Tópicos Especiais em Processamento Digital de Imagens:

Relatório do Primeiro Trabalho Prático

Luiz Fernando Fonseca Pinheiro de Lima

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2019

1 INTRODUÇÃO

Este é o relatório para o primeiro trabalho prático da disciplina **Tópicos Especiais em Processamento Digital de Imagens**, na qual é abordado assuntos sobre redes neurais e, mais especificamente, redes neurais convolucionais.

1.1 Definição do Trabalho

No primeiro trabalho da disciplina, deveríamos implementar a arquitetura LeNet-5 e trabalhar com a base de dados Mnist de dígitos, alterando e testando diversos hiperpâmetros da rede neural.

1.2 Objetivo Geral

Foram objetivos desse trabalho:

- permitir que os alunos implementassem a primeira rede convolucional na disciplina;
- por em prática conceitos de separação de base de dados, definição de hiperparâmetros, treino, validação e teste de uma rede;
- viabilizar contato com a biblioteca de aprendizado profundo para Python, o Keras.

1.3 Estrutura do Trabalho

Este trabalho está organizado da seguinte forma: A seção 2 traz uma breve explicação sobre a arquitetura de rede convolucional implementada, a LeNet-5 e sobre a base de dados Mnist. A seção 3 explica a metodologia do trabalho, trazendo as ferramentas utilizadas, importação e ajustes da base de dados e definição inicial de alguns hiperparâmetros. Na seção 4 são apresentados os testes feitos com as mudanças dos hiperparâmetros, seus resultados e as análises feitas a partir deles. Por fim, a seção 5 apresenta a conclusão do trabalho.

2 CONCEITOS GERAIS

A seção 2 traz uma breve explicação sobre a arquitetura de rede convolucional LeNet-5 e sobre a base de dados Mnist, usadas nesse trabalho.

2.1 LeNet-5

A arquitetura LeNet-5 consiste de duas duplas de camadas convolucionais e agrupamento (polling), seguidas por uma camada de achatamento (flatten), duas camadas densas (fully connected) e finalmente um classificador softmax (Figura 1).

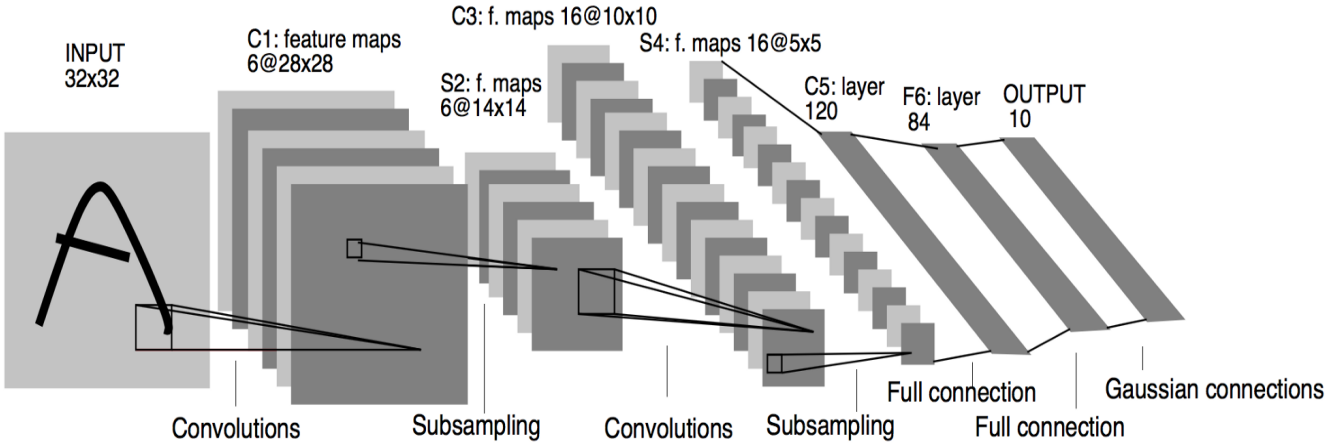

Figura 1: Arquitetura LeNet-5

A entrada da LeNet-5 é uma imagem em escala de cinza, com dimensões 32x32 pixels (1@32x32), que passa pela primeira camada convolucional com 6 filtros com tamanho 5x5 e passo (stride) de 1. A saída da primeira camada convolucional são 6 filtros de ativação com dimensões 28x28 (6@28x28).

Em seguida, é aplicada a camada de agrupamento (polling) com um tamanho de filtro 2x2 e passo (stride) de 2. A saída da primeira camada de polling são 6 filtros de ativação com dimensões 14x14 (6@14x14).

A terceira camada da LeNet-5 é uma segunda camada convolucional com 16 filtros de tamanho 5x5 e passo (stride) de 1. A saída da segunda camada convolucional são 16 filtros de ativação com dimensões 10x10 (16@10x10).

Depois, é aplicada a segunda camada de agrupamento (polling) com um tamanho de filtro 2x2 e passo (stride) de 2. A saída dessa camadasão 16 filtros de ativação com dimensões 5x5 (16@5x5).

Nesse momento, os 16 filtros com dimensões 5x5 passam por uma camada de achatamento (flatten), gerando um único conjunto de dados de tamanho 400 (16*5*5 = 400).

A próxima camada é a primeira camada densa, com 120 neurônios totalmente conectados aos 400 neurônios resultantes da camada anterior.

A rede chega a sua segunda camada densa, com 84 neurônios totalmente conectados às 120 saídas da última camada densa.

Por fim, há uma camada de saída, que é uma camada densa com 10 neurônios, onde o valor de saída de cada um deles representa os valores de confiança para cada um dos dígitos de 0 a 9, essa camada utiliza como função de ativação a função softmax.

2.2 Mnist

O Mnist é uma base de imagens com dimensões 28x28 em escala de cinza de dígitos (0 à 9) manuscritos e que conta com 70000 registros rotulados. A figura 2 conta com um exemplo de imagem dessa base.

