Assignment-2

*Convolution *

Downloading the data

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

Copying images to training, validation, and test directories

```
import os
import shutil
import pathlib

# Define original folders separately
original_train_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/train")
original_val_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/validation")
original_test_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/test")

# Define destination base directory
new_base_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small 2/small_dataset_balanced")

def make_subset_from_folder(subset_name, original_folder, start_index, end_index):

    print(f"\nCreating '{subset_name}' from {original_folder} indices {start_index} to {end_index}")

    for category in ["cats", "dogs"]:
        src_dir = original_folder / category
        dest_dir = new_base_dir / subset_name / category

        os.makedirs(dest_dir, exist_ok=True)

        files = sorted(os.listdir(src_dir))

        # Adjust end_index if it's too big
        if end_index > len(files):
            print(f" WARNING: end_index {end_index} exceeds available files ({len(files)}). Adjusting.")
            end_index = len(files)

        subset_files = files[start_index:end_index]
        print(f"Copying {len(subset_files)} '{category}' images to '{dest_dir}'...")

        for fname in subset_files:
            src_file = src_dir / fname
            dst_file = dest_dir / fname
            shutil.copyfile(src_file, dst_file)

    print(f" {subset_name} created from {original_folder}.")

def main():


   # 1000 images from original train folder for training
    make_subset_from_folder("train", original_train_dir, 0, 500)

    # 250 images from original validation folder for validation
    make_subset_from_folder("validation", original_val_dir, 0, 250)

    # 250 images from original test folder for testing
    make_subset_from_folder("test", original_test_dir, 0, 250)

if __name__ == "__main__":
    main()
```

```
Creating 'train' from /content/drive/MyDrive/AML/cats_vs_dogs_small/train indices 0 to 500
Copying 500 'cats' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small 2/small_dataset_balanced/train/cats'...
Copying 500 'dogs' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small 2/small_dataset_balanced/train/dogs'...
✅ train created from /content/drive/MyDrive/AML/cats_vs_dogs_small/train.

Creating 'validation' from /content/drive/MyDrive/AML/cats_vs_dogs_small/validation indices 0 to 250
Copying 250 'cats' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small 2/small_dataset_balanced/validation/cats'...
Copying 250 'dogs' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small 2/small_dataset_balanced/validation/dogs'...
```

```
✅ validation created from /content/drive/MyDrive/AML/cats_vs_dogs_small/validation.

Creating 'test' from /content/drive/MyDrive/AML/cats_vs_dogs_small/test indices 0 to 250
Copying 250 'cats' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small 2/small_dataset_balanced/test/cats'...
Copying 250 'dogs' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small 2/small_dataset_balanced/test/dogs'...
✅ test created from /content/drive/MyDrive/AML/cats_vs_dogs_small/test.
```

```
make_subset_from_folder("train", original_train_dir, 0, 500)
make_subset_from_folder("validation", original_val_dir, 0, 250)
make_subset_from_folder("test", original_test_dir, 0, 250)
```

```
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
)
```

Training the convent network from scratch

Model 1:Training sample of 1000, a validation sample of 500, and a test sample of 500

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

Instantiating a small convnet for dogs vs. cats classification

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

```
model.summary()
```

Configuring the model for training

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

Data preprocessing

```
from tensorflow.keras.utils import image_dataset_from_directory

# Base directory for the subsets
```

```
subset_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small 2/small_dataset_balanced")

# Load datasets
train_dataset = image_dataset_from_directory(
    subset_dir / "train",
    image_size=(180, 180),
    batch_size=32
)

validation_dataset = image_dataset_from_directory(
    subset_dir / "validation",
    image_size=(180, 180),
    batch_size=32
)

test_dataset = image_dataset_from_directory(
    subset_dir / "test",
    image_size=(180, 180),
    batch_size=32
)


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


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


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


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

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
```

Fitting the model using a Dataset

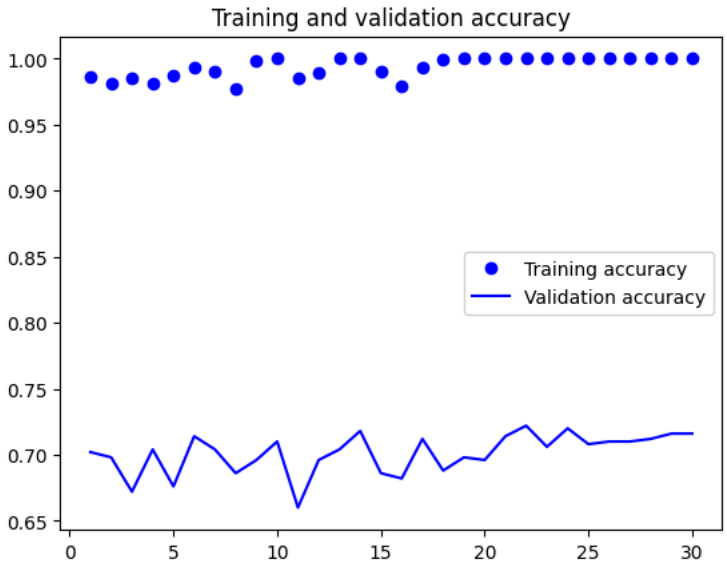
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

