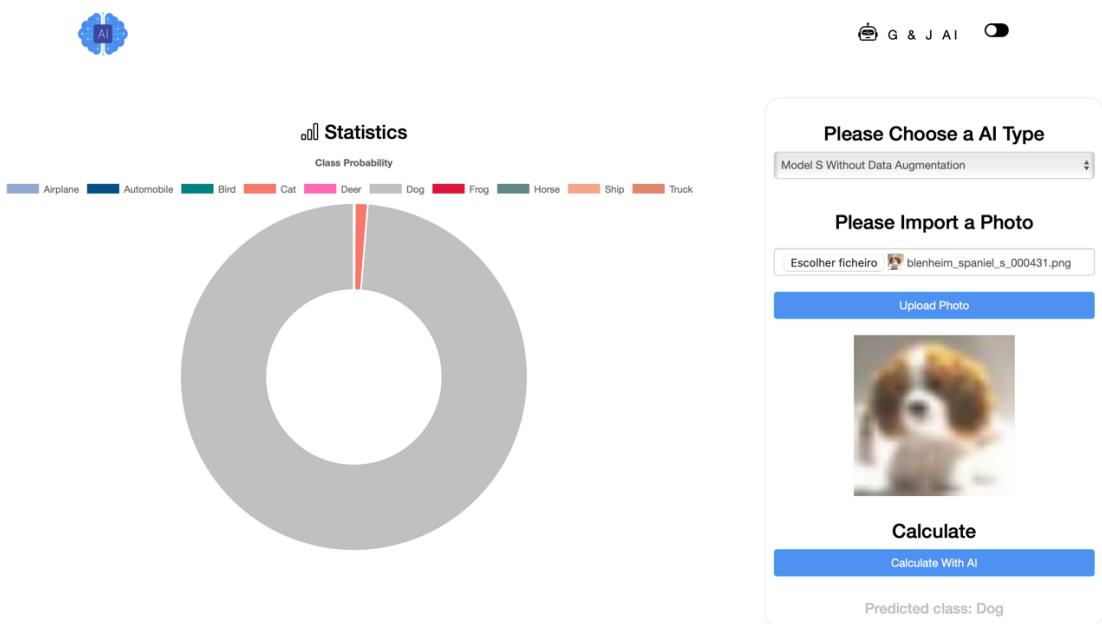


# ModelT\_transferLearning\_featureExtraction\_WithDataAugmentation Results

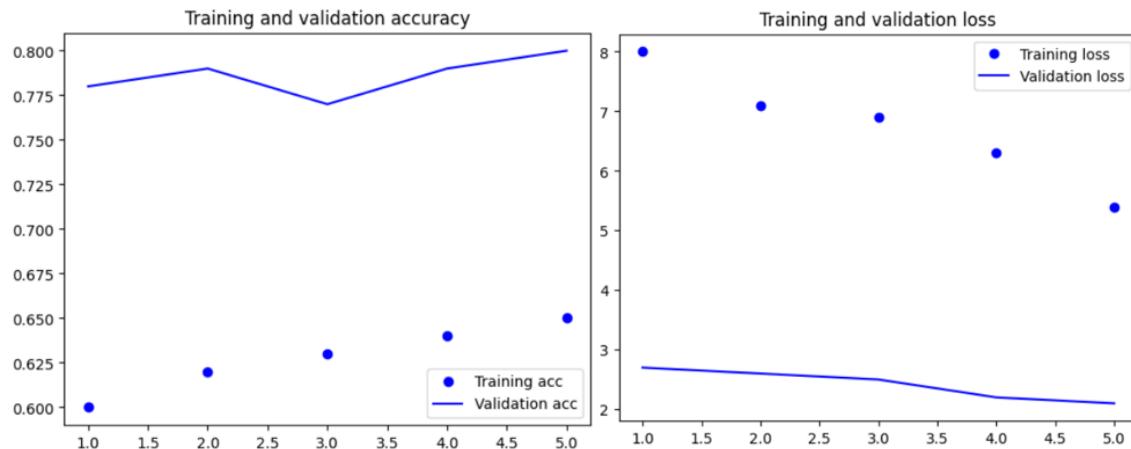


Powered By:

