nn_tutorial

March 23, 2024

1 Tutorial: Neural Networks using Keras

In this tutorial, you will work with a neural network on a subset of the Toronto Faces Dataset (TFD). You will use the neural network to determine when any of the faces fall into one of seven categories:

1-Anger, 2-Disgust, 3-Fear, 4-Happy, 5-Sad, 6-Surprise, 7-Neutral.

The data contains 3374 faces, 419 and 385 grayscale images as the training, validation and testing set respectively. The faces have been rotated, scaled, and aligned to make the task easier. Each image is of size 48×48 and containes a face that has been extracted from a variety of sources.

An example is shown below:

1.1 Part 1: Load the Toronto Faces Dataset (TFD)

Firstly, we load the data. The dataset has been prepared for you and is a `.npz` file. Run the code chunk below to load it

[2]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
```

```
2024-03-23 23:19:39.808141: I tensorflow/core/util/port.cc:113] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-03-23 23:19:39.810884: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-03-23 23:19:39.841621: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-03-23 23:19:39.841654: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-03-23 23:19:39.842468: E
```

```
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-03-23 23:19:39.847624: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-03-23 23:19:39.848332: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2024-03-23 23:19:40.584927: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
/opt/conda/lib/python3.11/site-packages/h5py/__init__.py:36: UserWarning: h5py
is running against HDF5 1.14.3 when it was built against 1.14.2, this may cause
problems
  _warn(("h5py is running against HDF5 {0} when it was built against {1}, "
```

[3]:
```
! wget "https://github.com/manchuran/introToML/raw/master/toronto_face.npz"
```

```
--2024-03-23 23:19:42--
https://github.com/manchuran/introToML/raw/master/toronto_face.npz
Resolving github.com (github.com)…

/opt/conda/lib/python3.11/pty.py:89: RuntimeWarning: os.fork() was called.
os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so
this will likely lead to a deadlock.
  pid, fd = os.forkpty()

140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location:
https://raw.githubusercontent.com/manchuran/introToML/master/toronto_face.npz
[following]
--2024-03-23 23:19:42--
https://raw.githubusercontent.com/manchuran/introToML/master/toronto_face.npz
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.109.133, 185.199.108.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 12375556 (12M) [application/octet-stream]
Saving to: ‘toronto_face.npz.1’

toronto_face.npz.1  100%[===================>]  11.80M  --.-KB/s    in 0.02s

2024-03-23 23:19:42 (478 MB/s) - ‘toronto_face.npz.1’ saved [12375556/12375556]
```

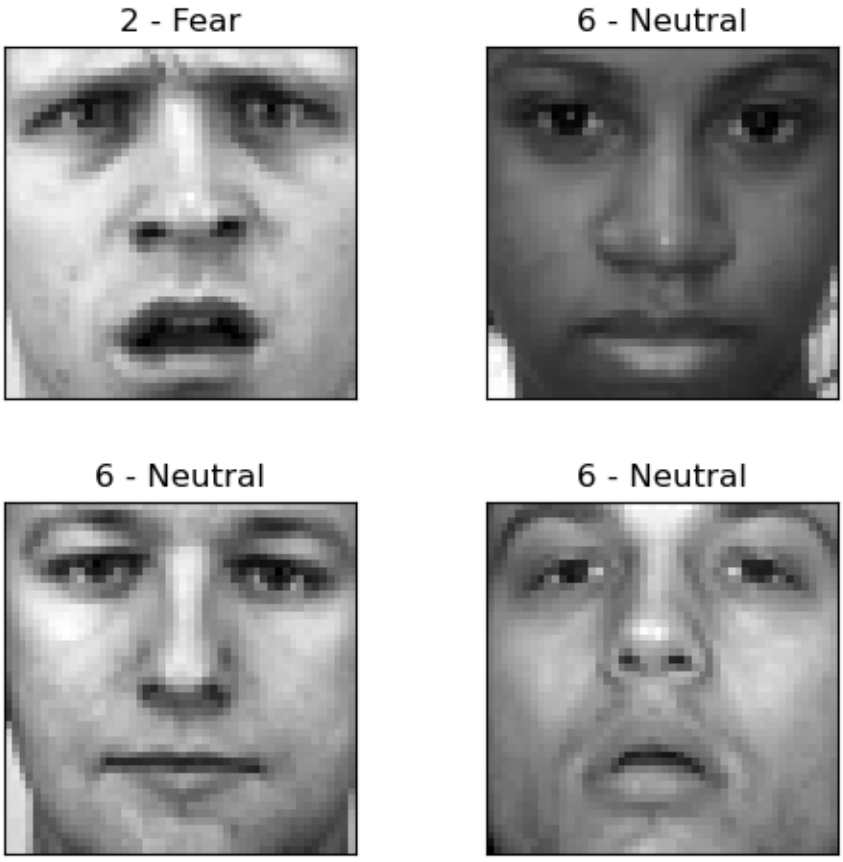
[4]:
```
## Use numpy to load in data and obtain train, test, and validation sets

with np.load("toronto_face.npz") as npzfile:
  inputs_train = npzfile["inputs_train"] / 255.0 #normalize input data
  inputs_valid = npzfile["inputs_valid"] / 255.0
  inputs_test = npzfile["inputs_test"] / 255.0
  target_train = npzfile["target_train"]
  target_valid = npzfile["target_valid"]
  target_test = npzfile["target_test"]
```

[5]:
```
# Prepare labels
# Note that to confirm with Python's 0-indexing, we have relabeled Anger to 0,␣
 ↪Digust to 1, etc.

target_labels = {
    0: "Anger",
    1: "Disgust",
    2: "Fear",
    3: "Happy",
    4: "Sad",
    5: "Surprise",
    6: "Neutral"
}
```

Display a few random samples from the training set. The following code creates a 2x2 grid of subplots, each displaying an image from the inputs_train dataset.

[6]:
```
# Set the desired image size (48x48 pixels)
img_size = 48

# Create a 2x2 grid of subplots with a specified figure size
fig, ax = plt.subplots(2, 2, figsize=(7, 6))

# Iterate over the 4 subplots
for i in range(4):
    # Randomly select an index from the training data
    s = np.random.choice(range(inputs_train.shape[0]))

    # Display the image at index 's' after reshaping it to the desired size
    ax[i // 2, i % 2].imshow(inputs_train[s, :].reshape((img_size, img_size)),␣
 ↪cmap="gray")

    # Set the title of the subplot with the corresponding emotion label
    ax[i // 2, i % 2].set_title(f"{target_train[s]} -␣
 ↪{target_labels[target_train[s]]}")

    # Hide x-axis and y-axis ticks
```

```
    ax[i // 2, i % 2].get_xaxis().set_visible(False)
    ax[i // 2, i % 2].get_yaxis().set_visible(False)

