aml-assignment2

October 24, 2023

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.

0.1 Introduction to convnets

Instantiating a small convnet

[ ]:
```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
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.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Displaying the model's summary

[ ]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )
```

```
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 10)                11530

=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
_________________________________________________________________
```

Training the convnet on MNIST images

```
[ ]: from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255
model.compile(optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
Epoch 1/5
938/938 [==============================] - 15s 5ms/step - loss: 0.1568 -
accuracy: 0.9513
Epoch 2/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0435 -
accuracy: 0.9865
Epoch 3/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0300 -
accuracy: 0.9906
Epoch 4/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0230 -
accuracy: 0.9933
Epoch 5/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0179 -
```

```
accuracy: 0.9947
```

[ ]:

```
<keras.callbacks.History at 0x7f953222afd0>
```

Evaluating the convnet

[ ]:

```python
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc:.3f}")
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0300 -
accuracy: 0.9910
Test accuracy: 0.991
```

0.1.1 The convolution operation

Understanding border effects and padding

Understanding convolution strides

0.1.2 The max-pooling operation

An incorrectly structured convnet missing its max-pooling layers

[ ]:

```python
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model_no_max_pool = keras.Model(inputs=inputs, outputs=outputs)
```

[ ]:

```python
model_no_max_pool.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d_3 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_4 (Conv2D)           (None, 24, 24, 64)        18496

 conv2d_5 (Conv2D)           (None, 22, 22, 128)       73856

 flatten_1 (Flatten)         (None, 61952)             0

 dense_1 (Dense)             (None, 10)                619530
```

```
=================================================================
Total params: 712,202
Trainable params: 712,202
Non-trainable params: 0
_________________________________________________________________
```

0.2 Training a convnet from scratch on a small dataset

0.2.1 The relevance of deep learning for small-data problems

0.2.2 Downloading the data

[ ]:
```python
from google.colab import files
files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle.json
```

[ ]:
```
{'kaggle.json':
 b'{"username":"svootkur","key":"6e82607283e5fea59ca1d17df8003a89"}'}
```

[ ]:
```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
!kaggle competitions download -c dogs-vs-cats
```

```
Downloading dogs-vs-cats.zip to /content
100% 812M/812M [00:20<00:00, 42.1MB/s]
100% 812M/812M [00:20<00:00, 42.4MB/s]
```

[ ]:
```
!unzip -qq dogs-vs-cats.zip
```

[ ]:
```
!unzip -qq train.zip
```

[ ]:
```
!unzip -qq test1.zip
```

Copying images to training, validation, and test directories

[ ]:
```python
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)

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)
```

0.2.3 Building the model

Instantiating a small convnet for dogs vs. cats classification

[ ]:

```
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)
```

[ ]:

```
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 180, 180, 3)]     0

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

 conv2d_6 (Conv2D)           (None, 178, 178, 32)      896
```

```
 max_pooling2d_2 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_7 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_8 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_9 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_5 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_10 (Conv2D)          (None, 7, 7, 256)         590080

 flatten_2 (Flatten)         (None, 12544)             0

 dropout (Dropout)           (None, 12544)             0

 dense_2 (Dense)             (None, 1)                 12545

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

Configuring the model for training

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

0.2.4 Data preprocessing

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.
```

[ ]:

```
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,)
```

[ ]:

```
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,)
```

Fitting the model using a `Dataset`

```
[ ]: 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 [==============================] - 8s 84ms/step - loss: 0.7599 - accuracy:
0.5240 - val_loss: 0.6936 - val_accuracy: 0.5000
Epoch 2/30
63/63 [==============================] - 5s 80ms/step - loss: 0.7250 - accuracy:
0.5775 - val_loss: 0.6379 - val_accuracy: 0.6300
Epoch 3/30
63/63 [==============================] - 5s 72ms/step - loss: 0.6677 - accuracy:
0.6035 - val_loss: 0.6441 - val_accuracy: 0.6240
Epoch 4/30
63/63 [==============================] - 5s 73ms/step - loss: 0.6187 - accuracy:
0.6625 - val_loss: 0.6331 - val_accuracy: 0.6340
Epoch 5/30
63/63 [==============================] - 5s 72ms/step - loss: 0.5872 - accuracy:
0.7005 - val_loss: 0.6783 - val_accuracy: 0.6210
Epoch 6/30
63/63 [==============================] - 5s 72ms/step - loss: 0.5658 - accuracy:
0.7005 - val_loss: 0.5834 - val_accuracy: 0.6910
Epoch 7/30
63/63 [==============================] - 5s 73ms/step - loss: 0.5199 - accuracy:
0.7530 - val_loss: 0.5543 - val_accuracy: 0.7360
Epoch 8/30
63/63 [==============================] - 5s 74ms/step - loss: 0.4714 - accuracy:
0.7795 - val_loss: 0.5843 - val_accuracy: 0.6900
Epoch 9/30
63/63 [==============================] - 5s 83ms/step - loss: 0.4364 - accuracy:
```