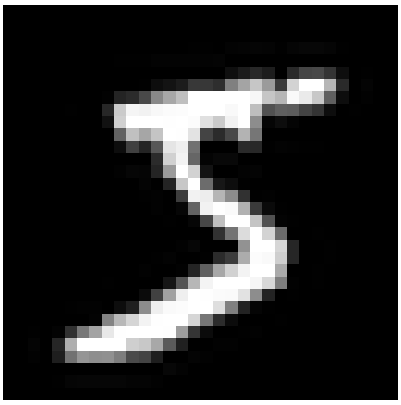

Figura 2: Exemplo de imagem do Mnist

3 METODOLOGIA

Para o desenvolvimento desse trabalho foram utilizadas as seguintes ferramentas:

- Linguagem de programação Python, versão 3.5.2, para a implementação da rede;
- Bilioteca Keras, versão 2.2.4, como interface para desenvolver a rede convolucional;
- Biblioteca TensorFlow, versão 1.11.0, como backend do Keras;
- Biblioteca numérica NumPy, versão 1.14.2, para manipulação de arrays, vetores e matrizes;
- Biblioteca Matplotlib, versão 2.2.2, para visualização de gráficos;
- Mósulo Image da biblioteca Pillow, versão 1.1.7, para manipulação e visualização de imagens.

3.1 Importando e Ajustando o Mnist

A biblioteca Keras possui um módulo chamado **datasets**, o qual disponibiliza para utilização diversas bases de dados comuns para trabalhos de aprendizado profundo, incluisve o Mnist, o qual usamos nesse trabalho.

Então, para importar a base de dados desejada foi utilizada a função **load_data()**, que retorna todas as 70000 imagens em formato de array e seus rótulos, já divididas para treino e teste, com 60000 imagens e 10000 imagens, respectivamente.

Entretanto, as imagens importadas apresentam o formato de 28x28 pixels e para a LeNet-5, elas devem ter 32x32 pixels para a camada de entrada.

Para fazer o ajuste das dimensões, as imagens em formato de array foram convertidas para imagens em escala de cinza utilizando a função **fromarray(array, 'L')** do módulo Image, onde o primeiro parâmetro é o array da imagem e o segundo indica que ela deve ser convertida em escala de cinza.

Após isso, todas as imagens foram redimensionadas utilizando a função **resize((32, 32), Image.ANTIALIAS)** do módulo Image, onde o primeiro parâmetro é uma tupla com as dimensões desejadas, aqui 32x32, e o segundo parâmetro indica o filtro que se deseja utilizar no redimensionamento, aqui o ANTIALIAS.

Por fim, para retornar as imagens para formato de array, foi utilizada a função do NumPy **asarray(img)**, que retorna o array da imagem passada como parâmetro.

3.2 Codificação dos Rótulos de Saída

O rótulo de uma imagem do Mnist é um número de 0 à 9, ou seja, o dígito que está escrito naquela imagem.

Como nosso problema é um problema de classificação multiclasse, o rótulo precisa ser codificado de modo que represente todas as saídas possíveis e indique a saída correta.

Pegando como exemplo a figura 2, seu rótulo de saída é o valor 5. Entretanto, seu rótulo deve representar todas as saídas possíveis e indicar a correta. Para isso, usa-se um array com o número de elementos igual ao número de possibilidades, no caso 10 elementos, e em cada elemento pode ter o valor 0, caso aquela seja a classe incorreta, ou o valor 1, caso aquela seja a classe correta.

Para o exemplo da figura 2, seu rótulo codificado deixaria de ser 5, e passaria a ser o array [0, 0, 0, 0, 0, 1, 0, 0, 0, 0].

A biblioteca Keras possui um módulo chamado **utils.np_utils** que contém a função **to_categorical(rótulos)**, que faz essa codificação para o array de rótulos passado como parâmetro.

3.3 Implementação Inicial

A implementação da rede seguiu todas as camadas e número de filtros e apresentada na arquitetura LeNet-5 da figura 1.

A partir de pesquisas e diversos exemplos, os hiperparâmetros foram definidos inicialmente da seguinte forma:

- Funções de ativação: a relu foi utilizada nas duas camadas convolucionais, bem como nas duas primeiras camadas densas, já na última camada densa foi utilizada a função softmax;
- Otimizador: o Adam foi utilizado como otimizador para a rede;
- Função de perda: a entropia cruzada foi utilizada como função de perda para a rede.

A proporção de dados de validação dentro do conjunto de treinamento foi sempre de 16.665%, ou seja, 9999 imagens foram definidas como imagens do conjunto de validação e 50001 imagens foram definidas como imagens do conjunto de treinamento. Esse hiperparâmetro é definido na função **fit()** da rede.

Para visualizar os resultados de acurácia de treino e validação a partir da época 0, foi usado o parâmetro **initial_epoch()**, também definido na função **fit()**, com o valor -1.

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Esta seção descreve os testes realizados com as modificações dos hiperparâmetros o problema do trabalho.

4.1 Teste 1

Para primeiro teste foram utilizados 50 épocas e batch de 32 imagens para o treinamento. Além disso, foram utilizados os hiperparâmetros padrões do Keras para o otimizador Adam, a taxa de aprendizado de 0.001 e decaimento da taxa de aprendizado de 0.0.

Algo influenciou negativamente a rede e ela obteve um comportamento divergente do que se esperava. Mantendo os valores de acurácia de treino e validação, suas curvas de acurácia de treino e acurácia de validação foram lineares (figura 3). Ou seja, ela não aprendeu.

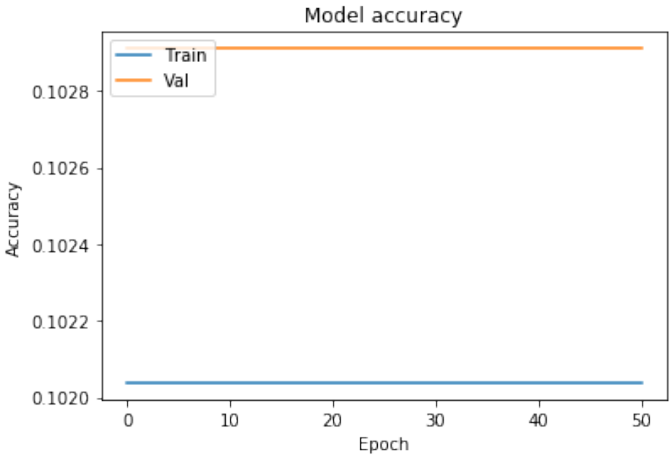

Figura 3: Curvas de acurácias de treino e validação para o teste 1

Uma hipótese do que pode ter acontecido é que ocorreu algum problema com a inicialização dos pesos da rede, que podem ter sido iniciados com valores muito baixos.

