UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

RAFAEL BIAZUS MANGOLIN

CLASSIFICAÇÃO DE IMAGENS DE COMIDA COM REDES NEURAIS CONVOLUCIONAIS

TRABALHO DE CONCLUSÃO DE CURSO

MARINGÁ
2017

UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

RAFAEL BIAZUS MANGOLIN

CLASSIFICAÇÃO DE IMAGENS DE COMIDA COM REDES NEURAIS CONVOLUCIONAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Informática, do Centro de Tecnologia, da Universidade Estadual de Maringá.

Orientador: Yandre Maldonado e Gomes da Costa
Universidade Estadual de Maringá

MARINGÁ
2017

AGRADECIMENTOS

A minha família pelo incentivo e apoio incondicional.

Aos meus amigos de graduação, pelas trocas de ideias, críticas e auxílios.

Ao meu orientador Dr. Yandre Maldonado e Gomes da Costa, por me apresentar a área de sistemas inteligentes, pelo auxílio e sugestões no desenvolvimento deste trabalho.

A todos que direta ou indiretamente fizeram parte da minha formação.

RESUMO

MANGOLIN, Rafael. Classificação de imagens de comida com redes neurais convolucionais. 2017. 28 f. Trabalho de Conclusão de Curso – Departamento de Informática, Universidade Estadual de Maringá. Maringá, 2017.

Esta monografia aborda o problema de classificação de imagens de comida, que está inserido na área de reconhecimento de padrões. A tarefa de classificação de imagens foi realizada neste trabalho utilizando redes neurais convolucionais (*convolutional neural networks*, CNN), uma técnica de *deep learning*. A CNN é uma rede neural de muitas camadas na qual é aplicada a operação de convolução. A utilização de CNN vem melhorando o estado da arte na área de classificação de imagens desde 2012. Um dos maiores problemas encontrados para o uso de CNN em tarefas de classificação é o *overfitting*, que ocorre devido a uma quantidade escassa de amostras ou dada a profundidade da rede criada. Para a solução desse problema, neste trabalho, foi realizado o *fine tunning* na taxa de *dropout* e foram utilizadas duas técnicas: o *data augmentation* e a inicialização dos pesos da rede a partir de uma rede treinada. A rede neural proposta neste projeto tem como base de estrutura a CNN *AlexNet*. A base de dados utilizada foi formulada com amostras retiradas das bases *ImageNet* e *Food-101*, contendo 16 classes de imagens. O melhor resultado foi obtido quando aplicada as técnicas de *data augmentation*, *dropout* e inicialização dos pesos, resultando em uma acurácia de 74,56%.
Palavras-chave: Classificação de imagens. Rede neural convolucional. *Deep learing*.

ABSTRACT

MANGOLIN, Rafael. Food images classification with convolutional neural networks. 2017. 28 f. Trabalho de Conclusão de Curso – Departamento de Informática, Universidade Estadual de Maringá. Maringá, 2017.

This work addresses the problem of food image classification, which is inserted in the area of pattern recognition. The task of images classification was perfomed in this work using convolutional neural networks (CNN), a deep learning technique. CNN is a neural network of many layers in which the convolution operation is applied. The use of CNN has been improving the state of the art in the area of image classification since 2012. One of the biggest problems encountered for the use of CNN in classification tasks is the overfitting, that occurs due to a small amount of samples or the depth of the network created. To solve this problem, in this work, the fine-tuning was performed at the dropout rate and two techniques were used: the data augmentation and the initialization of the weights of the network from a trained network. The neural network proposed in this project is based on the CNN AlexNet. The database was formulated with samples taken from the ImageNet and Food-101 datasets, containing 16 classes of images. The best result was obtained when applying data augmentation, dropout and weight initialization techniques, resulting in an accuracy of 74.56%.

Keywords: Image classification. Convolutional neural network. Deep learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN	*Convolutional neural network*
CUDA	*Compute Unified Device Architecture*
GPU	*Graphics Processing Unit*
GSIGMA	Grupo de Sistemas Inteligentes de Manufatura
MLP	*Multilayer perceptron*
ReLu	*Rectified linear unit*
RGB	Abreviação para o sistema aditivo de cores vermelho (*Red*), verde (*Green*) e azul (*Blue*)
SVM	*Support vector machine*
UFSC	Universidade Federal de Santa Catarina

SUMÁRIO

1 INTRODUÇÃO

O problema de classificação de imagens utilizando técnicas tradicionais de aprendizado de máquina requer conhecimento e experiência no domínio que será classificado, para extrair características (do inglês *features*) relevantes, que são utilizadas para obter a classificação. Assim, utilizando técnicas que não dependem de extratores especializados de *features*, como redes neurais convolucionais (CNN, *convolutional neural networks*), se torna mais fácil desenvolver modelos eficazes de aprendizado de máquina para novos conjuntos de dados. Nesse projeto foi utilizada CNN para a classificação de imagens com comida, assim não será preciso encontrar descritores especializados nesse domínio.

Segundo LeCun, Bengio e Hinton (2015), o *deep learning* vem sendo utilizado para resolver problemas computacionalmente complexos que temos no nosso dia-a-dia, e seu uso vem evoluindo o estado-da-arte de muitas áreas.

Um dos grandes problemas encontrados no uso de CNN é o *overfitting* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) que ocorre devido a muitos parâmetros na rede ou uma base de dados pequena. A utilização de técnicas como o *dropout* e o *data augmentation* vem sendo utilizas para reduzir o *overfitting* (SRIVASTAVA et al., 2014).

Neste projeto utilizamos de *deep learning*, mais especificamente, de rede neural convolucional, para fazer o reconhecimento e classificação de imagens de comida, visando reconhecer o tipo de comida descrito na imagem, a partir de um conjunto de tipos previamente definidos. Foi aplicada na arquitetura da rede a operação de *dropout* e a técnica de *data augmentation* na base de dados visando reduzir o *overfitting*.

No Capítulo 2 de revisão de literatura desse projeto são descritos conceitos que auxiliam no seu desenvolvimento, sendo dividida em dois tópicos principais: Redes Neurais e *Deep learning*. O Capítulo 3 de metodologia descreve como foi estruturado o projeto, e as etapas utilizadas na sua aplicação, composta por três tópicos principais: a definição e organização da base de dados; a arquitetura da rede neural proposta; e técnicas para melhorar o poder de classificação da rede neural.

O Capítulo 4 de resultados apresenta os valores obtidos com os experimentos realizados, contendo uma comparação entre os resultados e avaliando as melhorias obtidas conforme as técnicas eram adicionadas ao modelo. A melhora na acurácia e a redução do *overfitting* são pontos discutidos. No Capítulo 5 de conclusão é apresentada uma síntese sobre o modelo e os resultados obtidos, também é proposta atividades para a continuação deste trabalho.

2 REVISÃO DE LITERATURA

Neste capítulo será descrita a revisão de literatura sobre redes neurais, *deep learning* e redes neurais convolucionais.

Na Seção 2.1 são apresentados os conceitos básicos de redes neurais, mostrando sua composição e funções que influenciam no seu comportamento. Também é defina a estrutura da rede neural *Perceptron* e os tipos mais comuns de redes neurais.

A Seção 2.2 expõem o conceito de *deep learning*, a constituição de uma CNN, as operações de convolução, *pooling* e *soft max*, a função de ativação *ReLu*, a técnica de *dropout* e as camadas totalmente conectadas.

2.1 Rede neural

Uma rede neural artificial pode ser definida como sendo um conjunto interconectado de elementos básicos de processamento (GURNEY, 1997). Seu funcionamento é inspirado na capacidade de aprendizado do cérebro animal, que possui uma imensa estrutura com capacidade de definir regras a partir de experiências, que vão ocorrendo durante a vida, gerando ligações físicas (sinapses) mais fortes, aprimorando assim o que foi aprendido (HAYKIN, 2001).

> Uma rede neural é um processador maciçamente paralelo e distribuído, constituído de unidades de processamento simples, que tem a propensão natural para armazenar conhecimento experimental e torna-ló disponível para uso. Ela se assemelha ao cérebro em dois aspectos: o conhecimento é adquirido pela rede a partir de seu ambiente por meio de um processo de aprendizagem; forças de conexão entre neurônios, definidos como pesos sinápticos, são utilizados para armazenar o conhecimento adquirido (HAYKIN, 2001).

Dadas essas características, as redes neurais vêm sendo amplamente utilizadas para resolução de problemas que não possuem uma resolução trivial, como a classificação de imagens (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), a identificação de câncer de pele (ESTEVA et al., 2017), tomada de decisão no mercado de ações (GAMBOGI, 2013), e entre outras áreas. Essa versatilidade de áreas em que é utilizada ocorre devido a sua generalidade na maneira de encontrar pontos no problema que devem ter mais destaque, características mais relevantes que são identificadas pela rede durante seu processo de aprendizagem, sendo reavaliadas pela própria rede em cada instância testada. Segundo Zhang (2000), as redes neurais possuem a habilidade de se adaptar aos dados para realizar as classificações sem a necessidade de apontar explicitamente o que deve ser observado no modelo.