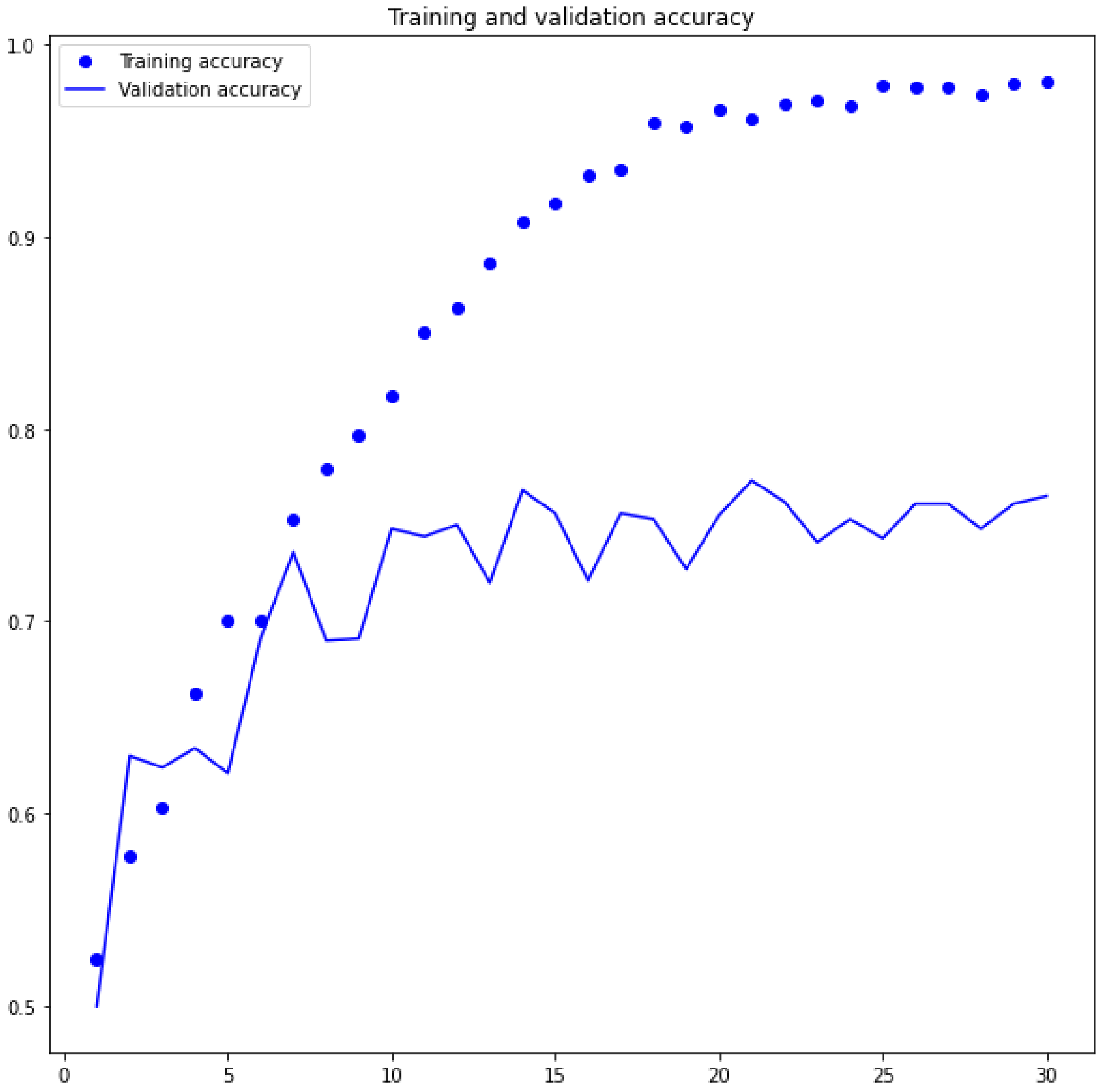
```
0.7970 - val_loss: 0.5950 - val_accuracy: 0.6910
Epoch 10/30
63/63 [==============================] - 6s 97ms/step - loss: 0.3896 - accuracy:
0.8175 - val_loss: 0.5491 - val_accuracy: 0.7480
Epoch 11/30
63/63 [==============================] - 5s 74ms/step - loss: 0.3570 - accuracy:
0.8500 - val_loss: 0.6349 - val_accuracy: 0.7440
Epoch 12/30
63/63 [==============================] - 5s 75ms/step - loss: 0.3170 - accuracy:
0.8630 - val_loss: 0.6339 - val_accuracy: 0.7500
Epoch 13/30
63/63 [==============================] - 5s 72ms/step - loss: 0.2905 - accuracy:
0.8865 - val_loss: 0.7233 - val_accuracy: 0.7200
Epoch 14/30
63/63 [==============================] - 5s 72ms/step - loss: 0.2217 - accuracy:
0.9075 - val_loss: 0.8079 - val_accuracy: 0.7680
Epoch 15/30
63/63 [==============================] - 5s 71ms/step - loss: 0.2034 - accuracy:
0.9175 - val_loss: 0.7765 - val_accuracy: 0.7560
Epoch 16/30
63/63 [==============================] - 5s 72ms/step - loss: 0.1801 - accuracy:
0.9315 - val_loss: 0.9353 - val_accuracy: 0.7210
Epoch 17/30
63/63 [==============================] - 5s 72ms/step - loss: 0.1512 - accuracy:
0.9345 - val_loss: 0.7709 - val_accuracy: 0.7560
Epoch 18/30
63/63 [==============================] - 5s 74ms/step - loss: 0.1095 - accuracy:
0.9590 - val_loss: 1.1703 - val_accuracy: 0.7530
Epoch 19/30
63/63 [==============================] - 5s 72ms/step - loss: 0.1139 - accuracy:
0.9575 - val_loss: 1.1641 - val_accuracy: 0.7270
Epoch 20/30
63/63 [==============================] - 5s 71ms/step - loss: 0.0948 - accuracy:
0.9655 - val_loss: 1.1235 - val_accuracy: 0.7550
Epoch 21/30
63/63 [==============================] - 5s 73ms/step - loss: 0.0995 - accuracy:
0.9610 - val_loss: 1.1120 - val_accuracy: 0.7730
Epoch 22/30
63/63 [==============================] - 6s 86ms/step - loss: 0.0814 - accuracy:
0.9685 - val_loss: 1.1004 - val_accuracy: 0.7620
Epoch 23/30
63/63 [==============================] - 5s 72ms/step - loss: 0.0808 - accuracy:
0.9705 - val_loss: 1.3163 - val_accuracy: 0.7410
Epoch 24/30
63/63 [==============================] - 5s 75ms/step - loss: 0.0817 - accuracy:
0.9675 - val_loss: 1.3291 - val_accuracy: 0.7530
Epoch 25/30
63/63 [==============================] - 5s 73ms/step - loss: 0.0609 - accuracy:
```

```
0.9780 - val_loss: 1.2666 - val_accuracy: 0.7430
Epoch 26/30
63/63 [==============================] - 5s 72ms/step - loss: 0.0602 - accuracy:
0.9775 - val_loss: 1.2825 - val_accuracy: 0.7610
Epoch 27/30
63/63 [==============================] - 5s 74ms/step - loss: 0.0580 - accuracy:
0.9775 - val_loss: 1.5052 - val_accuracy: 0.7610
Epoch 28/30
63/63 [==============================] - 5s 72ms/step - loss: 0.0710 - accuracy:
0.9735 - val_loss: 1.6293 - val_accuracy: 0.7480
Epoch 29/30
63/63 [==============================] - 5s 73ms/step - loss: 0.0536 - accuracy:
0.9790 - val_loss: 1.3950 - val_accuracy: 0.7610
Epoch 30/30
63/63 [==============================] - 5s 73ms/step - loss: 0.0586 - accuracy:
0.9805 - val_loss: 1.5287 - val_accuracy: 0.7650
```

