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.

Introduction to deep learning for computer vision

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-datas
11490434/11490434 [==============================] - 1s 0us/step
Epoch 1/5
938/938 [==============================] - 14s 4ms/step - loss: 0.1506 - accur
Epoch 2/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0438 - accura
Epoch 3/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0312 - accura
Epoch 4/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0229 - accura
Epoch 5/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0183 - accura
<keras.callbacks.History at 0x7f0020132390>
```

Evaluating the convnet

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

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0333 - accura
Test accuracy: 0.990
```

The convolution operation

Understanding border effects and padding

Understanding convolution strides

The max-pooling operation

An incorrectly structured convnet missing its max-pooling layers

```
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)
```

```
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
_________________________________________________________________
```

Training a convnet from scratch on a small dataset

The relevance of deep learning for small-data problems

Downloading the data

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

Choose files kaggle.json

- **kaggle.json**(application/json) - 64 bytes, last modified: 05/11/2022 - 100% done

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

{'kaggle.json':
b'{"username":"ribakhan","key":"2cb4a7bc3a38aecc4d227f3f9ae26a8b"}'}

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

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

```
Downloading dogs-vs-cats.zip to /content
100% 811M/812M [00:20<00:00, 42.2MB/s]
100% 812M/812M [00:20<00:00, 40.9MB/s]
```

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

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

Question-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. What performance did you achieve?

Creating a convolutional network. Here we are diving the data into test, train and validation as mentioned in the question.

```
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("test", start_index=0, end_index=500)
```

```
make_subset("validation", start_index=500, end_index=1000)
make_subset("train", start_index=1000, end_index=2000)
```

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)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_11 (Conv2D)          (None, 178, 178, 32)      896

 max_pooling2d_6 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_12 (Conv2D)          (None, 87, 87, 64)        18496

 max_pooling2d_7 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_13 (Conv2D)          (None, 41, 41, 128)       73856

 max_pooling2d_8 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_14 (Conv2D)          (None, 18, 18, 256)       295168
```

```
 max_pooling2d_9 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_15 (Conv2D)          (None, 7, 7, 256)         590080

 flatten_3 (Flatten)         (None, 12544)             0

 dense_3 (Dense)             (None, 1)                 12545

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

Total parameteres turned out to be 991,041. Parameters in general are weights that are learnt during training.

Configuring the model for training

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

Data preprocessing

Using `image_dataset_from_directory` to read images

Modifying the images to tensors.

```
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`

The reason we are using "callbacks" here is beacuse Callbacks can help prevent overfitting, visualize training progress and create a TensorBoard, etc. In order to avoid having to retrain the model, we will employ "callbacks," which will automatically save a file with the weights produced

Using only 30 epochs here.

