Assignment-2

This project showcases the trade-offs between training from scratch and transfer learning in image classification:

From-scratch models can be ~70% accurate when well regularized, even with 1000 images.Bigger training sizes more than 1500 provide decreasing returns unless augmented with regularization.Transfer learning beats from-scratch models by a huge margin, with >85% accuracy on less data and lower test loss.Best overall performance was obtained using InceptionV3 trained on 1700 images, at 86.5% test accuracy and 0.32 loss.

Data Loading

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

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

Organizing images into training, validation, and testing directories

```
import os
import shutil
import pathlib

# Define original folders separately
train_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/train")
val_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/validation")
test_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/test")

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

def create_dataset_subset(subset_type, src_folder, start_index, end_index):

    print(f"\nCreating '{subset_type}' from {src_folder} indices {start_index} to {end_index}")

    for label_type in ["cats", "dogs"]:
        src_label_path = src_folder / label_type
        dest_label_path = balanced_dataset_path / subset_type / label_type

        os.makedirs(dest_label_path, exist_ok=True)

        files = sorted(os.listdir(src_label_path))


        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)} '{label_type}' images to '{dest_label_path}'...")

        for fname in subset_files:
            src_file = src_label_path / fname
            dst_file = dest_label_path / fname
            shutil.copyfile(src_file, dst_file)

    print(f" {subset_type} created from {src_folder}.")

def main():



    create_dataset_subset("train", train_src_path, 0, 500)
    create_dataset_subset("validation", val_src_path,0,250)
    create_dataset_subset("test",   test_src_path, 0, 250)

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

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

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

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

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    balanced_dataset_path / "train",
    image_size=(180, 180),
    batch_size=32
)

val_dataset = image_dataset_from_directory(
    balanced_dataset_path / "validation",
    image_size=(180, 180),
    batch_size=32
)

test_dataset = image_dataset_from_directory(
    balanced_dataset_path / "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.
```

Training a convolutional neural network from scratch

Model 1:The dataset is divided into 1000 training samples, 500 validation samples, and 500 test samples.

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

Designing a simple deep learning model to differentiate between cat and dog images.

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

```
model.summary()
```

Model: "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
dense (Dense)	(None, 1)	12,545

Total params: 991,041 (3.78 MB)
Trainable params: 991,041 (3.78 MB)
Non-trainable params: 0 (0.00 B)

Preparing the model for training

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

Data preparation

```
from tensorflow.keras.utils import image_dataset_from_directory

# Base directory for the subsets
balanced_subset_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/small_dataset_balanced")

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

val_dataset = image_dataset_from_directory(
    balanced_subset_path / "validation",
    image_size=(180, 180),
    batch_size=32
)

test_dataset = image_dataset_from_directory(
    balanced_subset_path / "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.
```

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


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

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

```
(16,)
```

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

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

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

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

Displaying the shape of input features and corresponding labels from 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,)
```

Model training using the dataset

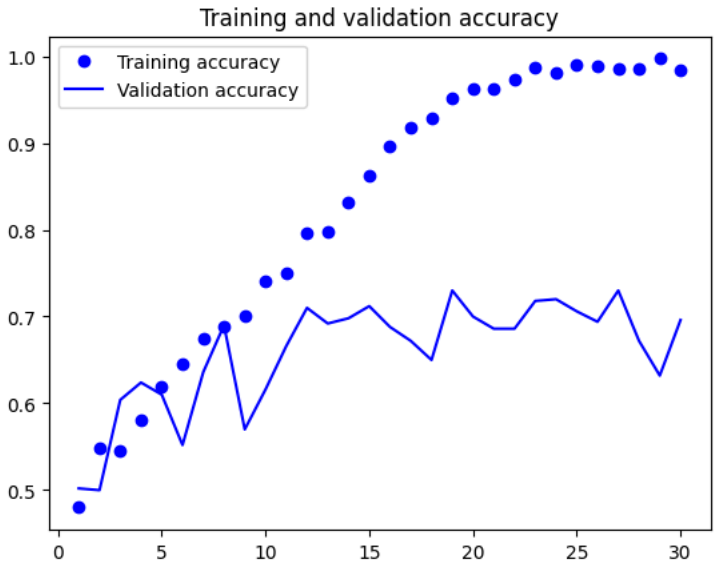
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=val_dataset,
    callbacks=callbacks)
```

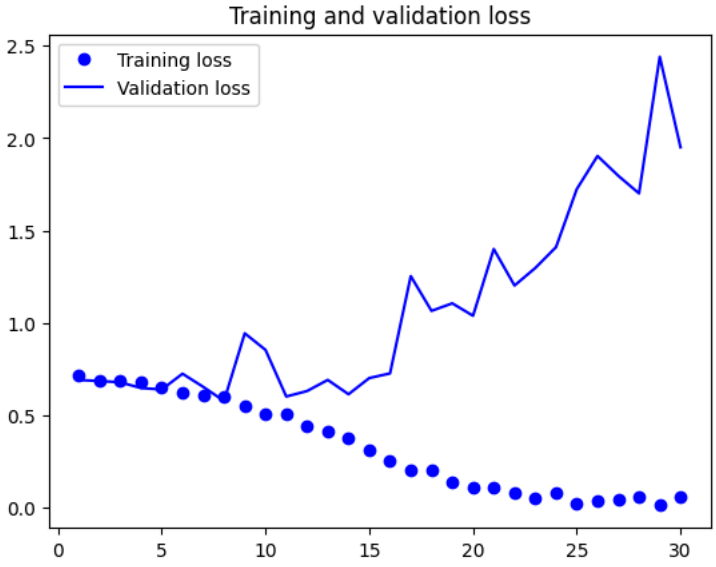
```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 168ms/step - accuracy: 0.5253 - loss: 0.6913 - val_accuracy: 0.5000 - val_loss: 0.6870
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 192ms/step - accuracy: 0.5525 - loss: 0.6899 - val_accuracy: 0.6040 - val_loss: 0.6799
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 153ms/step - accuracy: 0.5873 - loss: 0.6857 - val_accuracy: 0.6240 - val_loss: 0.6488
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 154ms/step - accuracy: 0.6042 - loss: 0.6499 - val_accuracy: 0.6100 - val_loss: 0.6408
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 187ms/step - accuracy: 0.6538 - loss: 0.6153 - val_accuracy: 0.5520 - val_loss: 0.7262
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 185ms/step - accuracy: 0.6616 - loss: 0.6160 - val_accuracy: 0.6360 - val_loss: 0.6550
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 186ms/step - accuracy: 0.6945 - loss: 0.5885 - val_accuracy: 0.6900 - val_loss: 0.5790
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 185ms/step - accuracy: 0.7264 - loss: 0.5269 - val_accuracy: 0.5700 - val_loss: 0.9447
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 169ms/step - accuracy: 0.7376 - loss: 0.5126 - val_accuracy: 0.6160 - val_loss: 0.8551
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 12s 214ms/step - accuracy: 0.7228 - loss: 0.5506 - val_accuracy: 0.6660 - val_loss: 0.6030
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.8024 - loss: 0.4456 - val_accuracy: 0.7100 - val_loss: 0.6324
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 183ms/step - accuracy: 0.7831 - loss: 0.4266 - val_accuracy: 0.6920 - val_loss: 0.6927
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 195ms/step - accuracy: 0.8313 - loss: 0.3689 - val_accuracy: 0.6980 - val_loss: 0.6153
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 160ms/step - accuracy: 0.8637 - loss: 0.3024 - val_accuracy: 0.7120 - val_loss: 0.7028
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 187ms/step - accuracy: 0.8814 - loss: 0.2717 - val_accuracy: 0.6880 - val_loss: 0.7279
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 163ms/step - accuracy: 0.9056 - loss: 0.2141 - val_accuracy: 0.6720 - val_loss: 1.2541
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.8969 - loss: 0.3036 - val_accuracy: 0.6500 - val_loss: 1.0661
```

```
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 180ms/step - accuracy: 0.9517 - loss: 0.1190 - val_accuracy: 0.7000 - val_loss: 1.0398
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 147ms/step - accuracy: 0.9700 - loss: 0.0873 - val_accuracy: 0.6860 - val_loss: 1.4006
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 187ms/step - accuracy: 0.9873 - loss: 0.0446 - val_accuracy: 0.6860 - val_loss: 1.2032
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 182ms/step - accuracy: 0.9905 - loss: 0.0457 - val_accuracy: 0.7180 - val_loss: 1.2979
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 185ms/step - accuracy: 0.9912 - loss: 0.0360 - val_accuracy: 0.7200 - val_loss: 1.4110
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 181ms/step - accuracy: 0.9769 - loss: 0.0440 - val_accuracy: 0.7060 - val_loss: 1.7246
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.9901 - loss: 0.0359 - val_accuracy: 0.6940 - val_loss: 1.9039
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 149ms/step - accuracy: 0.9748 - loss: 0.0664 - val_accuracy: 0.7300 - val_loss: 1.7970
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.9888 - loss: 0.0481 - val_accuracy: 0.6720 - val_loss: 1.7015
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 179ms/step - accuracy: 0.9969 - loss: 0.0139 - val_accuracy: 0.6320 - val_loss: 2.4406
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 146ms/step - accuracy: 0.9871 - loss: 0.0366 - val_accuracy: 0.6960 - val_loss: 1.9511
```

