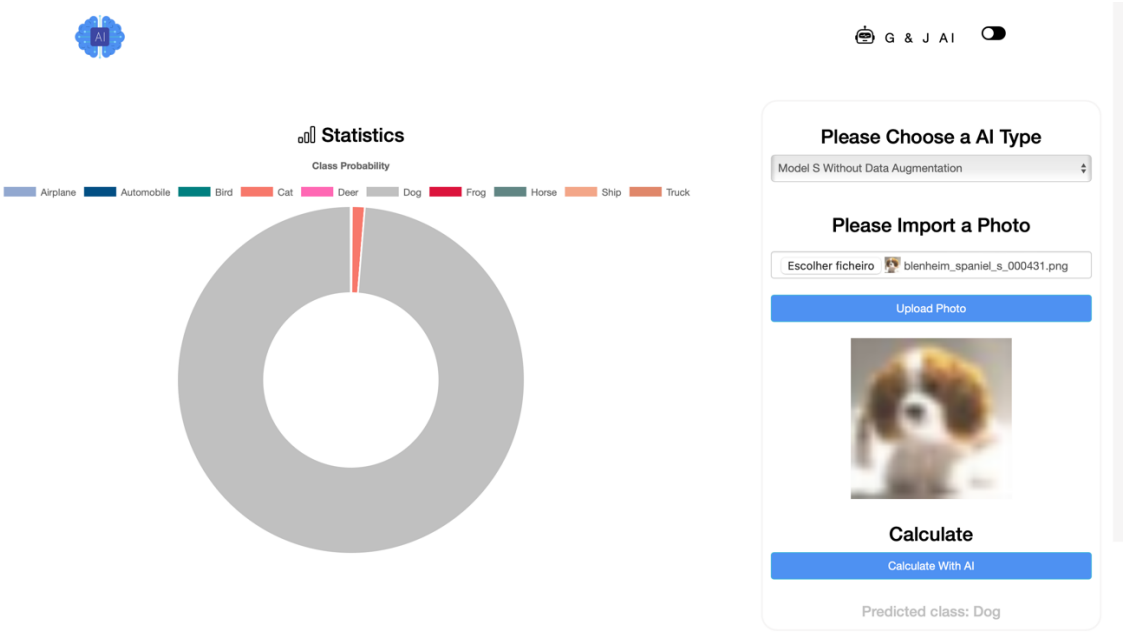
ModelT Transfer Learning Fine Tuning

With 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 e data augmentation, reutilizamos várias 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. Em seguida, aplicamos uma sequência de data augmentation a esse input. Após a data augmentation, passamos o input processado 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 input, 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 à data augmentation usamos cinco técnicas:

1. Flip Horizontal
    a. Esta técnica inverte a imagem ao longo do eixo horizontal, ajuda o modelo a reconhecer objetos independentemente da orientação (esquerda/direita).
2. Rotação
    a. Aplica uma rotação à imagem dentro de um intervalo específico, ajuda o modelo a aprender independentemente da orientação dos objetos.
3. Zoom
    a. Aplica um zoom, aumentando ou diminuindo o tamanho da imagem, isto ajuda o modelo a aprender objetos de diferentes tamanhos.

4. Contraste
    a. Altera o contraste da imagem, aumentando ou diminuindo a diferença entre as partes claras e escuras da imagem, isto ajuda o modelo a aprender objetos sob diferentes condições de luz e contraste.
5. Ruído Gaussiano
    a. Adiciona ruido gaussiano à imagem, isto ajuda o modelo a ser mais robusto a pequenas variações nos dados, imitando o ruído natural que pode estar presente nas imagens naturalmente em imagens do mundo real.

A seguinte imagem ilustra na prática o uso do ruído Gaussiano

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 seis partes igualmente proporcionais, garantindo que cada parte representasse aproximadamente um sexto do tamanho original do conjunto, realizamos esta operação com o objetivo de os treinos serem menos demorados, conseguindo obter feedback mais rápido pelo modelo.

