**Assignment 2: Convolution**

By: Arvind Chaurasia and Harika Beesetti

*As you saw in the chapter "Deep learning for computer vision," convents can be used successfully to classify images. Specifically, consider the Cats & Dogs example. There were two broad approaches to classifying Cats & Dogs usings convnets: Training a network from scratch, versus using a pretrained convnet. While small datasets can lead to overfitting, thus making training a network for prediction more difficult, you also saw several techniques to reduce overfitting, and these include data augmentation and regularization.*

*In this assignment, you will examine the relationship between training samples and the choice of training your model from scratch, versus using a pretrained convnet. Specifically, answer the following questions:*

*1. Consider the Cats & Dogs example. Start initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text). Use any technique to reduce overfitting and improve performance in developing a network that you train from scratch. What performance did you achieve?*

```
#Lest's use Kaggle API which will eliminate the maual authentication

import os
os.environ['KAGGLE_CONFIG_DIR'] = '/content'


#Using API of the dataset
!kaggle competitions download -c dogs-vs-cats
```

```
     Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /content/kaggle.json'
     Downloading dogs-vs-cats.zip to /content
     100% 810M/812M [00:36<00:00, 24.6MB/s]
     100% 812M/812M [00:36<00:00, 23.6MB/s]
```

```
#Now we will unzip all the file for data set.
!unzip /content/dogs-vs-cats.zip
```

```
     Archive:  /content/dogs-vs-cats.zip
       inflating: sampleSubmission.csv
       inflating: test1.zip
       inflating: train.zip
```

```
!unzip /content/test1.zip
!unzip /content/train.zip
```

```
     Streaming output truncated to the last 5000 lines.
       inflating: train/dog.5499.jpg
```

```
inflating: train/dog.55.jpg
inflating: train/dog.550.jpg
inflating: train/dog.5500.jpg
inflating: train/dog.5501.jpg
inflating: train/dog.5502.jpg
inflating: train/dog.5503.jpg
inflating: train/dog.5504.jpg
inflating: train/dog.5505.jpg
inflating: train/dog.5506.jpg
inflating: train/dog.5507.jpg
inflating: train/dog.5508.jpg
inflating: train/dog.5509.jpg
inflating: train/dog.551.jpg
inflating: train/dog.5510.jpg
inflating: train/dog.5511.jpg
inflating: train/dog.5512.jpg
inflating: train/dog.5513.jpg
inflating: train/dog.5514.jpg
inflating: train/dog.5515.jpg
inflating: train/dog.5516.jpg
inflating: train/dog.5517.jpg
inflating: train/dog.5518.jpg
inflating: train/dog.5519.jpg
inflating: train/dog.552.jpg
inflating: train/dog.5520.jpg
inflating: train/dog.5521.jpg
inflating: train/dog.5522.jpg
inflating: train/dog.5523.jpg
inflating: train/dog.5524.jpg
inflating: train/dog.5525.jpg
inflating: train/dog.5526.jpg
inflating: train/dog.5527.jpg
inflating: train/dog.5528.jpg
inflating: train/dog.5529.jpg
inflating: train/dog.553.jpg
inflating: train/dog.5530.jpg
inflating: train/dog.5531.jpg
inflating: train/dog.5532.jpg
inflating: train/dog.5533.jpg
inflating: train/dog.5534.jpg
inflating: train/dog.5535.jpg
inflating: train/dog.5536.jpg
inflating: train/dog.5537.jpg
inflating: train/dog.5538.jpg
inflating: train/dog.5539.jpg
inflating: train/dog.554.jpg
inflating: train/dog.5540.jpg
inflating: train/dog.5541.jpg
inflating: train/dog.5542.jpg
inflating: train/dog.5543.jpg
inflating: train/dog.5544.jpg
inflating: train/dog.5545.jpg
inflating: train/dog.5546.jpg
inflating: train/dog.5547.jpg
```

```
       inflating: train/dog.5548.jpg
```

Now we will be copying images to training, validation, and test directories.

```
#Importing Modules
import os, shutil, pathlib

#Setting Up Paths
original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

#Defining a Function to Create Subsets
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

#Calling the Function to Create Subsets
make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2000)
```

Data preprocessing

We will be using image_dataset_from_directory to read images

```
#Importing the Utility
from tensorflow.keras.utils import image_dataset_from_directory

#Creating Image Datasets
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)

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

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

     Found 2000 files belonging to 2 classes.
     Found 1000 files belonging to 2 classes.
     Found 1000 files belonging to 2 classes.


#Importing Libraries
import numpy as np
import tensorflow as tf

#Generating Random Numbers
random_numbers = np.random.normal(size=(1000, 16))

#Creating a TensorFlow Dataset
dataset = tf.data.Dataset.from_tensor_slices(random_numbers)


#Iteration and Indexing

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

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

```
#Creating a Batched Dataset

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)


#Reshaping Elements with map

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)
```

Further displaying the shapes of the data and labels yielded by the Dataset

```
for data_batch, labels_batch in train_dataset:
    print("data batch shape is", data_batch.shape)
    print("labels batch shape is", labels_batch.shape)
    break

     data batch shape is (32, 180, 180, 3)
     labels batch shape is (32,)
```

Building our model

A small convnet for dogs vs. cats classification model using Convolutional Neural Network (CNN).

```
#Importing Libraries
from tensorflow import keras
from tensorflow.keras import layers

#Input Layer
inputs = keras.Input(shape=(180, 180, 3))

#Rescaling
h = layers.Rescaling(1./255)(inputs)

#Convolutional Layers
h = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(h)
h = layers.MaxPooling2D(pool_size=2)(h)
h = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(h)
h = layers.MaxPooling2D(pool_size=2)(h)
h = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(h)
h = layers.MaxPooling2D(pool_size=2)(h)
h = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(h)
h = layers.MaxPooling2D(pool_size=2)(h)
h = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(h)

#Flattening
h = layers.Flatten()(h)

#Output Layer
outputs = layers.Dense(1, activation="sigmoid")(h)

#Model Creation
model = keras.Model(inputs=inputs, outputs=outputs)
```

Let's print a summary of the model's architecture.

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

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

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2  (None, 89, 89, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496
```

```
 max_pooling2d_1 (MaxPoolin  (None, 43, 43, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 20, 20, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 9, 9, 256)         0
 g2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 256)         590080

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 1)                 12545

=================================================================
Total params: 991041 (3.78 MB)
Trainable params: 991041 (3.78 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Configuring the model for training

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop", #Please note that we can use "adam" as well
              metrics=["accuracy"])
```