# Adjust spacing between subplots
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.8, top=0.8, wspace=0.05,␣
 ↪hspace=0.3)
```



Verify how many examples are in the train, validation, and test sets.

---

[7]:
```
train_num = inputs_train.shape[0]
val_num = inputs_valid.shape[0]
test_num = inputs_test.shape[0]

print(f"There are {train_num} training samples, {val_num} validation samples,␣
 ↪and {test_num} test samples")

#*************************************************************#
```

```
There are 3374 training samples, 419 validation samples, and 385 test samples
```

Verify how many classes are present in the label.

---

[8]:
```
# Determine the unique values in the label
# Hint: Use np.unique to output the unique values amongst the targets, then␣
 ↪count the classes

num_classes = len(np.unique(target_train))

print(f"There are {num_classes} classes in this labelled dataset")

#************************************************************#
```

```
There are 7 classes in this labelled dataset
```

1.2 Part 2: Train and Test Model

Task: Use Keras to train a classification model that predicts the correct face.

Run the code cell below. Note that each image consists of 2304 pixels. The training set contains 3374 of them.

[9]:
```
# Define model parameters. These parameters are defined for you.
# Run the code cells below

input_shape = inputs_train.shape[1]
learning_rate = 0.001
EPOCHS = 100
```

You will use Adam optimization to conduct the gradient descent. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

Run the code cell below

[10]:
```
# Create Neural Network Model

def create_model(num_classes,
                 input_shape,
                 learning_rate):
    """
    Creates a neural network model for classification tasks.

    Args:
        num_classes (int): Number of output classes (target categories).
        input_shape (int): Dimensionality of input features.
```

```
        learning_rate (float): Learning rate for the optimizer.

    Returns:
        tf.keras.models.Sequential: Compiled neural network model.
    """

    # Define a sequential model
    model = tf.keras.models.Sequential([
        # First hidden layer with 64 neurons and ReLU activation
        tf.keras.layers.Dense(64, activation="relu",␣
↪input_shape=(input_shape,)),
        # Second hidden layer with 128 neurons and ReLU activation
        tf.keras.layers.Dense(128, activation="relu"),
        # Third hidden layer with 256 neurons and ReLU activation
        tf.keras.layers.Dense(256, activation='relu'),
        # Output layer with 'num_classes' neurons and softmax activation
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])

    # Compile the model
    model.compile(optimizer=tf.keras.optimizers.
↪Adam(learning_rate=learning_rate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

[11]:
```
# Get untrained model. Run this code cell
model = create_model(num_classes,
                     input_shape,
                     learning_rate)

# Train the model
history = model.fit(inputs_train, target_train,
                    epochs=EPOCHS,
                    verbose=1,
                    validation_data=(inputs_valid, target_valid))
```

```
Epoch 1/100
106/106 [==============================] - 1s 4ms/step - loss: 1.7188 -
accuracy: 0.3435 - val_loss: 1.4825 - val_accuracy: 0.4248
Epoch 2/100
106/106 [==============================] - 0s 2ms/step - loss: 1.3516 -
accuracy: 0.5178 - val_loss: 1.3139 - val_accuracy: 0.4940
Epoch 3/100
106/106 [==============================] - 0s 2ms/step - loss: 1.2656 -
accuracy: 0.5453 - val_loss: 1.1338 - val_accuracy: 0.5752
```

```
Epoch 4/100
106/106 [==============================] - 0s 2ms/step - loss: 1.2310 -
accuracy: 0.5516 - val_loss: 1.1912 - val_accuracy: 0.5919
Epoch 5/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1039 -
accuracy: 0.5999 - val_loss: 1.2718 - val_accuracy: 0.5227
Epoch 6/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1085 -
accuracy: 0.5999 - val_loss: 1.2200 - val_accuracy: 0.5609
Epoch 7/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0562 -
accuracy: 0.6067 - val_loss: 0.9702 - val_accuracy: 0.6444
Epoch 8/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0139 -
accuracy: 0.6286 - val_loss: 1.0878 - val_accuracy: 0.6062
Epoch 9/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0017 -
accuracy: 0.6369 - val_loss: 1.0391 - val_accuracy: 0.6181
Epoch 10/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9573 -
accuracy: 0.6429 - val_loss: 1.0639 - val_accuracy: 0.6110
Epoch 11/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9688 -
accuracy: 0.6479 - val_loss: 0.9375 - val_accuracy: 0.6539
Epoch 12/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9531 -
accuracy: 0.6550 - val_loss: 0.8607 - val_accuracy: 0.6635
Epoch 13/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9188 -
accuracy: 0.6716 - val_loss: 1.1316 - val_accuracy: 0.6134
Epoch 14/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9499 -
accuracy: 0.6603 - val_loss: 0.9966 - val_accuracy: 0.6444
Epoch 15/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8737 -
accuracy: 0.6787 - val_loss: 0.9317 - val_accuracy: 0.6754
Epoch 16/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9048 -
accuracy: 0.6731 - val_loss: 0.8479 - val_accuracy: 0.6850
Epoch 17/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8061 -
accuracy: 0.7090 - val_loss: 0.8840 - val_accuracy: 0.6683
Epoch 18/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8511 -
accuracy: 0.6944 - val_loss: 1.0137 - val_accuracy: 0.6348
Epoch 19/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8379 -
accuracy: 0.6953 - val_loss: 0.7858 - val_accuracy: 0.6921
```

```
Epoch 20/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8026 -
accuracy: 0.7004 - val_loss: 0.7982 - val_accuracy: 0.6945
Epoch 21/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8139 -
accuracy: 0.7015 - val_loss: 0.8445 - val_accuracy: 0.6874
Epoch 22/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7927 -
accuracy: 0.7116 - val_loss: 0.9012 - val_accuracy: 0.6563
Epoch 23/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8078 -
accuracy: 0.7009 - val_loss: 0.8010 - val_accuracy: 0.6969
Epoch 24/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7451 -
accuracy: 0.7226 - val_loss: 0.7664 - val_accuracy: 0.6897
Epoch 25/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7720 -
accuracy: 0.7104 - val_loss: 0.9117 - val_accuracy: 0.6706
Epoch 26/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7262 -
accuracy: 0.7380 - val_loss: 0.7857 - val_accuracy: 0.6993
Epoch 27/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7591 -
accuracy: 0.7250 - val_loss: 0.7573 - val_accuracy: 0.7112
Epoch 28/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6951 -
accuracy: 0.7389 - val_loss: 0.7811 - val_accuracy: 0.7208
Epoch 29/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6879 -
accuracy: 0.7460 - val_loss: 0.8853 - val_accuracy: 0.6802
Epoch 30/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7074 -
accuracy: 0.7401 - val_loss: 0.8185 - val_accuracy: 0.7255
Epoch 31/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7222 -
accuracy: 0.7445 - val_loss: 0.8537 - val_accuracy: 0.6993
Epoch 32/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6838 -
accuracy: 0.7395 - val_loss: 0.7731 - val_accuracy: 0.7064
Epoch 33/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6317 -
accuracy: 0.7605 - val_loss: 0.7538 - val_accuracy: 0.7375
Epoch 34/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6627 -
accuracy: 0.7647 - val_loss: 1.0365 - val_accuracy: 0.6420
Epoch 35/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6900 -
accuracy: 0.7475 - val_loss: 0.8023 - val_accuracy: 0.7184
```

```
Epoch 36/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6026 -
accuracy: 0.7739 - val_loss: 0.8743 - val_accuracy: 0.7017
Epoch 37/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6232 -
accuracy: 0.7662 - val_loss: 0.8277 - val_accuracy: 0.6993
Epoch 38/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5740 -
accuracy: 0.7857 - val_loss: 0.8656 - val_accuracy: 0.7041
Epoch 39/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5756 -
accuracy: 0.7881 - val_loss: 0.9983 - val_accuracy: 0.6993
Epoch 40/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6412 -
accuracy: 0.7703 - val_loss: 0.8851 - val_accuracy: 0.7112
Epoch 41/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5538 -
accuracy: 0.7931 - val_loss: 0.7650 - val_accuracy: 0.7494
Epoch 42/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5607 -
accuracy: 0.7955 - val_loss: 0.9005 - val_accuracy: 0.6921
Epoch 43/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5512 -
accuracy: 0.7896 - val_loss: 0.9260 - val_accuracy: 0.7088
Epoch 44/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5847 -
accuracy: 0.7813 - val_loss: 0.9321 - val_accuracy: 0.6754
Epoch 45/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5793 -
accuracy: 0.7881 - val_loss: 0.7674 - val_accuracy: 0.7327
Epoch 46/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5619 -
accuracy: 0.7976 - val_loss: 0.8277 - val_accuracy: 0.6993
Epoch 47/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4919 -
accuracy: 0.8156 - val_loss: 0.7791 - val_accuracy: 0.7160
Epoch 48/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5207 -
accuracy: 0.8038 - val_loss: 0.8082 - val_accuracy: 0.7422
Epoch 49/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5150 -
accuracy: 0.8112 - val_loss: 0.8328 - val_accuracy: 0.7208
Epoch 50/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5198 -
accuracy: 0.8068 - val_loss: 0.8311 - val_accuracy: 0.7160
Epoch 51/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4974 -
accuracy: 0.8159 - val_loss: 0.8009 - val_accuracy: 0.7566
```

