Trabalho de Programação 1

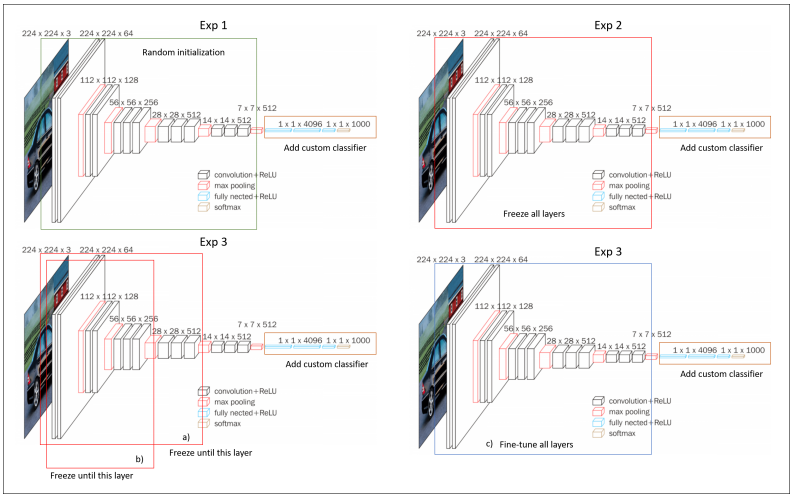
Lívia Meinhardt

September 2, 2022

Contents

1 Introdução

O objetivo deste primeiro trabalho é praticar redes neurais convolucionais, *transfer learning* e usar o VGG16. Para isso, o trabalho consiste na realização de três experimentos, em que o VGG16 é utilizado como *backbone*. Os experimentos são resumidos na imagem abaixo e descritos nas próximas seções.



2 Experimento 1: Treinando do início

Como ilustrado na introdução, o primeiro experimento foi realizado utilizando o VGG6 com inicialização aleatória e substituindo a camada de predição por um classificador.

Neste caso, foi utilizada uma camada com ativação *softmax*. O modelo pode ser resumido no código abaixo (em que a camada densa extra está inclusa):

```
base_model = VGG16(include_top=False, weights=None,input_shape=(224,224,3))
base_model.summary()

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(3,activation="softmax")
dense_1 = tf.keras.layers.Dense(64)
preprocess_input = tf.keras.applications.vgg16.preprocess_input
inputs = tf.keras.Input(shape=(224, 224, 3))

x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x)
x = global_average_layer(x)
x = dense_1(x)

outputs = prediction_layer(x) # Add classification layer
model = tf.keras.Model(inputs, outputs)
```

O modelo foi treinado e testado usando batch size 32, incluindo e excluindo a última camada densa. Os resultados do teste estão explicitados na tabela abaixo.

BATCH SIZE	DENSE	ACCURACY	LOSS
32	VERDADEIRO	82.03%	0,38509
32	FALSO	90.62%	0,301316

Em que a não inclusão da última camada melhora o modelo em aproximadamente 8%, além de diminuir a perda em 0.085 pontos. O resultado é interessante pois é possível entender na prática que a inclusão de uma camada não necessariamente melhora a capacidade de classificação da rede, como a intuição leiga poderia nos levar a acreditar.

3 Experimento 2: Usando uma rede pré-treinada com ImageNet como um *feature extractor*

Neste segundo experimento utilizamos os parâmetros da ImageNet e adaptamos a rede mudando a camada de predição para o classificador com ativação softmax. O modelo:

```
base_model = VGG16(include_top=False, weights='imagenet',input_shape=(224,224,3))
base_model.trainable = False #freeze
base_model.summary()

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(3,activation="softmax")
dense_1 = tf.keras.layers.Dense(64)
preprocess_input = tf.keras.applications.vgg16.preprocess_input
inputs = tf.keras.Input(shape=(224, 224, 3))

x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x)
x = global_average_layer(x)
x = dense_1(x)

outputs = prediction_layer(x) # Add classification layer
model = tf.keras.Model(inputs, outputs)
model.summary()
```

Usando batch size 32 incluindo ou não a última camada densa, os resultados são apresentados na tabela abaixo:

BATCH SIZE	DENSE	ACCURACY	LOSS
32	VERDADEIRO	81.25%	0,410752
32	FALSO	89.84%	0,267284

Dos resultados confirmamos a conclusão anterior sobre a camada densa. Apesar de utilizar os parâmetros da ImageNet e não os treinamos com o próprio dataset de interesse, os resultados são próximos ao do primeiro experimento. O que é bastante interessante, visto que em modelos de machine learning com dados numéricos este tipo de comportamento não seria reproduzido em quase nenhum contexto - e caso fosse, a desconfiança poderia ser muito grande.

Podemos priorizar a minimização da perda e, assim, a rede que não inclui a camada densa deste experimento é a melhor das 4. O que mostra a generalidade do modelo testado com o ImageNet.

