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

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"cchakradhar","key":"8d0f8292fffc941009a1e821cc65a300"}'}
```

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

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

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

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

```
Downloading dogs-vs-cats.zip to /content
 99% 804M/812M [00:05<00:00, 201MB/s]
100% 812M/812M [00:05<00:00, 156MB/s]
```

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

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

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?

As per the approach we need to train a network from scratch, from the above data i have loaded the dataset and need to import the required modules like operating system interface, high level file operations, object oriented filesystem paths etc,. are essential for working with direcrories and some other filesysytems.

```
import os, shutil, pathlib

old_dir = pathlib.Path("train")
new_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_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=old_dir/ fname,
                            dst=dir / fname)

make_subset("test", start_index=0, end_index=500)
make_subset("validation", start_index=500, end_index=1000)
make_subset("train", start_index=1000, end_index=2000)
```

Building the model

Here the input for this network is 3-D tensor which is images that needs to reshaped. For that we need to use the general model structure for the convnet with alternated Conv2D(with 'relu' activation) and maxpooling2D stages more due to its bigger images.

The following problem is a binary-classification either the output is determined as 'cat' or 'dog'. We need to end with dense layer and a 'sigmoid function'

First, we need to rescaling layer that are changes from [0,255] to [0,1]

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

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

Here we can see the dimensions of the feature maps changes in between the layers.

```
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)
_________________________________________________________________
```

From the above we can see there are 991041 trainable params needs to optimize.

Configuring the model

for the compilation step we need to use the 'binary crossentropy' as loss because we ended the model with a sigmoid function.

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

Before we fed to the model we need to format the data into appropriate preprocess floating point tensors. need to decode the JPEG to RGB grid of pixels, then to floating point tensors, then resize them to shared size and packing them into batches(by using batches of 32 images)

```
#image_dataset_from_directory is to setup a data pipeline that can automatically turn images to preprocessed tensors.
from tensorflow.keras.utils import image_dataset_from_directory

#this below directory it will do the subdirectories of directory and assume each one contains images from one of our classes.
#it will create and return tf.data.Dataset that inturns read, shuffle, and decode them.
train_datset = image_dataset_from_directory(
    new_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_datset = image_dataset_from_directory(
    new_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_datset = image_dataset_from_directory(
    new_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.
```

Displaying the Shapes and Labels of the Data and Dataset

```
for data_batch, labels_batch in train_datset:
    print("data_batch_shape:", data_batch.shape)
    print("labels_batch_shape:", labels_batch.shape)
    break
```

```
data_batch_shape: (32, 180, 180, 3)
labels_batch_shape: (32,)
```

Fitting the model using dataset

let's fit the model on our dataset, here the "callbacks" are used to save the model after each epoch(iteration).

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="conv_from_scratch1.keras",
#the below arguments tell the callback to only save a new file(replace) when val_loss metric is lower than any previous time during training
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_datset,
    epochs=30,
    validation_data=validation_datset,
    callbacks=callbacks)
```

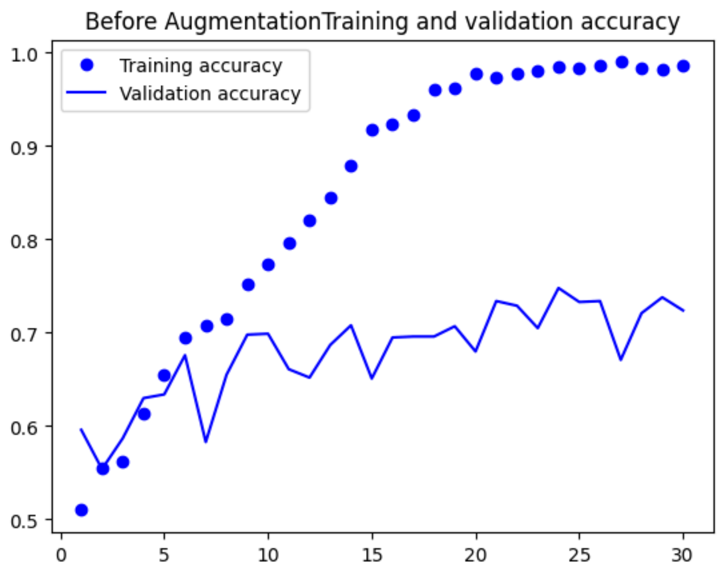
```
Epoch 1/30
63/63 [==============================] - 13s 111ms/step - loss: 0.7007 - accuracy: 0.5100 - val_loss: 0.6919 - val_accuracy: 0.5960
Epoch 2/30
63/63 [==============================] - 4s 62ms/step - loss: 0.6929 - accuracy: 0.5550 - val_loss: 0.6831 - val_accuracy: 0.5540
Epoch 3/30
63/63 [==============================] - 6s 87ms/step - loss: 0.6907 - accuracy: 0.5615 - val_loss: 0.6569 - val_accuracy: 0.5870
Epoch 4/30
63/63 [==============================] - 4s 63ms/step - loss: 0.6485 - accuracy: 0.6140 - val_loss: 0.6318 - val_accuracy: 0.6300
Epoch 5/30
63/63 [==============================] - 4s 63ms/step - loss: 0.6275 - accuracy: 0.6545 - val_loss: 0.6481 - val_accuracy: 0.6340
Epoch 6/30
63/63 [==============================] - 7s 108ms/step - loss: 0.5867 - accuracy: 0.6955 - val_loss: 0.6187 - val_accuracy: 0.6760
Epoch 7/30
63/63 [==============================] - 4s 61ms/step - loss: 0.5743 - accuracy: 0.7085 - val_loss: 0.7038 - val_accuracy: 0.5830
Epoch 8/30
63/63 [==============================] - 4s 66ms/step - loss: 0.5451 - accuracy: 0.7150 - val_loss: 0.7177 - val_accuracy: 0.6550
Epoch 9/30
63/63 [==============================] - 7s 111ms/step - loss: 0.5084 - accuracy: 0.7515 - val_loss: 0.6033 - val_accuracy: 0.6980
Epoch 10/30
63/63 [==============================] - 4s 64ms/step - loss: 0.4856 - accuracy: 0.7740 - val_loss: 0.6015 - val_accuracy: 0.6990
Epoch 11/30
63/63 [==============================] - 4s 60ms/step - loss: 0.4358 - accuracy: 0.7970 - val_loss: 0.8034 - val_accuracy: 0.6610
Epoch 12/30
63/63 [==============================] - 6s 97ms/step - loss: 0.4095 - accuracy: 0.8210 - val_loss: 0.7692 - val_accuracy: 0.6520
Epoch 13/30
63/63 [==============================] - 4s 66ms/step - loss: 0.3421 - accuracy: 0.8450 - val_loss: 0.7738 - val_accuracy: 0.6870
Epoch 14/30
63/63 [==============================] - 4s 64ms/step - loss: 0.2878 - accuracy: 0.8795 - val_loss: 0.8519 - val_accuracy: 0.7080
```

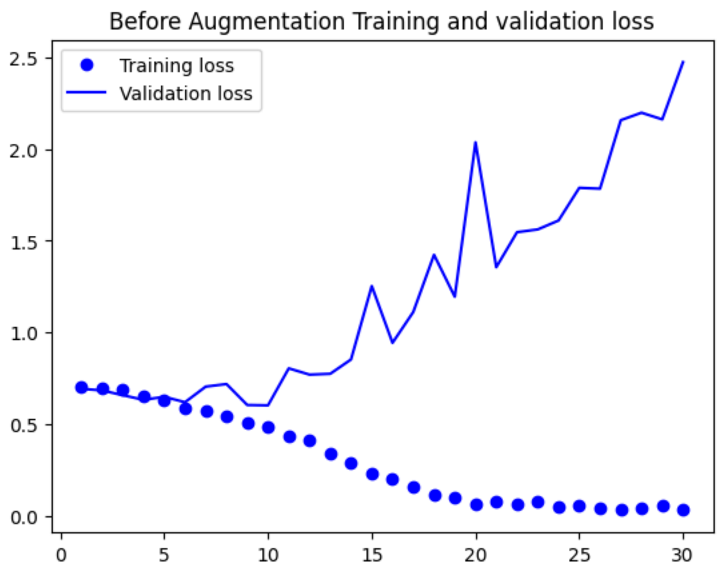
```
Epoch 15/30
63/63 [==============================] - 8s 113ms/step - loss: 0.2260 - accuracy: 0.9185 - val_loss: 1.2527 - val_accuracy: 0.6510
Epoch 16/30
63/63 [==============================] - 4s 61ms/step - loss: 0.2003 - accuracy: 0.9235 - val_loss: 0.9426 - val_accuracy: 0.6950
Epoch 17/30
63/63 [==============================] - 4s 61ms/step - loss: 0.1593 - accuracy: 0.9335 - val_loss: 1.1110 - val_accuracy: 0.6960
Epoch 18/30
63/63 [==============================] - 7s 100ms/step - loss: 0.1111 - accuracy: 0.9600 - val_loss: 1.4230 - val_accuracy: 0.6960
Epoch 19/30
63/63 [==============================] - 5s 76ms/step - loss: 0.0982 - accuracy: 0.9625 - val_loss: 1.1943 - val_accuracy: 0.7070
Epoch 20/30
63/63 [==============================] - 5s 73ms/step - loss: 0.0603 - accuracy: 0.9785 - val_loss: 2.0369 - val_accuracy: 0.6800
Epoch 21/30
63/63 [==============================] - 7s 97ms/step - loss: 0.0785 - accuracy: 0.9730 - val_loss: 1.3553 - val_accuracy: 0.7340
Epoch 22/30
63/63 [==============================] - 4s 59ms/step - loss: 0.0603 - accuracy: 0.9785 - val_loss: 1.5465 - val_accuracy: 0.7290
Epoch 23/30
63/63 [==============================] - 4s 58ms/step - loss: 0.0802 - accuracy: 0.9805 - val_loss: 1.5616 - val_accuracy: 0.7050
Epoch 24/30
63/63 [==============================] - 7s 102ms/step - loss: 0.0459 - accuracy: 0.9850 - val_loss: 1.6101 - val_accuracy: 0.7480
Epoch 25/30
63/63 [==============================] - 5s 73ms/step - loss: 0.0551 - accuracy: 0.9835 - val_loss: 1.7885 - val_accuracy: 0.7330
Epoch 26/30
63/63 [==============================] - 4s 60ms/step - loss: 0.0406 - accuracy: 0.9865 - val_loss: 1.7839 - val_accuracy: 0.7340
Epoch 27/30
63/63 [==============================] - 5s 83ms/step - loss: 0.0297 - accuracy: 0.9900 - val_loss: 2.1577 - val_accuracy: 0.6710
Epoch 28/30
63/63 [==============================] - 4s 63ms/step - loss: 0.0407 - accuracy: 0.9840 - val_loss: 2.1985 - val_accuracy: 0.7210
Epoch 29/30
63/63 [==============================] - 4s 62ms/step - loss: 0.0541 - accuracy: 0.9815 - val_loss: 2.1621 - val_accuracy: 0.7380
```