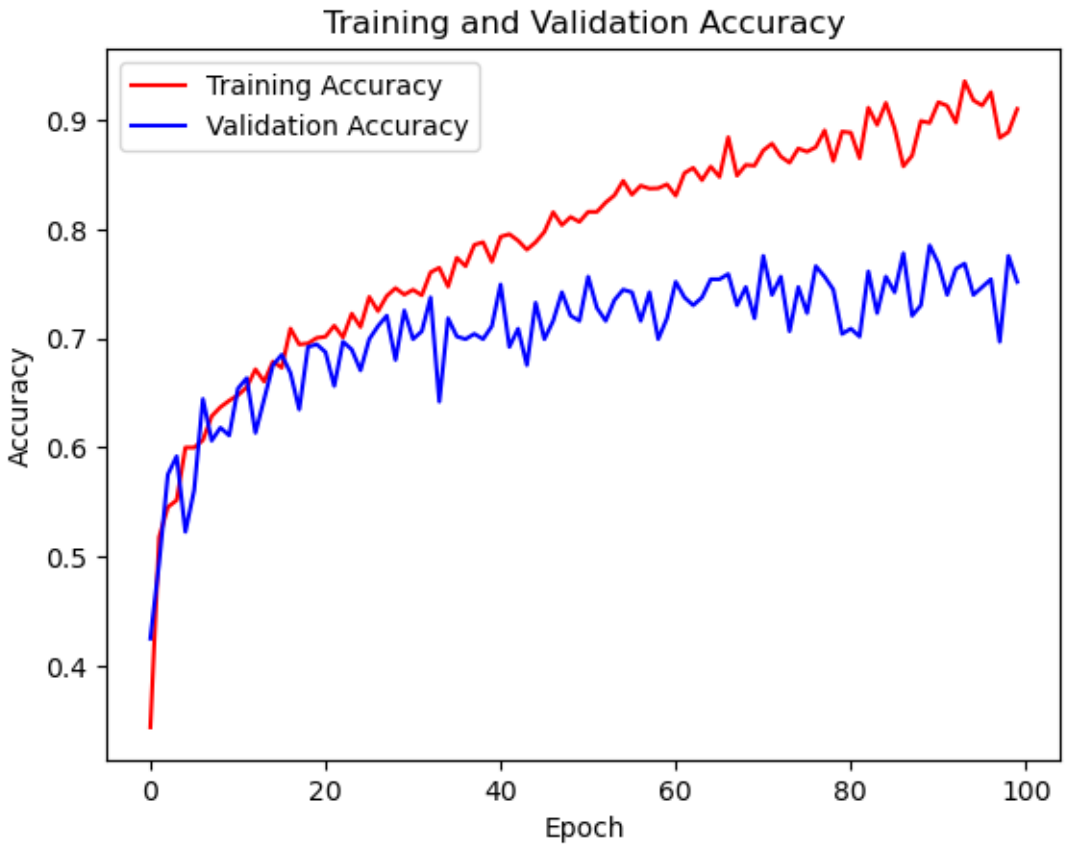
```
Epoch 52/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4988 -
accuracy: 0.8159 - val_loss: 0.8641 - val_accuracy: 0.7279
Epoch 53/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4769 -
accuracy: 0.8245 - val_loss: 0.8392 - val_accuracy: 0.7160
Epoch 54/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4631 -
accuracy: 0.8311 - val_loss: 0.8130 - val_accuracy: 0.7351
Epoch 55/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4325 -
accuracy: 0.8444 - val_loss: 0.8753 - val_accuracy: 0.7446
Epoch 56/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4610 -
accuracy: 0.8317 - val_loss: 0.8175 - val_accuracy: 0.7422
Epoch 57/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4450 -
accuracy: 0.8400 - val_loss: 0.9310 - val_accuracy: 0.7160
Epoch 58/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4533 -
accuracy: 0.8373 - val_loss: 0.7615 - val_accuracy: 0.7422
Epoch 59/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4313 -
accuracy: 0.8376 - val_loss: 0.9765 - val_accuracy: 0.6993
Epoch 60/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4328 -
accuracy: 0.8411 - val_loss: 0.9646 - val_accuracy: 0.7184
Epoch 61/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4588 -
accuracy: 0.8308 - val_loss: 0.8526 - val_accuracy: 0.7518
Epoch 62/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4075 -
accuracy: 0.8515 - val_loss: 0.8873 - val_accuracy: 0.7375
Epoch 63/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3765 -
accuracy: 0.8563 - val_loss: 0.9589 - val_accuracy: 0.7303
Epoch 64/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4204 -
accuracy: 0.8450 - val_loss: 0.8876 - val_accuracy: 0.7375
Epoch 65/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3996 -
accuracy: 0.8574 - val_loss: 0.8428 - val_accuracy: 0.7542
Epoch 66/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4159 -
accuracy: 0.8480 - val_loss: 0.8420 - val_accuracy: 0.7542
Epoch 67/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3370 -
accuracy: 0.8844 - val_loss: 0.8651 - val_accuracy: 0.7589
```

