This is a companion notebook for the book [Deep Learning with Python, Second Edition](). For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

Downloading the data

Uploading the json file from kaggle to access the data from dogs-vs-cats dataset.

```
from google.colab import files
files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
{'kaggle.json':
```

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle competitions download -c dogs-vs-cats
```

```
401 - Unauthorized
```

```
!unzip -qq dogs-vs-cats
```

```
!unzip -qq test1.zip
```

```
!unzip -qq train.zip
```

1. Consider the Cats & Dogs example. Start initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text). Use any technique to reduce

overfitting and improve performance in developing a network that you train from scratch.

Copying images to training, validation, and test directories

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)
```

```
# Training has 1000 samples, test has 500 samples and validation has 500 samples.

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2000)
```

Data processing

Before being input into the model, the data is converted into preprocessed floating point tensors as the data is in JPEG format, the preprocessing stages are as follows: 1.Read the pictures 2. convert the JPEG content in to RGB grid of pixels 3.convert the RGB grid of pixels in to floating point tensors. 4. Resize them 5. Make them in to batches

Using image_dataset_from_directory to read images

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
```

```
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

```
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

Create a dataset instance from NumPy array of random numbers of 1000 samples and each sample of vector size 16

```
import numpy as np
import tensorflow as tf
random_numbers = np.random.normal(size=(1000, 16))
dataset = tf.data.Dataset.from_tensor_slices(random_numbers)
```

```
for i, element in enumerate(dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(16,)
(16,)
(16,)
```

Batching the data into batches of size 32

```
batched_dataset = dataset.batch(32)
for i, element in enumerate(batched_dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(32, 16)
(32, 16)
(32, 16)
```

```
reshaped_dataset = dataset.map(lambda x: tf.reshape(x, (4, 4)))
for i, element in enumerate(reshaped_dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(4, 4)
(4, 4)
(4, 4)
```

Displaying the shapes of the data and labels yielded by the Dataset

```
for data_batch, labels_batch in train_dataset:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    break
```

```
data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)
```

Building the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Configuring the model for training

```
model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling (Rescaling)       (None, 180, 180, 3)       0
```

```
 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2D  (None, 89, 89, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 256)         590080

 flatten (Flatten)           (None, 12544)             0

 dropout (Dropout)           (None, 12544)             0

 dense (Dense)               (None, 1)                 12545

=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
_________________________________________________________________
```

Fitting the model using a Dataset