Displaying curves of loss and accuracy during training

let's plot the loss and accuracy of the model within the training and validation data during training.

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





from the above plots are the characteristics of overfitting, the training accuracy increases linearly overtime and nearly reaches 100% whereas validation accuracy is at only 73-75%. And the validation loss is stalls upto 10 epochs after it steadily increases where as training loss keeps decreasing linearly as training proceeds.

Evaluating the model on test set

Let's check test accuracy

```
test_model1 = keras.models.load_model("conv_from_scratch1.keras")
test_loss, test_acc = test_model1.evaluate(test_datset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 50ms/step - loss: 0.5577 - accuracy: 0.7190
Test accuracy: 0.719
```

we got a test accuracy of 70% because of less training data that leads to overfitting etc,. so that we need to work with specific one to computer vision when processing images with Deep learning models called Data Augmentation

Data Augmentation

Defining a data augmentation stage to add an image model

```
#we are doing random flip, random rotation, random zoom
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

Displaying randomly Augmented training images

It's just like dropout where it overcome overfitting they're inactive during inference, it will behave as same model like when we not include data augmentation and dropout.

Defining a convnet that includes image augmentation and dropout

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

Training the regularized convnet

we train the model using data augmentation and dropout to overcome overfitting we will train as many number of times----100

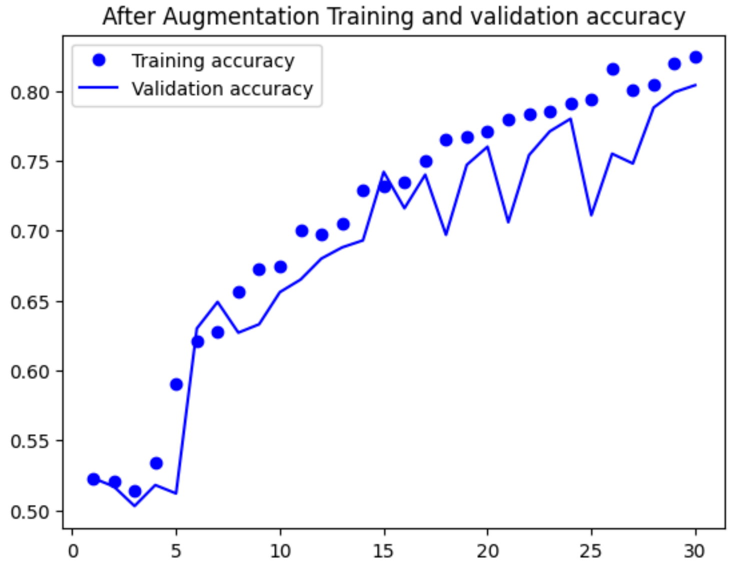
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="conv_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_datset,
    epochs=30,
    validation_data=validation_datset,
     callbacks=callbacks)
```

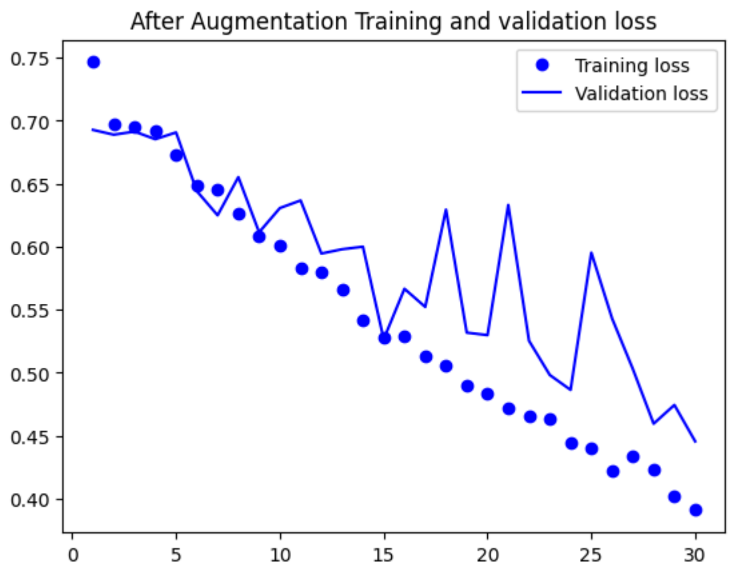
```
Epoch 1/30
63/63 [==============================] - 13s 85ms/step - loss: 0.7465 - accuracy: 0.5225 - val_loss: 0.6927 - val_accuracy: 0.5230
Epoch 2/30
63/63 [==============================] - 4s 60ms/step - loss: 0.6969 - accuracy: 0.5210 - val_loss: 0.6889 - val_accuracy: 0.5170
Epoch 3/30
63/63 [==============================] - 5s 83ms/step - loss: 0.6950 - accuracy: 0.5135 - val_loss: 0.6914 - val_accuracy: 0.5030
Epoch 4/30
63/63 [==============================] - 7s 96ms/step - loss: 0.6921 - accuracy: 0.5345 - val_loss: 0.6853 - val_accuracy: 0.5180
Epoch 5/30
63/63 [==============================] - 4s 60ms/step - loss: 0.6733 - accuracy: 0.5900 - val_loss: 0.6907 - val_accuracy: 0.5120
Epoch 6/30
63/63 [==============================] - 4s 60ms/step - loss: 0.6484 - accuracy: 0.6205 - val_loss: 0.6442 - val_accuracy: 0.6300
Epoch 7/30
63/63 [==============================] - 7s 101ms/step - loss: 0.6457 - accuracy: 0.6280 - val_loss: 0.6249 - val_accuracy: 0.6490
Epoch 8/30
63/63 [==============================] - 4s 59ms/step - loss: 0.6265 - accuracy: 0.6565 - val_loss: 0.6552 - val_accuracy: 0.6270
Epoch 9/30
63/63 [==============================] - 4s 64ms/step - loss: 0.6081 - accuracy: 0.6730 - val_loss: 0.6114 - val_accuracy: 0.6330
Epoch 10/30
63/63 [==============================] - 7s 109ms/step - loss: 0.6014 - accuracy: 0.6740 - val_loss: 0.6307 - val_accuracy: 0.6560
Epoch 11/30
63/63 [==============================] - 4s 60ms/step - loss: 0.5831 - accuracy: 0.7000 - val_loss: 0.6367 - val_accuracy: 0.6650
Epoch 12/30
```

```
63/63 [==============================] - 4s 60ms/step - loss: 0.5797 - accuracy: 0.6975 - val_loss: 0.5946 - val_accuracy: 0.6800
Epoch 13/30
63/63 [==============================] - 6s 96ms/step - loss: 0.5661 - accuracy: 0.7050 - val_loss: 0.5980 - val_accuracy: 0.6880
Epoch 14/30
63/63 [==============================] - 4s 60ms/step - loss: 0.5419 - accuracy: 0.7290 - val_loss: 0.6000 - val_accuracy: 0.6930
Epoch 15/30
63/63 [==============================] - 4s 60ms/step - loss: 0.5279 - accuracy: 0.7320 - val_loss: 0.5265 - val_accuracy: 0.7420
Epoch 16/30
63/63 [==============================] - 7s 108ms/step - loss: 0.5285 - accuracy: 0.7350 - val_loss: 0.5666 - val_accuracy: 0.7160
Epoch 17/30
63/63 [==============================] - 5s 65ms/step - loss: 0.5134 - accuracy: 0.7500 - val_loss: 0.5522 - val_accuracy: 0.7400
Epoch 18/30
63/63 [==============================] - 4s 67ms/step - loss: 0.5053 - accuracy: 0.7655 - val_loss: 0.6295 - val_accuracy: 0.6970
Epoch 19/30
63/63 [==============================] - 6s 87ms/step - loss: 0.4896 - accuracy: 0.7675 - val_loss: 0.5318 - val_accuracy: 0.7470
Epoch 20/30
63/63 [==============================] - 6s 87ms/step - loss: 0.4835 - accuracy: 0.7710 - val_loss: 0.5298 - val_accuracy: 0.7600
Epoch 21/30
63/63 [==============================] - 4s 60ms/step - loss: 0.4719 - accuracy: 0.7800 - val_loss: 0.6331 - val_accuracy: 0.7060
Epoch 22/30
63/63 [==============================] - 4s 60ms/step - loss: 0.4657 - accuracy: 0.7830 - val_loss: 0.5252 - val_accuracy: 0.7540
Epoch 23/30
63/63 [==============================] - 6s 91ms/step - loss: 0.4631 - accuracy: 0.7850 - val_loss: 0.4981 - val_accuracy: 0.7710
Epoch 24/30
63/63 [==============================] - 4s 66ms/step - loss: 0.4438 - accuracy: 0.7910 - val_loss: 0.4863 - val_accuracy: 0.7800
Epoch 25/30
63/63 [==============================] - 5s 74ms/step - loss: 0.4397 - accuracy: 0.7935 - val_loss: 0.5952 - val_accuracy: 0.7110
Epoch 26/30
63/63 [==============================] - 7s 109ms/step - loss: 0.4215 - accuracy: 0.8160 - val_loss: 0.5430 - val_accuracy: 0.7550
Epoch 27/30
63/63 [==============================] - 4s 65ms/step - loss: 0.4340 - accuracy: 0.8010 - val_loss: 0.5028 - val_accuracy: 0.7480
Epoch 28/30
63/63 [==============================] - 6s 94ms/step - loss: 0.4226 - accuracy: 0.8045 - val_loss: 0.4594 - val_accuracy: 0.7880
Epoch 29/30
63/63 [==============================] - 4s 62ms/step - loss: 0.4021 - accuracy: 0.8195 - val_loss: 0.4743 - val_accuracy: 0.7990
```

