aml-group-8-assignment-2

March 31, 2024

[ ]:

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

!mkdir ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

!kaggle competitions download -c dogs-vs-cats

!unzip -qq dogs-vs-cats.zip

!unzip -qq train.zip

[ ]:

<IPython.core.display.HTML object>

Saving kaggle.json to kaggle.json Downloading dogs-vs-cats.zip to /content 99% 805M/812M [00:04<00:00, 187MB/s]

100% 812M/812M [00:04<00:00, 194MB/s]

**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)

[ ]:

**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.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 D) | (None, | 89, | 89, | 32) | 0 |
| conv2d\_1 (Conv2D) | (None, | 87, | 87, | 64) | 18496 |
| max\_pooling2d\_1 (MaxPoolin g2D) | (None, | 43, | 43, | 64) | 0 |
| conv2d\_2 (Conv2D) | (None, | 41, | 41, | 128) | 73856 |
| max\_pooling2d\_2 (MaxPoolin g2D) | (None, | 20, | 20, | 128) | 0 |
| conv2d\_3 (Conv2D) | (None, | 18, | 18, | 256) | 295168 |
| max\_pooling2d\_3 (MaxPoolin g2D) | (None, | 9, 9, 256) | | | 0 |
| 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)

[ ]:

[ ]:

model.compile(optimizer = "rmsprop", loss = "binary\_crossentropy", metrics =␣

𝗌["accuracy"])

*#image\_dataset\_from\_directory is to setup a data pipeline that can*␣

𝗌*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*␣

𝗌*each one contains images from one of our classes.*

*#it will create and return tf.data.Dataset that inturns read, shuffle, and*␣

𝗌*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.

data\_batch\_shape: (32, 180, 180, 3)

**for** data\_batch, labels\_batch **in** train\_datset: print("data\_batch\_shape:", data\_batch.shape) print("labels\_batch\_shape:", labels\_batch.shape) **break**

labels\_batch\_shape: (32,)

[ ]:

callbacks = [

keras.callbacks.ModelCheckpoint( filepath="conv\_from\_scratch1.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 [==============================] - 207s 3s/step - loss: 0.7035 - accuracy:

0.4990 - val\_loss: 0.6922 - val\_accuracy: 0.5100 Epoch 2/20

63/63 [==============================] - 207s 3s/step - loss: 0.6959 - accuracy:

0.5540 - val\_loss: 0.6852 - val\_accuracy: 0.5510 Epoch 3/20

63/63 [==============================] - 204s 3s/step - loss: 0.6779 - accuracy:

0.5730 - val\_loss: 0.6550 - val\_accuracy: 0.6390 Epoch 4/20

63/63 [==============================] - 205s 3s/step - loss: 0.6604 - accuracy:

0.6125 - val\_loss: 0.7214 - val\_accuracy: 0.5640 Epoch 5/20

63/63 [==============================] - 191s 3s/step - loss: 0.6426 - accuracy:

0.6395 - val\_loss: 0.6729 - val\_accuracy: 0.5960 Epoch 6/20

63/63 [==============================] - 242s 4s/step - loss: 0.6230 - accuracy:

0.6615 - val\_loss: 0.6678 - val\_accuracy: 0.5700 Epoch 7/20

63/63 [==============================] - 206s 3s/step - loss: 0.5911 - accuracy:

0.6915 - val\_loss: 0.5860 - val\_accuracy: 0.7110 Epoch 8/20

63/63 [==============================] - 201s 3s/step - loss: 0.5649 - accuracy:

0.7290 - val\_loss: 0.6276 - val\_accuracy: 0.6530 Epoch 9/20

63/63 [==============================] - 196s 3s/step - loss: 0.5377 - accuracy:

0.7345 - val\_loss: 0.5605 - val\_accuracy: 0.7200 Epoch 10/20

63/63 [==============================] - 210s 3s/step - loss: 0.5048 - accuracy:

0.7550 - val\_loss: 0.5985 - val\_accuracy: 0.7000 Epoch 11/20

63/63 [==============================] - 217s 3s/step - loss: 0.4524 - accuracy:

0.7895 - val\_loss: 0.6128 - val\_accuracy: 0.7140 Epoch 12/20

[ ]:

63/63 [==============================] - 220s 4s/step - loss: 0.4054 - accuracy:

0.8110 - val\_loss: 0.7505 - val\_accuracy: 0.6970 Epoch 13/20

63/63 [==============================] - 218s 3s/step - loss: 0.3419 - accuracy:

0.8530 - val\_loss: 0.8088 - val\_accuracy: 0.7130 Epoch 14/20

63/63 [==============================] - 220s 4s/step - loss: 0.2903 - accuracy:

0.8660 - val\_loss: 0.7583 - val\_accuracy: 0.7340 Epoch 15/20

63/63 [==============================] - 210s 3s/step - loss: 0.2375 - accuracy:

0.8995 - val\_loss: 1.1255 - val\_accuracy: 0.6750 Epoch 16/20

63/63 [==============================] - 220s 3s/step - loss: 0.1763 - accuracy:

0.9275 - val\_loss: 1.2931 - val\_accuracy: 0.6810 Epoch 17/20

63/63 [==============================] - 229s 4s/step - loss: 0.1536 - accuracy:

0.9435 - val\_loss: 1.1366 - val\_accuracy: 0.7270 Epoch 18/20

63/63 [==============================] - 209s 3s/step - loss: 0.1202 - accuracy:

0.9580 - val\_loss: 1.4912 - val\_accuracy: 0.7080 Epoch 19/20

63/63 [==============================] - 201s 3s/step - loss: 0.0944 - accuracy:

0.9640 - val\_loss: 1.4747 - val\_accuracy: 0.7160 Epoch 20/20

63/63 [==============================] - 211s 3s/step - loss: 0.0797 - accuracy:

0.9695 - val\_loss: 1.5506 - val\_accuracy: 0.7050

# 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**

accuracy1 = history.history["accuracy"] val\_accuracy1 = history.history["val\_accuracy"] loss1 = history.history["loss"]

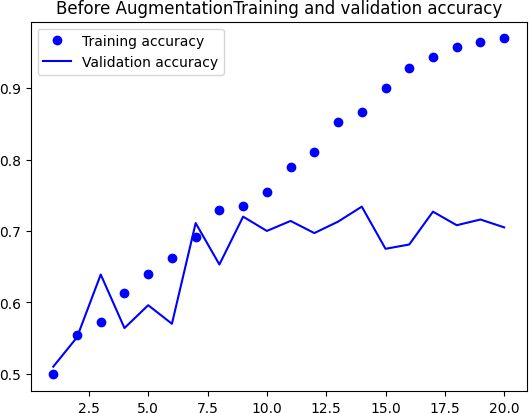
val\_loss1 = history.history["val\_loss"] epochs = range(1, len(accuracy1) + 1)

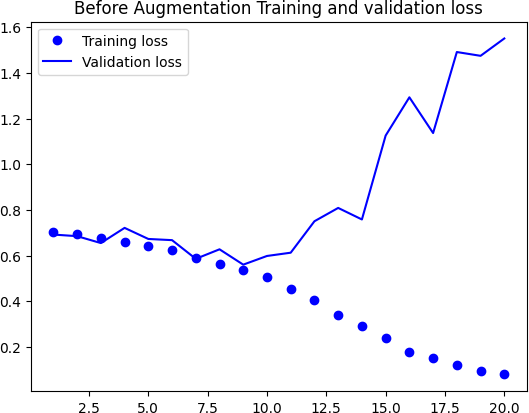
plt.plot(epochs, accuracy1, "bo", label="Training accuracy") plt.plot(epochs, val\_accuracy1, "b", label="Validation accuracy") plt.title("Before AugmentationTraining and validation accuracy") plt.legend()

plt.figure()

plt.plot(epochs, loss1, "bo", label="Training loss") plt.plot(epochs, val\_loss1, "b", label="Validation loss") plt.title("Before Augmentation Training and validation loss") plt.legend()

plt.show()





[ ]:

The overfitting qualities may be seen in the preceding plots, where validation accuracy is barely at 75% and training accuracy rises linearly over time to almost 100%. Additionally, after increasing rapidly for up to ten epochs, the validation loss stalls, whereas the training loss continues to decrease linearly as training goes on.