Para além das técnicas utilizadas nos modelos anteriores neste modelo usamos também o Model Checkpoint que resumidamente guarda o melhor modelo conseguido ao longo do processo de treino. Este mecanismo é útil para garantir que o melhor desempenho do modelo durante o treino seja preservado e utilizado posteriormente.

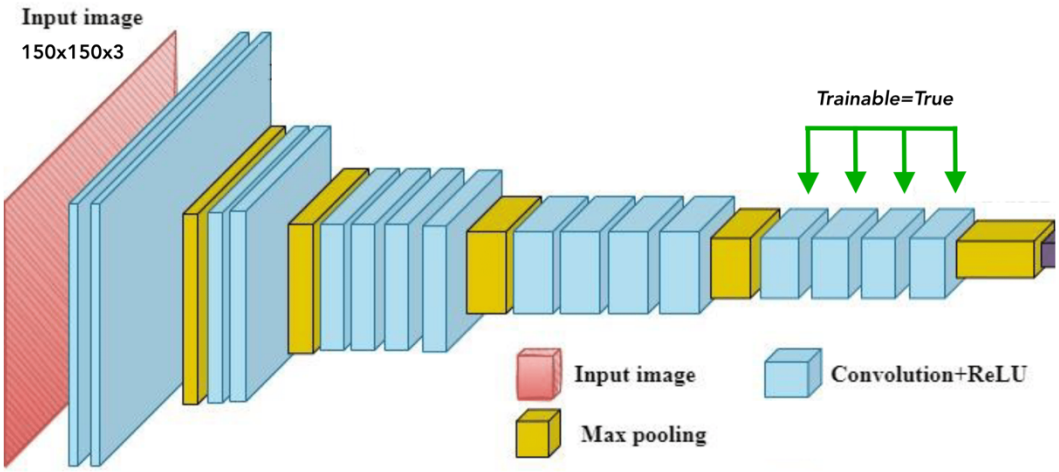
Para chegar a este resultado tivemos de realizar vários treinos, no notebook deste modelo estão todos os treinos realizados para chegar aos valores relatados neste documento.

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 as técnicas que usamos para realizar os treinos e os parâmetros que utilizamos para realizar o primeiro treino, nela conseguimos ver a nova técnica usada, o Model Checkpoint.

```python
#A partir deste bloco iremos treinar o modelo para os sub datasets
import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

checkpoint_callback = ModelCheckpoint(
    filepath='models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5',
    save_best_only=True,
    monitor='val_loss',
    verbose=1
)

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

✓ 0.0s Python

A seguinte figura mostra a realização do primeiro treino com o subdataset 1, no treino foram realizadas dez épocas e conseguimos ver que com o primeiro treino o modelo consegui logo chegar a uma validation accuracy de 87%, também é possível ver a importância do Model Checkpoint, pois no final da última época o modelo que terminou não foi o melhor, tendo sido guardado o melhor modelo no final da oitava época.

```
#Subset 1
history = model.fit(train_dataset_1, epochs=10, validation_data=validation_dataset_1, batch_size=128, callbacks=[checkpoint_callback,early_stopping, reduce_lr])
```

✓ 128m 11.1s Python