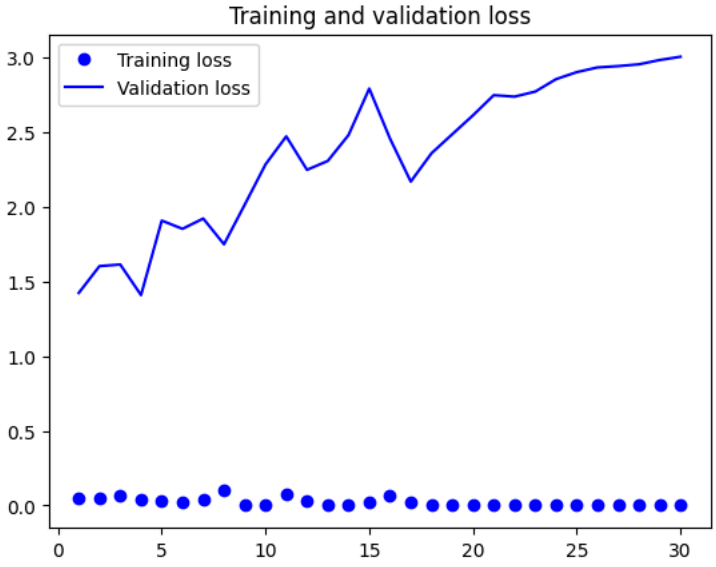
```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 181ms/step - accuracy: 0.9784 - loss: 0.0562 - val_accuracy: 0.6980 - val_loss: 1.6040
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 161ms/step - accuracy: 0.9907 - loss: 0.0445 - val_accuracy: 0.6720 - val_loss: 1.6145
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 12s 224ms/step - accuracy: 0.9748 - loss: 0.0598 - val_accuracy: 0.7040 - val_loss: 1.4100
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 183ms/step - accuracy: 0.9899 - loss: 0.0313 - val_accuracy: 0.6760 - val_loss: 1.9088
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 153ms/step - accuracy: 0.9866 - loss: 0.0358 - val_accuracy: 0.7140 - val_loss: 1.8536
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 190ms/step - accuracy: 0.9952 - loss: 0.0194 - val_accuracy: 0.7040 - val_loss: 1.9225
```

```
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 185ms/step - accuracy: 0.9979 - loss: 0.0115 - val_accuracy: 0.6960 - val_loss: 2.0160
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 221ms/step - accuracy: 1.0000 - loss: 0.0010 - val_accuracy: 0.7100 - val_loss: 2.2852
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 150ms/step - accuracy: 0.9975 - loss: 0.0142 - val_accuracy: 0.6600 - val_loss: 2.4729
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 188ms/step - accuracy: 0.9816 - loss: 0.0593 - val_accuracy: 0.6960 - val_loss: 2.2489
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 153ms/step - accuracy: 1.0000 - loss: 7.7117e-04 - val_accuracy: 0.7040 - val_loss: 2.3084
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 193ms/step - accuracy: 1.0000 - loss: 1.6007e-04 - val_accuracy: 0.7180 - val_loss: 2.4816
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 156ms/step - accuracy: 0.9983 - loss: 0.0046 - val_accuracy: 0.6860 - val_loss: 2.7933
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 192ms/step - accuracy: 0.9890 - loss: 0.0421 - val_accuracy: 0.6820 - val_loss: 2.4579
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 158ms/step - accuracy: 0.9849 - loss: 0.0349 - val_accuracy: 0.7120 - val_loss: 2.1702
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 191ms/step - accuracy: 0.9987 - loss: 0.0037 - val_accuracy: 0.6880 - val_loss: 2.3599
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 187ms/step - accuracy: 1.0000 - loss: 1.8574e-04 - val_accuracy: 0.6980 - val_loss: 2.4874
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 1.0000 - loss: 5.3500e-05 - val_accuracy: 0.6960 - val_loss: 2.6143
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 156ms/step - accuracy: 1.0000 - loss: 3.3315e-05 - val_accuracy: 0.7140 - val_loss: 2.7491
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 192ms/step - accuracy: 1.0000 - loss: 2.2664e-05 - val_accuracy: 0.7220 - val_loss: 2.7391
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 163ms/step - accuracy: 1.0000 - loss: 9.9674e-06 - val_accuracy: 0.7060 - val_loss: 2.7730
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - accuracy: 1.0000 - loss: 7.1740e-06 - val_accuracy: 0.7200 - val_loss: 2.856
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 194ms/step - accuracy: 1.0000 - loss: 6.3969e-06 - val_accuracy: 0.7080 - val_loss: 2.903
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 190ms/step - accuracy: 1.0000 - loss: 4.0050e-06 - val_accuracy: 0.7100 - val_loss: 2.9350
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 169ms/step - accuracy: 1.0000 - loss: 4.5808e-06 - val_accuracy: 0.7100 - val_loss: 2.9438
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 159ms/step - accuracy: 1.0000 - loss: 3.3349e-06 - val_accuracy: 0.7120 - val_loss: 2.9558
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 189ms/step - accuracy: 1.0000 - loss: 3.5103e-06 - val_accuracy: 0.7160 - val_loss: 2.9847
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 186ms/step - accuracy: 1.0000 - loss: 3.1251e-06 - val_accuracy: 0.7160 - val_loss: 3.006
```

Displaying curves of loss and accuracy during training

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





Evaluating the model on the test set

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

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 98ms/step - accuracy: 0.6954 - loss: 1.3531
Test accuracy: 0.680
```

Since the validation and the tes accuracy of the model is very low that is 68%

To improve performance in developing a network that we trained from scratch, we will train our model on following techniques.

Model 1a: Using Data Augmentation

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

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = 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)
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")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
     callbacks=callbacks)
```

```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 161ms/step - accuracy: 0.5202 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.8568
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 183ms/step - accuracy: 0.4901 - loss: 0.7313 - val_accuracy: 0.5000 - val_loss: 0.6929
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 169ms/step - accuracy: 0.4770 - loss: 0.6940 - val_accuracy: 0.5000 - val_loss: 0.7123
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 153ms/step - accuracy: 0.5161 - loss: 0.6943 - val_accuracy: 0.5340 - val_loss: 0.6901
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 227ms/step - accuracy: 0.5673 - loss: 0.6866 - val_accuracy: 0.5560 - val_loss: 0.6743
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 183ms/step - accuracy: 0.6490 - loss: 0.6765 - val_accuracy: 0.5560 - val_loss: 0.7657
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 154ms/step - accuracy: 0.5770 - loss: 0.6862 - val_accuracy: 0.6180 - val_loss: 0.6675
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 209ms/step - accuracy: 0.6222 - loss: 0.6679 - val_accuracy: 0.6540 - val_loss: 0.6293
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 185ms/step - accuracy: 0.6550 - loss: 0.6397 - val_accuracy: 0.6440 - val_loss: 0.6309
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 162ms/step - accuracy: 0.6436 - loss: 0.6462 - val_accuracy: 0.5980 - val_loss: 0.6575
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 208ms/step - accuracy: 0.6542 - loss: 0.6226 - val_accuracy: 0.6820 - val_loss: 0.5952
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.6781 - loss: 0.5927 - val_accuracy: 0.5760 - val_loss: 0.9135
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 150ms/step - accuracy: 0.6993 - loss: 0.5801 - val_accuracy: 0.6520 - val_loss: 0.6098
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 198ms/step - accuracy: 0.7040 - loss: 0.5573 - val_accuracy: 0.7000 - val_loss: 0.5678
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 151ms/step - accuracy: 0.7013 - loss: 0.5459 - val_accuracy: 0.5780 - val_loss: 0.7547
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 217ms/step - accuracy: 0.7116 - loss: 0.5781 - val_accuracy: 0.6120 - val_loss: 0.6559
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 152ms/step - accuracy: 0.7029 - loss: 0.5773 - val_accuracy: 0.6960 - val_loss: 0.5807
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 190ms/step - accuracy: 0.7262 - loss: 0.5331 - val_accuracy: 0.7100 - val_loss: 0.5447
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 148ms/step - accuracy: 0.7343 - loss: 0.5324 - val_accuracy: 0.6080 - val_loss: 0.6452
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 221ms/step - accuracy: 0.7282 - loss: 0.5447 - val_accuracy: 0.6540 - val_loss: 0.6239
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 164ms/step - accuracy: 0.7288 - loss: 0.5270 - val_accuracy: 0.6760 - val_loss: 0.6904
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 144ms/step - accuracy: 0.7667 - loss: 0.5307 - val_accuracy: 0.6960 - val_loss: 0.6706
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 220ms/step - accuracy: 0.7295 - loss: 0.5244 - val_accuracy: 0.7500 - val_loss: 0.5293
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 155ms/step - accuracy: 0.7840 - loss: 0.4791 - val_accuracy: 0.6800 - val_loss: 0.6528
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 174ms/step - accuracy: 0.7563 - loss: 0.5179 - val_accuracy: 0.7140 - val_loss: 0.5965
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 146ms/step - accuracy: 0.7792 - loss: 0.4753 - val_accuracy: 0.7460 - val_loss: 0.5566
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 195ms/step - accuracy: 0.7752 - loss: 0.4720 - val_accuracy: 0.7320 - val_loss: 0.5612
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 145ms/step - accuracy: 0.7693 - loss: 0.4993 - val_accuracy: 0.7160 - val_loss: 0.5738
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 156ms/step - accuracy: 0.7976 - loss: 0.4526 - val_accuracy: 0.7160 - val_loss: 0.5901
```

```
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}")
```