Para a base de teste a rede previu 1010 imagens corretamente e 8990 incorretamente, tendo uma acurácia de 10.1%.

4.2 Teste 2

Para garantir que a arquitetura estava implementada corretamente e que não havia nenhum problema com as imagens, o segundo teste repetiu exatamente as mesmas configurações do teste 1.

Nessa execução a rede apresentou um comportamento esperado (figura 4).

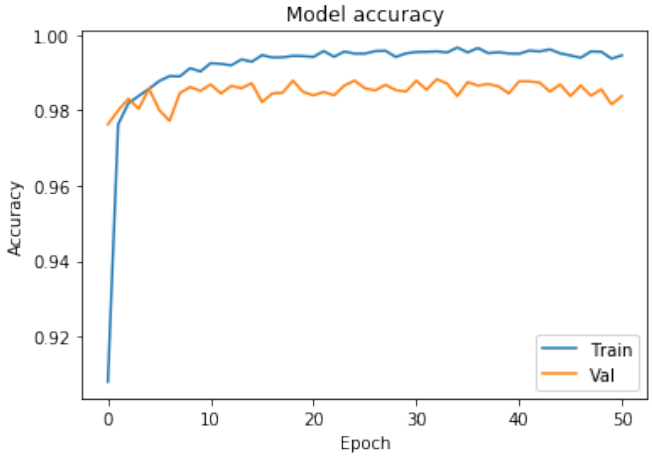

Figura 4: Curvas de acurácias de treino e validação para o teste 2

Para a base de teste a rede obteve uma acurácia de 98.62%.

4.3 Teste 3

No terceiro teste os hiperparâmetros foram quase todos mantidos dos testes 4.1 e 4.2, exceto o método de inicialização dos pesos da rede convolucional.

Nos testes 4.1 e 4.2 foi utilizada, para todas as camadas de convolução e densas, a inicialização padrão do Keras, o método **Glorot uniform**, já para o terceiro teste foi usado o **random uniform**.

A rede obteve uma acurácia de 98.9% para o conjunto de teste. As acurácias de treino e validação podem ser observadas na figura 5.

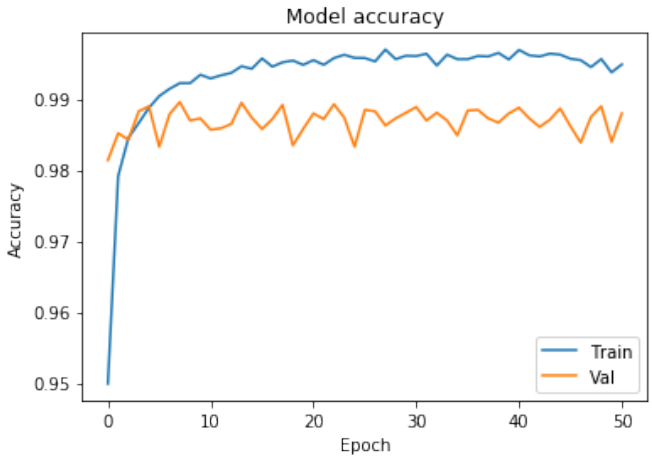

Figura 5: Curvas de acurácias de treino e validação para o teste 3

A rede teve um pequeno ganho na sua acurácia para o conjunto de teste no teste 4.3, em relação ao teste 4.2, com a mudança do metódo de inicialização de pesos do **Glorot uniform** para o **random uniform**.

4.4 Teste 4

No quarto teste os hiperparâmetros foram quase todos mantidos do teste 4.3, exceto o tamanho do batch.

No teste 4.3 foi utilizado batch de tamanho 32, já para o quarto teste, o tamanho do batch foi definido como 2% do tamanho do conjunto de treino, ou seja, 100 imagens.

Nesse teste, as curvas de acurácia de treino e validação foram semelhantes as curvas dos testes 4.2 e 4.3 (figura 6).

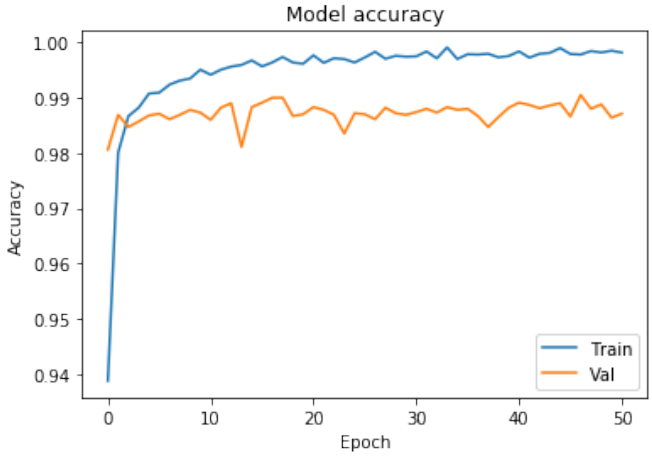

Figura 6: Curvas de acurácias de treino e validação para o teste 4

Entretanto, sua acurácia para as imagens de teste reduziu para 98.81%.

4.5 Teste 5

Para quinto teste, os hiperparâmetros do teste 4.4 foram quase todos mantidos, apenas o tamanho do batch foi novamente aumentado.

No teste 4.4 foi utilizado batch de tamanho 100, já para o quinto teste, o tamanho do batch foi definido como 8% do tamanho do conjunto de treino, ou seja, 400 imagens.

As curvas de acurácia de treino e validação do quinto teste podem ser vistas na figura 7).