Fitting the model

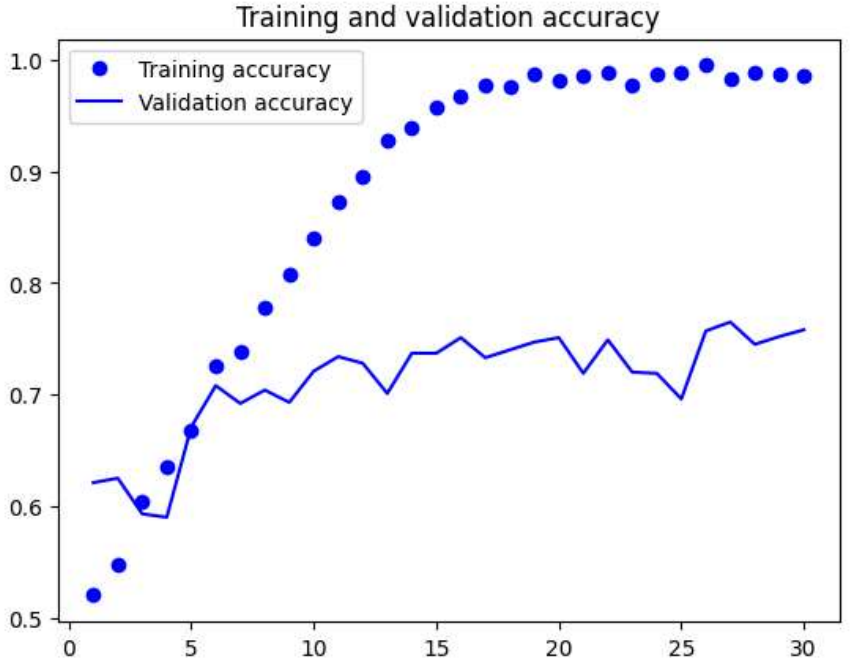
```
#Model Checkpoint Callback
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]

#Model Fitting
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)

#We can also consider using additional callbacks like EarlyStopping to stop training early if the validation loss doesn't improve for a certain number of epoc
```

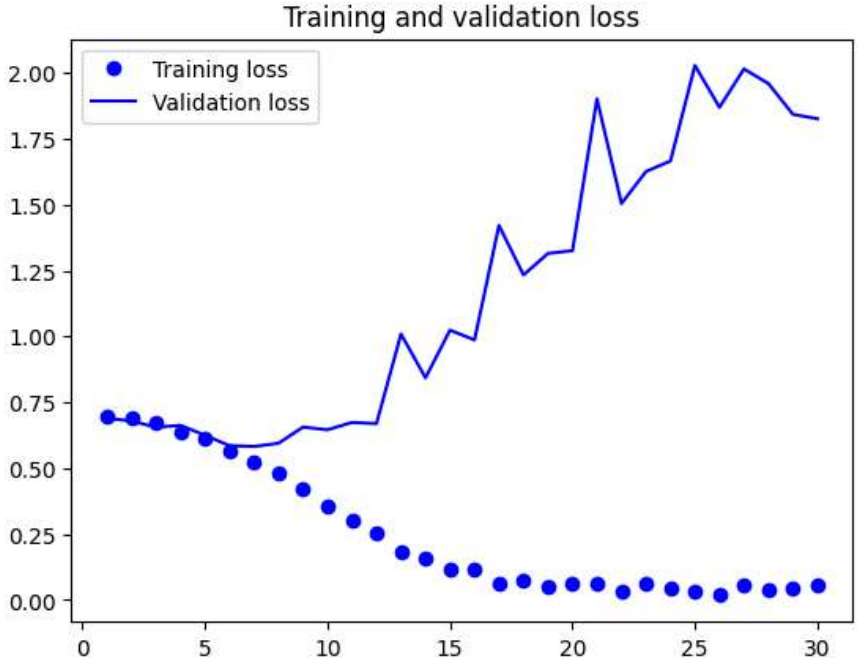
```
Epoch 1/30
63/63 [==============================] - 14s 121ms/step - loss: 0.6943 - accuracy: 0.5200 - val_loss: 0.6901 - val_accuracy: 0.6210
Epoch 2/30
63/63 [==============================] - 4s 57ms/step - loss: 0.6893 - accuracy: 0.5475 - val_loss: 0.6789 - val_accuracy: 0.6250
Epoch 3/30
63/63 [==============================] - 4s 60ms/step - loss: 0.6706 - accuracy: 0.6045 - val_loss: 0.6557 - val_accuracy: 0.5930
Epoch 4/30
63/63 [==============================] - 6s 94ms/step - loss: 0.6386 - accuracy: 0.6350 - val_loss: 0.6626 - val_accuracy: 0.5900
Epoch 5/30
63/63 [==============================] - 6s 84ms/step - loss: 0.6137 - accuracy: 0.6670 - val_loss: 0.6249 - val_accuracy: 0.6710
Epoch 6/30
63/63 [==============================] - 4s 58ms/step - loss: 0.5624 - accuracy: 0.7250 - val_loss: 0.5856 - val_accuracy: 0.7080
Epoch 7/30
63/63 [==============================] - 4s 64ms/step - loss: 0.5241 - accuracy: 0.7380 - val_loss: 0.5826 - val_accuracy: 0.6920
Epoch 8/30
63/63 [==============================] - 7s 98ms/step - loss: 0.4803 - accuracy: 0.7780 - val_loss: 0.5948 - val_accuracy: 0.7040
Epoch 9/30
63/63 [==============================] - 4s 58ms/step - loss: 0.4190 - accuracy: 0.8070 - val_loss: 0.6562 - val_accuracy: 0.6930
Epoch 10/30
63/63 [==============================] - 4s 56ms/step - loss: 0.3539 - accuracy: 0.8405 - val_loss: 0.6461 - val_accuracy: 0.7210
Epoch 11/30
63/63 [==============================] - 6s 88ms/step - loss: 0.3015 - accuracy: 0.8730 - val_loss: 0.6737 - val_accuracy: 0.7340
Epoch 12/30
63/63 [==============================] - 4s 56ms/step - loss: 0.2527 - accuracy: 0.8950 - val_loss: 0.6695 - val_accuracy: 0.7280
Epoch 13/30
63/63 [==============================] - 4s 57ms/step - loss: 0.1851 - accuracy: 0.9275 - val_loss: 1.0091 - val_accuracy: 0.7010
Epoch 14/30
63/63 [==============================] - 6s 91ms/step - loss: 0.1584 - accuracy: 0.9390 - val_loss: 0.8430 - val_accuracy: 0.7370
Epoch 15/30
63/63 [==============================] - 4s 56ms/step - loss: 0.1140 - accuracy: 0.9570 - val_loss: 1.0239 - val_accuracy: 0.7370
Epoch 16/30
63/63 [==============================] - 4s 57ms/step - loss: 0.1153 - accuracy: 0.9675 - val_loss: 0.9866 - val_accuracy: 0.7510
Epoch 17/30
63/63 [==============================] - 7s 103ms/step - loss: 0.0599 - accuracy: 0.9775 - val_loss: 1.4211 - val_accuracy: 0.7330
Epoch 18/30
```

```
63/63 [==============================] - 4s 56ms/step - loss: 0.0756 - accuracy: 0.9755 - val_loss: 1.2331 - val_accuracy: 0.7400
Epoch 19/30
63/63 [==============================] - 4s 57ms/step - loss: 0.0512 - accuracy: 0.9865 - val_loss: 1.3149 - val_accuracy: 0.7470
Epoch 20/30
63/63 [==============================] - 7s 108ms/step - loss: 0.0631 - accuracy: 0.9820 - val_loss: 1.3253 - val_accuracy: 0.7510
Epoch 21/30
63/63 [==============================] - 4s 57ms/step - loss: 0.0625 - accuracy: 0.9860 - val_loss: 1.9012 - val_accuracy: 0.7190
Epoch 22/30
63/63 [==============================] - 5s 80ms/step - loss: 0.0347 - accuracy: 0.9885 - val_loss: 1.5042 - val_accuracy: 0.7490
Epoch 23/30
63/63 [==============================] - 4s 58ms/step - loss: 0.0616 - accuracy: 0.9770 - val_loss: 1.6255 - val_accuracy: 0.7200
Epoch 24/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0427 - accuracy: 0.9875 - val_loss: 1.6656 - val_accuracy: 0.7190
Epoch 25/30
63/63 [==============================] - 7s 107ms/step - loss: 0.0351 - accuracy: 0.9890 - val_loss: 2.0278 - val_accuracy: 0.6960
Epoch 26/30
63/63 [==============================] - 5s 75ms/step - loss: 0.0181 - accuracy: 0.9950 - val_loss: 1.8689 - val_accuracy: 0.7570
Epoch 27/30
63/63 [==============================] - 4s 58ms/step - loss: 0.0583 - accuracy: 0.9830 - val_loss: 2.0146 - val_accuracy: 0.7650
Epoch 28/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0368 - accuracy: 0.9880 - val_loss: 1.9590 - val_accuracy: 0.7450
Epoch 29/30
63/63 [==============================] - 7s 97ms/step - loss: 0.0425 - accuracy: 0.9865 - val_loss: 1.8424 - val_accuracy: 0.7530
```