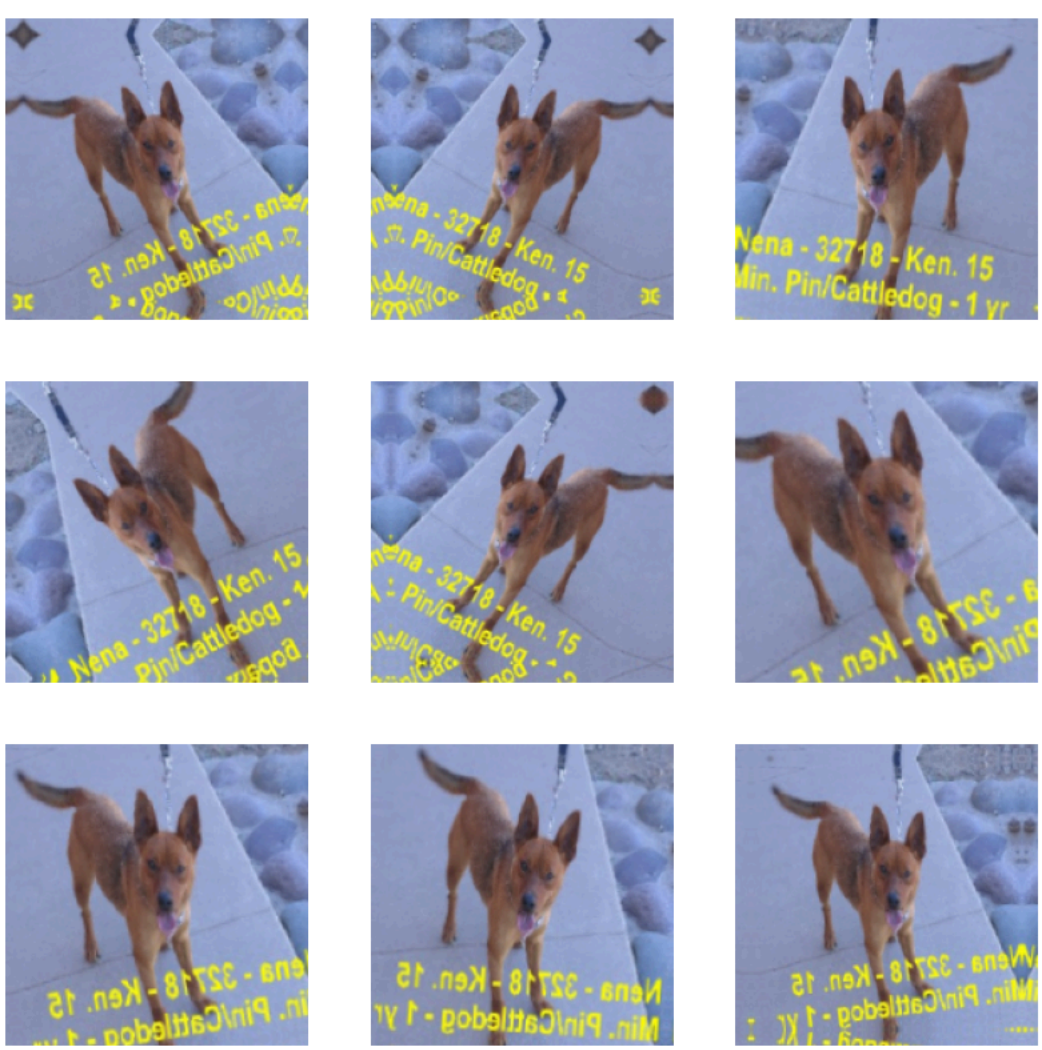
```
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 92ms/step - accuracy: 0.7073 - loss: 0.5761
Test accuracy: 0.698
```

Defining 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 1b: Using Dropout Method

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
```

```
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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


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

```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 16s 179ms/step - accuracy: 0.5061 - loss: 0.6937 - val_accuracy: 0.5000 - val_loss: 0.6934
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 179ms/step - accuracy: 0.5237 - loss: 0.6947 - val_accuracy: 0.5000 - val_loss: 0.6923
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 186ms/step - accuracy: 0.5815 - loss: 0.7001 - val_accuracy: 0.5000 - val_loss: 0.6910
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 177ms/step - accuracy: 0.5315 - loss: 0.6916 - val_accuracy: 0.5080 - val_loss: 0.6986
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 180ms/step - accuracy: 0.5505 - loss: 0.6859 - val_accuracy: 0.6500 - val_loss: 0.6789
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 142ms/step - accuracy: 0.6091 - loss: 0.6677 - val_accuracy: 0.5320 - val_loss: 0.9232
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 180ms/step - accuracy: 0.5876 - loss: 0.6922 - val_accuracy: 0.6400 - val_loss: 0.6296
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 141ms/step - accuracy: 0.6460 - loss: 0.6246 - val_accuracy: 0.5720 - val_loss: 0.7152
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 155ms/step - accuracy: 0.6590 - loss: 0.6233 - val_accuracy: 0.5500 - val_loss: 0.7159
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 208ms/step - accuracy: 0.6745 - loss: 0.6024 - val_accuracy: 0.6020 - val_loss: 0.7066
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 152ms/step - accuracy: 0.6813 - loss: 0.5918 - val_accuracy: 0.6560 - val_loss: 0.6261
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 218ms/step - accuracy: 0.7195 - loss: 0.5382 - val_accuracy: 0.6220 - val_loss: 0.8872
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 147ms/step - accuracy: 0.7211 - loss: 0.5528 - val_accuracy: 0.6480 - val_loss: 0.6442
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 191ms/step - accuracy: 0.7541 - loss: 0.5042 - val_accuracy: 0.6680 - val_loss: 0.6225
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.7734 - loss: 0.4610 - val_accuracy: 0.6860 - val_loss: 0.6184
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 172ms/step - accuracy: 0.7998 - loss: 0.4282 - val_accuracy: 0.6540 - val_loss: 0.6433
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 144ms/step - accuracy: 0.8029 - loss: 0.4348 - val_accuracy: 0.5560 - val_loss: 2.2641
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 195ms/step - accuracy: 0.7926 - loss: 0.5551 - val_accuracy: 0.6940 - val_loss: 0.6944
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 148ms/step - accuracy: 0.8343 - loss: 0.3702 - val_accuracy: 0.6320 - val_loss: 0.7278
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 181ms/step - accuracy: 0.8573 - loss: 0.3300 - val_accuracy: 0.6800 - val_loss: 0.6121
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 152ms/step - accuracy: 0.9148 - loss: 0.2353 - val_accuracy: 0.7100 - val_loss: 0.7081
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 143ms/step - accuracy: 0.8963 - loss: 0.2329 - val_accuracy: 0.7100 - val_loss: 0.6670
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 205ms/step - accuracy: 0.9212 - loss: 0.2164 - val_accuracy: 0.7080 - val_loss: 0.9501
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 141ms/step - accuracy: 0.9127 - loss: 0.2064 - val_accuracy: 0.7180 - val_loss: 0.8849
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 171ms/step - accuracy: 0.9381 - loss: 0.1694 - val_accuracy: 0.6680 - val_loss: 0.9731
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 190ms/step - accuracy: 0.9624 - loss: 0.1118 - val_accuracy: 0.7040 - val_loss: 0.9450
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 190ms/step - accuracy: 0.9716 - loss: 0.0864 - val_accuracy: 0.7300 - val_loss: 1.0035
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 179ms/step - accuracy: 0.9721 - loss: 0.0809 - val_accuracy: 0.6900 - val_loss: 1.7237
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 143ms/step - accuracy: 0.9353 - loss: 0.1628 - val_accuracy: 0.7360 - val_loss: 1.2185
```