Em comparação com o teste 4.4, a rede tem uma melhora na sua acurácia, atingindo 99.03% para a base de testes.

4.6 Teste 6

No sexto teste, a rede teve seus hiperparâmetros definidos com os valores da rede 4.5, exceto a taxa de aprendizado, que antes utilizava o valor padrão do Keras, 0.001, e para o sexto teste foi utilizado um valor ainda menor, 0.00001.

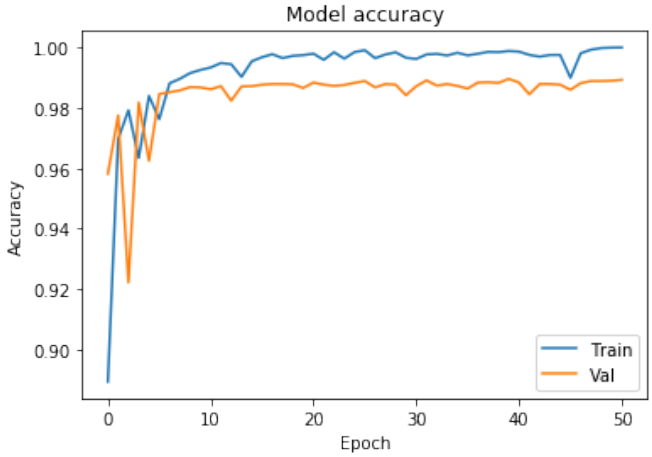

Figura 7: Curvas de acurácias de treino e validação para o teste 5

A ideia deste teste é mostrar como um baixo valor de taxa de aprendizado dificulta que a convergência da rede.

De fato, sua acurácia de testes reduziu em ralação ao teste 4.5, pontuando 96.84%. E suas acurácias de treino e validação podem ser observadas na figura 8.

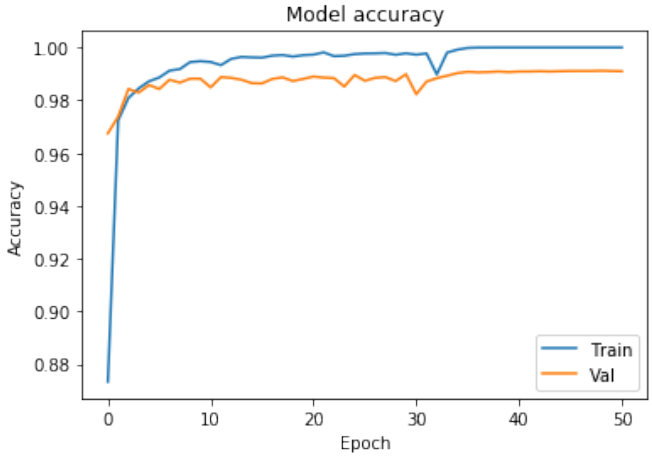

Figura 8: Curvas de acurácias de treino e validação para o teste 6

4.7 Teste 7

O sétimo teste, assim como o teste 4.6, teve como objetivo mostrar como a convergência da rede é afetada pela taxa de aprendizado, entretanto, para um valor alto de taxa de aprendizado.

Desse modo, os hiperparâmetos do sétimo teste foram iguais ao do teste 4.6 e, consequentemente, ao teste 4.5, exceto a taxa de aprendizado, para qual foi utilizado o valor 0.1.

Com a alta taxa de aprendizado, a acurácia da rede para a base de testes foi pior que a rede 4.6, pontuando apenas 8.92%. E suas acurácias de treino e validação podem ser observadas na figura 9.

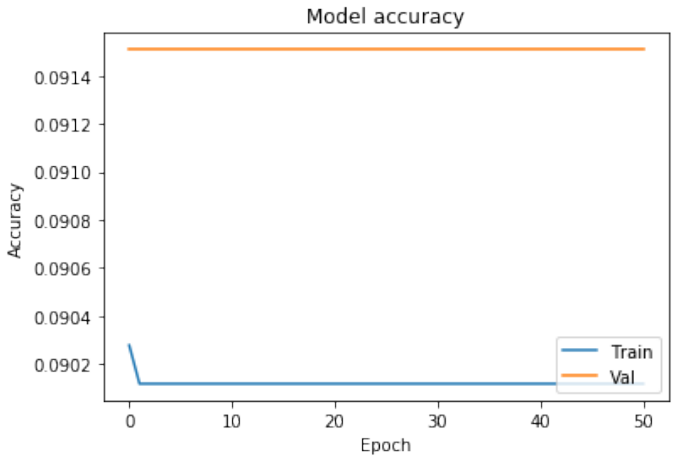

Figura 9: Curvas de acurácias de treino e validação para o teste 7

4.8 Teste 8

O oitavo teste, repete os hiperparâmetros do teste 4.7, exceto o valor de decaimento da taxa de aprendizado, que foi definido com 0.001.

A ideia foi começar com a alta taxa de aprendizado do teste 4.7, mas melhorando o resultado de acurácia, devido a dimunição do valor da taxa de aprendizado.

Diferente do esperado, a acurácia de testes teve uma piora em relação ao teste 4.7, o oitavo teste pontuou apenas 0.98%. E suas acurácias de treino e validação podem ser observadas na figura 10.

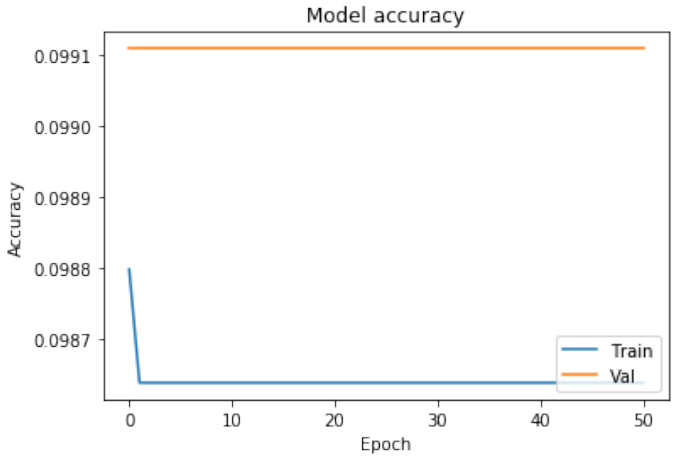

Figura 10: Curvas de acurácias de treino e validação para o teste 8

4.9 Teste 9

O nono teste, repete os hiperparâmetros do teste 4.5, exceto o valor de decaimento da taxa de aprendizado, que foi definido com 0.00001.

