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

Training a convnet from scratch on a small dataset

The relevance of deep learning for small-data problems

Downloading the data

Reading in training, validation, test datasets

- Step 1: Download the compressed dataset cats_vs_dogs_small.zip from Canvas. Note: this is a slightly different dataset as the book as it contains 2000 pictures for training, 1000 for validation and 1000 (as opposed to 2000) for testing.
- Step 2: Unzip the file onto your local drive.
- Step 3: Upload your unzipped files (all folders and files) to your Google Drive

Instructions: log into your Google Drive using the same google account of your Google Colab. Find the "Colab Notebooks" folder. Drag the cats_vs_dogs_small folder into the "Colab Notebooks" folder. This should upload all subfolders and pictures onto your Google Drive.

- Step 4: mount your Google Drive within Colab using the following code

```
from google.colab import drive
drive.mount('/content/drive')
```

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

- Step 5: find the path of your datafiles in Google Drive. On the left panel of Colab, Click File -> click folder content -> drive -> ...

For example, my path would look like something in the following: "/content/drive/MyDrive/Colab Notebooks/cats_vs_dogs_small"

Then you can set the path to your files with the following code:

```
import os, shutil, pathlib

new_base_dir = pathlib.Path("drive/MyDrive/Colab Notebooks/cats_vs_dogs_small")
print(new_base_dir)
```

```
drive/MyDrive/Colab Notebooks/cats_vs_dogs_small
```

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. Use any technique to reduce overfitting and improve performance in developing a network that you train from scratch. What performance did you achieve?

```
import os
import shutil
from pathlib import Path

original_data_dir = pathlib.Path("drive/MyDrive/Colab Notebooks/cats_vs_dogs_small")
new_base_dir = Path("cats_vs_dogs_small/train1")

def make_subset(subset_name, start_index, end_index):
    for category in ("cats", "dogs"):
        src_dir = original_data_dir /subset_name/ category
        dst_dir = new_base_dir / subset_name / category

        print(f"Creating subset: {subset_name}, Category: {category}")
        print(f"Destination directory: {dst_dir}")

        os.makedirs(dst_dir, exist_ok=True)
```

```
        fnames = [f"{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            src_file = src_dir / fname
            dst_file = dst_dir / fname


            if src_file.exists():
                shutil.copyfile(src_file, dst_file)
            else:
                print(f"Warning: Source file {src_file} not found.")

make_subset("train", start_index=0, end_index=500)
make_subset("validation", start_index=1000, end_index=1250)
make_subset("test", start_index=1500, end_index=1750)
```

```
Creating subset: train, Category: cats
Destination directory: cats_vs_dogs_small/train_1/train/cats
Creating subset: train, Category: dogs
Destination directory: cats_vs_dogs_small/train_1/train/dogs
Creating subset: validation, Category: cats
Destination directory: cats_vs_dogs_small/train_1/validation/cats
Creating subset: validation, Category: dogs
Destination directory: cats_vs_dogs_small/train_1/validation/dogs
Creating subset: test, Category: cats
Destination directory: cats_vs_dogs_small/train_1/test/cats
Creating subset: test, Category: dogs
Destination directory: cats_vs_dogs_small/train_1/test/dogs
```

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 1000 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
```

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,)
```

Double-click (or enter) to edit

Building the model

Model 1

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: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_1 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_2 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_3 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_4 (Conv2D)	(None, 7, 7, 256)	590,080
flatten (Flatten)	(None, 12544)	0
dropout (Dropout)	(None, 12544)	0
dense (Dense)	(None, 1)	12,545

Configuring the model for training

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

Fitting the model using a `Dataset`

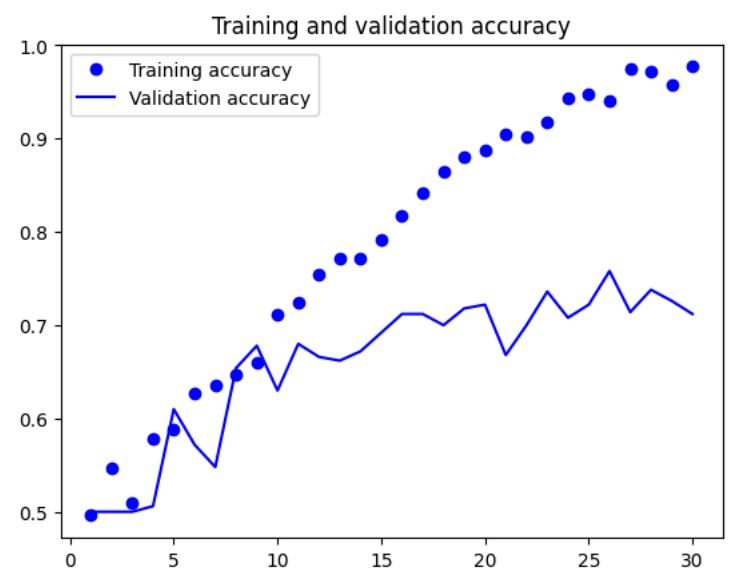
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model1 = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

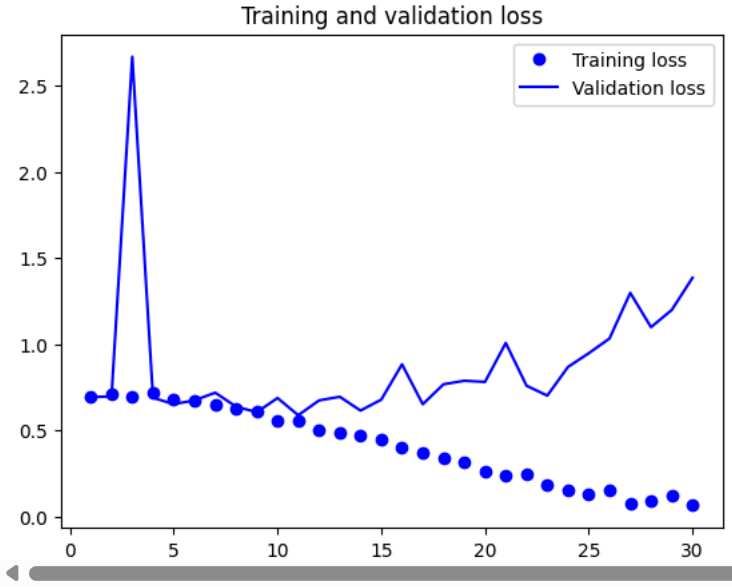
```
Epoch 1/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 126ms/step - accuracy: 0.4886 - loss: 0.7005 - val_accuracy: 0.5000 - val_loss: 0.6927
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5465 - loss: 0.7212 - val_accuracy: 0.5000 - val_loss: 0.6959
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.4924 - loss: 0.6935 - val_accuracy: 0.5000 - val_loss: 2.6693
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5716 - loss: 0.8152 - val_accuracy: 0.5060 - val_loss: 0.6882
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5719 - loss: 0.6857 - val_accuracy: 0.6100 - val_loss: 0.6516
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6105 - loss: 0.6924 - val_accuracy: 0.5720 - val_loss: 0.6736
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6105 - loss: 0.6534 - val_accuracy: 0.5480 - val_loss: 0.7180
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6520 - loss: 0.6254 - val_accuracy: 0.6540 - val_loss: 0.6372
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6512 - loss: 0.5983 - val_accuracy: 0.6780 - val_loss: 0.6046
```

```
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7065 - loss: 0.5643 - val_accuracy: 0.6300 - val_loss: 0.6869
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7337 - loss: 0.5330 - val_accuracy: 0.6800 - val_loss: 0.5872
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7375 - loss: 0.4985 - val_accuracy: 0.6660 - val_loss: 0.6730
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7559 - loss: 0.5148 - val_accuracy: 0.6620 - val_loss: 0.6943
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7499 - loss: 0.5016 - val_accuracy: 0.6720 - val_loss: 0.6141
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8001 - loss: 0.4364 - val_accuracy: 0.6920 - val_loss: 0.6777
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7864 - loss: 0.4122 - val_accuracy: 0.7120 - val_loss: 0.8828
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8226 - loss: 0.4062 - val_accuracy: 0.7120 - val_loss: 0.6509
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8442 - loss: 0.3937 - val_accuracy: 0.7000 - val_loss: 0.7669
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8791 - loss: 0.3205 - val_accuracy: 0.7180 - val_loss: 0.7875
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8728 - loss: 0.3058 - val_accuracy: 0.7220 - val_loss: 0.7808
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8847 - loss: 0.2661 - val_accuracy: 0.6680 - val_loss: 1.0068
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8866 - loss: 0.2728 - val_accuracy: 0.7000 - val_loss: 0.7586
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9279 - loss: 0.1649 - val_accuracy: 0.7360 - val_loss: 0.7010
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9549 - loss: 0.1297 - val_accuracy: 0.7080 - val_loss: 0.8677
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9307 - loss: 0.1464 - val_accuracy: 0.7220 - val_loss: 0.9471
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9447 - loss: 0.1445 - val_accuracy: 0.7580 - val_loss: 1.0328
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9662 - loss: 0.0934 - val_accuracy: 0.7140 - val_loss: 1.2979
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9705 - loss: 0.0873 - val_accuracy: 0.7380 - val_loss: 1.0985
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9678 - loss: 0.0899 - val_accuracy: 0.7260 - val_loss: 1.1985
```

Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
accuracy = Model1.history["accuracy"]
val_accuracy = Model1.history["val_accuracy"]
loss = Model1.history["loss"]
val_loss = Model1.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}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.6719 - loss: 0.6039
Test accuracy: 0.674
Test loss: 0.602
```

Using data augmentation

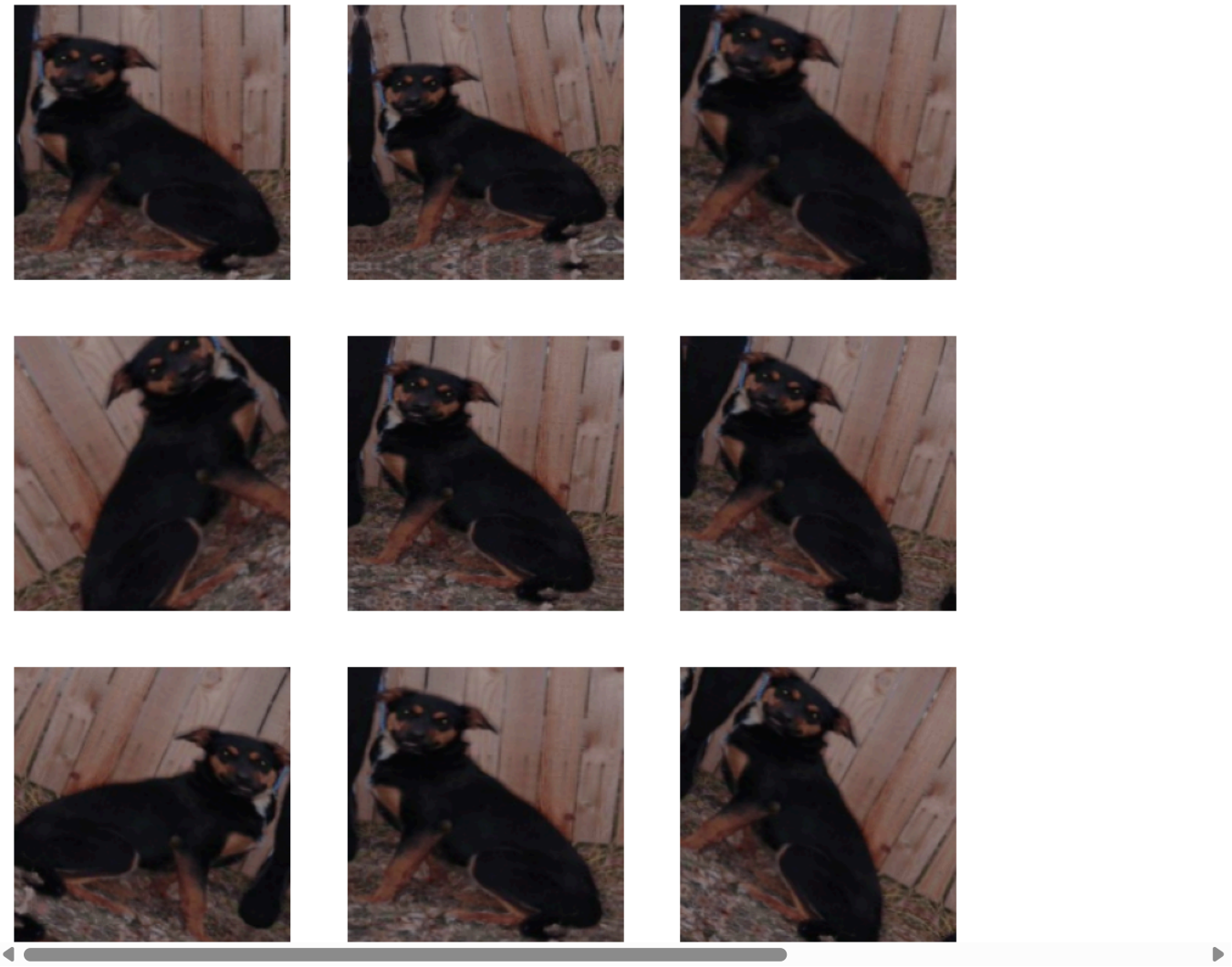
Define a data augmentation stage to add to an image model

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

Displaying some randomly augmented training images

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
```

```
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```



Model 2

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", padding="same")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu", padding="same")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu", padding="same")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu", padding="same")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu", padding="same")(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

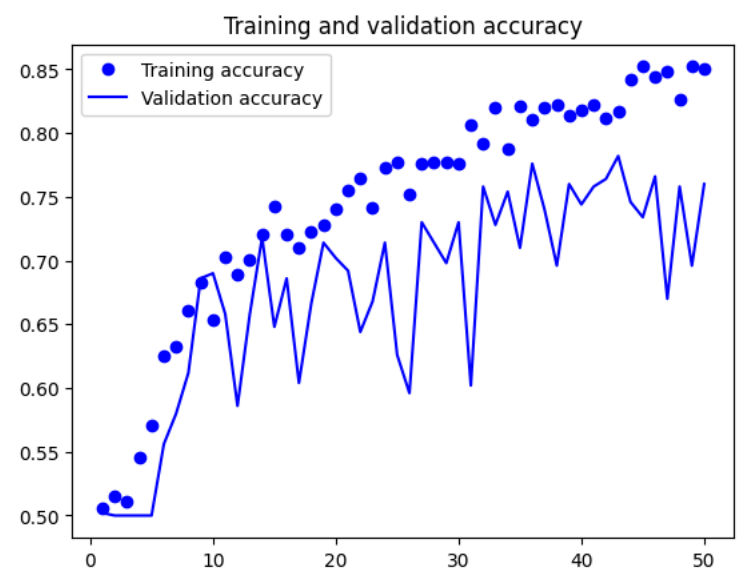
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
```