```
from keras.callbacks import ModelCheckpoint, EarlyStopping

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

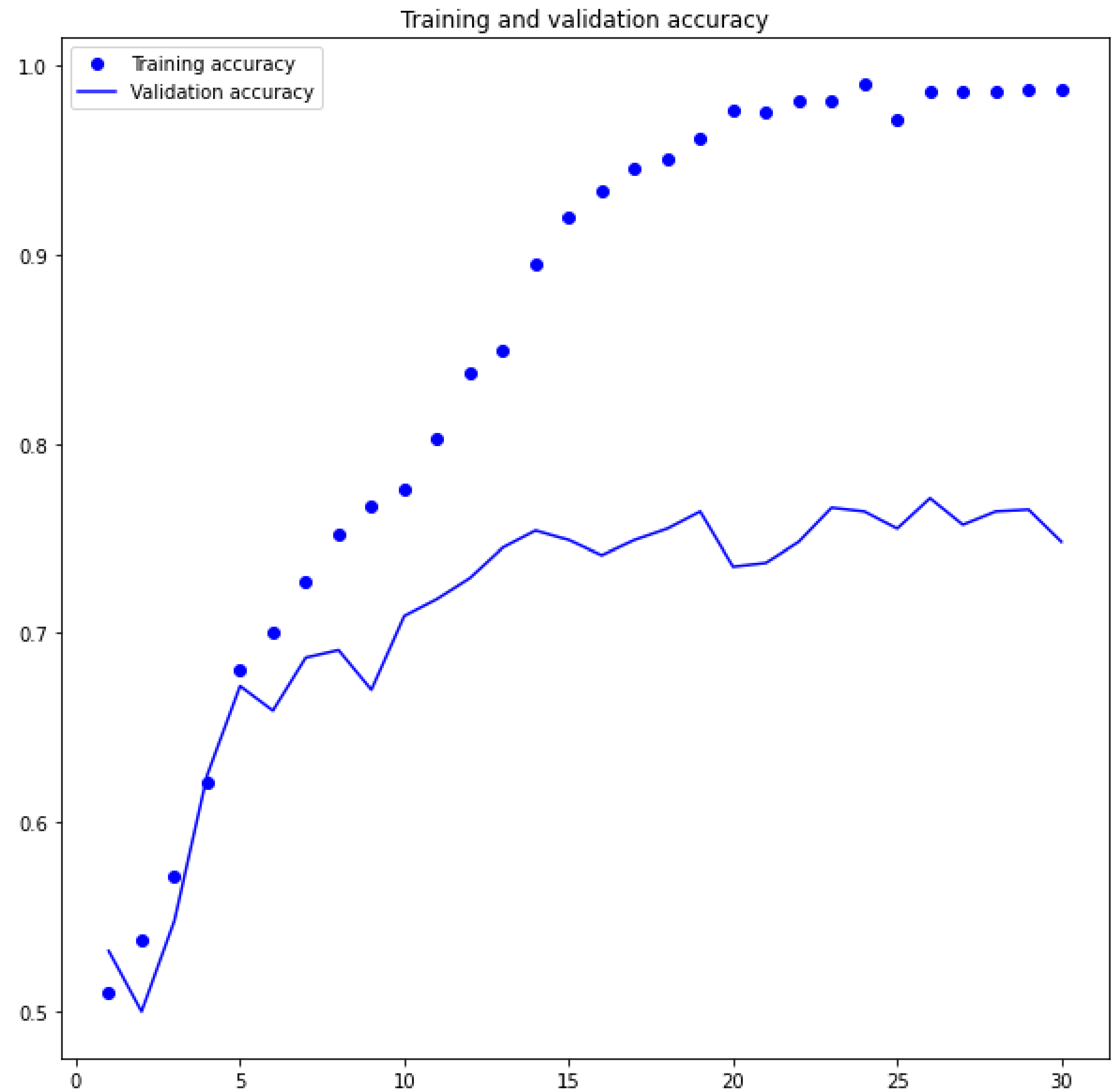
```
Epoch 1/30
63/63 [==============================] - 16s 94ms/step - loss: 0.6926 - accuracy: 0.
```

```
Epoch 2/30
63/63 [==============================] - 6s 88ms/step - loss: 0.6917 - accuracy: 0.5
Epoch 3/30
63/63 [==============================] - 5s 81ms/step - loss: 0.6770 - accuracy: 0.5
Epoch 4/30
63/63 [==============================] - 5s 82ms/step - loss: 0.6465 - accuracy: 0.6
Epoch 5/30
63/63 [==============================] - 5s 83ms/step - loss: 0.6048 - accuracy: 0.6
Epoch 6/30
63/63 [==============================] - 5s 83ms/step - loss: 0.5707 - accuracy: 0.7
Epoch 7/30
63/63 [==============================] - 5s 82ms/step - loss: 0.5440 - accuracy: 0.7
Epoch 8/30
63/63 [==============================] - 5s 81ms/step - loss: 0.5153 - accuracy: 0.7
Epoch 9/30
63/63 [==============================] - 5s 80ms/step - loss: 0.4820 - accuracy: 0.7
Epoch 10/30
63/63 [==============================] - 5s 77ms/step - loss: 0.4633 - accuracy: 0.7
Epoch 11/30
63/63 [==============================] - 5s 79ms/step - loss: 0.4262 - accuracy: 0.8
Epoch 12/30
63/63 [==============================] - 5s 79ms/step - loss: 0.3689 - accuracy: 0.8
Epoch 13/30
63/63 [==============================] - 5s 81ms/step - loss: 0.3270 - accuracy: 0.8
Epoch 14/30
63/63 [==============================] - 5s 82ms/step - loss: 0.2533 - accuracy: 0.8
Epoch 15/30
63/63 [==============================] - 5s 82ms/step - loss: 0.2091 - accuracy: 0.9
Epoch 16/30
63/63 [==============================] - 5s 81ms/step - loss: 0.1790 - accuracy: 0.9
Epoch 17/30
63/63 [==============================] - 5s 81ms/step - loss: 0.1434 - accuracy: 0.9
Epoch 18/30
63/63 [==============================] - 5s 80ms/step - loss: 0.1184 - accuracy: 0.9
Epoch 19/30
63/63 [==============================] - 5s 82ms/step - loss: 0.1014 - accuracy: 0.9
Epoch 20/30
63/63 [==============================] - 5s 82ms/step - loss: 0.0731 - accuracy: 0.9
Epoch 21/30
63/63 [==============================] - 5s 81ms/step - loss: 0.0668 - accuracy: 0.9
Epoch 22/30
63/63 [==============================] - 5s 82ms/step - loss: 0.0574 - accuracy: 0.9
Epoch 23/30
63/63 [==============================] - 5s 81ms/step - loss: 0.0570 - accuracy: 0.9
Epoch 24/30
63/63 [==============================] - 5s 82ms/step - loss: 0.0369 - accuracy: 0.9
Epoch 25/30
63/63 [==============================] - 5s 82ms/step - loss: 0.0833 - accuracy: 0.9
Epoch 26/30
63/63 [==============================] - 5s 81ms/step - loss: 0.0376 - accuracy: 0.9
Epoch 27/30
63/63 [==============================] - 5s 82ms/step - loss: 0.0503 - accuracy: 0.9
Epoch 28/30
63/63 [==============================] - 5s 80ms/step - loss: 0.0366 - accuracy: 0.9
Epoch 29/30
```