A acurácia de testes teve uma significante melhora, o nono teste pontuou 99.23%. E suas acurácias de treino e validação podem ser observadas na figura 11.

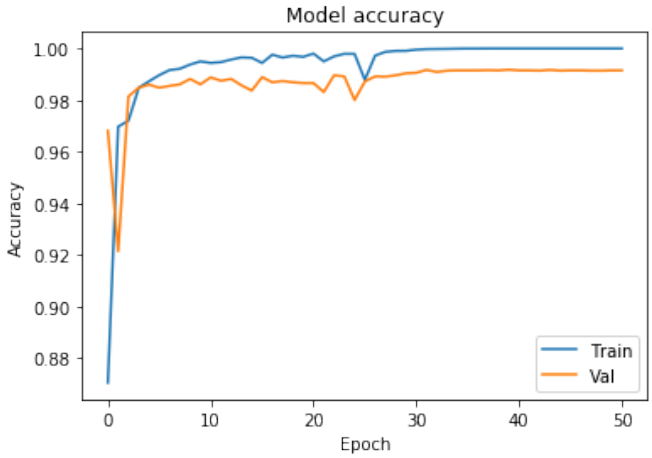

Figura 11: Curvas de acurácias de treino e validação para o teste 9

4.10 Teste 10

O décimo teste, repete novamente os hiperparâmetros do teste 4.5, exceto o valor de decaimento da taxa de aprendizado, que, dessa vez, foi definido com 0.0001.

A ideia foi identificar o comportamento da rede com uma maior taxa dacaimento, comparado ao teste 4.9.

Para a base de testes a rede do décimo teste obteve uma acurácia de 99.12%. E suas acurácias de treino e validação podem ser observadas na figura 12.

4.11 Teste 11

A ideia do décimo primeiro teste é repetir a estrutura da rede que obteve a melhor acurácia até o momento, no caso o teste 4.9, exceto o algoritmo otimizador e seus hiperparâmetros. Estava sendo utilizado o algoritmo Adam e, para o décimo primeiro teste, foi utilizado o SGD.

Então, o décimo primeiro teste utilizou o otimizador SGD com taxa de aprendizado e taxa de decaimento padrão, 0.01 e 0.0, respectivamente.

Para a base de testes a rede do décimo primeiro teste obteve uma acurácia de 9.74%. E suas acurácias de treino e validação podem ser observadas na figura 13.

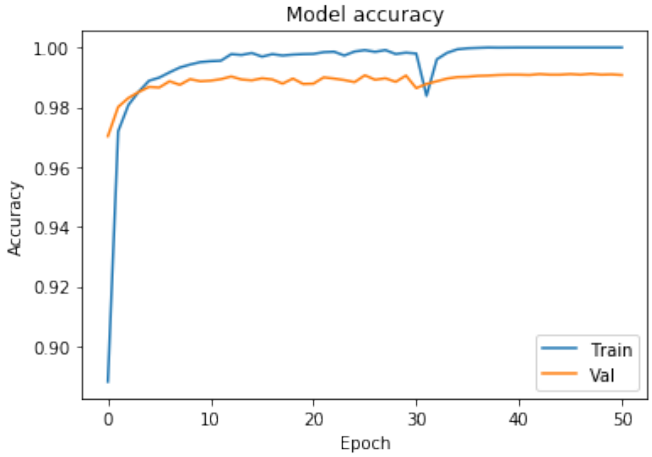

Figura 12: Curvas de acurácias de treino e validação para o teste 10

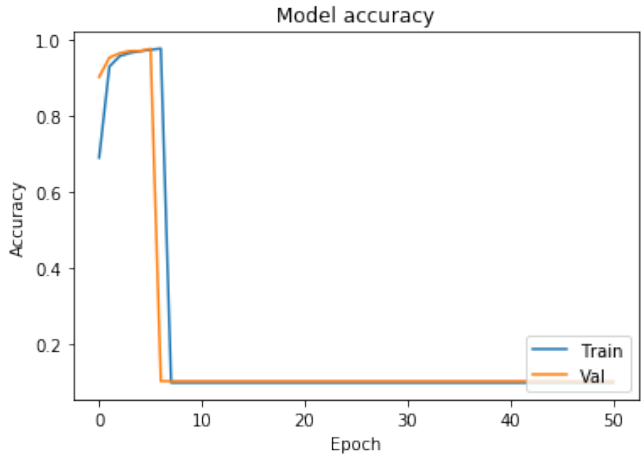

Figura 13: Curvas de acurácias de treino e validação para o teste 11

4.12 Teste 12

A ideia do décimo segundo teste é repetir os hiperparâmetros da rede do teste 4.11, exceto a taxa de aprendizado, que estava sendo utilizado o padrão do SGD e para o décimo segundo teste foi utilizado o valor 0.001.

Para a base de testes a rede do décimo segundo teste obteve uma acurácia de 97.88%. E suas acurácias de treino e validação podem ser observadas na figura 14.

4.13 Teste 13

A ideia do décimo teceiro teste, assim como o teste 4.12, foi repetir os hiperparâmetros da rede do teste 4.11, exceto a taxa de aprendizado, que estava sendo utilizado o padrão do SGD e para o décimo segundo teste foi utilizado um valor mais alto, de 0.1.