Let's display the curves of loss and accuracy during training

```
#Importing Libraries
import matplotlib.pyplot as plt

#Extracting Training Data
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

#Defining Epochs
epochs = range(1, len(accuracy) + 1)

#Plotting Accuracy
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()

#Plotting Loss
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()
```




Further, evaluating our model on the test set.

```
#Loading the Model
test_model = keras.models.load_model("convnet_from_scratch.keras")

#Evaluating the Model
test_loss, test_acc = test_model.evaluate(test_dataset)

#Printing Test Accuracy
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 48ms/step - loss: 0.5939 - accuracy: 0.6810
Test accuracy: 0.681
```

```
#Save model
model.save('Q1_cats_vs_dogs.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
  saving_api.save_model(
```

*2. Increase your training sample size. You may pick any amount. Keep the validation and test samples the same as above. Optimize your network (again training from scratch). What performance did you achieve?*

In this question we will increase the training sample size without making any changes to the test and validation sample same.

```
#Importing Libraries
import os, shutil, pathlib

#Defining Paths
original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_2")

#Creating a Subset Function
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

#Creating Training, Validation, and Test Sets
make_subset("train", start_index=0, end_index=2000)
make_subset("validation", start_index=2000, end_index=2500)
make_subset("test", start_index=2500, end_index=3000)
```

In this part of the question we will apply be using data augmentation as well in order to combat the overfitting.

Define a data augmentation stage to add to an image model

```
#Data Augmentation with keras.Sequential

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

Now we will be displaying some randomly augmented training images from our dataset.

```
#Setting Up the Plot
plt.figure(figsize=(8, 8)) #Please note that this size can be adjusted

#Looping Through Training Images and Applying Augmentation & Plotting
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")
```

Now, we have define a new convnet that includes image augmentation and dropout

```
#Model Definition
inputs = keras.Input(shape=(180, 180, 3))

#Data Augmentation
i = data_augmentation(inputs)

#Preprocessing
i = layers.Rescaling(1./255)(i)

#Convolutional Layers
i = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(i)
i = layers.MaxPooling2D(pool_size=2)(i)
i = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(i)
i = layers.MaxPooling2D(pool_size=2)(i)
i = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(i)
i = layers.MaxPooling2D(pool_size=2)(i)
i = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(i)
i = layers.MaxPooling2D(pool_size=2)(i)
i = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(i)

#Flattening and Dropout
i = layers.Flatten()(i)
i = layers.Dropout(0.5)(i)

#Output Layer
outputs = layers.Dense(1, activation="sigmoid")(i)

#Model Creation and Summary
model = keras.Model(inputs=inputs, outputs=outputs)
model.summary()

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

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

 sequential (Sequential)     (None, 180, 180, 3)       0

 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_10 (Conv2D)          (None, 178, 178, 32)      896

 max_pooling2d_8 (MaxPoolin  (None, 89, 89, 32)        0
 g2D)
```

```
 conv2d_11 (Conv2D)          (None, 87, 87, 64)        18496

 max_pooling2d_9 (MaxPoolin  (None, 43, 43, 64)        0
 g2D)

 conv2d_12 (Conv2D)          (None, 41, 41, 128)       73856

 max_pooling2d_10 (MaxPooli  (None, 20, 20, 128)       0
 ng2D)

 conv2d_13 (Conv2D)          (None, 18, 18, 256)       295168

 max_pooling2d_11 (MaxPooli  (None, 9, 9, 256)         0
 ng2D)

 conv2d_14 (Conv2D)          (None, 7, 7, 256)         590080

 flatten_2 (Flatten)         (None, 12544)             0

 dropout_1 (Dropout)         (None, 12544)             0

 dense_2 (Dense)             (None, 1)                 12545

=================================================================
Total params: 991041 (3.78 MB)
Trainable params: 991041 (3.78 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training the regularized convnet

```
#Model Checkpoint Callback
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]

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