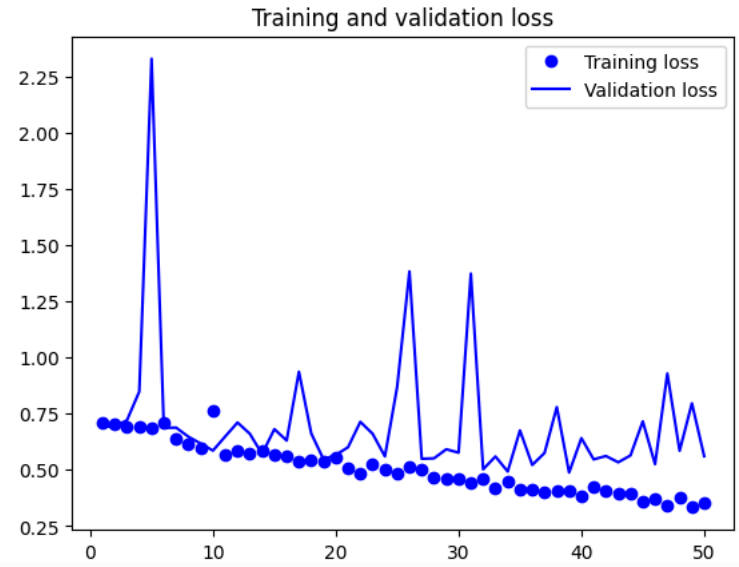
```
]
Model2 = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 35ms/step - accuracy: 0.5098 - loss: 0.7302 - val_accuracy: 0.5020 - val_loss: 0.6922
Epoch 2/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4983 - loss: 0.7051 - val_accuracy: 0.5000 - val_loss: 0.6927
Epoch 3/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4998 - loss: 0.6935 - val_accuracy: 0.5000 - val_loss: 0.7207
Epoch 4/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5354 - loss: 0.6995 - val_accuracy: 0.5000 - val_loss: 0.8479
Epoch 5/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5304 - loss: 0.7106 - val_accuracy: 0.5000 - val_loss: 2.3275
Epoch 6/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6170 - loss: 0.8709 - val_accuracy: 0.5560 - val_loss: 0.6865
Epoch 7/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6132 - loss: 0.6473 - val_accuracy: 0.5800 - val_loss: 0.6875
Epoch 8/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6533 - loss: 0.6262 - val_accuracy: 0.6120 - val_loss: 0.6471
Epoch 9/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6735 - loss: 0.6071 - val_accuracy: 0.6860 - val_loss: 0.6166
Epoch 10/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6507 - loss: 0.7279 - val_accuracy: 0.6900 - val_loss: 0.5854
Epoch 11/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6923 - loss: 0.5763 - val_accuracy: 0.6580 - val_loss: 0.6485
Epoch 12/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6597 - loss: 0.5971 - val_accuracy: 0.5860 - val_loss: 0.7102
Epoch 13/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6875 - loss: 0.5777 - val_accuracy: 0.6580 - val_loss: 0.6603
Epoch 14/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.7081 - loss: 0.5896 - val_accuracy: 0.7180 - val_loss: 0.5702
Epoch 15/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7251 - loss: 0.6169 - val_accuracy: 0.6480 - val_loss: 0.6810
Epoch 16/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7084 - loss: 0.5715 - val_accuracy: 0.6860 - val_loss: 0.6303
Epoch 17/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6819 - loss: 0.5560 - val_accuracy: 0.6040 - val_loss: 0.9359
Epoch 18/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7038 - loss: 0.5928 - val_accuracy: 0.6660 - val_loss: 0.6621
Epoch 19/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.6785 - loss: 0.5910 - val_accuracy: 0.7140 - val_loss: 0.5467
Epoch 20/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7496 - loss: 0.5431 - val_accuracy: 0.7020 - val_loss: 0.5661
Epoch 21/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7522 - loss: 0.5024 - val_accuracy: 0.6920 - val_loss: 0.6004
Epoch 22/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7657 - loss: 0.4744 - val_accuracy: 0.6440 - val_loss: 0.7133
Epoch 23/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7291 - loss: 0.5454 - val_accuracy: 0.6680 - val_loss: 0.6603
Epoch 24/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7466 - loss: 0.5376 - val_accuracy: 0.7140 - val_loss: 0.5597
Epoch 25/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7584 - loss: 0.5064 - val_accuracy: 0.6260 - val_loss: 0.8692
Epoch 26/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7137 - loss: 0.5974 - val_accuracy: 0.5960 - val_loss: 1.3820
Epoch 27/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7417 - loss: 0.6418 - val_accuracy: 0.7300 - val_loss: 0.5491
Epoch 28/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7537 - loss: 0.4925 - val_accuracy: 0.7140 - val_loss: 0.5510
Epoch 29/50
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7683 - loss: 0.4809 - val_accuracy: 0.6980 - val_loss: 0.5914
```

Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
accuracy = Model2.history["accuracy"]
val_accuracy = Model2.history["val_accuracy"]

loss = Model2.history["loss"]
val_loss = Model2.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_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7159 - loss: 0.6086
Test accuracy: 0.742
Test loss: 0.579
```

Model 3

```
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.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=512, 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"])



model.summary()
```

Model: "functional_6"

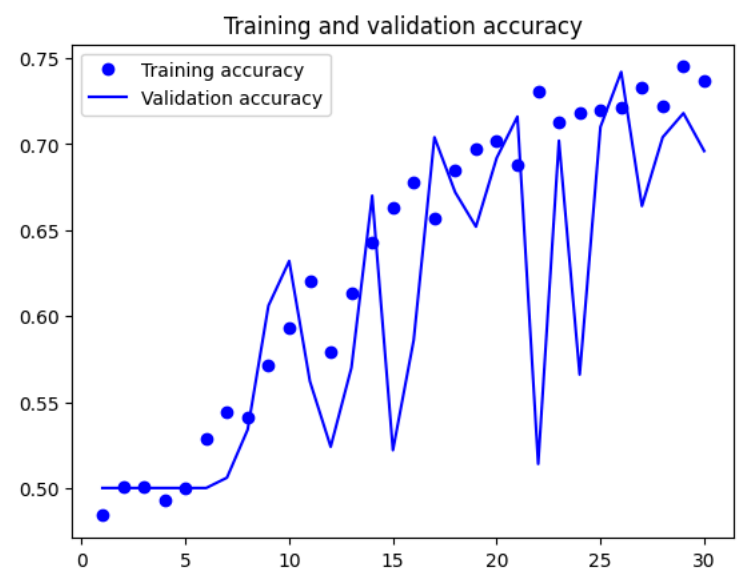
Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 180, 180, 3)	0
sequential (Sequential)	(None, 180, 180, 3)	0
rescaling_2 (Rescaling)	(None, 180, 180, 3)	0
conv2d_10 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_11 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_9 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_12 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_10 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_13 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_11 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_14 (Conv2D)	(None, 7, 7, 256)	590,080
max_pooling2d_12 (MaxPooling2D)	(None, 3, 3, 256)	0
conv2d_15 (Conv2D)	(None, 1, 1, 512)	1,180,160
flatten_2 (Flatten)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513

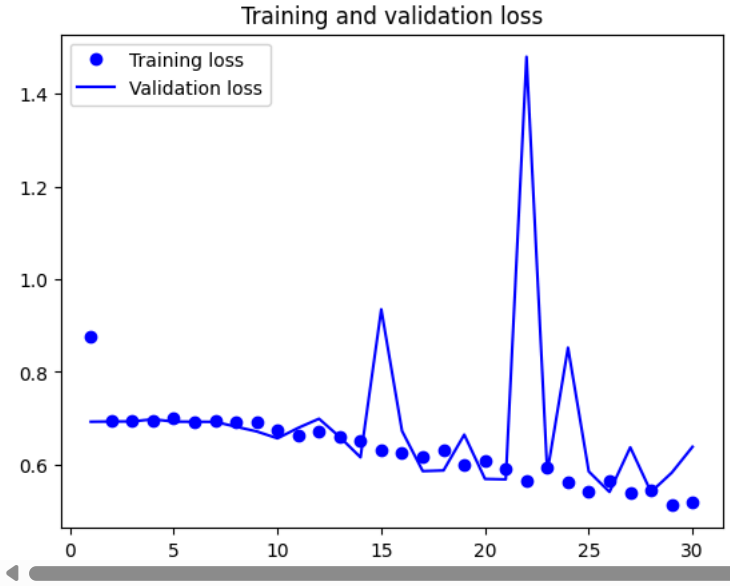
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model3 = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.5079 - loss: 1.1414 - val_accuracy: 0.5000 - val_loss: 0.6927
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5140 - loss: 0.6946 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5028 - loss: 0.6936 - val_accuracy: 0.5000 - val_loss: 0.6933
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4792 - loss: 0.6944 - val_accuracy: 0.5000 - val_loss: 0.6979
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4878 - loss: 0.7008 - val_accuracy: 0.5000 - val_loss: 0.6929
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5326 - loss: 0.6919 - val_accuracy: 0.5000 - val_loss: 0.6927
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.5508 - loss: 0.6927 - val_accuracy: 0.5060 - val_loss: 0.6923
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.5263 - loss: 0.6900 - val_accuracy: 0.5340 - val_loss: 0.6819
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.5767 - loss: 0.6874 - val_accuracy: 0.6060 - val_loss: 0.6717
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.5824 - loss: 0.6746 - val_accuracy: 0.6320 - val_loss: 0.6568
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6308 - loss: 0.6589 - val_accuracy: 0.5620 - val_loss: 0.6794
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5939 - loss: 0.6746 - val_accuracy: 0.5240 - val_loss: 0.6990
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6109 - loss: 0.6592 - val_accuracy: 0.5700 - val_loss: 0.6597
Epoch 14/30
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.6300 - loss: 0.6515 - val_accuracy: 0.6700 - val_loss: 0.6159
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6387 - loss: 0.6442 - val_accuracy: 0.5220 - val_loss: 0.9351
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6613 - loss: 0.6440 - val_accuracy: 0.5860 - val_loss: 0.6728
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.6433 - loss: 0.6281 - val_accuracy: 0.7040 - val_loss: 0.5861
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7010 - loss: 0.6194 - val_accuracy: 0.6720 - val_loss: 0.5875
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6813 - loss: 0.6122 - val_accuracy: 0.6520 - val_loss: 0.6647
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.7117 - loss: 0.6065 - val_accuracy: 0.6920 - val_loss: 0.5692
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.6825 - loss: 0.5858 - val_accuracy: 0.7160 - val_loss: 0.5683
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7228 - loss: 0.5749 - val_accuracy: 0.5140 - val_loss: 1.4800
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6894 - loss: 0.6754 - val_accuracy: 0.7020 - val_loss: 0.5829
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7118 - loss: 0.5700 - val_accuracy: 0.5660 - val_loss: 0.8524
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7202 - loss: 0.5377 - val_accuracy: 0.7100 - val_loss: 0.5853
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.7076 - loss: 0.5775 - val_accuracy: 0.7420 - val_loss: 0.5411
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7297 - loss: 0.5366 - val_accuracy: 0.6640 - val_loss: 0.6374
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7105 - loss: 0.5445 - val_accuracy: 0.7040 - val_loss: 0.5411
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7267 - loss: 0.5144 - val_accuracy: 0.7180 - val_loss: 0.5827
```

```
import matplotlib.pyplot as plt
accuracy = Model3.history["accuracy"]
val_accuracy = Model3.history["val_accuracy"]

loss = Model3.history["loss"]
val_loss = Model3.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()
```






```
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}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6887 - loss: 0.6122
Test accuracy: 0.692
Test loss: 0.611
```

Model 4

```
inputs= keras.Input(shape=(180,180,3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(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=512, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=1024, 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"])
```

```
model.summary()
```

Model: "functional_10"

