Instituto Federal de Educação, Ciência e Tecnologia - Câmpus Campinas

Pós-graduação em Ciência de Dados

APLICAÇÕES EM CIÊNCIA DE DADOS

Atividade 2 - Face Recognition

Alunos:

Alexandre Freire da Silva Osorio

Weld Lucas Cunha

Problema

Implementar uma rede neural para resolver o problema de identificação de faces usando o dataset PubFig83.

Solução

Redes Neurais Adotadas

Primeiramente foi testada uma rede própria, chamada de ***baseline***, composta pelas seguintes camadas:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 100, 100, 32)	896
activation (Activation)	(None, 100, 100, 32)	0
conv2d_1 (Conv2D)	(None, 98, 98, 32)	9248
activation_1 (Activation)	(None, 98, 98, 32)	0
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
dropout (Dropout)	(None, 49, 49, 32)	0
conv2d_2 (Conv2D)	(None, 49, 49, 64)	18496
activation_2 (Activation)	(None, 49, 49, 64)	0
conv2d_3 (Conv2D)	(None, 49, 49, 64)	36928
activation_3 (Activation)	(None, 49, 49, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
dropout_1 (Dropout)	(None, 24, 24, 64)	0
conv2d_4 (Conv2D)	(None, 24, 24, 128)	73856
activation_4 (Activation)	(None, 24, 24, 128)	0
conv2d_5 (Conv2D)	(None, 24, 24, 128)	147584
activation_5 (Activation)	(None, 24, 24, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
conv2d_6 (Conv2D)	(None, 12, 12, 256)	295168
activation_6 (Activation)	(None, 12, 12, 256)	0
conv2d_7 (Conv2D)	(None, 12, 12, 256)	590080
activation_7 (Activation)	(None, 12, 12, 256)	0
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 256)	0

dropout_3 (Dropout)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 1024)	9438208
dropout_4 (Dropout)	(None, 1024)	0
activation_8 (Activation)	(None, 1024)	0
dense_1 (Dense)	(None, 256)	262400
dropout_5 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 83)	21331

Total params: 10,894,195
Trainable params: 10,894,195
Non-trainable params: 0

Os outros dois experimentos foram realizados usando *transfer learning*, utilizando arquiteturas pré-treinadas no dataset *Imagenet*, com camadas adicionais. As configurações de cada experimento estão descritas a seguir.

Modelo 1

efficientnetv2-b0 + Dense(256) + Dropout + Dense(n_classes, activation='softmax')

Layer (type)	Output Shape	Param #
keras_layer (KerasLayer)	(None, 1280)	5919312
dense (Dense)	(None, 256)	327936
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 83)	21331

Total params: 6,268,579
Trainable params: 349,267
Non-trainable params: 5,919,312

Modelo 2

VGG 16 + GlobalAveragePooling2D + Dense(1024, activation='relu')) + Dropout + Dense(256, activation='relu') + Dropout + Dense(n_classes, activation='softmax', kernel_regularizer=tf.keras.regularizers.l2(0.0001))

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
global_average_pooling2d (Gl	(None, 512)	0
dense (Dense)	(None, 1024)	525312
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 256)	262400
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 83)	21331

Total params: 15,523,731
Trainable params: 809,043
Non-trainable params: 14,714,688

Pré-processamento

Para os dados de treino, foi realizada uma normalização, dividindo o valor de cada pixel pela constante 255, seguido de *data augmentation*, com os seguintes parâmetros:

- rotation_range=30,
- width_shift_range=0.2,
- height_shift_range=0.2,
- zoom_range=0.2,
- horizontal_flip=True,
- vertical_flip=True,
- fill_mode='nearest'

Para os dados de teste, foi realizada a normalização pelo fator 255. O pré-processamento de ambos conjuntos foi realizado utilizando-se a classe *ImageDataGenerator* do pacote de pré-processamento do Keras.

Treinamento

Para o **modelo *baseline***, foram treinadas 100 épocas, com *batch size* de 128 e *learning rate* de 10e-4. Foi utilizada uma *callback* de *checkpoint* e *early stopping*, tendo como monitor a acurácia de validação. O otimizador foi o Adam e a função de *loss* escolhida foi a *Categorical Crossentropy*.

Os resultados do treino e da validação estão ilustrados no gráfico abaixo.