# 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 33ms/step - loss: 0.5411 - accuracy:

0.7330

Test accuracy: 0.733

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

To add an image model, define a data augmentation stage

[ ]:

*#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=20, validation\_data=validation\_datset, callbacks=callbacks)

Epoch 1/20

63/63 [==============================] - 8s 83ms/step - loss: 0.7123 - accuracy:

0.5000 - val\_loss: 0.6921 - val\_accuracy: 0.5000 Epoch 2/20

63/63 [==============================] - 4s 59ms/step - loss: 0.6882 - accuracy:

0.5565 - val\_loss: 0.6748 - val\_accuracy: 0.5940 Epoch 3/20

63/63 [==============================] - 4s 60ms/step - loss: 0.6779 - accuracy:

0.5980 - val\_loss: 0.6638 - val\_accuracy: 0.6420 Epoch 4/20

63/63 [==============================] - 7s 106ms/step - loss: 0.6644 -

accuracy: 0.6210 - val\_loss: 0.6529 - val\_accuracy: 0.6430 Epoch 5/20

63/63 [==============================] - 4s 58ms/step - loss: 0.6586 - accuracy:

0.6160 - val\_loss: 0.8199 - val\_accuracy: 0.5080 Epoch 6/20

63/63 [==============================] - 6s 84ms/step - loss: 0.6383 - accuracy:

0.6520 - val\_loss: 0.6372 - val\_accuracy: 0.6550 Epoch 7/20

63/63 [==============================] - 4s 58ms/step - loss: 0.6209 - accuracy:

0.6625 - val\_loss: 0.5997 - val\_accuracy: 0.6910 Epoch 8/20

63/63 [==============================] - 4s 63ms/step - loss: 0.5939 - accuracy:

0.6890 - val\_loss: 0.6702 - val\_accuracy: 0.6400 Epoch 9/20

63/63 [==============================] - 7s 108ms/step - loss: 0.5932 -

accuracy: 0.6955 - val\_loss: 0.6321 - val\_accuracy: 0.6540 Epoch 10/20

63/63 [==============================] - 4s 58ms/step - loss: 0.5735 - accuracy:

0.7065 - val\_loss: 0.7696 - val\_accuracy: 0.5620 Epoch 11/20

63/63 [==============================] - 4s 62ms/step - loss: 0.5695 - accuracy:

0.6970 - val\_loss: 0.5502 - val\_accuracy: 0.7350 Epoch 12/20

63/63 [==============================] - 6s 94ms/step - loss: 0.5578 - accuracy:

0.7080 - val\_loss: 0.5627 - val\_accuracy: 0.7350 Epoch 13/20

63/63 [==============================] - 4s 60ms/step - loss: 0.5453 - accuracy:

0.7330 - val\_loss: 0.5130 - val\_accuracy: 0.7550 Epoch 14/20

63/63 [==============================] - 6s 86ms/step - loss: 0.5432 - accuracy:

0.7295 - val\_loss: 0.6262 - val\_accuracy: 0.6770 Epoch 15/20

[ ]:

63/63 [==============================] - 4s 59ms/step - loss: 0.5219 - accuracy:

0.7385 - val\_loss: 0.5040 - val\_accuracy: 0.7590 Epoch 16/20

63/63 [==============================] - 5s 74ms/step - loss: 0.5004 - accuracy:

0.7550 - val\_loss: 0.5202 - val\_accuracy: 0.7630 Epoch 17/20

63/63 [==============================] - 6s 86ms/step - loss: 0.5075 - accuracy:

0.7515 - val\_loss: 0.5105 - val\_accuracy: 0.7640 Epoch 18/20

63/63 [==============================] - 4s 63ms/step - loss: 0.4981 - accuracy:

0.7545 - val\_loss: 0.4977 - val\_accuracy: 0.7660 Epoch 19/20

63/63 [==============================] - 4s 58ms/step - loss: 0.4847 - accuracy:

0.7755 - val\_loss: 0.5324 - val\_accuracy: 0.7570 Epoch 20/20

63/63 [==============================] - 7s 102ms/step - loss: 0.4787 -

accuracy: 0.7715 - val\_loss: 0.5174 - val\_accuracy: 0.7680

# 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 [==============================] - 2s 29ms/step - loss: 0.4765 - accuracy:

0.7810

Test accuracy: 0.781

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=20,

validation\_data=validation\_datset, callbacks=callbacks)

Epoch 1/20

63/63 [==============================] - 6s 60ms/step - loss: 0.7177 - accuracy:

0.4905 - val\_loss: 0.6926 - val\_accuracy: 0.5000 Epoch 2/20

63/63 [==============================] - 5s 83ms/step - loss: 0.6972 - accuracy:

0.5185 - val\_loss: 0.7041 - val\_accuracy: 0.5000 Epoch 3/20

63/63 [==============================] - 4s 58ms/step - loss: 0.6962 - accuracy:

0.5380 - val\_loss: 0.6879 - val\_accuracy: 0.5640 Epoch 4/20

63/63 [==============================] - 6s 85ms/step - loss: 0.6759 - accuracy:

0.5670 - val\_loss: 0.6629 - val\_accuracy: 0.5850 Epoch 5/20

63/63 [==============================] - 5s 80ms/step - loss: 0.6722 - accuracy:

0.5955 - val\_loss: 0.6787 - val\_accuracy: 0.5760 Epoch 6/20

63/63 [==============================] - 4s 57ms/step - loss: 0.6502 - accuracy:

0.6200 - val\_loss: 0.6512 - val\_accuracy: 0.5910 Epoch 7/20

63/63 [==============================] - 4s 56ms/step - loss: 0.6349 - accuracy:

0.6320 - val\_loss: 0.6709 - val\_accuracy: 0.5840 Epoch 8/20

63/63 [==============================] - 6s 96ms/step - loss: 0.6090 - accuracy:

0.6690 - val\_loss: 0.6200 - val\_accuracy: 0.6570 Epoch 9/20

63/63 [==============================] - 5s 67ms/step - loss: 0.5830 - accuracy:

0.6885 - val\_loss: 0.5859 - val\_accuracy: 0.6980 Epoch 10/20

63/63 [==============================] - 4s 57ms/step - loss: 0.5563 - accuracy:

0.7180 - val\_loss: 0.7046 - val\_accuracy: 0.6620 Epoch 11/20

63/63 [==============================] - 4s 64ms/step - loss: 0.5215 - accuracy:

0.7355 - val\_loss: 0.5486 - val\_accuracy: 0.7160 Epoch 12/20

[ ]:

63/63 [==============================] - 6s 96ms/step - loss: 0.4872 - accuracy:

0.7625 - val\_loss: 0.6544 - val\_accuracy: 0.6960 Epoch 13/20

63/63 [==============================] - 4s 57ms/step - loss: 0.4651 - accuracy:

0.7910 - val\_loss: 0.8187 - val\_accuracy: 0.6060 Epoch 14/20

63/63 [==============================] - 4s 57ms/step - loss: 0.4347 - accuracy:

0.8105 - val\_loss: 0.5246 - val\_accuracy: 0.7380 Epoch 15/20

63/63 [==============================] - 5s 83ms/step - loss: 0.3883 - accuracy:

0.8245 - val\_loss: 0.6111 - val\_accuracy: 0.7220 Epoch 16/20

63/63 [==============================] - 4s 59ms/step - loss: 0.3552 - accuracy:

0.8435 - val\_loss: 0.5590 - val\_accuracy: 0.7350 Epoch 17/20

63/63 [==============================] - 5s 83ms/step - loss: 0.3028 - accuracy:

0.8770 - val\_loss: 0.7316 - val\_accuracy: 0.7410 Epoch 18/20

63/63 [==============================] - 6s 82ms/step - loss: 0.2647 - accuracy:

0.8925 - val\_loss: 0.6243 - val\_accuracy: 0.7350 Epoch 19/20

63/63 [==============================] - 4s 57ms/step - loss: 0.2329 - accuracy:

0.9065 - val\_loss: 0.6565 - val\_accuracy: 0.7430 Epoch 20/20

63/63 [==============================] - 7s 102ms/step - loss: 0.1893 -

accuracy: 0.9240 - val\_loss: 0.8561 - val\_accuracy: 0.7530

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 29ms/step - loss: 0.5081 - accuracy:

0.7570

Test accuracy: 0.757

# 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=20,

validation\_data=validation\_datset, callbacks=callbacks)

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch 1/20 |  | | |
| 63/63 [==============================] - 6s 67ms/step | - loss: | 0.6952 | - accuracy: |
| 0.5070 - val\_loss: 0.6934 - val\_accuracy: 0.5000 |  |  |  |
| Epoch 2/20 |  |  |  |
| 63/63 [==============================] - 6s 86ms/step | - loss: | 0.7000 | - accuracy: |
| 0.5250 - val\_loss: 0.6895 - val\_accuracy: 0.6000 |  |  |  |
| Epoch 3/20 |  |  |  |
| 63/63 [==============================] - 6s 82ms/step | - loss: | 0.7043 | - accuracy: |
| 0.5610 - val\_loss: 0.6792 - val\_accuracy: 0.5790 |  |  |  |
| Epoch 4/20 |  |  |  |
| 63/63 [==============================] - 4s 58ms/step | - loss: | 0.6857 | - accuracy: |
| 0.5790 - val\_loss: 0.6813 - val\_accuracy: 0.5540 |  |  |  |
| Epoch 5/20 |  |  |  |

63/63 [==============================] - 4s 59ms/step - loss: 0.6743 - accuracy:

0.5995 - val\_loss: 0.6409 - val\_accuracy: 0.6250 Epoch 6/20

63/63 [==============================] - 7s 95ms/step - loss: 0.6681 - accuracy:

0.6295 - val\_loss: 0.6607 - val\_accuracy: 0.5970 Epoch 7/20

63/63 [==============================] - 4s 60ms/step - loss: 0.6352 - accuracy:

0.6405 - val\_loss: 0.6290 - val\_accuracy: 0.6440 Epoch 8/20

63/63 [==============================] - 4s 59ms/step - loss: 0.6335 - accuracy:

0.6465 - val\_loss: 0.6054 - val\_accuracy: 0.6760 Epoch 9/20

63/63 [==============================] - 7s 105ms/step - loss: 0.6341 -

accuracy: 0.6655 - val\_loss: 0.6263 - val\_accuracy: 0.6580 Epoch 10/20

63/63 [==============================] - 4s 64ms/step - loss: 0.5924 - accuracy:

0.6810 - val\_loss: 0.5807 - val\_accuracy: 0.6930 Epoch 11/20

63/63 [==============================] - 4s 59ms/step - loss: 0.5930 - accuracy:

0.6915 - val\_loss: 0.6790 - val\_accuracy: 0.6430 Epoch 12/20

63/63 [==============================] - 6s 86ms/step - loss: 0.5920 - accuracy:

0.6995 - val\_loss: 0.7611 - val\_accuracy: 0.5970 Epoch 13/20

63/63 [==============================] - 4s 59ms/step - loss: 0.5745 - accuracy:

0.6905 - val\_loss: 0.5513 - val\_accuracy: 0.7220 Epoch 14/20

63/63 [==============================] - 4s 62ms/step - loss: 0.5892 - accuracy:

0.6995 - val\_loss: 0.5943 - val\_accuracy: 0.6770 Epoch 15/20

63/63 [==============================] - 10s 154ms/step - loss: 0.5601 -

accuracy: 0.7230 - val\_loss: 0.5842 - val\_accuracy: 0.7100 Epoch 16/20

63/63 [==============================] - 4s 60ms/step - loss: 0.5547 - accuracy:

0.7145 - val\_loss: 0.5345 - val\_accuracy: 0.7350 Epoch 17/20

63/63 [==============================] - 5s 83ms/step - loss: 0.5324 - accuracy:

0.7285 - val\_loss: 0.5537 - val\_accuracy: 0.7400 Epoch 18/20

63/63 [==============================] - 6s 86ms/step - loss: 0.5325 - accuracy:

0.7375 - val\_loss: 0.5490 - val\_accuracy: 0.7170 Epoch 19/20

63/63 [==============================] - 4s 63ms/step - loss: 0.5134 - accuracy:

0.7465 - val\_loss: 0.4904 - val\_accuracy: 0.7680 Epoch 20/20

63/63 [==============================] - 7s 100ms/step - loss: 0.5188 -

accuracy: 0.7500 - val\_loss: 0.5499 - val\_accuracy: 0.7460

# 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 [==============================] - 1s 29ms/step - loss: 0.4599 - accuracy:

0.7840

Test accuracy: 0.784

1. 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 5000 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("train 2", start\_index=1000, end\_index=8000)

train\_dataset\_2 = image\_dataset\_from\_directory( new\_dir / "train 2",

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=20,

validation\_data=validation\_datset, callbacks=callbacks)

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch 1/20 |  | | |
| 438/438 [==============================] - 129s 292ms/step | - loss: | 0.1675 | - |
| accuracy: 0.9354 - val\_loss: 0.0674 - val\_accuracy: 0.9800 |  |  |  |
| Epoch 2/20 |  |  |  |
| 438/438 [==============================] - 134s 304ms/step | - loss: | 0.1503 | - |
| accuracy: 0.9417 - val\_loss: 0.0588 - val\_accuracy: 0.9800 |  |  |  |
| Epoch 3/20 |  |  |  |
| 438/438 [==============================] - 133s 302ms/step | - loss: | 0.1334 | - |
| accuracy: 0.9493 - val\_loss: 0.0569 - val\_accuracy: 0.9770 |  |  |  |
| Epoch 4/20 |  |  |  |
| 438/438 [==============================] - 131s 299ms/step | - loss: | 0.1255 | - |
| accuracy: 0.9515 - val\_loss: 0.0590 - val\_accuracy: 0.9790 |  |  |  |
| Epoch 5/20 |  |  |  |
| 438/438 [==============================] - 132s 300ms/step | - loss: | 0.1201 | - |
| accuracy: 0.9561 - val\_loss: 0.3475 - val\_accuracy: 0.9320 |  |  |  |
| Epoch 6/20 |  |  |  |
| 438/438 [==============================] - 131s 300ms/step | - loss: | 0.1147 | - |
| accuracy: 0.9569 - val\_loss: 0.0686 - val\_accuracy: 0.9830 |  |  |  |
| Epoch 7/20 |  |  |  |
| 438/438 [==============================] - 132s 301ms/step | - loss: | 0.1196 | - |
| accuracy: 0.9564 - val\_loss: 0.0552 - val\_accuracy: 0.9820 |  |  |  |
| Epoch 8/20 |  |  |  |
| 438/438 [==============================] - 131s 298ms/step | - loss: | 0.1021 | - |
| accuracy: 0.9614 - val\_loss: 0.0977 - val\_accuracy: 0.9740 |  |  |  |
| Epoch 9/20 |  |  |  |
| 438/438 [==============================] - 131s 299ms/step | - loss: | 0.1086 | - |
| accuracy: 0.9618 - val\_loss: 0.0675 - val\_accuracy: 0.9770 |  |  |  |
| Epoch 10/20 |  |  |  |
| 438/438 [==============================] - 132s 300ms/step | - loss: | 0.1129 | - |
| accuracy: 0.9605 - val\_loss: 0.0555 - val\_accuracy: 0.9780 |  |  |  |
| Epoch 11/20 |  |  |  |
| 438/438 [==============================] - 132s 300ms/step | - loss: | 0.0967 | - |

