de *Hardware*

Para realizar o treinamento das redes neurais estabelecidas nesse projeto, foi utilizado um computador com as seguintes especificações: (*i*) sistema operacional Windows 10 Ultimate; (*ii*) processador Intel Core *i*5 – 8400, 2.80 GHz com 6 núcleos físicos; (*iii*) memória RAM de 16 GB, 2.400 MHZ, DDR4; (*iv*) unidade de armazenamento de 1 TB (disco rígido); (*v*) placa de vídeo Galax Geforce GTX 1070 Ti, com 2304 núcleos CUDA e 8 GB de memória dedicada.

4.4 Descrição dos Hiperparâmetros da Rede

Todas as arquiteturas testadas foram compostas por cinco camadas, sendo essas 1 camada de entrada de 1024 neurônios, três camadas ocultas, cada uma de 1600 neurônios, e uma camada de saída de 10 neurônios, contando com um total de 6.779.210 parâmetros treináveis.

Os hiperparâmetros da rede compartilhados entre todas as arquiteturas testadas estão listados no Quadro 1. Esses parâmetros foram escolhidos a partir de múltiplas execuções das arquiteturas avaliadas.

Quadro 1 – Hiperparâmetros compartilhados entre as arquiteturas analisadas

Parâmetro	Valor
Tamanho de *Batch*	128
Taxa de Aprendizado no Pré-Treino	0,005
Taxa de Aprendizado no Treinamento	0,01
Otimizador no Pré-treinamento	Adam
Otimizador no Treinamento	SGD
Função de Erro no Pré-treinamento	MSE
Função de Erro no Treinamento	*Weighted Categorical Crossentropy*

Fonte: Produção do próprio autor.

Por tratar-se de um classificador de múltiplas classes treinado sobre um conjunto de dados desbalanceado, foi utilizada a função de perda *Weighted Categorical Cross-Entropy*, que calcula o valor de entropia cruzada entre a predição do modelo e o valor esperado da classificação, considerando o peso inferido a aquela determinada classe.

4.5 Experimentos

4.5.1 Métricas

Para fazer a análise de desempenho das arquiteturas avaliadas foi utilizada a métrica de acurácia categórica ($Ac(\%)$). Tal métrica é definida pela equação (4.1), onde: C é o número total de classes; N é o número total de instâncias no conjunto em análise; N_i é o número de instâncias classificadas corretamente da classe i.

$$Ac = \sum_{i=1}^{C} \frac{N_i}{N} \tag{4.1}$$

4.5.2 Experimentos efetuados

Foram efetuados dois experimentos:

- **Primeiro experimento** – É efetuada a comparação da arquitetura SSAE + SMC com e sem pré-treino considerando que o SSAE usa a função de ativação ReLU/Sigmoide (APÊNDICE A) nas camadas ocultas. Basicamente, aqui é analisado o impacto no desempenho do modelo quando é usado o pré-treinamento baseado no método *greedy layer-wise training* apresentado na seção 2.5.3;
- **Segundo experimento** – É efetuada a comparação da arquitetura SSAE + SMC com diferentes taxas de penalidade do regularizador LASSO considerando que o SSAE usa a função de ativação ReLU/Sigmoide nas camadas ocultas. Este experimento esta orientado a analisar a efetividade da regularização por esparsidade no desempenho do modelo.

4.5.2.1 Primeiro experimento

A metodologia usada neste experimento foi:

a) **Arquitetura SSAE+SMC com pré-treinamento** – Cada camada oculta do SSAE foi inicializada usando o método *Greedy Layer-wise Training*. Essa etapa foi efetuada em 30 épocas. Em seguida, após remover a camada de decodificação do SSAE e adicionar o SMC, a rede foi treinada por mais 120 épocas;
b) **Arquitetura SSAE+SMC sem pré-treinamento** – Todas as camadas do SSAE e SMC são inicializadas aleatoriamente usando o método Glorot, efetuando-se o treinamento do modelo em 120 épocas.

Os resultados dos experimentos, sobre o conjunto de teste, é sumarizado na Tabela 1. Pode-se observar que o melhor resultado foi obtido usando a função de ativação ReLU sem pré-treinamento ($91,11\%$), seguido pelo uso da função de ativação Sigmoide com pré-treinamento ($89,53\%$).

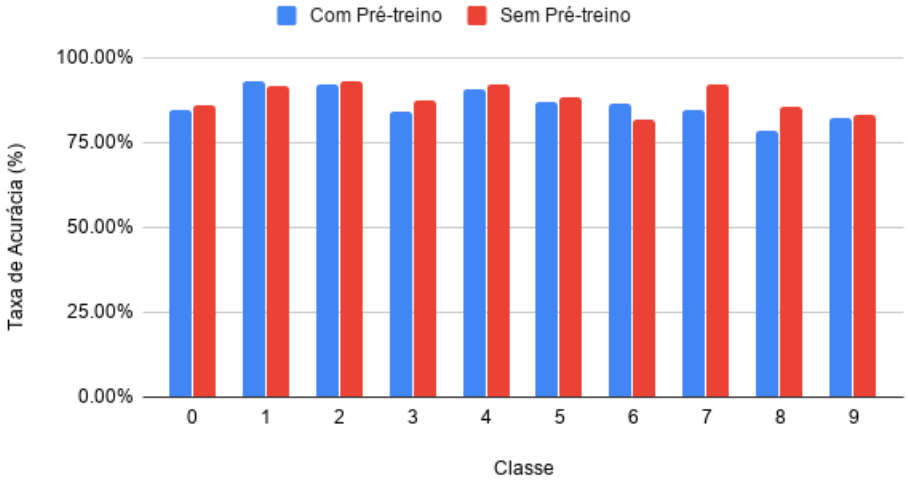
Os Gráficos 2 e 3 mostram as taxas de acurácia de classificação para cada uma das classes. Evidencia-se que: usar a função sigmoide uniformiza a acurácia entre as classes; o pré-treinamento favorece uma melhor acurácia na classe 1; a classe 9 é a que apresenta o maior desafio para uma correta classificação, como esperado devido ao desbalanceamento.