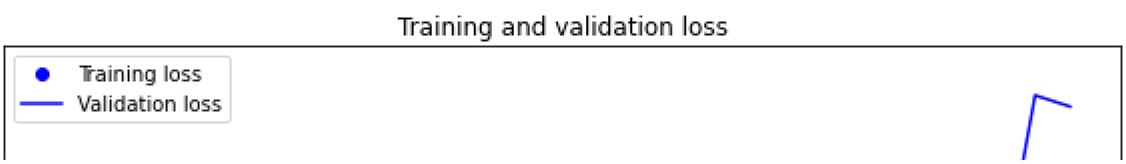
Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.figure(figsize=(10, 10))
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```



<Figure size 432x288 with 0 Axes>



Evaluating the model on the test set

```
test_model = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 40ms/step - loss: 0.5797 - accuracy: 0.7230
Test accuracy: 0.723
```

Observation, from the above result we can conclude: These two graphs are characterized by overfitting as the model is training well on the training set(98.70%) while validation & test accuracy is not improving.

We can use these three techniques namely, 1. Data augmentation, 2. Regularization, 3. Dropout to improve our validation and test accuracy and prevent overfitting.

2. Increase your training sample size. You may pick any amount. Keep the validation and test samples the same as above. Optimize your network (again training from scratch). What performance did you achieve?

Using Data augmentation

What exactly do we do in Data Augmentation-

In the simplest sense, we tend to flip, rotate, scale, crop, translate (moving image along x and y axis), Gaussian Noise (way of distorting high-frequesncy by adding some noise to them).

For our Neural Network, we will only use flipping, rotation and zooming.

```
import os, shutil, pathlib

shutil.rmtree("./cats_vs_dogs_small_Q2", ignore_errors=True)

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_Q2")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)
#Creating training, Test and validation sets.
#Training has 1500 samples, test has 500 samples and validation has 500 samples.
make_subset("train", start_index=0, end_index=1500)
make_subset("validation", start_index=1500, end_index=2000)
make_subset("test", start_index=2000, end_index=2500)
```

Define a data augmentation stage to add to an image model

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
```

```
    ]
)
```

Displaying some randomly augmented training images

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

Here, there are samples of 9 images that have been flipped, zoomed and rotated.

Defining a new convnet that includes image augmentation and dropout

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

Training the regularized convnet

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
63/63 [==============================] - 8s 109ms/step - loss: 0.6953 - accuracy: 0.
Epoch 2/50
63/63 [==============================] - 7s 104ms/step - loss: 0.6921 - accuracy: 0.
Epoch 3/50
63/63 [==============================] - 7s 105ms/step - loss: 0.6885 - accuracy: 0.
Epoch 4/50
63/63 [==============================] - 7s 105ms/step - loss: 0.6871 - accuracy: 0.
Epoch 5/50
63/63 [==============================] - 7s 111ms/step - loss: 0.6724 - accuracy: 0.
Epoch 6/50
63/63 [==============================] - 7s 106ms/step - loss: 0.6538 - accuracy: 0.
Epoch 7/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6707 - accuracy: 0.
```

```
Epoch 8/50
63/63 [==============================] - 7s 103ms/step - loss: 0.6784 - accuracy: 0.
Epoch 9/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6335 - accuracy: 0.
Epoch 10/50
63/63 [==============================] - 7s 104ms/step - loss: 0.6142 - accuracy: 0.
Epoch 11/50
63/63 [==============================] - 7s 105ms/step - loss: 0.6088 - accuracy: 0.
Epoch 12/50
63/63 [==============================] - 7s 105ms/step - loss: 0.6022 - accuracy: 0.
Epoch 13/50
63/63 [==============================] - 7s 106ms/step - loss: 0.5897 - accuracy: 0.
Epoch 14/50
63/63 [==============================] - 7s 104ms/step - loss: 0.5656 - accuracy: 0.
Epoch 15/50
63/63 [==============================] - 7s 105ms/step - loss: 0.5720 - accuracy: 0.
Epoch 16/50
63/63 [==============================] - 7s 105ms/step - loss: 0.5415 - accuracy: 0.
Epoch 17/50
63/63 [==============================] - 7s 105ms/step - loss: 0.5463 - accuracy: 0.
Epoch 18/50
63/63 [==============================] - 7s 105ms/step - loss: 0.5385 - accuracy: 0.
Epoch 19/50
63/63 [==============================] - 7s 107ms/step - loss: 0.5256 - accuracy: 0.
Epoch 20/50
63/63 [==============================] - 7s 107ms/step - loss: 0.4998 - accuracy: 0.
Epoch 21/50
63/63 [==============================] - 7s 107ms/step - loss: 0.4821 - accuracy: 0.
Epoch 22/50
63/63 [==============================] - 7s 106ms/step - loss: 0.4854 - accuracy: 0.
Epoch 23/50
63/63 [==============================] - 7s 105ms/step - loss: 0.4516 - accuracy: 0.
Epoch 24/50
63/63 [==============================] - 7s 104ms/step - loss: 0.4661 - accuracy: 0.
Epoch 25/50
63/63 [==============================] - 7s 104ms/step - loss: 0.4498 - accuracy: 0.
Epoch 26/50
63/63 [==============================] - 7s 106ms/step - loss: 0.4283 - accuracy: 0.
Epoch 27/50
63/63 [==============================] - 7s 104ms/step - loss: 0.4454 - accuracy: 0.
Epoch 28/50
63/63 [==============================] - 7s 108ms/step - loss: 0.4268 - accuracy: 0.
```

