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 podem ser encontrados no GitHub, https://github.com/marc-queiroz/ml_lab3/.

O trabalho está subdivido em seções, as primeiras apresentam as CNNs, LeNet-5 e a Outra CNN escolhida. Uma explicação dos métodos utilizados para augumentação de dados está presente em seção própria. Apresentadas as CNNs e as técnicas de augumentação utilizas, segue-se para os experimentos, neles são apresentadas os testes com e sem augumentação para as duas redes. Em seção própria, é apresentado os resultados dos experimentos. Também foi escolhido duas redes pré-treinadas da ImageNet, Inception v4 e VGG16 para realizar os experimentos de (Transfer Learning/Fine-Tuning). Chegando a parte final, utiliza-se a rede pré-treinada Inception v4 para extração de características. Na última sessão é apresentado os experimentos com o classificador SVM, utilizando os dados extraídos com a técnica *feature extraction*.

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.

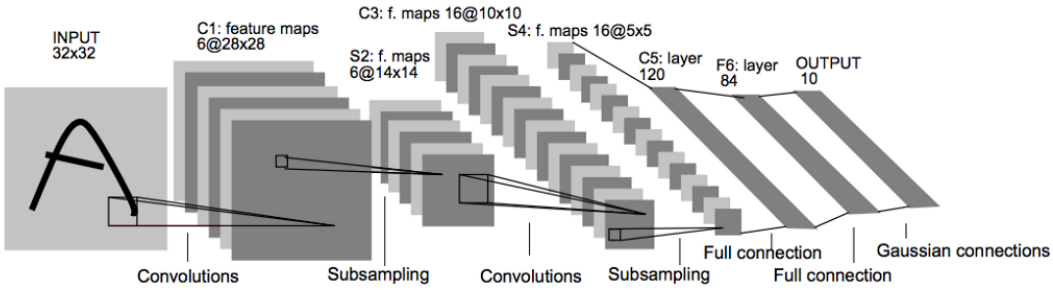

Figura 1: As 7 camadas do modelo LeNet-5[1]

```
model = keras.Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu',
    input_shape=(32,32,1)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='relu'))
model.add(layers.Dense(units=84, activation='relu'))
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.

```
dropout_dense_layer = 0.6

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(dropout_dense_layer))

model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(dropout_dense_layer))