```
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)
```

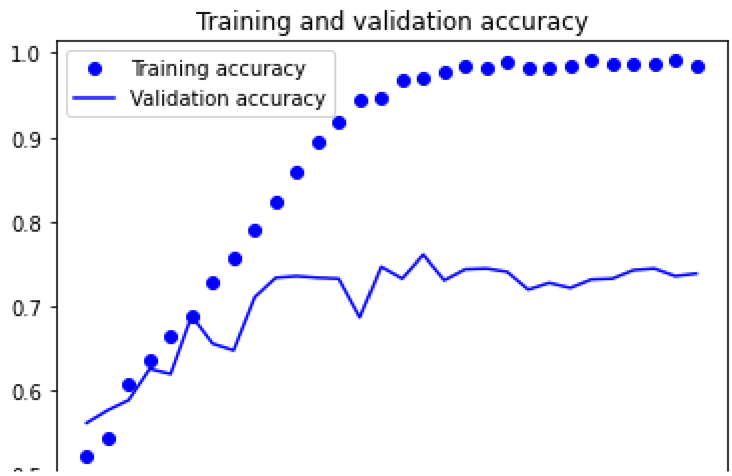
```
Epoch 2/30
63/63 [==============================] - 5s 72ms/step - loss: 0.6967 - accurac
Epoch 3/30
63/63 [==============================] - 5s 71ms/step - loss: 0.6758 - accurac
Epoch 4/30
63/63 [==============================] - 5s 71ms/step - loss: 0.6544 - accurac
Epoch 5/30
63/63 [==============================] - 5s 78ms/step - loss: 0.6143 - accurac
Epoch 6/30
63/63 [==============================] - 5s 72ms/step - loss: 0.5907 - accurac
Epoch 7/30
63/63 [==============================] - 5s 71ms/step - loss: 0.5405 - accurac
Epoch 8/30
63/63 [==============================] - 6s 83ms/step - loss: 0.5022 - accurac
Epoch 9/30
63/63 [==============================] - 5s 72ms/step - loss: 0.4530 - accurac
Epoch 10/30
63/63 [==============================] - 5s 72ms/step - loss: 0.3888 - accurac
Epoch 11/30
63/63 [==============================] - 5s 71ms/step - loss: 0.3384 - accurac
Epoch 12/30
63/63 [==============================] - 5s 70ms/step - loss: 0.2681 - accurac
Epoch 13/30
63/63 [==============================] - 5s 71ms/step - loss: 0.2060 - accurac
Epoch 14/30
63/63 [==============================] - 5s 71ms/step - loss: 0.1586 - accurac
Epoch 15/30
63/63 [==============================] - 6s 88ms/step - loss: 0.1308 - accurac
Epoch 16/30
63/63 [==============================] - 5s 71ms/step - loss: 0.0906 - accurac
Epoch 17/30
63/63 [==============================] - 5s 71ms/step - loss: 0.0808 - accurac
Epoch 18/30
63/63 [==============================] - 5s 71ms/step - loss: 0.0687 - accurac
Epoch 19/30
63/63 [==============================] - 5s 73ms/step - loss: 0.0628 - accurac
Epoch 20/30
63/63 [==============================] - 6s 84ms/step - loss: 0.0768 - accurac
Epoch 21/30

63/63 [==============================] - 5s 74ms/step - loss: 0.0406 - accurac
Epoch 22/30
63/63 [==============================] - 6s 86ms/step - loss: 0.0486 - accurac
```

```
Epoch 23/30
63/63 [==============================] - 5s 71ms/step - loss: 0.0557 - accurac
Epoch 24/30
63/63 [==============================] - 5s 72ms/step - loss: 0.0490 - accurac
Epoch 25/30
63/63 [==============================] - 5s 71ms/step - loss: 0.0419 - accurac
Epoch 26/30
63/63 [==============================] - 5s 74ms/step - loss: 0.0461 - accurac
Epoch 27/30
63/63 [==============================] - 5s 72ms/step - loss: 0.0636 - accurac
Epoch 28/30
63/63 [==============================] - 5s 72ms/step - loss: 0.0446 - accurac
Epoch 29/30
63/63 [==============================] - 5s 81ms/step - loss: 0.0240 - accurac
Epoch 30/30
63/63 [==============================] - 5s 73ms/step - loss: 0.0496 - accurac
```

Validation accuracy 73%.

Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
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.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()
```



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: 0.5290 - accurac
Test accuracy: 0.742
```

The accuracy turns out to be 74% which is not fairly very high. Next I will perform additional steps to improve model perfomance.

Now, in order to improve model performance I will use the dropout method. Using a dropout layer of 0.5

Including dropout in a new convent

```
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)
```

Compiling model

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

Now training the convent

Here, i am using only 50 epochs due to time constraint.

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