```
Epoch 1/10
208/208 [==============================] - ETA: 0s - loss: 1.3291 - accuracy: 0.6418
Epoch 1: val_loss improved from inf to 0.66434, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
208/208 [==============================] - 781s 4s/step - loss: 1.3291 - accuracy: 0.6418 - val_loss: 0.6643 - val_accuracy: 0.8209 - lr: 1.0000e-05
Epoch 2/10
208/208 [==============================] - ETA: 0s - loss: 1.0239 - accuracy: 0.7197
Epoch 2: val_loss improved from 0.66434 to 0.57426, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
208/208 [==============================] - 762s 4s/step - loss: 1.0239 - accuracy: 0.7197 - val_loss: 0.5743 - val_accuracy: 0.8341 - lr: 1.0000e-05
Epoch 3/10
208/208 [==============================] - ETA: 0s - loss: 0.9070 - accuracy: 0.7437
Epoch 3: val_loss improved from 0.57426 to 0.53203, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
208/208 [==============================] - 769s 4s/step - loss: 0.9070 - accuracy: 0.7437 - val_loss: 0.5320 - val_accuracy: 0.8462 - lr: 1.0000e-05
Epoch 4/10
208/208 [==============================] - ETA: 0s - loss: 0.8073 - accuracy: 0.7658
Epoch 4: val_loss improved from 0.53203 to 0.51997, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
208/208 [==============================] - 767s 4s/step - loss: 0.8073 - accuracy: 0.7658 - val_loss: 0.5200 - val_accuracy: 0.8492 - lr: 1.0000e-05
Epoch 5/10
208/208 [==============================] - ETA: 0s - loss: 0.7359 - accuracy: 0.7760
Epoch 5: val_loss did not improve from 0.51997
208/208 [==============================] - 760s 4s/step - loss: 0.7359 - accuracy: 0.7760 - val_loss: 0.5472 - val_accuracy: 0.8528 - lr: 1.0000e-05
Epoch 6/10
208/208 [==============================] - ETA: 0s - loss: 0.6912 - accuracy: 0.7907
Epoch 6: val_loss improved from 0.51997 to 0.50417, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
208/208 [==============================] - 765s 4s/step - loss: 0.6912 - accuracy: 0.7907 - val_loss: 0.5042 - val_accuracy: 0.8552 - lr: 2.0000e-06
Epoch 7/10
208/208 [==============================] - ETA: 0s - loss: 0.6733 - accuracy: 0.7984
Epoch 7: val_loss improved from 0.50417 to 0.49688, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
208/208 [==============================] - 771s 4s/step - loss: 0.6733 - accuracy: 0.7984 - val_loss: 0.4969 - val_accuracy: 0.8624 - lr: 2.0000e-06
...
Epoch 10/10
208/208 [==============================] - ETA: 0s - loss: 0.6443 - accuracy: 0.8005
Epoch 10: val_loss did not improve from 0.45966
208/208 [==============================] - 784s 4s/step - loss: 0.6443 - accuracy: 0.8005 - val_loss: 0.4684 - val_accuracy: 0.8726 - lr: 4.0000e-07
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

A seguinte figura mostra o último treino realizado com o block 5 da VGG19 congelado, na figura observamos que o modelo desde o primeiro treino até o último não teve melhorias, felizmente não obteve overfitting.

```
#Subset 3
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(learning_rate=1e-5, weight_decay=1e-1),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=5, validation_data=validation_dataset_3, batch_size=128, callbacks=[checkpoint_callback,early_stopping, reduce_lr])
```

✓ 63m 49.7s Python

```
Epoch 1/5
208/208 [==============================] - ETA: 0s - loss: 0.8383 - accuracy: 0.7620
Epoch 1: val_loss improved from 0.45966 to 0.45856, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
208/208 [==============================] - 772s 4s/step - loss: 0.8383 - accuracy: 0.7620 - val_loss: 0.4586 - val_accuracy: 0.8648 - lr: 1.0000e-05
Epoch 2/5
208/208 [==============================] - ETA: 0s - loss: 0.7557 - accuracy: 0.7733
Epoch 2: val_loss improved from 0.45856 to 0.44348, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
208/208 [==============================] - 766s 4s/step - loss: 0.7557 - accuracy: 0.7733 - val_loss: 0.4435 - val_accuracy: 0.8666 - lr: 1.0000e-05
Epoch 3/5
208/208 [==============================] - ETA: 0s - loss: 0.6888 - accuracy: 0.7901
Epoch 3: val_loss did not improve from 0.44348
208/208 [==============================] - 769s 4s/step - loss: 0.6888 - accuracy: 0.7901 - val_loss: 0.4573 - val_accuracy: 0.8648 - lr: 1.0000e-05
Epoch 4/5
208/208 [==============================] - ETA: 0s - loss: 0.6338 - accuracy: 0.8071
Epoch 4: val_loss improved from 0.44348 to 0.40545, saving model to models/ModelT_transferLearning_fineTuning_WithDataAumentation_best.h5
208/208 [==============================] - 769s 4s/step - loss: 0.6338 - accuracy: 0.8071 - val_loss: 0.4055 - val_accuracy: 0.8750 - lr: 2.0000e-06
Epoch 5/5
208/208 [==============================] - ETA: 0s - loss: 0.6139 - accuracy: 0.8069
Epoch 5: val_loss did not improve from 0.40545
208/208 [==============================] - 753s 4s/step - loss: 0.6139 - accuracy: 0.8069 - val_loss: 0.4293 - val_accuracy: 0.8678 - lr: 2.0000e-06
```