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



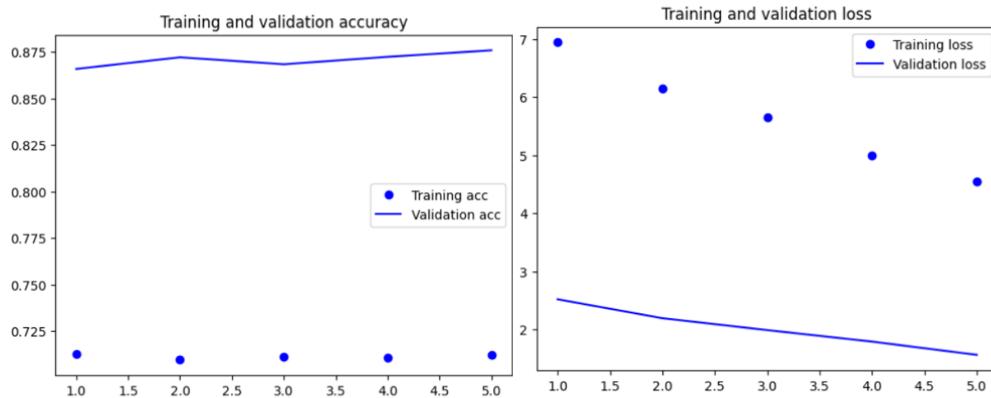
```
from tensorflow import keras
from keras import layers

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



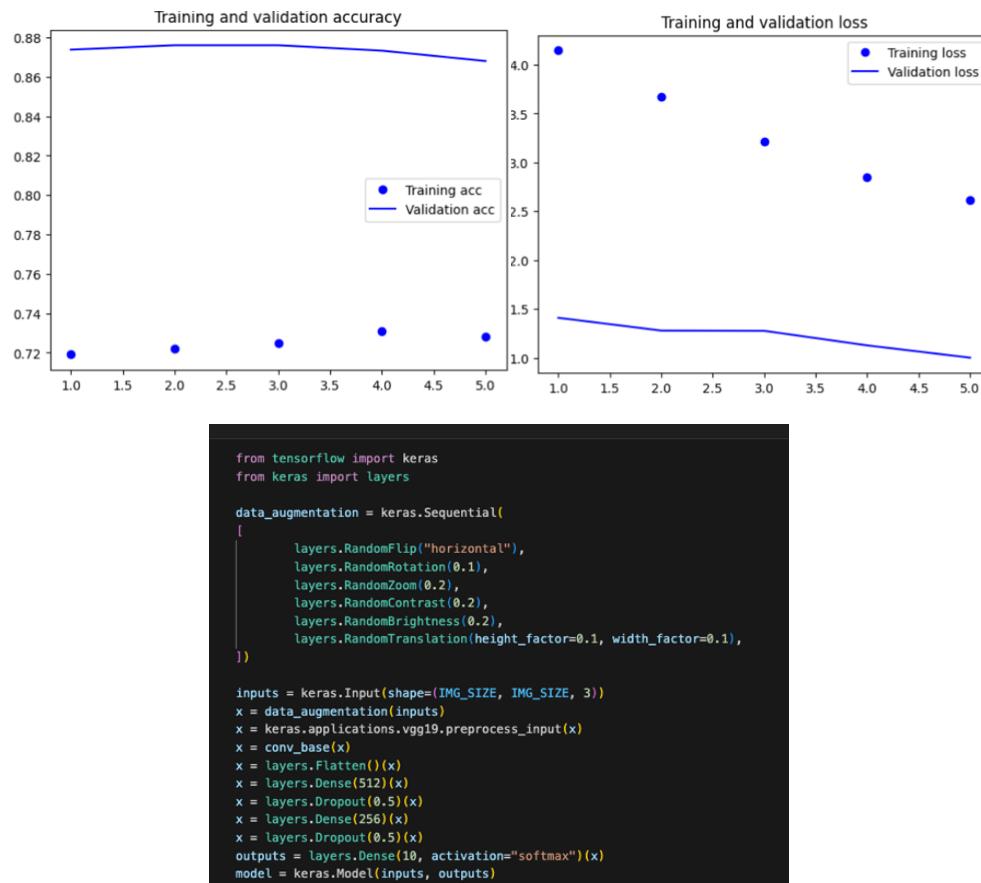
```
from tensorflow import keras
from keras import layers

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 0.87 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, 11(2+2+2+5) épocas Optimizer ADAM Com DropOut Com Data Augmentation Learning\_Rate 1e-6

