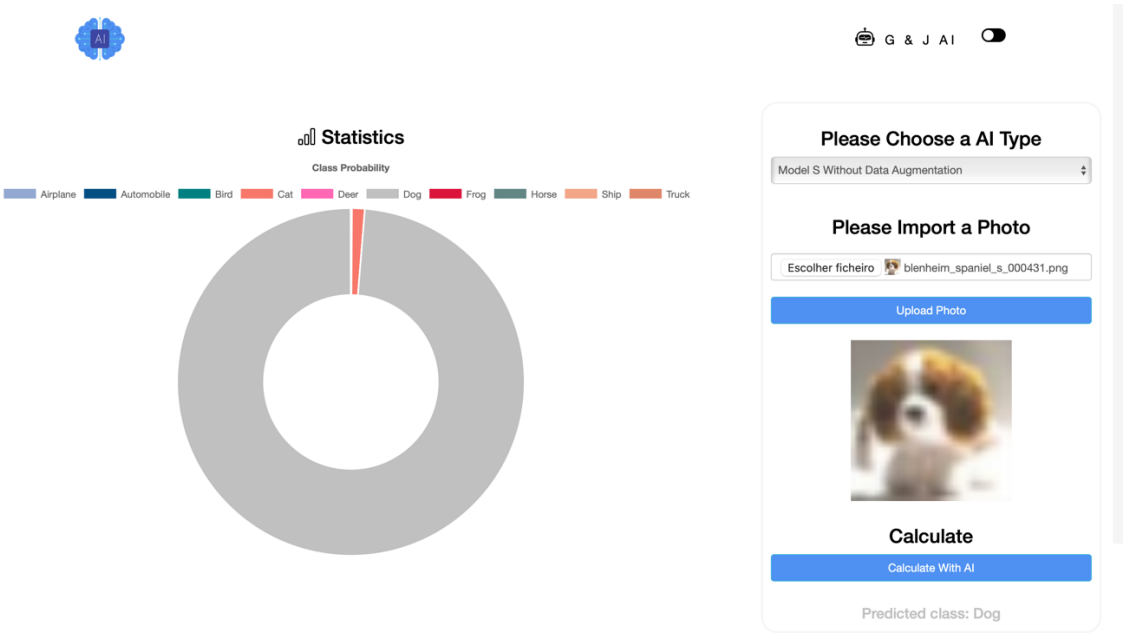
ModelT Transfer Learning Fine Tuning Without Data Augmentation



Powered By:

Gonçalo Ferreira, nº 2222051
José Delgado, nº 2222049

Introdução

Este documento descreve os resultados obtidos com o modelo que usa técnicas de transfer lerning, fine tunig mas sem data augmentation, reutilizamos varias técnicas que usamos ao longo do projeto, conforme descritas no documento 'ModelT_transferLearning_featureExtraction_WithoutDataAumentation', neste documento iremos focar nas funcionalidades novas.

Os passos para realizar o fine tuning a uma rede são os seguintes:

1. Adicionar a nossa rede personalizada em cima de uma rede base já treinada
2. Congelar a rede base (vgg19)
3. Treinar a parte que adicionámos (rede personalizada)
4. Descongelar algumas camadas da rede base (vgg19)
5. Treinar os dois modelos em conjunto o que adicionámos (rede personalizada) e o modelo base(vgg19)

Para acelerar o processo reutilizamos a rede classificadora que já tínhamos treinado no modelo de transfer lerning feature extraction sem data augmentation. Com esta abordagem podemos passar à frente as etapas um, dois e três. Então começamos por importar a VGG19 e de seguida importamos o modelo pré treinado apenas com a parte classificadora. Por fim bastou apenas ligar os modelos, primeiro criamos uma camada de input com o mesmo formato da VGG19, de seguida, passamos essa entrada pelo modelo base VGG19, obtendo a saída do modelo, a saída da vgg19 será o input do modelo pré treinado, finalmente, criamos um novo modelo que combina a camada de entrada, a saída processada pela VGG19 e o modelo pré treinado.

No notebook existe uma função chamada "print_layer_trainable_status", que usamos para verificar que camadas estão definidas como treináveis e também para verificar se a união dos modelos (VGG19 + rede pré treinada (classificação)).

