

Laboratório 03 - CNN

Aprendizado de Máquina - INFO7004

Prof. Luiz Eduardo S. Oliveira, Ph.D

Discente: Marc Queiroz

08 de setembro de 2020

1 Introdução

A prática do laboratório 3 apresenta uma base de dados de imagens com escritas manuais com os meses do ano. A tarefa consiste em analisar a implementação e treinamento de CNNs e também a geração de imagens utilizando a técnica de *data augmentation* para classificação dessas imagens.

Os códigos fontes utilizados nesse laboratório e a versão mais atual desse relatório pode ser encontrado no GitHub, https://github.com/marc-queiroz/ml_lab3/.

O trabalho está subdividido em seções, as primeiras apresentam as CNNs do trabalho. Uma explicação dos métodos de augmentação também faz parte do trabalho, em seção própria. Apresentadas as CNNs e técnicas utilizadas segue-se para os experimentos, neles são apresentadas os testes com e sem augmentação para as duas redes. Em seção própria, é apresentado os resultados dos experimentos. Também foi escolhido duas redes pré-treinadas da ImageNet, Inception v4, VGG16 para realizar os experimentos de (Transfer Learning/Fine-Tuning). Chegando a parte final, com a extração de características utilizando a rede pré-treinada Inception v4. Com a última seção mostrando os resultados do teste com o classificador SVM com kernel radial.

2 LeNet-5

A rede LeNet-5 é uma rede convolucional de 7 camadas, descrita no artigo [1]. O treinamento dessa rede é feita através de uma rede neural que aplica convoluções e filtros consecutivos, para assim encontrar os melhores pesos de acordo com sua base de treinamento.

O propósito inicial da LeNet-5 foi identificar dígitos escritos manualmente, figura 1. Para implementar essa CNN utilizou-se a API Keras, que permite uma representação de fácil leitura e manutenção. O código da rede pode ser encontrado na listagem 1.

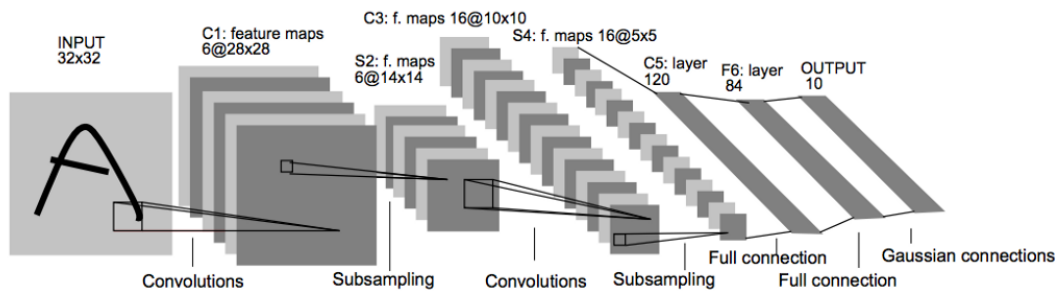


Figura 1: As 7 camadas do modelo LeNet-5[1]

```

1 model = keras.Sequential()
2 model.add(layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu',
3   input_shape=(32,32,1)))
4 model.add(layers.AveragePooling2D())
5 model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
6 model.add(layers.AveragePooling2D())
7 model.add(layers.Flatten())
8 model.add(layers.Dense(units=120, activation='relu'))
9 model.add(layers.Dense(units=84, activation='relu'))
10 model.add(layers.Dense(units=12, activation='softmax'))

```

Listing 1: LeNet-5 implementado em Keras

A rede é composta por 7 camadas, sendo elas:

1. C1 convolucional (tamanho de saída: 28x28x6)
2. S2 pooling (agrupamento) (tamanho de saída: 14x14x6)
3. C3 convolucional (tamanho de saída: 10x10x6)
4. S4 pooling (agrupamento) (tamanho de saída: 5x5x16)
5. C5 classificação (tamanho de saída: 120)
6. F6 classificação (tamanho de saída: 84)
7. OUTPUT (tamanho de saída: 12)

Observa-se que o tamanho do vetor OUTPUT tem a quantidade de categorias sendo classificadas, 12 meses.

3 Outra CNN

A CNN escolhida foi adaptada do autor, Francois Lemarchand, no site, <https://www.kaggle.com/frlemarchand/simple-cnn-using-keras>, refere-se a um experimento para reconhecimento de cactus em imagens. Seguindo o desenvolvimento sugerido para redes convolucionais o autor procurou criar uma rede convolucional para resolver um problema binário.

