Model S with data augmentation

Arquitetura

A arquitetura deste modelo é baseada no modelo sem data augmentation que tem a seguinte arquitetura e a seguinte data Augmentation:

```python
# Definindo o input
inputs = Input(shape=(IMG_SIZE, IMG_SIZE, 3))

# Aplicando Data Augmentation
x = data_augmentation(inputs)

# Primeira camada convolucional
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.0001))(inputs)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.3)(x)

# Segunda camada convolucional
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.0001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.3)(x)

# Terceira camada convolucional
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.0001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.3)(x)

# Quarta camada convolucional
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', kernel_regularizer=l2(0.0001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Dropout(0.4)(x)

# Camada de Flatten
x = layers.Flatten()(x)

# Camada totalmente conectada
x = layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001))(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
# Camada de saída
outputs = layers.Dense(10, activation='softmax')(x)  # Supondo 10 classes
```

Figura 1 - Arquitetura

```python
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.2),
        layers.RandomZoom(0.2),
        layers.RandomContrast(0.2),
    ]
)
```

Figura 2 - Augmentation Utilizada

Neste modelo, usamos exatamente a mesma estrutura de rede sem data augmentation. Apenas aumentámos o número de neurónios na quarta camada para o dobro. Fizemos isso porque, com data augmentation, existem mais variações na apresentação das fotos, o que requer mais capacidade de processamento para alcançar uma melhor precisão.

Inicialmente, começámos por executar a mesma rede, mantendo tudo igual com as 40 mil imagens, e apenas acrescentando a data augmentation. Com estas alterações, obtivemos uma accuracy de 83%.

```
313/313 [==============================] - 10s 27ms/step - loss: 0.7107 - accuracy: 0.8329
val_acc: 0.8328999876976013
313/313 [==============================] - 10s 31ms/step - loss: 0.7320 - accuracy: 0.8267
Test accuracy: 0.82669997215271
```

Figura 3 - Accuracy Sem mexer na rede

Treinos

Visto isto, realizámos outros testes para tentar melhorar a accuracy, porque com data augmentation poderíamos obter melhores resultados. Começámos por tentar retirar os DropOuts, mas sem sucesso, pois o desempenho ficou pior do que já estava. Assim, considerando que tínhamos mais características na rede, decidimos aumentar o número de neurónios nas últimas camadas para extrair mais características.

Fizemos vários testes com estas alterações e verificámos que, com mais alguns neurónios, a nossa rede apresentava melhores resultados. No entanto, não foi possível treinar com 1024 neurónios na última camada, pois isso tornava o treino demasiado pesado.

Além disso, mantivemos os *Regularizers* e o *BatchNormalization* para ajudar a rede na aprendizagem. Estes elementos são importantes para prevenir overfitting e melhorar a estabilidade e eficiência do treino.

No final, optámos por acrescentar apenas numa camada, pois esta solução se revelou um meio-termo ideal entre as questões de treino e o overfitting, como podemos ver nas imagens.