```
 Epoch 1/50
 63/63 [==============================] - 9s 70ms/step - loss: 0.7518 - accuracy: 0.4940 - val_loss: 0.6923 - val_accuracy: 0.5000
 Epoch 2/50
```

```
63/63 [==============================] - 4s 63ms/step - loss: 0.6941 - accuracy: 0.5150 - val_loss: 0.6917 - val_accuracy: 0.5000
Epoch 3/50
63/63 [==============================] - 5s 69ms/step - loss: 0.6918 - accuracy: 0.5375 - val_loss: 0.6959 - val_accuracy: 0.5060
Epoch 4/50
63/63 [==============================] - 7s 100ms/step - loss: 0.6663 - accuracy: 0.6075 - val_loss: 0.6562 - val_accuracy: 0.6020
Epoch 5/50
63/63 [==============================] - 4s 58ms/step - loss: 0.6442 - accuracy: 0.6250 - val_loss: 0.6238 - val_accuracy: 0.6360
Epoch 6/50
63/63 [==============================] - 4s 58ms/step - loss: 0.6355 - accuracy: 0.6480 - val_loss: 0.6683 - val_accuracy: 0.5810
Epoch 7/50
63/63 [==============================] - 7s 103ms/step - loss: 0.6036 - accuracy: 0.6780 - val_loss: 0.5799 - val_accuracy: 0.6800
Epoch 8/50
63/63 [==============================] - 5s 70ms/step - loss: 0.6127 - accuracy: 0.6745 - val_loss: 0.6071 - val_accuracy: 0.6510
Epoch 9/50
63/63 [==============================] - 4s 57ms/step - loss: 0.5910 - accuracy: 0.6765 - val_loss: 0.5868 - val_accuracy: 0.6810
Epoch 10/50
63/63 [==============================] - 6s 91ms/step - loss: 0.5952 - accuracy: 0.6930 - val_loss: 1.1321 - val_accuracy: 0.5510
Epoch 11/50
63/63 [==============================] - 6s 92ms/step - loss: 0.5797 - accuracy: 0.7060 - val_loss: 0.5724 - val_accuracy: 0.6810
Epoch 12/50
63/63 [==============================] - 4s 61ms/step - loss: 0.5760 - accuracy: 0.6970 - val_loss: 0.5366 - val_accuracy: 0.7190
Epoch 13/50
63/63 [==============================] - 7s 102ms/step - loss: 0.5529 - accuracy: 0.7210 - val_loss: 0.5786 - val_accuracy: 0.6990
Epoch 14/50
63/63 [==============================] - 4s 64ms/step - loss: 0.5530 - accuracy: 0.7445 - val_loss: 0.5484 - val_accuracy: 0.7090
Epoch 15/50
63/63 [==============================] - 4s 61ms/step - loss: 0.5275 - accuracy: 0.7365 - val_loss: 0.7643 - val_accuracy: 0.6620
Epoch 16/50
63/63 [==============================] - 7s 99ms/step - loss: 0.5357 - accuracy: 0.7350 - val_loss: 0.6164 - val_accuracy: 0.6840
Epoch 17/50
63/63 [==============================] - 4s 65ms/step - loss: 0.5241 - accuracy: 0.7390 - val_loss: 0.5432 - val_accuracy: 0.7280
Epoch 18/50
63/63 [==============================] - 4s 58ms/step - loss: 0.5040 - accuracy: 0.7560 - val_loss: 0.5595 - val_accuracy: 0.7060
Epoch 19/50
63/63 [==============================] - 6s 93ms/step - loss: 0.4938 - accuracy: 0.7575 - val_loss: 0.4957 - val_accuracy: 0.7570
Epoch 20/50
63/63 [==============================] - 6s 90ms/step - loss: 0.4911 - accuracy: 0.7540 - val_loss: 0.6718 - val_accuracy: 0.6890
Epoch 21/50
63/63 [==============================] - 4s 59ms/step - loss: 0.4753 - accuracy: 0.7815 - val_loss: 0.6140 - val_accuracy: 0.7020
Epoch 22/50
63/63 [==============================] - 4s 57ms/step - loss: 0.4679 - accuracy: 0.7820 - val_loss: 0.5178 - val_accuracy: 0.7560
Epoch 23/50
63/63 [==============================] - 7s 98ms/step - loss: 0.4620 - accuracy: 0.7795 - val_loss: 0.5728 - val_accuracy: 0.7420
Epoch 24/50
63/63 [==============================] - 5s 70ms/step - loss: 0.4673 - accuracy: 0.7830 - val_loss: 0.5033 - val_accuracy: 0.7610
Epoch 25/50
63/63 [==============================] - 4s 61ms/step - loss: 0.4587 - accuracy: 0.7835 - val_loss: 0.4860 - val_accuracy: 0.7690
Epoch 26/50
63/63 [==============================] - 4s 68ms/step - loss: 0.4474 - accuracy: 0.8010 - val_loss: 0.4387 - val_accuracy: 0.7960
Epoch 27/50
63/63 [==============================] - 7s 112ms/step - loss: 0.4299 - accuracy: 0.8070 - val_loss: 0.4387 - val_accuracy: 0.8030
Epoch 28/50
63/63 [==============================] - 4s 60ms/step - loss: 0.4253 - accuracy: 0.8145 - val_loss: 0.5188 - val_accuracy: 0.7670
Epoch 29/50
```

Evaluating the model on the test set

```
#Loading the Model
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")

#Evaluating the Model
test_loss, test_acc = test_model.evaluate(test_dataset)

#Printing Test Accuracy
print(f"Test accuracy: {test_acc:.3f}")
```

```
     32/32 [==============================] - 2s 37ms/step - loss: 0.4704 - accuracy: 0.8090
     Test accuracy: 0.809
```

*3. Now change your training sample so that you achieve better performance than those from Steps 1 and 2. This sample size may be larger, or smaller than those in the previous steps. The objective is to find the ideal training sample size to get best prediction results.*

In the question number 2, we increased the training sample size from 1000 to 2000. Now we will increase the training sample size from 2000 to 3000.

```
#Importing Libraries
import os, shutil, pathlib

#Defining Paths
original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_3")

#Creating a Subset Function
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

#Creating Training, Validation, and Test Sets
make_subset("train", start_index=0, end_index=3000)
make_subset("validation", start_index=3000, end_index=3500)
make_subset("test", start_index=3500, end_index=4000)
```

A new convnet with increased training samples,image augmentation and dropout.