[ ]:

accuracy: 0.9645 - val\_loss: 0.1096 - val\_accuracy: 0.9730 Epoch 12/20

438/438 [==============================] - 133s 304ms/step - loss: 0.0967 -

accuracy: 0.9659 - val\_loss: 0.0773 - val\_accuracy: 0.9800 Epoch 13/20

438/438 [==============================] - 131s 298ms/step - loss: 0.1057 -

accuracy: 0.9623 - val\_loss: 0.0588 - val\_accuracy: 0.9760 Epoch 14/20

438/438 [==============================] - 132s 300ms/step - loss: 0.0912 -

accuracy: 0.9686 - val\_loss: 0.0702 - val\_accuracy: 0.9730 Epoch 15/20

438/438 [==============================] - 133s 302ms/step - loss: 0.1007 -

accuracy: 0.9678 - val\_loss: 0.0809 - val\_accuracy: 0.9780 Epoch 16/20

438/438 [==============================] - 132s 301ms/step - loss: 0.0931 -

accuracy: 0.9685 - val\_loss: 0.0599 - val\_accuracy: 0.9780 Epoch 17/20

438/438 [==============================] - 132s 302ms/step - loss: 0.1056 -

accuracy: 0.9651 - val\_loss: 0.0645 - val\_accuracy: 0.9810 Epoch 18/20

438/438 [==============================] - 134s 306ms/step - loss: 0.0959 -

accuracy: 0.9679 - val\_loss: 0.0691 - val\_accuracy: 0.9710 Epoch 19/20

438/438 [==============================] - 133s 303ms/step - loss: 0.0976 -

accuracy: 0.9666 - val\_loss: 0.0550 - val\_accuracy: 0.9820 Epoch 20/20

438/438 [==============================] - 134s 306ms/step - loss: 0.0965 -

accuracy: 0.9681 - val\_loss: 0.0586 - val\_accuracy: 0.9850

test\_model= keras.models.load.model( "convent\_from\_scrath3.keras")

test\_loss, test\_acc=test\_model.evaluate(test\_dataset) printf(f"Test Accuracy: **{**test\_acc**:**.3f**}**")

[ ]:

1. 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\_dataset2 = image\_dataset\_from\_directory( new\_dir / "train\_3",

image\_size=(180, 180), batch\_size=32)

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=20,

validation\_data=validation\_datset, callbacks=callbacks)

Epoch 1/20

438/438 [==============================] - 25s 54ms/step - loss: 0.6584 -

accuracy: 0.5966 - val\_loss: 0.5972 - val\_accuracy: 0.7100 Epoch 2/20

438/438 [==============================] - 22s 50ms/step - loss: 0.5350 -

accuracy: 0.7309 - val\_loss: 0.5364 - val\_accuracy: 0.7410 Epoch 3/20

438/438 [==============================] - 22s 50ms/step - loss: 0.4547 -

accuracy: 0.7919 - val\_loss: 0.5353 - val\_accuracy: 0.7330

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch 4/20 |  | | |
| 438/438 [==============================] - 27s 61ms/step - | loss: | 0.3883 | - |
| accuracy: 0.8268 - val\_loss: 0.3925 - val\_accuracy: 0.8390 |  |  |  |
| Epoch 5/20 |  |  |  |
| 438/438 [==============================] - 22s 50ms/step - | loss: | 0.3225 | - |
| accuracy: 0.8598 - val\_loss: 0.3900 - val\_accuracy: 0.8320 |  |  |  |
| Epoch 6/20 |  |  |  |
| 438/438 [==============================] - 22s 49ms/step - | loss: | 0.2668 | - |
| accuracy: 0.8901 - val\_loss: 0.4118 - val\_accuracy: 0.8450 |  |  |  |
| Epoch 7/20 |  |  |  |
| 438/438 [==============================] - 22s 50ms/step - | loss: | 0.2155 | - |
| accuracy: 0.9109 - val\_loss: 0.3531 - val\_accuracy: 0.8780 |  |  |  |
| Epoch 8/20 |  |  |  |
| 438/438 [==============================] - 24s 54ms/step - | loss: | 0.1632 | - |
| accuracy: 0.9358 - val\_loss: 0.4064 - val\_accuracy: 0.8630 |  |  |  |
| Epoch 9/20 |  |  |  |
| 438/438 [==============================] - 25s 56ms/step - | loss: | 0.1296 | - |
| accuracy: 0.9486 - val\_loss: 0.4851 - val\_accuracy: 0.8550 |  |  |  |
| Epoch 10/20 |  |  |  |
| 438/438 [==============================] - 24s 54ms/step - | loss: | 0.1018 | - |
| accuracy: 0.9634 - val\_loss: 0.6415 - val\_accuracy: 0.8350 |  |  |  |
| Epoch 11/20 |  |  |  |
| 438/438 [==============================] - 28s 64ms/step - | loss: | 0.0834 | - |
| accuracy: 0.9684 - val\_loss: 0.5291 - val\_accuracy: 0.8690 |  |  |  |
| Epoch 12/20 |  |  |  |
| 438/438 [==============================] - 25s 58ms/step - | loss: | 0.0765 | - |
| accuracy: 0.9736 - val\_loss: 0.6307 - val\_accuracy: 0.8590 |  |  |  |
| Epoch 13/20 |  |  |  |
| 438/438 [==============================] - 28s 63ms/step - | loss: | 0.0705 | - |
| accuracy: 0.9759 - val\_loss: 0.5666 - val\_accuracy: 0.8670 |  |  |  |
| Epoch 14/20 |  |  |  |
| 438/438 [==============================] - 22s 50ms/step - | loss: | 0.0616 | - |
| accuracy: 0.9799 - val\_loss: 0.7903 - val\_accuracy: 0.8480 |  |  |  |
| Epoch 15/20 |  |  |  |
| 438/438 [==============================] - 28s 63ms/step - | loss: | 0.0597 | - |
| accuracy: 0.9805 - val\_loss: 0.6777 - val\_accuracy: 0.8720 |  |  |  |
| Epoch 16/20 |  |  |  |
| 438/438 [==============================] - 25s 57ms/step - | loss: | 0.0628 | - |
| accuracy: 0.9806 - val\_loss: 0.7201 - val\_accuracy: 0.8850 |  |  |  |
| Epoch 17/20 |  |  |  |
| 438/438 [==============================] - 24s 55ms/step - | loss: | 0.0625 | - |
| accuracy: 0.9805 - val\_loss: 0.8220 - val\_accuracy: 0.8900 |  |  |  |
| Epoch 18/20 |  |  |  |
| 438/438 [==============================] - 23s 52ms/step - | loss: | 0.0619 | - |
| accuracy: 0.9808 - val\_loss: 0.7733 - val\_accuracy: 0.8840 |  |  |  |
| Epoch 19/20 |  |  |  |
| 438/438 [==============================] - 22s 51ms/step - | loss: | 0.0532 | - |
| accuracy: 0.9850 - val\_loss: 0.9373 - val\_accuracy: 0.8770 |  |  |  |