Now let's see again the curves for loss and accuracy during training

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





Re-evaluating the model on the test dataset

```
test_model2 = keras.models.load_model(
    "conv_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model2.evaluate(test_datset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 31ms/step - loss: 0.4280 - accuracy: 0.8130
Test accuracy: 0.813
```

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

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

Training the regularized convnet

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="conv_from_scratch_with_dropout.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_datset,
    epochs=30,
    validation_data=validation_datset,
    callbacks=callbacks)
```

```
Epoch 1/30
63/63 [==============================] - 10s 88ms/step - loss: 0.6965 - accuracy: 0.5225 - val_loss: 0.6893 - val_accuracy: 0.5370
Epoch 2/30
63/63 [==============================] - 4s 59ms/step - loss: 0.6917 - accuracy: 0.5585 - val_loss: 0.6930 - val_accuracy: 0.5330
Epoch 3/30
63/63 [==============================] - 4s 58ms/step - loss: 0.6700 - accuracy: 0.6075 - val_loss: 0.6606 - val_accuracy: 0.5880
Epoch 4/30
63/63 [==============================] - 6s 99ms/step - loss: 0.6568 - accuracy: 0.6075 - val_loss: 0.6331 - val_accuracy: 0.6460
Epoch 5/30
63/63 [==============================] - 5s 73ms/step - loss: 0.6049 - accuracy: 0.6625 - val_loss: 0.8061 - val_accuracy: 0.6000
Epoch 6/30
63/63 [==============================] - 4s 58ms/step - loss: 0.5877 - accuracy: 0.6940 - val_loss: 0.5810 - val_accuracy: 0.6890
Epoch 7/30
63/63 [==============================] - 5s 82ms/step - loss: 0.5686 - accuracy: 0.7020 - val_loss: 0.6076 - val_accuracy: 0.6770
Epoch 8/30
63/63 [==============================] - 4s 56ms/step - loss: 0.5390 - accuracy: 0.7335 - val_loss: 0.6135 - val_accuracy: 0.6910
Epoch 9/30
63/63 [==============================] - 4s 61ms/step - loss: 0.5123 - accuracy: 0.7600 - val_loss: 0.5449 - val_accuracy: 0.7340
Epoch 10/30
63/63 [==============================] - 7s 105ms/step - loss: 0.4633 - accuracy: 0.7865 - val_loss: 0.7674 - val_accuracy: 0.6850
Epoch 11/30
63/63 [==============================] - 5s 69ms/step - loss: 0.4518 - accuracy: 0.8000 - val_loss: 0.5611 - val_accuracy: 0.7380
Epoch 12/30
63/63 [==============================] - 4s 59ms/step - loss: 0.3944 - accuracy: 0.8270 - val_loss: 0.6863 - val_accuracy: 0.7310
Epoch 13/30
63/63 [==============================] - 5s 82ms/step - loss: 0.3780 - accuracy: 0.8330 - val_loss: 0.6635 - val_accuracy: 0.7040
Epoch 14/30
63/63 [==============================] - 4s 57ms/step - loss: 0.3182 - accuracy: 0.8585 - val_loss: 0.5881 - val_accuracy: 0.7530
Epoch 15/30
63/63 [==============================] - 4s 59ms/step - loss: 0.2725 - accuracy: 0.8815 - val_loss: 0.7313 - val_accuracy: 0.7370
Epoch 16/30
63/63 [==============================] - 6s 96ms/step - loss: 0.2408 - accuracy: 0.9030 - val_loss: 0.6847 - val_accuracy: 0.7590
Epoch 17/30
63/63 [==============================] - 4s 64ms/step - loss: 0.2078 - accuracy: 0.9105 - val_loss: 0.7902 - val_accuracy: 0.7590
Epoch 18/30
63/63 [==============================] - 4s 58ms/step - loss: 0.1711 - accuracy: 0.9345 - val_loss: 0.8610 - val_accuracy: 0.7490
Epoch 19/30
63/63 [==============================] - 6s 97ms/step - loss: 0.1548 - accuracy: 0.9410 - val_loss: 0.8095 - val_accuracy: 0.7680
Epoch 20/30
63/63 [==============================] - 4s 59ms/step - loss: 0.1270 - accuracy: 0.9505 - val_loss: 0.9124 - val_accuracy: 0.7380
Epoch 21/30
63/63 [==============================] - 4s 62ms/step - loss: 0.1200 - accuracy: 0.9585 - val_loss: 0.8908 - val_accuracy: 0.7710
Epoch 22/30
63/63 [==============================] - 7s 100ms/step - loss: 0.0973 - accuracy: 0.9700 - val_loss: 1.1998 - val_accuracy: 0.7380
Epoch 23/30
```

```
63/63 [==============================] - 4s 57ms/step - loss: 0.0897 - accuracy: 0.9670 - val_loss: 0.9446 - val_accuracy: 0.7610
Epoch 24/30
63/63 [==============================] - 4s 58ms/step - loss: 0.0636 - accuracy: 0.9790 - val_loss: 1.4739 - val_accuracy: 0.7480
Epoch 25/30
63/63 [==============================] - 7s 110ms/step - loss: 0.0838 - accuracy: 0.9685 - val_loss: 1.2676 - val_accuracy: 0.7570
Epoch 26/30
63/63 [==============================] - 4s 57ms/step - loss: 0.0785 - accuracy: 0.9735 - val_loss: 0.9805 - val_accuracy: 0.7740
Epoch 27/30
63/63 [==============================] - 4s 58ms/step - loss: 0.0587 - accuracy: 0.9785 - val_loss: 1.4318 - val_accuracy: 0.7170
Epoch 28/30
63/63 [==============================] - 6s 99ms/step - loss: 0.0644 - accuracy: 0.9770 - val_loss: 1.1747 - val_accuracy: 0.7760
Epoch 29/30
63/63 [==============================] - 4s 57ms/step - loss: 0.0618 - accuracy: 0.9755 - val_loss: 1.1728 - val_accuracy: 0.7590
```