Monitoring accuracy and loss curves throughout the training process

```
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 dataset

```
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 94ms/step - accuracy: 0.6797 - loss: 0.6229
Test accuracy: 0.684
```

Since the validation and the tes accuracy of the model is very low that is 68.4%

To enhance the performance of our CNN trained from scratch, we applied the following techniques during model development.

Model 1a: Applying 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=val_dataset,
     callbacks=callbacks)
```

```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 198ms/step - accuracy: 0.5041 - loss: 0.6970 - val_accuracy: 0.5020 - val_loss: 0.6912
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 183ms/step - accuracy: 0.5064 - loss: 0.6954 - val_accuracy: 0.5020 - val_loss: 0.6906
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 172ms/step - accuracy: 0.5338 - loss: 0.6937 - val_accuracy: 0.5060 - val_loss: 0.6883
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 195ms/step - accuracy: 0.5881 - loss: 0.6861 - val_accuracy: 0.5080 - val_loss: 0.6971
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 145ms/step - accuracy: 0.5966 - loss: 0.6783 - val_accuracy: 0.5700 - val_loss: 0.6726
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 184ms/step - accuracy: 0.6361 - loss: 0.6492 - val_accuracy: 0.6540 - val_loss: 0.6455
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 181ms/step - accuracy: 0.6223 - loss: 0.6495 - val_accuracy: 0.5900 - val_loss: 0.6897
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 142ms/step - accuracy: 0.6210 - loss: 0.6487 - val_accuracy: 0.5960 - val_loss: 0.7538
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 179ms/step - accuracy: 0.6460 - loss: 0.6927 - val_accuracy: 0.5740 - val_loss: 0.7202
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 151ms/step - accuracy: 0.6458 - loss: 0.6431 - val_accuracy: 0.5880 - val_loss: 0.6964
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 142ms/step - accuracy: 0.6665 - loss: 0.5986 - val_accuracy: 0.6240 - val_loss: 0.6996
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 220ms/step - accuracy: 0.6604 - loss: 0.6224 - val_accuracy: 0.6540 - val_loss: 0.6121
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 141ms/step - accuracy: 0.6850 - loss: 0.5877 - val_accuracy: 0.6480 - val_loss: 0.6169
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 176ms/step - accuracy: 0.6965 - loss: 0.5999 - val_accuracy: 0.6760 - val_loss: 0.5833
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 188ms/step - accuracy: 0.7106 - loss: 0.5714 - val_accuracy: 0.7060 - val_loss: 0.5676
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 150ms/step - accuracy: 0.7348 - loss: 0.5369 - val_accuracy: 0.7020 - val_loss: 0.5686
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 175ms/step - accuracy: 0.7169 - loss: 0.5504 - val_accuracy: 0.6460 - val_loss: 0.6427
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 147ms/step - accuracy: 0.7172 - loss: 0.5593 - val_accuracy: 0.6640 - val_loss: 0.6898
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 188ms/step - accuracy: 0.6764 - loss: 0.5817 - val_accuracy: 0.6300 - val_loss: 0.9670
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 186ms/step - accuracy: 0.7023 - loss: 0.5771 - val_accuracy: 0.7220 - val_loss: 0.5389
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 211ms/step - accuracy: 0.7572 - loss: 0.4932 - val_accuracy: 0.7100 - val_loss: 0.5573
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 180ms/step - accuracy: 0.7307 - loss: 0.5107 - val_accuracy: 0.7380 - val_loss: 0.5495
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 189ms/step - accuracy: 0.7655 - loss: 0.4892 - val_accuracy: 0.7360 - val_loss: 0.5446
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 188ms/step - accuracy: 0.7766 - loss: 0.4800 - val_accuracy: 0.7040 - val_loss: 0.5712
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 191ms/step - accuracy: 0.7836 - loss: 0.4926 - val_accuracy: 0.7380 - val_loss: 0.5318
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 160ms/step - accuracy: 0.7449 - loss: 0.5242 - val_accuracy: 0.7060 - val_loss: 0.5903
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 12s 221ms/step - accuracy: 0.7570 - loss: 0.4987 - val_accuracy: 0.7460 - val_loss: 0.5145
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 185ms/step - accuracy: 0.8019 - loss: 0.4339 - val_accuracy: 0.6620 - val_loss: 1.0475
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 190ms/step - accuracy: 0.7403 - loss: 0.5680 - val_accuracy: 0.7340 - val_loss: 0.5957
```

```
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 94ms/step - accuracy: 0.7337 - loss: 0.5534
Test accuracy: 0.736
```