Tabela 1 – Comparativo da acurácia de classificação da base de dados de teste na utilização da técnica de pré-treino *Greedy Layer-wise Training*

	Ac(%)	
Metodologia	ReLU	Sigmoide
Com Pré-Treino	89,44	89,53
Sem Pré-Treino	91,11	83,22

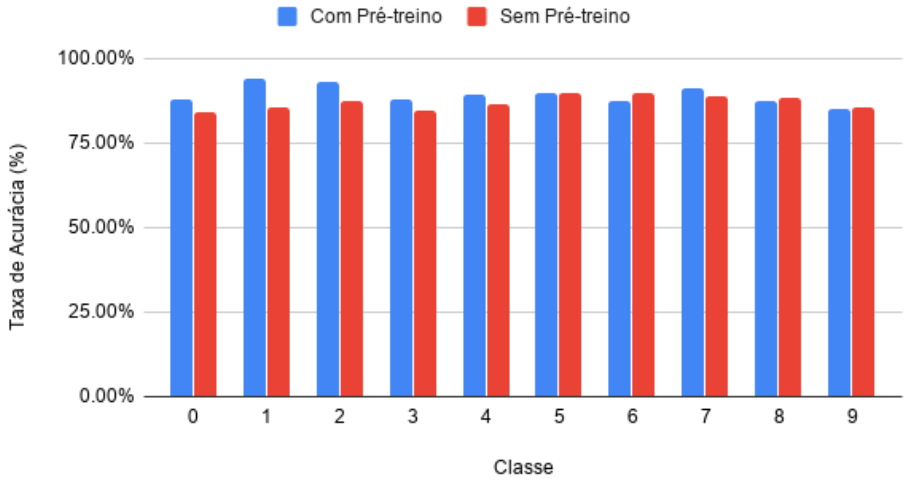
Fonte: Produção do próprio autor.

Gráfico 2 – Taxas de acurácia percentual por classe na tarefa de classificação da base de dados de teste utilizando função de ativação ReLU



Fonte: Produção do próprio autor.

Gráfico 3 – Taxas de acerto percentual por classe na tarefa de classificação da base de dados de teste utilizando função de ativação Sigmoide

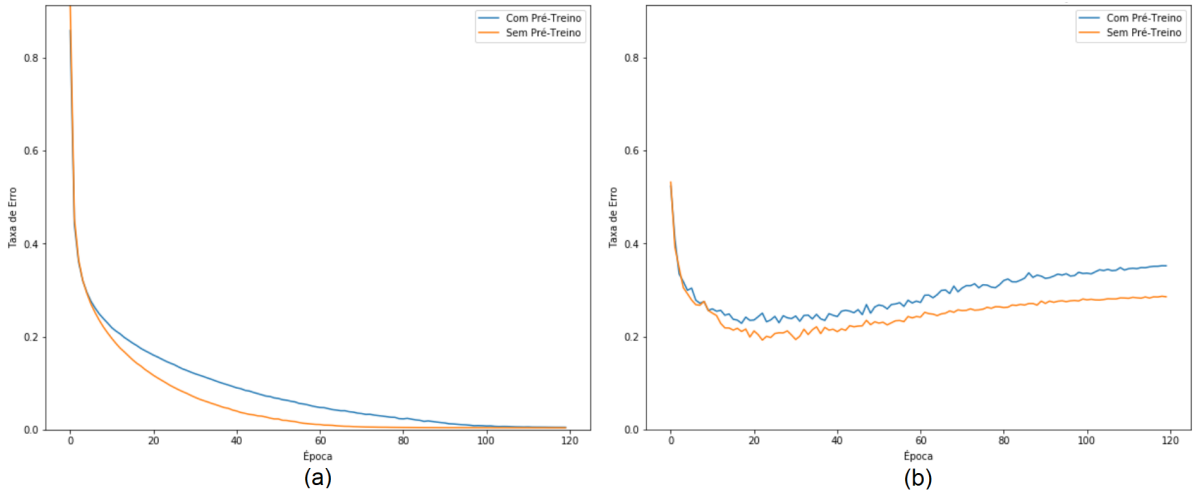


Fonte: Produção do próprio autor.

Observa-se na Tabela 1 que a utilização da metodologia de pré-treino na arquitetura que fez uso da função ReLU nas suas ativações fez com que essa tivesse um resultado inferior no conjunto de teste. Após uma comparação dos resultados do treinamento de ambas arquiteturas, observou-se que a rede com pré-treinamento apresentou um comportamento de *overfitting*, com um erro de

validação maior ao longo de todo o treino, quando comparado com o resultado de validação da arquitetura sem pré-treino, como observado nas curvas do Gráfico 4.