```
test_model2 = keras.models.load_model(
    "conv_from_scratch_with_dropout.keras")
test_loss, test_acc = test_model2.evaluate(test_datset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 30ms/step - loss: 0.5321 - accuracy: 0.7450
Test accuracy: 0.745
```

Using Image Augmentation and Dropout method

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

Here a new convnet that includes both image augmentation and dropout

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

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

Training the regularized convnet

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="conv_from_scratch_with_augmentation_dropout.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_datset,
    epochs=30,
    validation_data=validation_datset,
    callbacks=callbacks)
```

```
Epoch 1/30
63/63 [==============================] - 11s 96ms/step - loss: 0.6970 - accuracy: 0.4915 - val_loss: 0.6941 - val_accuracy: 0.5000
Epoch 2/30
63/63 [==============================] - 4s 58ms/step - loss: 0.6938 - accuracy: 0.5020 - val_loss: 0.6927 - val_accuracy: 0.5000
Epoch 3/30
```

```
63/63 [==============================] - 4s 60ms/step - loss: 0.6902 - accuracy: 0.5435 - val_loss: 0.6759 - val_accuracy: 0.5900
Epoch 4/30
63/63 [==============================] - 7s 109ms/step - loss: 0.6788 - accuracy: 0.5700 - val_loss: 0.6722 - val_accuracy: 0.5760
Epoch 5/30
63/63 [==============================] - 4s 59ms/step - loss: 0.6681 - accuracy: 0.6055 - val_loss: 0.6649 - val_accuracy: 0.5940
Epoch 6/30
63/63 [==============================] - 6s 85ms/step - loss: 0.6464 - accuracy: 0.6180 - val_loss: 0.6449 - val_accuracy: 0.6230
Epoch 7/30
63/63 [==============================] - 6s 90ms/step - loss: 0.6259 - accuracy: 0.6400 - val_loss: 0.6377 - val_accuracy: 0.6530
Epoch 8/30
63/63 [==============================] - 4s 59ms/step - loss: 0.6284 - accuracy: 0.6480 - val_loss: 0.6035 - val_accuracy: 0.6820
Epoch 9/30
63/63 [==============================] - 7s 103ms/step - loss: 0.6097 - accuracy: 0.6480 - val_loss: 0.6074 - val_accuracy: 0.6580
Epoch 10/30
63/63 [==============================] - 4s 60ms/step - loss: 0.5929 - accuracy: 0.6745 - val_loss: 0.6122 - val_accuracy: 0.6760
Epoch 11/30
63/63 [==============================] - 5s 71ms/step - loss: 0.6023 - accuracy: 0.6725 - val_loss: 0.6102 - val_accuracy: 0.6700
Epoch 12/30
63/63 [==============================] - 6s 98ms/step - loss: 0.5943 - accuracy: 0.6770 - val_loss: 0.6238 - val_accuracy: 0.6310
Epoch 13/30
63/63 [==============================] - 4s 58ms/step - loss: 0.5826 - accuracy: 0.7005 - val_loss: 0.5796 - val_accuracy: 0.7010
Epoch 14/30
63/63 [==============================] - 4s 59ms/step - loss: 0.5750 - accuracy: 0.6965 - val_loss: 0.5600 - val_accuracy: 0.7090
Epoch 15/30
63/63 [==============================] - 5s 79ms/step - loss: 0.5582 - accuracy: 0.7155 - val_loss: 0.5501 - val_accuracy: 0.7420
Epoch 16/30
63/63 [==============================] - 4s 60ms/step - loss: 0.5555 - accuracy: 0.7270 - val_loss: 0.7511 - val_accuracy: 0.5810
Epoch 17/30
63/63 [==============================] - 4s 58ms/step - loss: 0.5248 - accuracy: 0.7420 - val_loss: 0.5991 - val_accuracy: 0.6780
Epoch 18/30
63/63 [==============================] - 7s 110ms/step - loss: 0.5193 - accuracy: 0.7455 - val_loss: 0.5043 - val_accuracy: 0.7650
Epoch 19/30
63/63 [==============================] - 4s 59ms/step - loss: 0.5200 - accuracy: 0.7540 - val_loss: 0.4816 - val_accuracy: 0.7910
Epoch 20/30
63/63 [==============================] - 4s 63ms/step - loss: 0.5170 - accuracy: 0.7510 - val_loss: 0.4976 - val_accuracy: 0.7590
Epoch 21/30
63/63 [==============================] - 7s 99ms/step - loss: 0.4968 - accuracy: 0.7555 - val_loss: 0.4893 - val_accuracy: 0.7810
Epoch 22/30
63/63 [==============================] - 4s 58ms/step - loss: 0.4897 - accuracy: 0.7780 - val_loss: 0.4942 - val_accuracy: 0.7630
Epoch 23/30
63/63 [==============================] - 4s 58ms/step - loss: 0.4744 - accuracy: 0.7795 - val_loss: 0.4843 - val_accuracy: 0.7890
Epoch 24/30
63/63 [==============================] - 6s 89ms/step - loss: 0.4580 - accuracy: 0.7870 - val_loss: 0.5166 - val_accuracy: 0.7530
Epoch 25/30
63/63 [==============================] - 6s 87ms/step - loss: 0.4645 - accuracy: 0.7825 - val_loss: 0.4612 - val_accuracy: 0.8060
Epoch 26/30
63/63 [==============================] - 4s 58ms/step - loss: 0.4557 - accuracy: 0.7845 - val_loss: 0.4478 - val_accuracy: 0.8030
Epoch 27/30
63/63 [==============================] - 4s 57ms/step - loss: 0.4513 - accuracy: 0.7945 - val_loss: 0.4549 - val_accuracy: 0.8020
Epoch 28/30
63/63 [==============================] - 7s 100ms/step - loss: 0.4420 - accuracy: 0.7905 - val_loss: 0.4620 - val_accuracy: 0.7880
Epoch 29/30
```

Evaluating the model on the test set

```
test_model2 = keras.models.load_model(
    "conv_from_scratch_with_augmentation_dropout.keras")
test_loss, test_acc = test_model2.evaluate(test_datset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 30ms/step - loss: 0.3819 - accuracy: 0.8370
Test accuracy: 0.837
```

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?

Here i am increasing the samples to 8000 and the model performance needs to be evaluated.

The technique here i am using data augmentation and dropout due to the performance was high based on the previous models by using this.

```
make_subset("train2", start_index=1000, end_index=8000)

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

```
Found 14000 files belonging to 2 classes.
```

New convnet that includes both image augmentation and dropout

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

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

Training a reglularized convnet

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="conv_from_scratch1.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset_2,
    epochs=30,
    validation_data=validation_datset,
    callbacks=callbacks)
```

```
Epoch 1/30
438/438 [==============================] - 26s 52ms/step - loss: 0.6673 - accuracy: 0.5828 - val_loss: 0.5957 - val_accuracy: 0.7140
Epoch 2/30
438/438 [==============================] - 24s 54ms/step - loss: 0.5509 - accuracy: 0.7215 - val_loss: 0.5175 - val_accuracy: 0.7290
Epoch 3/30
438/438 [==============================] - 24s 55ms/step - loss: 0.4690 - accuracy: 0.7779 - val_loss: 0.4410 - val_accuracy: 0.7910
Epoch 4/30
438/438 [==============================] - 23s 52ms/step - loss: 0.3968 - accuracy: 0.8217 - val_loss: 0.4998 - val_accuracy: 0.7720
Epoch 5/30
438/438 [==============================] - 25s 55ms/step - loss: 0.3335 - accuracy: 0.8546 - val_loss: 0.3691 - val_accuracy: 0.8340
Epoch 6/30
438/438 [==============================] - 23s 51ms/step - loss: 0.2815 - accuracy: 0.8814 - val_loss: 0.3567 - val_accuracy: 0.8550
Epoch 7/30
438/438 [==============================] - 25s 56ms/step - loss: 0.2422 - accuracy: 0.8959 - val_loss: 0.3797 - val_accuracy: 0.8510
Epoch 8/30
438/438 [==============================] - 25s 55ms/step - loss: 0.2046 - accuracy: 0.9181 - val_loss: 0.2950 - val_accuracy: 0.8780
Epoch 9/30
438/438 [==============================] - 23s 51ms/step - loss: 0.1730 - accuracy: 0.9301 - val_loss: 0.2792 - val_accuracy: 0.8960
Epoch 10/30
438/438 [==============================] - 25s 57ms/step - loss: 0.1392 - accuracy: 0.9449 - val_loss: 0.2526 - val_accuracy: 0.9060
Epoch 11/30
438/438 [==============================] - 23s 52ms/step - loss: 0.1323 - accuracy: 0.9489 - val_loss: 0.3134 - val_accuracy: 0.9040
Epoch 12/30
438/438 [==============================] - 23s 51ms/step - loss: 0.1120 - accuracy: 0.9598 - val_loss: 0.2467 - val_accuracy: 0.9080
Epoch 13/30
438/438 [==============================] - 27s 60ms/step - loss: 0.1043 - accuracy: 0.9643 - val_loss: 0.3992 - val_accuracy: 0.9010
Epoch 14/30
438/438 [==============================] - 23s 52ms/step - loss: 0.0971 - accuracy: 0.9655 - val_loss: 0.3568 - val_accuracy: 0.9080
Epoch 15/30
438/438 [==============================] - 24s 54ms/step - loss: 0.0911 - accuracy: 0.9702 - val_loss: 0.3738 - val_accuracy: 0.8840
Epoch 16/30
438/438 [==============================] - 24s 54ms/step - loss: 0.0960 - accuracy: 0.9701 - val_loss: 0.4913 - val_accuracy: 0.8950
Epoch 17/30
438/438 [==============================] - 23s 51ms/step - loss: 0.0854 - accuracy: 0.9728 - val_loss: 0.3633 - val_accuracy: 0.9000
Epoch 18/30
438/438 [==============================] - 23s 52ms/step - loss: 0.1052 - accuracy: 0.9681 - val_loss: 0.3975 - val_accuracy: 0.9110
Epoch 19/30
438/438 [==============================] - 25s 56ms/step - loss: 0.0858 - accuracy: 0.9724 - val_loss: 0.3899 - val_accuracy: 0.8980
```

```
Epoch 20/30
438/438 [==============================] - 25s 55ms/step - loss: 0.0849 - accuracy: 0.9747 - val_loss: 0.4111 - val_accuracy: 0.9060
Epoch 21/30
438/438 [==============================] - 23s 53ms/step - loss: 0.0949 - accuracy: 0.9724 - val_loss: 0.4202 - val_accuracy: 0.9140
Epoch 22/30
438/438 [==============================] - 25s 57ms/step - loss: 0.0922 - accuracy: 0.9744 - val_loss: 0.4486 - val_accuracy: 0.9030
Epoch 23/30
438/438 [==============================] - 24s 53ms/step - loss: 0.0920 - accuracy: 0.9748 - val_loss: 0.5075 - val_accuracy: 0.8990
Epoch 24/30
438/438 [==============================] - 23s 52ms/step - loss: 0.0929 - accuracy: 0.9757 - val_loss: 0.8500 - val_accuracy: 0.9110
Epoch 25/30
438/438 [==============================] - 23s 53ms/step - loss: 0.1146 - accuracy: 0.9749 - val_loss: 0.5636 - val_accuracy: 0.9070
Epoch 26/30
438/438 [==============================] - 25s 55ms/step - loss: 0.1021 - accuracy: 0.9743 - val_loss: 0.9800 - val_accuracy: 0.8700
Epoch 27/30
438/438 [==============================] - 24s 55ms/step - loss: 0.1006 - accuracy: 0.9770 - val_loss: 0.5373 - val_accuracy: 0.9100
Epoch 28/30
438/438 [==============================] - 25s 57ms/step - loss: 0.0980 - accuracy: 0.9784 - val_loss: 0.4509 - val_accuracy: 0.9300
Epoch 29/30
438/438 [==============================] - 25s 56ms/step - loss: 0.0980 - accuracy: 0.9768 - val_loss: 0.5569 - val_accuracy: 0.9010
```

Evaluating the model with test set

```
test_model3 = keras.models.load_model(
    "conv_from_scratch1.keras")
test_loss, test_acc = test_model3.evaluate(test_datset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 48ms/step - loss: 0.2946 - accuracy: 0.9030
Test accuracy: 0.903
```

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.