Implementing image data augmentation for model training

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

Visualizing randomly augmented samples from the training dataset

```
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: Applying the Dropout Technique

```
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=val_dataset,
    callbacks=callbacks)
```

```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 183ms/step - accuracy: 0.4913 - loss: 0.6934 - val_accuracy: 0.5000 - val_loss: 0.6923
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 179ms/step - accuracy: 0.4840 - loss: 0.6936 - val_accuracy: 0.5000 - val_loss: 0.9418
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 176ms/step - accuracy: 0.5760 - loss: 0.7400 - val_accuracy: 0.5000 - val_loss: 0.7059
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 194ms/step - accuracy: 0.5389 - loss: 0.6901 - val_accuracy: 0.5020 - val_loss: 0.6954
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 201ms/step - accuracy: 0.5410 - loss: 0.6864 - val_accuracy: 0.5320 - val_loss: 0.6812
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 142ms/step - accuracy: 0.5741 - loss: 0.6782 - val_accuracy: 0.5280 - val_loss: 0.6784
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 180ms/step - accuracy: 0.6130 - loss: 0.6637 - val_accuracy: 0.5780 - val_loss: 0.6514
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 140ms/step - accuracy: 0.6373 - loss: 0.6312 - val_accuracy: 0.6140 - val_loss: 0.7329
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 210ms/step - accuracy: 0.6371 - loss: 0.6409 - val_accuracy: 0.6380 - val_loss: 0.6347
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 140ms/step - accuracy: 0.7278 - loss: 0.5454 - val_accuracy: 0.6540 - val_loss: 0.6431
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 216ms/step - accuracy: 0.7349 - loss: 0.5555 - val_accuracy: 0.6440 - val_loss: 0.7299
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 167ms/step - accuracy: 0.7439 - loss: 0.5273 - val_accuracy: 0.5620 - val_loss: 1.0724
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 160ms/step - accuracy: 0.7408 - loss: 0.5270 - val_accuracy: 0.6100 - val_loss: 0.7999
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 148ms/step - accuracy: 0.7370 - loss: 0.5123 - val_accuracy: 0.6620 - val_loss: 0.6223
Epoch 16/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 212ms/step - accuracy: 0.7949 - loss: 0.4660 - val_accuracy: 0.6920 - val_loss: 0.6123
Epoch 17/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 146ms/step - accuracy: 0.7882 - loss: 0.4421 - val_accuracy: 0.6120 - val_loss: 0.8673
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 174ms/step - accuracy: 0.7922 - loss: 0.4414 - val_accuracy: 0.6880 - val_loss: 0.5971
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 195ms/step - accuracy: 0.8261 - loss: 0.3765 - val_accuracy: 0.6560 - val_loss: 0.7737
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 180ms/step - accuracy: 0.8554 - loss: 0.3561 - val_accuracy: 0.6620 - val_loss: 0.8839
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 181ms/step - accuracy: 0.8527 - loss: 0.3483 - val_accuracy: 0.6880 - val_loss: 0.9825
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 186ms/step - accuracy: 0.8625 - loss: 0.3103 - val_accuracy: 0.6960 - val_loss: 0.7441
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 181ms/step - accuracy: 0.8951 - loss: 0.2565 - val_accuracy: 0.6840 - val_loss: 0.7468
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 163ms/step - accuracy: 0.9198 - loss: 0.2073 - val_accuracy: 0.6760 - val_loss: 1.1262
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 186ms/step - accuracy: 0.9186 - loss: 0.2195 - val_accuracy: 0.6940 - val_loss: 0.8937
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 140ms/step - accuracy: 0.9156 - loss: 0.2002 - val_accuracy: 0.7040 - val_loss: 0.8649
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 171ms/step - accuracy: 0.9346 - loss: 0.1577 - val_accuracy: 0.7200 - val_loss: 0.9130
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 177ms/step - accuracy: 0.9600 - loss: 0.1142 - val_accuracy: 0.6480 - val_loss: 1.5538
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 139ms/step - accuracy: 0.9209 - loss: 0.1480 - val_accuracy: 0.7160 - val_loss: 1.0177
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 177ms/step - accuracy: 0.9580 - loss: 0.1130 - val_accuracy: 0.7080 - val_loss: 1.1780
```

```
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 91ms/step - accuracy: 0.6541 - loss: 0.6612
Test accuracy: 0.658
```

Model 1c: Applying both Image Augmentation and Dropout techniques

```
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=val_dataset,
    callbacks=callbacks)
```

```
Epoch 2/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 201ms/step - accuracy: 0.5056 - loss: 0.6927 - val_accuracy: 0.5000 - val_loss: 0.7182
Epoch 3/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 193ms/step - accuracy: 0.5408 - loss: 0.6936 - val_accuracy: 0.5820 - val_loss: 0.6783
Epoch 4/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 159ms/step - accuracy: 0.5826 - loss: 0.6849 - val_accuracy: 0.6380 - val_loss: 0.6673
Epoch 5/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 206ms/step - accuracy: 0.6042 - loss: 0.6781 - val_accuracy: 0.6400 - val_loss: 0.6388
Epoch 6/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 189ms/step - accuracy: 0.5973 - loss: 0.6694 - val_accuracy: 0.6260 - val_loss: 0.6360
Epoch 7/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 175ms/step - accuracy: 0.6285 - loss: 0.6606 - val_accuracy: 0.5960 - val_loss: 0.6845
Epoch 8/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 154ms/step - accuracy: 0.6358 - loss: 0.6401 - val_accuracy: 0.5840 - val_loss: 0.6501
Epoch 9/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 189ms/step - accuracy: 0.6557 - loss: 0.6347 - val_accuracy: 0.6700 - val_loss: 0.5977
Epoch 10/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 185ms/step - accuracy: 0.6563 - loss: 0.6036 - val_accuracy: 0.6080 - val_loss: 0.6918
Epoch 11/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 173ms/step - accuracy: 0.6391 - loss: 0.6270 - val_accuracy: 0.6780 - val_loss: 0.6153
Epoch 12/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 184ms/step - accuracy: 0.6950 - loss: 0.5908 - val_accuracy: 0.6340 - val_loss: 0.6147
Epoch 13/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.6983 - loss: 0.5829 - val_accuracy: 0.6480 - val_loss: 0.6033
Epoch 14/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 185ms/step - accuracy: 0.6970 - loss: 0.5890 - val_accuracy: 0.6660 - val_loss: 0.5822
Epoch 15/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 181ms/step - accuracy: 0.6807 - loss: 0.5941 - val_accuracy: 0.6460 - val_loss: 0.6266
Epoch 16/30
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 153ms/step - accuracy: 0.7216 - loss: 0.5656 - val_accuracy: 0.7020 - val_loss: 0.5599
Epoch 18/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 189ms/step - accuracy: 0.7120 - loss: 0.5470 - val_accuracy: 0.6780 - val_loss: 0.6197
Epoch 19/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 150ms/step - accuracy: 0.7094 - loss: 0.5595 - val_accuracy: 0.7060 - val_loss: 0.6341
Epoch 20/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 181ms/step - accuracy: 0.7341 - loss: 0.5201 - val_accuracy: 0.6620 - val_loss: 0.6268
Epoch 21/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 168ms/step - accuracy: 0.7382 - loss: 0.5229 - val_accuracy: 0.7100 - val_loss: 0.5435
Epoch 22/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 201ms/step - accuracy: 0.7508 - loss: 0.5174 - val_accuracy: 0.6040 - val_loss: 0.9094
Epoch 23/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 141ms/step - accuracy: 0.7420 - loss: 0.5336 - val_accuracy: 0.7260 - val_loss: 0.5459
Epoch 24/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.7335 - loss: 0.5199 - val_accuracy: 0.6460 - val_loss: 0.6467
Epoch 25/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.7352 - loss: 0.5349 - val_accuracy: 0.7120 - val_loss: 0.6223
Epoch 26/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 203ms/step - accuracy: 0.7559 - loss: 0.4917 - val_accuracy: 0.7520 - val_loss: 0.5276
Epoch 27/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 196ms/step - accuracy: 0.7721 - loss: 0.4923 - val_accuracy: 0.7060 - val_loss: 0.6232
Epoch 28/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 142ms/step - accuracy: 0.7810 - loss: 0.4707 - val_accuracy: 0.7200 - val_loss: 0.6158
Epoch 29/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 182ms/step - accuracy: 0.7761 - loss: 0.4888 - val_accuracy: 0.7320 - val_loss: 0.5235
Epoch 30/30
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 149ms/step - accuracy: 0.7772 - loss: 0.4602 - val_accuracy: 0.7580 - val_loss: 0.5158
```