Segundo Kriesel (2007) uma rede neural contém três elementos:

- Unidades simples de processamento, ou neurônios.
- Elos de conexão entre os neurônios (sinapses).
- A importância entre a conexão de um neurônio com outro, descrita por uma função de peso *w*.

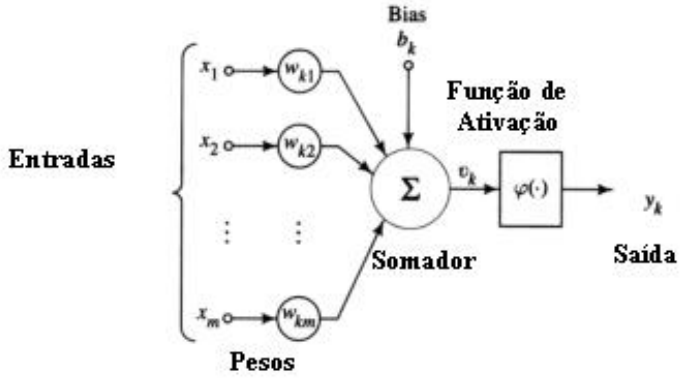
Uma rede neural pode ser descrita matematicamente pela tríplice (N,V,w), na qual N é um conjunto de neurônios, V é um conjunto de conexões entre os neurônios definida por $V = \{(i,j) \mid i,j \in N\}$ e w é a função que determina o peso das conexões definida por $w : V \implies \mathbb{R}$, descrita em $w(i,j)$.

2.1.1 Neurônio

HAYKIN (2001) define o neurônio como a unidade de processamento de informação que é primordial para o funcionamento de uma rede neural artificial. É nele que ocorre o processamento das entradas, e o redirecionamento da saída, indicando em que irá influenciar tal processamento.

Na Figura 1 é possível identificar os três elementos fundamentais de um neurônio:

Figura 1 – Modelo de um neurônio, contendo as entradas, funções de peso, função somadora, o *bias*, função de ativação da saída.



Fonte: GSIGMA: Grupo de Sistemas Inteligentes de Manufatura da UFSC [1]

- O fluxo de entrada dos dados, sendo ele um conjunto de conexões que serão sujeitadas a função de peso, para ser feito o uso na função somadora (HAYKIN, 2001). As conexões podem se originar tanto de uma entrada de dados na rede, quanto de neurônios que estão localizados em camadas superiores.
- A função somadora é responsável por realizar o processamento do fluxo de entrada. Um exemplo desse tipo de função é a soma dos pesos (KRIESEL, 2007), em que se realiza a multiplicação do peso w_{kj} com a entrada x_j, e depois realiza a soma das m entradas do neurônio representada pela função matemática:

$$u_k = sum_{j=1}^{m} w_{kj} x_j$$

O resultado dessa etapa é propagado pela rede dados os critérios da função de ativação.

[1] http://www.gsigma.ufsc.br/popov/aulas/rna/neuronio_artificial/neuronio_artificial.jpg, acessada em 02 de junho de 2017.

- A função de ativação é responsável por restringir a abrangência do dado gerado pelo processamento do neurônio. Foi identificado por HAYKIN (2001) três tipos básicos de função de ativação sendo elas:
  - **Função de limiar**:
  $$f(x) = \begin{cases} 1 & se\ x \geq 0 \\ -1 & se\ x < 0 \end{cases}$$
  É conhecida na literatura como função de *Heaviside*, definindo a saída de maneira binaria.
  - **Função linear por partes**:
  $$f(x) = \begin{cases} 1 & se\ x \geq 1 \\ x & se\ 0 \leq x < 1 \\ 0 & se\ x < 0 \end{cases}$$
  Esse tipo de função pode ser analisada como uma tentativa de simulação de um amplificador não linear, tendo sua área variável e seus pontos de saturação.
  - **Função Sigmoide**:
  $$f(x) = \frac{1}{1 + exp(-av)}$$
  Esse tipo de função de ativação é o mais utilizado na construção de redes neurais artificiais (HAYKIN, 2001). É definida por uma função crescente não linear, quando seu parâmetro de curva se aproxima do infinito, apresenta comportamento semelhante a funções de limiar.

O modelo neural da Figura 1 também inclui um *bias* (b_k) aplicado externamente. O *bias* tem como função aumentar ou diminuir a entrada mínima de dados e define um limiar no neurônio que é utilizado para o cálculo da função de ativação.

Dado as iteração, o *bias* e os pesos da entrada podem ser modificados pelo processo de aprendizagem, aprimorando sua resposta conforme a rede é treinada.

2.1.2 Processos de aprendizagem

A habilidade que se destaca de uma rede neural é a aprendizagem, adaptando-se aos dados que estão em seu ambiente para melhorar o seu desempenho. Essa habilidade vem de um processo realizado na rede neural artificial, que é definido por Demuth et al. (2014) como o procedimento de ajuste das funções de pesos e dos *bias* dos neurônios da rede, acontecendo na etapa de *"treino"* da rede e tendo como objetivo preparar a rede para executar uma tarefa.

Podemos dividir o processo de aprendizagem em três categorias principais sendo elas:

- **Aprendizado supervisionado:** método no qual uma parte da base é utilizada para treinar a rede. Assim após cada processamento é verificado o resultado da classificação e, se necessário, são feitas correções nos pesos dos neurônios que influenciaram esse resultado, para assim reforçar uma classificação boa ou corrigir uma classificação ruim.

- **Aprendizado por reforço:** método similar ao aprendizado supervisionado, tendo como diferença a forma de avaliação. Como dito por Kaelbling, Littman e Moore (1996), a principal diferença entre aprendizagem supervisionada e a aprendizagem por reforço é que o aprendizado por reforço não apresenta conjuntos de saídas corretos ou errados. Após cada ação, é aplicada uma taxa de correção e indicados os estados seguintes, mas não é informada qual escolha teria sido a melhor para o caso.
- **Aprendizado não-supervisionado:** método no qual não existe um avaliador ou dados pré-definidos informando a classe da entrada. A própria rede é responsável por associar os dados, ajustando os pesos e os *bias* a partir das entradas. A rede aprende como agrupar a entrada de dados em uma quantidade finita de classes.

2.1.3 *Perceptron*

O *perceptron* é tido como a forma mais simples de rede neural para classificar duas classes que são linearmente separáveis (HAYKIN, 2001). Como essa rede é composta por um único neurônio com pesos de conexões e *bias* ajustáveis, está limitado a classificar a entrada apenas em duas classes.

A rede é inicializada com pesos aleatórios e, após a execução de cada entrada, sua saída é comparada com o resultado esperado, obtendo assim um sinal de erro, que é utilizado para fazer ajustes nos pesos, como ocorre nos processos de aprendizado supervisionado.

Uma generalização do *perceptron* é o perceptron de múltiplas camadas (no inglês *multiple layer perceptron*, MLP), um MLP configurado em no mínimo três camadas. A primeira delas é a camada de entrada e a última é a camada de saída, que está contida a classificação da entrada. As camadas intermediárias têm a função de analisar características mais complexas da entrada, dando possibilidade de uma melhor classificação.

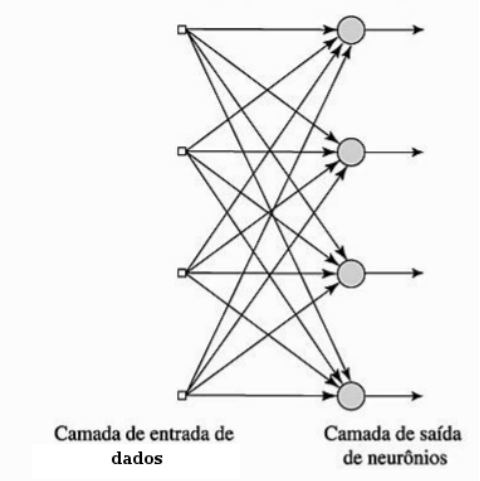
O método de aprendizagem utilizado pelo *perceptron* de múltiplas camadas é conhecido como *error backpropagation* (algoritimo de retropropagação de erro) (HAYKIN, 2001). Para isso, utiliza do método de aprendizado supervisionado de correção por erro. Quando é identificada a necessidade de ajuste nos pesos, ocorre uma retropropagação nos neurônios que influenciaram a classificação, ajustando seus pesos e *bias*.

2.1.4 Tipos de redes

Para problemas mais complexos, redes com apenas um neurônio tendem a não resolvê-los. Geralmente é necessário ter vários deles trabalhando em paralelo (uma camada de neurônios) (DEMUTH et al., 2014). HAYKIN (2001) descreve três classes de arquitetura de rede que geralmente são encontradas:

- **Redes alimentadas adiante de uma camada:** essa classe é a forma mais simples de rede em camada, na qual se tem uma camada de dados e uma camada de neurônios (camada de processamento), que também é a camada de saída, como na Figura 2. A camada de entrada de dados não é contada, pois nela não ocorre processamento.