model.add(Dense(12))
model.add(Activation('sigmoid'))
```

Listing 2: Modelo da Outra CNN

A rede é composta por 15 camadas, onde a camada de saída (OUTPUT) foi modificada para 12 classes, incluindo duas camadas de Dropout.

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, iniciou-se um teste com o exemplo de geração de imagens demostradas em sala de aula, utilizando o próprio gerador do Keras, de acordo com o código fornecido.

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 na Internet sobre o assunto, *data augmentation for handwritting*, 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 foram criadas. Uma delas transfere 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. Com o novo formato de arquivo, aplica-se o script de augumentação para gerar novas imagens. As novas imagens são salvas em novo diretório. Como os novos arquivos possuem o label em seu nome de arquivo, foi necessário criar outra ferramenta para popular o novo arquivo de entrada, que será utilizado para carregar os novos dados gerados. O arquivo pode ser encontrado no repositório do relatório, com o nome train_aug.txt, incluindo as ferramentas aqui mencionadas.

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.

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

Listing 3: Código utilizado para augumentar as imagens originais

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 augumentados, foi incluída a distorção gaussiana, com uma probabilide de 30% de ativação do filtro para cada imagem sendo gerada.

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

Listing 4: Segunda versão do código também aplicado as imagens originais

No total foram geradas 35780 imagens.