```
Epoch 22/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0831 - accurac
Epoch 23/50
63/63 [==============================] - 5s 73ms/step - loss: 0.1051 - accurac
Epoch 24/50
63/63 [==============================] - 5s 74ms/step - loss: 0.0621 - accurac
Epoch 25/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0874 - accurac
Epoch 26/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0357 - accurac
Epoch 27/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0895 - accurac
Epoch 28/50
63/63 [==============================] - 6s 89ms/step - loss: 0.0589 - accurac
Epoch 29/50
63/63 [==============================] - 6s 82ms/step - loss: 0.0376 - accurac
Epoch 30/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0688 - accurac
Epoch 31/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0555 - accurac
Epoch 32/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0581 - accurac
Epoch 33/50
63/63 [==============================] - 6s 87ms/step - loss: 0.0494 - accurac
Epoch 34/50
63/63 [==============================] - 5s 74ms/step - loss: 0.0785 - accurac
Epoch 35/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0208 - accurac
Epoch 36/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0580 - accurac
Epoch 37/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0443 - accurac
Epoch 38/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0470 - accurac
```

```
63/63 [==============================] - 5s 75ms/step - loss: 0.0470 - accurac
Epoch 39/50
63/63 [==============================] - 6s 89ms/step - loss: 0.0517 - accurac
Epoch 40/50
63/63 [==============================] - 5s 80ms/step - loss: 0.0473 - accurac
Epoch 41/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0363 - accurac
Epoch 42/50
63/63 [==============================] - 6s 92ms/step - loss: 0.0405 - accurac
Epoch 43/50
63/63 [==============================] - 5s 74ms/step - loss: 0.0420 - accurac
Epoch 44/50
63/63 [==============================] - 6s 87ms/step - loss: 0.0445 - accurac
Epoch 45/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0542 - accurac
Epoch 46/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0492 - accurac
Epoch 47/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0434 - accurac
Epoch 48/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0242 - accurac
Epoch 49/50
63/63 [==============================] - 5s 73ms/step - loss: 0.0794 - accurac
Epoch 50/50
63/63 [==============================] - 5s 72ms/step - loss: 0.0274 - accurac
```

Validation accuracy turns out to be 73%

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

```
32/32 [==============================] - 2s 38ms/step - loss: 0.5150 - accurac
Test accuracy: 0.740
```

The second method i will be using to improve the model performance is using data augmentaion technique combining with dropout function

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

including 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 model

I am using only 50 epochs here.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation_dropout.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 99ms/step - loss: 0.8148 - accurac
Epoch 2/50
63/63 [==============================] - 6s 95ms/step - loss: 0.6953 - accurac
Epoch 3/50
63/63 [==============================] - 6s 97ms/step - loss: 0.6917 - accurac
Epoch 4/50
63/63 [==============================] - 7s 108ms/step - loss: 0.6759 - accura
Epoch 5/50
63/63 [==============================] - 6s 97ms/step - loss: 0.6769 - accurac
Epoch 6/50
63/63 [==============================] - 6s 96ms/step - loss: 0.6278 - accurac
Epoch 7/50
63/63 [==============================] - 6s 96ms/step - loss: 0.6102 - accurac
Epoch 8/50
63/63 [==============================] - 6s 96ms/step - loss: 0.6065 - accurac
Epoch 9/50
63/63 [==============================] - 8s 117ms/step - loss: 0.5976 - accura
Epoch 10/50
```

```
63/63 [==============================] - 6s 94ms/step - loss: 0.5857 - accurac
Epoch 11/50
63/63 [==============================] - 6s 96ms/step - loss: 0.5812 - accurac
Epoch 12/50
63/63 [==============================] - 6s 96ms/step - loss: 0.5656 - accurac
Epoch 13/50
63/63 [==============================] - 6s 95ms/step - loss: 0.5496 - accurac
Epoch 14/50
63/63 [==============================] - 6s 95ms/step - loss: 0.5160 - accurac
Epoch 15/50
63/63 [==============================] - 7s 105ms/step - loss: 0.5267 - accura
Epoch 16/50
63/63 [==============================] - 6s 96ms/step - loss: 0.5273 - accurac
Epoch 17/50
63/63 [==============================] - 6s 95ms/step - loss: 0.5080 - accurac
Epoch 18/50
63/63 [==============================] - 6s 94ms/step - loss: 0.4829 - accurac
Epoch 19/50
63/63 [==============================] - 6s 97ms/step - loss: 0.4920 - accurac
Epoch 20/50
63/63 [==============================] - 7s 104ms/step - loss: 0.4664 - accura
Epoch 21/50
63/63 [==============================] - 6s 95ms/step - loss: 0.4822 - accurac
Epoch 22/50
63/63 [==============================] - 6s 95ms/step - loss: 0.4640 - accurac
Epoch 23/50
63/63 [==============================] - 6s 97ms/step - loss: 0.4506 - accurac
Epoch 24/50
63/63 [==============================] - 7s 106ms/step - loss: 0.4400 - accura
Epoch 25/50
63/63 [==============================] - 6s 93ms/step - loss: 0.4389 - accurac
Epoch 26/50
63/63 [==============================] - 6s 93ms/step - loss: 0.4089 - accurac
Epoch 27/50
63/63 [==============================] - 6s 92ms/step - loss: 0.4286 - accurac
Epoch 28/50
63/63 [==============================] - 6s 94ms/step - loss: 0.4067 - accurac
Epoch 29/50
```

