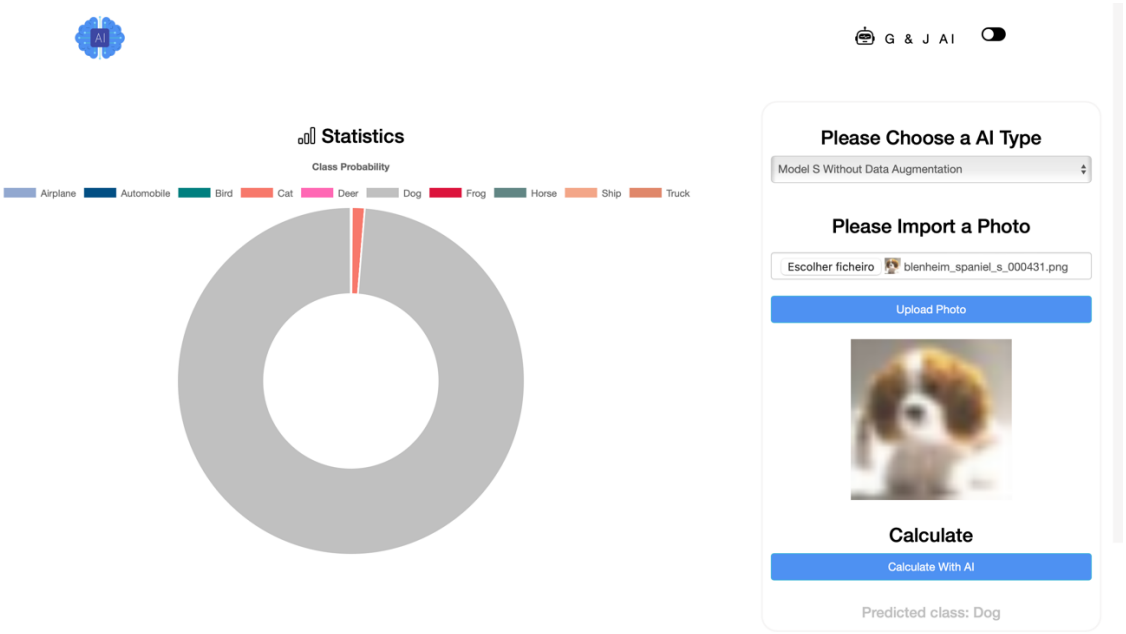
ModelT Transfer Learning Feature Extraction Without Data Augmentation Results



Powered By:

Gonçalo Ferreira, nº 2222051
José Delgado, nº 2222049

Introdução

Neste modelo com a técnica de transfer learning usamos a vgg19, a principal diferença entre a vgg16 e a vgg19 reside no número de camadas convolutional. A VGG16 possui 13 camadas convolutional e 3 camadas fully connected, totalizando 16 camadas treináveis. Em contrapartida, a vgg19 tem 16 camadas convolutional e 3 camadas fully connected, somando 19 camadas treináveis. Essa diferença de três camadas adicionais na vgg19 permite que a rede capture características mais complexas dos dados, aumentando o número total de parâmetros da rede e, consequentemente, a sua capacidade de representação. No entanto, isso também implica em maior complexidade computacional, maior tempo de treino e um potencial aumento no risco de overfitting se não for adequadamente gerenciada.

No início do projeto usamos três optimizadores distintos o RMSprop, o SGD (Stochastic Gradient Descent) e o Adam. Após alguns testes não gostamos dos resultados do SGD e acabamos por usar mais o RMSprop e depois o Adam, sendo por fim o Adam o optimizador que mais usamos. O Adam é um algoritmo de otimização popular usado em deep lerning, ele pode ser visto como a combinação do RMSprop e do SGD, ele combina os melhores recursos do momento (para acelerar a convergência) e da adaptação do lerning rate para cada parâmetro com base no histórico de gradientes (para controlar a atualização de maneira adaptativa).

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, cada um desses conjuntos foi dividido em quatro partes igualmente proporcionais, garantindo que cada parte representasse um quarto do tamanho original do conjunto, realizamos esta operação com o objetivo de os treinos serem menos demorados, conseguindo obter feedback mais rápido pelo modelo.