Gráfico 4 – Curvas de erro de treino (a) e validação (b) para ambas arquiteturas com e sem pré-treino utilizando ativação ReLU



Fonte: Produção do próprio autor.

A partir disso, para tentar diminuir esse comportamento, foi adicionada uma camada de *Dropout*, com $0,5$ de parâmetro, logo após do SSAE, efetuando-se um novo treinamento do SSAE+*Dropout*+SMC. Os resultados desse novo treino são mostrados no Gráfico 5. Observa-se que o uso de *Dropout* aumentou a acurácia do SSAE+SMC que fez uso do pré-treino. Na Tabela 2 nota-se que os erros de validação das arquiteturas com e sem pré-treino estão bem próximas com o uso dessa nova camada.

Tabela 2 – Comparativo da acurácia de classificação da base de dados de teste na utilização da técnica de pré-treino *Greedy Layer-wise Training* e uma camada *Dropout*

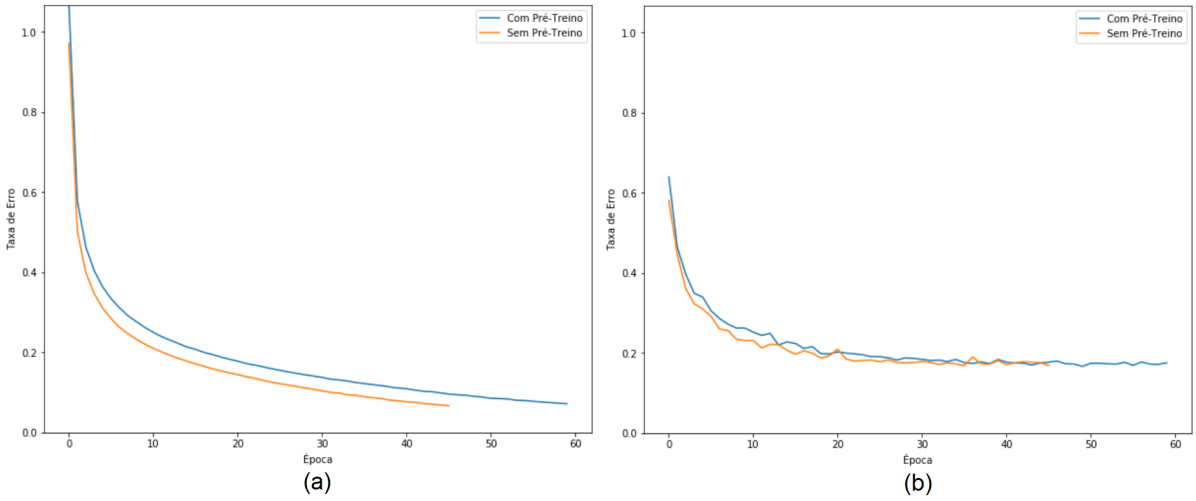
	Ac(%)	
Metodologia	Sem Droupout	Com Droupout
Com Pré-Treino	89,44	90,76
Sem Pré-Treino	91,11	90,37

Fonte: Produção do próprio autor.

Além disso, foi feito um experimento utilizando de camadas *Dropout* após cada uma das três camadas ocultas da redes, com parâmetros $0,2$, $0,3$ e $0,4$, respectivamente. Os resultados são observados nas curvas do Gráfico 6. Na Tabela 3 pode-se observar os valores de acurácia de classificação da base de dados de teste. Observa-se que ambas redes superaram 90% de acurácia,

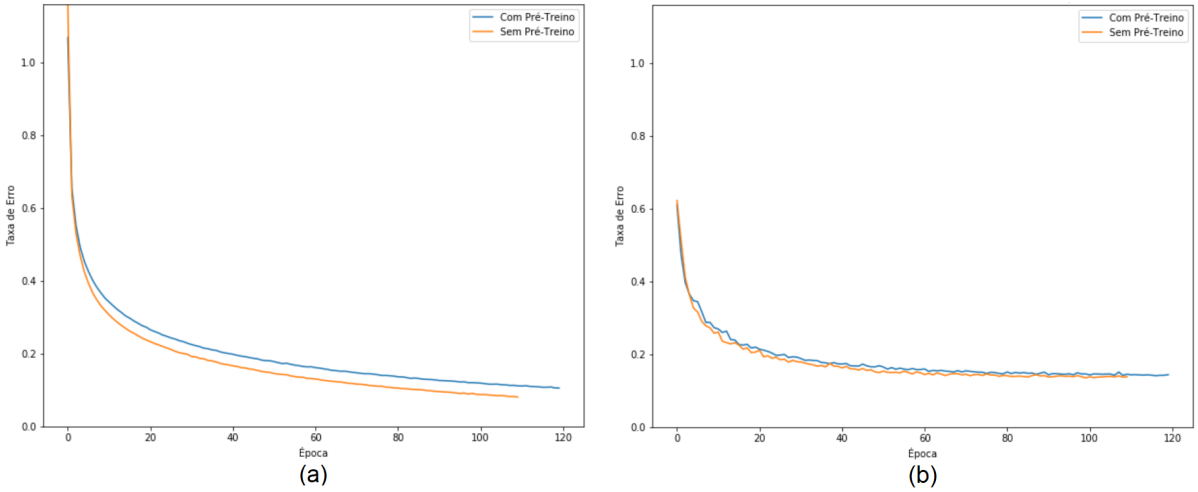
porém, novamente o uso do método de pré-treino não mostrou vantagem, alcançando acurácia de $91,71\%$.