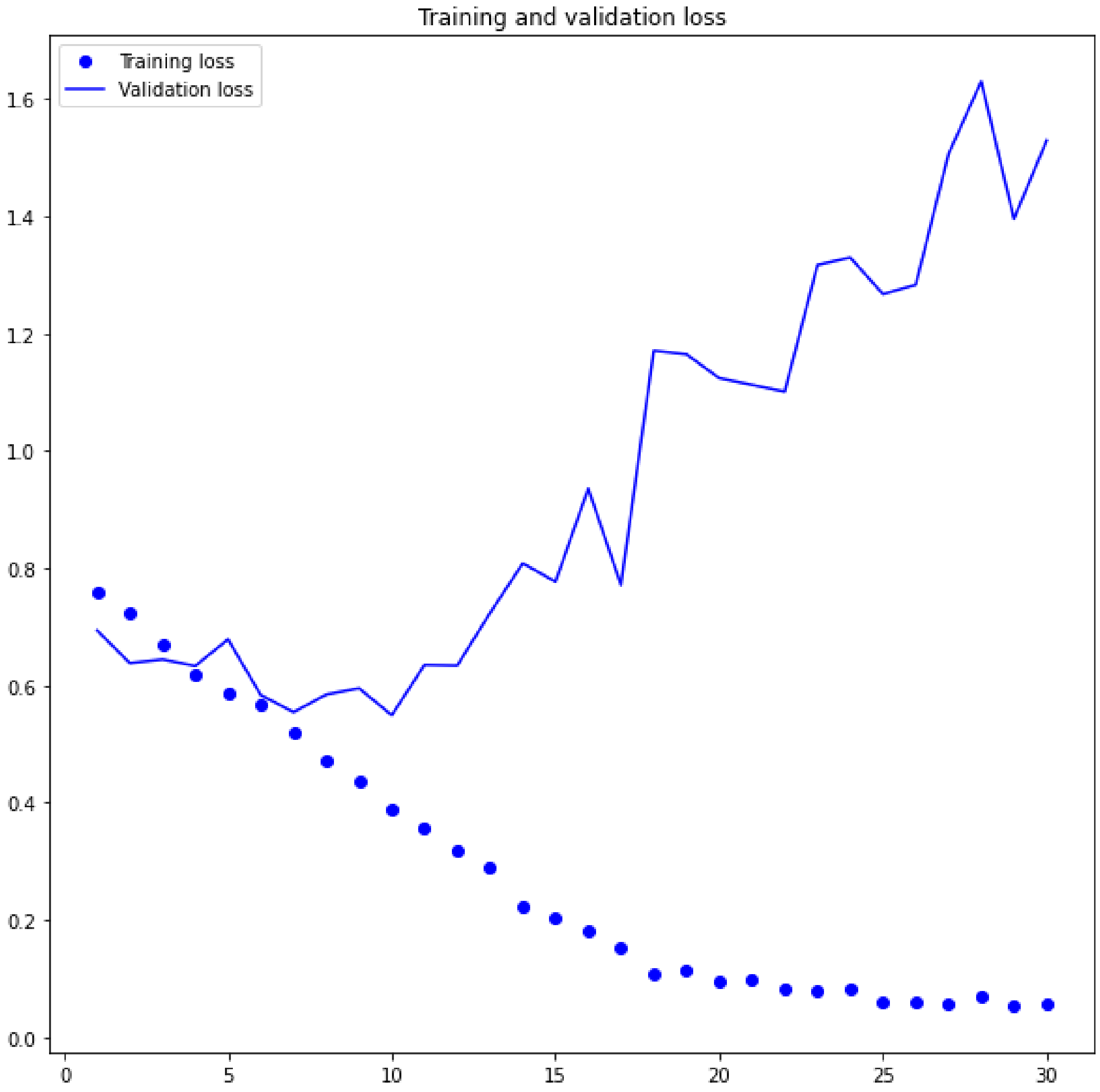
Displaying curves of loss and accuracy during training