```
#Model Inputs
inputs = keras.Input(shape=(180, 180, 3))

#Data Augmentation
j = data_augmentation(inputs)

#Preprocessing
j = layers.Rescaling(1./255)(j)

#Convolutional Layers
j = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(j)
j = layers.MaxPooling2D(pool_size=2)(j)
j = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(j)
j = layers.MaxPooling2D(pool_size=2)(j)
j = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(j)
j = layers.MaxPooling2D(pool_size=2)(j)
j = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(j)
j = layers.MaxPooling2D(pool_size=2)(j)
j = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(j)

#Flattening and Dropout
j = layers.Flatten()(j)
j = layers.Dropout(0.5)(j)

#Output Layer
outputs = layers.Dense(1, activation="sigmoid")(j)

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

Training the regularized convnet

```
#Model Checkpoint Callback
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]

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

```
Epoch 1/50
63/63 [==============================] - 5s 69ms/step - loss: 0.6732 - accuracy: 0.5925 - val_loss: 0.6479 - val_accuracy: 0.6220
Epoch 2/50
63/63 [==============================] - 7s 105ms/step - loss: 0.6396 - accuracy: 0.6475 - val_loss: 0.6433 - val_accuracy: 0.6150
Epoch 3/50
63/63 [==============================] - 4s 60ms/step - loss: 0.6317 - accuracy: 0.6570 - val_loss: 0.7416 - val_accuracy: 0.6090
Epoch 4/50
63/63 [==============================] - 4s 61ms/step - loss: 0.6047 - accuracy: 0.6655 - val_loss: 0.5848 - val_accuracy: 0.6900
Epoch 5/50
63/63 [==============================] - 6s 88ms/step - loss: 0.5957 - accuracy: 0.6970 - val_loss: 0.5716 - val_accuracy: 0.6950
Epoch 6/50
63/63 [==============================] - 6s 92ms/step - loss: 0.5925 - accuracy: 0.6900 - val_loss: 0.5872 - val_accuracy: 0.7000
Epoch 7/50
63/63 [==============================] - 4s 59ms/step - loss: 0.5921 - accuracy: 0.7025 - val_loss: 0.5580 - val_accuracy: 0.7090
Epoch 8/50
63/63 [==============================] - 4s 59ms/step - loss: 0.5794 - accuracy: 0.6865 - val_loss: 0.6040 - val_accuracy: 0.6640
Epoch 9/50
63/63 [==============================] - 6s 86ms/step - loss: 0.5572 - accuracy: 0.7065 - val_loss: 0.6087 - val_accuracy: 0.6910
Epoch 10/50
63/63 [==============================] - 6s 95ms/step - loss: 0.5558 - accuracy: 0.7265 - val_loss: 0.5472 - val_accuracy: 0.7220
Epoch 11/50
63/63 [==============================] - 5s 69ms/step - loss: 0.5393 - accuracy: 0.7310 - val_loss: 0.5939 - val_accuracy: 0.6890
Epoch 12/50
63/63 [==============================] - 4s 57ms/step - loss: 0.5436 - accuracy: 0.7230 - val_loss: 0.6177 - val_accuracy: 0.7000
Epoch 13/50
63/63 [==============================] - 7s 112ms/step - loss: 0.5348 - accuracy: 0.7425 - val_loss: 0.5438 - val_accuracy: 0.7200
Epoch 14/50
63/63 [==============================] - 4s 62ms/step - loss: 0.5105 - accuracy: 0.7425 - val_loss: 0.4959 - val_accuracy: 0.7520
Epoch 15/50
63/63 [==============================] - 5s 68ms/step - loss: 0.5126 - accuracy: 0.7545 - val_loss: 0.5597 - val_accuracy: 0.7260
Epoch 16/50
63/63 [==============================] - 8s 116ms/step - loss: 0.4791 - accuracy: 0.7695 - val_loss: 0.4789 - val_accuracy: 0.7920
Epoch 17/50
63/63 [==============================] - 4s 58ms/step - loss: 0.4802 - accuracy: 0.7665 - val_loss: 0.5271 - val_accuracy: 0.7670
Epoch 18/50
63/63 [==============================] - 4s 60ms/step - loss: 0.4713 - accuracy: 0.7840 - val_loss: 0.6261 - val_accuracy: 0.7270
Epoch 19/50
```

```
63/63 [==============================] - 6s 95ms/step - loss: 0.4730 - accuracy: 0.7730 - val_loss: 0.4655 - val_accuracy: 0.7760
Epoch 20/50
63/63 [==============================] - 4s 60ms/step - loss: 0.4558 - accuracy: 0.7985 - val_loss: 0.5406 - val_accuracy: 0.7520
Epoch 21/50
63/63 [==============================] - 4s 62ms/step - loss: 0.4611 - accuracy: 0.7850 - val_loss: 0.5594 - val_accuracy: 0.7480
Epoch 22/50
63/63 [==============================] - 7s 101ms/step - loss: 0.4478 - accuracy: 0.7880 - val_loss: 0.4416 - val_accuracy: 0.8060
Epoch 23/50
63/63 [==============================] - 4s 58ms/step - loss: 0.4407 - accuracy: 0.7975 - val_loss: 0.4790 - val_accuracy: 0.7930
Epoch 24/50
63/63 [==============================] - 4s 59ms/step - loss: 0.4252 - accuracy: 0.7985 - val_loss: 0.5017 - val_accuracy: 0.7950
Epoch 25/50
63/63 [==============================] - 6s 92ms/step - loss: 0.4160 - accuracy: 0.8065 - val_loss: 0.5173 - val_accuracy: 0.7590
Epoch 26/50
63/63 [==============================] - 6s 89ms/step - loss: 0.4268 - accuracy: 0.7955 - val_loss: 0.5307 - val_accuracy: 0.7720
Epoch 27/50
63/63 [==============================] - 4s 59ms/step - loss: 0.3859 - accuracy: 0.8265 - val_loss: 0.5453 - val_accuracy: 0.7690
Epoch 28/50
63/63 [==============================] - 7s 111ms/step - loss: 0.3988 - accuracy: 0.8165 - val_loss: 0.4161 - val_accuracy: 0.8120
Epoch 29/50
```

Evaluating the model on the test set

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

```
32/32 [==============================] - 1s 28ms/step - loss: 0.4706 - accuracy: 0.7950
Test accuracy: 0.795
```

*4. Repeat Steps 1-3, but now using a pretrained network. The sample sizes you use in Steps 2 and 3 for the pretrained network may be the same or different from those using the network where you trained from scratch. Again, use any and all optimization techniques to get best performance.*

In this part of the question we will be leveraging a pretrained model

Feature extraction with a pretrained model

Instantiating the VGG16 convolutional base