Increased the samples from 8000 to 10000 in order to check the efficiency of the model.

```
make_subset("train_3", start_index=1000, end_index=10000)

train_dataset_3 = image_dataset_from_directory(
    new_dir / "train_3",
    image_size=(180, 180),
    batch_size=32)
```

```
Found 18000 files belonging to 2 classes.
```

Model Building with both Image augmentation and dropout

new convnet that includes both image augmentation and dropout

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

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

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

Training the regularized convnet

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="conv_from_scratch_test1.keras",
        save_best_only=True,
        monitor="val_loss")
]

history = model.fit(
    train_dataset_2,
    epochs=25,
    validation_data=validation_datset,
    callbacks=callbacks)
```

```
Epoch 1/25
438/438 [==============================] - 27s 57ms/step - loss: 0.6595 - accuracy: 0.5914 - val_loss: 0.6023 - val_accuracy: 0.6850
Epoch 2/25
438/438 [==============================] - 23s 52ms/step - loss: 0.5421 - accuracy: 0.7316 - val_loss: 0.7450 - val_accuracy: 0.6050
Epoch 3/25
438/438 [==============================] - 25s 57ms/step - loss: 0.4407 - accuracy: 0.7956 - val_loss: 0.4489 - val_accuracy: 0.7950
Epoch 4/25
438/438 [==============================] - 25s 55ms/step - loss: 0.3680 - accuracy: 0.8350 - val_loss: 0.4123 - val_accuracy: 0.8320
Epoch 5/25
438/438 [==============================] - 24s 53ms/step - loss: 0.3038 - accuracy: 0.8701 - val_loss: 0.3439 - val_accuracy: 0.8510
Epoch 6/25
438/438 [==============================] - 25s 56ms/step - loss: 0.2429 - accuracy: 0.8995 - val_loss: 0.3424 - val_accuracy: 0.8780
Epoch 7/25
438/438 [==============================] - 25s 55ms/step - loss: 0.1961 - accuracy: 0.9194 - val_loss: 0.5182 - val_accuracy: 0.8270
Epoch 8/25
438/438 [==============================] - 24s 55ms/step - loss: 0.1526 - accuracy: 0.9404 - val_loss: 0.4974 - val_accuracy: 0.8430
Epoch 9/25
438/438 [==============================] - 26s 58ms/step - loss: 0.1220 - accuracy: 0.9531 - val_loss: 0.3628 - val_accuracy: 0.8660
Epoch 10/25
438/438 [==============================] - 23s 52ms/step - loss: 0.0938 - accuracy: 0.9656 - val_loss: 0.4571 - val_accuracy: 0.8710
Epoch 11/25
438/438 [==============================] - 23s 52ms/step - loss: 0.0861 - accuracy: 0.9687 - val_loss: 0.3912 - val_accuracy: 0.8910
Epoch 12/25
438/438 [==============================] - 24s 54ms/step - loss: 0.0807 - accuracy: 0.9718 - val_loss: 0.3778 - val_accuracy: 0.8900
Epoch 13/25
438/438 [==============================] - 26s 57ms/step - loss: 0.0645 - accuracy: 0.9781 - val_loss: 0.6134 - val_accuracy: 0.8860
Epoch 14/25
438/438 [==============================] - 23s 52ms/step - loss: 0.0696 - accuracy: 0.9779 - val_loss: 0.5577 - val_accuracy: 0.8980
Epoch 15/25
438/438 [==============================] - 23s 52ms/step - loss: 0.0638 - accuracy: 0.9802 - val_loss: 0.7219 - val_accuracy: 0.8760
Epoch 16/25
438/438 [==============================] - 27s 59ms/step - loss: 0.0632 - accuracy: 0.9799 - val_loss: 0.7278 - val_accuracy: 0.8950
Epoch 17/25
438/438 [==============================] - 23s 52ms/step - loss: 0.0671 - accuracy: 0.9826 - val_loss: 0.6153 - val_accuracy: 0.8910
Epoch 18/25
438/438 [==============================] - 24s 55ms/step - loss: 0.0803 - accuracy: 0.9798 - val_loss: 0.5751 - val_accuracy: 0.9080
Epoch 19/25
438/438 [==============================] - 24s 54ms/step - loss: 0.0554 - accuracy: 0.9851 - val_loss: 1.0150 - val_accuracy: 0.8760
Epoch 20/25
438/438 [==============================] - 23s 53ms/step - loss: 0.0646 - accuracy: 0.9837 - val_loss: 1.0091 - val_accuracy: 0.8850
Epoch 21/25
438/438 [==============================] - 23s 52ms/step - loss: 0.0693 - accuracy: 0.9836 - val_loss: 0.8578 - val_accuracy: 0.8770
Epoch 22/25
438/438 [==============================] - 25s 55ms/step - loss: 0.0651 - accuracy: 0.9829 - val_loss: 0.8537 - val_accuracy: 0.8850
Epoch 23/25
438/438 [==============================] - 23s 53ms/step - loss: 0.0679 - accuracy: 0.9838 - val_loss: 1.1654 - val_accuracy: 0.8830
Epoch 24/25
438/438 [==============================] - 25s 56ms/step - loss: 0.0586 - accuracy: 0.9873 - val_loss: 1.0750 - val_accuracy: 0.8930
Epoch 25/25
438/438 [==============================] - 24s 54ms/step - loss: 0.0726 - accuracy: 0.9856 - val_loss: 1.0076 - val_accuracy: 0.8940
```

Evaluating the model with test set

```
test_model4 = keras.models.load_model(
    "conv_from_scratch_test1.keras")
test_loss, test_acc = test_model4.evaluate(test_datset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 28ms/step - loss: 0.3098 - accuracy: 0.8840
Test accuracy: 0.884
```

with dropout

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

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

Training the regularized convnet

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="conv_from_scratch2.keras",
        save_best_only=True,
        monitor="val_loss")
]

history = model.fit(
    train_dataset_2,
    epochs=20,
    validation_data=validation_datset,
    callbacks=callbacks)
```

```
Epoch 1/20
438/438 [==============================] - 30s 53ms/step - loss: 0.6777 - accuracy: 0.5608 - val_loss: 0.6372 - val_accuracy: 0.6330
Epoch 2/20
438/438 [==============================] - 23s 52ms/step - loss: 0.5835 - accuracy: 0.6911 - val_loss: 0.5352 - val_accuracy: 0.7340
Epoch 3/20
438/438 [==============================] - 26s 58ms/step - loss: 0.4918 - accuracy: 0.7659 - val_loss: 0.4132 - val_accuracy: 0.8090
Epoch 4/20
438/438 [==============================] - 24s 53ms/step - loss: 0.4067 - accuracy: 0.8130 - val_loss: 0.3892 - val_accuracy: 0.8100
Epoch 5/20
438/438 [==============================] - 23s 52ms/step - loss: 0.3437 - accuracy: 0.8489 - val_loss: 0.6646 - val_accuracy: 0.7300
Epoch 6/20
438/438 [==============================] - 24s 55ms/step - loss: 0.2996 - accuracy: 0.8723 - val_loss: 0.4283 - val_accuracy: 0.8140
Epoch 7/20
438/438 [==============================] - 29s 65ms/step - loss: 0.2544 - accuracy: 0.8940 - val_loss: 0.2669 - val_accuracy: 0.8810
Epoch 8/20
438/438 [==============================] - 25s 56ms/step - loss: 0.2078 - accuracy: 0.9151 - val_loss: 0.4505 - val_accuracy: 0.8260
Epoch 9/20
438/438 [==============================] - 24s 55ms/step - loss: 0.1765 - accuracy: 0.9284 - val_loss: 0.4692 - val_accuracy: 0.8300
Epoch 10/20
438/438 [==============================] - 25s 56ms/step - loss: 0.1458 - accuracy: 0.9426 - val_loss: 0.3434 - val_accuracy: 0.8810
Epoch 11/20
438/438 [==============================] - 25s 56ms/step - loss: 0.1291 - accuracy: 0.9506 - val_loss: 0.3379 - val_accuracy: 0.9030
Epoch 12/20
438/438 [==============================] - 23s 53ms/step - loss: 0.1198 - accuracy: 0.9572 - val_loss: 0.3849 - val_accuracy: 0.8850
Epoch 13/20
438/438 [==============================] - 26s 58ms/step - loss: 0.1039 - accuracy: 0.9625 - val_loss: 0.3719 - val_accuracy: 0.8940
Epoch 14/20
438/438 [==============================] - 24s 55ms/step - loss: 0.0969 - accuracy: 0.9646 - val_loss: 0.4773 - val_accuracy: 0.8820
Epoch 15/20
438/438 [==============================] - 23s 52ms/step - loss: 0.0956 - accuracy: 0.9659 - val_loss: 0.5225 - val_accuracy: 0.8820
Epoch 16/20
438/438 [==============================] - 26s 59ms/step - loss: 0.0849 - accuracy: 0.9719 - val_loss: 0.4438 - val_accuracy: 0.9090
Epoch 17/20
438/438 [==============================] - 25s 56ms/step - loss: 0.0914 - accuracy: 0.9701 - val_loss: 0.4615 - val_accuracy: 0.8980
Epoch 18/20
438/438 [==============================] - 24s 54ms/step - loss: 0.0795 - accuracy: 0.9744 - val_loss: 0.6190 - val_accuracy: 0.8690
Epoch 19/20
438/438 [==============================] - 25s 57ms/step - loss: 0.0894 - accuracy: 0.9731 - val_loss: 0.5070 - val_accuracy: 0.8940
Epoch 20/20
438/438 [==============================] - 26s 59ms/step - loss: 0.0815 - accuracy: 0.9751 - val_loss: 0.4941 - val_accuracy: 0.9100
```