[ ]:

```python
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()
```


Training and validation accuracy
Training accuracy
Validation accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30

```
<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 36ms/step - loss: 0.6273 - accuracy:
0.7370
Test accuracy: 0.737
```

0.2.5 Using data augmentation

Define a data augmentation stage to add to an image model

```
[ ]: 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)
#Here I have increased training sample size to 1500 and keeping the validation␣
 ↪and test sample size to 500 each as before
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)
```

```
[ ]: 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")
```

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="rmsprop",
              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=20,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/20
63/63 [==============================] - 9s 104ms/step - loss: 0.7167 -
accuracy: 0.5005 - val_loss: 0.6920 - val_accuracy: 0.5000
Epoch 2/20
63/63 [==============================] - 8s 121ms/step - loss: 0.6985 -
accuracy: 0.5330 - val_loss: 0.6878 - val_accuracy: 0.5140
Epoch 3/20
63/63 [==============================] - 6s 96ms/step - loss: 0.6831 - accuracy:
0.5685 - val_loss: 0.6331 - val_accuracy: 0.6710
Epoch 4/20
63/63 [==============================] - 6s 97ms/step - loss: 0.6557 - accuracy:
0.6185 - val_loss: 0.6406 - val_accuracy: 0.6720
Epoch 5/20
63/63 [==============================] - 6s 94ms/step - loss: 0.6488 - accuracy:
0.6310 - val_loss: 0.6849 - val_accuracy: 0.5960
Epoch 6/20
63/63 [==============================] - 6s 94ms/step - loss: 0.6381 - accuracy:
0.6375 - val_loss: 0.6702 - val_accuracy: 0.6500
Epoch 7/20
63/63 [==============================] - 6s 95ms/step - loss: 0.6226 - accuracy:
0.6625 - val_loss: 0.6186 - val_accuracy: 0.6460
Epoch 8/20
63/63 [==============================] - 7s 106ms/step - loss: 0.6106 -
accuracy: 0.6710 - val_loss: 0.6010 - val_accuracy: 0.6780
Epoch 9/20
```