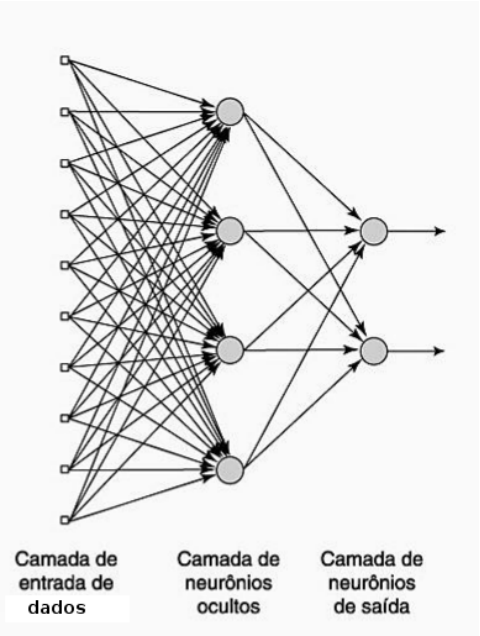
Figura 2 – Exemplo de rede alimentada adiante de uma camada.



Fonte: (HAYKIN, 2001)

- **Redes alimentadas adiante com múltiplas camadas:** essa classe de rede é também alimentada adiante, mas possui uma ou mais camadas ocultas. As camadas ocultas estão localizadas entre a camada de entrada de dados e a camada de saída. Ao adicionar camadas ocultas na rede é possível ter acesso a características mais específicas da entrada, melhorando o resultado da rede.
  Como representado na Figura 3, nesse modelo cada camada só fornece dados à camada posterior, e reciprocamente, só recebe dados da camada anterior. Exemplificando, a camada de entrada recebe os dados e formata a saída para a entrada da camada seguinte, a primeira camada oculta processa os dados fornecidos pela camada de entrada e o formata para a camada seguinte. Esse processo continua até chegar na camada de saída, conhecida também como camada final. A saída produzida por essa camada contém a resposta global produzida pela rede para a entrada fornecida na camada inicial.

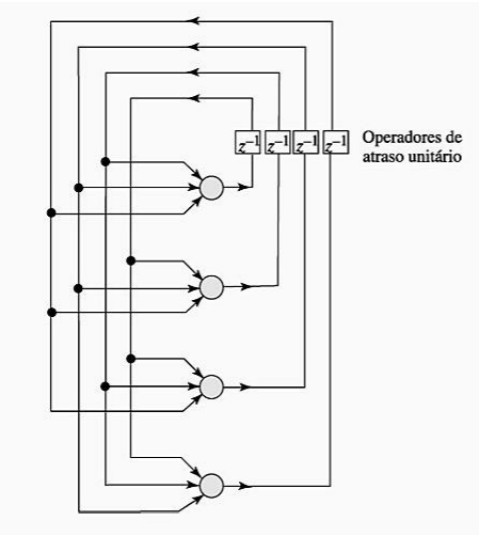
Figura 3 – Exemplo de rede alimentada adiante de múltiplas camadas.



Fonte: (HAYKIN, 2001)

- **Redes recorrentes:** essa classe de arquitetura se diferencia das anteriores pelo fato de possuir pelo menos uma camada com realimentação, ou seja, a saída da camada serve de entrada para a mesma, como exemplificado Figura 4. Os operadores de atraso unitário, representados na imagem pelo símbolo z^{-1}, são aplicados nas conexões de realimentação modificando de maneira dinâmica e não linear os valores informados. É dito que a rede possui uma realimentação quando a saída de um neurônio realimenta a sua entrada.

Figura 4 – Exemplo de rede recorrentes.



Fonte: (HAYKIN, 2001)

2.2 *Deep learning*

Deep learning pode ser definido como uma hierarquia de "*conceitos*" de aprendizagem, em que "*conceitos*" complexos se originam de grupos formados por "*conceitos*" mais simples. Se esses "*conceitos*" fossem representados em um grafo, seria possível ver como um "*conceito*" é montado baseado no outro, como se possuíssem muitas camadas (GOODFELLOW; BENGIO; COURVILLE, 2016).

> *Deep learning* permite que modelos computacionais compostos de múltiplas camadas de processamento aprendam representações de dados com múltiplos níveis de abstração. O *deep learning* descobre estruturas complexas em vastos conjuntos de dados com o uso do algoritmo de retropropagação para indicar como a máquina deve mudar seus parâmetros internos que são utilizados para computar a representação resultante da camada anterior em cada camada (LECUN; BENGIO; HINTON, 2015).

Bengio, Courville e Vincent (2013) categorizam o *deep learning* como um método de aprendizagem de representação (do inglês, *representation learning*). Métodos ditos como *representation learning* são capazes receber dados sem tratamento como entrada e a partir de processamentos internos encontrar automaticamente características relevantes para realizar a classificação. Sendo que, para o *deep learning*, cada camada oculta de processamento produz uma nova representação, podendo ser descrito como um método de aprendizado de multi representações. Dessa maneira a entrada pode ser uma imagem, descrita em um mapa de bits, na qual a primeira camada analisa informações mais superficiais, como contornos ou formas em certas áreas das imagens. Já na segunda camada seriam identificados padrões avaliando certas disposições de bordas ou formas em partes da imagem, ignorando pequenas variações. Na terceira camada seriam identificados os padrões que se assemelham a partes de objetos conhecidos, e nas camadas posteriores seria avaliada uma quantidade maior de padrões até chegar ao ponto de realizar a classificação dos objetos contidos na imagem (LECUN; BENGIO; HINTON, 2015).

Por essa versatilidade, o *deep learning* vem sendo usado na resolução de diversos problemas em que as características não estão perceptíveis. Classificação e identificação de objetos (FARABET et al., 2013) e reconhecimento de fala (HINTON et al., 2012) são problemas que vem sendo resolvidos com *deep learning*.

Na área de classificação de imagens uma técnica de *deep learning* muito utilizada é a rede neural convolucional (do inglês, *convolutional neural network*, CNN). Como mostrado por Krizhevsky, Sutskever e Hinton (2012), a CNN vem atualizando o estado da arte na área.

2.2.1 Redes neurais convolucionais

Como descrito por LeCun et al. (1989) redes neurais convolucionais são um tipo especializado de rede neural para processamento de dados que se organizam em grade (ou matriz), tendo como um exemplo de entrada uma imagem, uma matriz de *bits*.

Elas possuem o nome de rede convolucional pois, em algumas de suas camadas ocultas é aplicada a operação de convolução. Outra operação muito utilizada nessas redes é a operação de *pooling* (GOODFELLOW; BENGIO; COURVILLE, 2016).

Uma camada de uma rede neural convolucional geralmente é composta de três fases: a primeira fase na qual são aplicadas diversas convoluções em paralelo na mesma imagem gerando um conjunto de ativações lineares; a segunda fase aplica uma função de ativação não linear, sendo a unidade linear de correção (do inglês *rectified linear unit*, ReLU) muito utilizada atualmente (LECUN; BENGIO; HINTON, 2015); no terceiro e último estágio é utilizada uma função de *pooling* para modificar o dado que será fornecido para a próxima camada.

2.2.1.1 Operação de convolução

Camadas de convolução são baseadas essencialmente na operação de convolução. Segundo Goodfellow, Bengio e Courville (2016), a operação de convolução é descrita por uma operação que ocorre entre duas funções, podendo ser especificada da seguinte maneira:

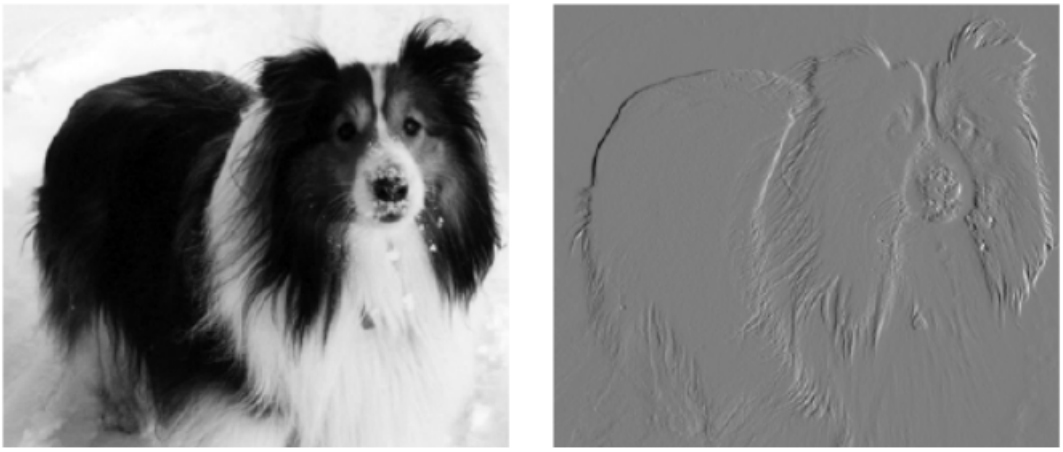
$$s(t) = (x * w)(t)$$

Em redes convolucionais os argumentos da função de convolução são geralmente compostos pela entrada de dado (x) e o *kernel* (w) utilizado para a modificação. A variável t indica o intervalo em que será aplicada a operação de convolução na entrada x. Sua saída é um mapa de características (*feature maps*) que é representada pela tríplice Altura, Largura e Quantidade de mapas. Exemplificando, uma imagem no modelo RGB que possue as 100×100 como suas dimensões (altura e largura) e possui os três filtros do RGB, apresenta a descrição de entrada seguinte $100 \times 100 \times 3$. Quando é aplicado nessa entrada 10 filtros com a operação de convolução, tendo em vista que esses filtros não modificam a dimensão da imagem, é gerado uma imagem com a descrição de $100 \times 100 \times 10$.