Gráfico 5 – Curvas de erro de treino (a) e validação (b) para ambas arquiteturas com e sem pré-treino utilizando ativação ReLU e uma camada *Dropout*



Fonte: Produção do próprio autor.

Gráfico 6 – Curvas de erro de treino (a) e validação (b) para ambas arquiteturas com e sem pré-treino utilizando ativação ReLU e camadas *Dropout* após todas as camadas ocultas



Fonte: Produção do próprio autor.

Por conseguinte, analisa-se que o uso do método de pré-treinamento não se mostrou tão efetivo quando combinado com a função ReLU. É possível que tal comportamento ocorreu devido à uma transferência de erro de camada para camada no pré-treino, ocasionado pelo congelamento de pesos.

Tabela 3 – Comparativo da acurácia de classificação da base de dados de teste na utilização da técnica de pré-treino *Greedy Layer-wise Training* e camadas *Dropout* após todas as camadas ocultas

	$Ac(\%)$	
Metodologia	Sem Droupout	Com Droupout
Com Pré-Treino	89,44	91,45
Sem Pré-Treino	91,11	91,71

Fonte: Produção do próprio autor.

4.5.2.2 Segundo experimento

A metodologia do primeiro experimento foi repetida, adicionando-se na função de perda o regularizador L1 para provocar esparsidade, além de ser estabelecida 60 épocas na etapa do treinamento. Em adição, todas as arquiteturas utilizaram do método de pré-treino *Greedy Layer-wise training*.

Na Tabela 4 observa-se a comparação de valores de acurácia de classificação no conjunto de teste após o treinamento. A arquitetura com o maior percentual de acerto de classificação foi o modelo não regularizado $(90,35\%)$ usando a função de ativação ReLU. Observa-se que a utilização da penalidade do regularizador faz as arquiteturas convergirem mais rápido para um valor mínimo no treinamento, porém essas apresentam taxas de erro superiores à arquitetura não regularizada na validação.

Para as arquiteturas que fizeram uso de uma Sigmoide, observa-se que a medida que o valor de penalidade do regularizador aumenta a acurácia diminui. Todavia, o melhor resultado dentre as redes foi obtido no uso da penalidade de $1e^{-7}$.

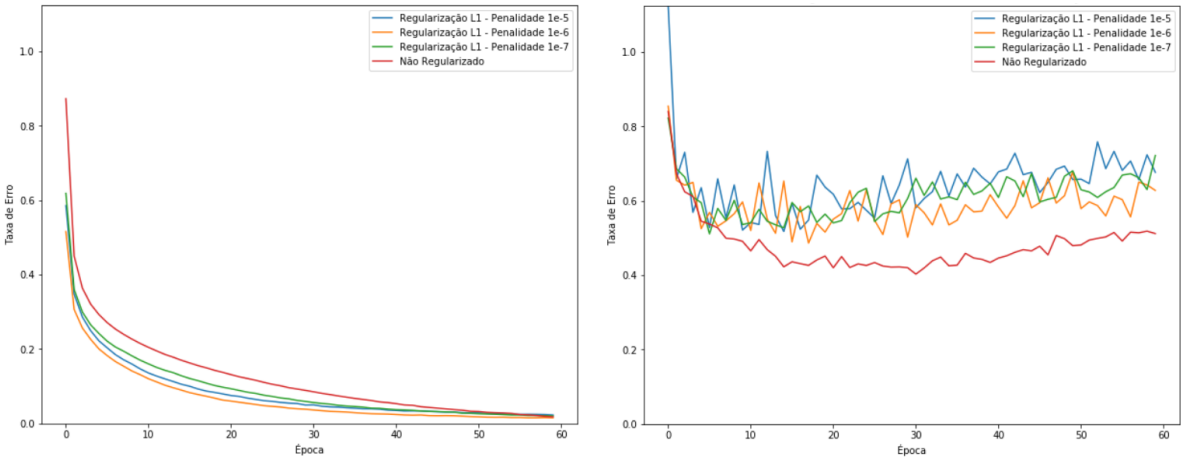
Tabela 4 – Comparativo da acurácia de classificação na utilização de regularização L1

	$Ac(\%)$	
Metodologia	ReLU	Sigmoide
Regularização L1 de 1e-5	88,76	86,10
Regularização L1 de 1e-6	88,96	88,34
Regularização L1 de 1e-7	87,58	89,08
Não Regularizado	90,35	88,65

Fonte: Produção do próprio autor.