```
63/63 [==============================] - 9s 130ms/step - loss: 0.6119 -
accuracy: 0.6815 - val_loss: 0.6627 - val_accuracy: 0.6260
Epoch 10/20
63/63 [==============================] - 7s 95ms/step - loss: 0.5890 - accuracy:
0.6710 - val_loss: 0.6733 - val_accuracy: 0.6180
Epoch 11/20
63/63 [==============================] - 6s 95ms/step - loss: 0.5766 - accuracy:
0.7045 - val_loss: 0.6360 - val_accuracy: 0.6780
Epoch 12/20
63/63 [==============================] - 6s 98ms/step - loss: 0.5804 - accuracy:
0.7050 - val_loss: 0.8437 - val_accuracy: 0.5900
Epoch 13/20
63/63 [==============================] - 6s 98ms/step - loss: 0.5657 - accuracy:
0.6990 - val_loss: 0.5659 - val_accuracy: 0.7280
Epoch 14/20
63/63 [==============================] - 6s 97ms/step - loss: 0.5433 - accuracy:
0.7350 - val_loss: 0.6211 - val_accuracy: 0.6880
Epoch 15/20
63/63 [==============================] - 6s 95ms/step - loss: 0.5473 - accuracy:
0.7370 - val_loss: 0.5824 - val_accuracy: 0.7260
Epoch 16/20
63/63 [==============================] - 6s 96ms/step - loss: 0.5345 - accuracy:
0.7370 - val_loss: 0.5307 - val_accuracy: 0.7290
Epoch 17/20
63/63 [==============================] - 6s 97ms/step - loss: 0.5229 - accuracy:
0.7350 - val_loss: 0.5094 - val_accuracy: 0.7530
Epoch 18/20
63/63 [==============================] - 8s 117ms/step - loss: 0.5186 -
accuracy: 0.7410 - val_loss: 0.4811 - val_accuracy: 0.7550
Epoch 19/20
63/63 [==============================] - 6s 95ms/step - loss: 0.4854 - accuracy:
0.7645 - val_loss: 0.6471 - val_accuracy: 0.7120
Epoch 20/20
63/63 [==============================] - 6s 96ms/step - loss: 0.4938 - accuracy:
0.7720 - val_loss: 0.5346 - val_accuracy: 0.7580
```

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 36ms/step - loss: 0.5350 - accuracy:
0.7410
Test accuracy: 0.741
```

[ ]:

```
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)
#Here I have increased training sample size to 1500 and keeping the validation␣
 ↪and test sample size to 500 each as before
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=3000)
```

[ ]:

```
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=20,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/20
63/63 [==============================] - 8s 98ms/step - loss: 0.6939 - accuracy:
0.5080 - val_loss: 0.6895 - val_accuracy: 0.5000
Epoch 2/20
63/63 [==============================] - 8s 119ms/step - loss: 0.6901 -
accuracy: 0.5465 - val_loss: 0.6890 - val_accuracy: 0.5000
Epoch 3/20
63/63 [==============================] - 7s 105ms/step - loss: 0.6874 -
accuracy: 0.5235 - val_loss: 0.6927 - val_accuracy: 0.5080
Epoch 4/20
63/63 [==============================] - 6s 96ms/step - loss: 0.6856 - accuracy:
0.5385 - val_loss: 0.6351 - val_accuracy: 0.6260
Epoch 5/20
63/63 [==============================] - 6s 97ms/step - loss: 0.6616 - accuracy:
0.6165 - val_loss: 0.6387 - val_accuracy: 0.6460
Epoch 6/20
63/63 [==============================] - 6s 93ms/step - loss: 0.6607 - accuracy:
0.5980 - val_loss: 0.7478 - val_accuracy: 0.5060
Epoch 7/20
63/63 [==============================] - 6s 95ms/step - loss: 0.6397 - accuracy:
0.6275 - val_loss: 0.6308 - val_accuracy: 0.6410
Epoch 8/20
63/63 [==============================] - 6s 95ms/step - loss: 0.6404 - accuracy:
0.6365 - val_loss: 0.6304 - val_accuracy: 0.6390
Epoch 9/20
63/63 [==============================] - 6s 95ms/step - loss: 0.6262 - accuracy:
0.6530 - val_loss: 0.6441 - val_accuracy: 0.6220
Epoch 10/20
63/63 [==============================] - 6s 96ms/step - loss: 0.6195 - accuracy:
0.6600 - val_loss: 0.6218 - val_accuracy: 0.6460
Epoch 11/20
63/63 [==============================] - 7s 112ms/step - loss: 0.5927 -
accuracy: 0.6865 - val_loss: 0.5919 - val_accuracy: 0.6730
Epoch 12/20
63/63 [==============================] - 7s 102ms/step - loss: 0.5919 -
accuracy: 0.6830 - val_loss: 0.5749 - val_accuracy: 0.6980
Epoch 13/20
63/63 [==============================] - 6s 95ms/step - loss: 0.5832 - accuracy:
0.6970 - val_loss: 0.5711 - val_accuracy: 0.6910
Epoch 14/20
```

```
63/63 [==============================] - 6s 97ms/step - loss: 0.5738 - accuracy:
0.6980 - val_loss: 0.5679 - val_accuracy: 0.6990
Epoch 15/20
63/63 [==============================] - 6s 95ms/step - loss: 0.5655 - accuracy:
0.7205 - val_loss: 0.5574 - val_accuracy: 0.6970
Epoch 16/20
63/63 [==============================] - 6s 94ms/step - loss: 0.5577 - accuracy:
0.7170 - val_loss: 0.5669 - val_accuracy: 0.6970
Epoch 17/20
63/63 [==============================] - 6s 95ms/step - loss: 0.5318 - accuracy:
0.7270 - val_loss: 0.5526 - val_accuracy: 0.7070
Epoch 18/20
63/63 [==============================] - 6s 96ms/step - loss: 0.5484 - accuracy:
0.7350 - val_loss: 0.5823 - val_accuracy: 0.6870
Epoch 19/20
63/63 [==============================] - 6s 95ms/step - loss: 0.5392 - accuracy:
0.7325 - val_loss: 0.5716 - val_accuracy: 0.6810
Epoch 20/20
63/63 [==============================] - 6s 95ms/step - loss: 0.5098 - accuracy:
0.7495 - val_loss: 0.4997 - val_accuracy: 0.7630
```