Para a base de testes a rede do décimo terceiro teste obteve uma acurácia de 11.35%, o que mostra a dificuldade da rede em convergir com um valor mais alto de taxa

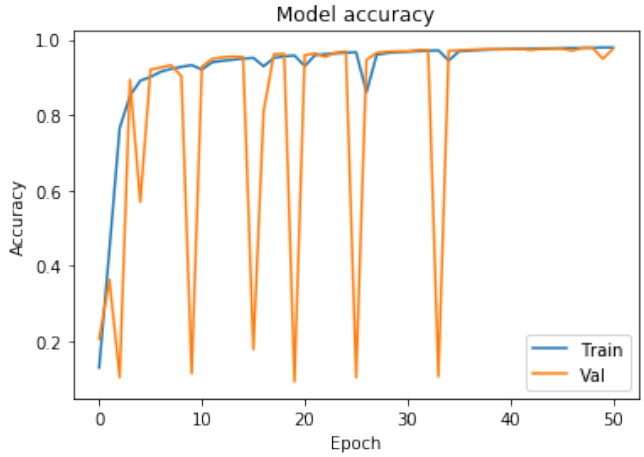

Figura 14: Curvas de acurácias de treino e validação para o teste 12

de aprendizado. E suas acurácias de treino e validação podem ser observadas na figura 15.

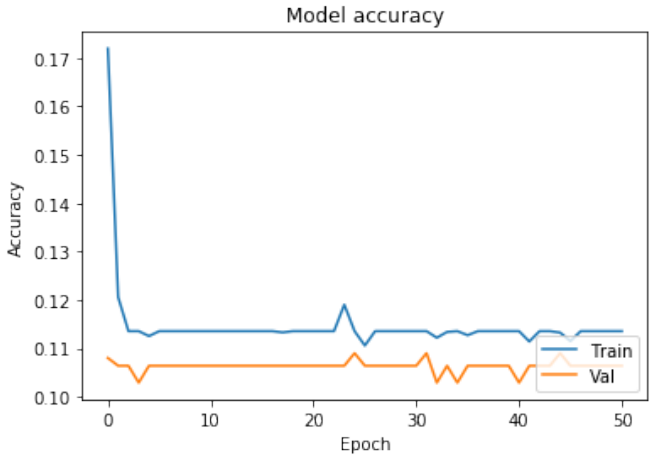

Figura 15: Curvas de acurácias de treino e validação para o teste 13

4.14 Teste 14

A ideia do décimo quarto teste foi repetir os hiperparâmetros da rede do teste 4.13, exceto a taxa de decaimento, que estava sendo utilizado o valor padrão 0.0 e para o décimo quarto teste foi utilizado o valor 0.001.

Mesmo com a taxa de decaimento, a rede do décimo quarto teste não alcançou um desempenho satisfatório, mantendo a acurácia de 11.35%. E suas acurácias de treino e validação podem ser observadas na figura 16.

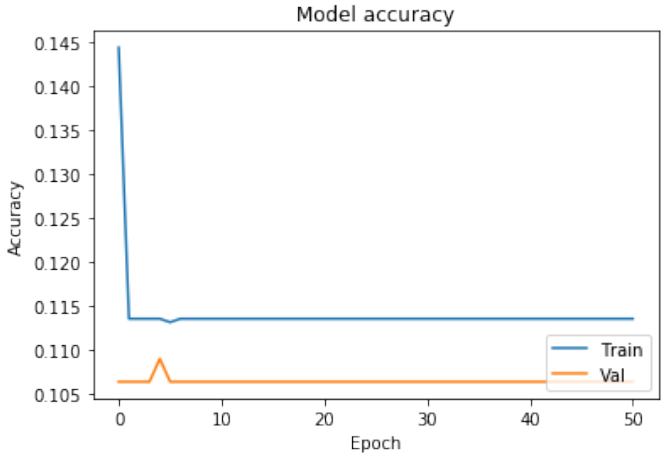

Figura 16: Curvas de acurácias de treino e validação para o teste 14

4.15 Teste 15

A ideia do décimo quinto teste também foi repetir os hiperparâmetros da rede do teste 4.13, exceto a taxa de decaimento, que estava sendo utilizado o valor padrão 0.0 e para o décimo quinto teste foi utilizado o valor 0.01.

Mesmo com a taxa de decaimento, a rede do décimo quarto teste não alcançou um desempenho satisfatório, mantendo a acurácia de 11.35%. E suas acurácias de treino e validação podem ser observadas na figura 17.

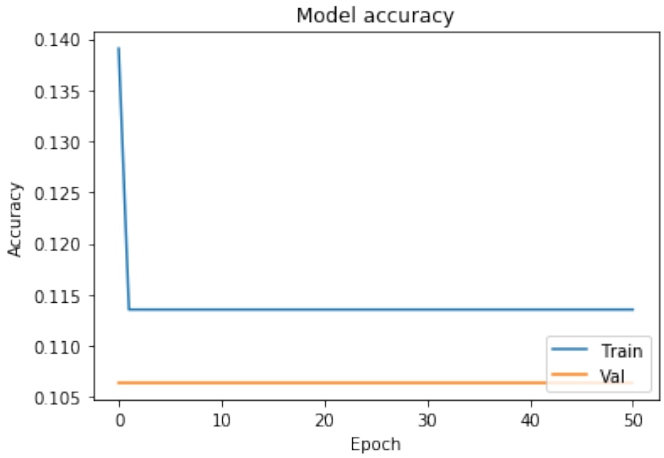

Figura 17: Curvas de acurácias de treino e validação para o teste 15

4.16 Teste 16

A rede do décimo sexto teste utilizou os mesmos hiperparâmetros da rede que atingiu a melhor acurácia, o teste 4.9, com tamanho de batch de 400, inicializador de pesos **random uniform**, algoritmo otimizador **Adam**, taxa de aprendizado 0.001, taxa

de decaimento da taxa de aprendizado 0.00001. Apenas o número de épocas foi definido com valor diferente.

O teste 4.9 utilizou 50 épocas, enquanto o décimo sexto teste utilizou metade desse valor, ou seja, 25 épocas.

A rede teve uma redução de precisão em relação ao teste 4.9, pontuando 98.59% de acurácia para a base de teste. Suas acurácias de treinamento e validação podem ser vistas na figura 18.