Assim, a entrada de dados normalmente é uma matriz, nesse caso uma imagem. O *kernel* utilizado também costuma ser uma matriz de parâmetros que podem ser ajustados pelo processo de aprendizagem (retropropagação). O resultado da aplicação da operação de convolução é uma matriz com os dados da entrada modificados. Dessa maneira, conforme o *kernel* informado, padrões são destacados na matriz resultante da operação de convolução.

A operação de convolução é uma maneira eficiente de descrever transformações para serem aplicadas em áreas menores mantendo a linearidade, em todo o dado de entrada. Como levantado por Goodfellow, Bengio e Courville (2016), para realizar uma operação de subtração entre os *pixels* de uma imagem para encontrar as bordas, como visto na Figura 5, é necessária uma quantidade muito menor de computação para obter o resultado desejado quando é utilizada a convolução.

Figura 5 – Exemplo da aplicação de operações para encontrar as bordas verticais de uma imagem. A esquerda a imagem normal em escala cinza e a direita a imagem aplicada a operação subtração dos *pixeis* vizinhos.



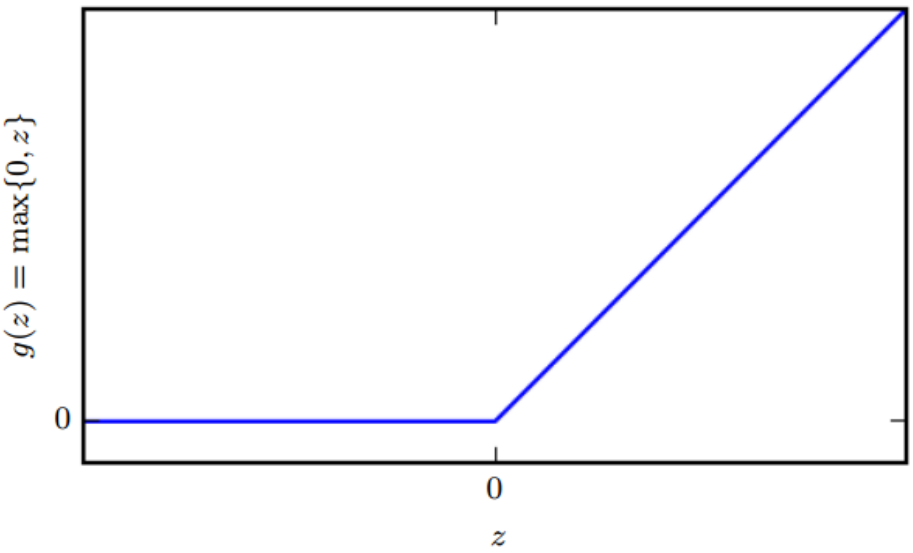
Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

2.2.1.2 ReLu

Neurônios com função de ativação ReLu (do inglês, *rectified linear unit*), vêm sendo utilizados em redes neurais alimentadas adiante dado seu baixo custo de processamento (GLOROT; BORDES; BENGIO, 2011). Como visto na Figura 6, a ReLu se mantém muito próxima de uma função linear, sendo a única diferença que metade do seu domínio é 0. Isso ocorre nos neurônio que não estão ativos, não participando assim dos processamentos dos dados e das correções que ocorrem na fase de *backpropagation*.

Sua função é descrita por pela equação $g(z) = max\{0,z\}$, na qual o z no contexto de CNN é o valor dos dados da matriz. Portanto, a aplicação dessa função em uma matriz serve para eliminar valores negativos, evidenciando os valores que se destacaram durante o processo de convolução. Possibilitando assim, padrões relevantes serem identificados.

Figura 6 – Gráfico da função de ativação não linear ReLu.



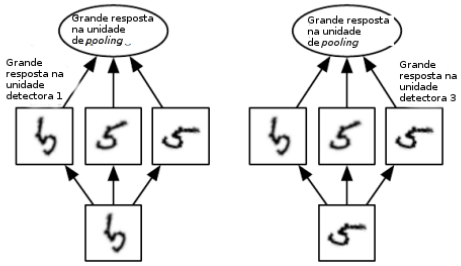
Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

2.2.1.3 Operação de *pooling*

A aplicação da função de *pooling* para modificar o dado de entrada, ajuda a tornar o modelo classificador adaptado a pequenas translações da entrada (GOODFELLOW; BENGIO; COURVILLE, 2016). Dessa forma a aplicação dessa operação permite identificar se o objeto está contido na imagem independente do local que aparece e das dimensões que apresenta. Essa característica se torna muito eficaz para ser aplicada em redes que necessitam dessa variabilidade de padrões de posição para uma mesma classe.

Exemplificando, em uma carta a aplicação da operação de *pooling* permite a rede identificar os números do código postal que estão escritos a mão, mesmo que estes não estejam localizados no mesmo local, ou inclinação de cada respectiva imagem, como ilustrado na Figura 7.

Figura 7 – Exemplo de como é feita a ativação de um neurônio na camada de *pooling*.



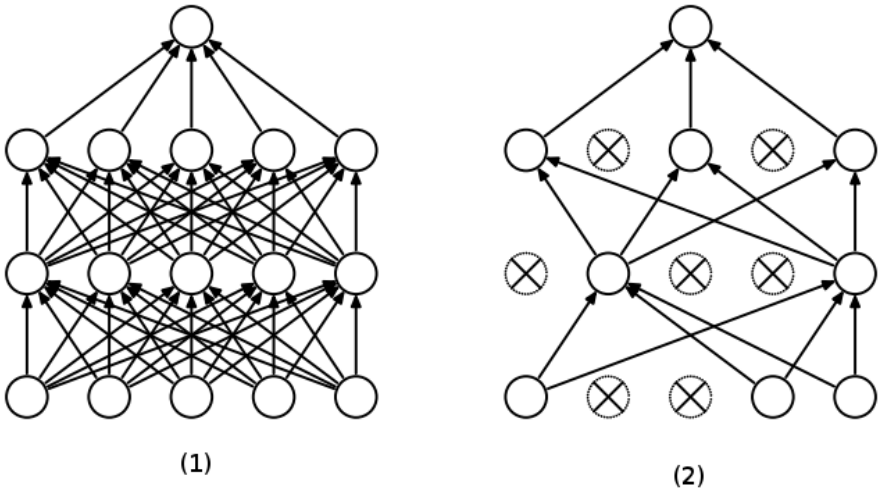
Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

2.2.1.4 *Dropout*

Um dos grandes problemas no uso de CNN é o *overfitting*, que ocorre quando a base disponível para o treino é pequena ou quando a rede neural possui muitas camadas. Uma estratégia utilizada por Krizhevsky, Sutskever e Hinton (2012) é a aplicação da técnica de *dropout* em algumas camadas da rede.

Srivastava et al. (2014) define o termo *dropout* como a remoção temporária de alguns neurônios da rede, junto com suas conexões de entradas e saídas como é possível visualizar na Figura 8. Essa operação, que remove virtualmente o neurônio da rede neural, ocorre somente na fase de treino, em que quando é ativada no neurônio o exclui do processo de aprendizagem da rede.

Figura 8 – Exemplo da aplicação do *dropout* em uma rede neural. Na imagem (1) apresenta uma rede normal sem remoção de neurônio. Na imagem (2) mostra a rede com o *dropout* ativado em alguns neurônios, removendo-os temporariamente.



Fonte: (SRIVASTAVA et al., 2014)

Essa operação é utilizada com o intuito de reduzir o *overfitting*. A redução do tempo de execução e o aprendizado de atributos mais relevantes pela rede são outras características que se destacam quando aplicada a técnica de *dropout*.

2.2.1.5 *Softmax*

Softmax é uma função estatística que realiza a normalização de um vetor de números para valores percentuais, na qual a soma de todos os números do vetor após a aplicação da função é de 100%. Na CNN, a função *softmax* é utilizada como função de ativação na ultima camada, onde é obtido a classificação da entrada (BISHOP, 2007).

A função *softmax* é descrita pela expressão $s(x,i) = \frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_k}}$, na qual N é o tamanho do vetor de números, x é vetor de números, i é a posição do vetor que está sendo obtido a porcentagem. A expressão deve ser aplicada em todas as posições do vetor para se obter a classificação.

3 METODOLOGIA

Neste capítulo são descritas as etapas do desenvolvimento desta monografia, sendo elas a formulação e pré-processamento da base de dados, a configuração da rede neural e sua codificação utilizando o *framework keras* (CHOLLET et al., 2015), além de pesquisas e aplicações de melhorias para a rede neural proposta, como o *data augmentation* e a inicialização dos pesos a partir de valores de uma rede já treinada.

3.1 Formulação da base

No aprendizado de máquina, é recomendável que a base de dados em que foram feitos os treinos e os testes do método escolhido possua balanceamento na distribuição de amostras por classes, e as amostras de uma mesma classe precisam ter características que as diferenciem das outras amostras de outras classes.