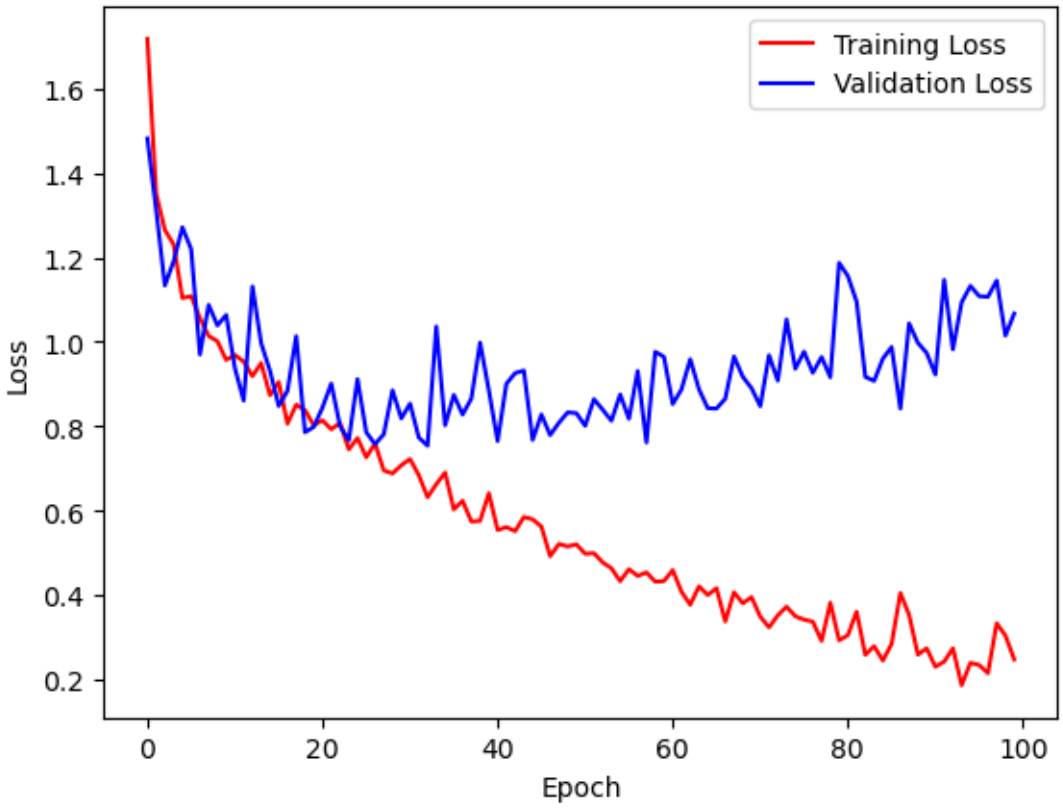
```
Epoch 68/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4061 -
accuracy: 0.8491 - val_loss: 0.9654 - val_accuracy: 0.7303
Epoch 69/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3800 -
accuracy: 0.8589 - val_loss: 0.9173 - val_accuracy: 0.7470
Epoch 70/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3947 -
accuracy: 0.8583 - val_loss: 0.8902 - val_accuracy: 0.7184
Epoch 71/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3484 -
accuracy: 0.8723 - val_loss: 0.8474 - val_accuracy: 0.7757
Epoch 72/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3230 -
accuracy: 0.8785 - val_loss: 0.9683 - val_accuracy: 0.7399
Epoch 73/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3514 -
accuracy: 0.8666 - val_loss: 0.9081 - val_accuracy: 0.7566
Epoch 74/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3721 -
accuracy: 0.8610 - val_loss: 1.0534 - val_accuracy: 0.7064
Epoch 75/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3492 -
accuracy: 0.8740 - val_loss: 0.9363 - val_accuracy: 0.7470
Epoch 76/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3413 -
accuracy: 0.8711 - val_loss: 0.9769 - val_accuracy: 0.7232
Epoch 77/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3365 -
accuracy: 0.8752 - val_loss: 0.9267 - val_accuracy: 0.7661
Epoch 78/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2908 -
accuracy: 0.8906 - val_loss: 0.9642 - val_accuracy: 0.7566
Epoch 79/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3815 -
accuracy: 0.8625 - val_loss: 0.9160 - val_accuracy: 0.7446
Epoch 80/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2922 -
accuracy: 0.8894 - val_loss: 1.1874 - val_accuracy: 0.7041
Epoch 81/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3043 -
accuracy: 0.8886 - val_loss: 1.1563 - val_accuracy: 0.7088
Epoch 82/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3602 -
accuracy: 0.8651 - val_loss: 1.0957 - val_accuracy: 0.7017
Epoch 83/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2578 -
accuracy: 0.9114 - val_loss: 0.9171 - val_accuracy: 0.7613
```

```
Epoch 84/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2784 -
accuracy: 0.8957 - val_loss: 0.9075 - val_accuracy: 0.7232
Epoch 85/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2442 -
accuracy: 0.9158 - val_loss: 0.9603 - val_accuracy: 0.7566
Epoch 86/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2839 -
accuracy: 0.8924 - val_loss: 0.9878 - val_accuracy: 0.7422
Epoch 87/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4044 -
accuracy: 0.8577 - val_loss: 0.8419 - val_accuracy: 0.7780
Epoch 88/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3524 -
accuracy: 0.8675 - val_loss: 1.0439 - val_accuracy: 0.7208
Epoch 89/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2583 -
accuracy: 0.8992 - val_loss: 0.9977 - val_accuracy: 0.7303
Epoch 90/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2732 -
accuracy: 0.8975 - val_loss: 0.9739 - val_accuracy: 0.7852
Epoch 91/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2295 -
accuracy: 0.9164 - val_loss: 0.9225 - val_accuracy: 0.7685
Epoch 92/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2417 -
accuracy: 0.9132 - val_loss: 1.1472 - val_accuracy: 0.7399
Epoch 93/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2734 -
accuracy: 0.8977 - val_loss: 0.9824 - val_accuracy: 0.7637
Epoch 94/100
106/106 [==============================] - 0s 2ms/step - loss: 0.1858 -
accuracy: 0.9357 - val_loss: 1.0940 - val_accuracy: 0.7685
Epoch 95/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2388 -
accuracy: 0.9182 - val_loss: 1.1328 - val_accuracy: 0.7399
Epoch 96/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2336 -
accuracy: 0.9135 - val_loss: 1.1084 - val_accuracy: 0.7470
Epoch 97/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2143 -
accuracy: 0.9256 - val_loss: 1.1070 - val_accuracy: 0.7542
Epoch 98/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3326 -
accuracy: 0.8838 - val_loss: 1.1457 - val_accuracy: 0.6969
Epoch 99/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3034 -
accuracy: 0.8892 - val_loss: 1.0149 - val_accuracy: 0.7757
```

```
Epoch 100/100
106/106 [==============================] - 0s 2ms/step - loss: 0.2470 -
accuracy: 0.9102 - val_loss: 1.0672 - val_accuracy: 0.7518
```