Validation accuracy turns out to be 81% which is improved by 8% than before.

Question 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?

```
make_subset("train3", start_index=1000, end_index=6000)

train_dataset_3 = image_dataset_from_directory(
    new_base_dir / "train3",
    image_size=(180, 180),
    batch_size=32)
```

```
Found 10000 files belonging to 2 classes.
```

I have increased the training sample size to 5000.

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

Training the regularized convent. I am using 60 epochs here.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset_3,
    epochs=70,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 42/70
313/313 [==============================] - 22s 69ms/step - loss: 0.1318 - accu
Epoch 43/70
313/313 [==============================] - 23s 73ms/step - loss: 0.1275 - accu
Epoch 44/70
313/313 [==============================] - 21s 65ms/step - loss: 0.1139 - accu
Epoch 45/70
313/313 [==============================] - 19s 59ms/step - loss: 0.1227 - accu
Epoch 46/70
313/313 [==============================] - 21s 66ms/step - loss: 0.1266 - accu
Epoch 47/70
313/313 [==============================] - 21s 66ms/step - loss: 0.1553 - accu
Epoch 48/70
313/313 [==============================] - 20s 64ms/step - loss: 0.1647 - accu
Epoch 49/70
313/313 [==============================] - 22s 71ms/step - loss: 0.1315 - accu
```

```
313/313 [==============================] - 22s 71ms/step - loss: 0.1515 - accu
Epoch 50/70
313/313 [==============================] - 21s 65ms/step - loss: 0.1501 - accu
Epoch 51/70
313/313 [==============================] - 22s 69ms/step - loss: 0.1431 - accu
Epoch 52/70
313/313 [==============================] - 21s 65ms/step - loss: 0.1543 - accu
Epoch 53/70
313/313 [==============================] - 21s 68ms/step - loss: 0.1696 - accu
Epoch 54/70
313/313 [==============================] - 23s 72ms/step - loss: 0.1312 - accu
Epoch 55/70
313/313 [==============================] - 21s 67ms/step - loss: 0.1072 - accu
Epoch 56/70
313/313 [==============================] - 21s 66ms/step - loss: 0.1478 - accu
Epoch 57/70
313/313 [==============================] - 20s 63ms/step - loss: 0.1218 - accu
Epoch 58/70
313/313 [==============================] - 20s 62ms/step - loss: 0.1368 - accu
Epoch 59/70
313/313 [==============================] - 20s 65ms/step - loss: 0.1433 - accu
Epoch 60/70
313/313 [==============================] - 24s 75ms/step - loss: 0.1672 - accu
Epoch 61/70
313/313 [==============================] - 22s 71ms/step - loss: 0.1445 - accu
Epoch 62/70
313/313 [==============================] - 21s 65ms/step - loss: 0.1788 - accu
Epoch 63/70
313/313 [==============================] - 21s 65ms/step - loss: 0.1608 - accu
Epoch 64/70
313/313 [==============================] - 24s 74ms/step - loss: 0.1395 - accu
Epoch 65/70
313/313 [==============================] - 21s 66ms/step - loss: 0.1429 - accu
Epoch 66/70
313/313 [==============================] - 23s 71ms/step - loss: 0.1357 - accu
Epoch 67/70
313/313 [==============================] - 21s 67ms/step - loss: 0.1816 - accu