[ ]:
```
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 35ms/step - loss: 0.5535 - accuracy:
0.7190
Test accuracy: 0.719
```

0.3 Leveraging a pretrained model

0.3.1 Feature extraction with a pretrained model

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/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 3s 0us/step
```

[ ]:
```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (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
-----------------------------------------------------------------
```

Fast feature extraction without data augmentation Extracting the VGG16 features and corresponding labels

[ ]:

```python
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 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
```

```
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
```

```
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 1s 850ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
```

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/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 15ms/step - loss: 17.7325 -
accuracy: 0.9210 - val_loss: 8.7942 - val_accuracy: 0.9460
Epoch 2/20
63/63 [==============================] - 0s 7ms/step - loss: 4.0179 - accuracy:
0.9755 - val_loss: 4.5752 - val_accuracy: 0.9670
Epoch 3/20
63/63 [==============================] - 0s 7ms/step - loss: 2.3702 - accuracy:
0.9830 - val_loss: 4.9859 - val_accuracy: 0.9700
Epoch 4/20
63/63 [==============================] - 0s 8ms/step - loss: 1.0627 - accuracy:
0.9920 - val_loss: 3.5379 - val_accuracy: 0.9770
Epoch 5/20
63/63 [==============================] - 0s 7ms/step - loss: 1.0989 - accuracy:
0.9910 - val_loss: 4.1865 - val_accuracy: 0.9780
Epoch 6/20
63/63 [==============================] - 0s 7ms/step - loss: 0.9384 - accuracy:
```

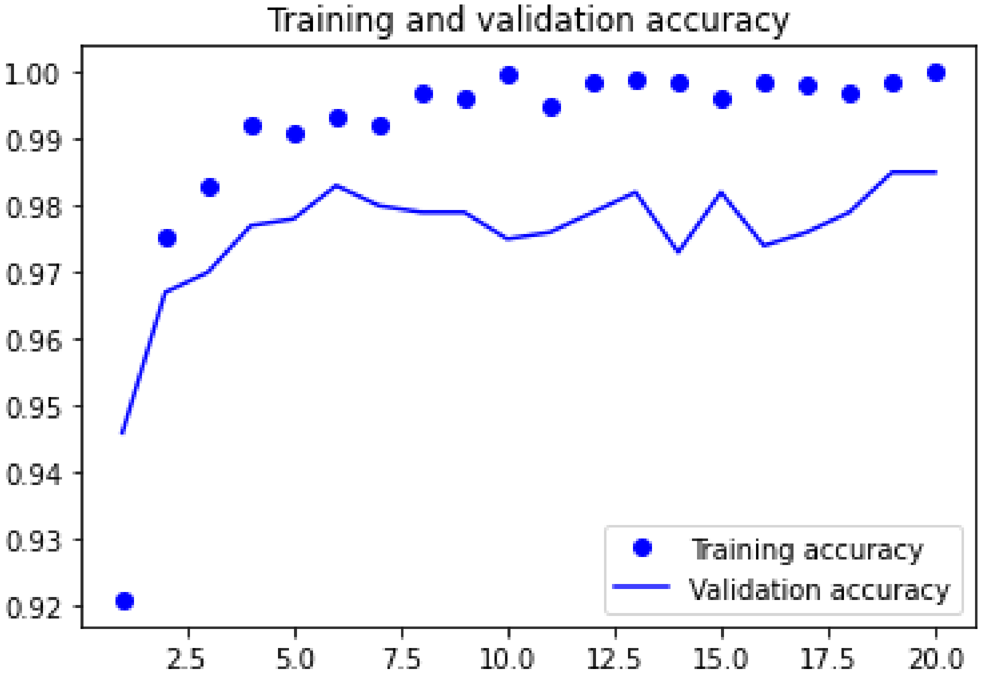
```
0.9935 - val_loss: 3.9540 - val_accuracy: 0.9830
Epoch 7/20
63/63 [==============================] - 0s 7ms/step - loss: 1.3168 - accuracy:
0.9920 - val_loss: 3.9114 - val_accuracy: 0.9800
Epoch 8/20
63/63 [==============================] - 0s 8ms/step - loss: 0.3205 - accuracy:
0.9970 - val_loss: 4.0782 - val_accuracy: 0.9790
Epoch 9/20
63/63 [==============================] - 0s 7ms/step - loss: 0.1869 - accuracy:
0.9960 - val_loss: 4.0056 - val_accuracy: 0.9790
Epoch 10/20
63/63 [==============================] - 0s 7ms/step - loss: 0.1071 - accuracy:
0.9995 - val_loss: 5.7270 - val_accuracy: 0.9750
Epoch 11/20
63/63 [==============================] - 0s 8ms/step - loss: 0.5015 - accuracy:
0.9950 - val_loss: 6.2384 - val_accuracy: 0.9760
Epoch 12/20
63/63 [==============================] - 0s 7ms/step - loss: 0.2811 - accuracy:
0.9985 - val_loss: 4.9711 - val_accuracy: 0.9790
Epoch 13/20
63/63 [==============================] - 0s 7ms/step - loss: 0.1516 - accuracy:
0.9990 - val_loss: 4.9069 - val_accuracy: 0.9820
Epoch 14/20
63/63 [==============================] - 0s 6ms/step - loss: 0.2287 - accuracy:
0.9985 - val_loss: 6.5530 - val_accuracy: 0.9730
Epoch 15/20
63/63 [==============================] - 0s 7ms/step - loss: 0.4308 - accuracy:
0.9960 - val_loss: 4.3745 - val_accuracy: 0.9820
Epoch 16/20
63/63 [==============================] - 0s 7ms/step - loss: 0.0987 - accuracy:
0.9985 - val_loss: 5.6163 - val_accuracy: 0.9740
Epoch 17/20
63/63 [==============================] - 0s 7ms/step - loss: 0.2331 - accuracy:
0.9980 - val_loss: 4.9386 - val_accuracy: 0.9760
Epoch 18/20
63/63 [==============================] - 0s 6ms/step - loss: 0.2972 - accuracy:
0.9970 - val_loss: 3.8418 - val_accuracy: 0.9790
Epoch 19/20
63/63 [==============================] - 0s 7ms/step - loss: 0.1034 - accuracy:
0.9985 - val_loss: 4.7512 - val_accuracy: 0.9850
Epoch 20/20
63/63 [==============================] - 0s 6ms/step - loss: 1.2127e-12 -
accuracy: 1.0000 - val_loss: 4.7504 - val_accuracy: 0.9850
```