```
#Importing VGG16 and Creating the Base Model
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 4s 0us/step
```

```
#To print a summary of the pre-trained VGG16 model
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 180, 180, 3)]     0

 block1_conv1 (Conv2D)       (None, 180, 180, 64)      1792

 block1_conv2 (Conv2D)       (None, 180, 180, 64)      36928

 block1_pool (MaxPooling2D)  (None, 90, 90, 64)        0

 block2_conv1 (Conv2D)       (None, 90, 90, 128)       73856

 block2_conv2 (Conv2D)       (None, 90, 90, 128)       147584

 block2_pool (MaxPooling2D)  (None, 45, 45, 128)       0

 block3_conv1 (Conv2D)       (None, 45, 45, 256)       295168

 block3_conv2 (Conv2D)       (None, 45, 45, 256)       590080

 block3_conv3 (Conv2D)       (None, 45, 45, 256)       590080

 block3_pool (MaxPooling2D)  (None, 22, 22, 256)       0

 block4_conv1 (Conv2D)       (None, 22, 22, 512)       1180160

 block4_conv2 (Conv2D)       (None, 22, 22, 512)       2359808

 block4_conv3 (Conv2D)       (None, 22, 22, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 11, 11, 512)       0

 block5_conv1 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_conv2 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_conv3 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 5, 5, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Below we will see fast feature extraction without data augmentation.

Here we will be extracting the VGG16 features and corresponding labels.

```
import numpy as np

#Function Definition and Feature Extraction Loop
def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)

  #Concatenation and Return
    return np.concatenate(all_features), np.concatenate(all_labels)

#Function Calls
train_features, train_labels =  get_features_and_labels(train_dataset)
val_features, val_labels =  get_features_and_labels(validation_dataset)
test_features, test_labels =  get_features_and_labels(test_dataset)
```

```
 1/1 [==============================] - 5s 5s/step
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 27ms/step
 1/1 [==============================] - 0s 28ms/step
 1/1 [==============================] - 0s 26ms/step
 1/1 [==============================] - 0s 28ms/step
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 24ms/step
 1/1 [==============================] - 0s 26ms/step
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 28ms/step
 1/1 [==============================] - 0s 28ms/step
 1/1 [==============================] - 0s 25ms/step
 1/1 [==============================] - 0s 25ms/step
 1/1 [==============================] - 0s 36ms/step
 1/1 [==============================] - 0s 22ms/step
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 22ms/step
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 29ms/step
 1/1 [==============================] - 0s 44ms/step
 1/1 [==============================] - 0s 30ms/step
 1/1 [==============================] - 0s 33ms/step
 1/1 [==============================] - 0s 46ms/step
```

```
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 31ms/step
```

```
train_features.shape
```

```
(2000, 5, 5, 512)
```

Defining and training the densely connected classifier

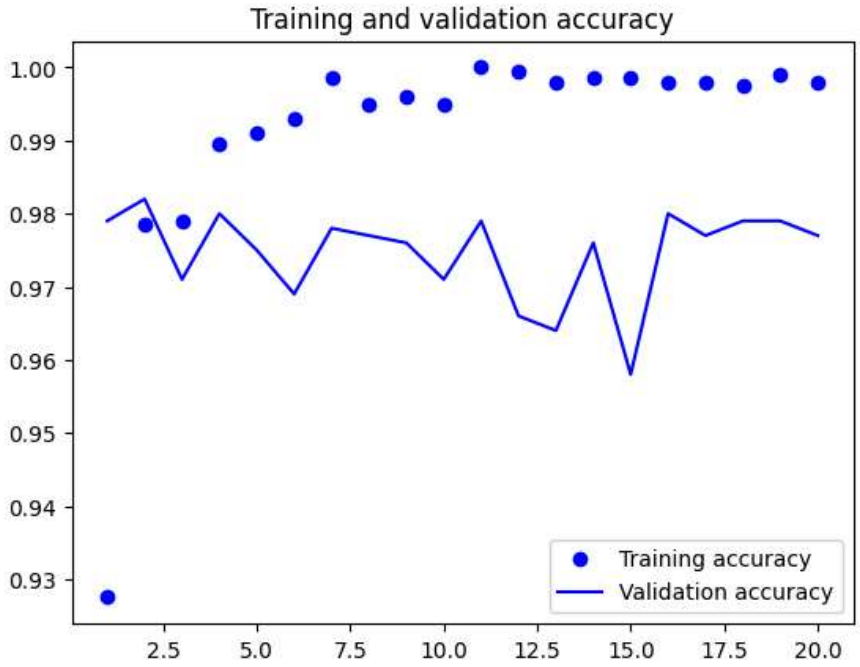
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