5 Experimentos

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

5.1 LeNet-5

Para encontrar o melhor conjunto de parâmetros para a CNN LeNet-5 com dados originais, procurou-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 matriz de 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

Os gráficos de acurácia e perda com o melhor desempenho.

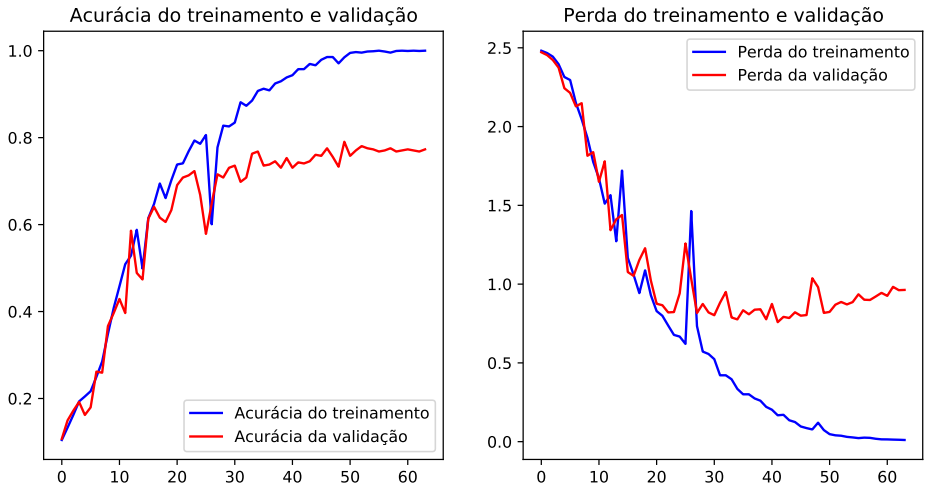

Figura 2: Acurácia e perda comparadas com treinamento e validação

O gráfico da matriz de confusão com o melhor desempenho.

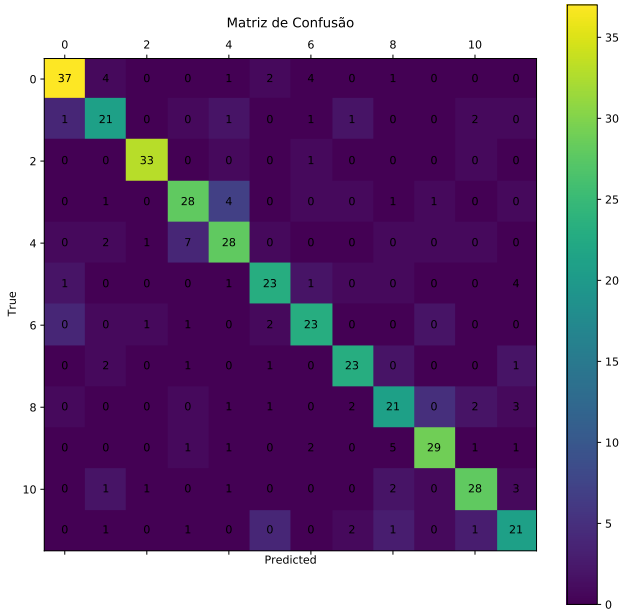

Figura 3: Matriz de confusão

5.2 Outra CNN

Será apresentado uma tabela com os valores encontrados para alguns dos experimentos, incluindo a apresentação dos gráficos e matriz 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

Os gráficos de acurácia e perda com o melhor desempenho.

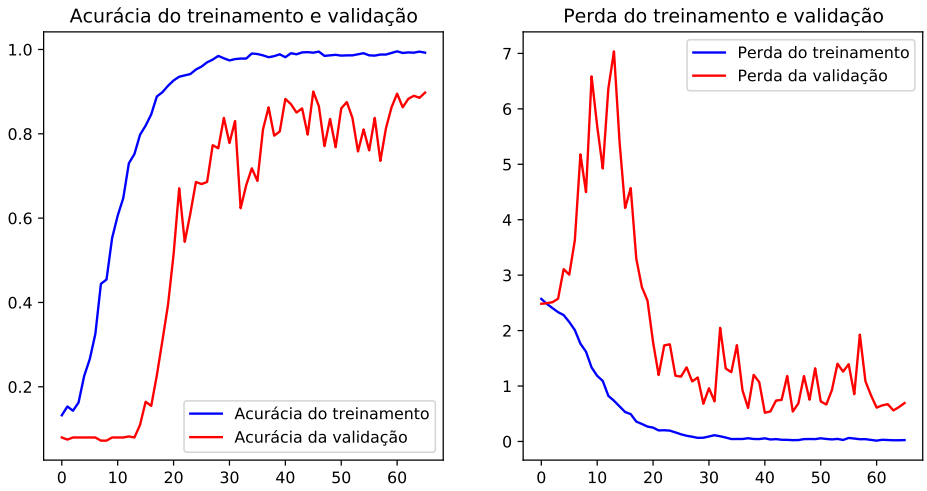

Figura 4: Acurácia e perda comparadas com treinamento e validação

O gráfico da matriz de confusão com o melhor desempenho.

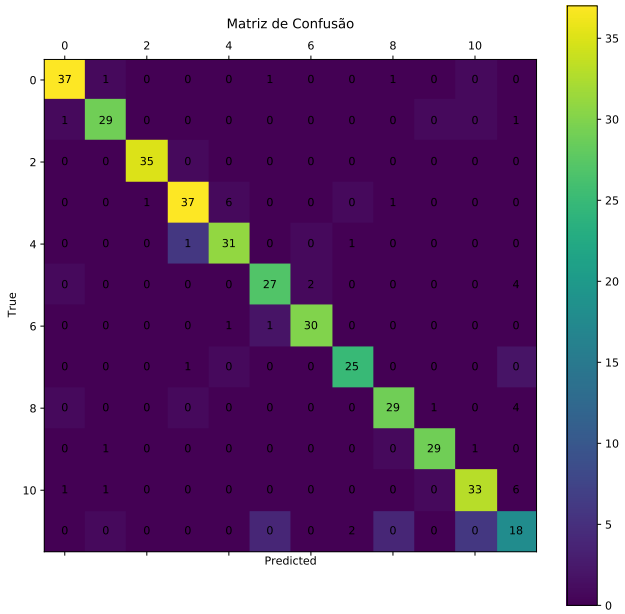

Figura 5: Matriz de confusão

Essa 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.

5.3 LeNet-5 com dados augumentados

A mesma CNN LeNet-5, com uma base de treinos maior, 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 com dados augumentados

Os gráficos de acurácia e perda com o melhor desempenho.

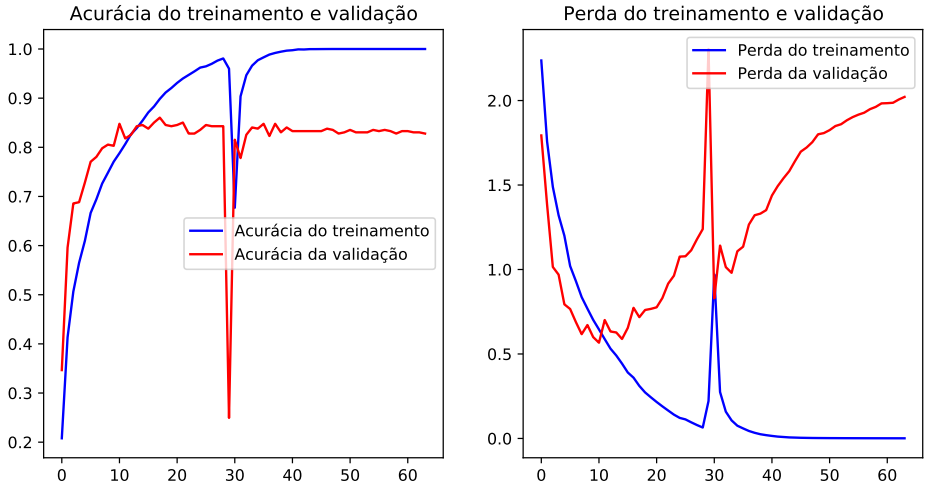

Figura 6: Acurácia e perda comparadas com treinamento e validação

O gráfico da matriz de confusão com o melhor desempenho.

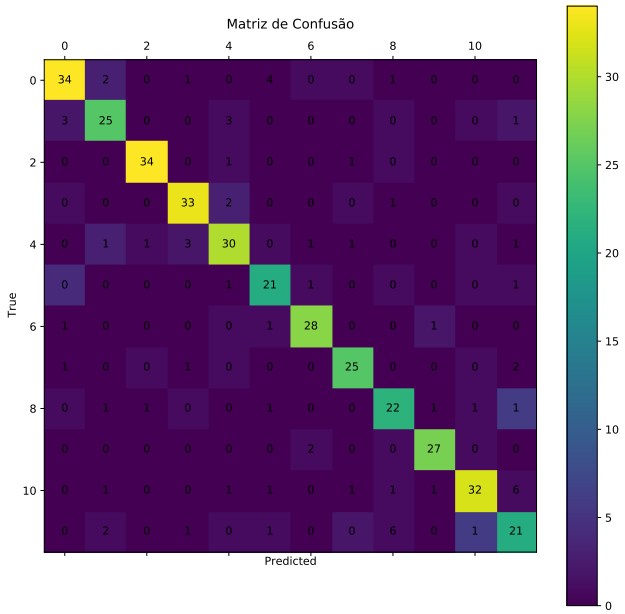

Figura 7: Matriz de confusão

5.4 Outra CNN com dados augumentados