Evaluating the model on the test set

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 40ms/step - loss: 0.4579 - accuracy: 0.8140
Test accuracy: 0.814
```

Test Accurcay noted - 81.4% Training Accuracy - 87.55% Validation Accuracy - 83.10%

As we can see that our test accuracy has already improved alot by using data augmentation and dropout and increasing the training sample size. However we do have to train the model for more epochs than usual.

Hence, we can say that by using Data Augmentation, dropout and Regularization we can some what mittigate the effects of Overfitting.

3.Now change your training sample so that you achieve better performance than those from Steps 1 and 2. This sample size may be larger, or smaller than those in the previous steps. The objective is to find the ideal training sample size to get best prediction results.

Increasing the training sample size to 2000 while maintaining the same validation and test sets as before 500 samples

```
original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_Q3")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)
#Creating training, Test and validation sets.
#Training has 2000 samples, test has 500 samples and validation has 500 samples.
make_subset("train", start_index=0, end_index=2000)
make_subset("validation", start_index=2000, end_index=2500)
make_subset("test", start_index=2500, end_index=3000)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-ce544ecec45c> in <module>
----> 1 original_dir = pathlib.Path("train")
      2 new_base_dir = pathlib.Path("cats_vs_dogs_small_Q3")
      3 
      4 def make_subset(subset_name, start_index, end_index):
      5     for category in ("cat", "dog"):

NameError: name 'pathlib' is not defined
```

SEARCH STACK OVERFLOW

Creating a new convnet with more training samples,image enhancement and dropuot

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation1.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
63/63 [==============================] - 8s 105ms/step - loss: 0.6954 - accuracy: 0.
Epoch 2/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6932 - accuracy: 0.
Epoch 3/50
63/63 [==============================] - 7s 104ms/step - loss: 0.6934 - accuracy: 0.
Epoch 4/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6934 - accuracy: 0.
Epoch 5/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6934 - accuracy: 0.
Epoch 6/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6939 - accuracy: 0.
Epoch 7/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6933 - accuracy: 0.
Epoch 8/50
```

```
63/63 [==============================] - 7s 99ms/step - loss: 0.6927 - accuracy: 0.5
Epoch 9/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6938 - accuracy: 0.
Epoch 10/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6933 - accuracy: 0.
Epoch 11/50
63/63 [==============================] - 7s 114ms/step - loss: 0.6935 - accuracy: 0.
Epoch 12/50
63/63 [==============================] - 7s 100ms/step - loss: 0.6934 - accuracy: 0.
Epoch 13/50
63/63 [==============================] - 7s 99ms/step - loss: 0.6931 - accuracy: 0.5
Epoch 14/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6933 - accuracy: 0.
Epoch 15/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6934 - accuracy: 0.
Epoch 16/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6933 - accuracy: 0.
Epoch 17/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6934 - accuracy: 0.
Epoch 18/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6932 - accuracy: 0.
Epoch 19/50
63/63 [==============================] - 7s 100ms/step - loss: 0.6932 - accuracy: 0.
Epoch 20/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6933 - accuracy: 0.
Epoch 21/50
63/63 [==============================] - 7s 101ms/step - loss: 0.7036 - accuracy: 0.
Epoch 22/50
63/63 [==============================] - 7s 100ms/step - loss: 0.6932 - accuracy: 0.
Epoch 23/50
63/63 [==============================] - 6s 99ms/step - loss: 0.6939 - accuracy: 0.4
Epoch 24/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6934 - accuracy: 0.
Epoch 25/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6931 - accuracy: 0.
Epoch 26/50
63/63 [==============================] - 7s 100ms/step - loss: 0.6931 - accuracy: 0.
Epoch 27/50
63/63 [==============================] - 7s 104ms/step - loss: 0.6934 - accuracy: 0.
Epoch 28/50
63/63 [==============================] - 7s 106ms/step - loss: 0.6933 - accuracy: 0.
```

```python
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation1.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 39ms/step - loss: 0.6922 - accuracy: 0.5110
Test accuracy: 0.511
```