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

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 92ms/step - accuracy: 0.6615 - loss: 0.6635
Test accuracy: 0.664
```

Model 1c: Using both Image Augmentation and Dropout mthod

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


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

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


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

```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 174ms/step - accuracy: 0.4889 - loss: 0.6943 - val_accuracy: 0.5000 - val_loss: 0.6927
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 160ms/step - accuracy: 0.4861 - loss: 0.6940 - val_accuracy: 0.5060 - val_loss: 0.6919
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 190ms/step - accuracy: 0.5275 - loss: 0.6920 - val_accuracy: 0.5600 - val_loss: 0.6873
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 186ms/step - accuracy: 0.5442 - loss: 0.6879 - val_accuracy: 0.5000 - val_loss: 0.7319
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 187ms/step - accuracy: 0.5297 - loss: 0.6964 - val_accuracy: 0.5860 - val_loss: 0.6790
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 212ms/step - accuracy: 0.6028 - loss: 0.6752 - val_accuracy: 0.6240 - val_loss: 0.6571
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 193ms/step - accuracy: 0.6163 - loss: 0.6623 - val_accuracy: 0.6180 - val_loss: 0.6577
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 184ms/step - accuracy: 0.6326 - loss: 0.6435 - val_accuracy: 0.6280 - val_loss: 0.6473
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 208ms/step - accuracy: 0.6724 - loss: 0.6371 - val_accuracy: 0.5460 - val_loss: 0.7497
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 190ms/step - accuracy: 0.6485 - loss: 0.6304 - val_accuracy: 0.6320 - val_loss: 0.6823
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 197ms/step - accuracy: 0.6483 - loss: 0.6449 - val_accuracy: 0.6000 - val_loss: 0.6348
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 155ms/step - accuracy: 0.6445 - loss: 0.6123 - val_accuracy: 0.6380 - val_loss: 0.6374
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 164ms/step - accuracy: 0.6701 - loss: 0.6309 - val_accuracy: 0.6640 - val_loss: 0.6255
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 152ms/step - accuracy: 0.6727 - loss: 0.5970 - val_accuracy: 0.6440 - val_loss: 0.6654
Epoch 16/30
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 198ms/step - accuracy: 0.6904 - loss: 0.5888 - val_accuracy: 0.6720 - val_loss: 0.6088
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 179ms/step - accuracy: 0.7405 - loss: 0.5391 - val_accuracy: 0.6680 - val_loss: 0.5902
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 12s 223ms/step - accuracy: 0.7441 - loss: 0.5356 - val_accuracy: 0.6980 - val_loss: 0.5843
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 181ms/step - accuracy: 0.7278 - loss: 0.5320 - val_accuracy: 0.6360 - val_loss: 0.6233
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 181ms/step - accuracy: 0.6804 - loss: 0.5742 - val_accuracy: 0.6980 - val_loss: 0.5615
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 194ms/step - accuracy: 0.7343 - loss: 0.5494 - val_accuracy: 0.6820 - val_loss: 0.5874
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 165ms/step - accuracy: 0.7396 - loss: 0.5173 - val_accuracy: 0.6680 - val_loss: 0.6786
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 201ms/step - accuracy: 0.7104 - loss: 0.5581 - val_accuracy: 0.7080 - val_loss: 0.5655
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 170ms/step - accuracy: 0.7514 - loss: 0.5413 - val_accuracy: 0.7320 - val_loss: 0.5456
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 178ms/step - accuracy: 0.7310 - loss: 0.5131 - val_accuracy: 0.7260 - val_loss: 0.5824
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 195ms/step - accuracy: 0.7075 - loss: 0.5627 - val_accuracy: 0.6300 - val_loss: 0.7055
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 222ms/step - accuracy: 0.7313 - loss: 0.5493 - val_accuracy: 0.7280 - val_loss: 0.5427
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 196ms/step - accuracy: 0.7176 - loss: 0.5136 - val_accuracy: 0.7340 - val_loss: 0.5235
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 162ms/step - accuracy: 0.7610 - loss: 0.4754 - val_accuracy: 0.7360 - val_loss: 0.5352
```

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

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 102ms/step - accuracy: 0.7453 - loss: 0.5599
Test accuracy: 0.696
```

Model 2) Increasing the Training sample size to 1500.Including Maxpooling,Data Augmentation and Dropout technique with dropout rate of 0.5

```
import os
import shutil
import pathlib

# Define original folders (your dataset path)
original_train_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/train")
original_val_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/validation")
original_test_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/test")

# Destination for balanced dataset subsets
new_base_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small_subset")

def make_subset_from_folder(subset_name, original_folder, start_index, end_index):
    print(f"\nCreating '{subset_name}' subset from {original_folder} [{start_index}:{end_index}]")

    for category in ["cats", "dogs"]:
        src_dir = original_folder / category
        dest_dir = new_base_dir / subset_name / category

        # Clean old files if they exist
        if dest_dir.exists():
            shutil.rmtree(dest_dir)

        os.makedirs(dest_dir, exist_ok=True)

        files = sorted(os.listdir(src_dir))

        # Validate end_index
        if end_index > len(files):
            print(f" WARNING: end_index {end_index} exceeds available files ({len(files)}). Adjusting.")
            end_index = len(files)

        subset_files = files[start_index:end_index]
        print(f"Copying {len(subset_files)} '{category}' images to '{dest_dir}'...")

        for fname in subset_files:
            src_file = src_dir / fname
            dst_file = dest_dir / fname
            shutil.copyfile(src_file, dst_file)

    print(f" Subset '{subset_name}' created.")

def main():
    # 1500 train: 750 cats + 750 dogs
```

```
    make_subset_from_folder("train", original_train_dir, 0, 750)

    # 600 validation: 300 cats + 300 dogs
    make_subset_from_folder("validation", original_val_dir, 0, 300)

    # 600 test: 300 cats + 300 dogs
    make_subset_from_folder("test", original_test_dir, 0, 300)

if __name__ == "__main__":
    main()
```

```
Creating 'train' subset from /content/drive/MyDrive/AML/cats_vs_dogs_small/train [0:750]
Copying 750 'cats' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small_subset/train/cats'...
Copying 750 'dogs' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small_subset/train/dogs'...
✅ Subset 'train' created.

Creating 'validation' subset from /content/drive/MyDrive/AML/cats_vs_dogs_small/validation [0:300]
Copying 300 'cats' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small_subset/validation/cats'...
Copying 300 'dogs' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small_subset/validation/dogs'...
✅ Subset 'validation' created.

Creating 'test' subset from /content/drive/MyDrive/AML/cats_vs_dogs_small/test [0:300]
Copying 300 'cats' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small_subset/test/cats'...
Copying 300 'dogs' images to '/content/drive/MyDrive/AML/cats_vs_dogs_small_subset/test/dogs'...
✅ Subset 'test' created.
```

```
from tensorflow.keras.utils import image_dataset_from_directory
from pathlib import Path

new_base_dir = Path("/content/drive/MyDrive/AML/cats_vs_dogs_small_subset")

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
)

# Prefetch for speed
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
Found 1500 files belonging to 2 classes.
Found 600 files belonging to 2 classes.
Found 600 files belonging to 2 classes.
```