Utilizando uma base de imagens com 37358 imagens, sendo 36000 imagens novas, geradas pelo processo descrito na seção 4. Os resultados podem ser vistos na tabela abaixo.

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

Tabela 4: Resultados para outra CNN com dados augumentados

Os gráficos de acurácia e perda com o melhor desempenho.

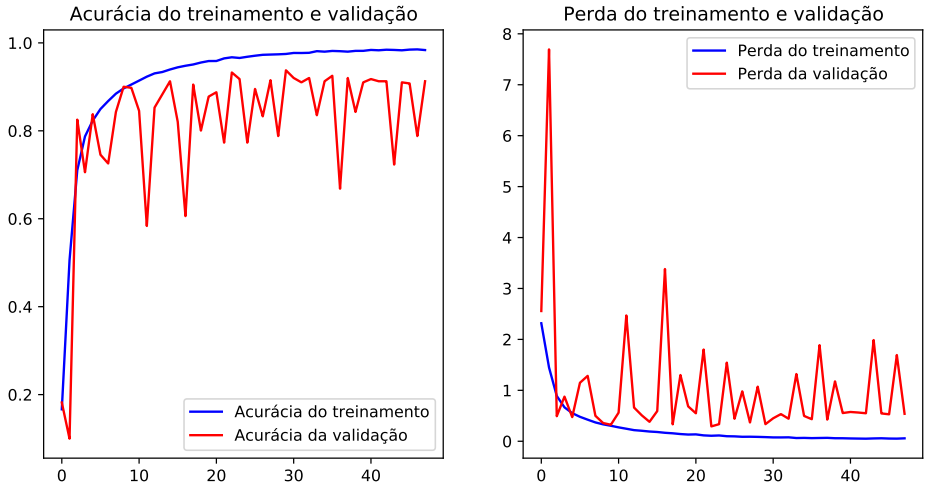

Figura 8: Acurácia e perda comparadas com treinamento e validação

O gráfico da matriz de confusão com o melhor desempenho.

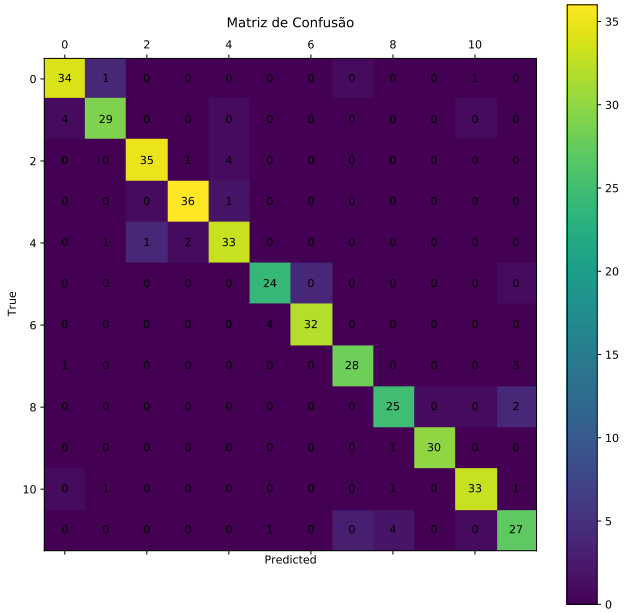

Figura 9: Matriz de confusão

6 Conclusão dos experimentos

A primeira conclusão é que inicialmente os dados de treino não eram 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. Originalmente a LeNet-5 trabalhava com tamanho da imagem 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 Ooutra 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 um número elevado de épocas, o modelo pode entrar em overfitting o que causa perda (loss) na validação.

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

A adição de outros 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

As CNNs escolhidas para essa etapa são a Inception-v4 e a VGG16.

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 abaixo.

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

Listing 5: Código utilizado para garantir que as camadas da CNN não sejam treinadas

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

O gráfico da matriz de confusão com o melhor desempenho para Inception-v4.

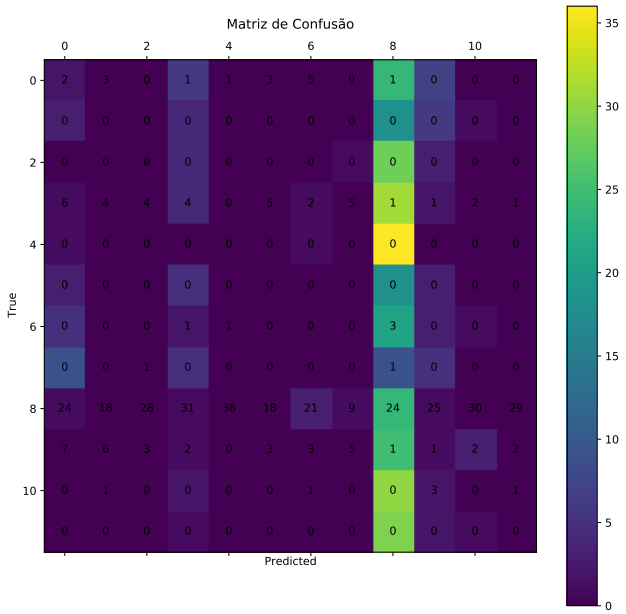

Figura 10: Matriz de confusão

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. O código abaixo apresenta o método utilizado para gerar o modelo pronto para ser compilado.