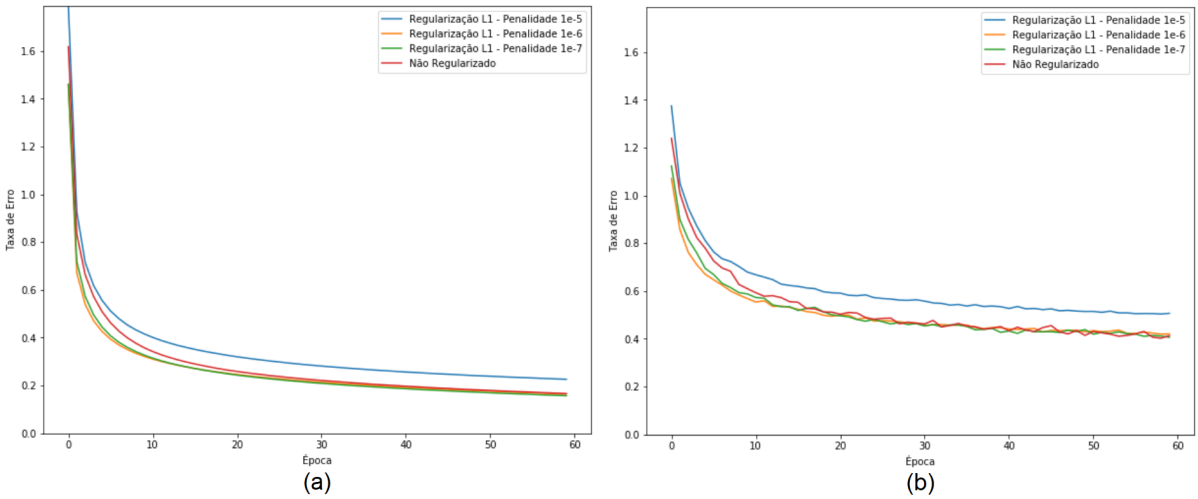
Nos Gráficos 7 e 8 são apresentadas as curvas de erro para validação e treino considerando o uso das funções de ativação ReLU e Sigmoide no SSAE.

Gráfico 7 – Curvas de erro de treino e (a) e validação (b) para ambas arquiteturas com diferentes níveis de esparsidade utilizando ativação ReLU



Fonte: Produção do próprio autor.

Gráfico 8 – Curvas de erro de treino (a) e validação (b) para ambas arquiteturas com diferentes níveis de esparsidade utilizando ativação Sigmoide



Fonte: Produção do próprio autor.

Para o caso da arquitetura SSAE com função de ativação ReLU (Gráfico 7), observa-se que as curvas da taxa de erro de validação decrescem a um ponto mínimo, próximo da 20^a época, e começam a crescer a partir dessa. Foram feitos testes treinando a rede por 20 épocas, porém essa não resultou em melhorias na acurácia de classificação da base de teste. Esse comportamento pode ser explicado pelo conjunto de validação não ser representativo para a base de teste utilizada.

Para o caso da arquitetura SSAE com função de ativação Sigmoide (Gráfico 8), é possível destacar uma diferença de comportamento das curvas de erro do modelo treinado com penalidade $1e^{-5}$ em comparação com as demais. Em adição, é evidente que a regularização limitou o treinamento da mesma, estagnando-a em uma taxa de erro acima de todas as outras redes.

4.5.3 Comparação com outros trabalhos

Alguns resultados apresentados nessa seção atingiram acurácia acima de 90%, tendo um máximo em $91,71\%$ para um SSVAE + SMC sem pré-treino e $91,45\%$ para um com pré-treino. Essas taxas podem ser diretamente comparadas a alguns resultados de outros trabalhos da literatura. A seguir, observa-se uma breve descrição dos métodos utilizados por alguns trabalhos relevantes que abordaram o problema de classificação do banco de dados SVHN.

a) ***Reading Digits in Natural Images with Unsupervised Feature Learning*** (NETZER et al., 2011). Esses utilizaram de uma arquitetura de *Stackes Sparse Autoencoder* em conjunto com o método de *Greedy Layer-wise Training* e alcançaram uma taxa de acurácia de $89,7\%$;

b) ***Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks*** (GOODFELLOW et al., 2013). Os autores fizeram uso de uma Rede CNN de 8 camadas de convolução, 1 camada localmente conectada e 2 camadas densas e alcançaram uma taxa de acurácia de $96,03\%$;

c) ***Winner-Take-All Autoencoders*** (MAKHZANI; FREY, 2015). Para esse trabalho foi usado um *sparse autoencoder* denominado *Stacked Convolutional Winner-Take-All Autoencoder* e obteve-se uma taxa de acurácia de $93,1\%$.

É importante observar que os melhores resultados foram dados pelas redes que fizeram uso de uma arquitetura convolucional, alcançando taxas de acurácia acima de 90%. Também é notável que o maior resultado obtido no presente trabalho foi superior ao apresentado em Netzer e outros (2011), ambas fazendo uso de um SSAE e utilizando a metodologia de pré-treinamento *Greedy layer-wise Training*.

5 CONCLUSÕES

O objetivo principal desse trabalho foi implementar uma técnica para classificação de imagens em ambientes naturais utilizando modelos baseados em RNA. Para isso, optou-se na utilização de uma abordagem baseada em *Deep learning*, tal que foi escolhida uma arquitetura na configuração de *Autoencoder* utilizando de regularizadores e do método de pré-treino das camadas ocultas, denominada *Stacked Sparse Autoencoder*. Em adição, fez-se uma análise da influência do método de pré-treinamento e da utilização de regularizadores no treinamento da rede proposta em combinação com a utilização de duas funções de ativações das camadas ocultas, sendo essas ReLU e Sigmoide.

A técnica proposta foi avaliada utilizando o conjunto de dados SVHN, atingindo, no melhor caso, uma acurácia de $91,71\%$. O referido valor se aproxima de resultados obtidos em outras abordagens na literatura, comprovando o potencial do modelo construído para a realização da tarefa descrita nesse trabalho. Todavia, devido à dependência do desempenho de uma rede neural nos hiperparâmetros utilizados na construção da mesma, o ajuste desses em experimentos futuros pode permitir alcançar melhores resultados.