Neste trabalho foi feita a classificação de imagens de comida, dessa maneira, a base montada contém amostras segregadas em classes como *pizza* e *sushi*. Também foram adicionadas classes de imagens que não estão relacionadas com comida, como *plant*(no português, planta) e *domestic animals*(no português, animais domésticos), visando melhorar o classificador, quando utilizado em bases que possuam imagens não associadas à comida.

As imagens selecionadas para a formulação do conjunto de dados deste trabalho foram retiradas das bases de dados *ImageNet*(DENG et al., 2009) e *Food-101*(BOSSARD; GUILLAUMIN; GOOL, 2014). A base *ImageNet* é densamente estruturada e organizada em hierarquia de árvore, facilitando assim encontrar as categorias que estão relacionadas ao tema abordado. Já a base *Food-101* é composta de 101 classes de imagens de comida, selecionadas e com um tamanho fixo de 1000 imagens por classe. A precisão de categorização das bases *ImageNet* e *Food-101* são necessárias para a formulação da base de teste deste projeto, tendo em vista que a categorização manual de imagens não seria uma abordagem viável, uma vez que redes neurais convolucionais demandam uma grande quantidade de imagens para um treinamento adequado.

A base formulada possui um total de 16000 imagens separadas em 16 classes, sendo dessas classes 13 relacionadas com comida (*chocolate cake, french fries, hamburger, ice cream, pizza, spaghetti bolognese, sushi,club sandwich, filet mignon, fried rice, hot dog, steak, tacos*) e três não relacionadas com comida (*domestic animal, people, plant*). Como descrito na Tabela 1, é possível identificar que as imagens das 13 classes relacionadas com comida foram retiradas da base de dados *Food-101*, elas foram selecionadas com base na popularidade dos tipos de comida que representam, como *pizza* e macarrão (*spaghetti bolognese*). As imagens das três classes restantes foram retiradas da base de dados *ImageNet*, foram escolhidas visando abranger imagens que possam ocorrer em bases de dados que não sejam só de comidas, buscando

melhorar o desempenho das classificações nessas bases. Na Figura 9 podemos ver um exemplo da diversidade dessas imagens.

Figura 9 – Exemplos de imagens encontradas na base de dados.

Fonte: imagens retiradas das bases *ImageNet*(DENG et al., 2009) e *Food-101*(BOSSARD; GUILLAUMIN; GOOL, 2014)

3.1.1 Pré-processamento

Redes neurais convolucionais requerem uma grande quantidade de imagens. Dessa maneira, as entradas fornecidas para a rede neural devem seguir um padrão, sendo que todas as imagens precisam possuir as mesmas dimensões. Assim as imagens foram redimensionadas para 227×227 *pixels*, tendo em vista que é um valor que proporcionou bons resultados em trabalhos semelhantes (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Nas alterações de dimensões das imagens foi realizado um corte nas imagens originais para forçar uma formato quadrado antes de ser aplicado o redimensionamento, preservando os formatos das imagens. As perdas de informações que podem ocorrer com os cortes realizados nas imagens são pequenas, sabendo que os objetos relevantes para a classificação estão apresentados no centro e os cortes são realizados nas extremidades das imagens. Com o uso da função de *max pooling* possíveis perdas de informações relevantes que podem ocorrer nos cortes das imagens são reparadas.

Para a realização dos treinos e testes com a rede neural, a base de dados foi separada em dados de treino e de teste. Como informado na Tabela 1, 70% das imagens (11200 imagens) de cada classe foram utilizadas para o treino da rede neural, e os 30% das imagens restantes (4800 imagens) foram utilizadas para a fase de teste da rede neural.

Tabela 1 – Separação da base de dados em classes, sendo informada a quantidade de amostras separadas para realizar as etapas de treino e teste, também é informado a base de origem do dado.

Classe	Amostras de treino	Amostras de teste	Amostras totais	Base de origem
chocolate cake	700	300	1000	*Food-101*
french fries	700	300	1000	*Food-101*
hamburger	700	300	1000	*Food-101*
ice cream	700	300	1000	*Food-101*
pizza	700	300	1000	*Food-101*
spaghetti bolognese	700	300	1000	*Food-101*
sushi	700	300	1000	*Food-101*
club sandwich	700	300	1000	*Food-101*
filet mignon	700	300	1000	*Food-101*
fried rice	700	300	1000	*Food-101*
hot dog	700	300	1000	*Food-101*
steak	700	300	1000	*Food-101*
tacos	700	300	1000	*Food-101*
domestic animal	700	300	1000	*ImageNet*
people	700	300	1000	*ImageNet*
plant	700	300	1000	*ImageNet*

3.2 Configuração da Rede Neural

A configuração da rede neural convolucional utilizada neste projeto é fundamentada na rede neural *AlexNet* definida por Krizhevsky, Sutskever e Hinton (2012). Essa rede utiliza de oito camadas ocultas de processamento, sendo cinco camadas convolucionais e três camadas fortemente conectadas. Tendo em vista que a rede escolhida como exemplo foi estruturada para classificar mil classes, foi necessário fazer algumas adaptações para classificar uma quantidade menor de classes. Uma dessas alterações foi a modificação da função *softmax* para obter o resultado da classificação na última camada conforme descrito na Figura 10. A função de ativação ReLu foi usada em todas as camadas da rede, exceto na camada de saída, em que foi aplicada a função *softmax*.

A primeira camada de convolução da rede separa a entrada, uma imagem de $227 \times 227 \times 3$ em 96 núcleos de $11 \times 11 \times 3$ (com uma distância de 4 *pixels* entre os centros das imagens vizinhas), em que cada imagem gerada é processada separadamente. Gerando uma imagem $55 \times 55 \times 96$, na qual 55×55 representa as dimensões da imagem e 96 a quantidade de filtros que foram gerados nela. Após essa camada, é aplicada uma camada de *max pooling* em regiões de 3×3 *pixels* da imagem, com uma distância de 2×2 *pixels* entre os centros da regiões, resultando em uma imagem de $27 \times 27 \times 96$.

A segunda camada convolucional tem como entrada a saída da primeira camada convolucional normalizada e com *pooling* aplicado. Essa entrada tem o formato de $27 \times 27 \times 96$ é filtrada em 256 núcleos de $5 \times 5 \times 96$, gerando uma saída de $27 \times 27 \times 256$ com as mesmas

dimensões aumentado a quantidade de filtros aplicados. Após essa camada também é aplicada uma camada de *max pooling* com as mesmas configurações da camada de *max pooling* anterior, resultando em uma saída de $13 \times 13 \times 256$.

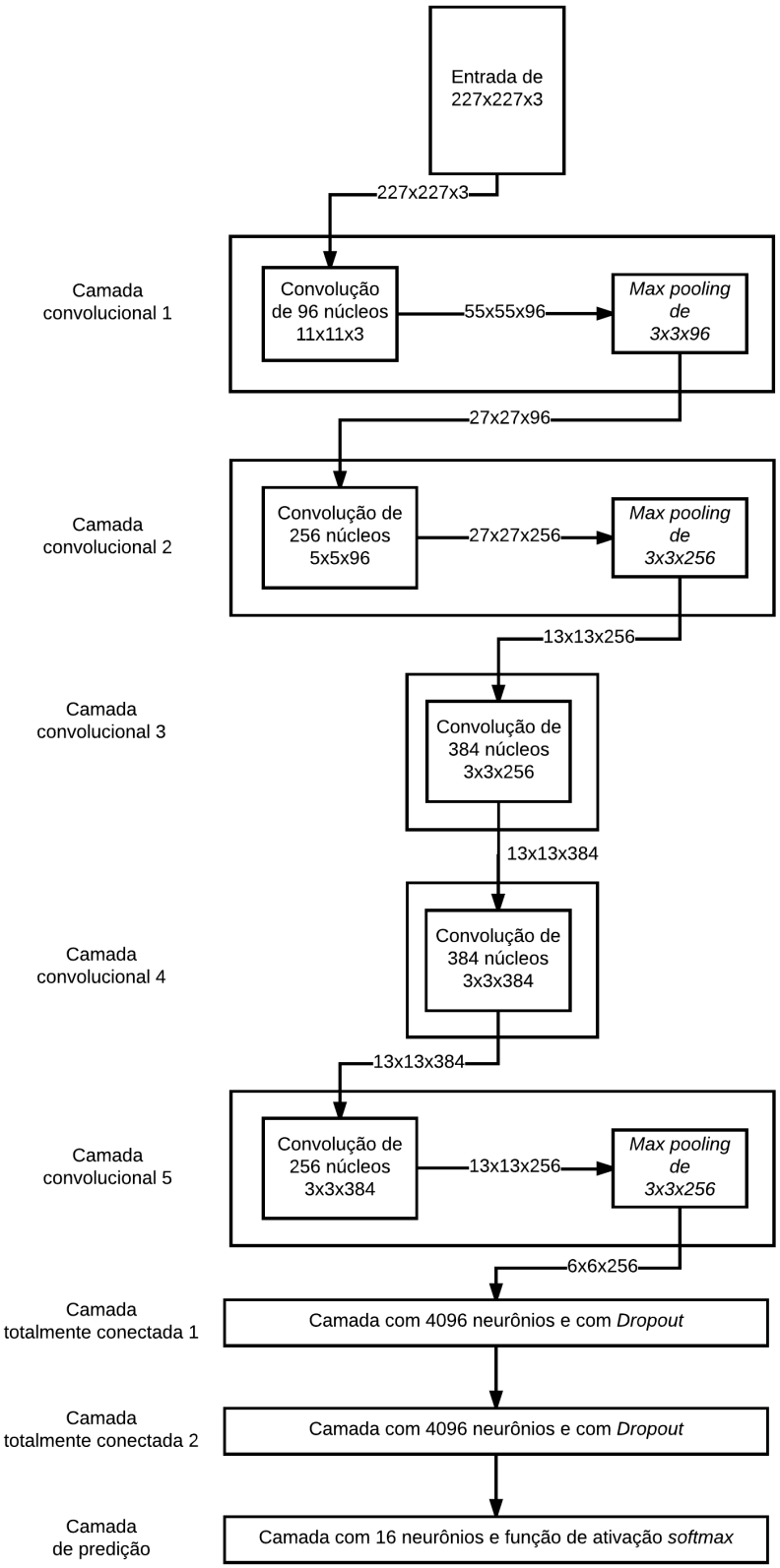
A terceira camada convolucional possui 384 núcleos de $3 \times 3 \times 256$ que são conectados à entrada fornecida pela saída da segunda camada convolucional normalizada e aplicado o *pooling*. O formato da saída gerada pela terceira camada é de $13 \times 13 \times 384$. As conexões entre a terceira, quarta e quinta camada convolucional não possuem nenhuma operação de *pooling* entre si.