Para utilizar essa rede nos experimentos, a sua camada de saída foi alterada para 12 classes, ao invés de 2, originais, listagem 8.

```

1 dropout_dense_layer = 0.6
2
3 model = Sequential()
4 model.add(Conv2D(32, (3, 3), input_shape=input_shape))
5 model.add(BatchNormalization())
6 model.add(Activation('relu'))
7 model.add(Conv2D(32, (3, 3)))
8 model.add(BatchNormalization())
9 model.add(Activation('relu'))
10 model.add(Conv2D(32, (3, 3)))
11 model.add(BatchNormalization())
12 model.add(Activation('relu'))
13 model.add(MaxPooling2D(pool_size=(2, 2)))
14
15 model.add(Conv2D(64, (3, 3)))
16 model.add(BatchNormalization())
17 model.add(Activation('relu'))
18 model.add(Conv2D(64, (3, 3)))
19 model.add(BatchNormalization())
20 model.add(Activation('relu'))
21 model.add(Conv2D(64, (3, 3)))
22 model.add(BatchNormalization())
23 model.add(Activation('relu'))
24 model.add(MaxPooling2D(pool_size=(2, 2)))
25
26 model.add(Conv2D(128, (3, 3)))
27 model.add(BatchNormalization())
28 model.add(Activation('relu'))
29
30 model.add(Flatten())
31 model.add(Dense(1024))
32 model.add(Activation('relu'))
33 model.add(Dropout(dropout_dense_layer))
34
35 model.add(Dense(256))
36 model.add(Activation('relu'))
37 model.add(Dropout(dropout_dense_layer))
38
39 model.add(Dense(12))
40 model.add(Activation('sigmoid'))

```

Listing 2: CNN escolhida

A rede é composta por 15 camadas, na qual a camada de OUTPUT foi modificada para 12 classes, incluindo duas camadas de Dropout, que a LeNet não possui.

4 Data Augmentation

Para realização do laboratório, um dos requisitos era a geração de imagens a partir da base de treinamento.

Para atingir esse objetivo, primeiramente iniciou-se um teste com o exemplo de geração de imagens demonstradas em sala de aula, utilizando o próprio gerador do Keras.

O código fonte apresentado em aula, fazia o uso de alguns filtros: *flipped vertical*, *horizontal*, *rotation*. Depois de análise dos resultados, observou-se que para escrita manual os filtros apresentados eram de pouca qualidade.

Iniciando uma pesquisa sobre o assunto, *data augmentation for handwriting*, no Google, descobriu-se várias pesquisas na área, sugerindo uma coleção de filtros comuns a serem utilizados na escrita manual, para geração de dados.

O trabalho de Kang et al.[2], sugere alguns métodos, entre eles: *shear*, *rotation* e *elastic transformations*. Após uma pesquisa para transformação das imagens chegou-se ao pacote do python, chamado Augmentor. O projeto encontra-se no GitHub,

<https://github.com/mdbloice/Augmentor> e pode ser instalado facilmente. Com esse pacote foi possível criar um script capaz de gerar dois conjuntos de dados, a partir das imagens de entrada.

Para lidar com os arquivos de entrada algumas ferramentas precisaram ser criadas, para transferir o label para o nome do arquivo, exemplo, imagem AD00017.jpg, label 1 (Fevereiro), torna-se AD00017_1.jpg. Com esse procedimento preparado, copia-se todos os arquivos do train.txt para novo diretório, já no novo formato de arquivo, aplica-se o script de augmentation para gerar novas imagens, salvando em um novo diretório que pode ser configurado pelo usuário. A partir desse novo diretório foi necessário criar outra ferramenta para extrair o train-aug.txt no formato dos arquivos de entrada.

A biblioteca Augmentor apresenta uma estrutura para geração de arquivos utilizando a imagem original e aplicando uma série de filtros configurados pelo usuário. Cada filtro possui um conjunto de parâmetros ajustáveis, uma das mais interessantes é a capacidade de escolher a probabilidade do filtro ser aplicado a imagem gerada. Dessa forma é possível aumentar a variabilidade das imagens. Abaixo é possível ver o código fonte para a primeira versão das imagens geradas.

```
1 p = Augmentor.Pipeline(source_directory=origem.absolute().as_posix(),
2   output_directory=destino.absolute().as_posix())
3 p.invert(probability=1)
4 p.rotate(probability=1, max_left_rotation=3, max_right_rotation=3)
5 p.skew_tilt(probability=1, magnitude=0.1)
6 p.random_distortion(probability=0.3, grid_width=10, grid_height=5, magnitude=8)
7 p.zoom(probability=1, min_factor=0.4, max_factor=1)
8 p.invert(probability=1)
9 p.sample(15780)
```

Listing 3: CNN escolhida

A partir da base de entrada, arquivos do train.txt, 1578 arquivos, foram gerados 15780 arquivos.

Na segunda versão do script de geração de dados aumentados, foi incluído a distorção gaussiana, com um probabilidade de 30% de ativação do filtro para cada imagem sendo gerada.

```
1 p = Augmentor.Pipeline(source_directory=origem.absolute().as_posix(),
2   output_directory=destino.absolute().as_posix())
3 p.invert(probability=1)
4 p.rotate(probability=1, max_left_rotation=3, max_right_rotation=3)
5 p.skew_tilt(probability=1, magnitude=0.2)
6 p.random_distortion(probability=0.3, grid_width=5, grid_height=5, magnitude=8)
7 p.gaussian_distortion(probability=0.3, grid_width=7, grid_height=7, magnitude=10,
8   corner='dl', method='out')
9 p.zoom(probability=1, min_factor=0.4, max_factor=1)
10 p.invert(probability=1)
11 p.sample(20000, multi_threaded=False)
```

Listing 4: CNN escolhida

No total foram geradas 35780 imagens.