```
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 95ms/step - accuracy: 0.7176 - loss: 0.5982
Test accuracy: 0.722
```

Model 2) Expanded the training dataset to 1500 samples and enhanced the model using MaxPooling layers, data augmentation techniques, and Dropout regularization with a 0.5 rate.

```
import os
import shutil
import pathlib

# Define original folders (your dataset path)
train_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/train")
val_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/validation")
test_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/test")

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

def create_dataset_subset(subset_type, src_folder, start_index, end_index):
    print(f"\nCreating '{subset_type}' subset from {src_folder} [{start_index}:{end_index}]")

    for label_type in ["cats", "dogs"]:
        src_label_path = src_folder / label_type
        dest_label_path = balanced_dataset_path / subset_type / label_type

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

        os.makedirs(dest_label_path, exist_ok=True)

        files = sorted(os.listdir(src_label_path))

        # 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)} '{label_type}' images to '{dest_label_path}'...")

        for fname in subset_files:
            src_file = src_label_path / fname
            dst_file = dest_label_path / fname
            shutil.copyfile(src_file, dst_file)

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

def main():
```

```
    create_dataset_subset("train", train_src_path, 0, 750)
    create_dataset_subset("validation", val_src_path, 0, 300)
    create_dataset_subset("test",   test_src_path, 0, 300)

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

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

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

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

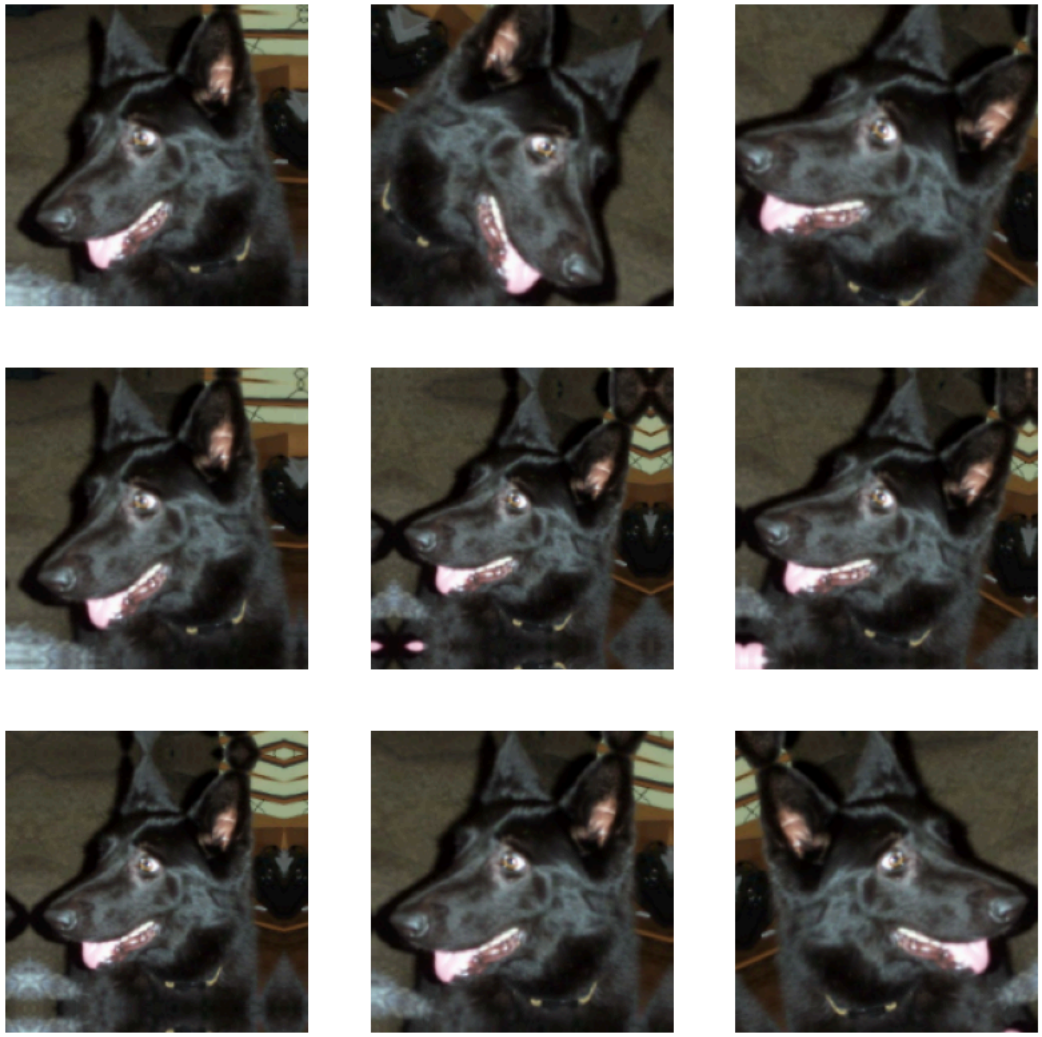
```
from tensorflow.keras.utils import image_dataset_from_directory
from pathlib import Path

balanced_dataset_path = Path("/content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset")

train_dataset = image_dataset_from_directory(
    balanced_dataset_path / "train",
    image_size=(180, 180),
    batch_size=32
)

val_dataset = image_dataset_from_directory(
    balanced_dataset_path / "validation",
    image_size=(180, 180),
    batch_size=32
)

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

# Prefetch for speed
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
val_dataset = val_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.
```