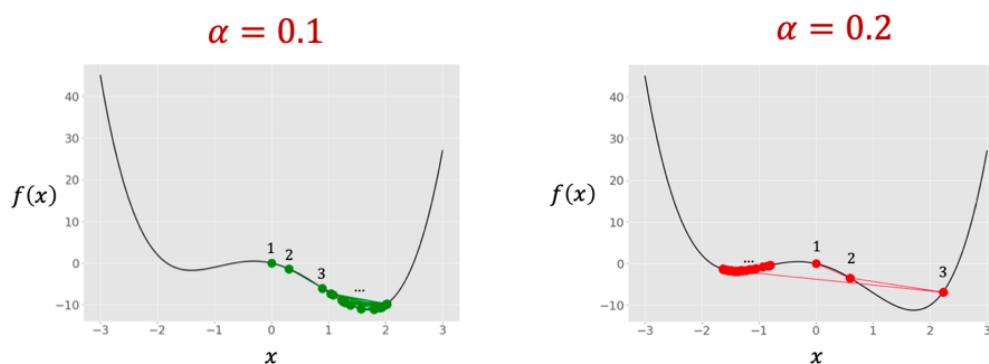


Com esta terceira e última versão da rede, conseguimos uma accuracy de sensivelmente 0.88 e uma loss de 1.12. Com um batch\_size de 64, o treino foi bastante demorado, sendo essa a razão de terem sido feitas somente 11 épocas (épocas distribuídas pelos 3 dataset's sub-divididos). Foi utilizado o optimizer 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-6, pois achámos que iríamos conseguir melhores resultados. Num teste não relatado neste word, tentamos fazer um treino com learning-rate de 1e-2, porém foi detetado overfitting, por se tratar de um valor bastante elevado. No entanto, é importante aferir que esta rede nunca mostrou overfitting tendo em conta o learning-rate de 1e-6, 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.

### Atenção:

Neste modelo, tivemos um problema. Como foi relatado, utilizamos o github para partilha do projeto entre os membros deste projeto. O github, não permite que sejam partilhados ficheiros com mais de 100MB. Assim sendo, perdemos o modelo.h5 desta rede (o modelo relatado até aqui) e tivemos de treinar novamente um modelo novo e não conseguimos atingir os resultados do modelo perdido. Deste ponto para a frente, serão relatados tópicos em relação ao novo modelo e daqui para trás ficam relatados tópicos da evolução do modelo perdido. Tendo em conta este problema, decidimos aumentar a learning-rate deste modelo para 1e-2 na expectativa de forçar a rede a aprender ainda mais rápido. Como pode ser visto, ao longo dos relatórios de resultados do nosso projeto, a learning-rate dos modelos T que melhor resultados nos deu rondava os 1e-5 a 1e-6. Obviamente, que 1e-2 é significativamente mais elevado do que os valores que estávamos habituados a usar. Assim, com learning-rate de 1e-2 obtivemos uma accuracy de 0.83. Um programador aprende com os seus próprios erros, e podemos afirmar, com muita certeza, que este foi um erro que nos criou dores de crescimento.

Na imagem seguinte, vemos um print dos slides das nossas aulas teóricas. Basicamente, o que aconteceu no nosso modelo (no modelo final treinado mais à pressa por falta de tempo), e como podemos exemplificar a partir da imagem, é que efetivamente 1e-2 é um valor de learning-rate muito alto para este tipo de modelo, e a rede acaba por achar, provavelmente, um mínimo local achando que é o melhor resultado possível. De outro modo, e de forma a conseguir o melhor resultado possível, devemos sempre utilizar o learning-rate a 1e-5 a 1e-7, de forma à rede dar passinhos mais pequenos e não ficar presa neste tipo de situações. Este tipo de situações não são as ideias, mas são as que nos fazem compreender melhor a matéria.