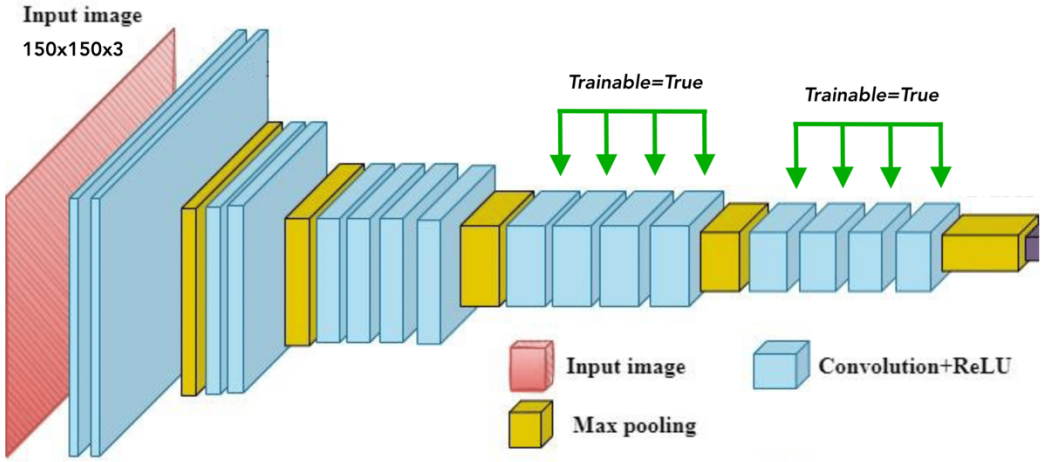
Devido ao resultado descrito acima iremos descongelar mais camadas da VGG19 e retirar novas conclusões sobre os resultados obtidos. Se forem melhores continuamos a descongelar camadas se não paramos e o modelo fica finalizado.

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 da VGG19 camadas ficaram definidas como treináveis.

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



Com o block 4 e o block 5 descongelados realizamos sete treinos, ao longo dos treinos o calback ReduceLROnPlateau reduzia cada vez mais o lerning rate e fomos à medida que os treinos avançavam usando o lerning rate que era sugerido, iniciamos estes treinos com um lerning rate de 1e-2 e acabamos por terminar o setimo treino com um lerning rate de 1e-12, o que demostra a importancia da utilizacao deste callback.