Não colocamos todos os testes que fizemos porque por vezes era uma alteração pequena em valores, como por exemplo o lerning rate, que variávamos várias vezes entre 1e-6 e 1e-2 e o Dropout, que variávamos dependendo do overfitting, etc... Para chegar aos resultados obtidos tivemos de fazer vários testes e não conseguimos registar todos, ao longo do relatório relatamos os mais importantes, no notebook deste modelo todos os treinos realizados para chegar aos valores relatados neste documento estão lá presentes.

Resultados

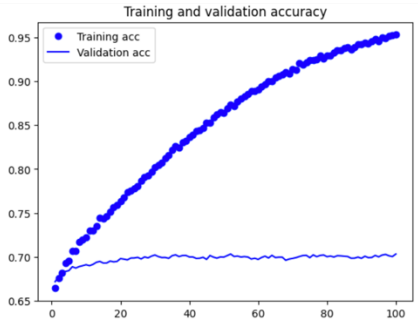
Resultado 1

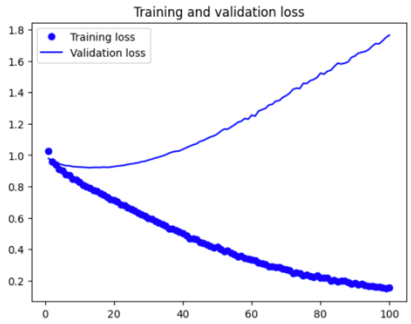
```python
from tensorflow import keras
from keras import layers

inputs = keras.Input(shape=(1, 1, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(4096, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(2048, activation='relu')(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(10, activation="softmax")(x)

model = keras.Model(inputs, outputs)
```

✓ 0.0s Python

```python
import tensorflow as tf
model.compile(loss='categorical_crossentropy',optimizer=tf.keras.optimizers.RMSprop(learning_rate=2e-5),metrics=['accuracy'])

history = model.fit(train_features, train_labels,epochs=100, batch_size=64,validation_data=(val_features, val_labels))
```

✓ 548m 17.9s Python





Este foi o primeiro treino que nós fizemos com os modelos T, os resultados são muito fracos o validation accuracy está quase nos 70% o training accuracy está nos 0.95%, temos, portanto, overfitting.

Este resultado já era esperado, pois a arquitetura do modelo é muito complexa, o que faz com que ele aprenda os padrões do conjunto de treino muito bem, mas falhe em generalizar para novos dados.

Este treino contém outro problema que é o tamanho definido das imagens, no início definimos as imagens de tamanho 32x32 o que faz com que o output do feature extraction da vgg19, os seus feature maps se tornem demasiado pequenos ou mesmo inválidos

Resultado 2

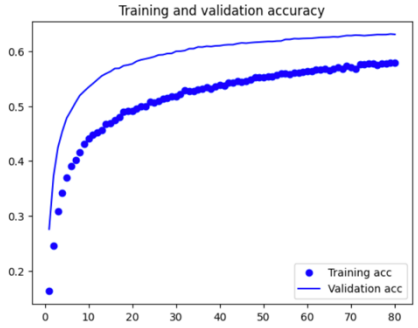
```python
from tensorflow import keras
from keras import layers
from tensorflow.keras.regularizers import l2

inputs = keras.Input(shape=(1, 1, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001))(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(10, activation="softmax")(x)

model = keras.Model(inputs, outputs)
```

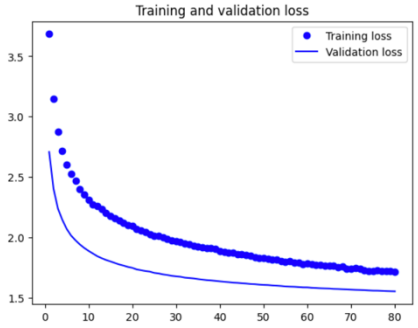
Python

+ Code + Markdown

```python
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau

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-5)

history = model.fit(train_features, train_labels,epochs=40, batch_size=32, validation_data=(val_features, val_labels),callbacks=[early_stopping, reduce_lr])
```