I began with the training a sample convnet on the 1,000 training samples with out any optimization to which resulted in the Test accuracy was around 72.30% and the Overfitting was recognized as the main problem. After applying data augmentation and other optimization strategiesdrop out

After that i looked for the best training sample to improve accuracy. The best approach to avoid overfitting have been discovered by manipulating the training sample and using the optimization techniques

1.Getting more training samples not always practical to expand the training sample. Our test accuracy has reduced by increasing training sample.

2.Reducing the capacity of the work:Overfitting is significantly reduced when the model's size is reduced, i.e. the number of learnable parameters in the model, which is effectively the number of layers and the number of units in layers.

3.Adding weight regularization:Limiting the complexity of a network by restricting the weights to accept only tiny values, which helps to regularize the distribution of the weight values and so prevents or minimizes overfitting.

4.Adding dropout- Overfitting is reduced by zeroing out a number of the layer's output characteristics during training. The percentage of features that are zeroed out is known as the dropout rate

4.Repeat Steps 1-3, but now using a pretrained network. The sample sizes you use in Steps 2 and 3 for the pretrained network may be the same or different from those using the network where you trained from scratch. Again, use any and all optimization techniques to get best performance.

** Using a pretrained model to apply deep learning to tiny image datasets is a highly effective method. A pretrained model is one that has been trained earlier on a big dataset, usually for a large-scale image classification problem.**

We will use a big convnet trained on the ImageNet dataset in this scenario (1.4 million labeled images and 1,000 different classes). We will use the VGG16 architecture, although there are a variety of other architectures to choose from, including VGG, ResNet, Inception, Xception, and so on.

Feature Extraction with a pretrained model

Feature extraction is the process of extracting important features from new samples using the representations acquired by a previously trained model (in our instance, ImageNet). These characteristics are then fed into a new classifier that has been trained from the ground up.

Instantiating the VGG16 convolutional base

```
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg1
58889256/58889256 [==============================] - 0s 0us/step
```

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 180, 180, 3)]     0

 block1_conv1 (Conv2D)       (None, 180, 180, 64)      1792

 block1_conv2 (Conv2D)       (None, 180, 180, 64)      36928

 block1_pool (MaxPooling2D)  (None, 90, 90, 64)        0

 block2_conv1 (Conv2D)       (None, 90, 90, 128)       73856

 block2_conv2 (Conv2D)       (None, 90, 90, 128)       147584

 block2_pool (MaxPooling2D)  (None, 45, 45, 128)       0

 block3_conv1 (Conv2D)       (None, 45, 45, 256)       295168

 block3_conv2 (Conv2D)       (None, 45, 45, 256)       590080

 block3_conv3 (Conv2D)       (None, 45, 45, 256)       590080

 block3_pool (MaxPooling2D)  (None, 22, 22, 256)       0

 block4_conv1 (Conv2D)       (None, 22, 22, 512)       1180160

 block4_conv2 (Conv2D)       (None, 22, 22, 512)       2359808

 block4_conv3 (Conv2D)       (None, 22, 22, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 11, 11, 512)       0

 block5_conv1 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_conv2 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_conv3 (Conv2D)       (None, 11, 11, 512)       2359808
```

```
 block5_pool (MaxPooling2D)  (None, 5, 5, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

```
import os, shutil, pathlib
from tensorflow.keras.utils import image_dataset_from_directory

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2500)

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

Feature extraction without data augmentation using a pretrained model

Extracting the VGG16 features and corresponding labels

```
import numpy as np

def get_features_and_labels(dataset):
    all_features = []
```

```
        all_labels = []
        for images, labels in dataset:
            preprocessed_images = keras.applications.vgg16.preprocess_input(images)
            features = conv_base.predict(preprocessed_images)
            all_features.append(features)
            all_labels.append(labels)
        return np.concatenate(all_features), np.concatenate(all_labels)

    train_features, train_labels =  get_features_and_labels(train_dataset)
    val_features, val_labels =  get_features_and_labels(validation_dataset)
    test_features, test_labels =  get_features_and_labels(test_dataset)
```

```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
```

```
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
```

```
train_features.shape
```

```
(2000, 5, 5, 512)
```

Defining and training the densely connected classifier