Developing a custom CNN architecture incorporating image augmentation and dropout for improved generalization.

```
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"])
```

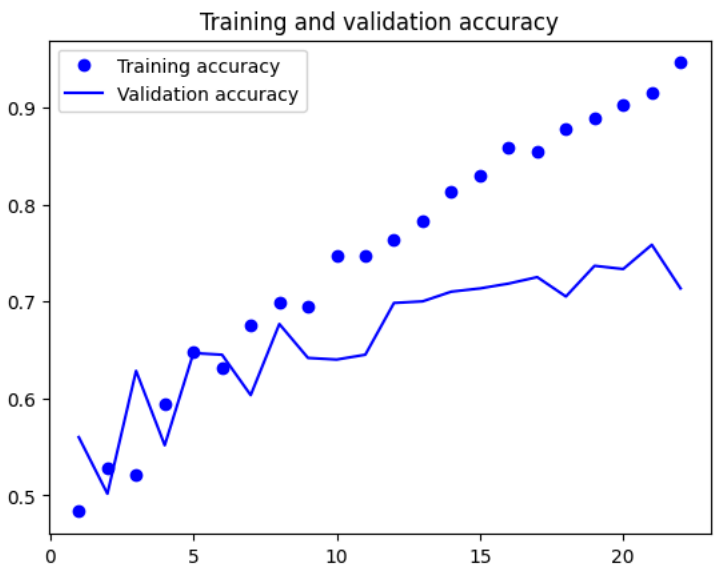
```
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from keras import regularizers

# 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")
```

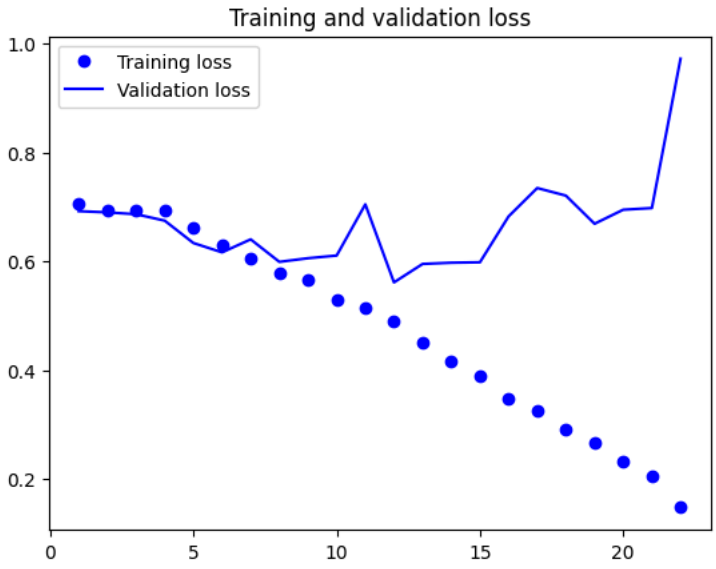


```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss"), early_stopping_monitor
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=val_dataset,
    callbacks=callbacks)
```

```
Epoch 1/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 18s 325ms/step - accuracy: 0.4780 - loss: 0.7347 - val_accuracy: 0.5600 - val_loss: 0.6922
Epoch 2/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 148ms/step - accuracy: 0.5181 - loss: 0.6943 - val_accuracy: 0.5017 - val_loss: 0.6900
Epoch 3/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 146ms/step - accuracy: 0.5092 - loss: 0.6945 - val_accuracy: 0.6283 - val_loss: 0.6866
Epoch 4/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 159ms/step - accuracy: 0.5771 - loss: 0.6901 - val_accuracy: 0.5517 - val_loss: 0.6747
Epoch 5/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 12s 202ms/step - accuracy: 0.6197 - loss: 0.6777 - val_accuracy: 0.6467 - val_loss: 0.6336
Epoch 6/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 224ms/step - accuracy: 0.6002 - loss: 0.6405 - val_accuracy: 0.6450 - val_loss: 0.6165
Epoch 7/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 172ms/step - accuracy: 0.6668 - loss: 0.6102 - val_accuracy: 0.6033 - val_loss: 0.6404
Epoch 8/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 188ms/step - accuracy: 0.6885 - loss: 0.5802 - val_accuracy: 0.6767 - val_loss: 0.5990
Epoch 9/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 222ms/step - accuracy: 0.6925 - loss: 0.5678 - val_accuracy: 0.6417 - val_loss: 0.6057
Epoch 10/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 153ms/step - accuracy: 0.7374 - loss: 0.5309 - val_accuracy: 0.6400 - val_loss: 0.6106
Epoch 11/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 187ms/step - accuracy: 0.7321 - loss: 0.5352 - val_accuracy: 0.6450 - val_loss: 0.7047
Epoch 12/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 183ms/step - accuracy: 0.7415 - loss: 0.5238 - val_accuracy: 0.6983 - val_loss: 0.5613
Epoch 13/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 157ms/step - accuracy: 0.7744 - loss: 0.4558 - val_accuracy: 0.7000 - val_loss: 0.5952
Epoch 14/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 181ms/step - accuracy: 0.7783 - loss: 0.4504 - val_accuracy: 0.7100 - val_loss: 0.5974
Epoch 15/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 163ms/step - accuracy: 0.8125 - loss: 0.4043 - val_accuracy: 0.7133 - val_loss: 0.5982
Epoch 16/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 137ms/step - accuracy: 0.8386 - loss: 0.3701 - val_accuracy: 0.7183 - val_loss: 0.6826
Epoch 17/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 180ms/step - accuracy: 0.8406 - loss: 0.3878 - val_accuracy: 0.7250 - val_loss: 0.7347
Epoch 18/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 141ms/step - accuracy: 0.8710 - loss: 0.2989 - val_accuracy: 0.7050 - val_loss: 0.7207
Epoch 19/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 156ms/step - accuracy: 0.8703 - loss: 0.2939 - val_accuracy: 0.7367 - val_loss: 0.6690
Epoch 20/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 177ms/step - accuracy: 0.8881 - loss: 0.2632 - val_accuracy: 0.7333 - val_loss: 0.6948
Epoch 21/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 152ms/step - accuracy: 0.9119 - loss: 0.2086 - val_accuracy: 0.7583 - val_loss: 0.6979
Epoch 22/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9423 - loss: 0.1560 - val_accuracy: 0.7133 - val_loss: 0.9723
```

```
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 92ms/step - accuracy: 0.7038 - loss: 0.6168
Test accuracy: 0.695
```