5 Experimentos

Nas seguintes subseções serão apresentados os experimentos com as CNNs, incluindo com e sem dados aumentados.

5.1 LeNet-5

Para encontrar o melhor conjunto de parâmetros para a CNN LeNet-5 com dados originais, procure-se executar algumas variações dos parâmetros. Com objetivo de encontrar o tamanho de imagem de entrada, o tamanho do lote, a quantidade de épocas e a função de ativação da rede.

Será apresentado uma tabela com os valores encontrados para alguns dos experimentos, incluindo a apresentação dos gráficos e matrix confusão para o melhor resultado. Os parâmetros a serem alterados são: `batch_size`, `epochs`, `image size` e `activation`.

Experimento	Loss	Acurácia	Tempo de Execução (s)
LeNet-5, batch_size=32, epochs=64, image=32x32, activation=relu	1.17	0.77	16.57
LeNet-5, batch_size=32, epochs=100, image=32x32, activation=relu	1.33	0.76	23.97
LeNet-5, batch_size=64, epochs=100, image=32x32, activation=relu	1.54	0.76	12.12
LeNet-5, batch_size=64, epochs=64, image=28x28, activation=relu	1.09	0.73	12.12
LeNet-5, batch_size=16, epochs=64, image=32x32, activation=relu	1.17	0.77	16.57
LeNet-5, batch_size=32, epochs=64, image=40x40, activation=relu	1.43	0.78	17.08
LeNet-5, batch_size=128, epochs=64, image=50x50, activation=relu	1.00	0.79	13.42

Tabela 1: Resultados para LeNet-5

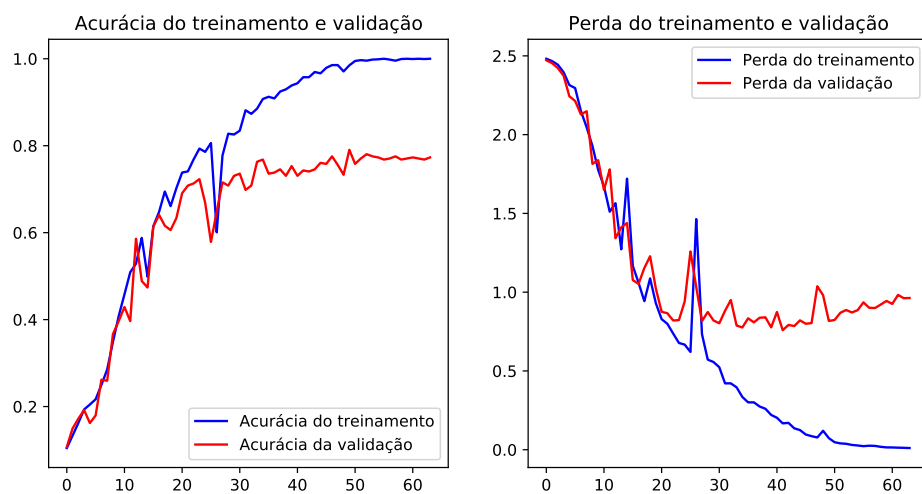


Figura 2: Acurácia e perda comparadas com treinamento e validação

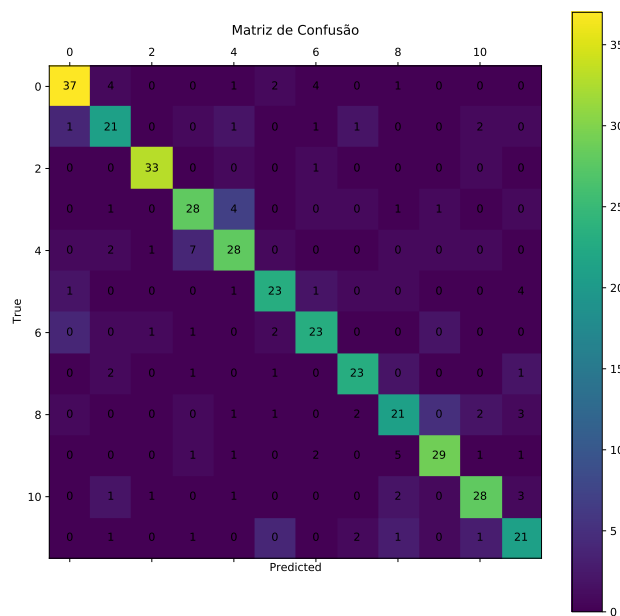


Figura 3: Matrix de confusão

5.2 Outra CNN

Será apresentado uma tabela com os valores encontrados para alguns dos experimentos, incluindo a apresentação dos gráficos e matrix confusão para o melhor resultado. Os parâmetros a serem alterados são: batch_size, epochs, image size e activation.