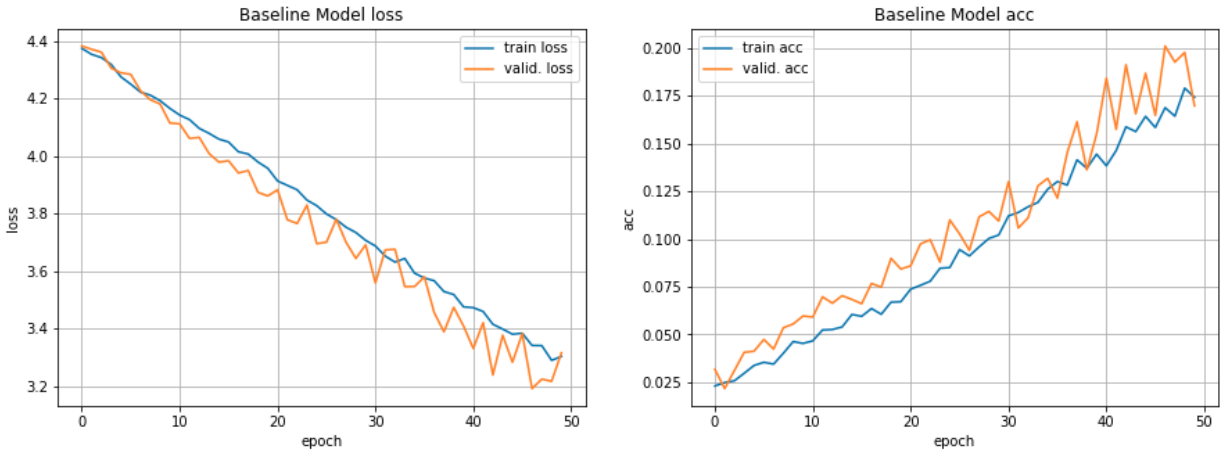

Fig. 1: Loss e acurácia de treino e validação para o modelo baseline

Da mesma forma que o modelo baseline, para o **Modelo 1** também foram treinadas 100 épocas, com *batch size* de 128 e *learning rate* de 10e-4. Foi utilizada uma *callback* de *checkpoint* e *early stopping*, tendo como monitor a acurácia de validação. O otimizador foi o Adam e a função de *loss* escolhida foi a *Categorical Crossentropy*.

Os resultados do treino e da validação estão ilustrados no gráfico abaixo. A acurácia média de teste ficou em 0,35.

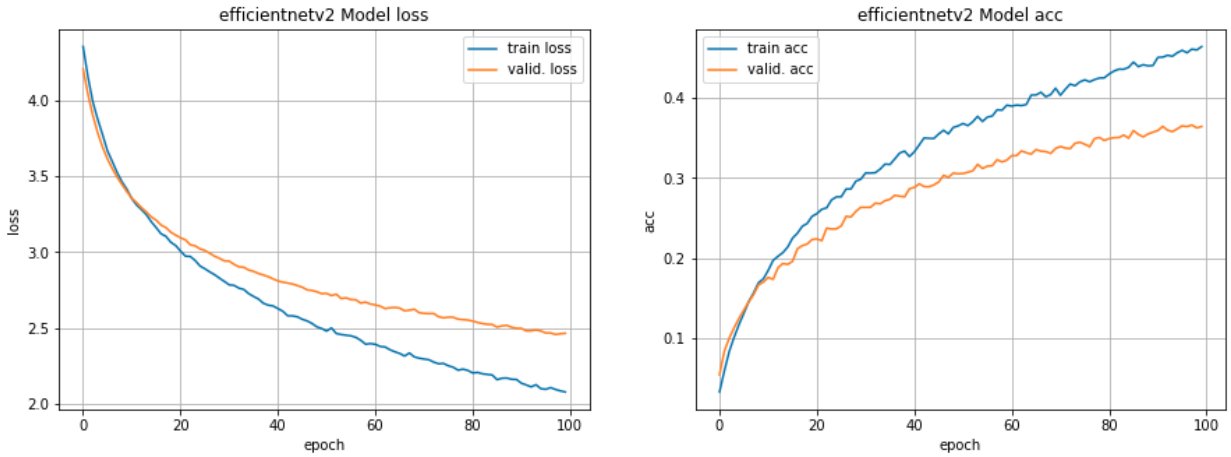

Fig. 2: Loss e acurácia de treino e validação para o Modelo 1 (VGG16 + camadas adicionais)

Para o **Modelo 2**, foi utilizada uma função de escalonamento do *learning rate*, de forma a seguir o perfil ao longo das 200 épocas desenhado no gráfico a seguir. Os demais parâmetros de treino permaneceram iguais aos dos modelos anteriores.