[12]:
```python
#------------------------------------------------------------
# Retrieve results on training and validation datasets for each training epoch
#------------------------------------------------------------
# Extract training accuracy, validation accuracy, training loss, and validation loss
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

# Get the total number of epochs
epochs = range(len(acc))

#------------------------------------------------
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot(epochs, acc, 'r', label="Training Accuracy")  # Plot training accuracy in red
plt.plot(epochs, val_acc, 'b', label="Validation Accuracy")  # Plot validation accuracy in blue
plt.title('Training and Validation Accuracy')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()  # Display legend
plt.show()  # Show the plot
print("")  # Print an empty line

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r', label="Training Loss")  # Plot training loss in red
plt.plot(epochs, val_loss, 'b', label="Validation Loss")  # Plot validation loss in blue
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()  # Display legend
plt.show()  # Show the plot
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
0.4
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100



Problem

What do you observe about the training loss and validation loss using the default learning rate of 10^{-3}?

YourAnswer :

Problem

Using the function `model.predict`, carry out predictions on the test set.

---

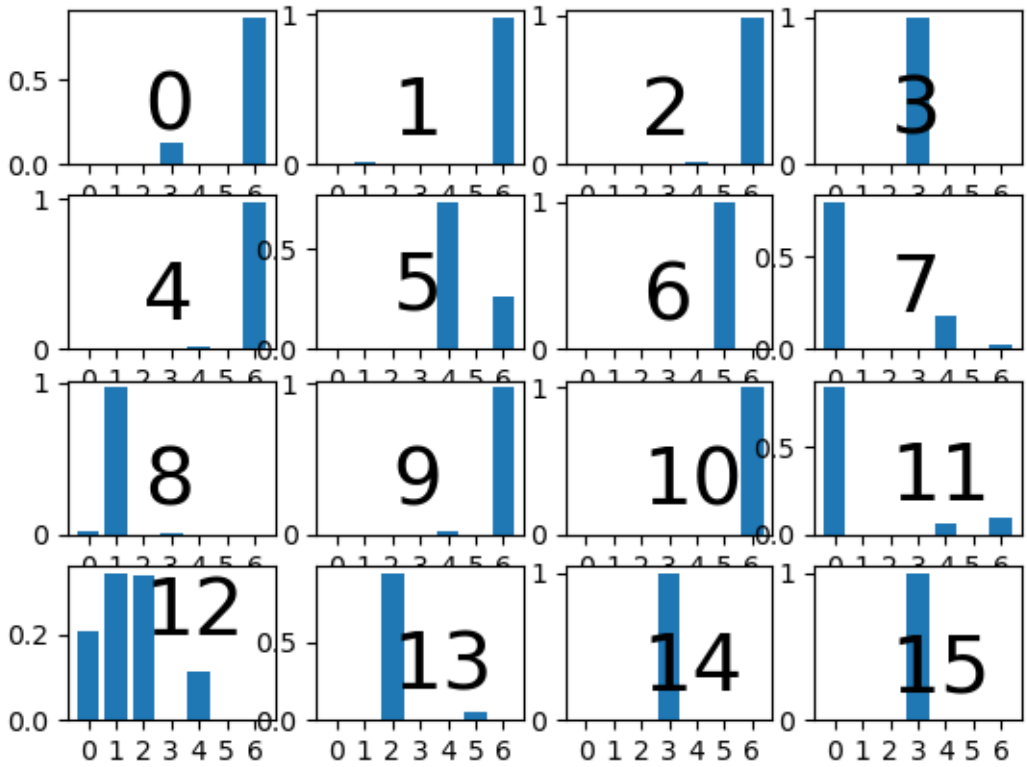
[13]:
```
#****************YOUR CODE HERE (Problem 2.2) ****************#

predicted_ytest = model.predict(inputs_test)

#*************************************************************#
```

```
13/13 [==============================] - 0s 1ms/step
```

The results are probabilities which, for each test sample, provides the probability of being any of the 7 classes. The most probable class has the highest probability.

You can visualize this for the first 16 samples using a barplot.

[14]:
```
h =4
m = n = h
labels = list(range(7))
u=m*n
plotted_predicted_test = predicted_ytest[:u, :]
fig, ax = plt.subplots(m,n)
for i, j in enumerate(range(m*n)):
  ax[i//m, i%n].bar(labels, plotted_predicted_test[j, :])
  ax[i//m, i%n].set_xticks(list(range(7)))
  ax[i//m, i%n].text(2, 0.2,str(j), fontsize=30)
plt.show()
```



This means for the first test sample (`sample 0`), the most probable output is the class with the longest bar. The sample applies for every other prediction.

Next you need to convert these probabilities to the required classes.

You use

$$\hat{c}_{pred} = \arg\max \hat{y}$$

Problem

Find the predicted class for each test sample using the result, `predicted_ytest`, from above and print out the first 10 results from your prediction. Use `np.argmax` to do this. Don't forget to set the right value for the axis parameter!

"axis 0" represents rows and "axis 1" represents columns.

---

[15]:
```
predicted_test = np.argmax(predicted_ytest, axis=1)

#************************************************************#
```

[16]:
```
# Print out the predicted labels for the first 10 examples in the test set
predicted_test[:10]
```

[16]:
```
array([6, 6, 6, 3, 6, 4, 5, 0, 1, 6])
```

Having obtained your prediction, you may now see the summary of this model that produced this output. The model summary is shown below. This model summary provides an outline of the neural network model. In the first layer, we have 64 units. In the second, we have 128 units, and in the third we have 512 units. The output layer has 7 units because we have 7 classes in the output label.

[17]:
```
# See the model summary. Run this code cell
model.summary()
48*48*64 + 64
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                147520