Experimento	Loss	Acurácia	Tempo de Execução (s)
Outra CNN, batch_size=32, epochs=100, image=32x32, activation=relu	1.14	0.80	13.47
Outra CNN, batch_size=64, epochs=100, image=32x32, activation=relu	1.38	0.76	11.74
Outra CNN, batch_size=16, epochs=100, image=32x32, activation=relu	1.44	0.71	21.95
Outra CNN, batch_size=16, epochs=160, image=32x32, activation=relu	0.98	0.82	34.69
Outra CNN, batch_size=16, epochs=160, image=32x32, activation=relu	0.98	0.82	34.69
Outra CNN, batch_size=16, epochs=160, image=50x50, activation=relu	2.33	0.09	80.93
Outra CNN, batch_size=128, epochs=160, image=50x50, activation=relu	0.96	0.83	50.10
Outra CNN, batch_size=128, epochs=160, image=48x48, activation=relu	0.55	0.89	31.37

Tabela 2: Resultados para outra CNN

Esse CNN faz uso de um callback chamado EarlyStop, uma implementação do Keras que utiliza a piora na curva de loss como critério de parada. Apesar das épocas utilizadas serem grandes, a maioria das execuções terminam sem atingir o número limite de épocas.

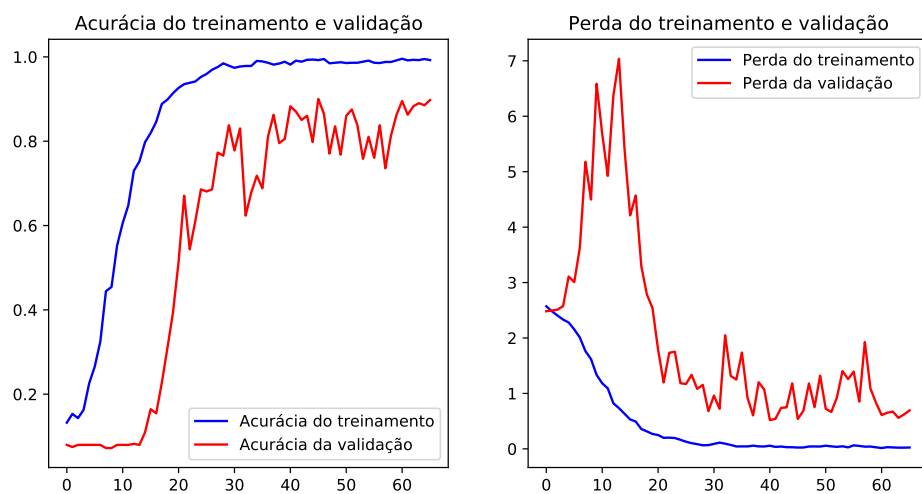


Figura 4: Acurácia e perda comparadas com treinamento e validação

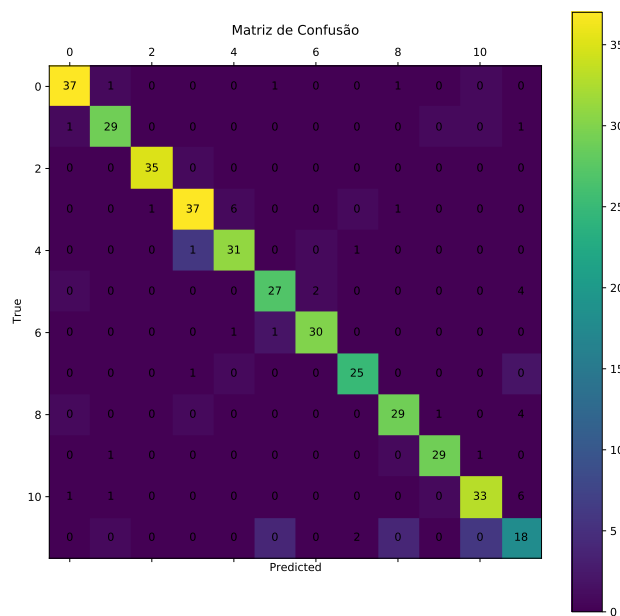


Figura 5: Matrix de confusão

5.3 LeNet-5 com dados aumentados

A mesma CNN LeNet-5, agora rodando com uma base maior de imagens, geradas pelo processo descrito na seção 4. A quantidade de imagens utilizadas no treinamento é de 37358. Os resultados podem ser vistos na tabela abaixo.

Experimento	Loss	Acurácia	Tempo de Execução (s)
LeNet-5 aug, batch_size=128, epochs=64, image=50x50, activation=relu	2.02	0.82	319.85
LeNet-5 aug, batch_size=128, epochs=13, image=50x50, activation=relu	0.47	0.85	74.565
LeNet-5 aug, batch_size=32, epochs=64, image=28x28, activation=relu	2.73	0.81	249.04
LeNet-5 aug, batch_size=128, epochs=64, image=32x32, activation=relu	0.52	0.83	48.62
LeNet-5 aug, batch_size=64, epochs=14, image=32x32, activation=relu	0.81	0.81	56.08
LeNet-5 aug, batch_size=32, epochs=14, image=32x32, activation=relu	0.76	0.81	64.367

Tabela 3: Resultados para LeNet-5

Os gráficos de acurácia e perda com o melhor desempenho.