```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint(
      filepath="feature_extraction.keras",
      save_best_only=True,
      monitor="val_loss")
]
history = model.fit(
    train_features, train_labels,
    epochs=20,
    validation_data=(val_features, val_labels),
    callbacks=callbacks)
```

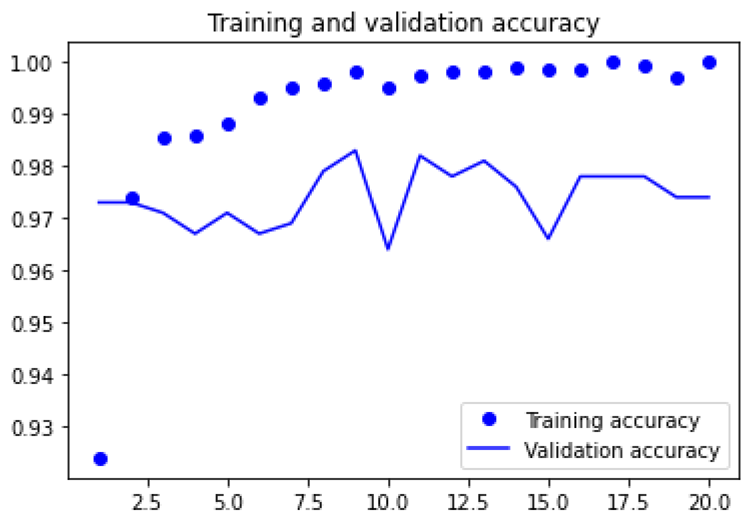
```
Epoch 1/20
63/63 [==============================] - 1s 12ms/step - loss: 19.2034 - accuracy: 0.924
Epoch 2/20
63/63 [==============================] - 0s 8ms/step - loss: 4.7623 - accuracy: 0.9740
Epoch 3/20
63/63 [==============================] - 1s 9ms/step - loss: 1.6389 - accuracy: 0.9855
Epoch 4/20
```

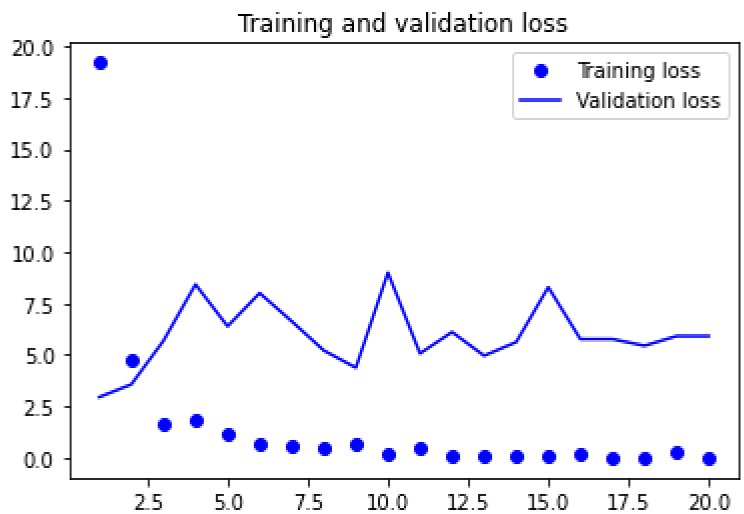
```
63/63 [==============================] - 0s 8ms/step - loss: 1.8649 - accuracy: 0.9860
Epoch 5/20
63/63 [==============================] - 1s 9ms/step - loss: 1.1956 - accuracy: 0.9880
Epoch 6/20
63/63 [==============================] - 1s 9ms/step - loss: 0.6759 - accuracy: 0.9930
Epoch 7/20
63/63 [==============================] - 1s 9ms/step - loss: 0.6015 - accuracy: 0.9950
Epoch 8/20
63/63 [==============================] - 1s 8ms/step - loss: 0.4297 - accuracy: 0.9960
Epoch 9/20
63/63 [==============================] - 1s 9ms/step - loss: 0.6604 - accuracy: 0.9980
Epoch 10/20
63/63 [==============================] - 0s 8ms/step - loss: 0.2026 - accuracy: 0.9950
Epoch 11/20
63/63 [==============================] - 1s 8ms/step - loss: 0.4857 - accuracy: 0.9975
Epoch 12/20
63/63 [==============================] - 1s 9ms/step - loss: 0.0949 - accuracy: 0.9980
Epoch 13/20
63/63 [==============================] - 1s 9ms/step - loss: 0.1101 - accuracy: 0.9980
Epoch 14/20
63/63 [==============================] - 1s 9ms/step - loss: 0.0991 - accuracy: 0.9990
Epoch 15/20
63/63 [==============================] - 1s 8ms/step - loss: 0.0836 - accuracy: 0.9985
Epoch 16/20
63/63 [==============================] - 1s 8ms/step - loss: 0.1965 - accuracy: 0.9985
Epoch 17/20
63/63 [==============================] - 1s 8ms/step - loss: 4.2965e-36 - accuracy: 1.0
Epoch 18/20
63/63 [==============================] - 1s 8ms/step - loss: 0.0070 - accuracy: 0.9995
Epoch 19/20
63/63 [==============================] - 1s 8ms/step - loss: 0.2455 - accuracy: 0.9970
Epoch 20/20
63/63 [==============================] - 1s 9ms/step - loss: 0.0000e+00 - accuracy: 1.0
```

Plotting the results