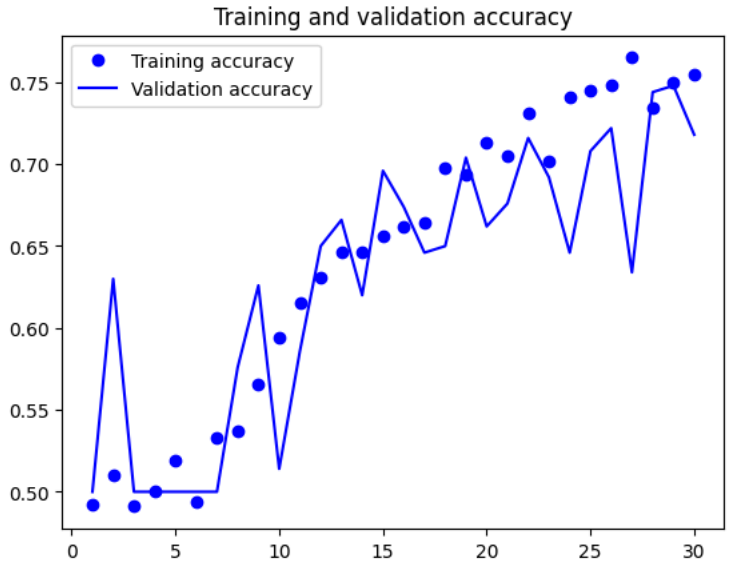
Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 180, 180, 3)	0
sequential (Sequential)	(None, 180, 180, 3)	0
rescaling_3 (Rescaling)	(None, 180, 180, 3)	0
conv2d_16 (Conv2D)	(None, 178, 178, 64)	1,792
max_pooling2d_13 (MaxPooling2D)	(None, 89, 89, 64)	0
conv2d_17 (Conv2D)	(None, 87, 87, 128)	73,856
max_pooling2d_14 (MaxPooling2D)	(None, 43, 43, 128)	0
conv2d_18 (Conv2D)	(None, 41, 41, 256)	295,168
max_pooling2d_15 (MaxPooling2D)	(None, 20, 20, 256)	0
conv2d_19 (Conv2D)	(None, 18, 18, 512)	1,180,160
max_pooling2d_16 (MaxPooling2D)	(None, 9, 9, 512)	0
conv2d_20 (Conv2D)	(None, 7, 7, 1024)	4,719,616
flatten_3 (Flatten)	(None, 50176)	0
dropout_3 (Dropout)	(None, 50176)	0
dense_3 (Dense)	(None, 1)	50,177

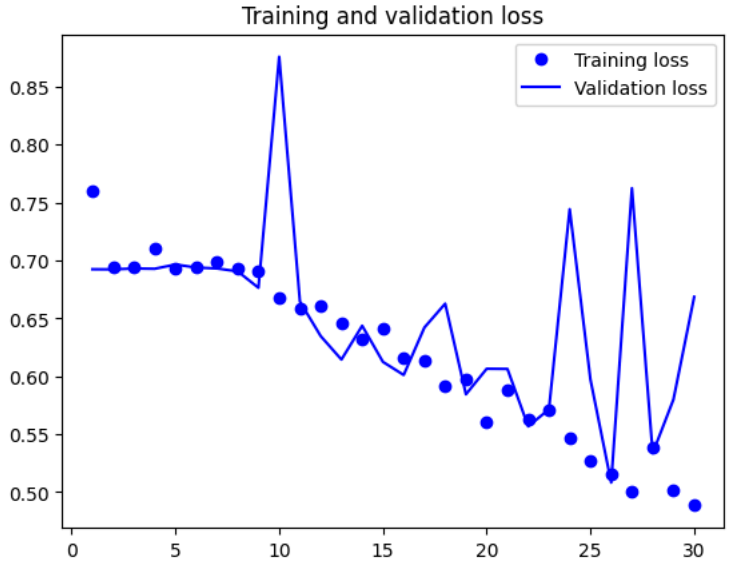
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model4 = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 49ms/step - accuracy: 0.5097 - loss: 0.8530 - val_accuracy: 0.5000 - val_loss: 0.6924
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.5252 - loss: 0.6941 - val_accuracy: 0.6300 - val_loss: 0.6923
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.4955 - loss: 0.6936 - val_accuracy: 0.5000 - val_loss: 0.6930
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.4999 - loss: 0.7193 - val_accuracy: 0.5000 - val_loss: 0.6928
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.5267 - loss: 0.6934 - val_accuracy: 0.5000 - val_loss: 0.6965
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.4917 - loss: 0.6960 - val_accuracy: 0.5000 - val_loss: 0.6939
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.5431 - loss: 0.7095 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.5281 - loss: 0.6947 - val_accuracy: 0.5760 - val_loss: 0.6906
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.5717 - loss: 0.6883 - val_accuracy: 0.6260 - val_loss: 0.6764
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.5929 - loss: 0.6712 - val_accuracy: 0.5140 - val_loss: 0.8759
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.5906 - loss: 0.6864 - val_accuracy: 0.5860 - val_loss: 0.6644
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.5999 - loss: 0.6880 - val_accuracy: 0.6500 - val_loss: 0.6347
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.6191 - loss: 0.6765 - val_accuracy: 0.6660 - val_loss: 0.6144
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.6404 - loss: 0.6475 - val_accuracy: 0.6200 - val_loss: 0.6436
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.6364 - loss: 0.6416 - val_accuracy: 0.6960 - val_loss: 0.6124
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.6449 - loss: 0.6269 - val_accuracy: 0.6740 - val_loss: 0.6011
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.6825 - loss: 0.6187 - val_accuracy: 0.6460 - val_loss: 0.6421
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.6739 - loss: 0.6216 - val_accuracy: 0.6500 - val_loss: 0.6627
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.6788 - loss: 0.6030 - val_accuracy: 0.7040 - val_loss: 0.5844
Epoch 20/30
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.6958 - loss: 0.5675 - val_accuracy: 0.6620 - val_loss: 0.6065
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.6785 - loss: 0.6180 - val_accuracy: 0.6760 - val_loss: 0.6063
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.7133 - loss: 0.5789 - val_accuracy: 0.7160 - val_loss: 0.5568
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.6808 - loss: 0.6216 - val_accuracy: 0.6920 - val_loss: 0.5714
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.7241 - loss: 0.5547 - val_accuracy: 0.6460 - val_loss: 0.7442
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.7354 - loss: 0.5624 - val_accuracy: 0.7080 - val_loss: 0.5972
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.7288 - loss: 0.5539 - val_accuracy: 0.7220 - val_loss: 0.5082
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.7504 - loss: 0.5249 - val_accuracy: 0.6340 - val_loss: 0.7624
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.6851 - loss: 0.6548 - val_accuracy: 0.7440 - val_loss: 0.5335
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.7148 - loss: 0.5360 - val_accuracy: 0.7480 - val_loss: 0.5799
```

```
import matplotlib.pyplot as plt
accuracy = Model4.history["accuracy"]
val_accuracy = Model4.history["val_accuracy"]

loss = Model4.history["loss"]
val_loss = Model4.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()
```



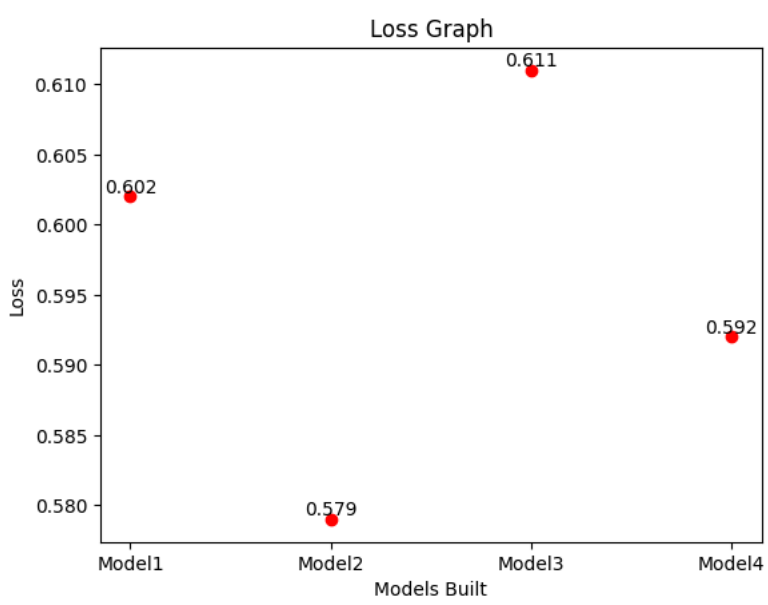



```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

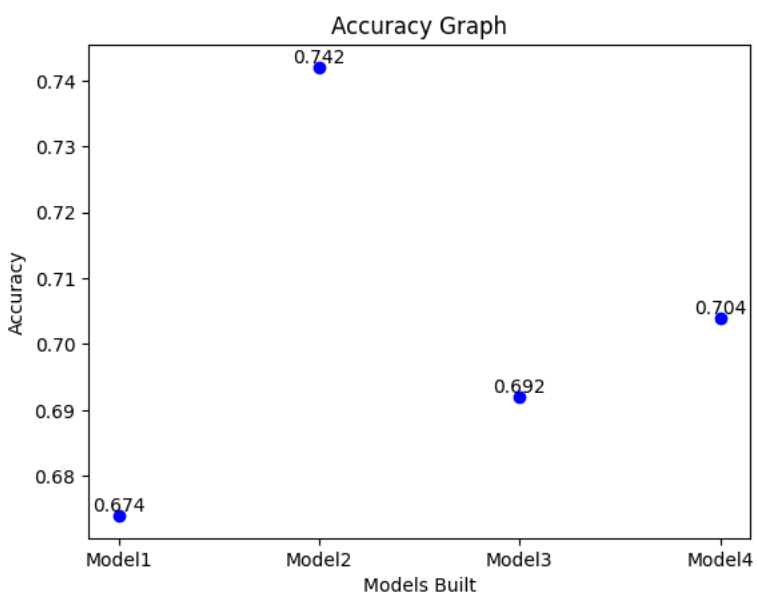
```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6895 - loss: 0.6043
Test accuracy: 0.704
Test loss: 0.592
```

```
Model1 = (0.602, 0.674)
Model2 = (0.579, 0.742)
Model3 = (0.611, 0.692)
Model4 = (0.592, 0.704)
```

```
Models= ("Model1","Model2","Model3","Model4")
Loss= (Model1[0],Model2[0],Model3[0],Model4[0])
Accuracy= (Model1[1],Model2[1],Model3[1],Model4[1])
plt.scatter(Models,Loss,color="red")
plt.title("Loss Graph")
plt.ylabel("Loss")
plt.xlabel("Models Built")

for (xi, yi) in zip(Models,Loss):
  plt.text(xi, yi, yi, va='bottom', ha='center')
plt.show()
```




```
plt.scatter(Models,Accuracy,color="blue")
plt.title("Accuracy Graph")
plt.ylabel("Accuracy")
plt.xlabel("Models Built")

for (xi, yi) in zip(Models,Accuracy):
  plt.text(xi, yi, yi, va='bottom', ha='center')
plt.show()
```



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?

```
import os
import shutil
from pathlib import Path


original_data_dir = pathlib.Path("drive/MyDrive/Colab Notebooks/cats_vs_dogs_small")
new_base_dir = Path("cats_vs_dogs_small/train2")

def make_subset(subset_name, start_index, end_index):
    for category in ("cats", "dogs"):
        src_dir = original_data_dir /subset_name/ category
        dst_dir = new_base_dir / subset_name / category

        print(f"Creating subset: {subset_name}, Category: {category}")
        print(f"Destination directory: {dst_dir}")

        os.makedirs(dst_dir, exist_ok=True)
```

```
        fnames = [f"{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            src_file = src_dir / fname
            dst_file = dst_dir / fname


            if src_file.exists():
                shutil.copyfile(src_file, dst_file)
            else:
                print(f"Warning: Source file {src_file} not found.")


make_subset("train", start_index=0, end_index=700)
make_subset("validation", start_index=1000, end_index=1250)
make_subset("test", start_index=1500, end_index=1750)
```

```
Creating subset: train, Category: cats
Destination directory: cats_vs_dogs_small/train2/train/cats
Creating subset: train, Category: dogs
Destination directory: cats_vs_dogs_small/train2/train/dogs
Creating subset: validation, Category: cats
Destination directory: cats_vs_dogs_small/train2/validation/cats
Creating subset: validation, Category: dogs
Destination directory: cats_vs_dogs_small/train2/validation/dogs
Creating subset: test, Category: cats
Destination directory: cats_vs_dogs_small/train2/test/cats
Creating subset: test, Category: dogs
Destination directory: cats_vs_dogs_small/train2/test/dogs
```

```
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 1400 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
```

```
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,)
```

```
data_augmentation1 = keras.Sequential(
  [
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.15),
    layers.RandomZoom(0.25)
  ]
)



plt.figure(figsize=(10, 10))
for images,_ in train_dataset.take(1):
  for i in range(6):
    augmented_images = data_augmentation1(images)
    ax= plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

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 = data_augmentation1(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.summary()
```

```
Model: "functional_15"
```

Layer (type)	Output Shape	Param #
input_layer_6 (InputLayer)	(None, 180, 180, 3)	0
sequential_1 (Sequential)	(None, 180, 180, 3)	0
rescaling_4 (Rescaling)	(None, 180, 180, 3)	0
conv2d_21 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_17 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_22 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_18 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_23 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_19 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_24 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_20 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_25 (Conv2D)	(None, 7, 7, 256)	590,080
flatten_4 (Flatten)	(None, 12544)	0
dropout_4 (Dropout)	(None, 12544)	0
dense_4 (Dense)	(None, 1)	12,545

```
Total params: 991,041 (3.78 MB)
Trainable params: 991,041 (3.78 MB)
```