Em relação aos dados, dividimos o conjunto completo em três partes distintas, treino, teste e validação. Essa abordagem permitiu-nos treinar o modelo sequencialmente com cada uma das partes do conjunto de treino, avaliando o desempenho em um conjunto de validação separado após cada fase de treino. Posteriormente, os datasets de treino e de validação cada um deles foi dividido em três partes igualmente proporcionais, garantindo que cada parte representasse aproximadamente um terço do tamanho original do conjunto, realizamos esta operação com o objetivo de os treinos serem menos demorados, conseguindo obter feedback mais rápido pelo modelo, porem notamos que deveríamos ter dividido os datasets em mais partes, para que cada época demorasse ainda menos tempo.

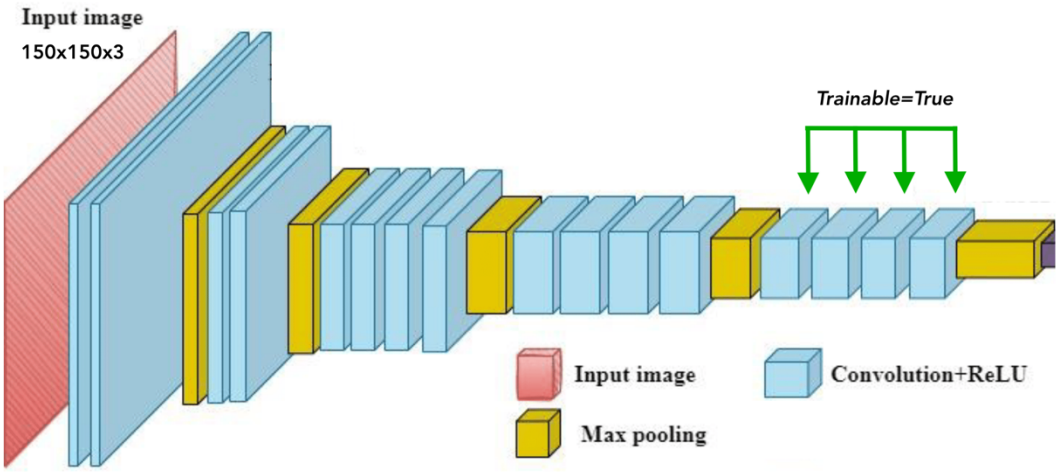
Para chegar a este resultado tivemos de realizar vários treinos, no notebook deste modelo estão todos os treinos realizados para chegar aos valores relatados neste documento.

Resultados

Resultado 1

Primeiramente começamos por descongelar todo o block 5 da VGG19, ou seja, as últimas 5 camadas, isto faz com que os pesos sejam atualizados ao longo do processo de treino, e realizamos alguns treinos.

A seguinte figura ilustra uma representação da VGG19 com o block 5 definido como treinável. Relembrar que a camada de max pooling do block 5 não possui pesos treináveis



A seguinte figura mostra o resultado que obtivemos após a realização de três treinos. Na figura conseguimos concluímos que o modelo não está com overfitting e que o validation accuracy está nos 88%.

```
#Subset 3 -> Continuar com os mesmos valores
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(learning_rate=1e-6, weight_decay=0.5),metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=1e-7)

history = model.fit(train_dataset_3, epochs=4, validation_data=validation_dataset_3, batch_size=64, callbacks=[early_stopping, reduce_lr])
```

✓ 92m 33.1s Python

```
Epoch 1/4
375/375 [==============================] - 1378s 4s/step - loss: 0.4968 - accuracy: 0.8505 - val_loss: 0.3921 - val_accuracy: 0.8743 - lr: 1.0000e-06
Epoch 2/4
375/375 [==============================] - 1389s 4s/step - loss: 0.4539 - accuracy: 0.8602 - val_loss: 0.3868 - val_accuracy: 0.8783 - lr: 1.0000e-06
Epoch 3/4
375/375 [==============================] - 1387s 4s/step - loss: 0.4227 - accuracy: 0.8673 - val_loss: 0.3817 - val_accuracy: 0.8780 - lr: 1.0000e-06
Epoch 4/4
375/375 [==============================] - 1399s 4s/step - loss: 0.3984 - accuracy: 0.8723 - val_loss: 0.3735 - val_accuracy: 0.8810 - lr: 1.0000e-06
```