Epoch 68/70
313/313 [==============================] - 20s 63ms/step - loss: 0.1727 - accu
Epoch 69/70
313/313 [==============================] - 18s 58ms/step - loss: 0.1965 - accu
Epoch 70/70
313/313 [==============================] - 20s 65ms/step - loss: 0.1768 - accu
```

Validation accuracy turns out to be 88%

Checking the model accuracy

```
test_model = keras.models.load_model(
    "convnet_from_scratch3.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 50ms/step - loss: 0.3202 - accurac
Test accuracy: 0.876
```

Question 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.

```
#import shutil
#shutil.rmtree('/content/cats_vs_dogs_small/train2')

make_subset("train2", start_index=1000, end_index=11000)

train_dataset2 = image_dataset_from_directory(
    new_base_dir / "train2",
    image_size=(180, 180),
    batch_size=32)
```

```
Found 20000 files belonging to 2 classes.
```

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

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_test3.keras",
        save_best_only=True,
        monitor="val_loss")
]

history = model.fit(
    train_dataset_3,
    epochs=50,
```

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

```
Epoch 22/50
313/313 [==============================] - 21s 67ms/step - loss: 0.0650 - accu
Epoch 23/50
313/313 [==============================] - 21s 66ms/step - loss: 0.0547 - accu
Epoch 24/50
313/313 [==============================] - 23s 72ms/step - loss: 0.0624 - accu
Epoch 25/50
313/313 [==============================] - 21s 66ms/step - loss: 0.0571 - accu
Epoch 26/50
313/313 [==============================] - 20s 62ms/step - loss: 0.0698 - accu
Epoch 27/50
313/313 [==============================] - 21s 67ms/step - loss: 0.0663 - accu
Epoch 28/50
313/313 [==============================] - 20s 65ms/step - loss: 0.0507 - accu
Epoch 29/50
313/313 [==============================] - 21s 67ms/step - loss: 0.0580 - accu
Epoch 30/50
313/313 [==============================] - 22s 69ms/step - loss: 0.0652 - accu
Epoch 31/50
313/313 [==============================] - 22s 70ms/step - loss: 0.0635 - accu
Epoch 32/50
313/313 [==============================] - 20s 65ms/step - loss: 0.0735 - accu
Epoch 33/50
313/313 [==============================] - 21s 66ms/step - loss: 0.0757 - accu
Epoch 34/50
313/313 [==============================] - 21s 66ms/step - loss: 0.0614 - accu
Epoch 35/50
313/313 [==============================] - 20s 62ms/step - loss: 0.0703 - accu
Epoch 36/50
313/313 [==============================] - 22s 68ms/step - loss: 0.0674 - accu
Epoch 37/50
313/313 [==============================] - 20s 63ms/step - loss: 0.0767 - accu
Epoch 38/50
313/313 [==============================] - 21s 65ms/step - loss: 0.0611 - accu
Epoch 39/50
313/313 [==============================] - 20s 63ms/step - loss: 0.0698 - accu
Epoch 40/50
313/313 [==============================] - 21s 67ms/step - loss: 0.0670 - accu
Epoch 41/50
313/313 [==============================] - 19s 61ms/step - loss: 0.0640 - accu
Epoch 42/50
313/313 [==============================] - 22s 69ms/step - loss: 0.0811 - accu
Epoch 43/50
313/313 [==============================] - 21s 66ms/step - loss: 0.0771 - accu
Epoch 44/50
313/313 [==============================] - 20s 62ms/step - loss: 0.0720 - accu
Epoch 45/50
313/313 [==============================] - 21s 65ms/step - loss: 0.0642 - accu
Epoch 46/50
313/313 [==============================] - 20s 63ms/step - loss: 0.0795 - accu
Epoch 47/50
313/313 [==============================] - 20s 63ms/step - loss: 0.0997 - accu
Epoch 48/50
313/313 [==============================] - 23s 72ms/step - loss: 0.0891 - accu
Epoch 49/50
313/313 [==============================] - 21s 66ms/step - loss: 0.0808 - accu
Epoch 50/50
313/313 [==============================] - 21s 65ms/step - loss: 0.0889 - accu
```

```
313/313 [==============================] - 21s 65ms/step - loss: 0.0889 - accu
```