Python



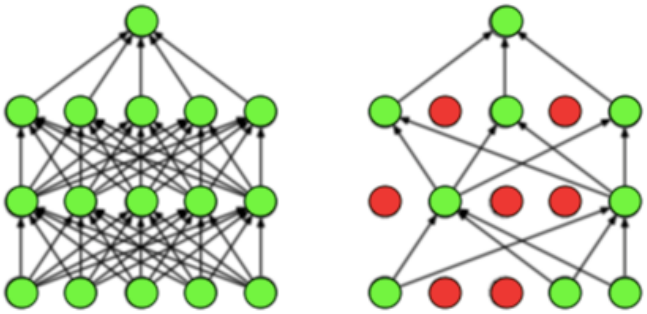


O melhor resultado que obtivemos com imagens de 32x32 foi o apresentado acima, reduzimos a complexidade da arquitetura para evitar o overfiting, e mudamos o optimizador para o Adam.

Também usamos calbacks, colocamos Early Stopping é uma técnica utilizada durante o treino de um modelo para evitar o overfitting, ela monitoriza uma métrica especificada (neste caso, 'val_loss' que é a perda no conjunto de validação) e interrompe o treino se a métrica parar de melhorar por um número especificado de épocas (definido pelo parâmetro patience), ou seja para que o modelo não continue a treinar desnecessariamente. O ReduceLROnPlateau é uma técnica que ajusta o lerning rate durante o treino do modelo. Esta técnica também acompanha uma métrica especificada (neste caso, 'val_loss').

Quando a métrica não mostra melhoria por um número determinado de épocas (definido pelo parâmetro patience), o lerning rate é reduzida por um fator especificado (definido pelo parâmetro factor). Isso permite que o modelo continue a aprender de maneira mais refinada, ajudando a alcançar uma convergência mais estável e precisa. Por fim temos a técnica de droupout, é uma técnica de regularização em redes neurais que combate o overfitting, durante o treino, o dropout desativa aleatoriamente um conjunto de unidades neurais em uma camada, forçando a rede a aprender características mais robustas e generalizáveis. Essencialmente, isso impede que unidades individuais se tornem muito dependentes umas das outras, promovendo uma representação mais diversificada dos dados.

A figura abaixo ilustra o funcionamento da técnica de dropout.



(a) Standard Neural Net (b) After applying dropout.

Resultado Final

```python
from tensorflow import keras
from keras import layers
from tensorflow.keras.regularizers import l2

inputs = keras.Input(shape=(4, 4, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(512, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.7)(x)
outputs = layers.Dense(10, activation="softmax")(x)

model = keras.Model(inputs, outputs)
```

✓ 0.2s Python

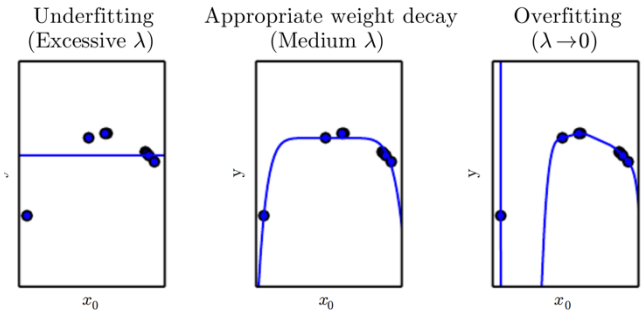
```python
# Aumentar o weight_decay para penalizar mais os pesos grandes e ajudar a reduzir o overfitting
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(learning_rate=1e-6, weight_decay=0.5),metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-7)

history = model.fit(train_features_3, train_labels_3,epochs=120, batch_size=32, validation_data=(val_features_3, val_labels_3),callbacks=[early_stopping,reduc
```

✓ 5m 10.0s Python

A imagem acima mostra a arquitetura obtida no final, nela podemos ver que o input já é de (4, 4, 512), o que significa que definimos o tamanho das imagens para 150x150, assim as features calculadas pela vgg19 já não irão ter tamanho inválidos.

Nos últimos treinos aumentamos o valor da variável weight_decay, pois o modelo estava com overffiting. O parâmetro weight_decay, também conhecido como regularização L2, adiciona uma penalização à função loss do modelo com base na magnitude dos pesos, incentivando o modelo a manter pesos pequenos. Ao penalizar pesos grandes, ajuda a evitar que o modelo se ajuste excessivamente aos dados de treino, promovendo melhor generalização para dados novos, o inverso ou seja baixar o valor do parâmetro significa menos regularização e mais overfitting, mas também mais flexibilidade e menos underfitting.

