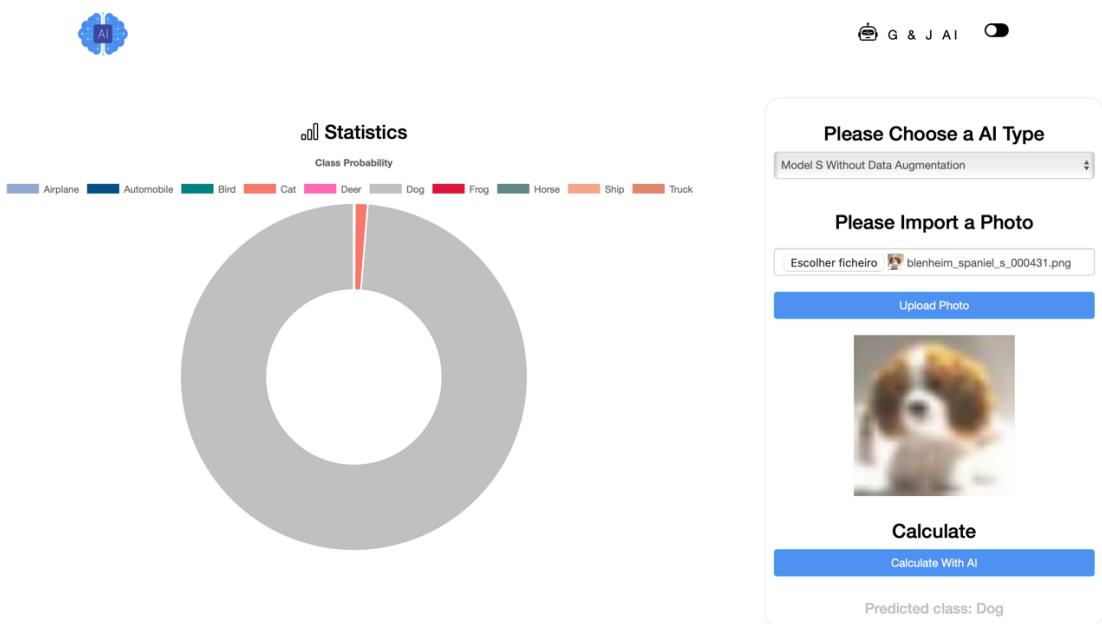


ModelT_transferLearning_featureExtraction_WithDataAugmentation Results

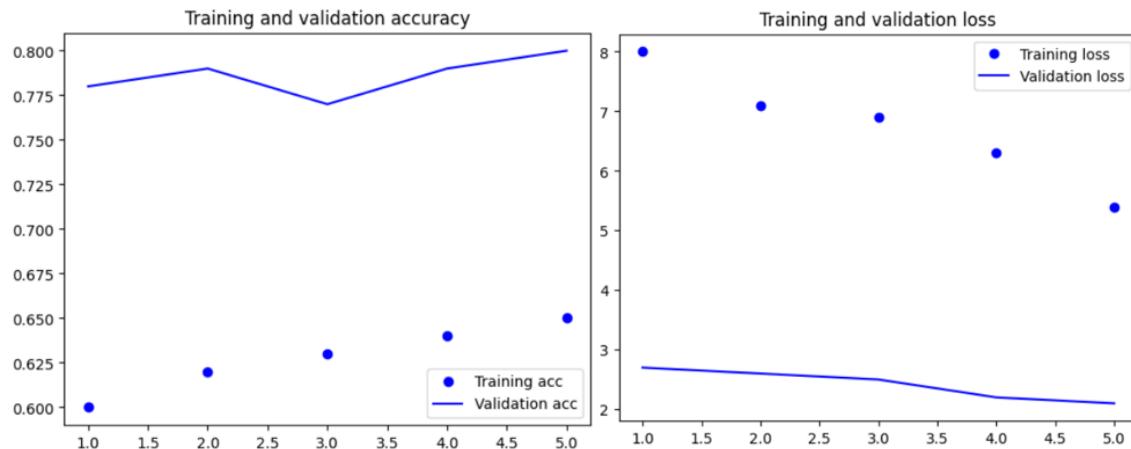


Powered By:

Gonçalo Ferreira, nº 2222051
José Delgado, nº 2222049

Results

Batch_Size 64, 5 épocas Optimizer ADAM Sem DropOut Com Data Augmentation Learning_Rate 1e-4



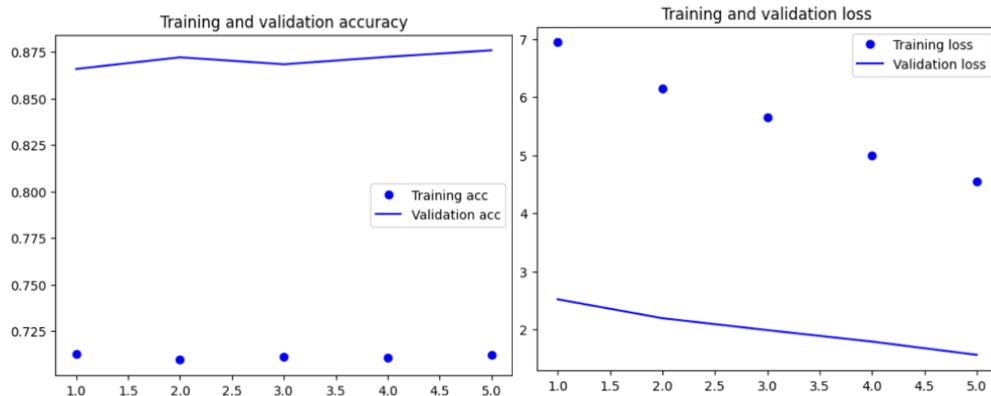
```
from tensorflow import keras
from keras import layers

data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
        layers.RandomContrast(0.2),
        layers.RandomBrightness(0.2),
        layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
    ]
)

inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg19.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(512)(x)
x = layers.Dense(256)(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs, outputs)
```