Check model accuracy

```
test_model = keras.models.load_model(
    "convnet_from_scratch_test3.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 37ms/step - loss: 0.4147 - accurac
Test accuracy: 0.834
```

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

Training regularized convent

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch2.keras",
        save_best_only=True,
        monitor="val_loss")
]

history = model.fit(
    train_dataset2,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 2/30
625/625 [==============================] - 34s 55ms/step - loss: 0.4786 - accu
Epoch 3/30
625/625 [==============================] - 34s 54ms/step - loss: 0.3925 - accu
```

```
625/625 [==============================] - 34s 54ms/step - loss: 0.3925 - accu
Epoch 4/30
625/625 [==============================] - 34s 54ms/step - loss: 0.3212 - accu
Epoch 5/30
625/625 [==============================] - 35s 56ms/step - loss: 0.2722 - accu
Epoch 6/30
625/625 [==============================] - 34s 55ms/step - loss: 0.2287 - accu
Epoch 7/30
625/625 [==============================] - 34s 54ms/step - loss: 0.2005 - accu
Epoch 8/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1757 - accu
Epoch 9/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1672 - accu
Epoch 10/30
625/625 [==============================] - 35s 55ms/step - loss: 0.1594 - accu
Epoch 11/30
625/625 [==============================] - 34s 55ms/step - loss: 0.1549 - accu
Epoch 12/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1540 - accu
Epoch 13/30
625/625 [==============================] - 34s 55ms/step - loss: 0.1492 - accu
Epoch 14/30
625/625 [==============================] - 35s 56ms/step - loss: 0.1426 - accu
Epoch 15/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1416 - accu
Epoch 16/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1594 - accu
Epoch 17/30
625/625 [==============================] - 35s 55ms/step - loss: 0.1448 - accu
Epoch 18/30
625/625 [==============================] - 35s 56ms/step - loss: 0.1565 - accu
Epoch 19/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1679 - accu
Epoch 20/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1618 - accu
Epoch 21/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1598 - accu
Epoch 22/30
625/625 [==============================] - 35s 56ms/step - loss: 0.1509 - accu
Epoch 23/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1736 - accu
Epoch 24/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1644 - accu
Epoch 25/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1751 - accu
Epoch 26/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1731 - accu
Epoch 27/30
625/625 [==============================] - 35s 56ms/step - loss: 0.1737 - accu
Epoch 28/30
625/625 [==============================] - 34s 54ms/step - loss: 0.1888 - accu
Epoch 29/30
625/625 [==============================] - 34s 53ms/step - loss: 0.1907 - accu
Epoch 30/30
625/625 [==============================] - 34s 55ms/step - loss: 0.1991 - accu
```

Now evelauting the test data

```
test_model = keras.models.load_model(
    "convnet_from_scratch2.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 36ms/step - loss: 0.2422 - accurac
Test accuracy: 0.917
```

Question 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 pre trained model. Using the VGG16 convolution base. Here using 1000 samples.