A figura abaixo ilustra a resposta do modelo a diferentes níveis de regularização, mostrando os efeitos de underfitting, regularização apropriada e overfitting.



As figuras abaixo ilustram a melhoria após mudar o parâmetro de weight_decay, a primeira figura mostra a pré mudança e a segunda figura mostra a pós mudança. Talvez o modelo iria continuar com overfiting se a rede continuasse a treinar, mas as escalas de diferenças mudaram, ou seja, o overfiting no segundo exemplo só começou depois dos 90%.

```
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(learning_rate=1e-5, weight_decay=1e-2),metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6)

history = model.fit(train_features_1, train_labels_1,epochs=120, batch_size=32, validation_data=(val_features_1, val_labels_1),callbacks=[early_stopping,reduce_lr])
```

✓ 14m 15.7s Python

```
Epoch 1/120
312/312 [==============================] - 12s 37ms/step - loss: 3.0947 - accuracy: 0.2507 - val_loss: 1.2068 - val_accuracy: 0.5998 - lr: 1.0000e-05
Epoch 2/120
312/312 [==============================] - 12s 37ms/step - loss: 1.7888 - accuracy: 0.4852 - val_loss: 0.8409 - val_accuracy: 0.7204 - lr: 1.0000e-05
Epoch 3/120
312/312 [==============================] - 11s 36ms/step - loss: 1.3575 - accuracy: 0.6006 - val_loss: 0.7064 - val_accuracy: 0.7696 - lr: 1.0000e-05
Epoch 4/120
312/312 [==============================] - 11s 36ms/step - loss: 1.1159 - accuracy: 0.6582 - val_loss: 0.6381 - val_accuracy: 0.7961 - lr: 1.0000e-05
Epoch 5/120
312/312 [==============================] - 11s 37ms/step - loss: 1.0179 - accuracy: 0.6916 - val_loss: 0.5953 - val_accuracy: 0.8105 - lr: 1.0000e-05
Epoch 6/120
312/312 [==============================] - 11s 37ms/step - loss: 0.8739 - accuracy: 0.7325 - val_loss: 0.5673 - val_accuracy: 0.8213 - lr: 1.0000e-05
Epoch 7/120
312/312 [==============================] - 11s 36ms/step - loss: 0.7888 - accuracy: 0.7524 - val_loss: 0.5455 - val_accuracy: 0.8261 - lr: 1.0000e-05
Epoch 8/120
312/312 [==============================] - 12s 38ms/step - loss: 0.7323 - accuracy: 0.7715 - val_loss: 0.5298 - val_accuracy: 0.8301 - lr: 1.0000e-05
Epoch 9/120
312/312 [==============================] - 13s 41ms/step - loss: 0.6778 - accuracy: 0.7838 - val_loss: 0.5174 - val_accuracy: 0.8345 - lr: 1.0000e-05
Epoch 10/120
312/312 [==============================] - 11s 37ms/step - loss: 0.6257 - accuracy: 0.7954 - val_loss: 0.5076 - val_accuracy: 0.8373 - lr: 1.0000e-05
Epoch 11/120
312/312 [==============================] - 12s 37ms/step - loss: 0.5847 - accuracy: 0.8127 - val_loss: 0.4965 - val_accuracy: 0.8397 - lr: 1.0000e-05
Epoch 12/120
312/312 [==============================] - 11s 36ms/step - loss: 0.5358 - accuracy: 0.8241 - val_loss: 0.4896 - val_accuracy: 0.8429 - lr: 1.0000e-05
Epoch 13/120
...
Epoch 68/120
312/312 [==============================] - 13s 41ms/step - loss: 0.1176 - accuracy: 0.9619 - val_loss: 0.4476 - val_accuracy: 0.8618 - lr: 1.0000e-06
Epoch 69/120
312/312 [==============================] - 14s 45ms/step - loss: 0.1082 - accuracy: 0.9650 - val_loss: 0.4489 - val_accuracy: 0.8618 - lr: 1.0000e-06
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
# Aumentar o weight_decay para penalizar mais os pesos grandes e ajudar a reduzir o overfitting
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=15, restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-7)

history = model.fit(train_features_3, train_labels_3,epochs=120, batch_size=32, validation_data=(val_features_3, val_labels_3),callbacks=[early_stopping,reduce_lr])
```