Ao longo das análises realizadas em torno da utilização de funções de ativação ReLU e Sigmoide, observou-se que ambas possuem potencial para resolução da tarefa descrita, porém apresentando comportamentos diferentes nessa. Foi notado que a utilização do método de pré-treinamento possivelmente provocou um comportamento de *overfitting* nas arquiteturas que faziam uso da função de ativação ReLU em suas camadas ocultas, mesmo quando utilizado de diferentes penalidades do regularizador Lasso. Ao combater esse comportamento com a utilização de uma camada *Dropout*, observou-se um aumento na acurácia na tarefa de classificação, porém, ficou evidente que o pré-treino não resultou em melhora de performance significativa. Em contrapartida, a rede que fazia uso da função de ativação Sigmoide apresentou uma melhor inicialização de seus pesos ao aplicar a etapa de pré-treino, permitindo que essa começasse a convergir imediatamente no início do treinamento de classificação. Todavia, nenhum resultado obtido em arquiteturas que utilizavam de Sigmoide conseguiu ultrapassar 90% de acurácia.

Uma alternativa na estruturação do *Autoencoder* é a utilização de camadas de convolução no lugar das totalmente conectadas empregadas nesse projeto. Este recurso permite o uso de todas as três

camadas de cor fornecidas por imagem, além de ser mais computacionalmente eficiente e menos afetada pela translação e rotação dos algarismos no interior das imagens, como apresentado em Goodfellow e outros (2013).

Como incremento para a metodologia de regularização, a posteriori será estudada a influência da utilização da metodologia de *data augmentation* na etapa de treino e serão feitos novos estudos se aprofundando no uso da camada de *Dropout* ao longo da arquitetura, alterando parâmetros e número de camadas.

Além disso, futuramente, será feita uma análise na utilização da função de ativação Leaky ReLU, introduzido em Maas, Hannun e Ng (2013), devido a esse não limitar a saída do neurônio em zero, podendo trazer benefícios na etapa de treinamento.

Em desenvolvimentos futuros, também serão abordados a utilização de redes mais profundas, com mais camadas ocultas, e com períodos de treinamento não previamente definidos, ao aplicar o método de *early stopping*, permitindo a arquitetura treinar até alcançar um valor determinado de acurácia.

Finalmente, como anteriormente citado, existe a possibilidade de melhora do desempenho na tarefa de classificação das redes neurais abordadas através da otimização dos hiperparâmetros utilizados. Tais ajustes possibilitam uma melhora tanto na etapa de pré-treinamento das redes, quanto na de classificação.

REFERÊNCIAS BIBLIOGRÁFICAS

ABADI, M.; BARHAM, P.; CHEN, J.; CHEN, Z.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; IRVING, G.; ISARD, M.; KUDLUR, M. Tensorflow: A system for large-scale machine learning. *In*: USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 12., 2016, Savannah. **Proceedings [...].** Savannah: [s.n.], 2016, p. 265–283.

BENGIO, Y. Learning deep architectures for AI. **Foundations and Trends® in Machine Learning**, Now Publishers, v. 2, n. 1, p. 1–127, nov. 2009.

BENGIO, Y.; LAMBLIN, P.; POPOVICI, D.; LAROCHELLE, H. Greedy layer-wise training of deep networks. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 21., 2007, Vancouver. **Proceedings [...]**. Vancouver: [s.n.], 2007, p. 153–160.

BOURLARD, H.; KAMP, Y. Auto-association by multilayer perceptrons and singular value decomposition. **Biological Cybernetics**, Springer, v. 59, n. 4-5, p. 291–294, set. 1988.

CONNOR, P.; ROSS, A. Biometric recognition by gait: A survey of modalities and features. **Computer Vision and Image Understanding**, Elsevier, v. 167, p. 1–27, fev. 2018.

CUBUK, E. D.; ZOPH, B.; MANE, D.; VASUDEVAN, V.; LE, Q. V. Autoaugment: Learning augmentation strategies from data. *In*: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2019, California. **Proceedings [...]**. California: [s.n.], 2019, p. 113–123.

ERHAN, D.; MANZAGOL, P. A.; BENGIO, Y.; BENGIO, S.; VINCENT, P. The difficulty of training deep architectures and the effect of unsupervised pre-training. *In*: ARTIFICIAL INTELLIGENCE AND STATISTICS, 12., 2009, Flórida. **Proceedings [...]**. Florida: [s.n.], 2009, p. 153–160.

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. **Ciência da Informação**, SciELO Brasil, v. 35, n. 1, p. 25-30, ago. 2006.

GIBBS, J. W. **Elementary principles in statistical mechanics**. [S.l.]: Courier Corporation, 2014.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. *In*: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS, 13., 2010, Sardinia. **Proceedings [...]**. Sardinia: [s.n.], 2010, p. 249–256.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. *In*: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS, 14., 2011, Fort Lauderdale. **Proceedings [...]**. Fort Lauderdale: [s.n.], 2011, p. 315–323.

GOODFELLOW, I. J.; BULATOV, Y.; IBARZ, J.; ARNOUD, S.; SHET, V. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *In*: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, 2., 2014, Banff. **Proceedings [...]** Disponível em: https://arxiv.org/pdf/1312.6082.pdf. Acesso em: 12 dez. 2018.

GOOGLE. **Google Brain Team**. 2011. Disponível em: https://research.google/teams/brain. Acesso em: 25 mar. 2019.

HINTON, G. E.; OSINDERO, S.; TEH, Y. W. A fast learning algorithm for deep belief nets. **Neural Computation**, MIT Press, v. 18, n. 7, p. 1527–1554, jul. 2006.