evaluating the model with test set

```
test_model = keras.models.load_model(
    "conv_from_scratch2.keras")
test_loss, test_acc = test_model.evaluate(test_datset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 29ms/step - loss: 0.2562 - accuracy: 0.8920
Test accuracy: 0.892
```

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

Pre-training model--1000 training samples

Here install and freezing the VGG16 convolution base

```
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.
58889256/58889256 [==============================] - 0s 0us/step
```

Let's get the summary of the convbase.

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_8 (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)
 _________________________________________________________________
```

Feature extraction with data augmentation

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

Adding a data augmentation and a classifier to the convnet base.

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.3),
        layers.RandomZoom(0.5),
    ]
)

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

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

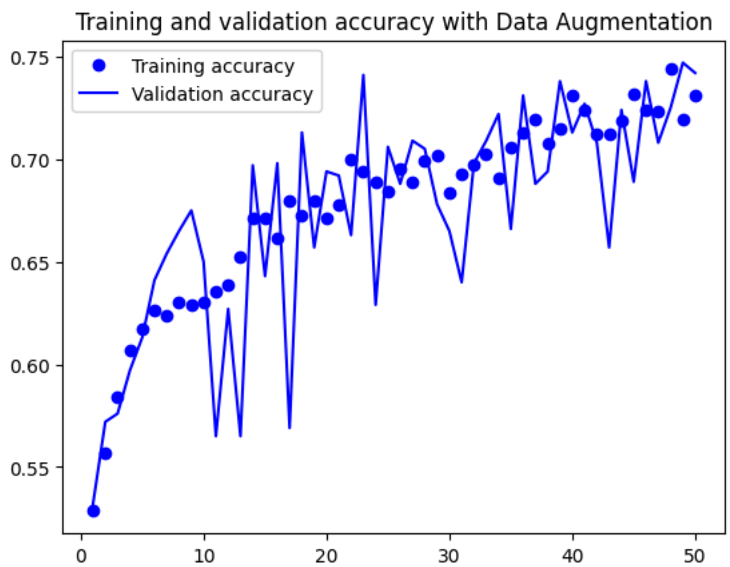
Training the regularized convnet

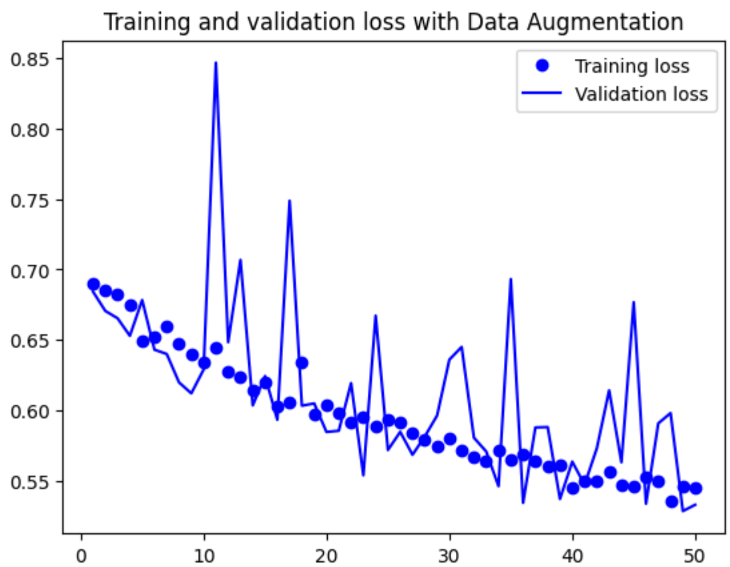
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
   train_datset,
   epochs=50,
   validation_data=validation_datset,
   callbacks=callbacks)
```

```
63/63 [==============================] - 4s 59ms/step - loss: 0.5667 - accuracy: 0.6970 - val_loss: 0.5805 - val_accuracy: 0.6980
Epoch 33/50
63/63 [==============================] - 7s 105ms/step - loss: 0.5643 - accuracy: 0.7025 - val_loss: 0.5707 - val_accuracy: 0.7090
Epoch 34/50
63/63 [==============================] - 5s 72ms/step - loss: 0.5717 - accuracy: 0.6905 - val_loss: 0.5461 - val_accuracy: 0.7220
Epoch 35/50
63/63 [==============================] - 7s 101ms/step - loss: 0.5653 - accuracy: 0.7055 - val_loss: 0.6932 - val_accuracy: 0.6660
Epoch 36/50
63/63 [==============================] - 8s 111ms/step - loss: 0.5684 - accuracy: 0.7130 - val_loss: 0.5343 - val_accuracy: 0.7310
Epoch 37/50
63/63 [==============================] - 4s 58ms/step - loss: 0.5641 - accuracy: 0.7195 - val_loss: 0.5878 - val_accuracy: 0.6880
Epoch 38/50
63/63 [==============================] - 4s 58ms/step - loss: 0.5602 - accuracy: 0.7075 - val_loss: 0.5881 - val_accuracy: 0.6940
Epoch 39/50
63/63 [==============================] - 6s 95ms/step - loss: 0.5609 - accuracy: 0.7145 - val_loss: 0.5370 - val_accuracy: 0.7380
Epoch 40/50
63/63 [==============================] - 4s 59ms/step - loss: 0.5452 - accuracy: 0.7310 - val_loss: 0.5636 - val_accuracy: 0.7130
Epoch 41/50
63/63 [==============================] - 5s 84ms/step - loss: 0.5498 - accuracy: 0.7240 - val_loss: 0.5472 - val_accuracy: 0.7270
Epoch 42/50
63/63 [==============================] - 4s 58ms/step - loss: 0.5495 - accuracy: 0.7125 - val_loss: 0.5731 - val_accuracy: 0.7080
Epoch 43/50
63/63 [==============================] - 4s 63ms/step - loss: 0.5562 - accuracy: 0.7120 - val_loss: 0.6142 - val_accuracy: 0.6570
Epoch 44/50
63/63 [==============================] - 7s 106ms/step - loss: 0.5472 - accuracy: 0.7185 - val_loss: 0.5631 - val_accuracy: 0.7240
Epoch 45/50
63/63 [==============================] - 5s 72ms/step - loss: 0.5458 - accuracy: 0.7320 - val_loss: 0.6768 - val_accuracy: 0.6890
Epoch 46/50
63/63 [==============================] - 4s 63ms/step - loss: 0.5530 - accuracy: 0.7240 - val_loss: 0.5337 - val_accuracy: 0.7380
Epoch 47/50
63/63 [==============================] - 5s 71ms/step - loss: 0.5501 - accuracy: 0.7230 - val_loss: 0.5907 - val_accuracy: 0.7080
Epoch 48/50
63/63 [==============================] - 8s 113ms/step - loss: 0.5354 - accuracy: 0.7440 - val_loss: 0.5980 - val_accuracy: 0.7250
Epoch 49/50
63/63 [==============================] - 4s 60ms/step - loss: 0.5460 - accuracy: 0.7195 - val_loss: 0.5284 - val_accuracy: 0.7470
Epoch 50/50
63/63 [==============================] - 4s 59ms/step - loss: 0.5449 - accuracy: 0.7310 - val_loss: 0.5329 - val_accuracy: 0.7420
```

Plotting the curves for loss and accuracy during training

```python
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 with Data Augmentation")
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 with Data Augmentation")
plt.legend()
plt.show()
```





Evaluating the model on the test set

```
test_model5 = keras.models.load_model("convnet_from_scratch_augmentation.keras")
test_loss, test_acc = test_model5.evaluate(test_datset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 53ms/step - loss: 0.5003 - accuracy: 0.7650
Test accuracy: 0.765
```

Leveraging a Pretrained model

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

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_14 (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)
_________________________________________________________________
```

Extracting the VGG16 features and corresponding labels by calling predict() method of the convolution base without Data Augmentation

```
import numpy as np
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)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels = get_features_and_labels(train_datset)
val_features, val_labels = get_features_and_labels(validation_datset)
test_features, test_labels = get_features_and_labels(test_datset)
```