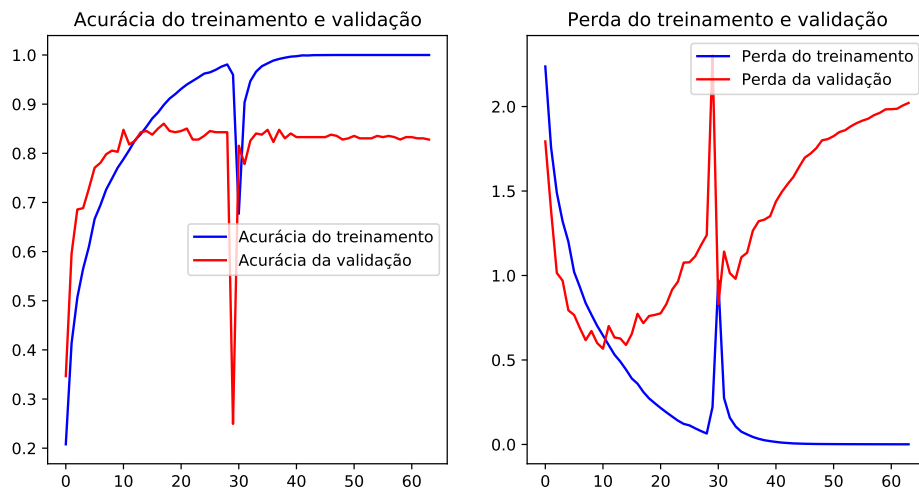


Figura 6: Acurácia e perda comparadas com treinamento e validação

O gráfico da matriz de confusão com o melhor desempenho.

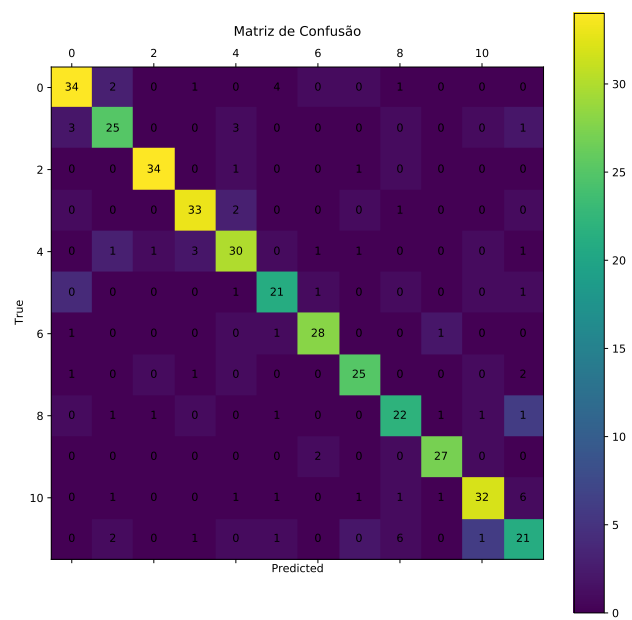


Figura 7: Matrix de confusão

5.4 Outra CNN com dados aumentados

Utilizando uma base de imagens com 37358 imagens, sendo 36000 imagens novas, geradas pelo processo descrito na seção 4.

Experimento	Loss	Acurácia	Tempo de Execução (s)
Outra CNN aumentada, batch_size=128, epochs=160, image=48x48	0.53	0.91	449.74
Outra cnn aumentada, batch_size=32, epochs=100, image=32x32	1.30	0.72	231.25
Outra cnn aumentada, batch_size=64, epochs=160, image=48x48	0.60	0.89	465.77