```
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-applicat
58889256/58889256 [==============================] - 2s 0us/step
```

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_11 (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
_________________________________________________________________
```

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

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

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

```
Epoch 1/20
63/63 [==============================] - 25s 337ms/step - loss: 3.9529 - accur
Epoch 2/20
63/63 [==============================] - 14s 216ms/step - loss: 1.0367 - accur
Epoch 3/20
63/63 [==============================] - 14s 226ms/step - loss: 0.4728 - accur
Epoch 4/20
63/63 [==============================] - 14s 217ms/step - loss: 0.3889 - accur
Epoch 5/20
63/63 [==============================] - 14s 219ms/step - loss: 0.2496 - accur
Epoch 6/20
63/63 [==============================] - 14s 218ms/step - loss: 0.2236 - accur
Epoch 7/20
63/63 [==============================] - 14s 217ms/step - loss: 0.2079 - accur
Epoch 8/20
63/63 [==============================] - 14s 218ms/step - loss: 0.1453 - accur
Epoch 9/20
63/63 [==============================] - 14s 225ms/step - loss: 0.1602 - accur
Epoch 10/20
63/63 [==============================] - 14s 212ms/step - loss: 0.1200 - accur
Epoch 11/20
63/63 [==============================] - 14s 222ms/step - loss: 0.1016 - accur
Epoch 12/20
63/63 [==============================] - 14s 214ms/step - loss: 0.0856 - accur
Epoch 13/20
63/63 [==============================] - 14s 217ms/step - loss: 0.0619 - accur
Epoch 14/20
63/63 [==============================] - 14s 220ms/step - loss: 0.0932 - accur
Epoch 15/20
63/63 [==============================] - 14s 215ms/step - loss: 0.0572 - accur
Epoch 16/20
63/63 [==============================] - 14s 214ms/step - loss: 0.0510 - accur
Epoch 17/20
63/63 [==============================] - 14s 215ms/step - loss: 0.0517 - accur
Epoch 18/20
63/63 [==============================] - 15s 227ms/step - loss: 0.0384 - accur
Epoch 19/20
63/63 [==============================] - 14s 213ms/step - loss: 0.0426 - accur
Epoch 20/20
63/63 [==============================] - 14s 214ms/step - loss: 0.0567 - accur
```

```
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 105ms/step - loss: 0.1480 - accura
Test accuracy: 0.977
```

Pre trained model - 10000 samples

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

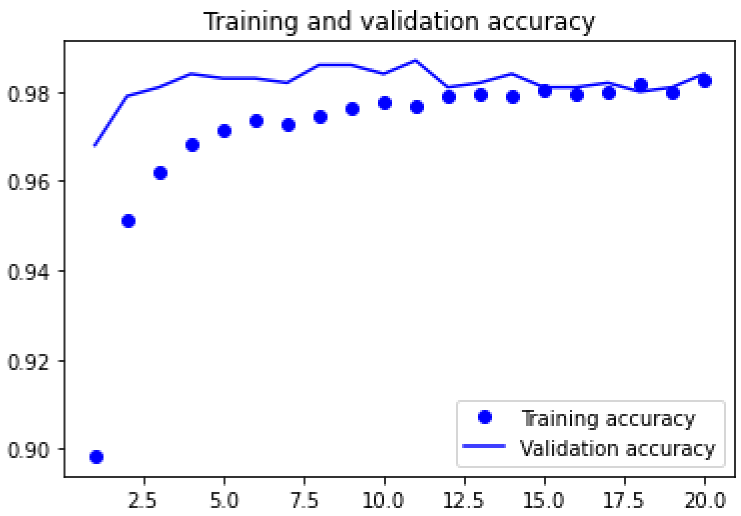
```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)

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