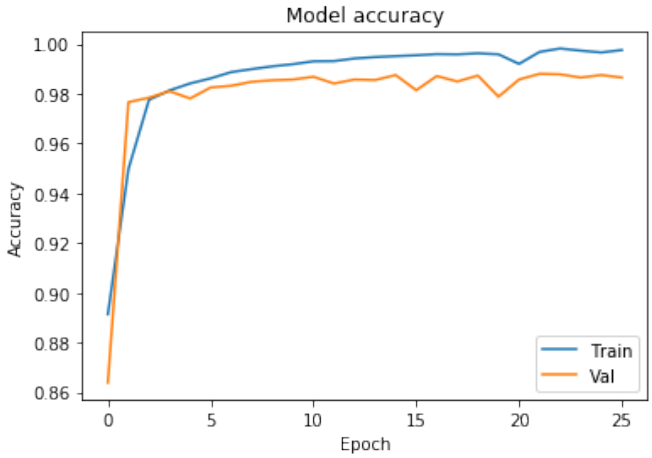

Figura 18: Curvas de acurácias de treino e validação para o teste 16

4.17 Teste 17

Assim como o teste 4.16, a rede do décimo sétimo teste utilizou os mesmos hiperparâmetros da rede que atingiu a melhor acurácia, o teste 4.9, sendo o número de épocas o único hiperparâmetro definido com valor diferente.

O teste 4.9 utilizou 50 épocas, enquanto o décimo sétimo teste utilizou o valor truncado de 75% desse valor, ou seja, 37 épocas.

A rede também teve uma redução de precisão em relação ao teste 4.9, pontuando 99.09% de acurácia para a base de teste. Suas acurácias de treinamento e validação podem ser vistas na figura 19.

4.18 Teste 18

Assim como os testes 4.16 e 4.17, a rede do décimo oitavo teste utilizou os mesmos hiperparâmetros da rede que atingiu a melhor acurácia, o teste 4.9, sendo o número de épocas o único hiperparâmetro definido com valor diferente.

O teste 4.9 utilizou 50 épocas, enquanto o décimo oitavo teste utilizou o valor de 75 épocas.

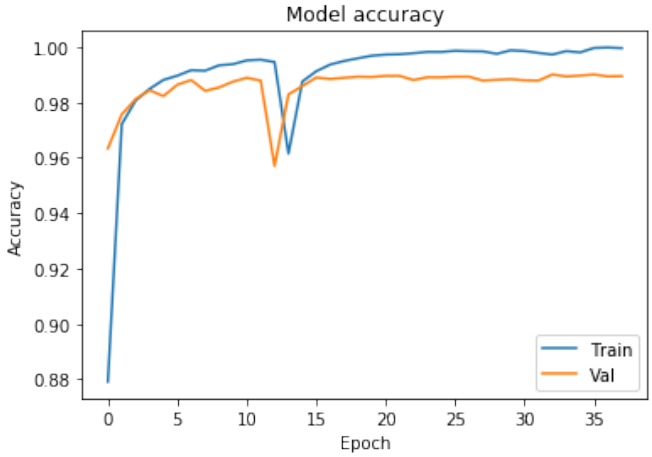

Figura 19: Curvas de acurácias de treino e validação para o teste 17

A rede também teve uma redução de precisão em relação ao teste 4.9, pontuando 99.1% de acurácia para a base de teste. Suas acurácias de treinamento e validação podem ser vistas na figura 20.

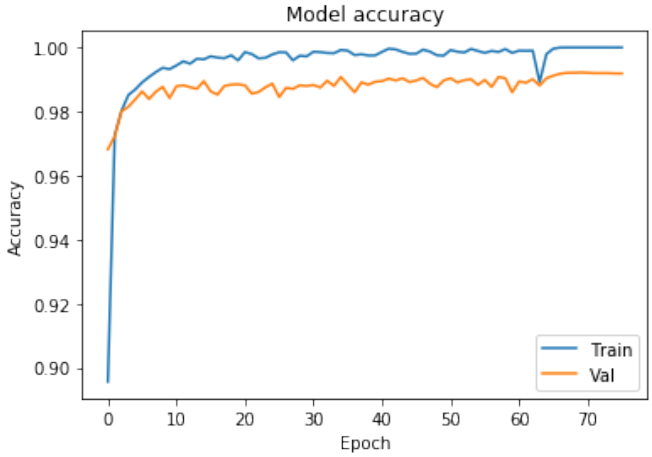

Figura 20: Curvas de acurácias de treino e validação para o teste 18

4.19 Teste 19

Assim como os testes 4.16, 4.17 e 4.18, a rede do décimo nono teste utilizou os mesmos hiperparâmetros da rede que atingiu a melhor acurácia, o teste 4.9, sendo o número de épocas o único hiperparâmetro definido com valor diferente.

O teste 4.9 utilizou 50 épocas, enquanto o décimo nono teste utilizou o valor de 100 épocas.

A rede também teve uma redução de precisão em relação ao teste 4.9, pontuando 99.16% de acurácia para a base de teste. Suas acurácias de treinamento e validação podem ser vistas na figura 21.

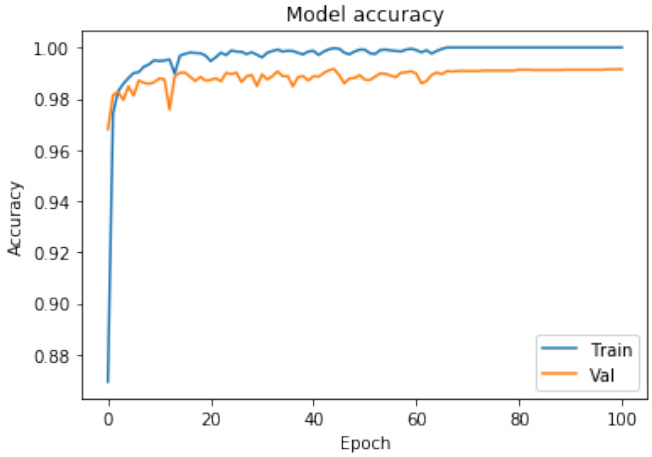

Figura 21: Curvas de acurácias de treino e validação para o teste 19

4.20 Teste 20

Até o momento, a rede que atingiu a melhor acurácia contiuna sendo o teste 4.9, com 50 épocas, tamanho de batch de 400, inicializador de pesos **random uniform**, algoritmo otimizador **Adam**, taxa de aprendizado 0.001 e taxa de decaimento da taxa de aprendizado 0.00001, tendo pontuado 99.29% de acurácia para as imagens de teste.