```
1
2 def vgg16(img_rows, img_cols, num_classes):
3   #Get back the convolutional part of a VGG network trained on ImageNet
4   model_vgg16_conv = VGG16(weights='imagenet', include_top=False, input_shape=(
      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 no to train first layer 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: Função para carregar o modelo da VGG16

Os resultados para Loss e Acurácia e Tempo de Execução podem ser vistos na tabela abaixo.

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

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

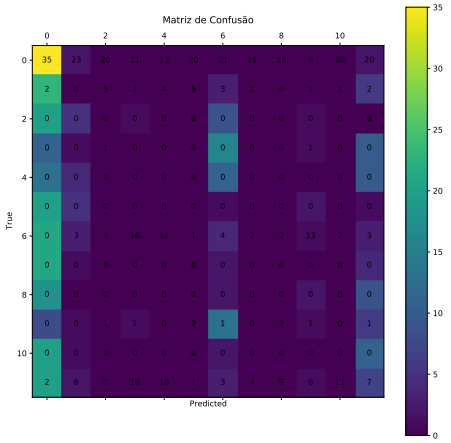

Figura 11: Matriz de confusão

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 utilizar a base augmentada, para o tamanho padrão das imagens das redes. A Inception v4, trabalha com imagens de 299x299 e VGG16 com 224x224. A técnica para carregamento das imagens, faz alocação em memória, quando utilizadas com um recorte de 299x299, por exemplo, para um número grande de testes. Chega-se a conclusão que a memória do Colab não é suficiente para rodar os testes com dados augumentados. 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 augumentados foram apresentados.