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

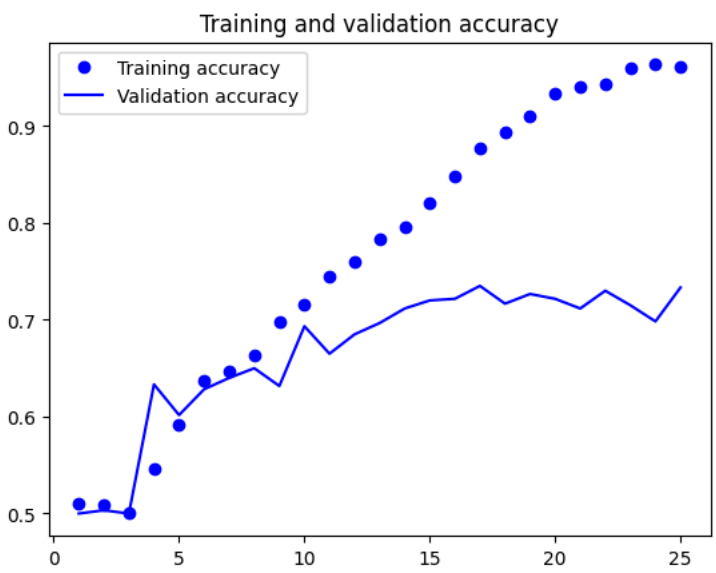
```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from keras import regularizers

# used early stopping to stop optimization when it isn't helping any more.
early_stopping_monitor = EarlyStopping(patience=10)


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



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")
```

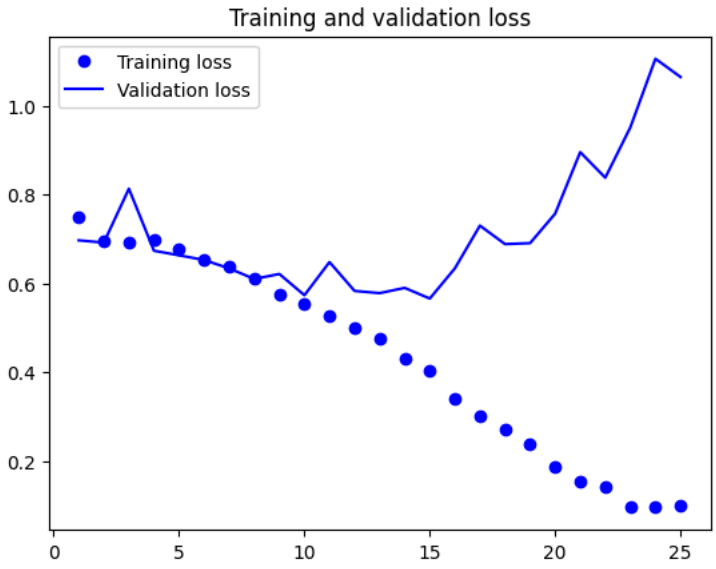
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss"), early_stopping_monitor
]
history = model.fit(
```

```
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 19s 297ms/step - accuracy: 0.5040 - loss: 0.8771 - val_accuracy: 0.5000 - val_loss: 0.6973
Epoch 2/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 171ms/step - accuracy: 0.4966 - loss: 0.6954 - val_accuracy: 0.5033 - val_loss: 0.6921
Epoch 3/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 160ms/step - accuracy: 0.5039 - loss: 0.6931 - val_accuracy: 0.5000 - val_loss: 0.8135
Epoch 4/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - accuracy: 0.5357 - loss: 0.7113 - val_accuracy: 0.6333 - val_loss: 0.6739
Epoch 5/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 173ms/step - accuracy: 0.5707 - loss: 0.6809 - val_accuracy: 0.6017 - val_loss: 0.6637
Epoch 6/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 12s 216ms/step - accuracy: 0.6440 - loss: 0.6583 - val_accuracy: 0.6283 - val_loss: 0.6530
Epoch 7/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 151ms/step - accuracy: 0.6293 - loss: 0.6469 - val_accuracy: 0.6400 - val_loss: 0.6341
Epoch 8/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 152ms/step - accuracy: 0.6791 - loss: 0.5949 - val_accuracy: 0.6500 - val_loss: 0.6103
Epoch 9/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 193ms/step - accuracy: 0.6958 - loss: 0.5725 - val_accuracy: 0.6317 - val_loss: 0.6216
Epoch 10/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 155ms/step - accuracy: 0.7140 - loss: 0.5517 - val_accuracy: 0.6933 - val_loss: 0.5737
Epoch 11/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 161ms/step - accuracy: 0.7427 - loss: 0.5354 - val_accuracy: 0.6650 - val_loss: 0.6483
Epoch 12/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 184ms/step - accuracy: 0.7659 - loss: 0.4897 - val_accuracy: 0.6850 - val_loss: 0.5834
Epoch 13/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 180ms/step - accuracy: 0.7704 - loss: 0.5031 - val_accuracy: 0.6967 - val_loss: 0.5785
Epoch 14/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 162ms/step - accuracy: 0.8111 - loss: 0.4059 - val_accuracy: 0.7117 - val_loss: 0.5903
Epoch 15/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 177ms/step - accuracy: 0.8238 - loss: 0.3858 - val_accuracy: 0.7200 - val_loss: 0.5662
Epoch 16/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 182ms/step - accuracy: 0.8642 - loss: 0.3256 - val_accuracy: 0.7217 - val_loss: 0.6342
Epoch 17/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 170ms/step - accuracy: 0.8832 - loss: 0.2952 - val_accuracy: 0.7350 - val_loss: 0.7304
Epoch 18/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 182ms/step - accuracy: 0.8984 - loss: 0.2597 - val_accuracy: 0.7167 - val_loss: 0.6889
Epoch 19/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 178ms/step - accuracy: 0.9106 - loss: 0.2286 - val_accuracy: 0.7267 - val_loss: 0.6909
Epoch 20/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 168ms/step - accuracy: 0.9509 - loss: 0.1593 - val_accuracy: 0.7217 - val_loss: 0.7573
Epoch 21/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 168ms/step - accuracy: 0.9424 - loss: 0.1561 - val_accuracy: 0.7117 - val_loss: 0.8958
Epoch 22/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 170ms/step - accuracy: 0.9529 - loss: 0.1217 - val_accuracy: 0.7300 - val_loss: 0.8383
Epoch 23/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 182ms/step - accuracy: 0.9668 - loss: 0.0847 - val_accuracy: 0.7150 - val_loss: 0.9517
Epoch 24/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 170ms/step - accuracy: 0.9704 - loss: 0.0901 - val_accuracy: 0.6983 - val_loss: 1.1060
Epoch 25/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 184ms/step - accuracy: 0.9645 - loss: 0.0917 - val_accuracy: 0.7333 - val_loss: 1.0650
```

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






```
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}")
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 96ms/step - accuracy: 0.7006 - loss: 0.6534
Test accuracy: 0.705
```

Model 3: Increasing the Training sample size to 1700

```
import os
import shutil
import pathlib
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.utils import image_dataset_from_directory

# 1. Function to Create Subsets
def make_subset(subset_name, original_dir, new_base_dir, start_index, end_index):
    print(f"\nCreating subset: {subset_name} from {original_dir} [{start_index}:{end_index}]")

    for category in ["cats", "dogs"]:
        src_dir = original_dir / category
        dest_dir = new_base_dir / subset_name / category

        if dest_dir.exists():
            shutil.rmtree(dest_dir)

        os.makedirs(dest_dir, exist_ok=True)

        files = sorted(os.listdir(src_dir))

        if end_index > len(files):
            print(f"⚠️ end_index {end_index} exceeds available files ({len(files)}). Adjusting...")
            end_index = len(files)
```

```
            end_index = len(files)

            subset_files = files[start_index:end_index]

            print(f"Copying {len(subset_files)} files from {src_dir} to {dest_dir}...")

            for fname in subset_files:
                src_file = src_dir / fname
                dst_file = dest_dir / fname

                shutil.copyfile(src_file, dst_file)

        print(f"Subset '{subset_name}' created successfully!")