Configuring the model for training

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

Fitting the model using a Dataset

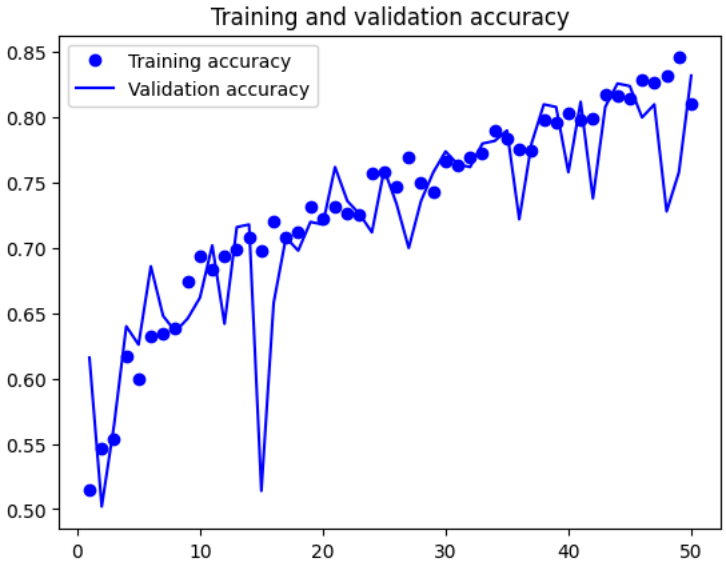
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model5 = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

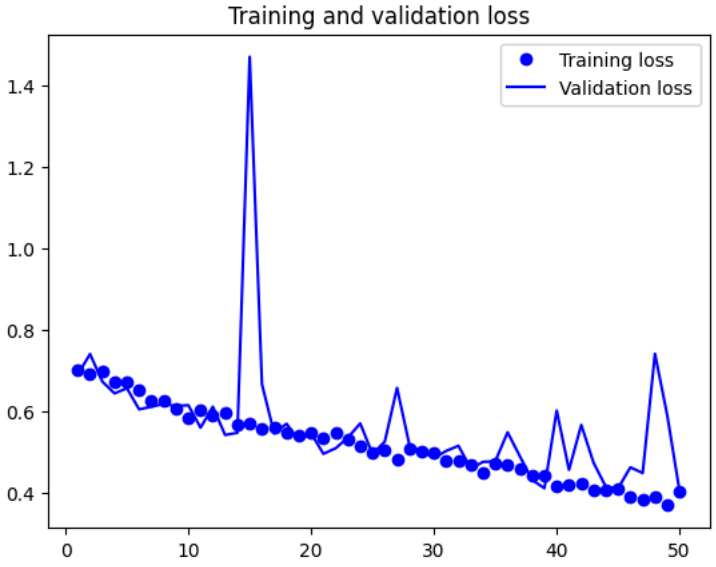
```
Epoch 1/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.5133 - loss: 0.7142 - val_accuracy: 0.6160 - val_loss: 0.6902
Epoch 2/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.5412 - loss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.7407
Epoch 3/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.5239 - loss: 0.7107 - val_accuracy: 0.5640 - val_loss: 0.6732
Epoch 4/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.6062 - loss: 0.6718 - val_accuracy: 0.6400 - val_loss: 0.6441
Epoch 5/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6003 - loss: 0.6672 - val_accuracy: 0.6260 - val_loss: 0.6569
Epoch 6/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6225 - loss: 0.6628 - val_accuracy: 0.6860 - val_loss: 0.6050
Epoch 7/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6005 - loss: 0.6406 - val_accuracy: 0.6480 - val_loss: 0.6115
Epoch 8/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6436 - loss: 0.6258 - val_accuracy: 0.6360 - val_loss: 0.6183
Epoch 9/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6823 - loss: 0.6026 - val_accuracy: 0.6460 - val_loss: 0.6129
Epoch 10/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6801 - loss: 0.5958 - val_accuracy: 0.6620 - val_loss: 0.6153
Epoch 11/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6745 - loss: 0.6098 - val_accuracy: 0.7020 - val_loss: 0.5600
Epoch 12/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6815 - loss: 0.5956 - val_accuracy: 0.6420 - val_loss: 0.6116
Epoch 13/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6956 - loss: 0.5908 - val_accuracy: 0.7160 - val_loss: 0.5421
Epoch 14/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7034 - loss: 0.5734 - val_accuracy: 0.7180 - val_loss: 0.5476
Epoch 15/50
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6850 - loss: 0.5665 - val_accuracy: 0.5140 - val_loss: 1.4698
Epoch 16/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7124 - loss: 0.5923 - val_accuracy: 0.6580 - val_loss: 0.6666
Epoch 17/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6949 - loss: 0.5690 - val_accuracy: 0.7080 - val_loss: 0.5457
Epoch 18/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7084 - loss: 0.5425 - val_accuracy: 0.6980 - val_loss: 0.5699
Epoch 19/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7380 - loss: 0.5332 - val_accuracy: 0.7200 - val_loss: 0.5283
Epoch 20/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7251 - loss: 0.5463 - val_accuracy: 0.7180 - val_loss: 0.5465
Epoch 21/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7420 - loss: 0.5307 - val_accuracy: 0.7620 - val_loss: 0.4961
Epoch 22/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7337 - loss: 0.5512 - val_accuracy: 0.7360 - val_loss: 0.5096
Epoch 23/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7061 - loss: 0.5379 - val_accuracy: 0.7260 - val_loss: 0.5358
Epoch 24/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7569 - loss: 0.5019 - val_accuracy: 0.7120 - val_loss: 0.5705
Epoch 25/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7479 - loss: 0.5103 - val_accuracy: 0.7600 - val_loss: 0.4957
Epoch 26/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7420 - loss: 0.5072 - val_accuracy: 0.7340 - val_loss: 0.5258
Epoch 27/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7769 - loss: 0.4859 - val_accuracy: 0.7000 - val_loss: 0.6577
Epoch 28/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7455 - loss: 0.5102 - val_accuracy: 0.7360 - val_loss: 0.5185
Epoch 29/50
```

Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
accuracy = Model5.history["accuracy"]
val_accuracy = Model5.history["val_accuracy"]
loss = Model5.history["loss"]
val_loss = Model5.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}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7834 - loss: 0.5456
Test accuracy: 0.796
Test loss: 0.514
```

Model 6

```
inputs= keras.Input(shape=(180, 180, 3))
x = data_augmentation1(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(filters=64, kernel_size=3, strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(filters=128, kernel_size=3, strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(filters=256, kernel_size=3, strides=2, activation="relu", padding="same")(x)
x = layers.Conv2D(filters=256, kernel_size=3, strides=2, activation="relu", padding="same")(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: "functional_19"

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, 180, 180, 3)	0
sequential_1 (Sequential)	(None, 180, 180, 3)	0
rescaling_5 (Rescaling)	(None, 180, 180, 3)	0
conv2d_26 (Conv2D)	(None, 90, 90, 32)	896
conv2d_27 (Conv2D)	(None, 45, 45, 64)	18,496
conv2d_28 (Conv2D)	(None, 23, 23, 128)	73,856
conv2d_29 (Conv2D)	(None, 12, 12, 256)	295,168
conv2d_30 (Conv2D)	(None, 6, 6, 256)	590,080
flatten_5 (Flatten)	(None, 9216)	0
dropout_5 (Dropout)	(None, 9216)	0
dense_5 (Dense)	(None, 1)	9,217

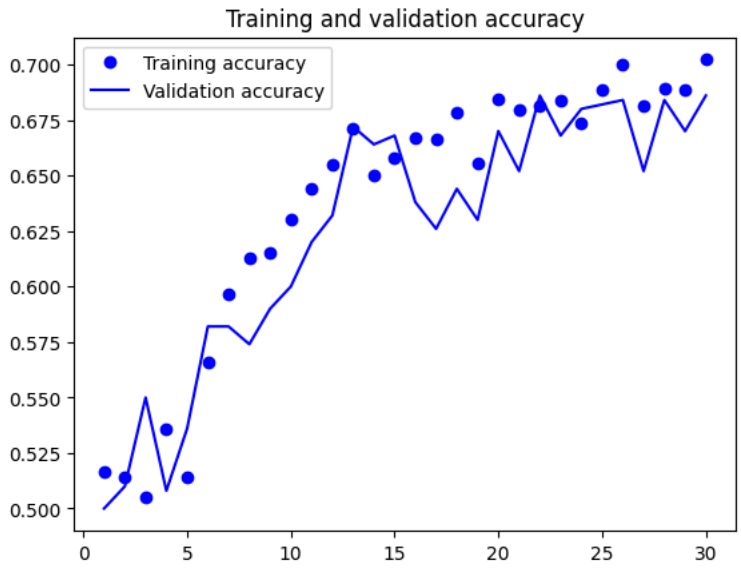
Total params: 987,713 (3.77 MB)
Trainable params: 987,713 (3.77 MB)

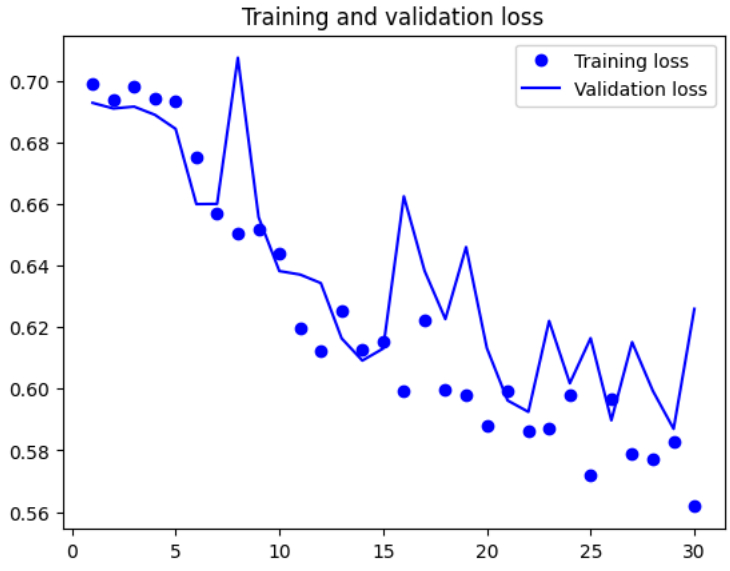
```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model6 = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.5435 - loss: 0.7068 - val_accuracy: 0.5000 - val_loss: 0.6928
Epoch 2/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5209 - loss: 0.6937 - val_accuracy: 0.5100 - val_loss: 0.6910
Epoch 3/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5019 - loss: 0.6947 - val_accuracy: 0.5500 - val_loss: 0.6916
Epoch 4/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5321 - loss: 0.6943 - val_accuracy: 0.5080 - val_loss: 0.6889
Epoch 5/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.4995 - loss: 0.6944 - val_accuracy: 0.5360 - val_loss: 0.6844
Epoch 6/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5525 - loss: 0.6839 - val_accuracy: 0.5820 - val_loss: 0.6600
Epoch 7/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5790 - loss: 0.6592 - val_accuracy: 0.5820 - val_loss: 0.6600
Epoch 8/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6001 - loss: 0.6576 - val_accuracy: 0.5740 - val_loss: 0.7075
Epoch 9/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5876 - loss: 0.6749 - val_accuracy: 0.5900 - val_loss: 0.6557
Epoch 10/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6144 - loss: 0.6434 - val_accuracy: 0.6000 - val_loss: 0.6382
Epoch 11/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6359 - loss: 0.6248 - val_accuracy: 0.6200 - val_loss: 0.6371
Epoch 12/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6345 - loss: 0.6239 - val_accuracy: 0.6320 - val_loss: 0.6343
Epoch 13/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6392 - loss: 0.6397 - val_accuracy: 0.6720 - val_loss: 0.6164
Epoch 14/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6305 - loss: 0.6158 - val_accuracy: 0.6640 - val_loss: 0.6091
Epoch 15/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6516 - loss: 0.6125 - val_accuracy: 0.6680 - val_loss: 0.6130
Epoch 16/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6351 - loss: 0.6229 - val_accuracy: 0.6380 - val_loss: 0.6625
Epoch 17/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6585 - loss: 0.6381 - val_accuracy: 0.6260 - val_loss: 0.6383
Epoch 18/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6520 - loss: 0.6219 - val_accuracy: 0.6440 - val_loss: 0.6226
Epoch 19/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6425 - loss: 0.6049 - val_accuracy: 0.6300 - val_loss: 0.6460
Epoch 20/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6557 - loss: 0.6083 - val_accuracy: 0.6700 - val_loss: 0.6132
Epoch 21/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6705 - loss: 0.6080 - val_accuracy: 0.6520 - val_loss: 0.5962
Epoch 22/30
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6571 - loss: 0.5931 - val_accuracy: 0.6860 - val_loss: 0.5925
Epoch 23/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6724 - loss: 0.5991 - val_accuracy: 0.6680 - val_loss: 0.6220
Epoch 24/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6568 - loss: 0.6106 - val_accuracy: 0.6800 - val_loss: 0.6018
Epoch 25/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6560 - loss: 0.5893 - val_accuracy: 0.6820 - val_loss: 0.6164
Epoch 26/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7010 - loss: 0.6471 - val_accuracy: 0.6840 - val_loss: 0.5897
Epoch 27/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6674 - loss: 0.5822 - val_accuracy: 0.6520 - val_loss: 0.6151
Epoch 28/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6641 - loss: 0.5861 - val_accuracy: 0.6840 - val_loss: 0.5993
Epoch 29/30
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6722 - loss: 0.6021 - val_accuracy: 0.6700 - val_loss: 0.5870
```

```
import matplotlib.pyplot as plt
accuracy = Model6.history["accuracy"]
val_accuracy = Model6.history["val_accuracy"]
loss = Model6.history["loss"]
val_loss = Model6.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()
```