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 256)               33024

 dense_3 (Dense)             (None, 7)                 1799

=================================================================
Total params: 190663 (744.78 KB)
Trainable params: 190663 (744.78 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

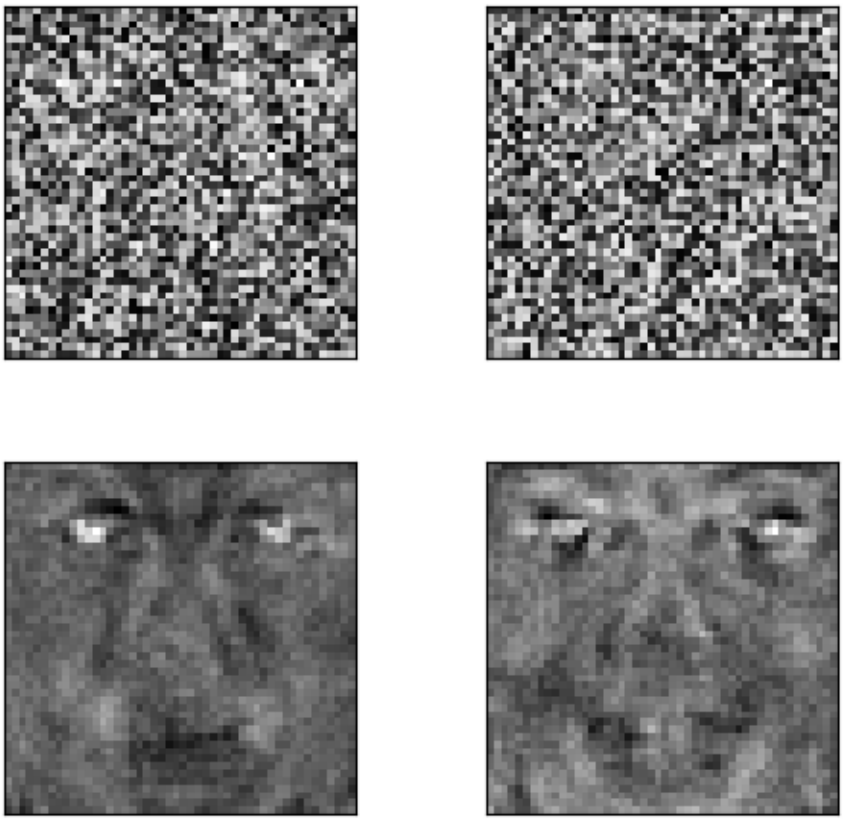
[17]:
```
147520
```

[18]:
```
# You can also obtain the weights obtained for each layer as discussed in class
weights = [layer.get_weights() for layer in model.layers]
```

For the first layer, we can obtain the weights as below

[19]:
```
# Set the layer index (e.g., layer 0)
layer = 0

# Print information about the weights of the specified layer
print(f"The weights of layer {layer} have shape {weights[layer][0].shape}")
print()

# Set the desired image size (48x48 pixels)
img_size = 48

# Create a 2x2 grid of subplots with a specified figure size
fig, ax = plt.subplots(2, 2, figsize=(7, 6))

# Iterate over the 4 subplots
for i in range(4):
    # Randomly select an index 's' from the weights of the specified layer
    s = np.random.choice(range(weights[layer][0].shape[1]))

    # Display the weight vector at index 's' after reshaping it to the desired␣
 ↪size
    ax[i // 2, i % 2].imshow(weights[layer][0][:, s].reshape((img_size,␣
 ↪img_size)), cmap="gray")

    # Hide x-axis and y-axis ticks
    ax[i // 2, i % 2].get_xaxis().set_visible(False)
    ax[i // 2, i % 2].get_yaxis().set_visible(False)

# Adjust spacing between subplots
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.8, top=0.8, wspace=0.05,␣
 ↪hspace=0.3)
```

```
The weights of layer 0 have shape (2304, 64)
```



The weights of the first layer has shape 2304×64 because the layer has 64 units and the input data has 2304 features. **You can change the value of `layer` to see the weights of other layers in the network.**

Problem

Find the accuracy of the model using the test data.

---

[20]:
```
# Calculate the accuracy

accuracy_value = np.mean(predicted_test == target_test)
print(f"Test Accuracy = {accuracy_value}")  # returns the fraction of test data␣
↪classified correctly

#************************************************************#
```

```
Test Accuracy = 0.7142857142857143
```

1.3 Part 3: Modify Model and Tune Learning Rate

A model's learning rate can be one of the most important parameters which could help with effective optimization via gradient descent.

To find this optimally, you can use a learning rate scheduler in `Keras`. This is a part of machine learning called hyperparameter tuning and is essential to obtaining a good model.

[21]:
```
# Run the code cells below

def adjust_learning_rate(num_classes,
                         input_shape,
                         learning_rate):

    model = create_model(num_classes,
                 input_shape,
                 learning_rate)

    # Select your optimizer
    optimizer = tf.keras.optimizers.Adam()

    # Compile the model passing in the appropriate loss
    model.compile(optimizer=optimizer,
                  loss="sparse_categorical_crossentropy",
                  metrics=["accuracy"])

    #history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])

    return model
```