HINTON, G. E.; ZEMEL, R. S. Autoencoders, minimum description length and helmholtz free energy. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 7., 1994, Denver. **Proceedings [...]**. Denver: [s.n.], 1994. p. 3–10.

IDE, H.; KURITA, T. Improvement of learning for cnn with relu activation by sparse regularization. *In*: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2017, Anchorage. **Proceedings [...]**. [S.l.]: IEEE, 2017. p. 2684–2691.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 25., 2012, Lake Tahoe. **Proceedings [...]**. Lake Tahoe: [s.n.], 2012. p. 1097–1105.

LAROCHELLE, H.; ERHAN, D.; COURVILLE, A.; BERGSTRA, J.; BENGIO, Y. An empirical evaluation of deep architectures on problems with many factors of variation. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 24., 2007, Corvalis. **Proceedings [...]**. Corvalis: ACM, 2007. p. 473–480.

LECUN, Y. Generalization and network design strategies. **Connectionism in Perspective**, Citeseer, v. 19, p. 143–155, jun. 1989.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 30., 2013, Atlanta. **Proceedings [...]**. Atlanta: [s.n.], 2013, p. 3.

MAKHZANI, A.; FREY, B. J. Winner-take-all autoencoders. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 28. 2015, Montreal. **Proceedings [...]**. Montreal: [s.n.], 2015, p. 2791–2799.

MARTINS, A.; ASTUDILLO, R. From softmax to sparsemax: A sparse model of attention and multi-label classification. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 33., 2016, New York. **Proceedings [...]**. New York: [s.n.], 2016, p. 1614–1623.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The Bulletin of Mathematical Biophysics**, Springer, v. 5, n. 4, p. 115–133, dez. 1943.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 27., 2010, Haifa. **Proceedings [...]**. Haifa: [s.n.], 2010, p. 807–814.

NETZER, Y.; WANG, T.; COATES, A.; BISSACCO, A.; WU, B.; NG, A. Y. Reading digits in natural images with unsupervised feature learning. *In:* NIPS WORKSHOP ON DEEP LEARNING AND UNSUPERVISED FEATURE LEARNING, 2., 2011, Granada. **Proceedings [...]**. Granada: [s.n.], 2011, p. 5-14.

PEARSON, K. On lines and planes of closest fit to systems of points in space. **The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science**, Taylor & Francis, v. 2, n. 11, p. 559–572, nov. 1901.

POULTNEY, C.; CHOPRA, S.; CUN, Y. L.; RANZATO, M. A. Efficient learning of sparse representations with an energy-based model. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 20., 2007, Vancouver. **Proceedings [...]**. Vancouver:[s.n.], 2007, p. 1137–1144.

PU, Y.; GAN, Z.; HENAO, R.; YUAN, X.; LI, C.; STEVENS, A.; CARIN, L. Variational autoencoder for deep learning of images, labels and captions. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 29., 2016, Barcelona. **Proceedings [...]**. Barcelona: [s.n.], 2016, p. 2352–2360.

RANZATO, M. A.; HUANG, F. J.; BOUREAU, Y. L.; LECUN, Y. Unsupervised learning of invariant feature hierarchies with applications to object recognition. *In*: COMPUTER VISION AND PATTERN RECOGNITION CONFERENCE, 7., 2007, Minneapolis. **Proceedings [...]**. Minneapolis: IEEE Press., 2007, p. 1-8.

RAUBER, T. W. Redes neurais artificiais. *In:* ENCONTRO REGIONAL DE INFORMÁTICA, 1998, Nova Friburgo. **Proceedings [...]**. Nova Friburgo: Sociedade Brasileira de Computação, 2005, p. 201-228.

RIFAI, S.; VINCENT, P.; MULLER, X.; GLOROT, X.; BENGIO, Y. Contractive auto-encoders: Explicit invariance during feature extraction. *In*: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 28., 2011, Bellevue. **Proceedings [...]**. Bellevue: OMNIPRESS, 2011, p. 833–840.

SHAH, S.; DEY, D.; LOVETT, C.; KAPOOR, A. A. High-fidelity visual and physical simulation for autonomous vehicles. *In*: FIELD AND SERVICE ROBOTICS, 11., 2018, Zurich. **Proceedings [...]**. Zurich: SPRINGER, 2018, p. 621–635.

SØNDERBY, C. K.; RAIKO, T.; MAALØE, L.; SØNDERBY, S. K.; WINTHER, O. Ladder variational autoencoders. *In*: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 29., 2016, Barcelona. **Proceedings [...]**. Barcelona: [s.n.], 2016, p. 3738–3746.

VINCENT, P.; LAROCHELLE, H.; LAJOIE, I.; BENGIO, Y.; MANZAGOL, P. A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. **Journal of Machine Learning Research**, v. 11, n. 12, p. 3371–3408, dez. 2010.

WANG, T.; WU, D. J.; COATES, A.; NG, A. Y. End-to-end text recognition with convolutional neural networks. *In*: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 21., 2012, Stockholm. **Proceedings [...]**. Stockholm: [s.n.], 2012, p. 3304–3308.

XU, J.; XIANG, L.; LIU, Q.; GILMORE, H.; WU, J.; TANG, J.; MADABHUSHI, A. Stacked sparse autoencoder (SSAE) for nuclei detection on breast cancer histopathology images. **IEEE Transactions on Medical Imaging**, v. 35, n. 1, p. 119–130, jul. 2015.