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

history = model.fit(
    train_features, train_labels,
    epochs=20,
    validation_data=(val_features, val_labels),
    callbacks=callbacks)
```

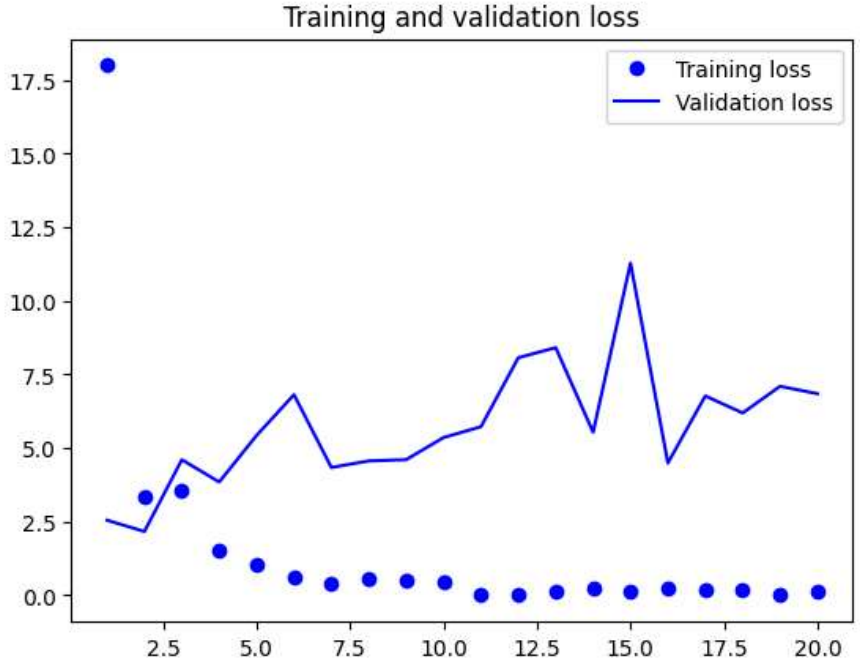
```
Epoch 1/20
63/63 [==============================] - 2s 15ms/step - loss: 17.9933 - accuracy: 0.9275 - val_loss: 2.5505 - val_accuracy: 0.9790
Epoch 2/20
63/63 [==============================] - 1s 10ms/step - loss: 3.3499 - accuracy: 0.9785 - val_loss: 2.1677 - val_accuracy: 0.9820
Epoch 3/20
63/63 [==============================] - 0s 7ms/step - loss: 3.5681 - accuracy: 0.9790 - val_loss: 4.6040 - val_accuracy: 0.9710
Epoch 4/20
63/63 [==============================] - 0s 8ms/step - loss: 1.5425 - accuracy: 0.9895 - val_loss: 3.8482 - val_accuracy: 0.9800
Epoch 5/20
63/63 [==============================] - 0s 6ms/step - loss: 1.0526 - accuracy: 0.9910 - val_loss: 5.4303 - val_accuracy: 0.9750
Epoch 6/20
63/63 [==============================] - 0s 5ms/step - loss: 0.6250 - accuracy: 0.9930 - val_loss: 6.8121 - val_accuracy: 0.9690
Epoch 7/20
63/63 [==============================] - 0s 6ms/step - loss: 0.4066 - accuracy: 0.9985 - val_loss: 4.3413 - val_accuracy: 0.9780
Epoch 8/20
63/63 [==============================] - 0s 6ms/step - loss: 0.5627 - accuracy: 0.9950 - val_loss: 4.5645 - val_accuracy: 0.9770
Epoch 9/20
63/63 [==============================] - 0s 5ms/step - loss: 0.5302 - accuracy: 0.9960 - val_loss: 4.6070 - val_accuracy: 0.9760
Epoch 10/20
63/63 [==============================] - 0s 6ms/step - loss: 0.4794 - accuracy: 0.9950 - val_loss: 5.3577 - val_accuracy: 0.9710
Epoch 11/20
63/63 [==============================] - 0s 5ms/step - loss: 5.9883e-05 - accuracy: 1.0000 - val_loss: 5.7253 - val_accuracy: 0.9790
Epoch 12/20
63/63 [==============================] - 0s 6ms/step - loss: 0.0301 - accuracy: 0.9995 - val_loss: 8.0692 - val_accuracy: 0.9660
Epoch 13/20
63/63 [==============================] - 0s 6ms/step - loss: 0.1077 - accuracy: 0.9980 - val_loss: 8.4101 - val_accuracy: 0.9640
Epoch 14/20
63/63 [==============================] - 0s 6ms/step - loss: 0.2427 - accuracy: 0.9985 - val_loss: 5.5384 - val_accuracy: 0.9760
Epoch 15/20
```

```
63/63 [==============================] - 0s 5ms/step - loss: 0.1257 - accuracy: 0.9985 - val_loss: 11.2804 - val_accuracy: 0.9580
Epoch 16/20
63/63 [==============================] - 0s 5ms/step - loss: 0.2647 - accuracy: 0.9980 - val_loss: 4.4940 - val_accuracy: 0.9800
Epoch 17/20
63/63 [==============================] - 0s 5ms/step - loss: 0.1657 - accuracy: 0.9980 - val_loss: 6.7704 - val_accuracy: 0.9770
Epoch 18/20
63/63 [==============================] - 0s 5ms/step - loss: 0.1919 - accuracy: 0.9975 - val_loss: 6.1922 - val_accuracy: 0.9790
Epoch 19/20
63/63 [==============================] - 0s 5ms/step - loss: 0.0370 - accuracy: 0.9990 - val_loss: 7.0976 - val_accuracy: 0.9790
Epoch 20/20
63/63 [==============================] - 0s 6ms/step - loss: 0.1298 - accuracy: 0.9980 - val_loss: 6.8465 - val_accuracy: 0.9770
```

Plotting the results

```
import matplotlib.pyplot as plt

acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(acc) + 1)

#Accuracy Plot
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

#Loss Plot
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```




Feature extraction together with data augmentation

Instantiating and freezing the VGG16 convolutional base

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

Printing the list of trainable weights before and after freezing

```
conv_base.trainable = True
print("This is the number of trainable weights "
      "before freezing the conv base:", len(conv_base.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 26
```

```
conv_base.trainable = False
print("This is the number of trainable weights "
      "after freezing the conv base:", len(conv_base.trainable_weights))
```

```
This is the number of trainable weights after freezing the conv base: 0
```

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

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

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

```
#Importing the Utility
from tensorflow.keras.utils import image_dataset_from_directory

#Creating Image Datasets
train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)

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

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

     Found 6000 files belonging to 2 classes.
     Found 1000 files belonging to 2 classes.
     Found 1000 files belonging to 2 classes.


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


from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor="val_loss", patience=2)
callbacks = [early_stopping]

history = model.fit(
    train_dataset,
    epochs=10,
    validation_data=validation_dataset,
    callbacks=callbacks)

     Epoch 1/10
     188/188 [==============================] - 23s 123ms/step - loss: 0.5913 - accuracy: 0.9730 - val_loss: 0.4472 - val_accuracy: 0.9850
     Epoch 2/10
     188/188 [==============================] - 23s 120ms/step - loss: 0.6783 - accuracy: 0.9712 - val_loss: 0.5408 - val_accuracy: 0.9810
     Epoch 3/10
     188/188 [==============================] - 23s 121ms/step - loss: 0.6275 - accuracy: 0.9688 - val_loss: 0.6430 - val_accuracy: 0.9790
```

Evaluating the model on the test set

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

```
32/32 [==============================] - 1s 29ms/step - loss: 0.4807 - accuracy: 0.7990
Test accuracy: 0.799
```