✓ 5m 10.0s Python

```
Epoch 1/120
312/312 [==============================] - 13s 39ms/step - loss: 0.2775 - accuracy: 0.9082 - val_loss: 0.3137 - val_accuracy: 0.8986 - lr: 1.0000e-06
Epoch 2/120
312/312 [==============================] - 12s 37ms/step - loss: 0.2872 - accuracy: 0.9031 - val_loss: 0.3135 - val_accuracy: 0.8998 - lr: 1.0000e-06
Epoch 3/120
312/312 [==============================] - 11s 37ms/step - loss: 0.2916 - accuracy: 0.8986 - val_loss: 0.3137 - val_accuracy: 0.8994 - lr: 1.0000e-06
Epoch 4/120
312/312 [==============================] - 11s 37ms/step - loss: 0.2787 - accuracy: 0.9040 - val_loss: 0.3143 - val_accuracy: 0.8990 - lr: 1.0000e-06
Epoch 5/120
312/312 [==============================] - 11s 37ms/step - loss: 0.2770 - accuracy: 0.9069 - val_loss: 0.3138 - val_accuracy: 0.8994 - lr: 1.0000e-06
Epoch 6/120
312/312 [==============================] - 12s 40ms/step - loss: 0.2710 - accuracy: 0.9072 - val_loss: 0.3139 - val_accuracy: 0.9002 - lr: 1.0000e-06
Epoch 7/120
312/312 [==============================] - 12s 37ms/step - loss: 0.2615 - accuracy: 0.9109 - val_loss: 0.3132 - val_accuracy: 0.8994 - lr: 1.0000e-06
Epoch 8/120
312/312 [==============================] - 12s 38ms/step - loss: 0.2696 - accuracy: 0.9101 - val_loss: 0.3138 - val_accuracy: 0.8994 - lr: 1.0000e-06
Epoch 9/120
312/312 [==============================] - 12s 37ms/step - loss: 0.2655 - accuracy: 0.9094 - val_loss: 0.3137 - val_accuracy: 0.8982 - lr: 1.0000e-06
Epoch 10/120
312/312 [==============================] - 12s 38ms/step - loss: 0.2700 - accuracy: 0.9068 - val_loss: 0.3139 - val_accuracy: 0.8990 - lr: 1.0000e-06
Epoch 11/120
312/312 [==============================] - 12s 38ms/step - loss: 0.2645 - accuracy: 0.9097 - val_loss: 0.3132 - val_accuracy: 0.9002 - lr: 1.0000e-06
Epoch 12/120
312/312 [==============================] - 12s 38ms/step - loss: 0.2597 - accuracy: 0.9126 - val_loss: 0.3141 - val_accuracy: 0.8998 - lr: 1.0000e-06
Epoch 13/120
...
Epoch 25/120
312/312 [==============================] - 12s 37ms/step - loss: 0.2592 - accuracy: 0.9116 - val_loss: 0.3141 - val_accuracy: 0.8990 - lr: 1.0000e-07
Epoch 26/120
312/312 [==============================] - 12s 38ms/step - loss: 0.2579 - accuracy: 0.9126 - val_loss: 0.3134 - val_accuracy: 0.9002 - lr: 1.0000e-07
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

A figura abaixo ilustra o resultado final com os datasets de teste e de validação, conseguimos observar que o validation accuracy varia entre 88% e 89%

```
from tensorflow import keras

loaded_model = keras.models.load_model('ModelT_transferLearning_featureExtraction_WithoutDataAumentation.h5')

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

[41] ✓ 18m 46.4s Python

```
313/313 [==============================] - 1125s 4s/step - loss: 0.3446 - accuracy: 0.8899
val_acc: 0.8899000287055969
```

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

[42] ✓ 18m 47.0s Python

```
313/313 [==============================] - 1127s 4s/step - loss: 0.3643 - accuracy: 0.8837
val_acc: 0.8837000131607056
```