```
import matplotlib.pyplot as plt
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
```

```
plt.show()
```





Features extraction with data agumentation using a pretrained model

Instantiating and freezing the VGG16 convolutional base

```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)
conv_base.trainable = False
```

Printing the list of trainable weights before and after freezing

```
conv_base.trainable = True
print("This is the number of trainable weights "
      "before freezing the conv base:", len(conv_base.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 26
```

```
conv_base.trainable = False
print("This is the number of trainable weights "
      "after freezing the conv base:", len(conv_base.trainable_weights))
```

```
This is the number of trainable weights after freezing the conv base: 0
```

Adding a data augmentation stage and a classifier to the convolutional base

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction_with_data_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
63/63 [==============================] - 15s 201ms/step - loss: 19.3874 - accuracy:
Epoch 2/50
63/63 [==============================] - 13s 199ms/step - loss: 7.5256 - accuracy: 0
Epoch 3/50
63/63 [==============================] - 14s 216ms/step - loss: 7.1699 - accuracy: 0
```

```
Epoch 4/50
63/63 [==============================] - 13s 203ms/step - loss: 5.9416 - accuracy: 0
Epoch 5/50
63/63 [==============================] - 13s 201ms/step - loss: 2.9664 - accuracy: 0
Epoch 6/50
63/63 [==============================] - 13s 202ms/step - loss: 3.2532 - accuracy: 0
Epoch 7/50
63/63 [==============================] - 13s 201ms/step - loss: 3.4705 - accuracy: 0
Epoch 8/50
63/63 [==============================] - 13s 206ms/step - loss: 3.4632 - accuracy: 0
Epoch 9/50
63/63 [==============================] - 13s 207ms/step - loss: 2.5165 - accuracy: 0
Epoch 10/50
63/63 [==============================] - 13s 200ms/step - loss: 2.1241 - accuracy: 0
Epoch 11/50
63/63 [==============================] - 13s 205ms/step - loss: 1.9573 - accuracy: 0
Epoch 12/50
63/63 [==============================] - 13s 201ms/step - loss: 1.9619 - accuracy: 0
Epoch 13/50
63/63 [==============================] - 13s 199ms/step - loss: 2.2476 - accuracy: 0
Epoch 14/50
63/63 [==============================] - 13s 201ms/step - loss: 2.0001 - accuracy: 0
Epoch 15/50
63/63 [==============================] - 14s 225ms/step - loss: 2.1588 - accuracy: 0
Epoch 16/50
63/63 [==============================] - 13s 197ms/step - loss: 1.0806 - accuracy: 0
Epoch 17/50
63/63 [==============================] - 13s 202ms/step - loss: 1.4148 - accuracy: 0
Epoch 18/50
63/63 [==============================] - 13s 201ms/step - loss: 1.9583 - accuracy: 0
Epoch 19/50
63/63 [==============================] - 13s 206ms/step - loss: 1.2177 - accuracy: 0
Epoch 20/50
63/63 [==============================] - 13s 203ms/step - loss: 1.1163 - accuracy: 0
Epoch 21/50
63/63 [==============================] - 13s 207ms/step - loss: 1.0462 - accuracy: 0
Epoch 22/50
63/63 [==============================] - 13s 202ms/step - loss: 1.1273 - accuracy: 0
Epoch 23/50
63/63 [==============================] - 13s 203ms/step - loss: 0.9226 - accuracy: 0
Epoch 24/50
63/63 [==============================] - 13s 204ms/step - loss: 1.2809 - accuracy: 0
Epoch 25/50
63/63 [==============================] - 13s 203ms/step - loss: 1.1635 - accuracy: 0
Epoch 26/50
63/63 [==============================] - 13s 203ms/step - loss: 0.7301 - accuracy: 0
Epoch 27/50
63/63 [==============================] - 13s 200ms/step - loss: 0.5739 - accuracy: 0
Epoch 28/50
63/63 [==============================] - 13s 201ms/step - loss: 0.7982 - accuracy: 0
```

Evaluating the model on the test set

```
test_model = keras.models.load_model(
    "feature_extraction_with_data_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 4s 112ms/step - loss: 1.6130 - accuracy: 0.980
Test accuracy: 0.980
```

A pretrained VGG16 model with Fine-tuning

Fine-tuning involves unfreezing a few of the top layers of a frozen model base used for feature extraction and training both the newly added element of the model (in this case, the fully connected classifier) and these top layers at the same time. Fine-tuning is the process of slightly adjusting the more abstract representations of the model that are being reused to make them more relevant to the task at hand.