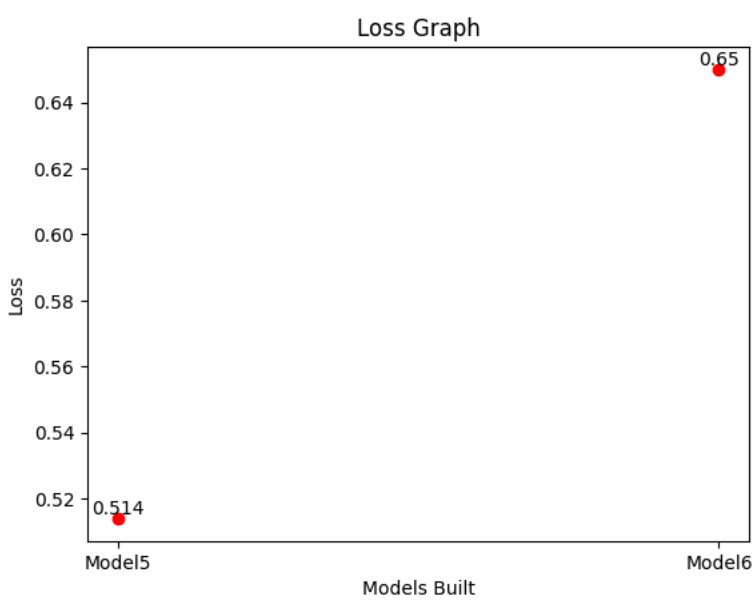

```
test_model = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
```

```
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

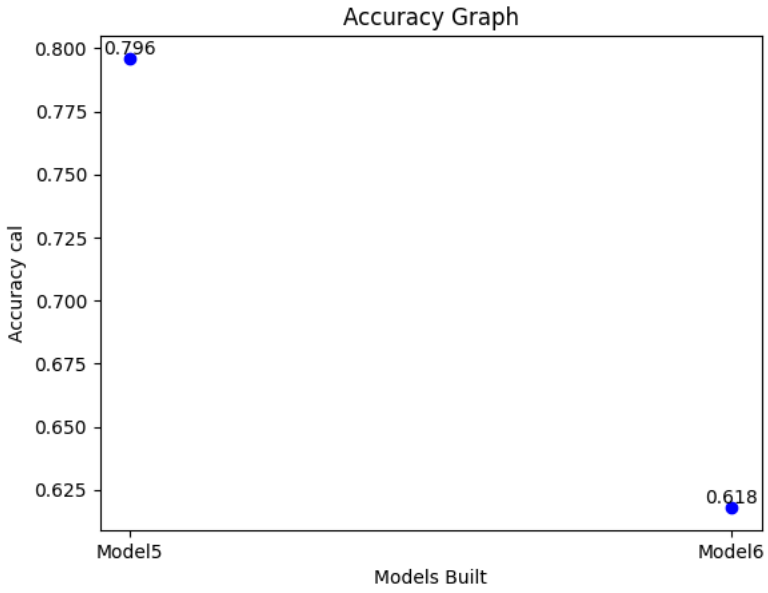
```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6096 - loss: 0.6549
Test accuracy: 0.618
Test loss: 0.650
```

```
Model5 = (0.514, 0.796)
Model6 = (0.650, 0.618)
```

```
Models= ("Model5","Model6")
Loss= (Model5[0], Model6[0])
Accuracy= (Model5[1], Model6[1])

plt.scatter(Models, Loss, color="red")
plt.title("Loss Graph")
plt.xlabel("Models Built")
plt.ylabel("Loss")

for (xi, yi) in zip(Models,Loss):
  plt.text(xi, yi, yi, va='bottom', ha='center')
plt.show()
```




```
plt.scatter(Models, Accuracy, color="blue")
plt.title("Accuracy Graph")
plt.xlabel("Models Built")
plt.ylabel("Accuracy cal")

for (xi, yi) in zip(Models,Accuracy):
  plt.text(xi, yi, yi, va='bottom', ha='center')
plt.show()
```



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.

Training has 1800 samples, test has 500 samples and validation has 500 samples.

```
import os
import shutil
from pathlib import Path

original_data_dir = pathlib.Path("drive/MyDrive/Colab Notebooks/cats_vs_dogs_small")
new_base_dir = Path("cats_vs_dogs_small/train3")

def make_subset(subset_name, start_index, end_index):
    for category in ("cats", "dogs"):
        src_dir = original_data_dir /subset_name/ category
        dst_dir = new_base_dir / subset_name / category

        print(f"Creating subset: {subset_name}, Category: {category}")
        print(f"Destination directory: {dst_dir}")

        os.makedirs(dst_dir, exist_ok=True)

        fnames = [f"{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            src_file = src_dir / fname
            dst_file = dst_dir / fname


            if src_file.exists():
                shutil.copyfile(src_file, dst_file)
            else:
                print(f"Warning: Source file {src_file} not found.")

make_subset("train", start_index=0, end_index=900)
make_subset("validation", start_index=1200, end_index=1450)
make_subset("test", start_index=1700, end_index=1950)
```

Show hidden output

```
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)
```

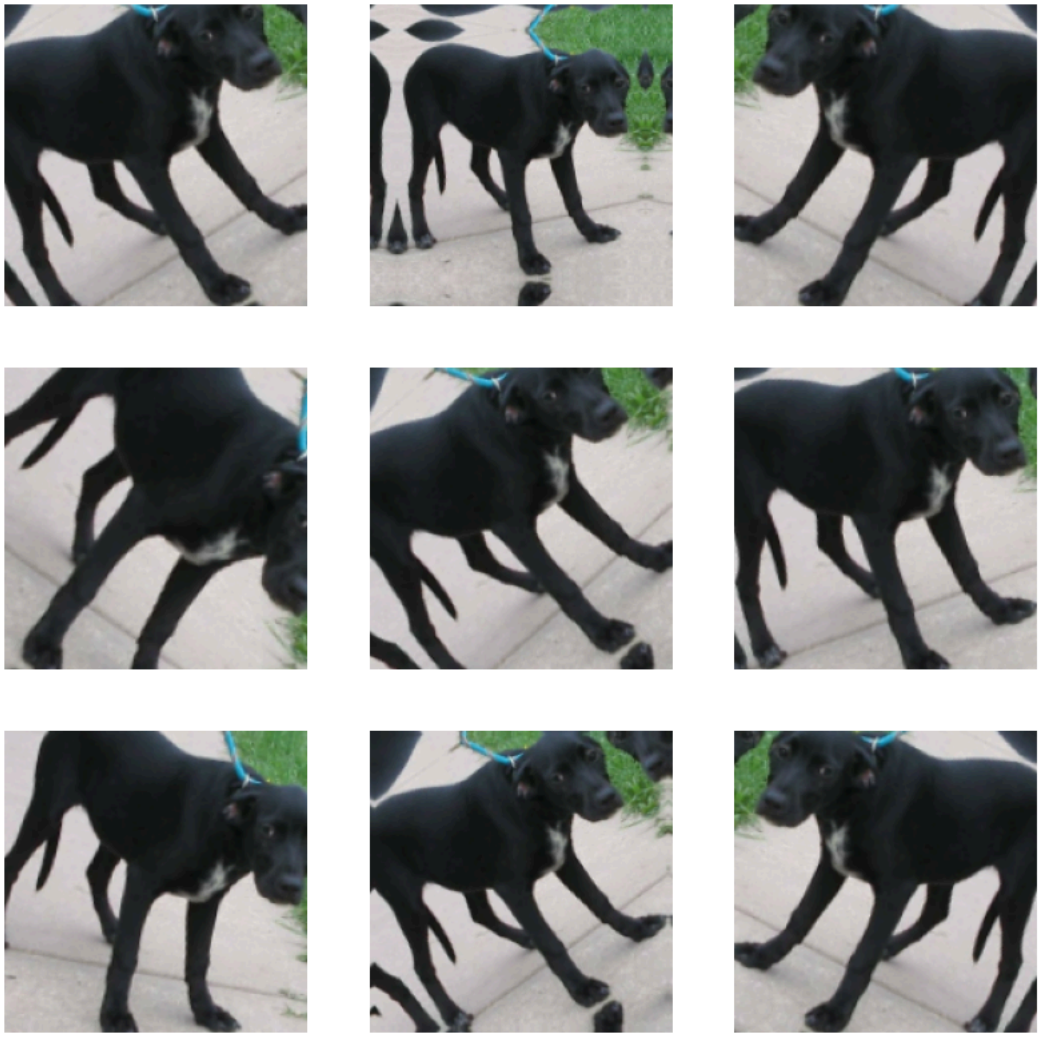
```
Found 1800 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
```

```
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,)
```

```
data_augmentation2 = keras.Sequential(
  [
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.15),
    layers.RandomZoom(0.25)
  ]
)
```

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation2(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```



Model 7

```
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: "functional_24"

Layer (type)	Output Shape	Param #
input_layer_9 (InputLayer)	(None, 180, 180, 3)	0
rescaling_6 (Rescaling)	(None, 180, 180, 3)	0
conv2d_31 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_21 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_32 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_22 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_33 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_23 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_34 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_24 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_35 (Conv2D)	(None, 7, 7, 256)	590,080
flatten_6 (Flatten)	(None, 12544)	0
dropout_6 (Dropout)	(None, 12544)	0
dense_6 (Dense)	(None, 1)	12,545

Total params: 991,041 (3.78 MB)
Trainable params: 991,041 (3.78 MB)

Configuring the model for training

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

Fitting the model using a Dataset

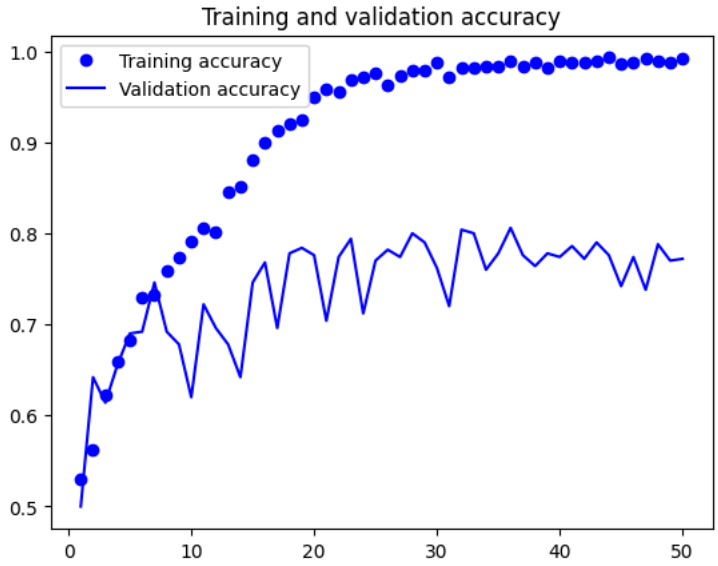
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model7 = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

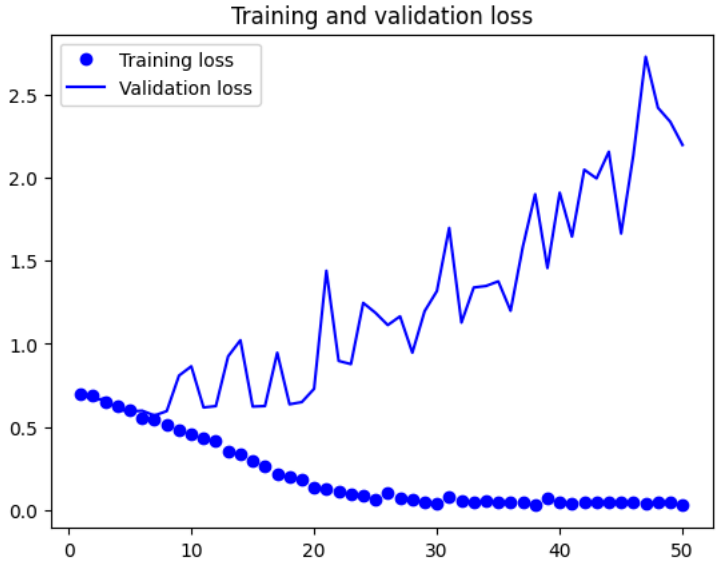
```
Epoch 1/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 6s 51ms/step - accuracy: 0.5297 - loss: 0.6962 - val_accuracy: 0.5000 - val_loss: 0.7021
Epoch 2/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5291 - loss: 0.6920 - val_accuracy: 0.6420 - val_loss: 0.6591
Epoch 3/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.6354 - loss: 0.6498 - val_accuracy: 0.6140 - val_loss: 0.6706
Epoch 4/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6461 - loss: 0.6415 - val_accuracy: 0.6560 - val_loss: 0.6024
Epoch 5/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6781 - loss: 0.6017 - val_accuracy: 0.6900 - val_loss: 0.5920
Epoch 6/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7268 - loss: 0.5460 - val_accuracy: 0.6920 - val_loss: 0.5968
Epoch 7/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.7242 - loss: 0.5372 - val_accuracy: 0.7460 - val_loss: 0.5679
```

```
Epoch 8/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7517 - loss: 0.5093 - val_accuracy: 0.6920 - val_loss: 0.5943
Epoch 9/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7578 - loss: 0.5000 - val_accuracy: 0.6780 - val_loss: 0.8083
Epoch 10/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7831 - loss: 0.4835 - val_accuracy: 0.6200 - val_loss: 0.8643
Epoch 11/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7892 - loss: 0.4861 - val_accuracy: 0.7220 - val_loss: 0.6165
Epoch 12/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7803 - loss: 0.4339 - val_accuracy: 0.6960 - val_loss: 0.6244
Epoch 13/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8426 - loss: 0.3671 - val_accuracy: 0.6780 - val_loss: 0.9244
Epoch 14/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8388 - loss: 0.3733 - val_accuracy: 0.6420 - val_loss: 1.0216
Epoch 15/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8679 - loss: 0.3271 - val_accuracy: 0.7460 - val_loss: 0.6219
Epoch 16/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8981 - loss: 0.2777 - val_accuracy: 0.7680 - val_loss: 0.6248
Epoch 17/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9071 - loss: 0.2147 - val_accuracy: 0.6960 - val_loss: 0.9463
Epoch 18/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9015 - loss: 0.2443 - val_accuracy: 0.7780 - val_loss: 0.6355
Epoch 19/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9148 - loss: 0.2101 - val_accuracy: 0.7840 - val_loss: 0.6488
Epoch 20/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9452 - loss: 0.1370 - val_accuracy: 0.7760 - val_loss: 0.7287
Epoch 21/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9504 - loss: 0.1369 - val_accuracy: 0.7040 - val_loss: 1.4398
Epoch 22/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9312 - loss: 0.1594 - val_accuracy: 0.7740 - val_loss: 0.8968
Epoch 23/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9666 - loss: 0.0917 - val_accuracy: 0.7940 - val_loss: 0.8776
Epoch 24/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9727 - loss: 0.0841 - val_accuracy: 0.7120 - val_loss: 1.2464
Epoch 25/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9683 - loss: 0.0851 - val_accuracy: 0.7700 - val_loss: 1.1864
Epoch 26/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9542 - loss: 0.1214 - val_accuracy: 0.7820 - val_loss: 1.1127
Epoch 27/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9754 - loss: 0.0581 - val_accuracy: 0.7740 - val_loss: 1.1655
Epoch 28/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9828 - loss: 0.0531 - val_accuracy: 0.8000 - val_loss: 0.9463
Epoch 29/50
```

Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
accuracy = Model7.history["accuracy"]
val_accuracy = Model7.history["val_accuracy"]
loss = Model7.history["loss"]
val_loss = Model7.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}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.7359 - loss: 0.5843
Test accuracy: 0.744
Test loss: 0.573
```

Model 8

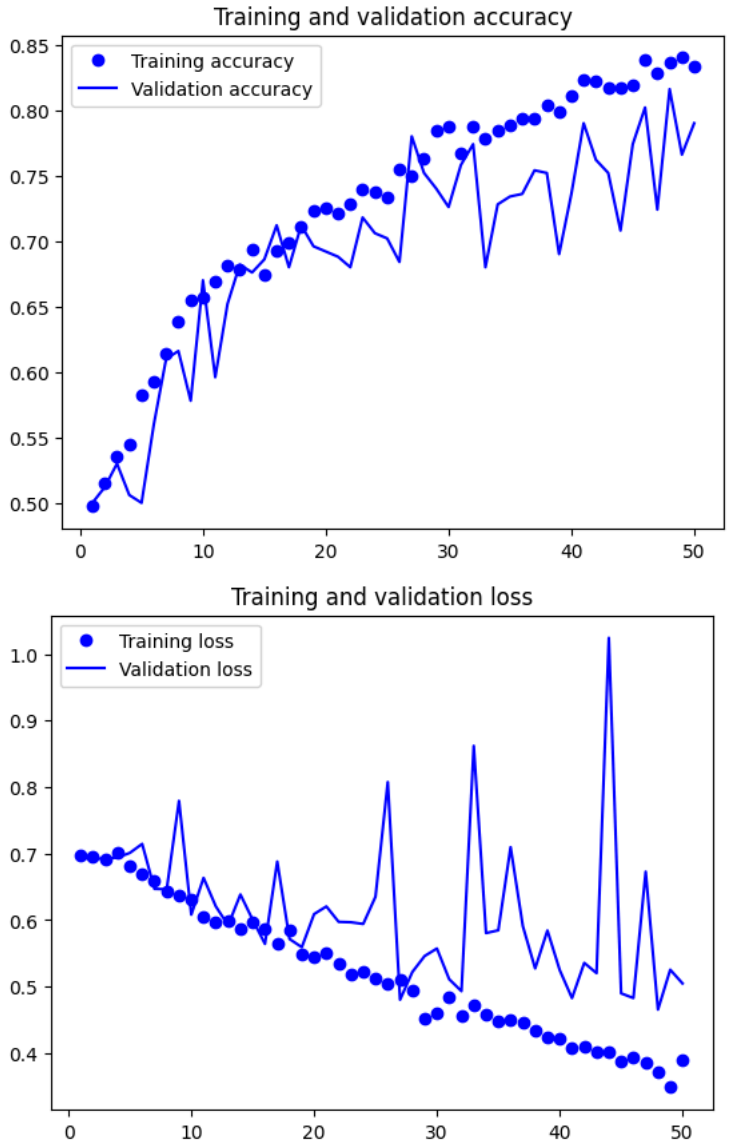
```
inputs= keras.Input(shape=(180, 180, 3))
x = data_augmentation2(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2,strides=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_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model8 = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.5197 - loss: 0.7029 - val_accuracy: 0.5000 - val_loss: 0.6934
Epoch 2/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.4993 - loss: 0.6957 - val_accuracy: 0.5120 - val_loss: 0.6924
Epoch 3/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5352 - loss: 0.6937 - val_accuracy: 0.5300 - val_loss: 0.6925
Epoch 4/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5166 - loss: 0.6962 - val_accuracy: 0.5060 - val_loss: 0.6941
Epoch 5/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5578 - loss: 0.6838 - val_accuracy: 0.5000 - val_loss: 0.7011
Epoch 6/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5761 - loss: 0.6757 - val_accuracy: 0.5600 - val_loss: 0.7146
Epoch 7/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6085 - loss: 0.6628 - val_accuracy: 0.6100 - val_loss: 0.6470
Epoch 8/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6275 - loss: 0.6481 - val_accuracy: 0.6160 - val_loss: 0.6465
Epoch 9/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6497 - loss: 0.6447 - val_accuracy: 0.5780 - val_loss: 0.7793
Epoch 10/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6586 - loss: 0.6145 - val_accuracy: 0.6700 - val_loss: 0.6082
Epoch 11/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6687 - loss: 0.6024 - val_accuracy: 0.5960 - val_loss: 0.6635
Epoch 12/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6906 - loss: 0.5879 - val_accuracy: 0.6520 - val_loss: 0.6206
Epoch 13/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6814 - loss: 0.5945 - val_accuracy: 0.6820 - val_loss: 0.5922
Epoch 14/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6872 - loss: 0.5913 - val_accuracy: 0.6760 - val_loss: 0.6386
Epoch 15/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6605 - loss: 0.5957 - val_accuracy: 0.6860 - val_loss: 0.6005
Epoch 16/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6920 - loss: 0.5886 - val_accuracy: 0.7120 - val_loss: 0.5643
Epoch 17/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6897 - loss: 0.5638 - val_accuracy: 0.6800 - val_loss: 0.6880
Epoch 18/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7164 - loss: 0.5686 - val_accuracy: 0.7120 - val_loss: 0.5716
Epoch 19/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7155 - loss: 0.5514 - val_accuracy: 0.6960 - val_loss: 0.5593
Epoch 20/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7409 - loss: 0.5322 - val_accuracy: 0.6920 - val_loss: 0.6091
Epoch 21/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7162 - loss: 0.5387 - val_accuracy: 0.6880 - val_loss: 0.6206
Epoch 22/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7206 - loss: 0.5427 - val_accuracy: 0.6800 - val_loss: 0.5974
Epoch 23/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7268 - loss: 0.5342 - val_accuracy: 0.7180 - val_loss: 0.5968
Epoch 24/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7446 - loss: 0.5103 - val_accuracy: 0.7060 - val_loss: 0.5943
Epoch 25/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7397 - loss: 0.4966 - val_accuracy: 0.7020 - val_loss: 0.6352
Epoch 26/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7460 - loss: 0.5014 - val_accuracy: 0.6840 - val_loss: 0.8076
Epoch 27/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7409 - loss: 0.5614 - val_accuracy: 0.7800 - val_loss: 0.4803
Epoch 28/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7767 - loss: 0.4822 - val_accuracy: 0.7520 - val_loss: 0.5221
Epoch 29/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7977 - loss: 0.4340 - val_accuracy: 0.7400 - val_loss: 0.5461
```

```
import matplotlib.pyplot as plt
accuracy = Model8.history["accuracy"]
val_accuracy = Model8.history["val_accuracy"]
loss = Model8.history["loss"]
val_loss = Model8.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_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8190 - loss: 0.5048
Test accuracy: 0.814
Test loss: 0.481
```

Model 9

```
inputs= keras.Input(shape=(180, 180, 3))
x = data_augmentation2(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2, padding="same")(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2, padding="same")(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, strides=2, padding="same")(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2,strides=2, padding="same")(x)
x = layers.Conv2D(filters=512, 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"])


model.summary()
```

Model: "functional_29"

Layer (type)	Output Shape	Param #
input_layer_11 (InputLayer)	(None, 180, 180, 3)	0
sequential_2 (Sequential)	(None, 180, 180, 3)	0
rescaling_8 (Rescaling)	(None, 180, 180, 3)	0
conv2d_41 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_29 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_42 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_30 (MaxPooling2D)	(None, 44, 44, 64)	0
conv2d_43 (Conv2D)	(None, 42, 42, 128)	73,856
max_pooling2d_31 (MaxPooling2D)	(None, 21, 21, 128)	0
conv2d_44 (Conv2D)	(None, 19, 19, 256)	295,168
max_pooling2d_32 (MaxPooling2D)	(None, 10, 10, 256)	0
conv2d_45 (Conv2D)	(None, 8, 8, 512)	1,180,160
flatten_8 (Flatten)	(None, 32768)	0
dropout_8 (Dropout)	(None, 32768)	0
dense_8 (Dense)	(None, 1)	32,769

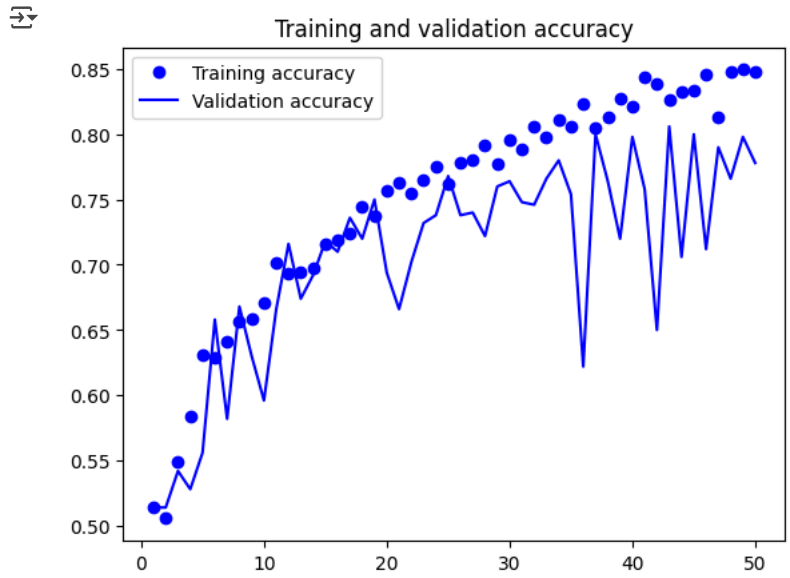
Total params: 1,601,345 (6.11 MB)
Trainable params: 1,601,345 (6.11 MB)

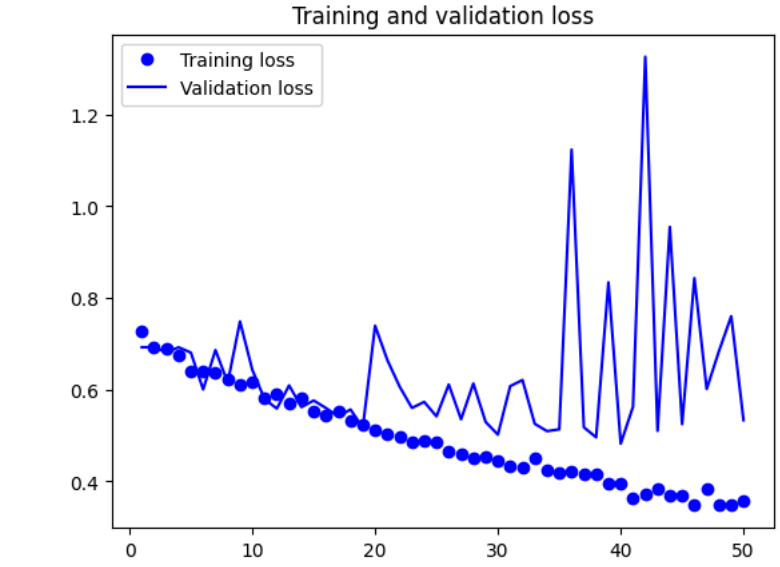
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model9 = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.5193 - loss: 0.7924 - val_accuracy: 0.5140 - val_loss: 0.6926
Epoch 2/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5037 - loss: 0.6935 - val_accuracy: 0.5140 - val_loss: 0.6928
Epoch 3/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.5211 - loss: 0.6911 - val_accuracy: 0.5420 - val_loss: 0.6818
Epoch 4/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5606 - loss: 0.6782 - val_accuracy: 0.5280 - val_loss: 0.6925
Epoch 5/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6150 - loss: 0.6540 - val_accuracy: 0.5560 - val_loss: 0.6809
Epoch 6/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6211 - loss: 0.6424 - val_accuracy: 0.6580 - val_loss: 0.6005
Epoch 7/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.6246 - loss: 0.6452 - val_accuracy: 0.5820 - val_loss: 0.6865
Epoch 8/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6471 - loss: 0.6287 - val_accuracy: 0.6680 - val_loss: 0.6155
Epoch 9/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6528 - loss: 0.6069 - val_accuracy: 0.6300 - val_loss: 0.7489
Epoch 10/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6647 - loss: 0.6540 - val_accuracy: 0.5960 - val_loss: 0.6436
Epoch 11/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6797 - loss: 0.5972 - val_accuracy: 0.6660 - val_loss: 0.5794
Epoch 12/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6937 - loss: 0.5943 - val_accuracy: 0.7160 - val_loss: 0.5589
Epoch 13/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6899 - loss: 0.5796 - val_accuracy: 0.6740 - val_loss: 0.6093
Epoch 14/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6980 - loss: 0.5810 - val_accuracy: 0.6920 - val_loss: 0.5620
Epoch 15/50
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7127 - loss: 0.5521 - val_accuracy: 0.7180 - val_loss: 0.5764
Epoch 16/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7274 - loss: 0.5264 - val_accuracy: 0.7100 - val_loss: 0.5600
Epoch 17/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7216 - loss: 0.5459 - val_accuracy: 0.7360 - val_loss: 0.5417
Epoch 18/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7452 - loss: 0.5262 - val_accuracy: 0.7200 - val_loss: 0.5568
Epoch 19/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7398 - loss: 0.5090 - val_accuracy: 0.7500 - val_loss: 0.5173
Epoch 20/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7574 - loss: 0.5131 - val_accuracy: 0.6940 - val_loss: 0.7395
Epoch 21/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7595 - loss: 0.5092 - val_accuracy: 0.6660 - val_loss: 0.6646
Epoch 22/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7683 - loss: 0.4876 - val_accuracy: 0.7020 - val_loss: 0.6067
Epoch 23/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7559 - loss: 0.4952 - val_accuracy: 0.7320 - val_loss: 0.5602
Epoch 24/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7771 - loss: 0.4907 - val_accuracy: 0.7380 - val_loss: 0.5734
Epoch 25/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7661 - loss: 0.4854 - val_accuracy: 0.7680 - val_loss: 0.5419
Epoch 26/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7840 - loss: 0.4596 - val_accuracy: 0.7380 - val_loss: 0.6113
Epoch 27/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7699 - loss: 0.4652 - val_accuracy: 0.7400 - val_loss: 0.5354
Epoch 28/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7914 - loss: 0.4568 - val_accuracy: 0.7220 - val_loss: 0.6135
Epoch 29/50
```

```
import matplotlib.pyplot as plt
accuracy = Model9.history["accuracy"]
val_accuracy = Model9.history["val_accuracy"]
loss = Model9.history["loss"]
val_loss = Model9.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()
```