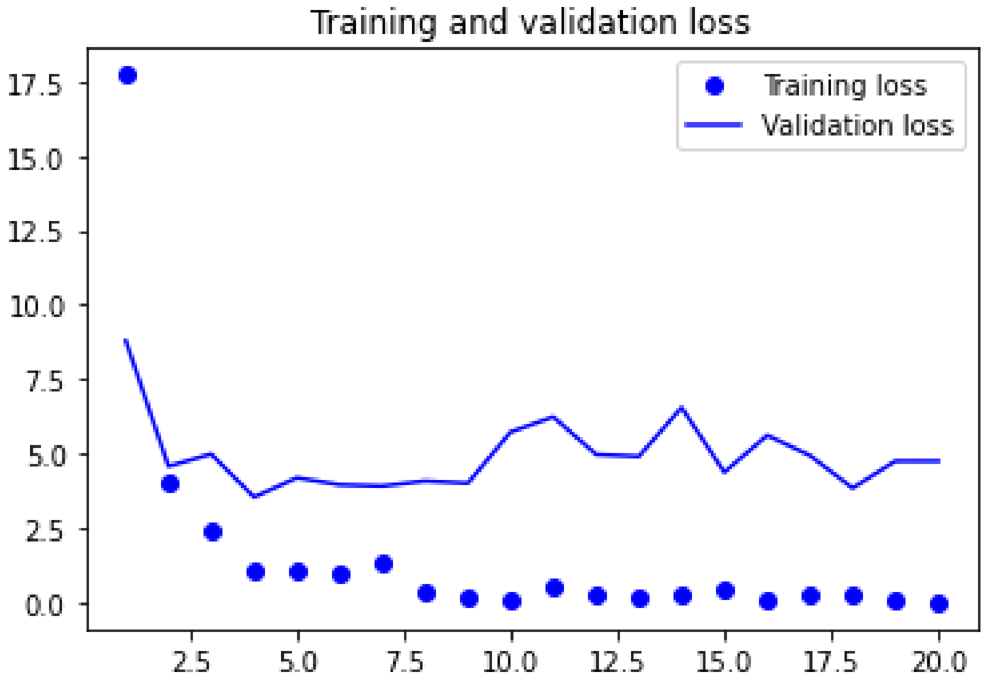
Plotting the results

[ ]:

```python
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()
```





Feature extraction together with data augmentation **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=10,
         validation_data=validation_dataset,
         callbacks=callbacks)
```

```
Epoch 1/10
63/63 [==============================] - 15s 205ms/step - loss: 18.6064 -
accuracy: 0.8980 - val_loss: 4.1063 - val_accuracy: 0.9700
Epoch 2/10
63/63 [==============================] - 13s 203ms/step - loss: 7.2732 -
accuracy: 0.9435 - val_loss: 4.8827 - val_accuracy: 0.9730
Epoch 3/10
63/63 [==============================] - 13s 203ms/step - loss: 5.5691 -
accuracy: 0.9580 - val_loss: 6.1887 - val_accuracy: 0.9660
Epoch 4/10
63/63 [==============================] - 13s 200ms/step - loss: 4.8310 -
accuracy: 0.9555 - val_loss: 8.2597 - val_accuracy: 0.9530
Epoch 5/10
63/63 [==============================] - 13s 205ms/step - loss: 4.3823 -
```