Tabela 4: Resultados para outra CNN

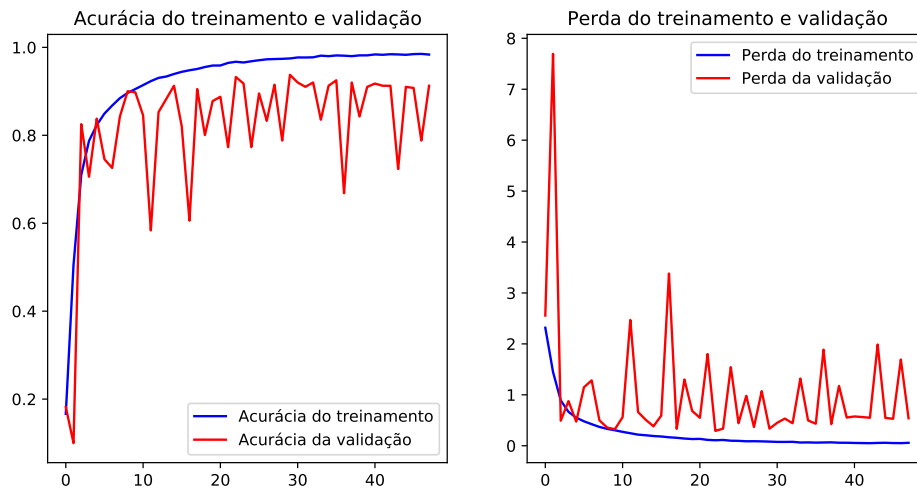


Figura 8: Acurácia e perda comparadas com treinamento e validação

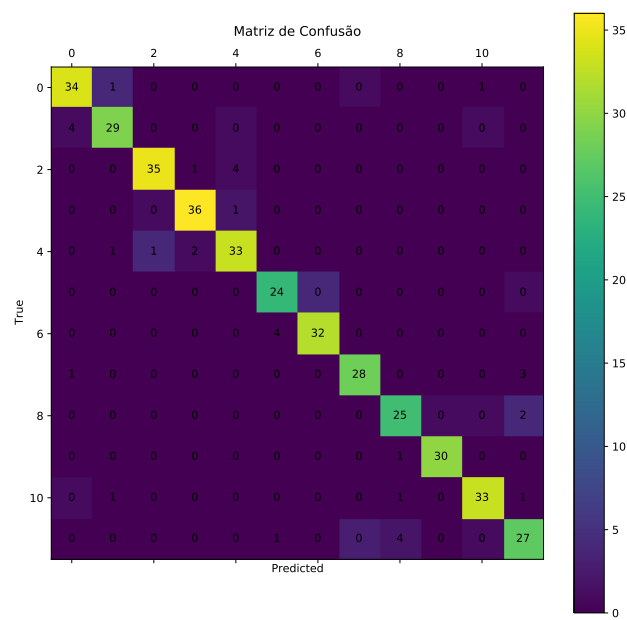


Figura 9: Matrix de confusão

6 Conclusão dos experimentos

A primeira conclusão é que os dados de treino não são suficientes para aumentar a acurácia da validação.

Os parâmetros que mais afetaram o desempenho das CNNs foram o tamanho do lote, `batch_size`, e o tamanho da imagem. Inicialmente a LeNet-5 contava com tamanho 28x28, mas normalmente ela é utilizada com tamanho 32x32, nos testes realizados o desempenho aumentou quando o recorte chegou ao tamanho 50x50.

No caso da outra CNN, o melhor recorte de imagem encontrado foi 48x48, com lotes de 128 para treinamento da rede. Como lotes grandes tem a tendência de overfitting, a estratégia estava em diminuir o tamanho das épocas.

Um problema comum ao modelo sendo treinado, é que dado o número longo de épocas, o modelo pode entrar em overfitting o que por sua vez aumenta a métrica loss da validação.

Com os dados gerados através da técnica de augmentação os modelos conseguiram aumentar sua acurácia, tanto para LeNet-5, quanto a Outra CNN.

Acredito que dois filtros poderiam melhorar os resultados. Um deles que pudesse deixar a imagem mais clara ou escura, e outro filtro que pudesse incluir ruídos diretamente nas linhas, apagando partes do traço, para tentar reproduzir uma caneta falhando ou os saltos da escrita das letras que podem variar de pessoa para pessoa.

O parâmetro que se mostrou mais prejudicial a sua alteração foi a função de ativação. Por isso os testes seguem a estratégia da rede original. Alguns testes mostraram que o simples fato de alterar a função de ativação nas camadas de aprendizado, causava um impacto tão negativo que muitas vezes o modelo não conseguia sair de 10% de acurácia.

7 Duas redes pré-treinadas na ImageNet

7.1 CNN Inception-v4

Para esse experimento, utilizou-se a CNN Inception v4 com os pesos de uma rede pré-treinada.

Para garantir que as camadas não sejam treinadas, mantendo assim os pesos originais foi utilizado o código:

```
1 for l in model.layers:
2     if l is not None: l.trainable = False
```

Listing 5: CNN escolhida

O melhor resultado com as 1578 imagens originais.

Experimento	Loss	Acurácia	Tempo de Execução (s)
Inception-v4, batch_size=128, epochs=160, image=48x48	0.82	0.07	17.69

Tabela 5: Resultados para Inception-v4

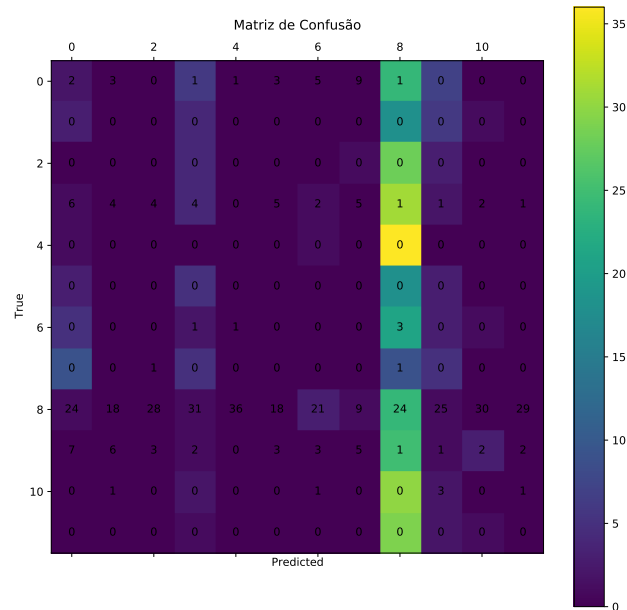


Figura 10: Matrix de confusão

7.2 CNN VGG16

Para esse experimento, utilizou-se a CNN VGG16 da ImageNet com os pesos de uma rede pré-treinada “imagenet”.