[ ]:

Epoch 20/20

438/438 [==============================] - 23s 51ms/step - loss: 0.0585 -

accuracy: 0.9840 - val\_loss: 1.1149 - val\_accuracy: 0.8640

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 [==============================] - 3s 66ms/step - loss: 0.3261 - accuracy:

0.8740

Test accuracy: 0.874

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 [==============================] - 25s 54ms/step - | loss: | 0.6954 | - |
| accuracy: 0.5421 - val\_loss: 0.7436 - val\_accuracy: 0.5240 |  |  |  |
| Epoch 2/20 |  |  |  |
| 438/438 [==============================] - 26s 59ms/step - | loss: | 0.5879 | - |
| accuracy: 0.6922 - val\_loss: 0.5297 - val\_accuracy: 0.7560 |  |  |  |
| Epoch 3/20 |  |  |  |
| 438/438 [==============================] - 29s 67ms/step - | loss: | 0.4926 | - |
| accuracy: 0.7676 - val\_loss: 0.4792 - val\_accuracy: 0.7700 |  |  |  |
| Epoch 4/20 |  |  |  |
| 438/438 [==============================] - 23s 52ms/step - | loss: | 0.4227 | - |
| accuracy: 0.8101 - val\_loss: 0.4400 - val\_accuracy: 0.7940 |  |  |  |
| Epoch 5/20 |  |  |  |
| 438/438 [==============================] - 25s 56ms/step - | loss: | 0.3610 | - |
| accuracy: 0.8407 - val\_loss: 0.4546 - val\_accuracy: 0.8020 |  |  |  |
| Epoch 6/20 |  |  |  |
| 438/438 [==============================] - 23s 51ms/step - | loss: | 0.3122 | - |
| accuracy: 0.8685 - val\_loss: 0.2915 - val\_accuracy: 0.8770 |  |  |  |
| Epoch 7/20 |  |  |  |
| 438/438 [==============================] - 22s 51ms/step - | loss: | 0.2650 | - |
| accuracy: 0.8896 - val\_loss: 0.4001 - val\_accuracy: 0.8230 |  |  |  |
| Epoch 8/20 |  |  |  |
| 438/438 [==============================] - 23s 52ms/step - | loss: | 0.2243 | - |
| accuracy: 0.9072 - val\_loss: 0.3046 - val\_accuracy: 0.8870 |  |  |  |
| Epoch 9/20 |  |  |  |
| 438/438 [==============================] - 25s 58ms/step - | loss: | 0.1903 | - |
| accuracy: 0.9237 - val\_loss: 0.4764 - val\_accuracy: 0.8230 |  |  |  |
| Epoch 10/20 |  |  |  |
| 438/438 [==============================] - 22s 51ms/step - | loss: | 0.1612 | - |
| accuracy: 0.9372 - val\_loss: 0.3615 - val\_accuracy: 0.8810 |  |  |  |
| Epoch 11/20 |  |  |  |
| 438/438 [==============================] - 25s 56ms/step - | loss: | 0.1389 | - |
| accuracy: 0.9459 - val\_loss: 0.4245 - val\_accuracy: 0.8590 |  |  |  |
| Epoch 12/20 |  |  |  |
| 438/438 [==============================] - 22s 50ms/step - | loss: | 0.1236 | - |
| accuracy: 0.9518 - val\_loss: 0.3406 - val\_accuracy: 0.8860 |  |  |  |
| Epoch 13/20 |  |  |  |
| 438/438 [==============================] - 23s 52ms/step - | loss: | 0.1112 | - |
| accuracy: 0.9588 - val\_loss: 0.3176 - val\_accuracy: 0.8870 |  |  |  |
| Epoch 14/20 |  |  |  |
| 438/438 [==============================] - 22s 50ms/step - | loss: | 0.1047 | - |
| accuracy: 0.9612 - val\_loss: 0.4277 - val\_accuracy: 0.8930 |  |  |  |
| Epoch 15/20 |  |  |  |

[ ]:

438/438 [==============================] - 22s 50ms/step - loss: 0.0978 -

accuracy: 0.9650 - val\_loss: 0.5801 - val\_accuracy: 0.8860 Epoch 16/20

438/438 [==============================] - 24s 54ms/step - loss: 0.1002 -

accuracy: 0.9666 - val\_loss: 0.3523 - val\_accuracy: 0.8980 Epoch 17/20

438/438 [==============================] - 22s 50ms/step - loss: 0.0946 -

accuracy: 0.9684 - val\_loss: 0.5883 - val\_accuracy: 0.8890 Epoch 18/20

438/438 [==============================] - 24s 55ms/step - loss: 0.0925 -

accuracy: 0.9698 - val\_loss: 0.5656 - val\_accuracy: 0.8840 Epoch 19/20

438/438 [==============================] - 22s 49ms/step - loss: 0.0928 -

accuracy: 0.9702 - val\_loss: 0.6549 - val\_accuracy: 0.8770 Epoch 20/20

438/438 [==============================] - 22s 49ms/step - loss: 0.0899 -

accuracy: 0.9718 - val\_loss: 0.6822 - val\_accuracy: 0.8880

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 31ms/step - loss: 0.2916 - accuracy:

0.8730

Test accuracy: 0.873

1. 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.h5 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)

Fine-tuning a pretrained model

[ ]:

conv\_base = keras.applications.vgg16.VGG16( weights="imagenet",

include\_top=**False**)

[ ]:

Freezing all layers except the last

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.

𝗌compile(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)

Epoch 1/50

63/63 [==============================] - 7s 68ms/step - loss: 0.7011 - accuracy:

0.4860 - val\_loss: 0.6929 - val\_accuracy: 0.5000 Epoch 2/50

63/63 [==============================] - 4s 60ms/step - loss: 0.6943 - accuracy:

0.5005 - val\_loss: 0.6928 - val\_accuracy: 0.5030 Epoch 3/50

63/63 [==============================] - 6s 94ms/step - loss: 0.6948 - accuracy:

0.5160 - val\_loss: 0.6916 - val\_accuracy: 0.5080 Epoch 4/50

63/63 [==============================] - 5s 70ms/step - loss: 0.6957 - accuracy:

0.5230 - val\_loss: 0.6867 - val\_accuracy: 0.5450 Epoch 5/50

63/63 [==============================] - 4s 61ms/step - loss: 0.6846 - accuracy:

0.5845 - val\_loss: 0.7106 - val\_accuracy: 0.5040 Epoch 6/50

63/63 [==============================] - 4s 62ms/step - loss: 0.6762 - accuracy:

0.5845 - val\_loss: 0.7215 - val\_accuracy: 0.5690 Epoch 7/50

63/63 [==============================] - 7s 98ms/step - loss: 0.6655 - accuracy:

0.6050 - val\_loss: 0.6518 - val\_accuracy: 0.6520 Epoch 8/50

63/63 [==============================] - 4s 60ms/step - loss: 0.6900 - accuracy:

0.6235 - val\_loss: 0.6275 - val\_accuracy: 0.6480 Epoch 9/50

63/63 [==============================] - 4s 60ms/step - loss: 0.6287 - accuracy:

0.6285 - val\_loss: 0.6347 - val\_accuracy: 0.6720 Epoch 10/50

63/63 [==============================] - 5s 78ms/step - loss: 0.6394 - accuracy:

0.6360 - val\_loss: 0.6503 - val\_accuracy: 0.6250 Epoch 11/50

63/63 [==============================] - 6s 81ms/step - loss: 0.6430 - accuracy:

0.6245 - val\_loss: 0.6832 - val\_accuracy: 0.5170 Epoch 12/50

63/63 [==============================] - 4s 58ms/step - loss: 0.6462 - accuracy:

0.6315 - val\_loss: 0.6115 - val\_accuracy: 0.6770 Epoch 13/50

63/63 [==============================] - 4s 60ms/step - loss: 0.6252 - accuracy:

0.6485 - val\_loss: 0.6279 - val\_accuracy: 0.6470 Epoch 14/50

63/63 [==============================] - 6s 91ms/step - loss: 0.6312 - accuracy:

0.6390 - val\_loss: 0.6399 - val\_accuracy: 0.6200 Epoch 15/50

63/63 [==============================] - 4s 58ms/step - loss: 0.6255 - accuracy:

0.6380 - val\_loss: 0.6081 - val\_accuracy: 0.6890

Epoch 16/50

63/63 [==============================] - 4s 59ms/step - loss: 0.6264 - accuracy:

0.6530 - val\_loss: 0.6029 - val\_accuracy: 0.6750 Epoch 17/50

63/63 [==============================] - 7s 111ms/step - loss: 0.6205 -

accuracy: 0.6625 - val\_loss: 0.6385 - val\_accuracy: 0.6250 Epoch 18/50

63/63 [==============================] - 4s 57ms/step - loss: 0.6126 - accuracy:

0.6530 - val\_loss: 0.5956 - val\_accuracy: 0.6920 Epoch 19/50

63/63 [==============================] - 4s 57ms/step - loss: 0.6082 - accuracy:

0.6580 - val\_loss: 0.5826 - val\_accuracy: 0.7080 Epoch 20/50

63/63 [==============================] - 5s 71ms/step - loss: 0.6088 - accuracy:

0.6705 - val\_loss: 0.5957 - val\_accuracy: 0.7040 Epoch 21/50

63/63 [==============================] - 5s 80ms/step - loss: 0.6039 - accuracy:

0.6800 - val\_loss: 0.6090 - val\_accuracy: 0.6780 Epoch 22/50

63/63 [==============================] - 4s 61ms/step - loss: 0.6080 - accuracy:

0.6680 - val\_loss: 0.5910 - val\_accuracy: 0.6810 Epoch 23/50

63/63 [==============================] - 6s 89ms/step - loss: 0.5986 - accuracy:

0.6815 - val\_loss: 0.5770 - val\_accuracy: 0.7080 Epoch 24/50

63/63 [==============================] - 4s 56ms/step - loss: 0.6026 - accuracy:

0.6720 - val\_loss: 0.5993 - val\_accuracy: 0.6730 Epoch 25/50

63/63 [==============================] - 4s 57ms/step - loss: 0.5843 - accuracy:

0.6810 - val\_loss: 0.6718 - val\_accuracy: 0.6190 Epoch 26/50

63/63 [==============================] - 6s 87ms/step - loss: 0.5859 - accuracy:

0.6845 - val\_loss: 0.5662 - val\_accuracy: 0.7320 Epoch 27/50

63/63 [==============================] - 5s 74ms/step - loss: 0.5837 - accuracy:

0.6845 - val\_loss: 0.6239 - val\_accuracy: 0.6640 Epoch 28/50

63/63 [==============================] - 4s 57ms/step - loss: 0.5861 - accuracy:

0.6880 - val\_loss: 0.6243 - val\_accuracy: 0.6530 Epoch 29/50

63/63 [==============================] - 4s 60ms/step - loss: 0.5863 - accuracy:

0.6940 - val\_loss: 0.6229 - val\_accuracy: 0.6820 Epoch 30/50

63/63 [==============================] - 7s 107ms/step - loss: 0.5868 -

accuracy: 0.6845 - val\_loss: 0.5698 - val\_accuracy: 0.7180 Epoch 31/50

63/63 [==============================] - 4s 60ms/step - loss: 0.5829 - accuracy:

0.6960 - val\_loss: 0.5786 - val\_accuracy: 0.6950

Epoch 32/50

63/63 [==============================] - 4s 58ms/step - loss: 0.5772 - accuracy:

0.6915 - val\_loss: 0.6261 - val\_accuracy: 0.6550 Epoch 33/50

63/63 [==============================] - 6s 94ms/step - loss: 0.5705 - accuracy:

0.6985 - val\_loss: 0.5872 - val\_accuracy: 0.6970 Epoch 34/50

63/63 [==============================] - 5s 67ms/step - loss: 0.5861 - accuracy:

0.6870 - val\_loss: 0.5748 - val\_accuracy: 0.7040 Epoch 35/50

63/63 [==============================] - 4s 61ms/step - loss: 0.5724 - accuracy:

0.6985 - val\_loss: 0.5538 - val\_accuracy: 0.7290 Epoch 36/50

63/63 [==============================] - 5s 84ms/step - loss: 0.5685 - accuracy:

0.7025 - val\_loss: 0.5744 - val\_accuracy: 0.7030 Epoch 37/50

63/63 [==============================] - 6s 82ms/step - loss: 0.5698 - accuracy:

0.7110 - val\_loss: 0.5814 - val\_accuracy: 0.6860 Epoch 38/50

63/63 [==============================] - 4s 59ms/step - loss: 0.5787 - accuracy:

0.6960 - val\_loss: 0.5577 - val\_accuracy: 0.7160 Epoch 39/50

63/63 [==============================] - 4s 59ms/step - loss: 0.5640 - accuracy:

0.7115 - val\_loss: 0.5778 - val\_accuracy: 0.7060 Epoch 40/50

63/63 [==============================] - 7s 101ms/step - loss: 0.5578 -

accuracy: 0.7165 - val\_loss: 0.5721 - val\_accuracy: 0.7040 Epoch 41/50

63/63 [==============================] - 4s 61ms/step - loss: 0.5525 - accuracy:

0.7145 - val\_loss: 0.5483 - val\_accuracy: 0.7050 Epoch 42/50

63/63 [==============================] - 4s 59ms/step - loss: 0.5574 - accuracy:

0.7215 - val\_loss: 0.5526 - val\_accuracy: 0.7330 Epoch 43/50

63/63 [==============================] - 5s 71ms/step - loss: 0.5681 - accuracy:

0.7035 - val\_loss: 0.5617 - val\_accuracy: 0.7310 Epoch 44/50

63/63 [==============================] - 6s 85ms/step - loss: 0.5486 - accuracy:

0.7190 - val\_loss: 0.6529 - val\_accuracy: 0.6880 Epoch 45/50

63/63 [==============================] - 4s 59ms/step - loss: 0.5600 - accuracy:

0.7165 - val\_loss: 0.5997 - val\_accuracy: 0.7150 Epoch 46/50

63/63 [==============================] - 4s 59ms/step - loss: 0.5536 - accuracy:

0.7155 - val\_loss: 0.5378 - val\_accuracy: 0.7420 Epoch 47/50

63/63 [==============================] - 7s 111ms/step - loss: 0.5464 -

accuracy: 0.7240 - val\_loss: 0.5575 - val\_accuracy: 0.7010

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Epoch 48/50 |  | | |
| 63/63 [==============================] - 4s 59ms/step | - loss: | 0.5602 | - accuracy: |
| 0.7125 - val\_loss: 0.6616 - val\_accuracy: 0.6670 |  |  |  |
| Epoch 49/50 |  |  |  |
| 63/63 [==============================] - 4s 58ms/step | - loss: | 0.5606 | - accuracy: |
| 0.7010 - val\_loss: 0.5458 - val\_accuracy: 0.7250 |  |  |  |
| Epoch 50/50 |  |  |  |
| 63/63 [==============================] - 6s 89ms/step | - loss: | 0.5349 | - accuracy: |
| 0.7275 - val\_loss: 0.5900 - val\_accuracy: 0.6790  Plotting the curves for loss and accuracy during training |  |  |  |
| [ | ]: |  |  |  |  |

Evaluating the model on the test set

**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()

[ ]:

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**}**")

Leveraging a Pretrained model

[ ]:

conv\_base = keras.applications.vgg16.VGG16( weights="imagenet",

include\_top=**False**, input\_shape=(180, 180, 3))

conv\_base.summary()

[ ]:

**import numpy as np**

**def** get\_features\_and\_labels(dataset): all\_features = []

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

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)

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",␣

𝗌metrics=["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)

Plotting the results

[ ]:

**import matplotlib.pyplot as plt**

acc = history.history["accuracy"]

val\_acc = history.history["val\_accuracy"] loss = history.history["loss"]

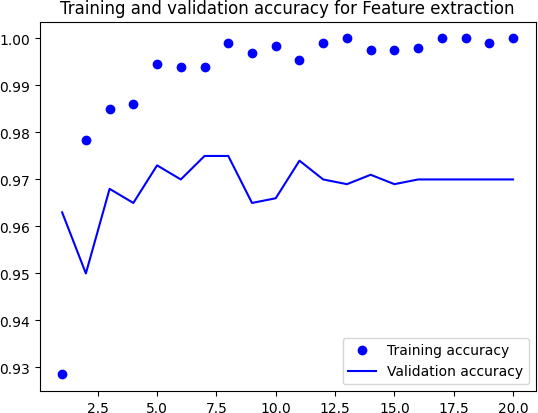
val\_loss = history.history["val\_loss"] epochs = range(1, len(acc) + 1)

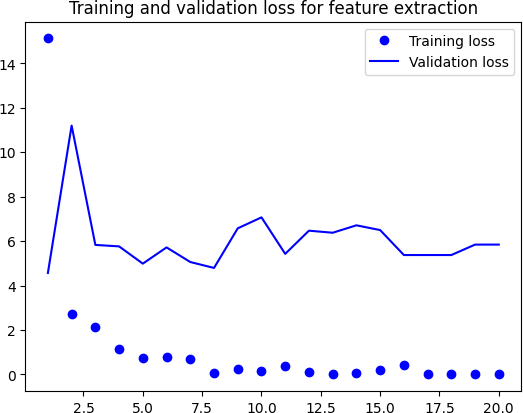
plt.plot(epochs, acc, "bo", label="Training accuracy") plt.plot(epochs, val\_acc, "b", label="Validation accuracy") plt.title("Training and validation accuracy for Feature extraction") plt.legend()

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()





Feature extraction with data augmentation

[ ]:

conv\_base = keras.applications.vgg16.VGG16(weights="imagenet",include\_top=**False**) conv\_base.trainable = **False**

[ ]:

[ ]:

conv\_base.summary()

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.

𝗌preprocess\_input(x))(x)

conv\_base = keras.applications.VGG16(weights="imagenet", include\_top=**False**,␣

𝗌input\_shape=(180, 180, 3)) 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"])

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 [==============================] - 41s 453ms/step - loss: | 2.3936 | - |
| accuracy: 0.5970 - val\_loss: 0.5101 - val\_accuracy: 0.7740 |  |  |
| Epoch 2/20 |  |  |
| 63/63 [==============================] - 23s 357ms/step - loss: | 0.6690 | - |
| accuracy: 0.6635 - val\_loss: 0.2966 - val\_accuracy: 0.8620 |  |  |
| Epoch 3/20 |  |  |
| 63/63 [==============================] - 23s 358ms/step - loss: | 0.5144 | - |
| accuracy: 0.7465 - val\_loss: 0.2505 - val\_accuracy: 0.8890 |  |  |
| Epoch 4/20 |  |  |
| 63/63 [==============================] - 23s 361ms/step - loss: | 0.4001 | - |
| accuracy: 0.8105 - val\_loss: 0.1462 - val\_accuracy: 0.9430 |  |  |
| Epoch 5/20 |  |  |
| 63/63 [==============================] - 22s 345ms/step - loss: | 0.3500 | - |
| accuracy: 0.8350 - val\_loss: 0.2113 - val\_accuracy: 0.9370 |  |  |
| Epoch 6/20 |  |  |

63/63 [==============================] - 23s 367ms/step - loss: 0.2978 -

accuracy: 0.8595 - val\_loss: 0.1065 - val\_accuracy: 0.9580 Epoch 7/20

63/63 [==============================] - 23s 357ms/step - loss: 0.2683 -

accuracy: 0.8730 - val\_loss: 0.1021 - val\_accuracy: 0.9640 Epoch 8/20

63/63 [==============================] - 22s 349ms/step - loss: 0.2318 -

accuracy: 0.8975 - val\_loss: 0.1425 - val\_accuracy: 0.9570 Epoch 9/20

63/63 [==============================] - 23s 364ms/step - loss: 0.2081 -

accuracy: 0.9090 - val\_loss: 0.0993 - val\_accuracy: 0.9640 Epoch 10/20

63/63 [==============================] - 23s 355ms/step - loss: 0.2194 -

accuracy: 0.9165 - val\_loss: 0.0929 - val\_accuracy: 0.9670 Epoch 11/20

63/63 [==============================] - 23s 360ms/step - loss: 0.1810 -

accuracy: 0.9270 - val\_loss: 0.0837 - val\_accuracy: 0.9800 Epoch 12/20

63/63 [==============================] - 22s 349ms/step - loss: 0.1828 -

accuracy: 0.9280 - val\_loss: 0.0839 - val\_accuracy: 0.9680 Epoch 13/20

63/63 [==============================] - 24s 382ms/step - loss: 0.1602 -

accuracy: 0.9350 - val\_loss: 0.0896 - val\_accuracy: 0.9700 Epoch 14/20

63/63 [==============================] - 22s 342ms/step - loss: 0.1623 -

accuracy: 0.9350 - val\_loss: 0.1312 - val\_accuracy: 0.9600 Epoch 15/20

63/63 [==============================] - 23s 356ms/step - loss: 0.1431 -

accuracy: 0.9470 - val\_loss: 0.0668 - val\_accuracy: 0.9700 Epoch 16/20

63/63 [==============================] - 21s 337ms/step - loss: 0.1562 -

accuracy: 0.9335 - val\_loss: 0.0723 - val\_accuracy: 0.9770 Epoch 17/20

63/63 [==============================] - 24s 373ms/step - loss: 0.1458 -

accuracy: 0.9395 - val\_loss: 0.2243 - val\_accuracy: 0.9460 Epoch 18/20

63/63 [==============================] - 21s 334ms/step - loss: 0.1056 -

accuracy: 0.9530 - val\_loss: 0.0687 - val\_accuracy: 0.9780 Epoch 19/20

63/63 [==============================] - 21s 335ms/step - loss: 0.1084 -

accuracy: 0.9575 - val\_loss: 0.1144 - val\_accuracy: 0.9700 Epoch 20/20

63/63 [==============================] - 22s 340ms/step - loss: 0.1124 -

accuracy: 0.9595 - val\_loss: 0.1622 - val\_accuracy: 0.9680

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") test\_loss, test\_acc = model.evaluate(test\_datset) print(f"Test accuracy: **{**test\_acc**:**.3f**}**")