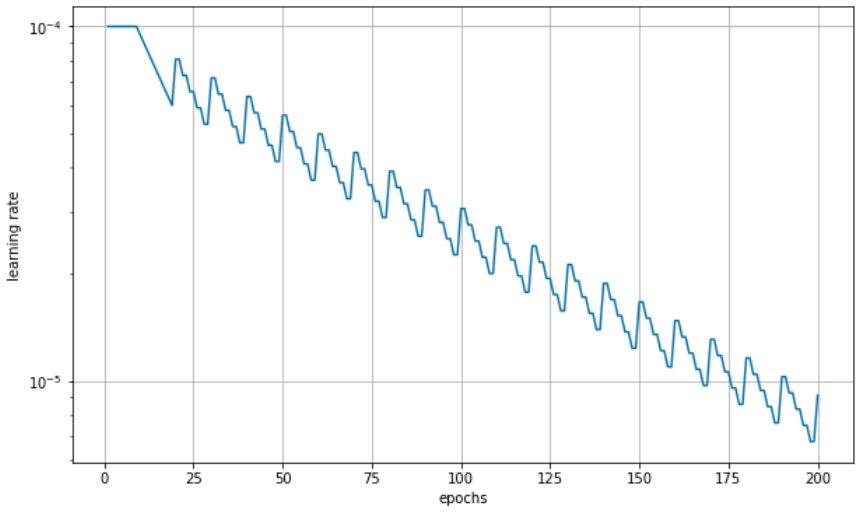

Fig. 3: Função de escalonamento do learning rate para o Modelo 2 (VGG16 + camadas adicionais)

Os resultados do treino e da validação estão ilustrados abaixo.

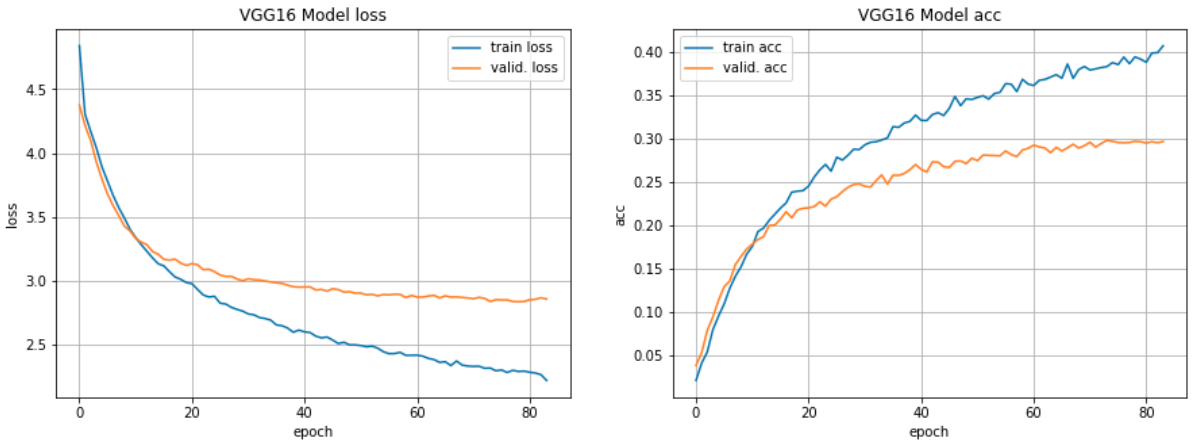

Fig. 4: Loss e acurácia de treino e validação para o Modelo 2

Posteriormente, foi incluído um teste adicional, treinando a mesma rede e usando agora um ***fine-tuning*** em que as 3 últimas camadas da rede VGG16 foram retreinadas (os pesos de todas as outras camadas se mantiveram congelados). Para esse último experimento, a função do escalonador do *learning rate* assumiu o perfil do gráfico a seguir:

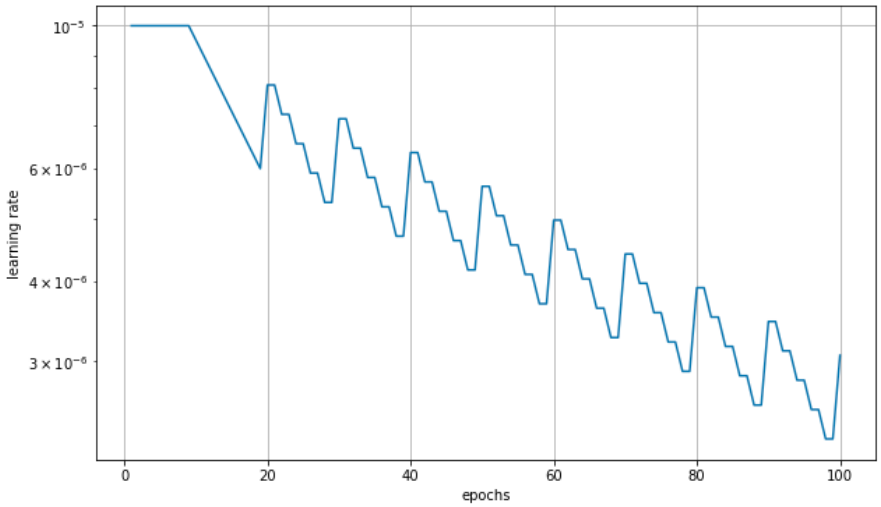

Fig. 5: Função de escalonamento do learning rate para o Modelo 2, com fine-tuning