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_type, original_dir, balanced_dataset_path, start_index, end_index):
    print(f"\nCreating subset: {subset_type} from {original_dir} [{start_index}:{end_index}]")

    for label_type in ["cats", "dogs"]:
        src_label_path = original_dir / label_type
        dest_label_path = balanced_dataset_path / subset_type / label_type

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

        os.makedirs(dest_label_path, exist_ok=True)

        files = sorted(os.listdir(src_label_path))

        if end_index > len(files):
            print(f" 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)} files from {src_label_path} to {dest_label_path}...")

        for fname in subset_files:
            src_file = src_label_path / fname
            dst_file = dest_label_path / fname

            shutil.copyfile(src_file, dst_file)

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

# 2. Paths Setup
train_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/train")
val_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/validation")
test_src_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/test")

balanced_dataset_path = pathlib.Path("/content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset")

# 3. Create Subsets
make_subset("train_4", train_src_path, balanced_dataset_path, 0, 850)
make_subset("validation_4", val_src_path, balanced_dataset_path, 0, 250)
make_subset("test_4",   test_src_path, balanced_dataset_path, 0, 250)

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

val_dataset_4 = image_dataset_from_directory(
    balanced_dataset_path / "validation_4",
    image_size=(180, 180),
    batch_size=32
)

test_dataset_4 = image_dataset_from_directory(
    balanced_dataset_path / "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']
)
```

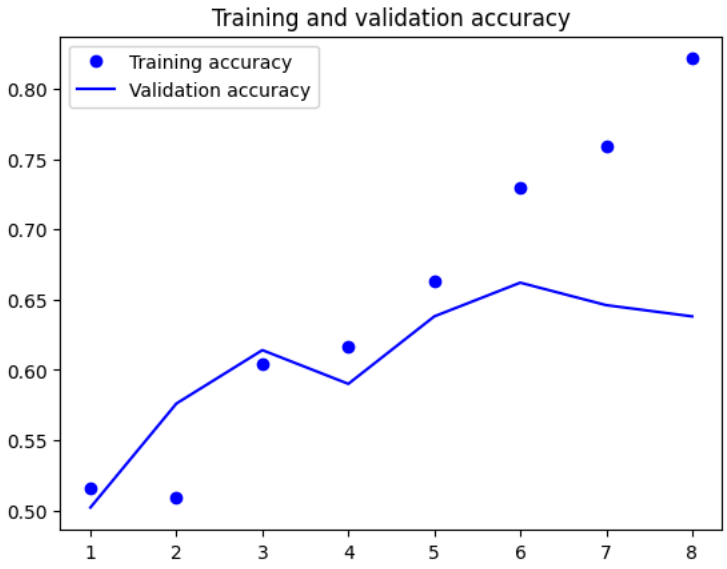
```
# 7. Train the Model
history = model.fit(
    train_dataset_4,
    epochs=30,
    validation_data=val_dataset_4,
    callbacks=callbacks
)
```

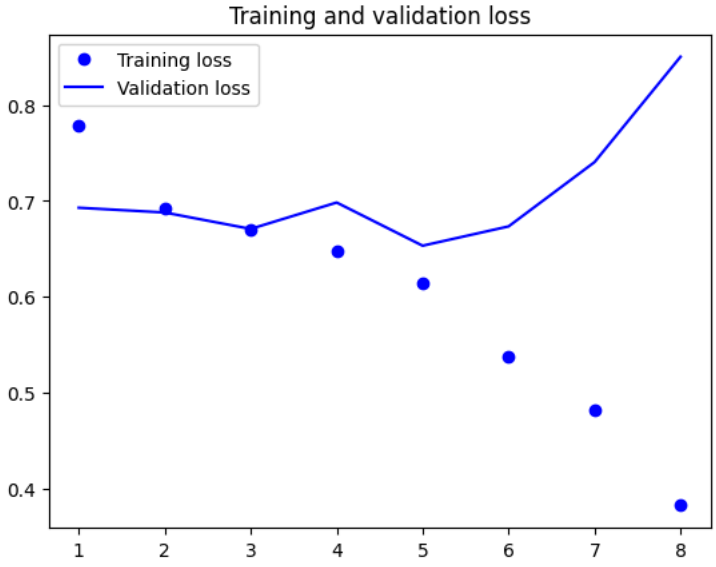
```
Creating subset: train_4 from /content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/train [0:850]
 end_index 850 exceeds available files (750). Adjusting...
Copying 750 files from /content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/train/cats to /content/drive/
Copying 750 files from /content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/train/dogs to /content/drive/
Subset 'train_4' created successfully!

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

Creating subset: test_4 from /content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/test [0:250]
Copying 250 files from /content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/test/cats to /content/drive/M
Copying 250 files from /content/drive/MyDrive/cats_vs_dogs_small/cats_vs_dogs_small_subset/test/dogs to /content/drive/M
Subset 'test_4' created successfully!
Found 1500 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)
47/47 ━━━━━━━━━━━━━━━━━━━━ 21s 273ms/step - accuracy: 0.5057 - loss: 0.9617 - val_accuracy: 0.5020 - val_loss: 0.6930
Epoch 2/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 13s 234ms/step - accuracy: 0.4999 - loss: 0.6939 - val_accuracy: 0.5760 - val_loss: 0.6880
Epoch 3/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 205ms/step - accuracy: 0.5959 - loss: 0.6812 - val_accuracy: 0.6140 - val_loss: 0.6709
Epoch 4/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 145ms/step - accuracy: 0.5938 - loss: 0.6648 - val_accuracy: 0.5900 - val_loss: 0.6984
Epoch 5/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 16s 271ms/step - accuracy: 0.6591 - loss: 0.6378 - val_accuracy: 0.6380 - val_loss: 0.6534
Epoch 6/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 16s 170ms/step - accuracy: 0.7164 - loss: 0.5587 - val_accuracy: 0.6620 - val_loss: 0.6735
Epoch 7/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 141ms/step - accuracy: 0.7433 - loss: 0.5090 - val_accuracy: 0.6460 - val_loss: 0.7406
Epoch 8/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 156ms/step - accuracy: 0.8055 - loss: 0.4039 - val_accuracy: 0.6380 - val_loss: 0.8505
```

```
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 ━━━━━━━━━━━━━━━━━━━━ 3s 131ms/step - accuracy: 0.6025 - loss: 0.6756
Test accuracy: 0.628
```

Feature extraction enhanced through data augmentation

Creating and locking 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_12 (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
```

```
**Integrating a data augmentation pipeline and a classificatio
of the convolutional base.**
```