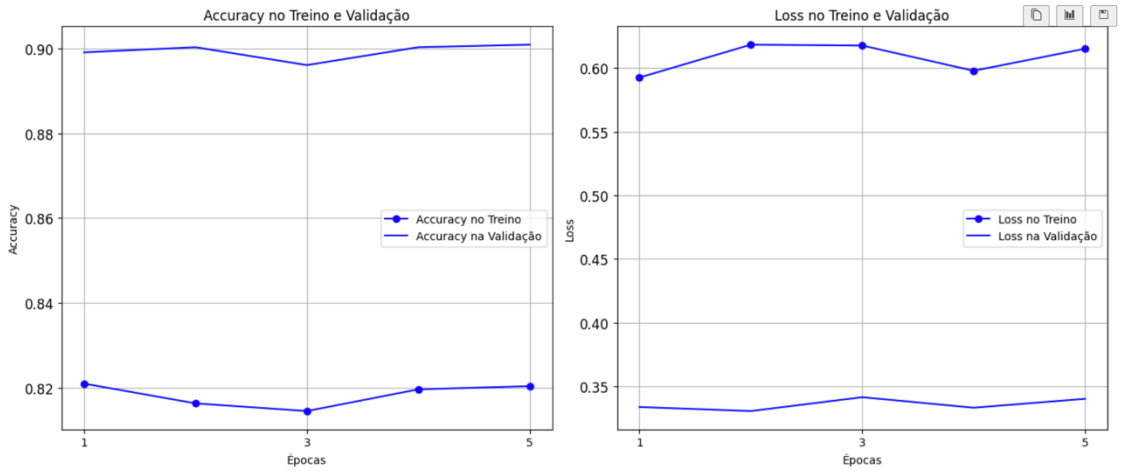
A seguinte figura mostra o resultado do último treino que fizemos com as camadas do block 4 e block 5 defenidos como treináveis. Conseguimos observar que os resultados são bastante positivos, chegando agora a uma validation accuracy de 90%.

```
#Subset 2
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(learning_rate=1e-10, weight_decay=1e-2),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-12)

history = model.fit(train_dataset_2, epochs=5, validation_data=validation_dataset_2, batch_size=128, callbacks=[checkpoint_callback,early_stopping, reduce_lr])
```

✓ 101m 44.8s Python

```
Epoch 1/5
208/208 [==============================] - ETA: 0s - loss: 0.5927 - accuracy: 0.8211
Epoch 1: val_loss did not improve from 0.28645
208/208 [==============================] - 1228s 6s/step - loss: 0.5927 - accuracy: 0.8211 - val_loss: 0.3338 - val_accuracy: 0.8990 - lr: 1.0000e-10
Epoch 2/5
208/208 [==============================] - ETA: 0s - loss: 0.6185 - accuracy: 0.8164
Epoch 2: val_loss did not improve from 0.28645
208/208 [==============================] - 1202s 6s/step - loss: 0.6185 - accuracy: 0.8164 - val_loss: 0.3306 - val_accuracy: 0.9002 - lr: 1.0000e-10
Epoch 3/5
208/208 [==============================] - ETA: 0s - loss: 0.6178 - accuracy: 0.8146
Epoch 3: val_loss did not improve from 0.28645
208/208 [==============================] - 1268s 6s/step - loss: 0.6178 - accuracy: 0.8146 - val_loss: 0.3415 - val_accuracy: 0.8960 - lr: 1.0000e-10
Epoch 4/5
208/208 [==============================] - ETA: 0s - loss: 0.5980 - accuracy: 0.8197
Epoch 4: val_loss did not improve from 0.28645
208/208 [==============================] - 1185s 6s/step - loss: 0.5980 - accuracy: 0.8197 - val_loss: 0.3333 - val_accuracy: 0.9002 - lr: 2.0000e-11
Epoch 5/5
208/208 [==============================] - ETA: 0s - loss: 0.6153 - accuracy: 0.8205
Epoch 5: val_loss did not improve from 0.28645
208/208 [==============================] - 1221s 6s/step - loss: 0.6153 - accuracy: 0.8205 - val_loss: 0.3402 - val_accuracy: 0.9008 - lr: 4.0000e-12
```