    # 2. Paths Setup
    original_train_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/train")
    original_val_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/validation")
    original_test_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small/test")

    new_base_dir = pathlib.Path("/content/drive/MyDrive/AML/cats_vs_dogs_small_subset")

    # 3. Create Subsets
    make_subset("train_4", original_train_dir, new_base_dir, 0, 850)
    make_subset("validation_4", original_val_dir, new_base_dir, 0, 250)
    make_subset("test_4", original_test_dir, new_base_dir, 0, 250)

    # 4. Load the Subset Datasets
    train_dataset_4 = image_dataset_from_directory(
        new_base_dir / "train_4",
        image_size=(180, 180),
        batch_size=32
    )

    validation_dataset_4 = image_dataset_from_directory(
        new_base_dir / "validation_4",
        image_size=(180, 180),
        batch_size=32
    )

    test_dataset_4 = image_dataset_from_directory(
        new_base_dir / "test_4",
        image_size=(180, 180),
        batch_size=32
    )

    # 5. Define the Callbacks
    early_stopping_monitor = keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=3,
        restore_best_weights=True
    )

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

    # 6. Build Your Model
    model = keras.Sequential([
        keras.layers.Rescaling(1./255, input_shape=(180, 180, 3)),
        keras.layers.Conv2D(32, (3, 3), activation='relu'),
        keras.layers.MaxPooling2D(),
        keras.layers.Conv2D(64, (3, 3), activation='relu'),
        keras.layers.MaxPooling2D(),
        keras.layers.Conv2D(128, (3, 3), activation='relu'),
        keras.layers.MaxPooling2D(),
        keras.layers.Flatten(),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(512, activation='relu'),
        keras.layers.Dense(1, activation='sigmoid')
    ])

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

    # 7. Train the Model
    history = model.fit(
```

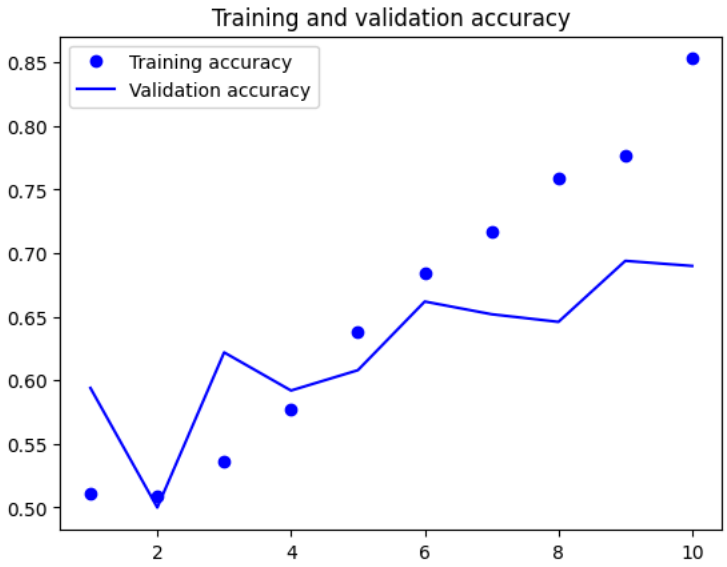
```
history = model.fit(
    train_dataset_4,
    epochs=30,
    validation_data=validation_dataset_4,
    callbacks=callbacks
)
```

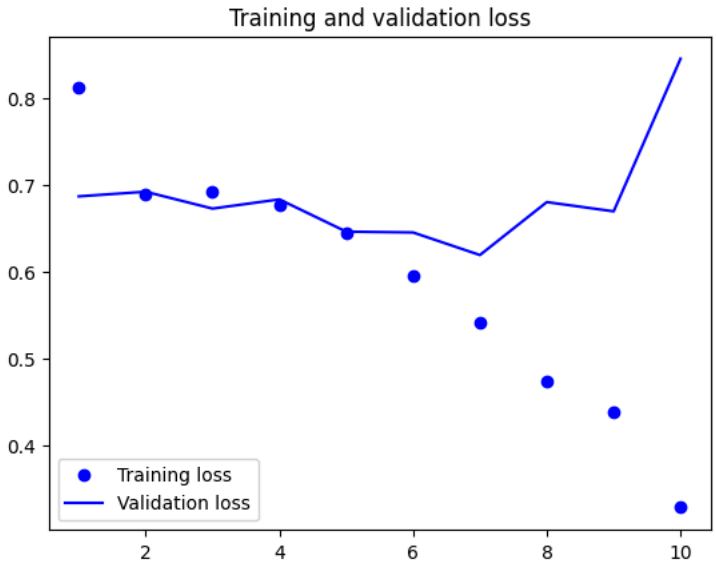
```
Creating subset: train_4 from /content/drive/MyDrive/AML/cats_vs_dogs_small/train [0:850]
Copying 850 files from /content/drive/MyDrive/AML/cats_vs_dogs_small/train/cats to /content/drive/MyDrive/AML/cats_vs_do
Copying 850 files from /content/drive/MyDrive/AML/cats_vs_dogs_small/train/dogs to /content/drive/MyDrive/AML/cats_vs_do
✅ Subset 'train_4' created successfully!

Creating subset: validation_4 from /content/drive/MyDrive/AML/cats_vs_dogs_small/validation [0:250]
Copying 250 files from /content/drive/MyDrive/AML/cats_vs_dogs_small/validation/cats to /content/drive/MyDrive/AML/cats_
Copying 250 files from /content/drive/MyDrive/AML/cats_vs_dogs_small/validation/dogs to /content/drive/MyDrive/AML/cats_
✅ Subset 'validation_4' created successfully!

Creating subset: test_4 from /content/drive/MyDrive/AML/cats_vs_dogs_small/test [0:250]
Copying 250 files from /content/drive/MyDrive/AML/cats_vs_dogs_small/test/cats to /content/drive/MyDrive/AML/cats_vs_dog
Copying 250 files from /content/drive/MyDrive/AML/cats_vs_dogs_small/test/dogs to /content/drive/MyDrive/AML/cats_vs_dog
✅ Subset 'test_4' created successfully!
Found 1700 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Found 500 files belonging to 2 classes.
Epoch 1/30
/usr/local/lib/python3.11/dist-packages/keras/src/layers/preprocessing/tf_data_layer.py:19: UserWarning: Do not pass an
  super().__init__(**kwargs)
54/54 ━━━━━━━━━━━━━━━━━━━━ 22s 293ms/step - accuracy: 0.5070 - loss: 1.0374 - val_accuracy: 0.5940 - val_loss: 0.6870
Epoch 2/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 192ms/step - accuracy: 0.5158 - loss: 0.6891 - val_accuracy: 0.5000 - val_loss: 0.6923
Epoch 3/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 20s 191ms/step - accuracy: 0.5276 - loss: 0.6942 - val_accuracy: 0.6220 - val_loss: 0.6729
Epoch 4/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 19s 157ms/step - accuracy: 0.5965 - loss: 0.6696 - val_accuracy: 0.5920 - val_loss: 0.6835
Epoch 5/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 16s 305ms/step - accuracy: 0.6088 - loss: 0.6624 - val_accuracy: 0.6080 - val_loss: 0.6463
Epoch 6/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 189ms/step - accuracy: 0.6745 - loss: 0.6144 - val_accuracy: 0.6620 - val_loss: 0.6453
Epoch 7/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 11s 201ms/step - accuracy: 0.7066 - loss: 0.5574 - val_accuracy: 0.6520 - val_loss: 0.6193
Epoch 8/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 9s 162ms/step - accuracy: 0.7519 - loss: 0.4839 - val_accuracy: 0.6460 - val_loss: 0.6804
Epoch 9/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 9s 141ms/step - accuracy: 0.7442 - loss: 0.4786 - val_accuracy: 0.6940 - val_loss: 0.6697
Epoch 10/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 176ms/step - accuracy: 0.8376 - loss: 0.3515 - val_accuracy: 0.6900 - val_loss: 0.8457
```

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

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


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



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")
```