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808
```

```
 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

Freezing all layers until the fourth from the last

```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

Fine-tuning the model

```
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/30
63/63 [==============================] - 19s 245ms/step - loss: 0.5018 - accuracy: 0
Epoch 2/30
63/63 [==============================] - 14s 221ms/step - loss: 0.4305 - accuracy: 0
Epoch 3/30
63/63 [==============================] - 14s 216ms/step - loss: 0.2245 - accuracy: 0
Epoch 4/30
63/63 [==============================] - 14s 217ms/step - loss: 0.2367 - accuracy: 0
Epoch 5/30
```

```
63/63 [==============================] - 14s 223ms/step - loss: 0.0721 - accuracy: 0
Epoch 6/30
63/63 [==============================] - 14s 221ms/step - loss: 0.3203 - accuracy: 0
Epoch 7/30
63/63 [==============================] - 14s 223ms/step - loss: 0.1740 - accuracy: 0
Epoch 8/30
63/63 [==============================] - 14s 222ms/step - loss: 0.1991 - accuracy: 0
Epoch 9/30
63/63 [==============================] - 14s 221ms/step - loss: 0.1139 - accuracy: 0
Epoch 10/30
63/63 [==============================] - 14s 222ms/step - loss: 0.1253 - accuracy: 0
Epoch 11/30
63/63 [==============================] - 14s 224ms/step - loss: 0.1533 - accuracy: 0
Epoch 12/30
63/63 [==============================] - 14s 215ms/step - loss: 0.1304 - accuracy: 0
Epoch 13/30
63/63 [==============================] - 14s 218ms/step - loss: 0.0790 - accuracy: 0
Epoch 14/30
63/63 [==============================] - 14s 214ms/step - loss: 0.2995 - accuracy: 0
Epoch 15/30
63/63 [==============================] - 14s 217ms/step - loss: 0.1001 - accuracy: 0
Epoch 16/30
63/63 [==============================] - 14s 221ms/step - loss: 0.0482 - accuracy: 0
Epoch 17/30
63/63 [==============================] - 14s 222ms/step - loss: 0.0622 - accuracy: 0
Epoch 18/30
63/63 [==============================] - 14s 219ms/step - loss: 0.1573 - accuracy: 0
Epoch 19/30
63/63 [==============================] - 14s 222ms/step - loss: 0.1471 - accuracy: 0
Epoch 20/30
63/63 [==============================] - 14s 223ms/step - loss: 0.0438 - accuracy: 0
Epoch 21/30
63/63 [==============================] - 14s 224ms/step - loss: 0.1099 - accuracy: 0
Epoch 22/30
63/63 [==============================] - 14s 222ms/step - loss: 0.1876 - accuracy: 0
Epoch 23/30
63/63 [==============================] - 14s 222ms/step - loss: 0.0860 - accuracy: 0
Epoch 24/30
63/63 [==============================] - 14s 224ms/step - loss: 0.0863 - accuracy: 0
Epoch 25/30
63/63 [==============================] - 14s 221ms/step - loss: 0.0837 - accuracy: 0
Epoch 26/30
63/63 [==============================] - 14s 222ms/step - loss: 0.0806 - accuracy: 0
Epoch 27/30
63/63 [==============================] - 14s 218ms/step - loss: 0.0557 - accuracy: 0
Epoch 28/30
63/63 [==============================] - 14s 218ms/step - loss: 0.1279 - accuracy: 0
Epoch 29/30
```

```
model = keras.models.load_model("fine_tuning.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 5s 116ms/step - loss: 1.2238 - accuracy: 0.979
```

```
Test accuracy: 0.979
```

```
Here we can clearly see the changes in validat
previous model. It's almost 0.98, which is clo
accuracy of 0.9787. So we can conclude that it
overfitting.
In order to fine-tune a pre-trained model, we 
learning rate of the model. The learning rate 
local minimums and maximums. Large adjustments
learning rate will have drastic impacts on the
```

Here we can clearly see the changes in validation accuracy from the previous model. It's almost 0.98, which is close to the testing accuracy of 0.9787. So we can conclude that it's not overfitting. In order to fine-tune a pre-trained model, we need to adjust the learning rate of the model. The learning rate has an impact on finding local minimums and maximums. Large adjustments to the learning rate will have drastic impacts on the model.