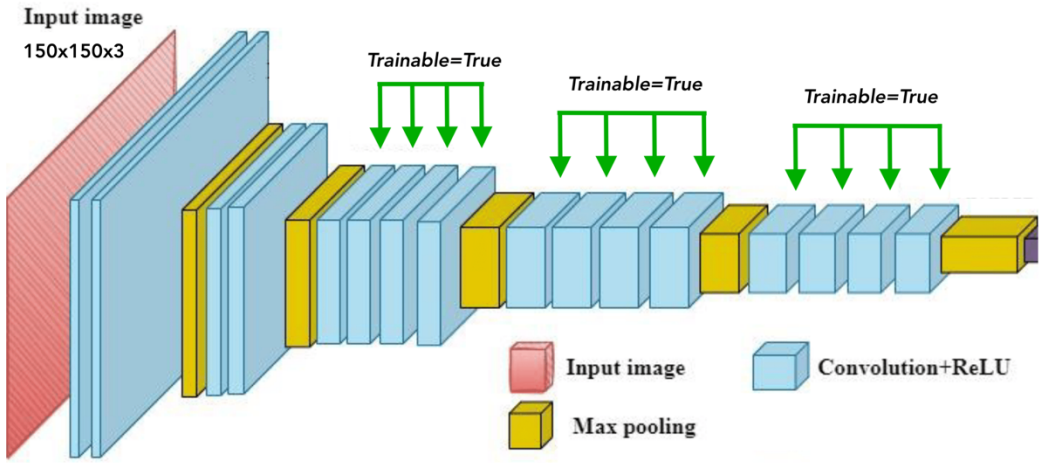
A seguinte figura mostra o gráfico que ilustra os resultados do modelo, nele conseguimos ver que o modelo não tem overfitting e que a validation accuracy se encontra nos 90%, iremos descongelar o block 3 da VGG19 para tentar obter melhores resultados.



Resultado Final

Fomos ainda descongelar o block 3 e após alguns treinos verificamos que o modelo não conseguiu obter melhores resultados, na verdade até começou a dar sinais de overfitting e a accuracy da validação mantendo-se nos 90%. Portanto devido ao uso do calback Model Checkpoint o modelo final ficou guardado e foi atingido em treinos anteriores com apenas o block 4 e block 5 descongelados.

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.



As seguintes figuras mostram os resultados ao descongelar as camadas do block 3, nela conseguimos verificar o que foi relatado anteriormente, ou seja, o modelo não obteve melhores resultados, começando a dar evidencias de overfitting.

```
#Subset 2
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(learning_rate=1e-11, weight_decay=1e-2),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-12)

history = model.fit(train_dataset_2, epochs=5, validation_data=validation_dataset_2, batch_size=128, callbacks=[checkpoint_callback,early_stopping, reduce_lr])
```