[22]:
```
# Get untrained model
model = adjust_learning_rate(num_classes,
                      input_shape,
                      learning_rate)

# Train the model
lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-6 *␣
 ↪10**(epoch / 20))
history = model.fit(inputs_train, target_train,
                    epochs=EPOCHS,
                    verbose=1,
                    callbacks=[lr_schedule],
                    validation_data=(inputs_valid, target_valid))
```

```
Epoch 1/100
106/106 [==============================] - 1s 3ms/step - loss: 1.9075 -
accuracy: 0.2018 - val_loss: 1.8956 - val_accuracy: 0.2530 - lr: 1.0000e-06
Epoch 2/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8885 -
```

```
accuracy: 0.2774 - val_loss: 1.8809 - val_accuracy: 0.2888 - lr: 1.1220e-06
Epoch 3/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8726 -
accuracy: 0.2949 - val_loss: 1.8683 - val_accuracy: 0.2840 - lr: 1.2589e-06
Epoch 4/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8593 -
accuracy: 0.2869 - val_loss: 1.8580 - val_accuracy: 0.2792 - lr: 1.4125e-06
Epoch 5/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8499 -
accuracy: 0.2854 - val_loss: 1.8507 - val_accuracy: 0.2792 - lr: 1.5849e-06
Epoch 6/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8432 -
accuracy: 0.2854 - val_loss: 1.8453 - val_accuracy: 0.2792 - lr: 1.7783e-06
Epoch 7/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8385 -
accuracy: 0.2854 - val_loss: 1.8415 - val_accuracy: 0.2792 - lr: 1.9953e-06
Epoch 8/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8349 -
accuracy: 0.2854 - val_loss: 1.8375 - val_accuracy: 0.2792 - lr: 2.2387e-06
Epoch 9/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8318 -
accuracy: 0.2854 - val_loss: 1.8341 - val_accuracy: 0.2792 - lr: 2.5119e-06
Epoch 10/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8287 -
accuracy: 0.2854 - val_loss: 1.8308 - val_accuracy: 0.2792 - lr: 2.8184e-06
Epoch 11/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8255 -
accuracy: 0.2854 - val_loss: 1.8277 - val_accuracy: 0.2792 - lr: 3.1623e-06
Epoch 12/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8218 -
accuracy: 0.2854 - val_loss: 1.8230 - val_accuracy: 0.2792 - lr: 3.5481e-06
Epoch 13/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8177 -
accuracy: 0.2854 - val_loss: 1.8177 - val_accuracy: 0.2792 - lr: 3.9811e-06
Epoch 14/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8129 -
accuracy: 0.2854 - val_loss: 1.8123 - val_accuracy: 0.2792 - lr: 4.4668e-06
Epoch 15/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8068 -
accuracy: 0.2860 - val_loss: 1.8069 - val_accuracy: 0.2816 - lr: 5.0119e-06
Epoch 16/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8012 -
accuracy: 0.2863 - val_loss: 1.8006 - val_accuracy: 0.2816 - lr: 5.6234e-06
Epoch 17/100
106/106 [==============================] - 0s 2ms/step - loss: 1.7929 -
accuracy: 0.2872 - val_loss: 1.7913 - val_accuracy: 0.2816 - lr: 6.3096e-06
Epoch 18/100
106/106 [==============================] - 0s 2ms/step - loss: 1.7837 -
```

```
accuracy: 0.2905 - val_loss: 1.7814 - val_accuracy: 0.2840 - lr: 7.0795e-06
Epoch 19/100
106/106 [==============================] - 0s 2ms/step - loss: 1.7732 -
accuracy: 0.2949 - val_loss: 1.7670 - val_accuracy: 0.2840 - lr: 7.9433e-06
Epoch 20/100
106/106 [==============================] - 0s 2ms/step - loss: 1.7580 -
accuracy: 0.3112 - val_loss: 1.7533 - val_accuracy: 0.2983 - lr: 8.9125e-06
Epoch 21/100
106/106 [==============================] - 0s 2ms/step - loss: 1.7417 -
accuracy: 0.3171 - val_loss: 1.7287 - val_accuracy: 0.3246 - lr: 1.0000e-05
Epoch 22/100
106/106 [==============================] - 0s 2ms/step - loss: 1.7195 -
accuracy: 0.3417 - val_loss: 1.7148 - val_accuracy: 0.3174 - lr: 1.1220e-05
Epoch 23/100
106/106 [==============================] - 0s 2ms/step - loss: 1.6908 -
accuracy: 0.3726 - val_loss: 1.6682 - val_accuracy: 0.3699 - lr: 1.2589e-05
Epoch 24/100
106/106 [==============================] - 0s 3ms/step - loss: 1.6555 -
accuracy: 0.3912 - val_loss: 1.6275 - val_accuracy: 0.4391 - lr: 1.4125e-05
Epoch 25/100
106/106 [==============================] - 0s 2ms/step - loss: 1.6179 -
accuracy: 0.4259 - val_loss: 1.5836 - val_accuracy: 0.4773 - lr: 1.5849e-05
Epoch 26/100
106/106 [==============================] - 0s 2ms/step - loss: 1.5728 -
accuracy: 0.4585 - val_loss: 1.5426 - val_accuracy: 0.4869 - lr: 1.7783e-05
Epoch 27/100
106/106 [==============================] - 0s 2ms/step - loss: 1.5275 -
accuracy: 0.4715 - val_loss: 1.4830 - val_accuracy: 0.5251 - lr: 1.9953e-05
Epoch 28/100
106/106 [==============================] - 0s 3ms/step - loss: 1.4805 -
accuracy: 0.4908 - val_loss: 1.4369 - val_accuracy: 0.5394 - lr: 2.2387e-05
Epoch 29/100
106/106 [==============================] - 0s 2ms/step - loss: 1.4380 -
accuracy: 0.5033 - val_loss: 1.3993 - val_accuracy: 0.5227 - lr: 2.5119e-05
Epoch 30/100
106/106 [==============================] - 0s 2ms/step - loss: 1.3927 -
accuracy: 0.5252 - val_loss: 1.3491 - val_accuracy: 0.5561 - lr: 2.8184e-05
Epoch 31/100
106/106 [==============================] - 0s 2ms/step - loss: 1.3503 -
accuracy: 0.5344 - val_loss: 1.3188 - val_accuracy: 0.5489 - lr: 3.1623e-05
Epoch 32/100
106/106 [==============================] - 0s 2ms/step - loss: 1.3091 -
accuracy: 0.5525 - val_loss: 1.2615 - val_accuracy: 0.5561 - lr: 3.5481e-05
Epoch 33/100
106/106 [==============================] - 0s 2ms/step - loss: 1.2718 -
accuracy: 0.5714 - val_loss: 1.2276 - val_accuracy: 0.5800 - lr: 3.9811e-05
Epoch 34/100
106/106 [==============================] - 0s 3ms/step - loss: 1.2431 -
```

```
accuracy: 0.5705 - val_loss: 1.1815 - val_accuracy: 0.5847 - lr: 4.4668e-05
Epoch 35/100
106/106 [==============================] - 0s 3ms/step - loss: 1.2209 -
accuracy: 0.5779 - val_loss: 1.1833 - val_accuracy: 0.5800 - lr: 5.0119e-05
Epoch 36/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1890 -
accuracy: 0.5815 - val_loss: 1.1376 - val_accuracy: 0.6014 - lr: 5.6234e-05
Epoch 37/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1570 -
accuracy: 0.5972 - val_loss: 1.1116 - val_accuracy: 0.5847 - lr: 6.3096e-05
Epoch 38/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1191 -
accuracy: 0.6114 - val_loss: 1.0698 - val_accuracy: 0.6014 - lr: 7.0795e-05
Epoch 39/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1000 -
accuracy: 0.6147 - val_loss: 1.0281 - val_accuracy: 0.6277 - lr: 7.9433e-05
Epoch 40/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0731 -
accuracy: 0.6218 - val_loss: 1.0738 - val_accuracy: 0.6229 - lr: 8.9125e-05
Epoch 41/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0585 -
accuracy: 0.6286 - val_loss: 1.0124 - val_accuracy: 0.6277 - lr: 1.0000e-04
Epoch 42/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0363 -
accuracy: 0.6340 - val_loss: 1.0324 - val_accuracy: 0.6062 - lr: 1.1220e-04
Epoch 43/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0384 -
accuracy: 0.6313 - val_loss: 1.1117 - val_accuracy: 0.6110 - lr: 1.2589e-04
Epoch 44/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0239 -
accuracy: 0.6337 - val_loss: 0.9755 - val_accuracy: 0.6659 - lr: 1.4125e-04
Epoch 45/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9947 -
accuracy: 0.6470 - val_loss: 0.9564 - val_accuracy: 0.6420 - lr: 1.5849e-04
Epoch 46/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9520 -
accuracy: 0.6589 - val_loss: 0.9343 - val_accuracy: 0.6516 - lr: 1.7783e-04
Epoch 47/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9492 -
accuracy: 0.6580 - val_loss: 0.9783 - val_accuracy: 0.6420 - lr: 1.9953e-04
Epoch 48/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9520 -
accuracy: 0.6550 - val_loss: 0.9139 - val_accuracy: 0.6277 - lr: 2.2387e-04
Epoch 49/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9288 -
accuracy: 0.6636 - val_loss: 0.8792 - val_accuracy: 0.6921 - lr: 2.5119e-04
Epoch 50/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9274 -
```