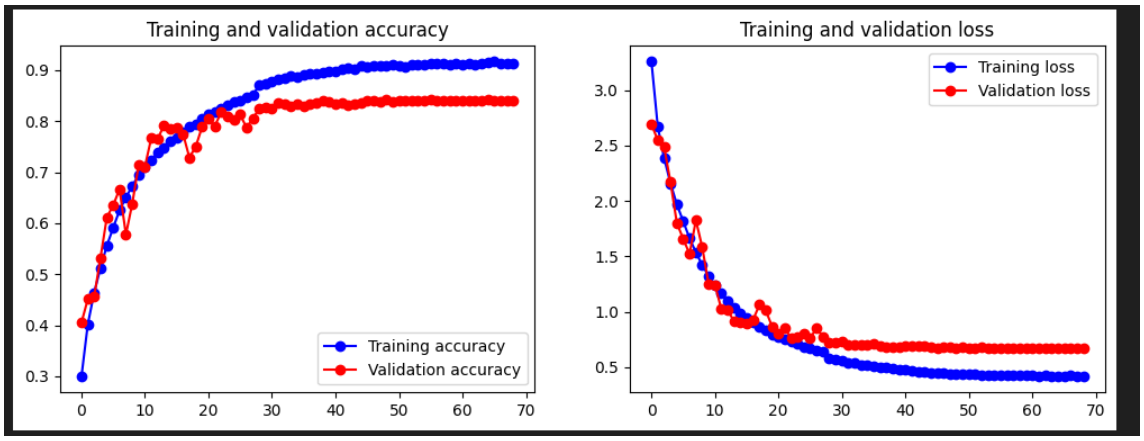

Figura 4 - Graficos do Treino

Tendo isto, ficámos com uma rede que, com data augmentation, alcançou uma accuracy de 85%, um valor superior aos 83% obtidos sem data augmentation, mas ainda abaixo do esperado. Embora tenhamos conseguido melhorar a precisão, os resultados não foram tão elevados quanto esperavamos.

```
313/313 [==============================] - 6s 20ms/step - loss: 0.6980 - accuracy: 0.8531
val_acc: 0.8531000018119812
313/313 [==============================] - 6s 20ms/step - loss: 0.7364 - accuracy: 0.8455
Test accuracy: 0.8454999923706055
```

Figura 5 - Accuracy máxima alcançada

```
Epoch 1/100
1250/1250 [==============================] - 91s 72ms/step - loss: 3.2582 - accuracy: 0.2993 - val_loss: 2.6926 - val_accuracy: 0.4049 - lr: 1.0000e-04
Epoch 2/100
1250/1250 [==============================] - 91s 72ms/step - loss: 2.6692 - accuracy: 0.4018 - val_loss: 2.5478 - val_accuracy: 0.4512 - lr: 1.0000e-04
Epoch 3/100
1250/1250 [==============================] - 87s 70ms/step - loss: 2.3859 - accuracy: 0.4638 - val_loss: 2.4864 - val_accuracy: 0.4559 - lr: 1.0000e-04
Epoch 4/100
1250/1250 [==============================] - 86s 69ms/step - loss: 2.1586 - accuracy: 0.5124 - val_loss: 2.1695 - val_accuracy: 0.5325 - lr: 1.0000e-04
Epoch 5/100
1250/1250 [==============================] - 86s 69ms/step - loss: 1.9696 - accuracy: 0.5560 - val_loss: 1.8008 - val_accuracy: 0.6115 - lr: 1.0000e-04
Epoch 6/100
1250/1250 [==============================] - 86s 69ms/step - loss: 1.8154 - accuracy: 0.5911 - val_loss: 1.6581 - val_accuracy: 0.6351 - lr: 1.0000e-04
Epoch 7/100
1250/1250 [==============================] - 86s 69ms/step - loss: 1.6646 - accuracy: 0.6255 - val_loss: 1.5218 - val_accuracy: 0.6669 - lr: 1.0000e-04
Epoch 8/100
1250/1250 [==============================] - 86s 69ms/step - loss: 1.5346 - accuracy: 0.6498 - val_loss: 1.8255 - val_accuracy: 0.5776 - lr: 1.0000e-04
Epoch 9/100
1250/1250 [==============================] - 86s 69ms/step - loss: 1.4222 - accuracy: 0.6735 - val_loss: 1.5818 - val_accuracy: 0.6381 - lr: 1.0000e-04
Epoch 10/100
1250/1250 [==============================] - 86s 69ms/step - loss: 1.3210 - accuracy: 0.6936 - val_loss: 1.2506 - val_accuracy: 0.7137 - lr: 1.0000e-04
Epoch 11/100
1250/1250 [==============================] - 86s 69ms/step - loss: 1.2356 - accuracy: 0.7098 - val_loss: 1.2415 - val_accuracy: 0.7111 - lr: 1.0000e-04
Epoch 12/100
1250/1250 [==============================] - 86s 68ms/step - loss: 1.1684 - accuracy: 0.7227 - val_loss: 1.0265 - val_accuracy: 0.7672 - lr: 1.0000e-04
Epoch 13/100
...
Epoch 68/100
1250/1250 [==============================] - 87s 70ms/step - loss: 0.4181 - accuracy: 0.9134 - val_loss: 0.6701 - val_accuracy: 0.8400 - lr: 1.0000e-06
Epoch 69/100
1250/1250 [==============================] - 87s 70ms/step - loss: 0.4177 - accuracy: 0.9137 - val_loss: 0.6711 - val_accuracy: 0.8410 - lr: 1.0000e-06
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figura 6 - Treino executado