A CNN VGG16 carregada direto da biblioteca é um modelo funcional, portanto é necessário modificar sua entrada e sua saída para aceitar parâmetros diferentes. Por isso, um novo método para carregar a CNN foi criado, já que a original tem 1000 classes e estamos utilizando para 12 classes. Mesmo sem essa modificação, é possível fazer o predict das classes, mas ficará esparsa em uma matriz de 1000, tornando o método para gerar a matriz de confusão incompatível, já que os labels tem 12 níveis apenas.

```

1 def vgg16(img_rows, img_cols, num_classes):
2     #Get back the convolutional part of a VGG network trained on ImageNet
3     model_vgg16_conv = VGG16(weights='imagenet', include_top=False, input_shape=(
4         img_rows, img_cols, 3), classes=num_classes)
5
6     #Create your own input format
7     input = Input(shape=(img_rows, img_cols, 3), name = 'image_input')
8
9     #Use the generated model
10    output_vgg16_conv = model_vgg16_conv(input)
11
12    #Add the fully-connected layers
13    x = Flatten(name='flatten')(output_vgg16_conv)
14    x = Dense(num_classes, activation='softmax', name='predictions')(x)
15
16    #Create your own model
17    model = Model(inputs=input, outputs=x)
18
19    #In the summary, weights and layers from VGG part will be hidden, but they will
    be fit during the training

```

```

20 # my_model.summary()
21
22 # Say not to train first layer (ResNet) model. It is already trained
23 for l in model.layers:
24     if l is not None: l.trainable = False
25 return model

```

Listing 6: CNN escolhida

A matriz de confusão para o experimento utilizando os valores padrões.

Experimento	Loss	Acurácia	Tempo de Execução (s)
VGG16, batch_size=32, epochs=64, image=224x224	2.69	0.11	6.56

Tabela 6: Resultados para VGG16

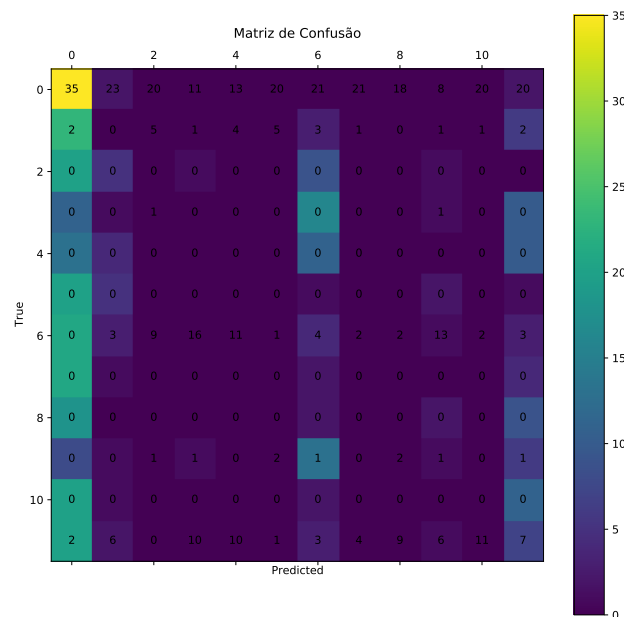


Figura 11: Matrix de confusão

8 Conclusão sobre as duas redes pré-treinadas da ImageNet

Como as duas redes foram treinadas para classificar 1000 itens, sua rede está calibrada para um propósito diferente daquela que estamos estudando. Mesmo assim, como explicado em sala de aula, a mesma pode ser utilizada para *feature extraction*.

Realizando os testes como instruído ocorreu um problema para trabalhar com a base aumentada, para o tamanho padrão das redes, a Inception v4, trabalha com

imagens de 299 e VGG16 com 224. O que acontece é que as imagens são carregadas em memória, para um número grande de testes a memória do Colab não é suficiente para rodar os testes com dados aumentados. Outra estratégia para lidar com o problema deve ser implementada. Uma possível solução seria utilizar o método generator do Keras, que faz a carga das imagens por Batch. Verificando a estrutura do train.txt e as mudanças necessárias para criar um sistema de arquivo compatível, chegou-se a conclusão que seria preciso criar uma nova ferramenta. Devido ao grande número de atividades para criar os testes já necessários, o mesmo não foi criado. Portanto, apenas testes sem dados aumentados foram apresentados. Sem os testes, mas já reparando no resultado ruim das redes pré-treinadas, supõe-se que o resultado não seja muito diferente.