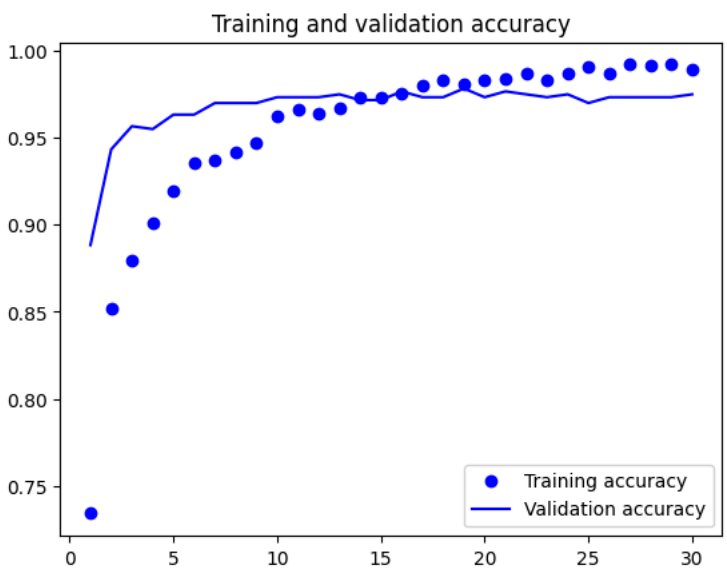
Integrating a data augmentation pipeline and a classification head on top of the convolutional base.

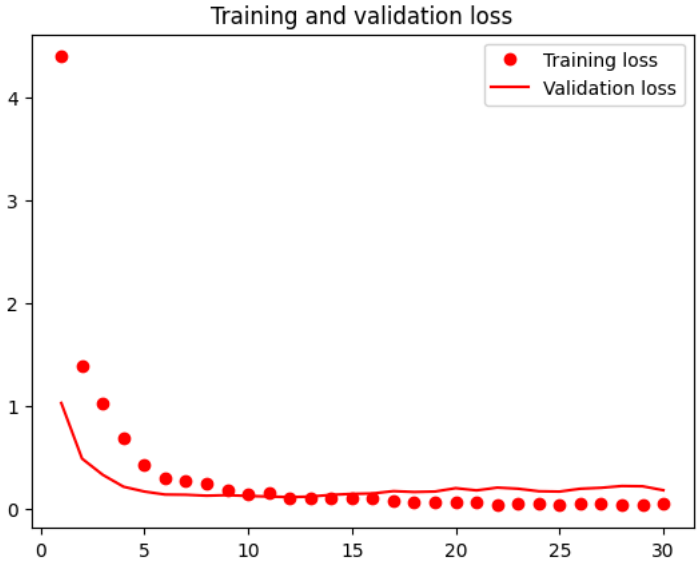
```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)


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=val_dataset,
    callbacks=callbacks)
```

```
Epoch 2/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 12s 185ms/step - accuracy: 0.8336 - loss: 1.6062 - val_accuracy: 0.9433 - val_loss: 0.4845
Epoch 3/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 181ms/step - accuracy: 0.8763 - loss: 1.0595 - val_accuracy: 0.9567 - val_loss: 0.3266
Epoch 4/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 207ms/step - accuracy: 0.9040 - loss: 0.7964 - val_accuracy: 0.9550 - val_loss: 0.2109
Epoch 5/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 245ms/step - accuracy: 0.9139 - loss: 0.4335 - val_accuracy: 0.9633 - val_loss: 0.1640
Epoch 6/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 189ms/step - accuracy: 0.9392 - loss: 0.2720 - val_accuracy: 0.9633 - val_loss: 0.1348
Epoch 7/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 162ms/step - accuracy: 0.9350 - loss: 0.2994 - val_accuracy: 0.9700 - val_loss: 0.1329
Epoch 8/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 200ms/step - accuracy: 0.9516 - loss: 0.1791 - val_accuracy: 0.9700 - val_loss: 0.1236
Epoch 9/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 235ms/step - accuracy: 0.9515 - loss: 0.1792 - val_accuracy: 0.9700 - val_loss: 0.1292
Epoch 10/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 17s 166ms/step - accuracy: 0.9706 - loss: 0.1092 - val_accuracy: 0.9733 - val_loss: 0.1206
Epoch 11/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 12s 189ms/step - accuracy: 0.9698 - loss: 0.1424 - val_accuracy: 0.9733 - val_loss: 0.1156
Epoch 12/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 219ms/step - accuracy: 0.9733 - loss: 0.0820 - val_accuracy: 0.9733 - val_loss: 0.1100
Epoch 13/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 161ms/step - accuracy: 0.9701 - loss: 0.0958 - val_accuracy: 0.9750 - val_loss: 0.1148
Epoch 14/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 163ms/step - accuracy: 0.9751 - loss: 0.0781 - val_accuracy: 0.9717 - val_loss: 0.1318
Epoch 15/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 166ms/step - accuracy: 0.9743 - loss: 0.0951 - val_accuracy: 0.9717 - val_loss: 0.1412
Epoch 16/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 178ms/step - accuracy: 0.9769 - loss: 0.0795 - val_accuracy: 0.9767 - val_loss: 0.1454
Epoch 17/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 162ms/step - accuracy: 0.9829 - loss: 0.0726 - val_accuracy: 0.9733 - val_loss: 0.1681
Epoch 18/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.9839 - loss: 0.0406 - val_accuracy: 0.9733 - val_loss: 0.1596
Epoch 19/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 158ms/step - accuracy: 0.9808 - loss: 0.0635 - val_accuracy: 0.9783 - val_loss: 0.1641
Epoch 20/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 180ms/step - accuracy: 0.9770 - loss: 0.0768 - val_accuracy: 0.9733 - val_loss: 0.1969
Epoch 21/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 220ms/step - accuracy: 0.9873 - loss: 0.0465 - val_accuracy: 0.9767 - val_loss: 0.1751
Epoch 22/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 7s 151ms/step - accuracy: 0.9888 - loss: 0.0334 - val_accuracy: 0.9750 - val_loss: 0.2023
Epoch 23/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 189ms/step - accuracy: 0.9806 - loss: 0.0489 - val_accuracy: 0.9733 - val_loss: 0.1916
Epoch 24/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 168ms/step - accuracy: 0.9888 - loss: 0.0368 - val_accuracy: 0.9750 - val_loss: 0.1669
Epoch 25/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 152ms/step - accuracy: 0.9917 - loss: 0.0246 - val_accuracy: 0.9700 - val_loss: 0.1637
Epoch 26/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 150ms/step - accuracy: 0.9857 - loss: 0.0522 - val_accuracy: 0.9733 - val_loss: 0.1915
Epoch 27/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 172ms/step - accuracy: 0.9891 - loss: 0.0525 - val_accuracy: 0.9733 - val_loss: 0.2004
Epoch 28/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 227ms/step - accuracy: 0.9937 - loss: 0.0367 - val_accuracy: 0.9733 - val_loss: 0.2179
Epoch 29/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 17s 158ms/step - accuracy: 0.9935 - loss: 0.0205 - val_accuracy: 0.9733 - val_loss: 0.2154
Epoch 30/30
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 174ms/step - accuracy: 0.9910 - loss: 0.0362 - val_accuracy: 0.9750 - val_loss: 0.1769
```

```
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 127ms/step - accuracy: 0.9619 - loss: 0.2732
Test accuracy: 0.963
```