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






```
test_model = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_model.evaluate(test_dataset_4)
print(f"Test accuracy: {test_acc:.3f}")
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 98ms/step - accuracy: 0.6135 - loss: 0.6971
Test accuracy: 0.620
```

Feature extraction together with data augmentation

Instantiating and freezing the VGG16 convolutional base

Pre-Trained Model - 1000 Training samples

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_t
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_13 (InputLayer)	(None, 180, 180, 3)	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1,792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36,928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73,856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147,584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295,168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590,080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590,080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 11, 11, 512)	0
block5_conv1 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0

Total params: 14,714,688 (56.13 MB)
Trainable params: 14,714,688 (56.13 MB)
Non-trainable params: 0 (0.00 B)

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

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

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

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

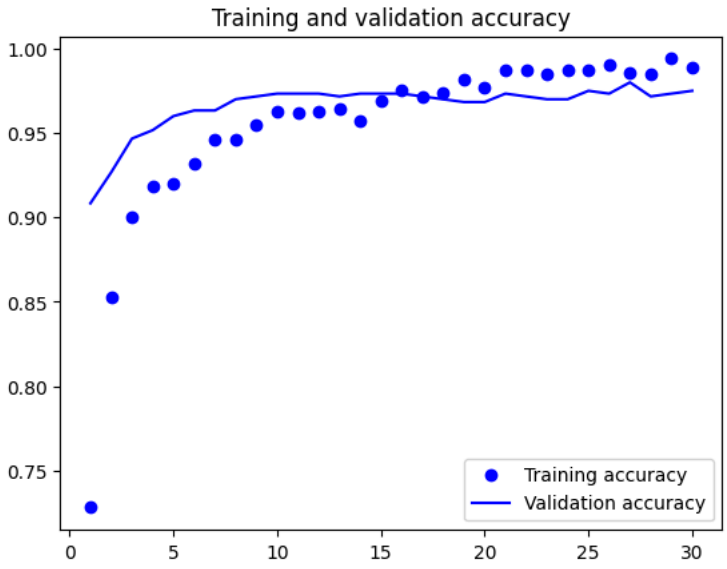
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])
```

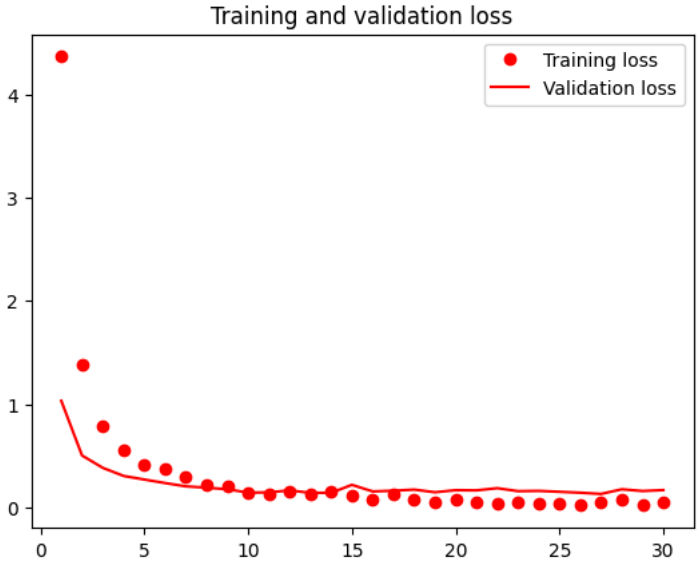
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
```

```
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 18s 289ms/step - accuracy: 0.6527 - loss: 6.6579 - val_accuracy: 0.9083 - val_loss: 1.0326
Epoch 2/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 14s 189ms/step - accuracy: 0.8351 - loss: 1.6189 - val_accuracy: 0.9267 - val_loss: 0.5032
Epoch 3/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 197ms/step - accuracy: 0.8742 - loss: 1.0359 - val_accuracy: 0.9467 - val_loss: 0.3834
Epoch 4/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 200ms/step - accuracy: 0.9015 - loss: 0.6770 - val_accuracy: 0.9517 - val_loss: 0.3052
Epoch 5/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 185ms/step - accuracy: 0.9178 - loss: 0.3953 - val_accuracy: 0.9600 - val_loss: 0.2702
Epoch 6/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 207ms/step - accuracy: 0.9236 - loss: 0.3897 - val_accuracy: 0.9633 - val_loss: 0.2361
Epoch 7/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 22s 243ms/step - accuracy: 0.9547 - loss: 0.2364 - val_accuracy: 0.9633 - val_loss: 0.2040
Epoch 8/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 187ms/step - accuracy: 0.9494 - loss: 0.2120 - val_accuracy: 0.9700 - val_loss: 0.1913
Epoch 9/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 167ms/step - accuracy: 0.9511 - loss: 0.2429 - val_accuracy: 0.9717 - val_loss: 0.1762
Epoch 10/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 189ms/step - accuracy: 0.9620 - loss: 0.1499 - val_accuracy: 0.9733 - val_loss: 0.1433
Epoch 11/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 191ms/step - accuracy: 0.9679 - loss: 0.1199 - val_accuracy: 0.9733 - val_loss: 0.1457
Epoch 12/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 12s 232ms/step - accuracy: 0.9587 - loss: 0.2294 - val_accuracy: 0.9733 - val_loss: 0.1653
Epoch 13/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 18s 170ms/step - accuracy: 0.9712 - loss: 0.1122 - val_accuracy: 0.9717 - val_loss: 0.1409
Epoch 14/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 185ms/step - accuracy: 0.9569 - loss: 0.1681 - val_accuracy: 0.9733 - val_loss: 0.1419
Epoch 15/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 183ms/step - accuracy: 0.9655 - loss: 0.1274 - val_accuracy: 0.9733 - val_loss: 0.2186
Epoch 16/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 170ms/step - accuracy: 0.9765 - loss: 0.0849 - val_accuracy: 0.9733 - val_loss: 0.1551
Epoch 17/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 168ms/step - accuracy: 0.9743 - loss: 0.1204 - val_accuracy: 0.9717 - val_loss: 0.1618
Epoch 18/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - accuracy: 0.9767 - loss: 0.0668 - val_accuracy: 0.9700 - val_loss: 0.1724
Epoch 19/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 181ms/step - accuracy: 0.9798 - loss: 0.0516 - val_accuracy: 0.9683 - val_loss: 0.1470
Epoch 20/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 176ms/step - accuracy: 0.9755 - loss: 0.0783 - val_accuracy: 0.9683 - val_loss: 0.1670
Epoch 21/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - accuracy: 0.9859 - loss: 0.0465 - val_accuracy: 0.9733 - val_loss: 0.1652
Epoch 22/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 169ms/step - accuracy: 0.9892 - loss: 0.0295 - val_accuracy: 0.9717 - val_loss: 0.1866
Epoch 23/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 183ms/step - accuracy: 0.9870 - loss: 0.0362 - val_accuracy: 0.9700 - val_loss: 0.1576
Epoch 24/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 181ms/step - accuracy: 0.9866 - loss: 0.0281 - val_accuracy: 0.9700 - val_loss: 0.1601
Epoch 25/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 153ms/step - accuracy: 0.9849 - loss: 0.0519 - val_accuracy: 0.9750 - val_loss: 0.1514
Epoch 26/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 152ms/step - accuracy: 0.9959 - loss: 0.0156 - val_accuracy: 0.9733 - val_loss: 0.1425
Epoch 27/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 201ms/step - accuracy: 0.9882 - loss: 0.0408 - val_accuracy: 0.9800 - val_loss: 0.1307
Epoch 28/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 172ms/step - accuracy: 0.9895 - loss: 0.0506 - val_accuracy: 0.9717 - val_loss: 0.1756
Epoch 29/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 177ms/step - accuracy: 0.9944 - loss: 0.0240 - val_accuracy: 0.9733 - val_loss: 0.1590
```