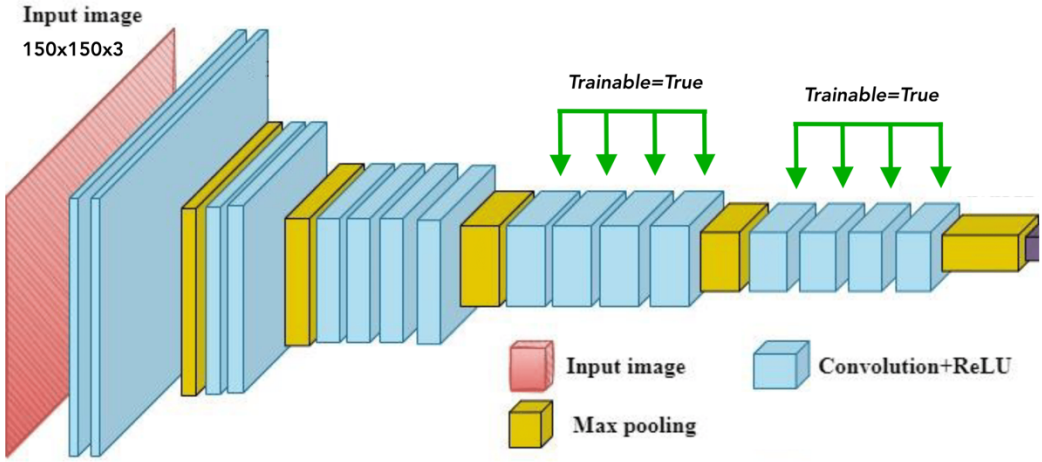
Devido ao resultado descrito acima iremos descongelar mais camadas da VGG19 e retirar novas conclusões sobre os resultados obtidos. Se forem melhores continuamos a descongelar camadas se não paramos e o modelo fica finalizado.

Resultado 2

Como referido no resultado 1 iriamos descongelar mais camadas e está parte demostrará os resultados obtidos.

Primeiramente começamos por descongelar todo o block 4, ou seja, agora as últimas 10 camadas ficaram definidas como treináveis.

A seguinte figura ilustra uma representação da VGG19 com o block 5 e o block 4 definidos como treináveis.



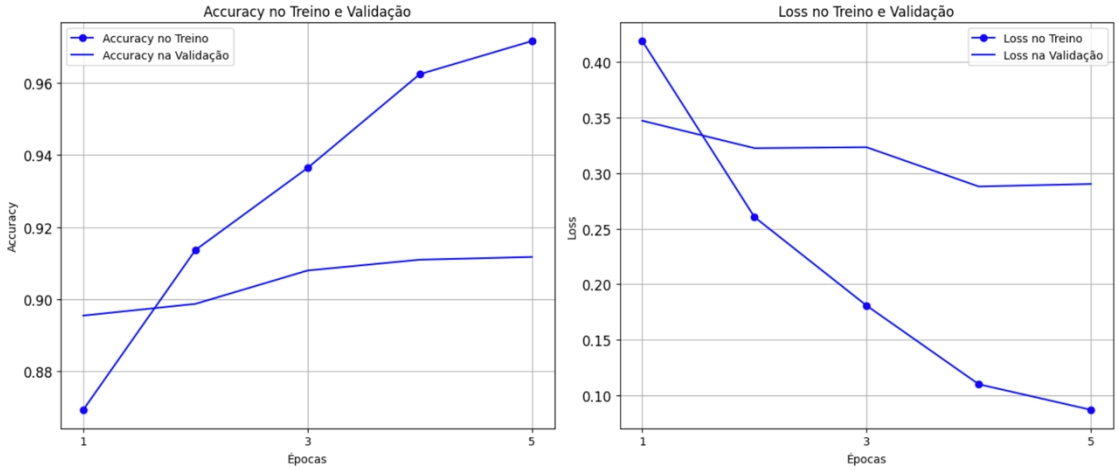
A seguinte figura mostra o resultado que obtivemos após a realização de três treinos. Na figura conseguimos concluímos que o modelo melhorou a validation accuracy agora com aproximadamente 91%, tendo uma melhoria aproximada de 3%. Inconvenientemente o modelo demostrou overfitting.

```
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(learning_rate=1e-5, weight_decay=0.5),metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=1e-7)

history = model.fit(train_dataset_1, epochs=5, validation_data=validation_dataset_1, batch_size=64, callbacks=[early_stopping, reduce_lr])
```