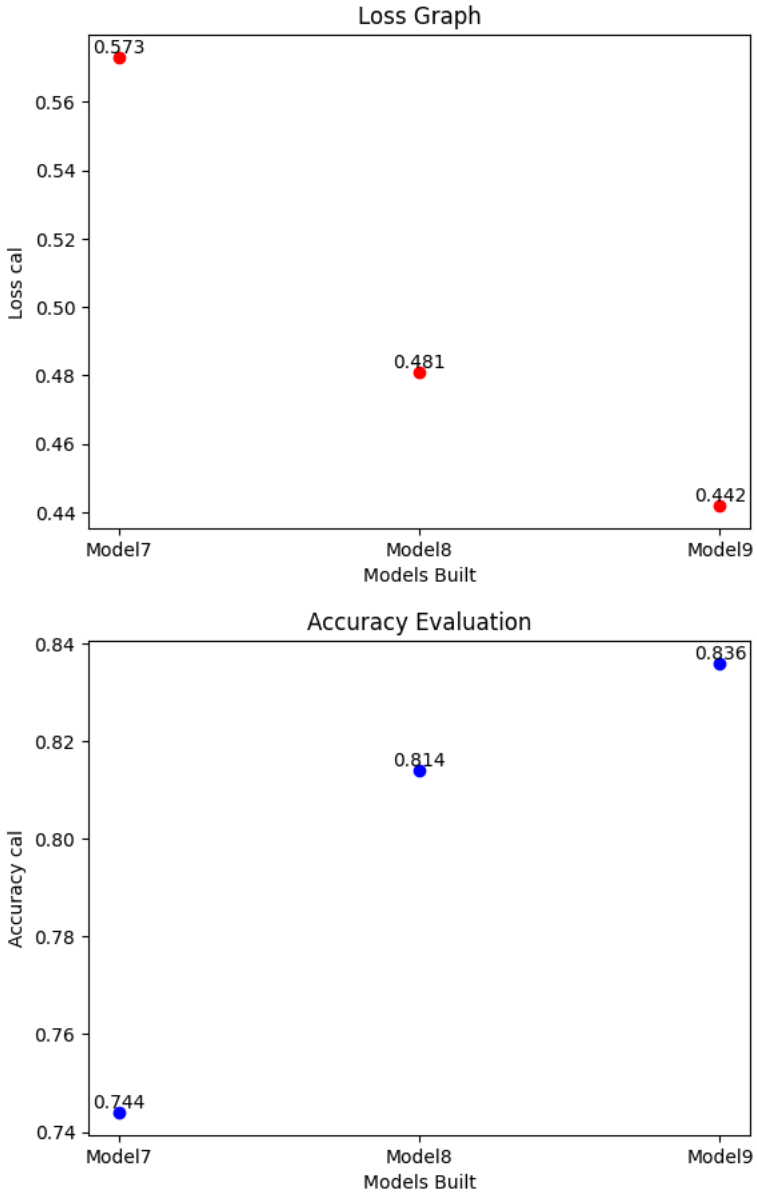



```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8561 - loss: 0.4528
Test accuracy: 0.836
Test loss: 0.442
```

```
Model7 = (0.573, 0.744)
Model8 = (0.481, 0.814)
Model9 = (0.442, 0.836)


Models= ("Model7","Model8","Model9")
Loss= (Model7[0], Model8[0],Model9[0])
Accuracy= (Model7[1], Model8[1],Model9[1])

plt.scatter(Models, Loss, color="red")
plt.title("Loss Graph")
plt.xlabel("Models Built")
plt.ylabel("Loss cal")

for (xi, yi) in zip(Models,Loss):
  plt.text(xi, yi, yi, va='bottom', ha='center')
plt.show()

plt.scatter(Models, Accuracy, color="blue")
plt.title("Accuracy Evaluation")
plt.xlabel("Models Built")
plt.ylabel("Accuracy cal")

for (xi, yi) in zip(Models,Accuracy):
```

```
  plt.text(xi, yi, yi, va='bottom', ha='center')
plt.show()
```



Building Model 10 with Train Size with 1900

```
import os
import shutil
from pathlib import Path

original_data_dir = pathlib.Path("drive/MyDrive/Colab Notebooks/cats_vs_dogs_small")
new_base_dir = Path("cats_vs_dogs_small/train4")

def make_subset(subset_name, start_index, end_index):
    for category in ("cats", "dogs"):
        src_dir = original_data_dir /subset_name/ category
        dst_dir = new_base_dir / subset_name / category

        print(f"Creating subset: {subset_name}, Category: {category}")
        print(f"Destination directory: {dst_dir}")

        os.makedirs(dst_dir, exist_ok=True)

        fnames = [f"{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            src_file = src_dir / fname
            dst_file = dst_dir / fname

            if src_file.exists():
                shutil.copyfile(src_file, dst_file)
            else:
                print(f"Warning: Source file {src_file} not found.")
```

```
make_subset("train", start_index=0, end_index=950)
make_subset("validation", start_index=1250, end_index=1500)
make_subset("test", start_index=1500, end_index=1750)
```

```
Creating subset: train, Category: cats
Destination directory: cats_vs_dogs_small/train4/train/cats
Creating subset: train, Category: dogs
Destination directory: cats_vs_dogs_small/train4/train/dogs
Creating subset: validation, Category: cats
Destination directory: cats_vs_dogs_small/train4/validation/cats
Creating subset: validation, Category: dogs
Destination directory: cats_vs_dogs_small/train4/validation/dogs
Creating subset: test, Category: cats
Destination directory: cats_vs_dogs_small/train4/test/cats
Creating subset: test, Category: dogs
Destination directory: cats_vs_dogs_small/train4/test/dogs
```

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train", #change to train1, ...for train5 for diferent models
    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 1900 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
```

```
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,)
```

```
data_augmentation3 = keras.Sequential(
  [
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.15),
    layers.RandomZoom(0.25)
  ]
)
```

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation3(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

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

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation3(inputs)
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=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=512, 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: "functional_34"

Layer (type)	Output Shape	Param #
input_layer_13 (InputLayer)	(None, 180, 180, 3)	0
rescaling_9 (Rescaling)	(None, 180, 180, 3)	0
conv2d_46 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_33 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_47 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_34 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_48 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_35 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_49 (Conv2D)	(None, 18, 18, 128)	147,584
max_pooling2d_36 (MaxPooling2D)	(None, 9, 9, 128)	0
conv2d_50 (Conv2D)	(None, 7, 7, 256)	295,168
max_pooling2d_37 (MaxPooling2D)	(None, 3, 3, 256)	0
conv2d_51 (Conv2D)	(None, 1, 1, 512)	1,180,160
flatten_9 (Flatten)	(None, 512)	0
dropout_9 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 1)	513

Total params: 1,716,673 (6.55 MB)

Configuring the model for training

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

Fitting the model using a Dataset

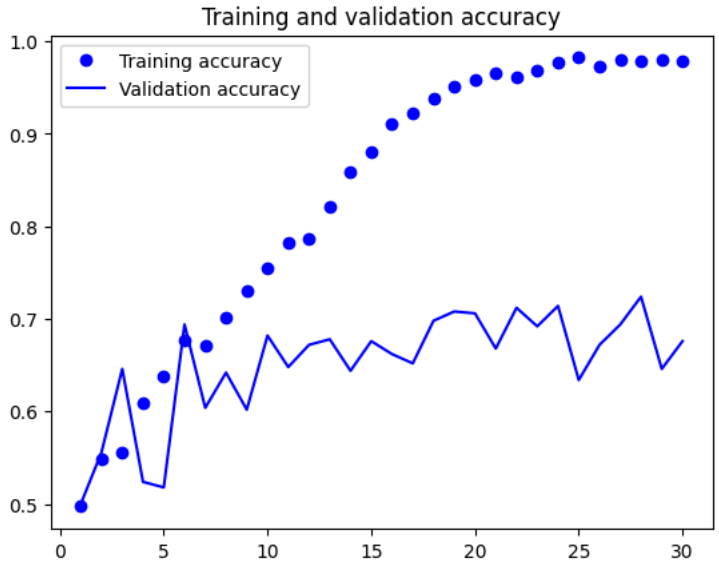
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
Model10 = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

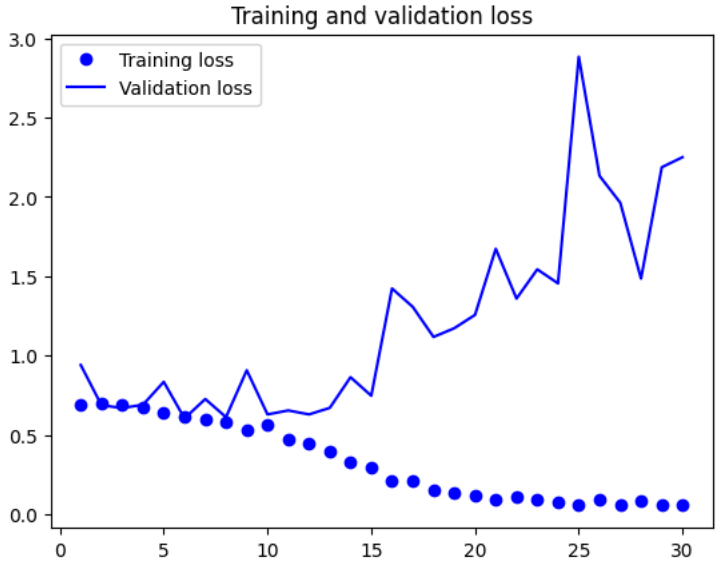
```
Epoch 1/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - accuracy: 0.4889 - loss: 0.6962 - val_accuracy: 0.5000 - val_loss: 0.9402
Epoch 2/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5347 - loss: 0.7074 - val_accuracy: 0.5560 - val_loss: 0.6862
Epoch 3/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5670 - loss: 0.6920 - val_accuracy: 0.6460 - val_loss: 0.6687
Epoch 4/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.6152 - loss: 0.6683 - val_accuracy: 0.5240 - val_loss: 0.6897
Epoch 5/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.6351 - loss: 0.6398 - val_accuracy: 0.5180 - val_loss: 0.8339
Epoch 6/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.6674 - loss: 0.6302 - val_accuracy: 0.6940 - val_loss: 0.6036
Epoch 7/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.6463 - loss: 0.6179 - val_accuracy: 0.6040 - val_loss: 0.7257
Epoch 8/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.6992 - loss: 0.5765 - val_accuracy: 0.6420 - val_loss: 0.6113
Epoch 9/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7329 - loss: 0.5214 - val_accuracy: 0.6020 - val_loss: 0.9068
Epoch 10/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7398 - loss: 0.6156 - val_accuracy: 0.6820 - val_loss: 0.6287
Epoch 11/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7880 - loss: 0.4724 - val_accuracy: 0.6480 - val_loss: 0.6535
Epoch 12/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7845 - loss: 0.4544 - val_accuracy: 0.6720 - val_loss: 0.6287
Epoch 13/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8246 - loss: 0.3893 - val_accuracy: 0.6780 - val_loss: 0.6697
Epoch 14/30
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8638 - loss: 0.3201 - val_accuracy: 0.6440 - val_loss: 0.8632
Epoch 15/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8829 - loss: 0.2880 - val_accuracy: 0.6760 - val_loss: 0.7465
Epoch 16/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8934 - loss: 0.2461 - val_accuracy: 0.6620 - val_loss: 1.4223
Epoch 17/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9120 - loss: 0.2420 - val_accuracy: 0.6520 - val_loss: 1.3067
Epoch 18/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9360 - loss: 0.1660 - val_accuracy: 0.6980 - val_loss: 1.1169
Epoch 19/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9607 - loss: 0.1075 - val_accuracy: 0.7080 - val_loss: 1.1720
Epoch 20/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9636 - loss: 0.1007 - val_accuracy: 0.7060 - val_loss: 1.2561
Epoch 21/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9633 - loss: 0.0917 - val_accuracy: 0.6680 - val_loss: 1.6725
Epoch 22/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9609 - loss: 0.1065 - val_accuracy: 0.7120 - val_loss: 1.3589
Epoch 23/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9655 - loss: 0.1060 - val_accuracy: 0.6920 - val_loss: 1.5431
Epoch 24/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9717 - loss: 0.0828 - val_accuracy: 0.7140 - val_loss: 1.4547
Epoch 25/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9882 - loss: 0.0433 - val_accuracy: 0.6340 - val_loss: 2.8835
Epoch 26/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9576 - loss: 0.1742 - val_accuracy: 0.6720 - val_loss: 2.1326
Epoch 27/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9685 - loss: 0.0786 - val_accuracy: 0.6940 - val_loss: 1.9632
Epoch 28/30
60/60 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9751 - loss: 0.0753 - val_accuracy: 0.7240 - val_loss: 1.4850
Epoch 29/30
```

Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
accuracy = Model10.history["accuracy"]
val_accuracy = Model10.history["val_accuracy"]
loss = Model10.history["loss"]
val_loss = Model10.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()
```