Com esta primeira versão da rede, conseguimos uma accuracy de sensivelmente 0.81 e uma loss de 2.1. Com um batch_size de 64, o treino bastante demorado, sendo essa a razão de terem sido feitas somente 5 épocas. Foi utilizado o optimizer Adam, que ajusta a taxa de aprendizagem (learning-rate) de cada parâmetro individualmente, tornando-o muito mais eficaz em situações onde a escala dos gradientes varia. Foi utilizado data augmentation. Porém, nesta fase inicial, não introduzimos, nesta rede, o dropout. Foi utilizado um learning-rate de 1e-4, sendo que vamos testar esta mesma rede com dropout mais à frente.

Batch_Size 64, 5 épocas Optimizer ADAM Com DropOut Com Data Augmentation Learning_Rate 1e-4



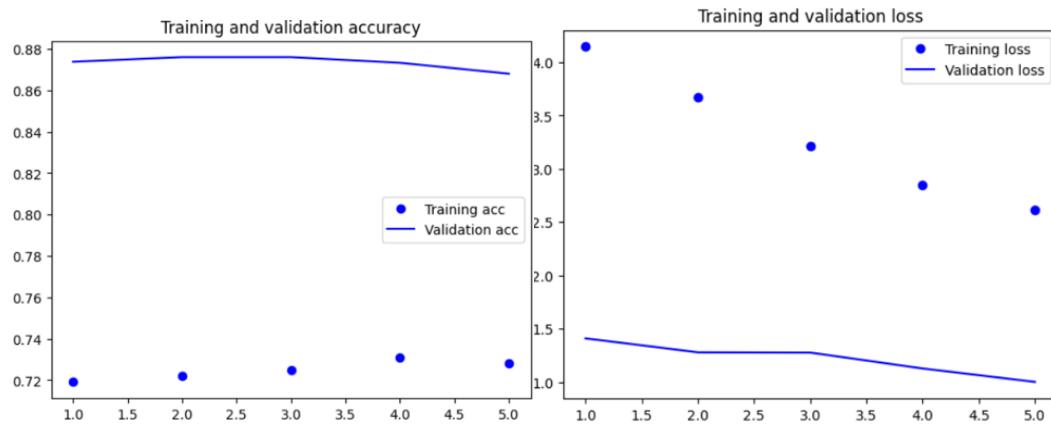
```
from tensorflow import keras
from keras import layers

data_augmentation = keras.Sequential(
[
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.2),
    layers.RandomContrast(0.2),
    layers.RandomBrightness(0.2),
    layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
])

inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg19.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(512)(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs, outputs)
```

Com esta segunda versão da rede, conseguimos uma accuracy de sensivelmente 0.88 e uma loss de 1.56. Com um batch_size de 64, o treino foi bastante demorado, sendo essa a razão de terem sido feitas somente 5 épocas. Foi utilizado o optimizer Adam, que ajusta a taxa de aprendizagem (learning-rate) de cada parâmetro individualmente, tornando-o muito mais eficaz em situações onde a escala dos gradientes varia. Foi utilizado data augmentation. Porém introduzimos também nesta rede o dropout, que desliga aleatoriamente as ligações entre neurónios entre as respetivas camadas. Efetivamente, conseguimos resultados muito melhores utilizando o dropout, um batch_size menor e o optimizer Adam. Foi utilizado um learning-rate de 1e-4, sendo que vamos testar um learning-rate maior, para, no caso de não ocorrer overfitting, termos melhores resultados com um número curto de épocas.

Batch_Size 64, 5 épocas Optimizer ADAM Com DropOut Com Data Augmentation Learning_Rate 1e-2



```
from tensorflow import keras
from keras import layers

data_augmentation = keras.Sequential(
[
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.2),
    layers.RandomContrast(0.2),
    layers.RandomBrightness(0.2),
    layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
])

inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg19.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(512)(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs, outputs)
```

Com esta terceira e última versão da rede, conseguimos uma accuracy de sensivelmente 0.88 e uma loss de 1.12. Com um batch_size de 64, o treino foi bastante demorado, sendo essa a razão de terem sido feitas somente 5 épocas. Foi utilizado o optimizer Adam, que ajusta a taxa de aprendizagem (learning-rate) de cada parâmetro individualmente, tornando-o muito mais eficaz em situações onde a escala dos gradientes varia. Foi utilizado data augmentation. Porém introduzimos também nesta rede o dropout, que desliga aleatoriamente as ligações entre neurónios entre as respetivas camadas. Efetivamente, conseguimos resultados muito melhores utilizando o dropout, um batch_size menor e o optimizer Adam. Foi ajustado o learning-rate para 1e-2, mas os resultados não foram muito diferentes da rede anteriormente relatada. No entanto, é importante aferir que esta rede nunca mostrou overfitting, e eventualmente com mais épocas conseguiríamos accuracy's mais altas, porém, o treino é efetivamente muito pesado e demorado, o que dificulta o treino deste modelo.

Número de Épocas (Dinâmico)

Como descrito no notebook destinado a esta rede, o número de épocas definido inicialmente foi de 5 épocas. No entanto, foram utilizadas outras tecnologias, estando elas relatadas no respetivo notebook.

```
model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learning_rate=1e-4), metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6, verbose=1)
```