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






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

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - accuracy: 0.9706 - loss: 0.2744
Test accuracy: 0.968
```

Pre-Trained Model - 1500 Training samples

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


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

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


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

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning2.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=10,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 13s 284ms/step - accuracy: 0.9547 - loss: 0.1862 - val_accuracy: 0.9650 - val_loss: 0.2023
Epoch 2/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 224ms/step - accuracy: 0.9537 - loss: 0.2013 - val_accuracy: 0.9667 - val_loss: 0.2331
Epoch 3/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 17s 166ms/step - accuracy: 0.9573 - loss: 0.1198 - val_accuracy: 0.9667 - val_loss: 0.2546
Epoch 4/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 191ms/step - accuracy: 0.9638 - loss: 0.1789 - val_accuracy: 0.9667 - val_loss: 0.2199
Epoch 5/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 182ms/step - accuracy: 0.9718 - loss: 0.0955 - val_accuracy: 0.9667 - val_loss: 0.2299
Epoch 6/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 178ms/step - accuracy: 0.9609 - loss: 0.1488 - val_accuracy: 0.9683 - val_loss: 0.2285
Epoch 7/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 174ms/step - accuracy: 0.9690 - loss: 0.1466 - val_accuracy: 0.9683 - val_loss: 0.2194
Epoch 8/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 193ms/step - accuracy: 0.9747 - loss: 0.0786 - val_accuracy: 0.9700 - val_loss: 0.2320
Epoch 9/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 192ms/step - accuracy: 0.9790 - loss: 0.0690 - val_accuracy: 0.9700 - val_loss: 0.2194
Epoch 10/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 181ms/step - accuracy: 0.9781 - loss: 0.1182 - val_accuracy: 0.9733 - val_loss: 0.2140
```

```
model = keras.models.load_model("fine_tuning2.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - accuracy: 0.9655 - loss: 0.2444
Test accuracy: 0.970
```

Pre-Trained Model - 1700 samples

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

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

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

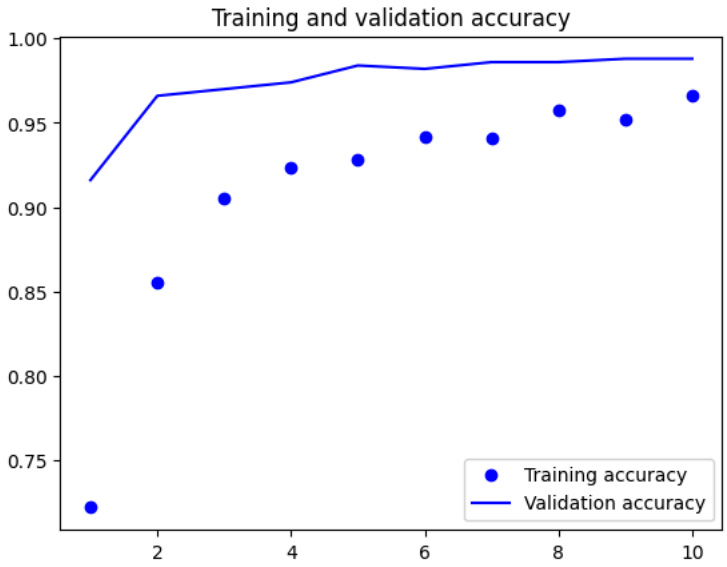
```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

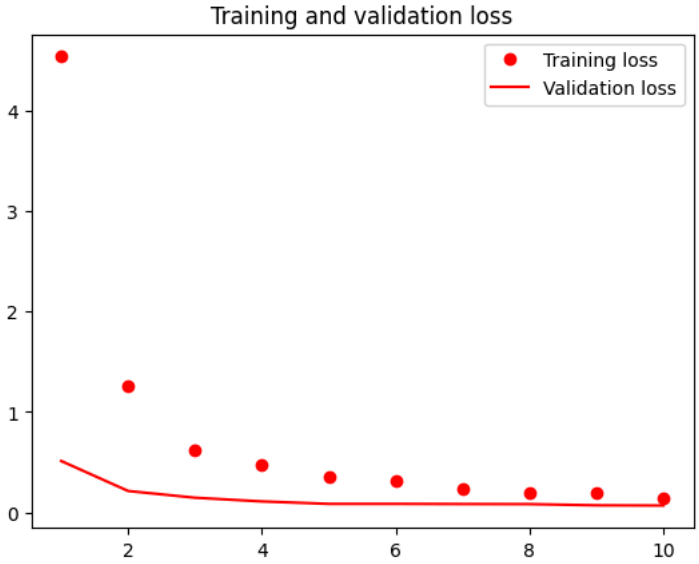
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
```

```
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning3.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset_4,
    epochs=10,
    validation_data=validation_dataset_4,
    callbacks=callbacks)
```

```
Epoch 1/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.6422 - loss: 6.8720 - val_accuracy: 0.9160 - val_loss: 0.5122
Epoch 2/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 32s 170ms/step - accuracy: 0.8464 - loss: 1.3208 - val_accuracy: 0.9660 - val_loss: 0.2141
Epoch 3/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.9149 - loss: 0.5365 - val_accuracy: 0.9700 - val_loss: 0.1471
Epoch 4/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 11s 189ms/step - accuracy: 0.9198 - loss: 0.4791 - val_accuracy: 0.9740 - val_loss: 0.1103
Epoch 5/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 19s 173ms/step - accuracy: 0.9230 - loss: 0.3690 - val_accuracy: 0.9840 - val_loss: 0.0859
Epoch 6/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 8s 151ms/step - accuracy: 0.9363 - loss: 0.2982 - val_accuracy: 0.9820 - val_loss: 0.0860
Epoch 7/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 12s 190ms/step - accuracy: 0.9461 - loss: 0.2176 - val_accuracy: 0.9860 - val_loss: 0.0844
Epoch 8/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 11s 196ms/step - accuracy: 0.9510 - loss: 0.2066 - val_accuracy: 0.9860 - val_loss: 0.0835
Epoch 9/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 19s 173ms/step - accuracy: 0.9442 - loss: 0.2365 - val_accuracy: 0.9880 - val_loss: 0.0717
Epoch 10/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 10s 190ms/step - accuracy: 0.9657 - loss: 0.1672 - val_accuracy: 0.9880 - val_loss: 0.0696
```

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






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

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 125ms/step - accuracy: 0.9616 - loss: 0.2552
Test accuracy: 0.965
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.