```
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
```

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
```

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"])
```

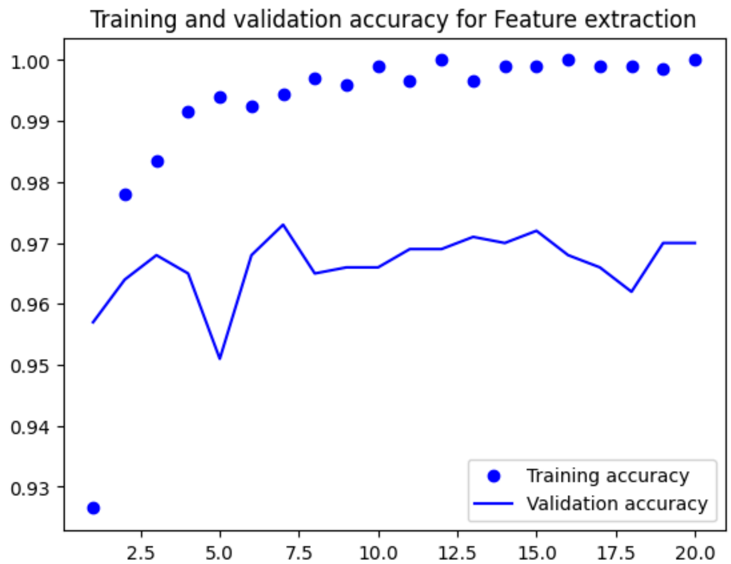
```
callbacks = [
 keras.callbacks.ModelCheckpoint(
 filepath="feature_extraction.keras",
 save_best_only=True,
 monitor="val_loss")
]
history = model.fit(
 train_features, train_labels,
 epochs=20,
 validation_data=(val_features, val_labels),
 callbacks=callbacks)
```

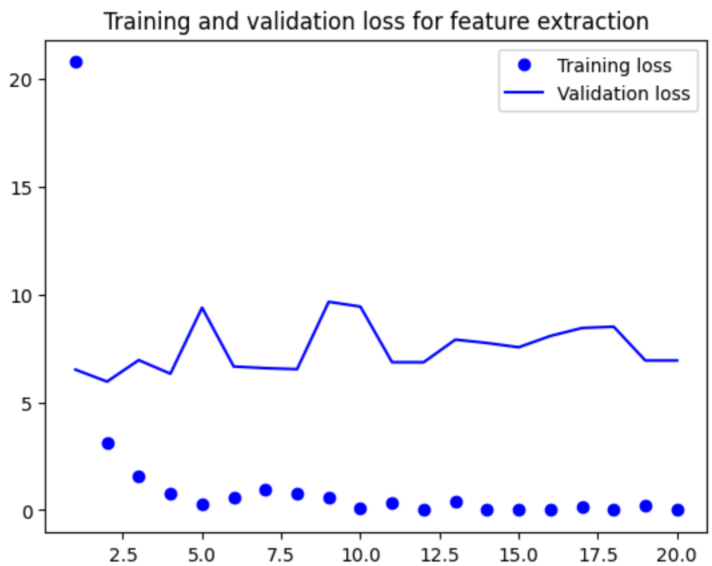
```
Epoch 1/20
63/63 [==============================] - 1s 10ms/step - loss: 20.7716 - accuracy: 0.9265 - val_loss: 6.5151 - val_accuracy: 0.9570
Epoch 2/20
63/63 [==============================] - 0s 7ms/step - loss: 3.1361 - accuracy: 0.9780 - val_loss: 5.9588 - val_accuracy: 0.9640
Epoch 3/20
63/63 [==============================] - 1s 8ms/step - loss: 1.5683 - accuracy: 0.9835 - val_loss: 6.9548 - val_accuracy: 0.9680
Epoch 4/20
63/63 [==============================] - 0s 8ms/step - loss: 0.7939 - accuracy: 0.9915 - val_loss: 6.3243 - val_accuracy: 0.9650
Epoch 5/20
63/63 [==============================] - 0s 8ms/step - loss: 0.2789 - accuracy: 0.9940 - val_loss: 9.3879 - val_accuracy: 0.9510
Epoch 6/20
63/63 [==============================] - 1s 12ms/step - loss: 0.5762 - accuracy: 0.9925 - val_loss: 6.6592 - val_accuracy: 0.9680
Epoch 7/20
63/63 [==============================] - 1s 10ms/step - loss: 0.9676 - accuracy: 0.9945 - val_loss: 6.5832 - val_accuracy: 0.9730
Epoch 8/20
63/63 [==============================] - 1s 11ms/step - loss: 0.7521 - accuracy: 0.9970 - val_loss: 6.5349 - val_accuracy: 0.9650
Epoch 9/20
63/63 [==============================] - 1s 14ms/step - loss: 0.5552 - accuracy: 0.9960 - val_loss: 9.6579 - val_accuracy: 0.9660
Epoch 10/20
63/63 [==============================] - 0s 8ms/step - loss: 0.1100 - accuracy: 0.9990 - val_loss: 9.4365 - val_accuracy: 0.9660
Epoch 11/20
63/63 [==============================] - 0s 8ms/step - loss: 0.3276 - accuracy: 0.9965 - val_loss: 6.8576 - val_accuracy: 0.9690
Epoch 12/20
63/63 [==============================] - 0s 8ms/step - loss: 7.4806e-11 - accuracy: 1.0000 - val_loss: 6.8576 - val_accuracy: 0.9690
Epoch 13/20
```

```
63/63 [==============================] - 1s 8ms/step - loss: 0.4075 - accuracy: 0.9965 - val_loss: 7.9060 - val_accuracy: 0.9710
Epoch 14/20
63/63 [==============================] - 1s 9ms/step - loss: 0.0547 - accuracy: 0.9990 - val_loss: 7.7495 - val_accuracy: 0.9700
Epoch 15/20
63/63 [==============================] - 0s 8ms/step - loss: 0.0370 - accuracy: 0.9990 - val_loss: 7.5507 - val_accuracy: 0.9720
Epoch 16/20
63/63 [==============================] - 1s 8ms/step - loss: 6.8109e-05 - accuracy: 1.0000 - val_loss: 8.0750 - val_accuracy: 0.9680
Epoch 17/20
63/63 [==============================] - 1s 9ms/step - loss: 0.1381 - accuracy: 0.9990 - val_loss: 8.4446 - val_accuracy: 0.9660
Epoch 18/20
63/63 [==============================] - 1s 8ms/step - loss: 0.0091 - accuracy: 0.9990 - val_loss: 8.5057 - val_accuracy: 0.9620
Epoch 19/20
63/63 [==============================] - 0s 7ms/step - loss: 0.2090 - accuracy: 0.9985 - val_loss: 6.9366 - val_accuracy: 0.9700
Epoch 20/20
63/63 [==============================] - 0s 7ms/step - loss: 3.4679e-24 - accuracy: 1.0000 - val_loss: 6.9366 - val_accuracy: 0.9700
```

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)
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "b", label="Validation accuracy")
plt.title("Training and validation accuracy for Feature extraction")
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 for feature extraction")
plt.legend()
plt.show()
```






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

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_16 (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)
_________________________________________________________________
```

Freezing all layers

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

Fine tuning a model

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.3),
        layers.RandomZoom(0.5),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.layers.Lambda(lambda x: keras.applications.vgg16.preprocess_input(x))(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"])
```

In above code we use 'lambda' function make sure that preprocessing function is correctly applied with allowing serialization

Training the regularized network

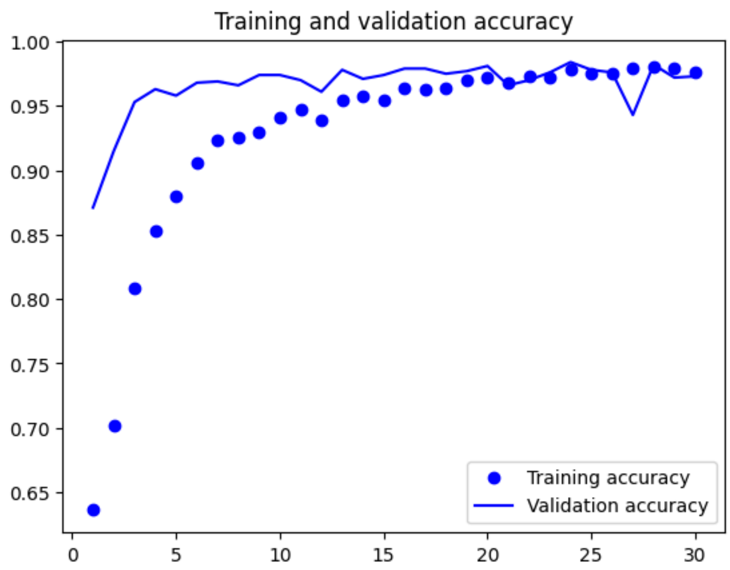
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_datset,
    epochs=20,
    validation_data=validation_datset,
    callbacks=callbacks)
```

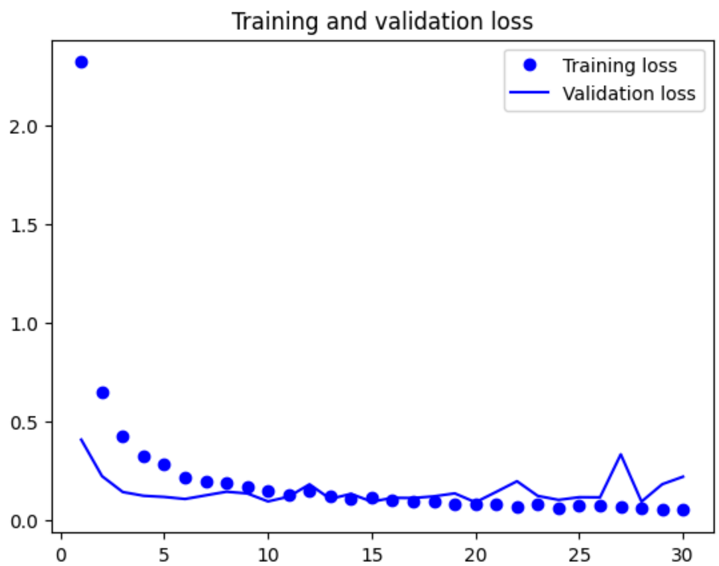
```
Epoch 1/20
63/63 [==============================] - 13s 141ms/step - loss: 7.0957 - accuracy: 0.6540 - val_loss: 1.7090 - val_accuracy: 0.8900
Epoch 2/20
63/63 [==============================] - 10s 148ms/step - loss: 4.6842 - accuracy: 0.7475 - val_loss: 1.2454 - val_accuracy: 0.9140
Epoch 3/20
63/63 [==============================] - 9s 147ms/step - loss: 3.4183 - accuracy: 0.7955 - val_loss: 0.7474 - val_accuracy: 0.9420
Epoch 4/20
```