```
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=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning3.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset2,
    epochs=20,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
och 1/20
5/625 [==============================] - 104s 164ms/step - loss: 0.8159 - accur
och 2/20
5/625 [==============================] - 105s 167ms/step - loss: 0.1522 - accur
och 3/20
5/625 [==============================] - 104s 166ms/step - loss: 0.1170 - accur
```

```
och 4/20
5/625 [==============================] - 104s 166ms/step - loss: 0.0936 - accur
och 5/20
5/625 [==============================] - 105s 168ms/step - loss: 0.0880 - accur
och 6/20
5/625 [==============================] - 104s 166ms/step - loss: 0.0815 - accur
och 7/20
5/625 [==============================] - 105s 168ms/step - loss: 0.0816 - accur
och 8/20
5/625 [==============================] - 104s 165ms/step - loss: 0.0747 - accur
och 9/20
5/625 [==============================] - 105s 168ms/step - loss: 0.0788 - accur
och 10/20
5/625 [==============================] - 105s 168ms/step - loss: 0.0751 - accur
och 11/20
5/625 [==============================] - 104s 165ms/step - loss: 0.0703 - accur
och 12/20
5/625 [==============================] - 104s 166ms/step - loss: 0.0735 - accur
och 13/20
5/625 [==============================] - 104s 166ms/step - loss: 0.0681 - accur
och 14/20
5/625 [==============================] - 104s 166ms/step - loss: 0.0748 - accur
och 15/20
5/625 [==============================] - 104s 166ms/step - loss: 0.0772 - accur
och 16/20
5/625 [==============================] - 103s 165ms/step - loss: 0.0798 - accur
och 17/20
5/625 [==============================] - 105s 167ms/step - loss: 0.0769 - accur
och 18/20
5/625 [==============================] - 105s 168ms/step - loss: 0.0765 - accur
och 19/20
5/625 [==============================] - 105s 168ms/step - loss: 0.0871 - accur
och 20/20
5/625 [==============================] - 104s 166ms/step - loss: 0.0734 - accur
```

```
import matplotlib.pyplot as plt
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.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()
```




```
model = keras.models.load_model("fine_tuning3.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 4s 110ms/step - loss: 0.0866 - accura
Test accuracy: 0.986
```

SUMMARY

Q1) In question 1 the above plots clearly show that the model is facing the overfitting issue. Training accuracy increases gradually until it reaches the highest point. While validation accuracy reaches only 72- 76%.

To avoid overfitting, a few techniques can be implemented.
I used the following two techniques to improve the model performance:

1) Using a dropout layer of 0.5
2) Combining data augmentation technique with dropout

Techniques used	Test Accuracy	Validation Accuracy
Unregularized Model	74%	73%
Model with Dropout Layer	74%	73%
Model with Data augmentation and dropout layer	81%	81%

Observations from the above table

We can clearly see that test and validation accuracy doesn't show any difference when we used unregularized model and a model with a dropout layer of 0.5. However, the model performance improved by 7% when we used data augmentation with dropout function.

Q2)
In this question, I have increased the training sample to 5000 and a regularized model is used.

Results:
Test loss: 17%
Test accuracy: 87%
Validation accuracy: 88%

In comparison with the unregularized model, the regularized model with the training size of 5000 performs better with higher accuracy.

Q4)

Using a pre-trained model with VGG16 convolution base. Using 1000 samples
Test accuracy: 99%
Validation accuracy: 97%

Using a pre-trained model with 10000 samples
Validation accuracy: 98%
Test accuracy: 98%

Final Conclusion and Recommendations

- **Convolutional network-based machine learning models are the best for computer vision.**
- **Overfitting is the main issue with a tiny dataset. Data augmentation can be a powerful tool for preventing overfitting when working with image data. Model performance improves when the training sample amount is increased.**
- We can use "callbacks" because it prevents overfitting, visualise training progress and create a Tesnorboard. In order to avoid having to retain the model. We employ "callbacks" which will automatically save a file with the weights produced from the best epoch.