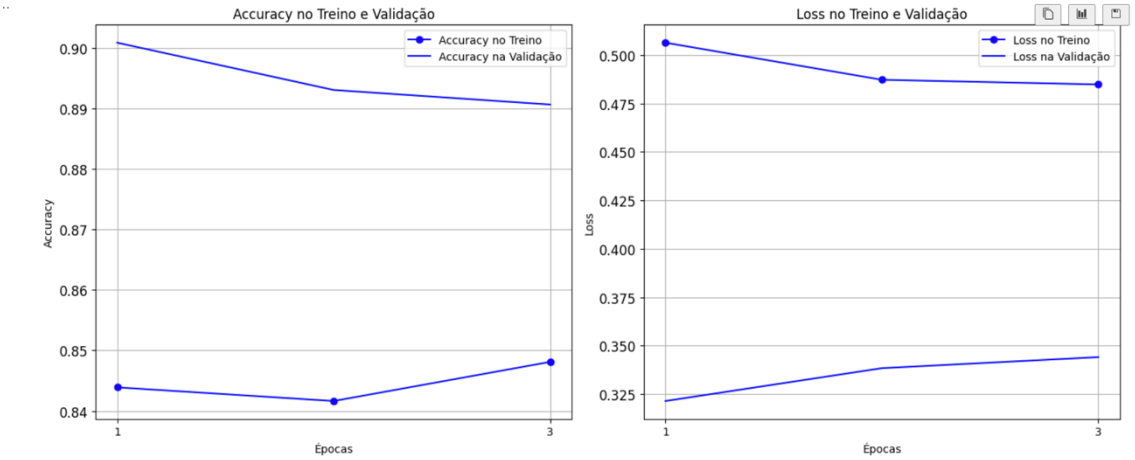
✓ 133m 1.5s Python

```
Epoch 1/5
208/208 [==============================] - ETA: 0s - loss: 0.6051 - accuracy: 0.8143
Epoch 1: val_loss did not improve from 0.28645
208/208 [==============================] - 1509s 7s/step - loss: 0.6051 - accuracy: 0.8143 - val_loss: 0.3361 - val_accuracy: 0.8978 - lr: 1.0000e-11
Epoch 2/5
208/208 [==============================] - ETA: 0s - loss: 0.6090 - accuracy: 0.8199
Epoch 2: val_loss did not improve from 0.28645
208/208 [==============================] - 1649s 8s/step - loss: 0.6090 - accuracy: 0.8199 - val_loss: 0.3455 - val_accuracy: 0.8990 - lr: 1.0000e-11
Epoch 3/5
208/208 [==============================] - ETA: 0s - loss: 0.5990 - accuracy: 0.8190
Epoch 3: val_loss did not improve from 0.28645
208/208 [==============================] - 1665s 8s/step - loss: 0.5990 - accuracy: 0.8190 - val_loss: 0.3233 - val_accuracy: 0.9038 - lr: 2.0000e-12
Epoch 4/5
208/208 [==============================] - ETA: 0s - loss: 0.5992 - accuracy: 0.8190
Epoch 4: val_loss did not improve from 0.28645
208/208 [==============================] - 1532s 7s/step - loss: 0.5992 - accuracy: 0.8190 - val_loss: 0.3416 - val_accuracy: 0.9002 - lr: 2.0000e-12
Epoch 5/5
208/208 [==============================] - ETA: 0s - loss: 0.6100 - accuracy: 0.8167
Epoch 5: val_loss did not improve from 0.28645
208/208 [==============================] - 1627s 8s/step - loss: 0.6100 - accuracy: 0.8167 - val_loss: 0.3271 - val_accuracy: 0.9032 - lr: 1.0000e-12
```

```python
#Subset 4
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(learning_rate=1e-12, weight_decay=1e-2),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-14)

history = model.fit(train_dataset_4, epochs=3, validation_data=validation_dataset_4, batch_size=128, callbacks=[checkpoint_callback,early_stopping, reduce_lr])
```

✓ 75m 49.7s Python

```
Epoch 1/3
208/208 [==============================] - ETA: 0s - loss: 0.5064 - accuracy: 0.8439
Epoch 1: val_loss did not improve from 0.28645
208/208 [==============================] - 1506s 7s/step - loss: 0.5064 - accuracy: 0.8439 - val_loss: 0.3214 - val_accuracy: 0.9008 - lr: 1.0000e-12
Epoch 2/3
208/208 [==============================] - ETA: 0s - loss: 0.4873 - accuracy: 0.8416
Epoch 2: val_loss did not improve from 0.28645
208/208 [==============================] - 1516s 7s/step - loss: 0.4873 - accuracy: 0.8416 - val_loss: 0.3384 - val_accuracy: 0.8930 - lr: 1.0000e-12
Epoch 3/3
208/208 [==============================] - ETA: 0s - loss: 0.4849 - accuracy: 0.8481
Epoch 3: val_loss did not improve from 0.28645
208/208 [==============================] - 1527s 7s/step - loss: 0.4849 - accuracy: 0.8481 - val_loss: 0.3441 - val_accuracy: 0.8906 - lr: 2.0000e-13
```



A figura abaixo ilustra o resultado final com o dataset de teste, conseguimos observar que a validation accuracy está nos 89%.

```python
from tensorflow import keras

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

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

✓ 13m 31.3s Python

```
313/313 [==============================] - 808s 3s/step - loss: 0.3491 - accuracy: 0.8910
val_acc: 0.890999972820282
```

O modelo anterior igual a este, mas sem data augmentation teve resultados bastante parecidos. Depois de algumas pesquisas podemos ter algumas respostas para este acontecimento, como, a arquitetura escolhida pois ambos usavam a mesma, a complexidade do problema, a qualidade do dataset e as técnicas utilizadas na data augmentation. Podem existir inúmeras razoes, tentamos perceber quais delas poderão ter acontecido.