Pre-Trained Model Utilizing 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=val_dataset,
    callbacks=callbacks)
```

```
Epoch 1/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 181ms/step - accuracy: 0.7279 - loss: 4.7850 - val_accuracy: 0.9467 - val_loss: 0.7728
Epoch 2/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 197ms/step - accuracy: 0.8726 - loss: 1.4896 - val_accuracy: 0.9550 - val_loss: 0.5806
Epoch 3/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 189ms/step - accuracy: 0.8888 - loss: 1.0350 - val_accuracy: 0.9633 - val_loss: 0.4393
Epoch 4/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 175ms/step - accuracy: 0.9307 - loss: 0.6375 - val_accuracy: 0.9650 - val_loss: 0.3539
Epoch 5/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 182ms/step - accuracy: 0.9414 - loss: 0.4189 - val_accuracy: 0.9683 - val_loss: 0.3067
Epoch 6/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 188ms/step - accuracy: 0.9528 - loss: 0.2771 - val_accuracy: 0.9650 - val_loss: 0.2771
Epoch 7/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 208ms/step - accuracy: 0.9482 - loss: 0.3108 - val_accuracy: 0.9683 - val_loss: 0.2327
Epoch 8/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 185ms/step - accuracy: 0.9580 - loss: 0.1995 - val_accuracy: 0.9717 - val_loss: 0.2093
Epoch 9/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 202ms/step - accuracy: 0.9510 - loss: 0.2260 - val_accuracy: 0.9683 - val_loss: 0.2021
Epoch 10/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 168ms/step - accuracy: 0.9584 - loss: 0.1424 - val_accuracy: 0.9683 - val_loss: 0.2206
```

```
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 140ms/step - accuracy: 0.9800 - loss: 0.1081
Test accuracy: 0.968
```

Pre-Trained Model Utilizing 1700 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
```

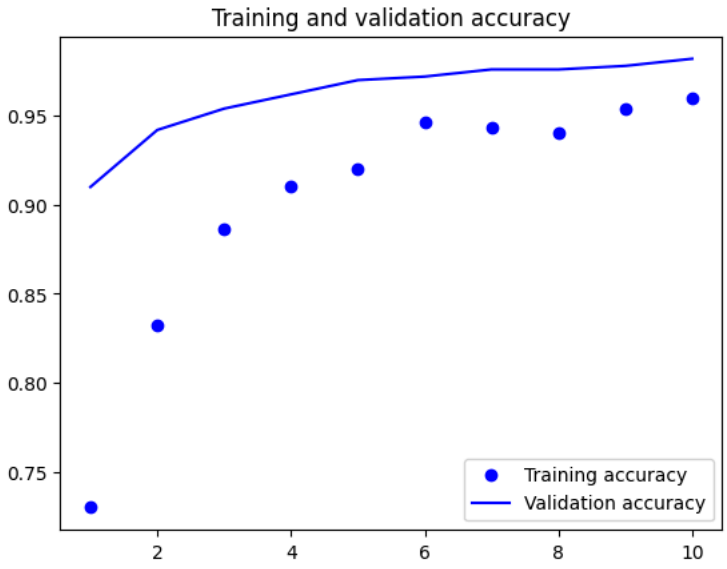
```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

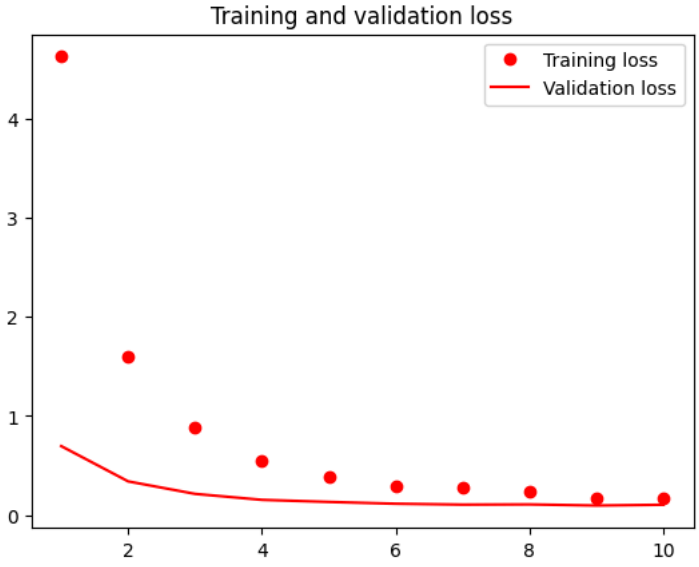
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
```

```
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_tuning3.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset_4,
    epochs=10,
    validation_data=val_dataset_4,
    callbacks=callbacks)
```

```
Epoch 1/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 12s 221ms/step - accuracy: 0.6586 - loss: 7.1900 - val_accuracy: 0.9100 - val_loss: 0.6967
Epoch 2/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 18s 162ms/step - accuracy: 0.8097 - loss: 2.0329 - val_accuracy: 0.9420 - val_loss: 0.3413
Epoch 3/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 181ms/step - accuracy: 0.8771 - loss: 0.9488 - val_accuracy: 0.9540 - val_loss: 0.2153
Epoch 4/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 164ms/step - accuracy: 0.9049 - loss: 0.5581 - val_accuracy: 0.9620 - val_loss: 0.1551
Epoch 5/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 11s 174ms/step - accuracy: 0.9158 - loss: 0.4496 - val_accuracy: 0.9700 - val_loss: 0.1341
Epoch 6/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 9s 186ms/step - accuracy: 0.9324 - loss: 0.3564 - val_accuracy: 0.9720 - val_loss: 0.1160
Epoch 7/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 180ms/step - accuracy: 0.9320 - loss: 0.2760 - val_accuracy: 0.9760 - val_loss: 0.1071
Epoch 8/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 165ms/step - accuracy: 0.9306 - loss: 0.2936 - val_accuracy: 0.9760 - val_loss: 0.1092
Epoch 9/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 8s 180ms/step - accuracy: 0.9440 - loss: 0.2471 - val_accuracy: 0.9780 - val_loss: 0.0980
Epoch 10/10
47/47 ━━━━━━━━━━━━━━━━━━━━ 10s 167ms/step - accuracy: 0.9580 - loss: 0.1596 - val_accuracy: 0.9820 - val_loss: 0.1056
```

```
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 105ms/step - accuracy: 0.9687 - loss: 0.3499
Test accuracy: 0.967
```

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.
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.
```