```
63/63 [==============================] - 10s 156ms/step - loss: 3.2694 - accuracy: 0.8110 - val_loss: 0.6005 - val_accuracy: 0.9570
Epoch 5/20
63/63 [==============================] - 11s 158ms/step - loss: 2.9622 - accuracy: 0.8305 - val_loss: 0.5358 - val_accuracy: 0.9620
Epoch 6/20
63/63 [==============================] - 12s 178ms/step - loss: 2.6729 - accuracy: 0.8395 - val_loss: 0.5813 - val_accuracy: 0.9620
Epoch 7/20
63/63 [==============================] - 11s 164ms/step - loss: 2.5039 - accuracy: 0.8545 - val_loss: 0.5217 - val_accuracy: 0.9660
Epoch 8/20
63/63 [==============================] - 13s 191ms/step - loss: 2.2743 - accuracy: 0.8615 - val_loss: 0.5001 - val_accuracy: 0.9660
Epoch 9/20
63/63 [==============================] - 12s 179ms/step - loss: 1.9923 - accuracy: 0.8735 - val_loss: 0.5512 - val_accuracy: 0.9650
Epoch 10/20
63/63 [==============================] - 10s 147ms/step - loss: 1.9921 - accuracy: 0.8830 - val_loss: 0.5310 - val_accuracy: 0.9680
Epoch 11/20
63/63 [==============================] - 11s 163ms/step - loss: 2.1710 - accuracy: 0.8795 - val_loss: 0.5319 - val_accuracy: 0.9670
Epoch 12/20
63/63 [==============================] - 12s 177ms/step - loss: 2.0032 - accuracy: 0.8840 - val_loss: 0.5363 - val_accuracy: 0.9670
Epoch 13/20
63/63 [==============================] - 12s 182ms/step - loss: 1.8925 - accuracy: 0.8875 - val_loss: 0.5032 - val_accuracy: 0.9690
Epoch 14/20
63/63 [==============================] - 10s 152ms/step - loss: 1.8080 - accuracy: 0.8890 - val_loss: 0.5225 - val_accuracy: 0.9680
Epoch 15/20
63/63 [==============================] - 10s 158ms/step - loss: 1.9790 - accuracy: 0.8905 - val_loss: 0.5119 - val_accuracy: 0.9660
Epoch 16/20
63/63 [==============================] - 11s 176ms/step - loss: 2.0117 - accuracy: 0.8915 - val_loss: 0.5524 - val_accuracy: 0.9670
Epoch 17/20
63/63 [==============================] - 10s 145ms/step - loss: 1.9053 - accuracy: 0.8870 - val_loss: 0.5368 - val_accuracy: 0.9710
Epoch 18/20
63/63 [==============================] - 9s 140ms/step - loss: 1.7374 - accuracy: 0.8895 - val_loss: 0.5544 - val_accuracy: 0.9680
Epoch 19/20
63/63 [==============================] - 9s 145ms/step - loss: 1.9448 - accuracy: 0.8900 - val_loss: 0.6539 - val_accuracy: 0.9660
Epoch 20/20
63/63 [==============================] - 9s 145ms/step - loss: 1.7414 - accuracy: 0.8890 - val_loss: 0.5746 - val_accuracy: 0.9730
```

Plotting the curves of loss and accuracy during training for fine-tuning model

```
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 test set for fine-tuning

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

```
32/32 [==============================] - 3s 87ms/step - loss: 0.8877 - accuracy: 0.9570
Test accuracy: 0.957
```

'safe_mode'= False indicates that we can load the model successfully

Pre-trianed model-8000 Training samples

same as we did above by install and freezing the VGG16 conv base.

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

Fine tuning the pretrained model by freezing the layers

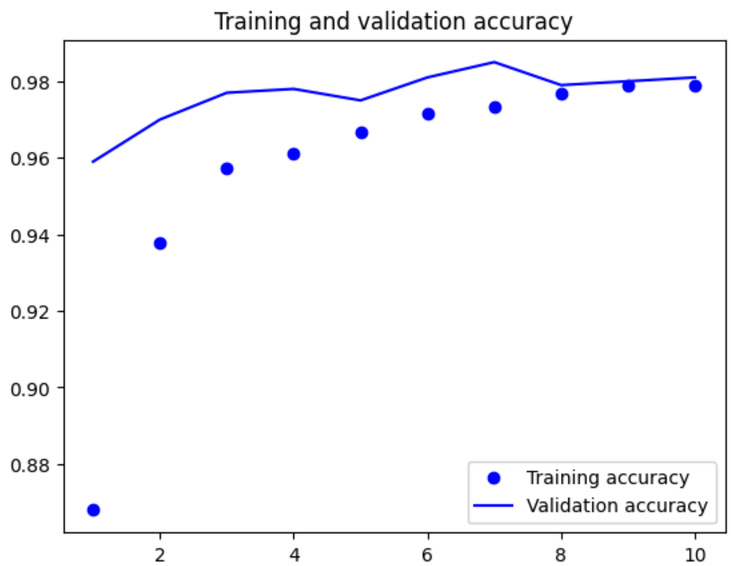
```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)

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

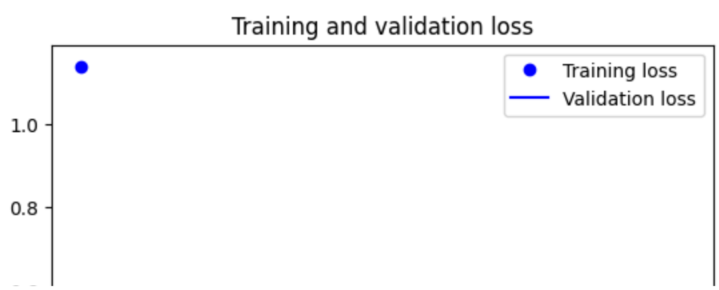
By adding of augmentation and classifier to conv base

```
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.layers.Lambda(lambda x: keras.applications.vgg16.preprocess_input(x))(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_2,
    epochs=10,
    validation_data=validation_datset,
    callbacks=callbacks)
```

```
Epoch 1/10
438/438 [==============================] - 61s 120ms/step - loss: 1.1343 - accuracy: 0.8679 - val_loss: 0.1446 - val_accuracy: 0.9590
Epoch 2/10
438/438 [==============================] - 53s 120ms/step - loss: 0.1742 - accuracy: 0.9376 - val_loss: 0.1231 - val_accuracy: 0.9700
Epoch 3/10
438/438 [==============================] - 53s 120ms/step - loss: 0.1243 - accuracy: 0.9574 - val_loss: 0.1230 - val_accuracy: 0.9770
Epoch 4/10
438/438 [==============================] - 55s 124ms/step - loss: 0.1156 - accuracy: 0.9611 - val_loss: 0.1010 - val_accuracy: 0.9780
Epoch 5/10
438/438 [==============================] - 54s 122ms/step - loss: 0.0915 - accuracy: 0.9669 - val_loss: 0.1190 - val_accuracy: 0.9750
Epoch 6/10
438/438 [==============================] - 52s 118ms/step - loss: 0.0902 - accuracy: 0.9715 - val_loss: 0.1281 - val_accuracy: 0.9810
Epoch 7/10
438/438 [==============================] - 54s 123ms/step - loss: 0.0762 - accuracy: 0.9734 - val_loss: 0.1525 - val_accuracy: 0.9850
Epoch 8/10
438/438 [==============================] - 53s 120ms/step - loss: 0.0758 - accuracy: 0.9767 - val_loss: 0.1617 - val_accuracy: 0.9790
Epoch 9/10
438/438 [==============================] - 51s 116ms/step - loss: 0.0694 - accuracy: 0.9789 - val_loss: 0.1413 - val_accuracy: 0.9800
Epoch 10/10
438/438 [==============================] - 51s 116ms/step - loss: 0.0621 - accuracy: 0.9788 - val_loss: 0.1374 - val_accuracy: 0.9810
```

```
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 with test set

```
model = keras.models.load_model("fine_tuning2.keras", safe_mode = False)
```