[ ]:

Pre-trianed model-5000 Training samples

same as we did above by installing 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.applications.vgg16.preprocess\_input(x) x = conv\_base(x)

x = layers.Flatten()(x) x = layers.Dense(256)(x)

x = layers.Dropout(0.5)(x)

outputs = layers.Dense(1, activation="sigmoid")(x) model = keras.Model(inputs, outputs) model.compile(loss="binary\_crossentropy",

optimizer=keras.optimizers.RMSprop(learning\_rate=1e-5), metrics=["accuracy"])

callbacks = [

keras.callbacks.ModelCheckpoint( filepath="fine\_tuning2.keras", save\_best\_only=**True**, monitor="val\_loss")

]

history = model.fit( train\_dataset\_2, epochs=10,

validation\_data=validation\_datset, callbacks=callbacks)

evaluating the model with test set

[ ]:

model = keras.models.load\_model("fine\_tuning2.keras") test\_loss, test\_acc = model.evaluate(test\_datset) print(f"Test accuracy: **{**test\_acc**:**.3f**}**")

Pre-trained model with 10000 samples by install and freezing the VGG16 comv base

[ ]:

conv\_base = keras.applications.vgg16.VGG16( weights="imagenet",

include\_top=**False**, input\_shape=(180, 180, 3))

Fine tuning the pretrained model and freezing the layers except last one

[ ]:

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 augmentation and classifier to the 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.applications.vgg16.preprocess\_input(x) x = conv\_base(x)

x = layers.Flatten()(x) x = layers.Dense(256)(x)

x = layers.Dropout(0.5)(x)

outputs = layers.Dense(1, activation="sigmoid")(x) model = keras.Model(inputs, outputs) model.compile(loss="binary\_crossentropy",

optimizer=keras.optimizers.RMSprop(learning\_rate=1e-5), metrics=["accuracy"])

callbacks = [

keras.callbacks.ModelCheckpoint( filepath="fine\_tuning3.keras", save\_best\_only=**True**, monitor="val\_loss")

]

history = model.fit( train\_dataset2, epochs=30,

validation\_data=validation\_datset, callbacks=callbacks)

Plotting the curves for loss and accuracy during training for fine tuning with 10000 samples

[ ]:

**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\_tuning3.keras") test\_loss, test\_acc = model.evaluate(test\_datset) print(f"Test accuracy: **{**test\_acc**:**.3f**}**")

# Summary:

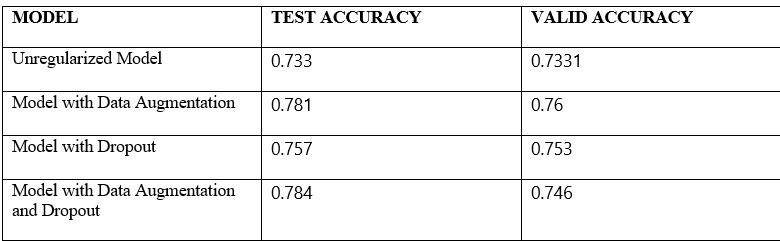
1. Consider the Cats & Dogs example. Start initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text). Use any technique to reduce overfitting and improve performance in developing a network that you train from scratch. What performance did you achieve?

# Observations:

* + The graphs shown above are examples of overfitting. Whereas validation accuracy only reaches 70–72%, training accuracy rises linearly over time to almost 100%. • Our primary problem will be overfitting because there aren’t many training data. A number of strategies, including dropout, regularization, and data augmentation, can be used to lessen overfitting.

I have used three techniques to improve the performance of the model and evaluated all those three on test dataset on 100 epochs.

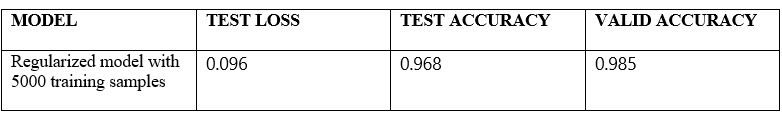
* + 1. Drop out Method
    2. Data Augmentation
    3. Data Augmentation and drop out method.



# Observations:

* + Based on the performance metrics of the models that combine the unregularized model with three performance improvement strategies, we can infer from the values in the above table that the model that incorporates both the dropout technique and data augmentation performs well. • To regularize the model, I utilized the best-performing method—data augmentation and dropout—for the remaining training samples.

1. 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?\*\*
   * For this model I have increased the training sample size to 5000.

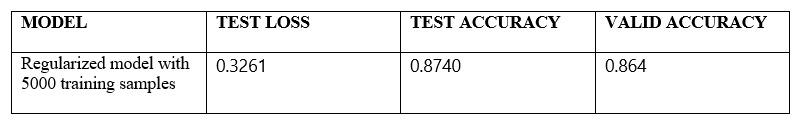


Observations:

•For regularized model it is observed that the loss: 0.3669 - accuracy: 0.848.

* + In contrast to the unregularized model regularized model seems to have a bit higher accuracy.
  + In comparison to the previous model the accuracy seems to be improved while the loss is slightly reduced.

1. Now change your training sample so that you achieve better performance than those from Steps1 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?
   * For this model,I have increased sample size to

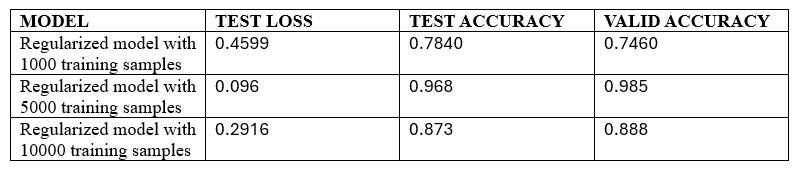


Observations:

From the above table,

* + For regularized model it is observed that the loss: 0.22 - accuracy: 0.912
  + In comparison with the unregularized model, this model is better.

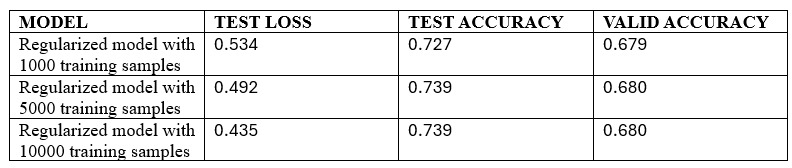
Below is the chart that describes the comparison of test and validation accuracies for the different training samples size.



# Observations:

* + There is a correlation between test loss and training sample size, which shows that test loss decreases over time and the test accuracy increases from 86% to 92.2%, which shows a better improvement over time.
  + Therefore, we can say that the performance of the model increases as the number of training samples increases.

# 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.

****

**Observations:**

We can observe from the data in the above table that when the training sample size rises, both the testing and validation accuracy tend to get better. As sample size grows, we observe a stronger improvement when test loss is taken into account.

# Recommendations:

* + Convolutional network-based machine learning models are the most successful in computer vision applications.
  + When starting from scratch and training from a relatively little dataset, the outcomes can still be respectable.
  + Overfitting is the fundamental issue with short datasets. Preventing overfitting in picture data can be effectively achieved by data augmentation techniques.
  + Model performance rises as the amount of the training sample grows.
  + We can improve the model’s performance even further by fine-tuning the previously trained model.