9 Extração de características - *Feature Extraction*

Para realizar a extração de características faremos uso da CNN Inception v4. Mas para isso é necessário alterar o número de características de 12 para 1000, valor original para a ImageNet. O script utilizado pode ser visto na listagem abaixo.

```
1  ## Star Time
2  checkpoint_time = get_time()
3
4  INPUT_FILE_TEST = "./ml_lab3/test.txt"
5  OUTPUT_FILE_TEST = "./ml_lab3/svm/test.svm"
6  INPUT_FILE_TRAIN = TRAIN_FINAL
7  OUTPUT_FILE_TRAIN = "./ml_lab3/svm/train.svm"
8  IMG_ROWS = 100
9  IMG_COLS = 100
10 DIR_DATASET = "./ml_lab3/"
11
12 model = get_model()
13
14 # Train
15 extract_features(model, INPUT_FILE_TRAIN, OUTPUT_FILE_TRAIN, IMG_ROWS, IMG_COLS,
16                  DIR_DATASET)
17
18 # Test
19 extract_features(model, INPUT_FILE_TEST, OUTPUT_FILE_TEST, IMG_ROWS, IMG_COLS,
20                  DIR_DATASET)
21
22 ## Execution Time
23 print(f'Execution Time: {get_time_diff(checkpoint_time)}')
```

Listing 7: CNN escolhida

O arquivo de treino possui as imagens aumentadas e o tamanho da imagem foi alterada para 100x100, para evitar problemas de memória.

O processo tomou aproximadamente 67 segundos para cada lote de 1000 arquivos, resultando no total de 45 minutos para extrair as características de 37759 imagens. O arquivo final é um arquivo train.svm, qu possui um tamanho aproximado de 608MB.

Devido ao tamanho dos arquivos e a execução estar acontecendo em ambiente virtualizado, no Colab, decidiu-se apenas um lote de testes com a CNN Inception v4.

10 Classificador SVM

Nesta etapa utiliza-se o classificador SVM para realizar a classificação das imagens.

Para fazer as escolhas das melhores características utilizou-se o script abaixo. Neste script é utilizado o GridSearchCV da biblioteca do scikit learn, para executar uma validação cruzada dos parâmetros **C**, e **gamma**. Dois kernels são testados, o linear e o radial. O resultado é o melhor modelo testado contra a base de dados.

```
1 ## Star Time
2 checkpoint_time = get_time()
3
4 INPUT_TRAIN = "../ml_lab3/svm/train.svm"
5 INPUT_TEST = "../ml_lab3/svm/test.svm"
6
7 x_train, y_train = load_svmlight_file(INPUT_TRAIN)
8 x_test, y_test = load_svmlight_file(INPUT_TEST)
9
10 x_train = x_train.toarray()
11 x_test = x_test.toarray()
12
13 # Create the parameter grid based on the results of random search
14 params_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
15                  'C': [1, 10, 100, 1000]},
16                {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
17
18 # Performing CV to tune parameters for best SVM fit
19 svm_model = GridSearchCV(SVC(), params_grid, cv=5, verbose=1, n_jobs=-1)
20 svm_model.fit(x_train, y_train)
21 ## Execution Time
22 print(f'Execution Time: {get_time_diff(checkpoint_time)}')
```

Listing 8: CNN escolhida

Os melhores parâmetros encontrados foram $C = 1000$, $\gamma = 0.001$. A tabela abaixo apresenta os resultados de acurácia do melhor modelo.

Experimento	F1 Score	Acurácia	Tempo de Execução (s)
SVM kernel rbf $C = 1000$ $\gamma = 1e-3$	0	0.097	0.177

Tabela 7: Resultado para o SVM

O resultado da matriz de confusão, a seguir, mostra que todos os valores ficaram na classe 0. Isso indica que o modelo está errado, provavelmente relacionado ao número de dimensões utilizados para classificação. Mesmo com esse resultado, o tempo para avaliação é de aproximadamente 74 minutos.

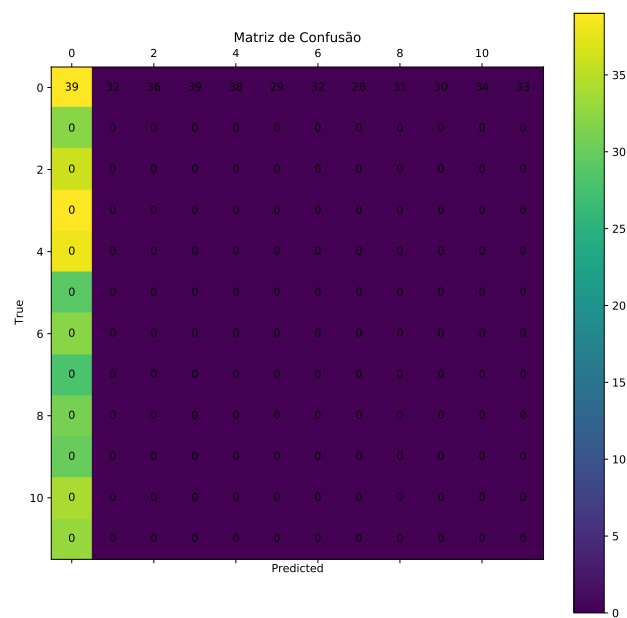


Figura 12: Matriz de confusão

11 Referências

Referências

- [1] LECUN, Y. et al. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1998. p. 2278–2324.
- [2] KANG, L. et al. Candidate fusion: Integrating language modelling into a sequence-to-sequence handwritten word recognition architecture. *arXiv preprint arXiv:1912.10308*, 2019.