Os resultados do treino e da validação estão ilustrados no gráfico abaixo. A acurácia média de teste ficou em 0,51.

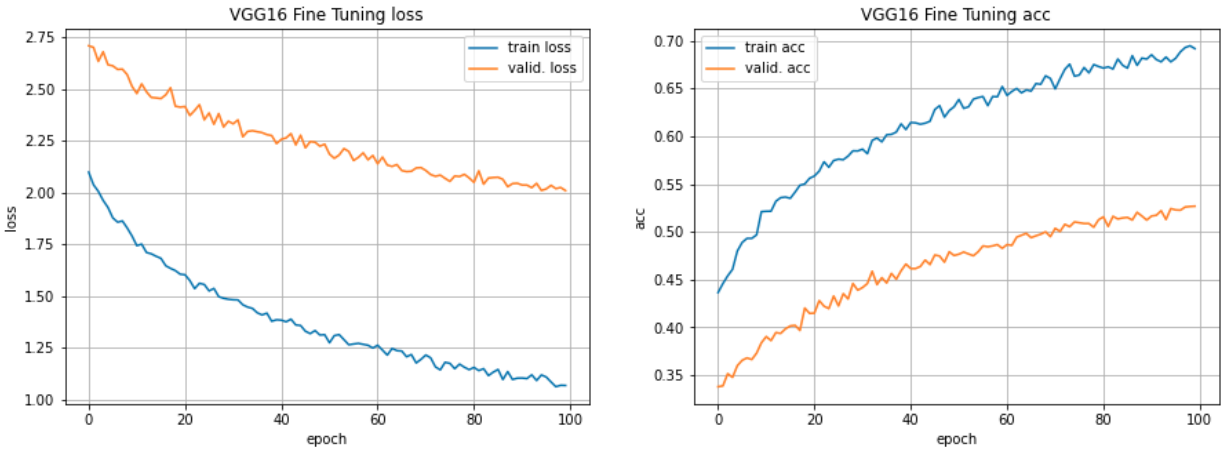

Fig. 6: Loss e acurácia de treino e validação para o Modelo 2 com fine tuning

Discussão dos resultados

A tabela a seguir dá um resumo dos resultados dos experimentos:

Modelo	Acc. Validação	Acc. Teste
Baseline	0.2	0.19
Modelo 1	0.37	0.35
Modelo 2	0.53	0.51

Pudemos verificar que a introdução do *transfer learning* foi o fator que propiciou uma melhoria substantiva do desempenho (Modelo 1). Outro fator de melhoria do desempenho foi o uso de *fine tuning*, pelo qual as 3 últimas camadas da VGG16 foram retreinadas com os dados do *dataset*.

O ImageDataGenerator se mostrou um recurso bastante prático para realizar pré-processamento e normalização de dados de imagens. Da mesma forma, o Tensorflow Hub se mostrou uma ferramenta bastante útil para escolher entre um conjunto grande de modelos pré-treinados, possuindo um conjunto interessante de utilitários.

Trabalhos futuros

Como melhoria do trabalho, podemos pensar e repetir um dos experimentos com *transfer learning*, reconfigurando o *fine tuning* para retreinar 100% dos coeficientes do modelo pré-treinado. Essa suposição tem base num experimento que realizamos usando a mesma rede Efficientnet do Modelo 1 com 100% dos seus pesos retreinados. Foi usado o mesmo dataset PubFig83, com a diferença de que o conjunto de treino e validação foi definido automaticamente usando a classe *preprocessing.image_dataset_from_directory* do Keras.

Devido ao aumento expressivo do tempo requerido para treino (cerca de 775 segundos por época), não conseguimos treinar todas as 100 épocas planejadas. No entanto, pelo resultado da última época executada no treinamento, podemos inferir que haverá um ganho expressivo de performance do modelo:

Epoch 42/100

691/691 [==============================] - 775s 1s/step - loss: 0.9050 - accuracy: 0.9912 - val_loss: 1.0878 - val_accuracy: 0.9262