O vigésimo teste repete os hiperparâmetros do teste 4.9, e modifica o algoritmo que é utilizado nas camadas de polling que são aplicadas após as camadas de convolução. No teste 4.9 foi utilizado o **MaxPolling** e no vigésimo teste o **AveragePooling**.

Com o **AveragePolling** a rede teve uma redução de precisão em relação ao teste 4.9, pontuando 98.71% de acurácia para a base de teste. Suas acurácias de treinamento e validação podem ser vistas na figura 22.

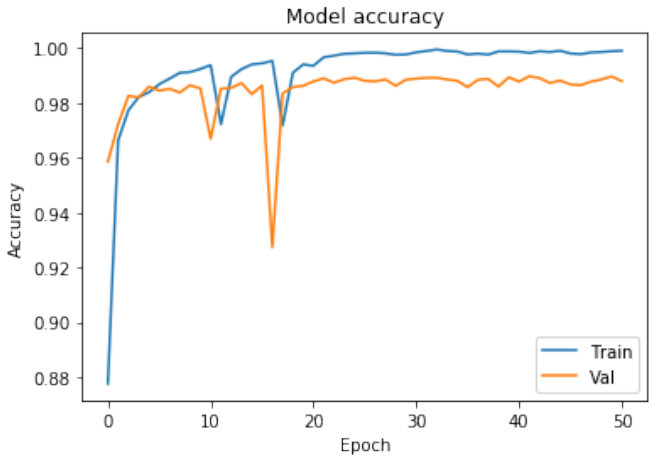

Figura 22: Curvas de acurácias de treino e validação para o teste 20

4.21 Teste 21

O vigésimo primeiro teste repete os hiperparâmetros do teste 4.9, e adicionia camadas **Dropout**, com taxa de 0.3, após as camadas de convolução e camadas densas.

A rede do vigésimo primeiro teste teve um pequeno aumento de 0.05 pontos percentuais em relação ao teste 4.9, pontuando 99.28% de acurácia para a base de teste. Suas acurácias de treinamento e validação podem ser vistas na figura 23.

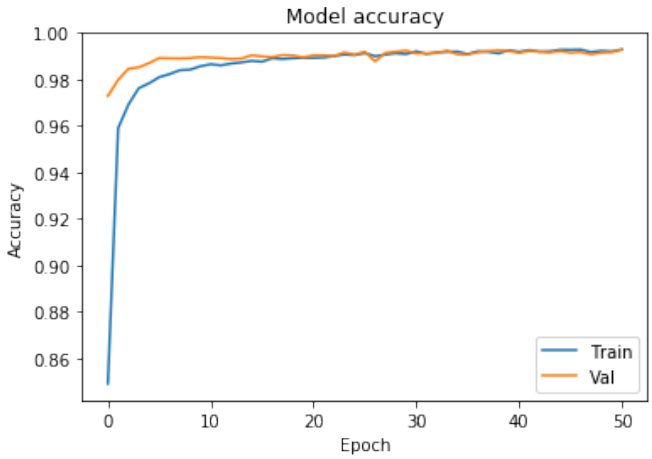

Figura 23: Curvas de acurácias de treino e validação para o teste 21

4.22 Teste 22

Assim como o teste 4.21, vigésimo segundo teste repete os hiperparâmetros do teste 4.9, mas adicionia camadas **Dropout**, com taxa de 0.5, após as camadas de convolução e camadas densas.

A rede do vigésimo segundo teste teve uma redução de precisão em relação ao teste 4.9, pontuando 99.11% de acurácia para a base de teste. Suas acurácias de treinamento e validação podem ser vistas na figura 24.

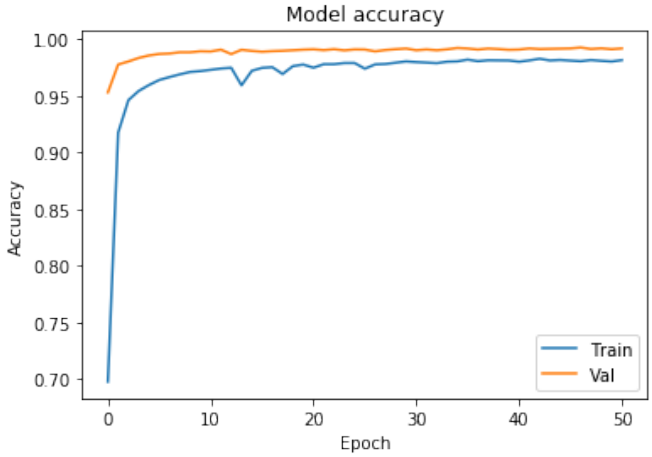

Figura 24: Curvas de acurácias de treino e validação para o teste 22

5 CONCLUSÕES

Os objetivos do trabalho foram alcançados, uma vez que pôde-se testar diversas combinações para os hiperparâmetros e observar como eles afetam o desempenho da rede. Além disso, o trabalho proporcionou uma grande experiência com as ferramentas de desenvolvimento que são, comumente, usadas no desenvolvimento de redes convolucionais, e também com a ferramenta Overleaf/Latex, para a escrita do relatório.

O teste com melhor desempenho de acurácia para a base de testes foi o 4.21, que obteve 99.28% de acurácia.

Pretendia-se utilizar a abordagem de data augmentation, para gerar mais imagens de treino para a rede, com o objetivo de criar mais experiência para a mesma.

Entretanto, por não ter nenhum conhecimento prévio sobre os módulos da biblioteca Keras para essa abordagem e por ter enfrentado alguns contratempos, esses testes não foram feitos. Ficando como objetivo utilizar essa técnica em um próximo trabalho.