```
accuracy: 0.9600 - val_loss: 3.3014 - val_accuracy: 0.9800
Epoch 6/10
63/63 [==============================] - 14s 214ms/step - loss: 3.6643 -
accuracy: 0.9715 - val_loss: 4.2373 - val_accuracy: 0.9770
Epoch 7/10
63/63 [==============================] - 13s 201ms/step - loss: 3.2210 -
accuracy: 0.9725 - val_loss: 3.6654 - val_accuracy: 0.9790
Epoch 8/10
63/63 [==============================] - 13s 202ms/step - loss: 3.5239 -
accuracy: 0.9705 - val_loss: 3.5401 - val_accuracy: 0.9750
Epoch 9/10
63/63 [==============================] - 13s 202ms/step - loss: 2.6867 -
accuracy: 0.9720 - val_loss: 3.5508 - val_accuracy: 0.9830
Epoch 10/10
63/63 [==============================] - 13s 200ms/step - loss: 3.2733 -
accuracy: 0.9655 - val_loss: 11.2718 - val_accuracy: 0.9440
```

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: 2.9723 -
accuracy: 0.9760
Test accuracy: 0.976
```

0.3.2 Fine-tuning a pretrained model

[ ]:
```
conv_base.summary()
```

```
Model: "vgg16"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (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

[ ]:

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

Fine-tuning the model

[ ]:

```python
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=10,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/10
63/63 [==============================] - 16s 228ms/step - loss: 1.9848 -
accuracy: 0.9740 - val_loss: 2.8764 - val_accuracy: 0.9790
Epoch 2/10
63/63 [==============================] - 14s 226ms/step - loss: 1.0059 -
accuracy: 0.9835 - val_loss: 2.4184 - val_accuracy: 0.9820
Epoch 3/10
63/63 [==============================] - 14s 222ms/step - loss: 1.0069 -
accuracy: 0.9855 - val_loss: 4.0815 - val_accuracy: 0.9690
Epoch 4/10
63/63 [==============================] - 14s 220ms/step - loss: 0.7896 -
accuracy: 0.9870 - val_loss: 2.9842 - val_accuracy: 0.9750
Epoch 5/10
63/63 [==============================] - 14s 220ms/step - loss: 0.6948 -
accuracy: 0.9890 - val_loss: 2.9258 - val_accuracy: 0.9780
Epoch 6/10
63/63 [==============================] - 15s 229ms/step - loss: 0.5724 -
accuracy: 0.9910 - val_loss: 3.2697 - val_accuracy: 0.9750
Epoch 7/10
63/63 [==============================] - 14s 220ms/step - loss: 0.6571 -
accuracy: 0.9885 - val_loss: 2.5963 - val_accuracy: 0.9820
Epoch 8/10
63/63 [==============================] - 14s 219ms/step - loss: 0.6482 -
accuracy: 0.9880 - val_loss: 2.6384 - val_accuracy: 0.9810
Epoch 9/10
63/63 [==============================] - 15s 228ms/step - loss: 0.4116 -
accuracy: 0.9910 - val_loss: 2.4082 - val_accuracy: 0.9820
Epoch 10/10
63/63 [==============================] - 15s 227ms/step - loss: 0.3960 -
accuracy: 0.9910 - val_loss: 2.1156 - val_accuracy: 0.9820
```

[ ]:

```
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 [==============================] - 4s 112ms/step - loss: 1.9253 -
accuracy: 0.9770
Test accuracy: 0.977
```

0.4 Summary

We can observe from the above outputs of three completely new models that the accuracy of the model improves as the training sample size increases. This demonstrates that adding more data constantly improves data training and boosts accuracy. I selected epoch sizes of 15 for model 4, in which we use a pretrained network for classification, and 20 for the first three models, below.1000 training samples and 500 each for validation and test sets. The test accuracy for this model was only 70.10, which is far from acceptable. Additionally, this model was created using solely data augmentation for better data, and no additional optimization approaches were used.As a result, the accuracy value is low.

Increasing Using training samples and the previously calculated sample sizes for validation and test sets The output of the test accuracy was 81.50 for this model. For the goal of optimization, I launched this model by incorporating learning rate and dropout strategies. With a larger sample size and the usage of optimizers, the new model performed better than the older one.

For the model to perform better, increase the amount of the training sample once more. The output of the test accuracy was 88.90 for this model. The model's classification accuracy has definitely improved as compared to the prior model as a result of the larger training sample set.Applying a trained network to the preceding steps. VGG16 Pretrained Convnet was used to create this model. Even though there were only 15 epochs chosen, the pretrained network was used, which considerably improved the model's accuracy while using the same sample sizes as the prior model. Therefore, it is clear that a pretrained network can be useful in creating a better model with less input and greater accuracy as a result of earlier extensive training.