Assim, a quarta camada convolucional possui 384 núcleos de $3 \times 3 \times 384$ e gera uma saída de formato $13 \times 13 \times 384$, na qual só é aplicada a convolução e não é modificada a entrada.

A quinta camada possui 256 núcleos de $3 \times 3 \times 384$ e gera uma saída com o formato de $13 \times 13 \times 256$, reduzindo a quantidades de filtros aplicados na imagem. Após a quinta camada convolucional é aplicada uma camada de *max pooling* com as mesmas configurações das outras camadas de *max pooling* utilizadas na rede, gerando uma saída de $6 \times 6 \times 256$.

As duas camadas seguintes são camadas fortemente conectadas com 4096 neurônios cada, com no fim de cada uma aplicada a função de *dropout* inicialmente com uma taxa de 50%. A função de ativação das camadas fortemente conectadas foi o ReLu. Por fim uma camada com 16 neurônios (um neurônio para cada classe) fortemente conectada com a operação de *softmax* para obter a predição da entrada. O processo de aprendizagem utilizados nas camadas fortemente conectadas foi o *backpropagation*. A taxa de aprendizagem utilizada durante o *backpropagation* foi de $0{,}01\%$, tendo em vista que foram obtidos bons resultados utilizando esse valor em outras redes neurais convolucionais (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Figura 10 – Diagrama que representa a arquitetura da rede neural convolucional proposta. O formato de cada camada da rede é descrito na seta que incide nele, e o formato que ele gera é infomado na seta que sai dele. Como exemplo a operação de convolução da camada convolucional 1, na qual a sua entrada é de $227 \times 227 \times 3$ e sua saída é de $55 \times 55 \times 96$.



Foi utilizado o *framework Keras* (CHOLLET et al., 2015) para implementar a rede neural. *Keras* é um *framework* em *python* para execução de redes neurais utilizando *GPU*(*Graphics Processing Unit*). Com o *keras* é possível configurar em alto nível uma rede neural, abstraindo a complexidade da descrição e implementação das rotinas de execução das camadas. Como exemplo de implementação, temos a Figura 11, contendo a codificação da primeira e segunda camada da rede neural proposta. Neste projeto o *keras* está sendo utilizado com o *back end Theano* (AL-RFOU et al., 2016) para a geração de código em *Cuda*, linguagem que compila codigo para ser executado em GPU.

Figura 11 – Trecho de código com implementação utilizando o *framework keras* das duas primeiras camadas convolucionais da rede neural, contendo a definição da entrada e as implementações das camadas de transição entre as camadas de convolução um e dois.

```
def AlexNet():
 # Descrição do formato da entrada
    entradas = Input(shape=(3,227,227))
 # Descrição da primeira camada convolucional contendo a quantidade de amostras que irá
 #  gerar, e a função de ativação que irá utilizar
    camada_conv_1 = Convolution2D(96, 11, 11,subsample=(4,4),activation='relu',
                                  name='camada_conv_1')(entradas)
 # Descrição da segunda camada convolucional, indica as camadas de pooling e normalização que
 #  ocorre antes da sua execução
    camanda_conv_2 = MaxPooling2D((3, 3), strides=(2,2))(camada_conv_1)
    camanda_conv_2 = crosschannelnormalization(name="camada_pool_1")(camanda_conv_2)
    camanda_conv_2 = ZeroPadding2D((2,2))(camanda_conv_2)
    camanda_conv_2 = merge([
        Convolution2D(128,5,5,activation="relu",name='camanda_conv_2_'+str(i+1))(
            segregracao_entradas(ratio_split=2,id_split=i)(camanda_conv_2)
        ) for i in range(2)], mode='concat',concat_axis=1,name="camanda_conv_2")

```

3.3 Melhorias para a rede neural proposta

Melhorias no desempenho de redes neurais podem ser aplicadas em diversas etapas do processo de aprendizado e classificação. O aumento da base de treino, a definição de quais camadas devem ser treinadas em determinadas épocas, a inicialização dos pesos da rede neural com valores de uma rede treinada com uma quantidade maior de amostras e o aprimoramento dos parâmetros de configuração da rede, são métodos que podem ser utilizados para aprimorar seu desempenho de classificação.

Neste projeto foram aplicadas técnicas para a melhorar a classificação, sendo elas o aumento dos dados na base de treino e a inicialização dos pesos da rede neural com pesos de uma rede já treinada.

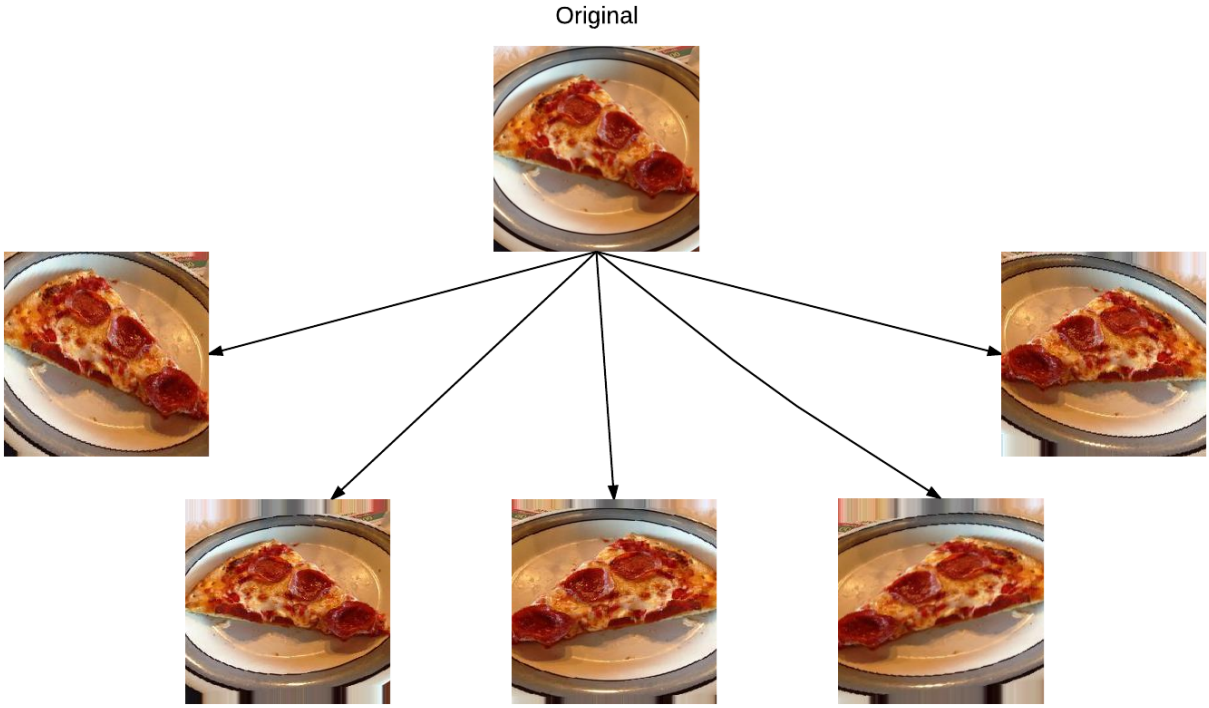
3.3.1 *Data augmentation*

Uma técnica que vem sendo muito utilizada no para a redução do *overfitting* na fase de treino de uma rede neural é a *data augmentation* (CUI; GOEL; KINGSBURY, 2015). Os dados da base são ligeiramente modificados para se obter o aumento na quantidade de amostras. Essas mudanças podem ser inversões nos eixos, pequenas rotações, aproximações em certas partes das imagens ou até a aplicação da imagem em escala cinza. Essa técnica tem o

propósito de aumentar a quantidade de amostras em que serão realizados os treinos, buscando evitar um *overfitting* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Nesse projeto foi utilizada a técnica de *data augmentation* aplicando inversão no eixo y, realizando um zoom de aproximação ou distanciamento de até 20% e aplicado uma taxa de inclinação de até 0,2 radianos. As amostras foram geradas com a combinação das transformações possíveis informadas, criando nove imagens a partir de cada imagem de treino, como representado na Figura 12. Essas imagens foram geradas durante a execução do treino da rede e foram armazenadas em memória.