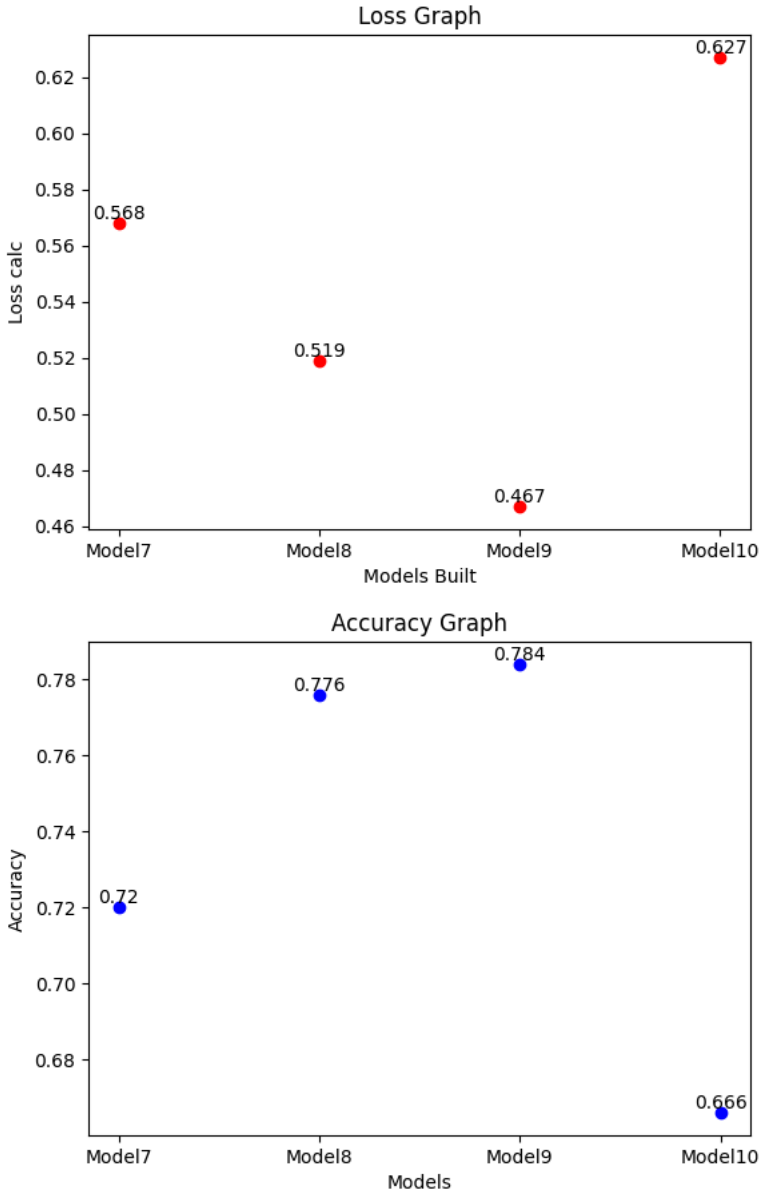
Evaluating the model on the test set

```
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}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.6739 - loss: 0.6099
Test accuracy: 0.666
Test loss: 0.627
```

```
Model7 = (0.568, 0.720)
Model8 = (0.519, 0.776)
Model9 = (0.467, 0.784)
Model10 = (0.627, 0.666)


Models= ("Model7","Model8","Model9","Model10")
Loss= (Model7[0], Model8[0],Model9[0],Model10[0])
Accuracy= (Model7[1], Model8[1],Model9[1],Model10[1])

plt.scatter(Models, Loss, color="red")
plt.title("Loss Graph")
plt.xlabel("Models Built")
plt.ylabel("Loss calc")

for (xi, yi) in zip(Models,Loss):
 plt.text(xi, yi, yi, va='bottom', ha='center')
plt.show()

plt.scatter(Models, Accuracy, color="blue")
plt.title("Accuracy Graph")
plt.xlabel("Models")
```

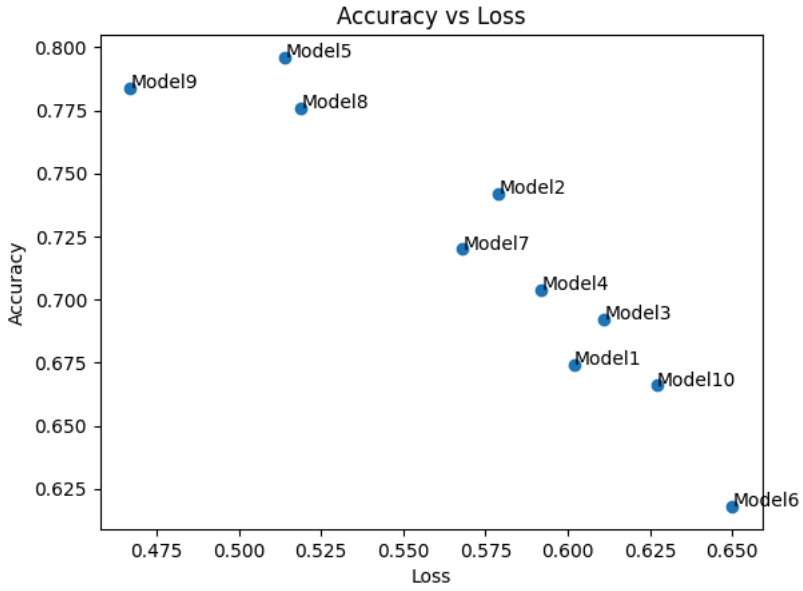
```
plt.ylabel("Accuracy")

for (xi, yi) in zip(Models,Accuracy):
 plt.text(xi, yi, yi, va='bottom', ha='center')
plt.show()
```



Comparing the model

```
import matplotlib.pyplot as plt

Model1 = (0.602, 0.674)
Model2 = (0.579, 0.742)
Model3 = (0.611, 0.692)
Model4 = (0.592, 0.704)
Model5 = (0.514, 0.796)
Model6 = (0.650, 0.618)
Model7 = (0.568, 0.720)
Model8 = (0.519, 0.776)
Model9 = (0.467, 0.784)
Model10 = (0.627, 0.666)

Models = ["Model1", "Model2", "Model3", "Model4", "Model5", "Model6", "Model7", "Model8", "Model9", "Model10"]
Loss_cal = [Model1[0], Model2[0], Model3[0], Model4[0], Model5[0], Model6[0], Model7[0], Model8[0], Model9[0], Model10[0]]
Accuracy_cal = [Model1[1], Model2[1], Model3[1], Model4[1], Model5[1], Model6[1], Model7[1], Model8[1], Model9[1], Model10[1]]

fig, ax = plt.subplots()
ax.scatter(Loss_cal, Accuracy_cal)

for i, txt in enumerate(Models):
    ax.annotate(txt, (Loss_cal[i], Accuracy_cal[i]))
```

```
plt.title("Accuracy vs Loss")
plt.ylabel("Accuracy")
plt.xlabel("Loss")


plt.show()
```



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

```
import os
import shutil
from pathlib import Path

#Building model with train size 1400 and same values of validation, test
original_data_dir = pathlib.Path("drive/MyDrive/Colab Notebooks/cats_vs_dogs_small")
new_base_dir = Path("cats_vs_dogs_small/train2")

def make_subset(subset_name, start_index, end_index):
    for category in ("cats", "dogs"):
        src_dir = original_data_dir /subset_name/ category
        dst_dir = new_base_dir / subset_name / category

        print(f"Creating subset: {subset_name}, Category: {category}")
        print(f"Destination directory: {dst_dir}")

        os.makedirs(dst_dir, exist_ok=True)

        fnames = [f"{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            src_file = src_dir / fname
            dst_file = dst_dir / fname

            # Copy only if the source file exists
            if src_file.exists():
                shutil.copyfile(src_file, dst_file)
            else:
                print(f"Warning: Source file {src_file} not found.")


make_subset("train", start_index=0, end_index=700)
make_subset("validation", start_index=1000, end_index=1250)
make_subset("test", start_index=1500, end_index=1750)
```

```
Creating subset: train, Category: cats
Destination directory: cats_vs_dogs_small/train2/train/cats
Creating subset: train, Category: dogs
Destination directory: cats_vs_dogs_small/train2/train/dogs
Creating subset: validation, Category: cats
Destination directory: cats_vs_dogs_small/train2/validation/cats
Creating subset: validation, Category: dogs
Destination directory: cats_vs_dogs_small/train2/validation/dogs
Creating subset: test, Category: cats
Destination directory: cats_vs_dogs_small/train2/test/cats
Creating subset: test, Category: dogs
```

```
Destination directory: cats_vs_dogs_small/train2/test/dogs
```

```
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 1400 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
```

```
from tensorflow import keras
from tensorflow.keras import layers, models

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

conv_base.trainable = False

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_nc
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 7s 91ms/step - accuracy: 0.9444 - loss: 0.2266 - val_accuracy: 0.7600 - val_loss: 1.3128
Epoch 2/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9928 - loss: 0.0362 - val_accuracy: 0.7140 - val_loss: 1.4544
Epoch 3/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9935 - loss: 0.0268 - val_accuracy: 0.7660 - val_loss: 1.5008
Epoch 4/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9926 - loss: 0.0415 - val_accuracy: 0.7220 - val_loss: 1.7632
Epoch 5/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9814 - loss: 0.0653 - val_accuracy: 0.7520 - val_loss: 1.4131
Epoch 6/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9906 - loss: 0.0357 - val_accuracy: 0.6920 - val_loss: 2.6058
Epoch 7/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9626 - loss: 0.1954 - val_accuracy: 0.7360 - val_loss: 2.1746
Epoch 8/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9783 - loss: 0.0807 - val_accuracy: 0.7260 - val_loss: 1.9941
Epoch 9/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9850 - loss: 0.0511 - val_accuracy: 0.7400 - val_loss: 2.0350
Epoch 10/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9897 - loss: 0.0402 - val_accuracy: 0.7240 - val_loss: 2.1393
Epoch 11/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9855 - loss: 0.0660 - val_accuracy: 0.7260 - val_loss: 2.1501
Epoch 12/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9813 - loss: 0.0600 - val_accuracy: 0.7460 - val_loss: 1.9721
Epoch 13/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9836 - loss: 0.0872 - val_accuracy: 0.7260 - val_loss: 1.7763
Epoch 14/50
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9891 - loss: 0.0589 - val_accuracy: 0.7400 - val_loss: 2.0187
Epoch 15/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9802 - loss: 0.0571 - val_accuracy: 0.7360 - val_loss: 1.8276
Epoch 16/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9898 - loss: 0.0277 - val_accuracy: 0.7360 - val_loss: 1.7471
Epoch 17/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9939 - loss: 0.0188 - val_accuracy: 0.7320 - val_loss: 1.9186
Epoch 18/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9942 - loss: 0.0326 - val_accuracy: 0.7120 - val_loss: 2.4470
Epoch 19/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9820 - loss: 0.0795 - val_accuracy: 0.7620 - val_loss: 2.2062
Epoch 20/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 9.2402e-04 - val_accuracy: 0.7540 - val_loss: 2.2379
Epoch 21/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 1.1570e-04 - val_accuracy: 0.7580 - val_loss: 2.5787
Epoch 22/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9987 - loss: 0.0045 - val_accuracy: 0.7020 - val_loss: 2.8691
Epoch 23/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9753 - loss: 0.0877 - val_accuracy: 0.7180 - val_loss: 2.5999
Epoch 24/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9954 - loss: 0.0235 - val_accuracy: 0.7460 - val_loss: 2.2616
Epoch 25/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9902 - loss: 0.0729 - val_accuracy: 0.7300 - val_loss: 2.2472
Epoch 26/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9989 - loss: 0.0038 - val_accuracy: 0.7340 - val_loss: 2.5100
Epoch 27/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9973 - loss: 0.0148 - val_accuracy: 0.7640 - val_loss: 2.4312
Epoch 28/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9913 - loss: 0.0591 - val_accuracy: 0.7540 - val_loss: 2.5351
Epoch 29/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9796 - loss: 0.0959 - val_accuracy: 0.7220 - val_loss: 2.3668
```

```
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}")
print(f"Test loss: {test_loss:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.6654 - loss: 0.6151
Test accuracy: 0.666
Test loss: 0.627
```

```
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```