Assim sendo, este foram os valores obtidos nas últimas épocas de treino deste modelo.

```

  val_loss, val_acc = model.evaluate(validation_dataset)
  print('val_acc:', val_acc)
  model.save('models/ModelT_transferLearning_featureExtraction_WithDataAumentation.h5')

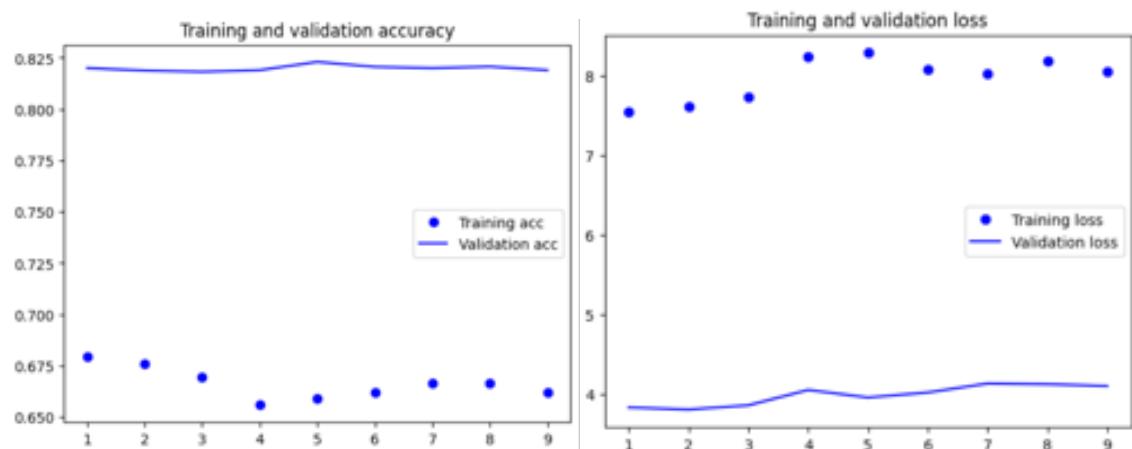
[34]

... 313/313 [=====] - 513s 2s/step - loss: 3830.9585 - accuracy: 0.8305
val acc: 0.830500066757202

Epoch 1/3
500/500 [=====] - 1871s 4s/step - loss: 7444.2793 - accuracy: 0.6895 - val_loss: 3736.1833 - val_accuracy: 0.8303 - lr: 1.0000e-06
Epoch 2/3
500/500 [=====] - 1632s 3s/step - loss: 7518.1289 - accuracy: 0.6858 - val_loss: 3710.3389 - val_accuracy: 0.8288 - lr: 1.0000e-06
Epoch 3/3
500/500 [=====] - 1477s 3s/step - loss: 7639.8521 - accuracy: 0.6796 - val_loss: 3763.7156 - val_accuracy: 0.8282 - lr: 1.0000e-06
Epoch 1/3
375/375 [=====] - 1113s 3s/step - loss: 8139.7832 - accuracy: 0.6658 - val_loss: 3956.6436 - val_accuracy: 0.8290 - lr: 1.0000e-06
Epoch 2/3
375/375 [=====] - 1165s 3s/step - loss: 8191.2666 - accuracy: 0.6692 - val_loss: 3861.0481 - val_accuracy: 0.8333 - lr: 1.0000e-06
Epoch 3/3
375/375 [=====] - 1179s 3s/step - loss: 7989.2520 - accuracy: 0.6722 - val_loss: 3924.6821 - val_accuracy: 0.8306 - lr: 1.0000e-06
Epoch 1/3
375/375 [=====] - 1173s 3s/step - loss: 7936.4229 - accuracy: 0.6764 - val_loss: 4037.8394 - val_accuracy: 0.8300 - lr: 1.0000e-06
Epoch 2/3
375/375 [=====] - 1215s 3s/step - loss: 8091.4941 - accuracy: 0.6765 - val_loss: 4028.4563 - val_accuracy: 0.8307 - lr: 1.0000e-06
Epoch 3/3
375/375 [=====] - 1240s 3s/step - loss: 7952.7788 - accuracy: 0.6721 - val_loss: 4006.9023 - val_accuracy: 0.8290 - lr: 1.0000e-06

```

Mais uma vez, é importante referir que é feita a avaliação do modelo com imagens que nunca foram utilizadas em treino. Assim, é utilizado o validation dataset. De outra forma, é possível testar o modelo, utilizando a APP web disponível no nosso projeto.



Assim, este novo modelo, terminou com uma accuracy de 0.83 e uma loss de 3.8.

De facto, não são os melhores números para este tipo de redes, no entanto, e tendo em conta a situação descrita anteriormente, foi o melhor que conseguimos fazer. Este foi um ponto extremamente importante, pois vimos claramente um problema que nos foi explicado nas aulas teóricas, acontecer-nos.

## 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)
```