Figura 12 – Imagem representado a técnica de *data augmentation* aplicada na base de treino. A imagem no centro é a original, e as outras são possíveis modificações aplicadas sobre ela, como a inversão do eixo y e o aumento e diminuição do *zoom*.



3.3.2 Inicialização dos pesos

Uma rede neural treinada do início ao fim tem seus pesos inicializados de maneira aleatória e, conforme vai realizando suas predições, os pesos são corrigidos para melhorar o poder de classificação. A inicialização dos pesos da rede neural com valores obtidos a partir de uma rede treinada vem sendo utilizada como maneira de melhorar a classificação (GIRSHICK et al., 2014). Geralmente os pesos vêm de redes que treinaram uma quantidade muito grande de dados, conseguindo de certa forma transferir o aprendizado obtido para a rede que está inicializando os pesos. Nesse projeto foi realizada a inicialização dos pesos da rede neural

utilizando os pesos obtidos no treinamento da rede desenvolvida por Krizhevsky, Sutskever e Hinton (2012) para melhorar o desempenho da rede neural proposta.

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

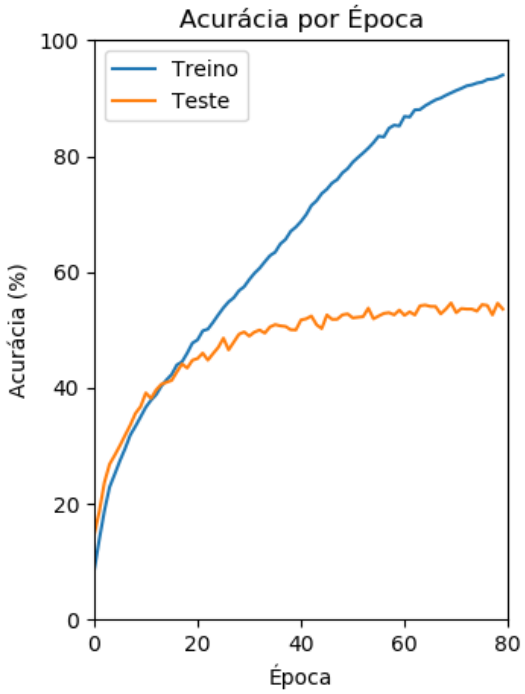
Nesta seção estão descritos os resultados obtidos a partir dos experimentos realizados na rede neural convolucional proposta, apresentando também os resultados obtidos com as variações das aplicações das técnicas propostas. Os experimentos foram iniciados a partir da rede neural convolucional sem nenhuma alteração, seguindo para a inclusão das alterações na base e melhorias na rede, com o propósito de avaliar os resultados obtidos e determinar o modelo com a melhor acurácia.

Nos experimentos iniciais realizados foi identificado que a partir de 80 épocas de execução não ocorria nenhuma modificação nas fases de treino e teste, sempre mantendo uma faixa de variação constante. Dessa maneira foi definido como padrão a quantidade de 80 épocas para a execução dos experimentos. A escolha dos parâmetros da rede como a taxa de *dropout*, foram definidos com base na rede *AlexNet* descrita por Krizhevsky, Sutskever e Hinton (2012). Como abordagem para a redução do *overfitting*, a taxa de *dropout* foi otimizada para o modelo proposto com base em testes empíricos.

4.1 Modelo inicial

O experimento realizado com o modelo inicial proposto utilizando a base de dados sem modificação obteve um resultado de 94,02% de acurácia na base de treino e 53,6% na base de teste, como apresentado na Tabela 2. Com esses valores é possível dizer que ocorreu um *overfitting* no modelo sobre as amostras da base de treino. O gráfico apresentado na Figura 13 mostra que a acurácia da fase de teste mantém valores constantes, a partir de 42 épocas, não apresentando um ganho expressivo, diferentemente da acurácia obtida na fase de treino, que mantém uma curva crescente.

Figura 13 – Gráfico contendo a acurácia obtida na fase de treino e teste de cada época do modelo de rede neural inicialmente proposta sem a aplicação de técnicas de melhorias.



4.2 Modelos com *data augmentation* e inicialização dos pesos

Também foram realizados experimentos com as estratégias de melhorias da rede neural, com o objetivo de reduzir o *overfitting* na rede. Foram realizados três experimentos: um aplicando a técnica de *data augmentation*; um com a técnica de inicialização dos pesos a partir dos valores treinados na rede *AlexNet* desenvolvida por Krizhevsky, Sutskever e Hinton (2012); e um com ambas as técnicas.

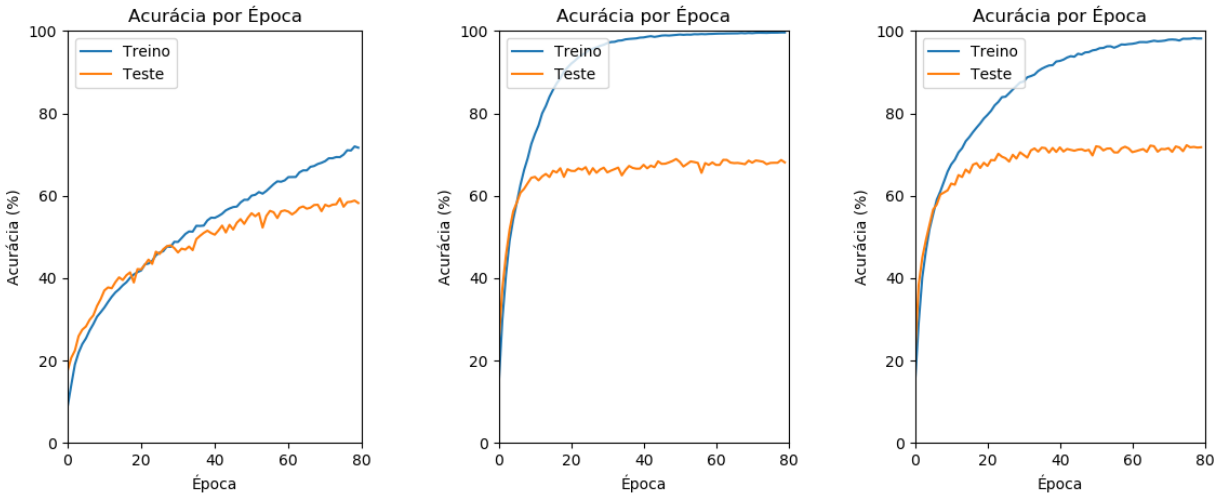
Como informado na Tabela 2, as acurácias obtidas na fase de teste e treino, respectivamente, apenas com *data augmentation* foram de 58,23% e 71,66%, mostrando como essa mudança causou uma redução significativa do *overfitting* na rede, além de uma melhora da acurácia na fase de teste.

Tabela 2 – Resultados dos experimentos no modelo inicialmente proposto, no modelo com a aplicação de *data augmentation*, no modelo com inicialização dos pesos e no modelo com ambas as técnicas aplicadas.

Modelo	Fase de treino	Fase de Teste
Modelo inicial	94,02%	53,6%
Modelo com *data augmentation*	71,66%	58,23%
Modelo com inicialização dos pesos	99,63%	68,08%
Modelo com *data augmentation* e inicialização dos pesos	98,18%	71,77%

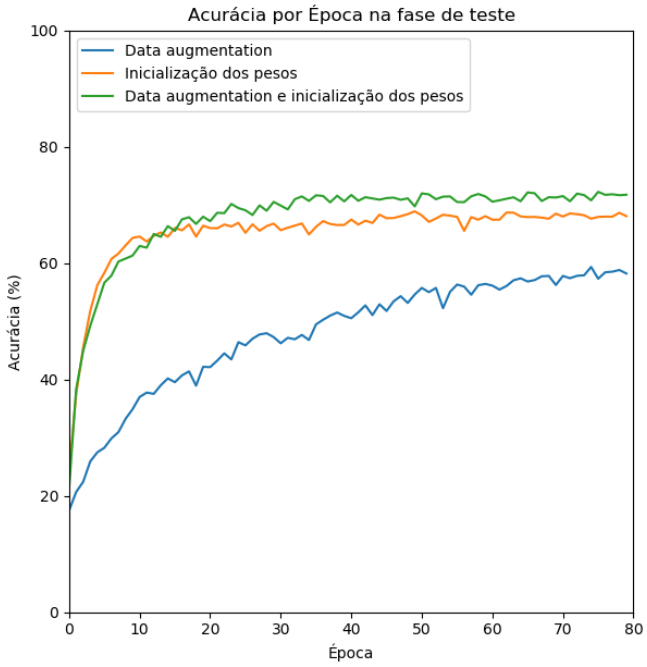
O experimento realizado com a inicialização dos pesos obteve o resultado mais expressivo se comparado com o teste apenas com *data augmentation*, obtendo as acurácias na fase de teste e treino, respectivamente, de 68,08% e 99,63%. Com a aplicação dessa técnica foi obtida uma melhora significativa na acurácia se comparado com o modelo inicial e com o de *data augmentation*, mas, assim como no modelo inicial, esse modelo apresentou *overfitting* na fase de treino. Como é possível observar no gráfico (2) da Figura 14, a partir da época 20 a acurácia da fase de teste permaneceu estável e a da fase de treino continuou crescendo até atingir valores próximos a 100%.