Fine Tunning

```
conv_base.summary()

#Freezing all layers until the fourth from the last

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

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808
```

```
 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 7079424 (27.01 MB)
Non-trainable params: 7635264 (29.13 MB)
_________________________________________________________________
```

```
# Compile the model
model.compile(
    loss="binary_crossentropy",
    optimizer=keras.optimizers.RMSprop(learning_rate=1e-4),
    metrics=["accuracy"]
)

# Define callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.h5",
        save_best_only=True,
        monitor="val_loss"
    )
]

# Train the model
history = model.fit(
    train_dataset,
    epochs=15,
    validation_data=validation_dataset,
    callbacks=callbacks
)

# Load the best model
model = keras.models.load_model("fine_tuning.h5")

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
Epoch 1/15
188/188 [==============================] - ETA: 0s - loss: 0.2749 - accuracy: 0.9205/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3
  saving_api.save_model(
188/188 [==============================] - 31s 145ms/step - loss: 0.2749 - accuracy: 0.9205 - val_loss: 0.1072 - val_accuracy: 0.9660
Epoch 2/15
188/188 [==============================] - 28s 146ms/step - loss: 0.1737 - accuracy: 0.9433 - val_loss: 0.1598 - val_accuracy: 0.9640
Epoch 3/15
188/188 [==============================] - 28s 145ms/step - loss: 0.1293 - accuracy: 0.9567 - val_loss: 0.1566 - val_accuracy: 0.9460
Epoch 4/15
188/188 [==============================] - 27s 143ms/step - loss: 0.1090 - accuracy: 0.9608 - val_loss: 0.0920 - val_accuracy: 0.9720
Epoch 5/15
188/188 [==============================] - 26s 139ms/step - loss: 0.1139 - accuracy: 0.9622 - val_loss: 0.0763 - val_accuracy: 0.9760
Epoch 6/15
188/188 [==============================] - 27s 140ms/step - loss: 0.0940 - accuracy: 0.9683 - val_loss: 0.0837 - val_accuracy: 0.9770
Epoch 7/15
188/188 [==============================] - 27s 140ms/step - loss: 0.0872 - accuracy: 0.9740 - val_loss: 0.1248 - val_accuracy: 0.9740
Epoch 8/15
188/188 [==============================] - 26s 136ms/step - loss: 0.0953 - accuracy: 0.9708 - val_loss: 0.0823 - val_accuracy: 0.9770
Epoch 9/15
```

```
188/188 [==============================] - 28s 147ms/step - loss: 0.0827 - accuracy: 0.9730 - val_loss: 0.1181 - val_accuracy: 0.9790
Epoch 10/15
188/188 [==============================] - 27s 139ms/step - loss: 0.0803 - accuracy: 0.9747 - val_loss: 0.1077 - val_accuracy: 0.9770
Epoch 11/15
188/188 [==============================] - 26s 139ms/step - loss: 0.0641 - accuracy: 0.9775 - val_loss: 0.0838 - val_accuracy: 0.9800
Epoch 12/15
188/188 [==============================] - 26s 138ms/step - loss: 0.0679 - accuracy: 0.9793 - val_loss: 0.1109 - val_accuracy: 0.9750
Epoch 13/15
188/188 [==============================] - 27s 142ms/step - loss: 0.0539 - accuracy: 0.9805 - val_loss: 0.0763 - val_accuracy: 0.9810
Epoch 14/15
188/188 [==============================] - 26s 138ms/step - loss: 0.0483 - accuracy: 0.9862 - val_loss: 0.0684 - val_accuracy: 0.9810
Epoch 15/15
188/188 [==============================] - 26s 138ms/step - loss: 0.0536 - accuracy: 0.9832 - val_loss: 0.0622 - val_accuracy: 0.9770
32/32 [==============================] - 4s 100ms/step - loss: 0.1527 - accuracy: 0.9820
Test accuracy: 0.982
```

```
#Importing Modules
import os, shutil, pathlib

#Setting Up Paths
original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

#Defining a Function to Create Subsets
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)


make_subset("train2", start_index=1000, end_index=2000)

train_dataset_2 = image_dataset_from_directory(
new_base_dir / "train2",
image_size=(180, 180),
batch_size=32)

make_subset("train3", start_index=1000, end_index=2000)

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

```
Found 2000 files belonging to 2 classes.
Found 2000 files belonging to 2 classes.
```

```
history = model.fit(
train_dataset_2,
epochs=15,
validation_data=validation_dataset,
callbacks=callbacks)
```

```
Epoch 1/15
63/63 [==============================] - 14s 197ms/step - loss: 0.0624 - accuracy: 0.9855 - val_loss: 0.1406 - val_accuracy: 0.9790
Epoch 2/15
63/63 [==============================] - 13s 202ms/step - loss: 0.0342 - accuracy: 0.9930 - val_loss: 0.1229 - val_accuracy: 0.9800
Epoch 3/15
63/63 [==============================] - 13s 205ms/step - loss: 0.0492 - accuracy: 0.9870 - val_loss: 0.0956 - val_accuracy: 0.9790
Epoch 4/15
63/63 [==============================] - 11s 177ms/step - loss: 0.0388 - accuracy: 0.9875 - val_loss: 0.2153 - val_accuracy: 0.9820
Epoch 5/15
63/63 [==============================] - 13s 198ms/step - loss: 0.0519 - accuracy: 0.9885 - val_loss: 0.2136 - val_accuracy: 0.9770
Epoch 6/15
63/63 [==============================] - 11s 173ms/step - loss: 0.0359 - accuracy: 0.9905 - val_loss: 0.1748 - val_accuracy: 0.9780
Epoch 7/15
63/63 [==============================] - 12s 195ms/step - loss: 0.0266 - accuracy: 0.9925 - val_loss: 0.2205 - val_accuracy: 0.9790
Epoch 8/15
63/63 [==============================] - 11s 174ms/step - loss: 0.0583 - accuracy: 0.9910 - val_loss: 0.1124 - val_accuracy: 0.9810
Epoch 9/15
63/63 [==============================] - 12s 178ms/step - loss: 0.0242 - accuracy: 0.9910 - val_loss: 0.5353 - val_accuracy: 0.9630
Epoch 10/15
63/63 [==============================] - 11s 166ms/step - loss: 0.0299 - accuracy: 0.9925 - val_loss: 0.3805 - val_accuracy: 0.9720
Epoch 11/15
63/63 [==============================] - 11s 176ms/step - loss: 0.0282 - accuracy: 0.9915 - val_loss: 0.2792 - val_accuracy: 0.9760
Epoch 12/15
63/63 [==============================] - 13s 199ms/step - loss: 0.0159 - accuracy: 0.9965 - val_loss: 0.1909 - val_accuracy: 0.9790
Epoch 13/15
63/63 [==============================] - 11s 173ms/step - loss: 0.0434 - accuracy: 0.9935 - val_loss: 0.1864 - val_accuracy: 0.9820
Epoch 14/15
63/63 [==============================] - 12s 180ms/step - loss: 0.0174 - accuracy: 0.9955 - val_loss: 0.3985 - val_accuracy: 0.9730
Epoch 15/15
63/63 [==============================] - 12s 183ms/step - loss: 0.0241 - accuracy: 0.9935 - val_loss: 0.2655 - val_accuracy: 0.9800
```

```
import matplotlib.pyplot as plt

acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(acc) + 1)
```