YE, Q.; DOERMANN, D. Text detection and recognition in imagery: A survey. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 37, n. 7, p. 1480–1500, nov. 2015.

APÊNDICE A – FUNÇÕES DE ATIVAÇÃO

A.1 Função de ativação ReLU

A fim de melhorar o desempenho de Máquinas de Boltzmann restritas, em Nair e Hinton (2010) introduziu-se unidades de retificação linear (ReLU, do inglês *rectified linear units*). Posteriormente, foi demonstrado em Glorot, Bordes e Bengio (2011) que funções de ativação ReLU nas camadas ocultas de uma rede conseguem melhorar a velocidade de aprendizado de variadas redes neurais profundas. Os gradientes da função ReLU nos valores positivos se tornam constantes e não desaparecem e assim, naturalmente, o problema de desaparecimento de gradiente consegue ser evitado ao usar tal função. Em virtude disso, unidades de retificação linear podem melhorar a velocidade do aprendizado de redes neurais profundas (IDE; KURITA, 2017).

Definida por $f(x) = max(x, 0)$, o uso da função ReLU foi uma importante descoberta que possibilitou o treinamento completamente supervisionado de redes neurais do estado da arte (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Redes neurais profundas com ReLU são mais facilmente otimizadas que redes que utilizam de unidades Sigmoide e Tangente Hiperbólica (tanh), devido aos gradientes conseguirem fluir quando a entrada da função é positiva.

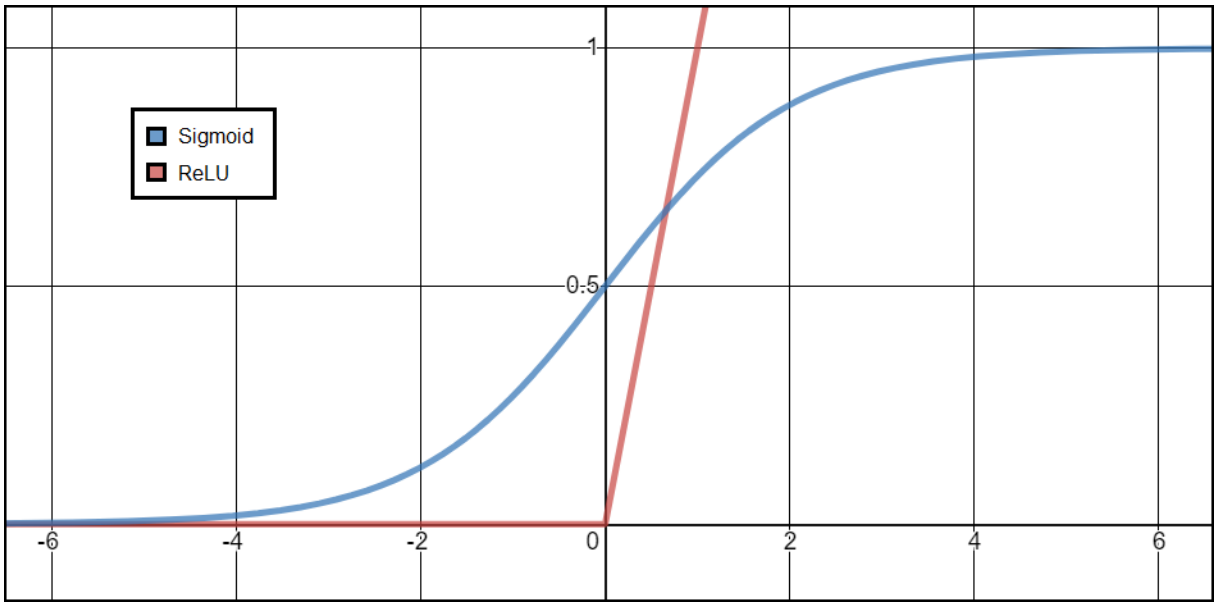
Na Figura 10 é observada a representação gráfica da função ReLU (a), ao lado da função de ativação Sigmoide. Observa-se que diferente desta segunda, a função ReLU não apresenta limite superior, porém ambas convergem para 0 para valores negativos de x.

A.2 Função de ativação Softmax

Também conhecida como Função Exponencial Normalizada (do inglês *Normalized Exponential Function*), a função de ativação Softmax, formalizada e popularizada no livro Gibbs (2014), é uma função que recebe como entrada um vetor K de números reais e normaliza-os em uma distribuição probabilística que consiste em K probabilidades.

Softmax é comumente utilizado como um mapa para conversão de vetores de pesos reais (e.g., pontuação de rótulos) em distribuição probabilística (e.g. probabilidade posterior de rótulos).

Figura 10 – Representação gráfica das funções ReLU e Sigmoide



Fonte: Produção do próprio autor.

Este utilizando do simplex de (K-1) dimensões

$$\Delta^{K-1} := p \in \mathbb{R}^K | 1^\intercal p = 1, p \geq 0 \tag{A.1}$$

para mapear entradas $\mathbb{R}^K$ em saídas Δ^{K-1}. A função softmax pode ser definida por componente como,

$$softmax_i(z) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{A.2}$$

Uma limitação da transformação Softmax é que a distribuição probabilística resultante vai sempre ter suporte completo, i.e., $softmax_i(z) \neq 0$ para todo z e i. Tal fato é uma desvantagem em aplicações na qual uma distribuição de probabilidade esparsa é desejada (MARTINS; ASTUDILLO, 2016).