Figura 14 – Gráficos contendo as acurácias obtidas nas fases de treino e teste dos modelos com as melhorias aplicadas. No gráfico (1) apresenta os resultados do experimento com a utilização de *data augmentation*. No gráfico (2) apresentado os resultados do experimento com a inicialização dos pesos a partir de uma rede treinada. O gráfico (3) apresenta os resultados obtidos com o modelo com a técnica de *data augmentation* e inicialização dos pesos.



Com o intuito de solucionar o problema de *overfitting* e obter melhoria na classificação, foi realizado o teste utilizando as duas técnicas. A acurácia obtida na fase de treino foi de 98,18% e na fase de teste obteve um valor de 71,77%, apresentando o melhor resultado entre os três testes realizados, como pode ser observado no gráfico comparativo na Figura 15.

Figura 15 – Gráficos contendo as acurácias obtidas nas fases de teste dos modelos com as melhorias aplicadas.



4.3 Aprimoramento da taxa de *dropout*

Foram realizados também experimentos com a taxa de *dropout* buscando diminuir o *overfitting* na rede. A taxa de *dropout* foi variada de 50% a 90%, com intervalos de 10% entre cada experimento. O modelo utilizado para a execução dos experimentos foi o com a aplicação de *data augmentation* e inicialização dos pesos. Como pode ser verificado na Tabela 3, a taxa de 80% foi a que apresentou melhor resultado, com uma acurácia de 74,56% na fase de teste e 93,96% de acurácia na fase de treino.

Tabela 3 – Resultados da execução com a variação na taxa de *dropout*.

Taxa de *dropout* (%)	Fase de treino	Fase de Teste
Modelo com taxa de 60%	97,78%	72%
Modelo com taxa de 70%	96,61%	72,77%
Modelo com taxa de 80%	93,96%	74,56%
Modelo com taxa de 90%	95,37%	72,24%

A partir da Tabela 3 é possível concluir que o experimento realizado com a taxa de *dropout* 80% apresentou a maior acurácia na fase de teste e apresentou a menor diferença entre os resultados da fase de treino e teste, conseguindo reduzir o *overfitting* que ocorre na rede.

5 CONCLUSÃO

A tarefa de classificação de imagem vem sendo um dos grandes desafios na área de visão computacional. Utilizar *deep learning*, com CNN, para a classificação de imagens é algo que ocorre desde 2012 e vem atualizando o estado da arte nessa área. Neste trabalho utilizamos de CNN para realizar a classificação de imagens de comida, utilizando técnicas de *data augmentation* e inicialização dos pesos da rede para obter uma melhor classificação e reduzir o *overfitting*, uns dos grandes problemas no uso de CNN.

O *overfitting* é um dos grandes obstáculos para o uso de CNNs. Uma base de treino muito pequena ou uma rede neural muito profunda são possíveis causas do *overfitting*. A aplicação de técnicas para aumento da base de dados e a utilização do *dropout* em algumas camadas da rede são medidas que reduzem o *overfitting*, conforme foi aplicado e constatado neste trabalho.

Data augmentation vem sendo aplicado nos modelos de classificação que utilizam CNN. A aplicação do *data augmentation* na base foi feita com o objetivo de modificar ligeiramente as imagens originais. Também foi utilizado para melhorar o resultado de classificação a inicialização dos pesos da rede com os valores obtidos a partir de um rede já treinada, ao invés de realizar a inicialização com dados aleatórios.

Nos experimentos realizados, o emprego da técnica de *data augmentation*, como relatado por outros trabalhos, obteve um resultado expressivo para realizar a redução do *overfitting*, além que alcançar uma melhora na acurácia. A aplicação em conjunto do *data augmentation*, inicialização dos pesos e otimização da taxa de *dropout* obteve um resultado de 74,56% de acurácia, aumentando em 20,96% a acurácia se comparado com o modelo proposto sem as melhorias (53,6% de acurácia).

Outra abordagem que pode ser realizada futuramente é utilizar o modelo para aplicar a extração das características em conjunto com outro classificador como SVM (do inglês *suport vector machine*) para realizar a predição. A aplicação de vetores de dissimilaridade nas características extraídas com a CNN para a classificação seria outro caminho a ser abordado. Otimizações de outros parâmetros da rede como a taxa de aprendizagem e o estudo de outras arquiteturas de CNN como a *GoogLeNet* (SZEGEDY et al., 2015) para melhorar o desempenho do classificador. A aplicação de técnicas para redução dos parâmetros, possibilitaria o uso desse classificador em dispositivos com uma menor capacidade de processamento.

Referências

AL-RFOU, R. et al. Theano: A Python framework for fast computation of mathematical expressions. **arXiv e-prints**, abs/1605.02688, maio 2016. Acessado em 27 de outubro de 2017. Disponível em: <http://arxiv.org/abs/1605.02688>. Citado na página 19.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 35, n. 8, p. 1798–1828, 2013. Citado na página 8.

BISHOP, C. Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. corr. 2nd printing edn. **Springer, New York**, 2007. Citado na página 12.

BOSSARD, L.; GUILLAUMIN, M.; GOOL, L. V. Food-101 – mining discriminative components with random forests. In: **European Conference on Computer Vision**. [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 14 e 15.

CHOLLET, F. et al. **Keras**. [S.l.]: GitHub, 2015. <https://github.com/fchollet/keras>. Citado 2 vezes nas páginas 14 e 19.

CUI, X.; GOEL, V.; KINGSBURY, B. Data augmentation for deep neural network acoustic modeling. **IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)**, IEEE Press, v. 23, n. 9, p. 1469–1477, 2015. Citado na página 19.

DEMUTH, H. B. et al. **Neural Network Design**. 2nd. ed. USA: Martin Hagan, 2014. ISBN 0971732116, 9780971732117. Citado 2 vezes nas páginas 4 e 5.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. **Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on**. [S.l.], 2009. p. 248–255. Citado 2 vezes nas páginas 14 e 15.

ESTEVA, A. et al. Dermatologist-level classification of skin cancer with deep neural networks. **Nature**, Nature Research, v. 542, n. 7639, p. 115–118, 2017. Citado na página 2.

FARABET, C. et al. Learning hierarchical features for scene labeling. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 35, n. 8, p. 1915–1929, 2013. Citado na página 8.

GAMBOGI, J. A. **Aplicação de redes neurais na tomada de decisão no mercado de ações.** Tese (Doutorado) — Universidade de São Paulo, 2013. Citado na página 2.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: **The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2014. Citado na página 20.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: **Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics**. [S.l.: s.n.], 2011. p. 315–323. Citado na página 10.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. Acessado em 13 de outubro de 2017. Citado 4 vezes nas páginas 8, 9, 10 e 11.

GURNEY, K. **An Introduction to Neural Networks**. Taylor & Francis, 1997. (An Introduction to Neural Networks). Acessado em 23 de outubro de 2017. ISBN 9781857285031. Disponível em: <https://books.google.com.br/books?id=HOsvIIRMMP8C>. Citado na página 2.

HAYKIN, S. **Redes Neurais - 2ed.** [S.l.]: BOOKMAN COMPANHIA ED, 2001. ISBN 9788573077186. Citado 6 vezes nas páginas 2, 3, 4, 5, 6 e 7.

HINTON, G. et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. **IEEE Signal Processing Magazine**, IEEE, v. 29, n. 6, p. 82–97, 2012. Citado na página 8.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of artificial intelligence research**, v. 4, p. 237–285, 1996. Citado na página 5.

KRIESEL, D. **A Brief Introduction to Neural Networks**. [s.n.], 2007. Acessado em 28 de outubro de 2017. Disponível em: <http://www.dkriesel.com>. Citado 2 vezes nas páginas 2 e 3.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F. et al. (Ed.). **Advances in Neural Information Processing Systems 25**. Curran Associates, Inc., 2012. p. 1097–1105. Acessado em 27 de outubro de 2017. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>. Citado 11 vezes nas páginas 1, 2, 8, 12, 15, 16, 17, 20, 21, 22 e 23.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Research, v. 521, n. 7553, p. 436–444, 2015. Citado 3 vezes nas páginas 1, 8 e 9.

LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. **Neural computation**, MIT Press, v. 1, n. 4, p. 541–551, 1989. Citado na página 8.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. **Journal of machine learning research**, v. 15, n. 1, p. 1929–1958, 2014. Citado 2 vezes nas páginas 1 e 12.

SZEGEDY, C. et al. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9. Citado na página 26.

ZHANG, G. P. Neural networks for classification: a survey. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 30, n. 4, p. 451–462, 2000. Citado na página 2.