✓ 221m 47.5s Python

```
Epoch 1/5
500/500 [==============================] - 2654s 5s/step - loss: 0.4189 - accuracy: 0.8694 - val_loss: 0.3472 - val_accuracy: 0.8955 - lr: 1.0000e-05
Epoch 2/5
500/500 [==============================] - 2676s 5s/step - loss: 0.2605 - accuracy: 0.9137 - val_loss: 0.3225 - val_accuracy: 0.8988 - lr: 1.0000e-05
Epoch 3/5
500/500 [==============================] - 2655s 5s/step - loss: 0.1808 - accuracy: 0.9364 - val_loss: 0.3233 - val_accuracy: 0.9080 - lr: 1.0000e-05
Epoch 4/5
500/500 [==============================] - 2664s 5s/step - loss: 0.1100 - accuracy: 0.9624 - val_loss: 0.2880 - val_accuracy: 0.9110 - lr: 2.0000e-06
Epoch 5/5
500/500 [==============================] - 2658s 5s/step - loss: 0.0870 - accuracy: 0.9716 - val_loss: 0.2903 - val_accuracy: 0.9118 - lr: 2.0000e-06
```

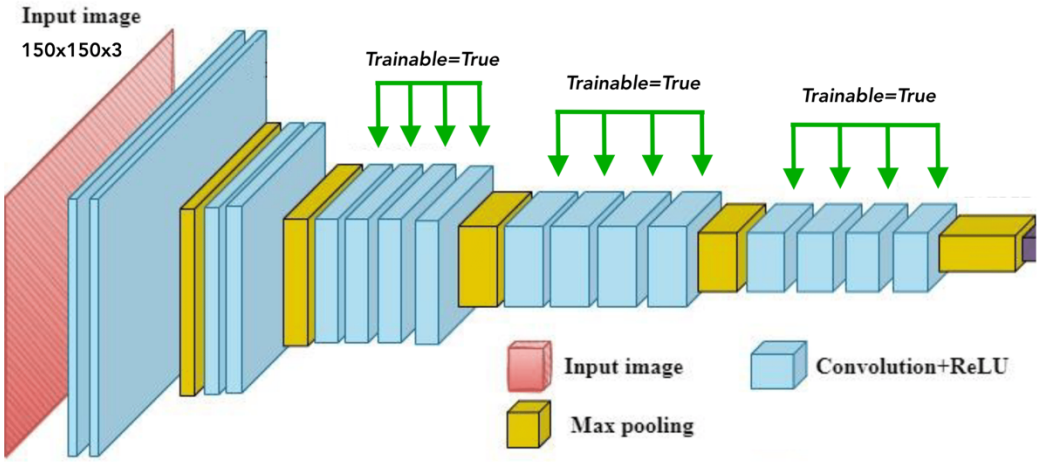
A seguinte figura mostra que o modelo está com overfitting, estando a accuracy do treino quase nos 96% e a accuracy da validação nos 91%.



Resultado Final

Fomos ainda descongelar o block 3 e logo no primeiro treino verificamos que o modelo não conseguiu obter melhores resultados, continuando com o overfitting e a accuracy da validação mantendo-se nos 91%. Portanto não chegamos a guardar o modelo e ficamos com o que relatamos no resultado anterior.

A seguinte figura ilustra uma representação da VGG19 com o block 5, block 4 e o block 3 definidos como treináveis, ou seja, as últimas 15 camadas ajustaram os seus pesos ao longo do processo de treino.



A figura abaixo ilustra o resultado final com os datasets de teste e de validação, conseguimos observar que a validation accuracy para ambos os datasets está nos 90%, que consideramos um bom resultado.

```
from tensorflow import keras

loaded_model = keras.models.load_model('models/ModelT_transferLearning_fineTuning_WithoutDataAumentation.h5')

val_loss, val_acc = loaded_model.evaluate(validation_dataset)
print('val_acc:', val_acc)
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras
313/313 [==============================] - 649s 2s/step - loss: 0.2793 - accuracy: 0.9146
val_acc: 0.9146000146865845
```

```
val_loss, val_acc = loaded_model.evaluate(test_dataset)
print('val_acc:', val_acc)
```

```
313/313 [==============================] - 743s 2s/step - loss: 0.2896 - accuracy: 0.9110
val_acc: 0.9110000133514404
```