9 Extração de características - *Feature Extraction*

Para realizar a extração de caracteristicas faz-se 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.

```
## Star Time
checkpoint_time = get_time()

INPUT_FILE_TEST = "./ml_lab3/test.txt"
OUTPUT_FILE_TEST = "./ml_lab3/svm/test.svm"
INPUT_FILE_TRAIN = TRAIN_FINAL
OUTPUT_FILE_TRAIN = "./ml_lab3/svm/train.svm"
IMG_ROWS = 100
IMG_COLS = 100
DIR_DATASET = "./ml_lab3/"

model = get_model()

# Train
extract_features(model, INPUT_FILE_TRAIN, OUTPUT_FILE_TRAIN, IMG_ROWS, IMG_COLS,
    DIR_DATASET)

# Test
extract_features(model, INPUT_FILE_TEST, OUTPUT_FILE_TEST, IMG_ROWS, IMG_COLS,
    DIR_DATASET)

## Execution Time
print(f'Execution Time: {get_time_diff(checkpoint_time)}')

print('DONE')
```

Listing 7: Código para extração de características

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

O processo tomou aproximamente 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, que possui um tamanho aproximado de 608MB (não compactados).

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 código da listagem abaixo. 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.

```
## Star Time
checkpoint_time = get_time()

INPUT_TRAIN = "../ml_lab3/svm/train.svm"
INPUT_TEST = "../ml_lab3/svm/test.svm"

x_train, y_train = load_svmlight_file(INPUT_TRAIN)
x_test, y_test = load_svmlight_file(INPUT_TEST)

x_train = x_train.toarray()
x_test = x_test.toarray()

# Create the parameter grid based on the results of random search
params_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                'C': [1, 10, 100, 1000]},
               {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

# Performing CV to tune parameters for best SVM fit
svm_model = GridSearchCV(SVC(), params_grid, cv=5, verbose=1, n_jobs=-1)
svm_model.fit(x_train, y_train)
## Execution Time
print(f'Execution Time: {get_time_diff(checkpoint_time)}')
```

Listing 8: Código utilizado para encontrar os melhores parâmetros

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 preditos 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 foi de aproximadamente 74 minutos.

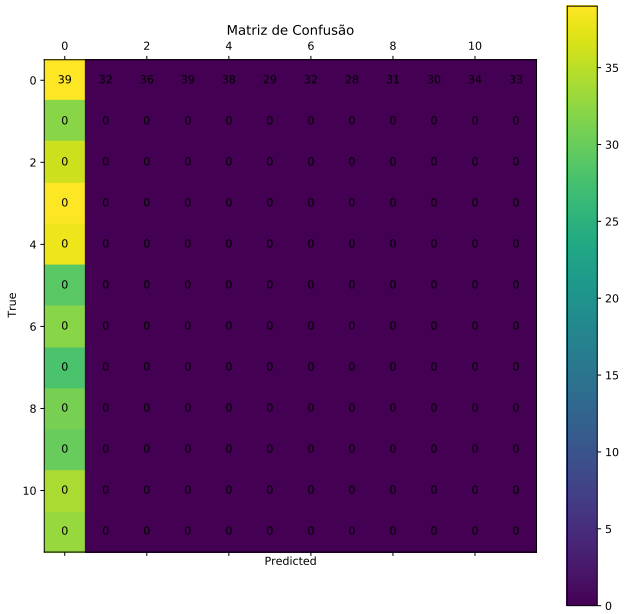

Figura 12: Matriz de confusão

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.