4 Experimento 3: *Fine-tuning* as últimas camadas

No experimento 3 congelamos algumas camadas finais. O experimento é repetido em três cenários: congelando a partir do bloco "block5_conv1", a partir do "block4_conv1" e, por fim, congelando todos os blocos convolucionais. Todos as redes são treinadas com o batch size 32. Os resultados dos testes estão resumidos na tabela abaixo

UNFREEZE	ACCURACY	LOSS
block5_conv1	95.31%	0,10658
block4_conv1	93.75%	0,17673
All	93.75%	0,14233

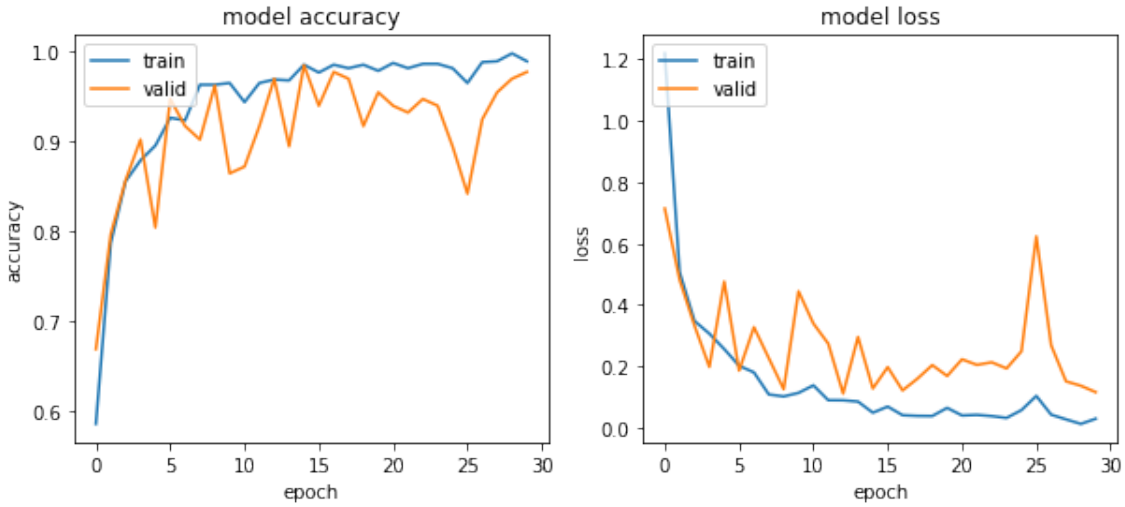
Nesse experimento, considerando ambas as métricas, o melhor modelo é o que descongelamos a partir do "block5_conv1", ou seja, aquele em que treinamos mais camadas com o dataset desejado.

Além disso, vale comentar que, dentre todas as redes treinadas, esses são as com maior acurária e menor perda. Ou seja, a "mistura" dos parâmetros treinamos com ImageNet e nossos dados é a melhor combinação.

Ou seja, a partir dos resultados, podemos entender que - para esse caso - a combinação da generalidade e grande volume de dados do ImageNet para as camadas iniciais e a especificidade dos dados de interesse nas camadas fianis é a mellhor solução para o nosso classificador.

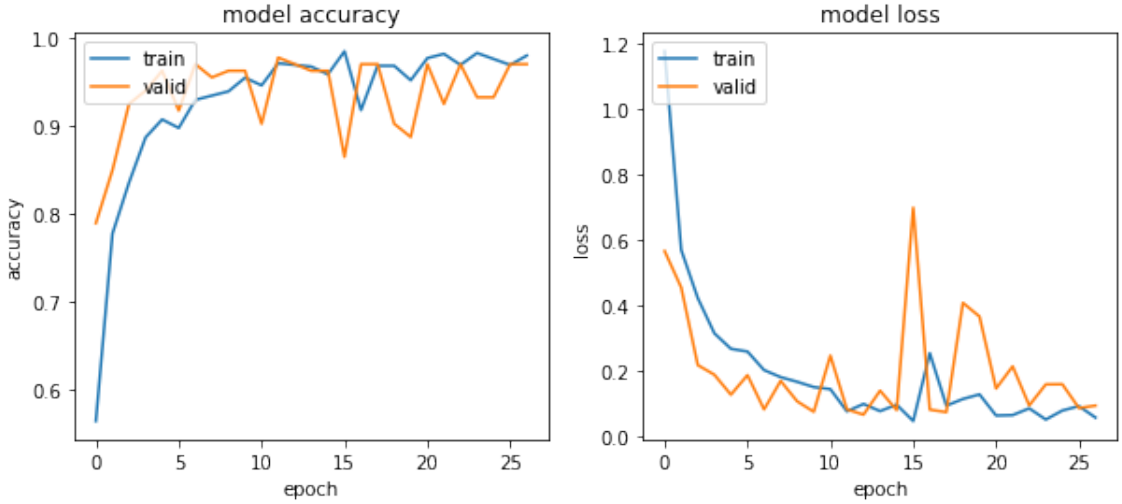
4.1 Descongelando os últimos blocos convolucionais (a partir do "block5_conv1")

Pela tabela acima esse foi o experimento que resultou na melhor rede (mehor acurária e menor perda no teste). A acurária e a *loss* tem uma variância bastante grande na etapa de validação, quando comparada com os dados de treino, porém ao final ambas são próximas.



4.2 Descongelando os últimos blocos convolucionais (a partir de "block4_conv1")

Este é o *pior* dentre os três cenários desse teste. Apesar dos resultados serem satisfatórios (93.75% de acurácia no teste). Apesar da acurária ser exatamente igual ao de quando descongelamos toda a as camadas convolucionais, a perda (loss) é maior. O que é um resultado interessante, visto que quando descongelamos menos camadas (teste anterior) a perda diminui ainda mais.



4.3 Descongelando todos os últimos blocos convolucionais

Como comentado anteriormente, a acurária é exatamente igual ao cenário anterior, mas a perda é um pouco menor. Este também é o cenário em que a variância das métricas na validação é menor, comparado ao treinamento.