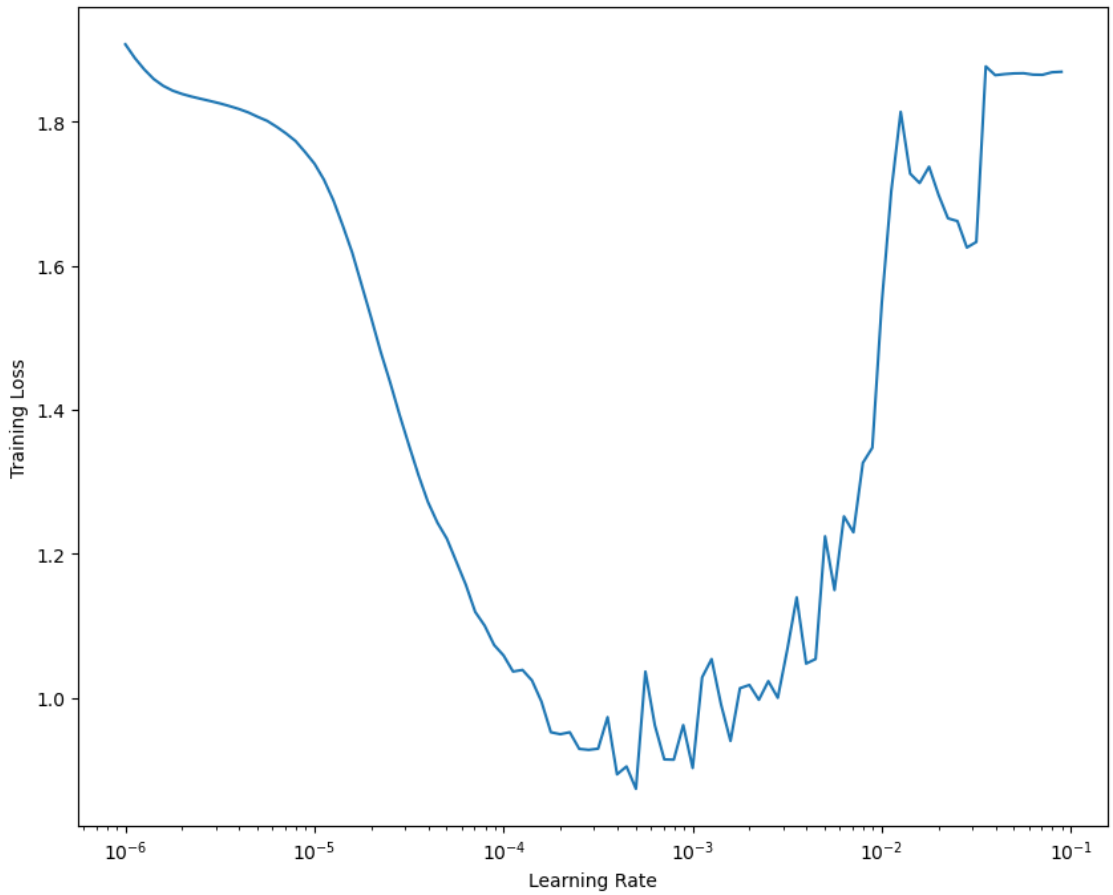
```
accuracy: 0.6636 - val_loss: 0.8750 - val_accuracy: 0.6778 - lr: 2.8184e-04
Epoch 51/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9291 -
accuracy: 0.6683 - val_loss: 1.0910 - val_accuracy: 0.6229 - lr: 3.1623e-04
Epoch 52/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9728 -
accuracy: 0.6476 - val_loss: 1.0165 - val_accuracy: 0.6539 - lr: 3.5481e-04
Epoch 53/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8934 -
accuracy: 0.6761 - val_loss: 0.8729 - val_accuracy: 0.7017 - lr: 3.9811e-04
Epoch 54/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9044 -
accuracy: 0.6829 - val_loss: 0.8561 - val_accuracy: 0.6897 - lr: 4.4668e-04
Epoch 55/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8733 -
accuracy: 0.6811 - val_loss: 1.3730 - val_accuracy: 0.5800 - lr: 5.0119e-04
Epoch 56/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0362 -
accuracy: 0.6360 - val_loss: 1.0262 - val_accuracy: 0.6325 - lr: 5.6234e-04
Epoch 57/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9615 -
accuracy: 0.6609 - val_loss: 1.1585 - val_accuracy: 0.6348 - lr: 6.3096e-04
Epoch 58/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9142 -
accuracy: 0.6778 - val_loss: 1.1442 - val_accuracy: 0.6086 - lr: 7.0795e-04
Epoch 59/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9138 -
accuracy: 0.6642 - val_loss: 0.8472 - val_accuracy: 0.6778 - lr: 7.9433e-04
Epoch 60/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9619 -
accuracy: 0.6633 - val_loss: 0.8401 - val_accuracy: 0.6683 - lr: 8.9125e-04
Epoch 61/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9023 -
accuracy: 0.6755 - val_loss: 1.0112 - val_accuracy: 0.6134 - lr: 0.0010
Epoch 62/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0282 -
accuracy: 0.6337 - val_loss: 1.1216 - val_accuracy: 0.5800 - lr: 0.0011
Epoch 63/100
106/106 [==============================] - 0s 3ms/step - loss: 1.0536 -
accuracy: 0.6233 - val_loss: 0.8698 - val_accuracy: 0.6635 - lr: 0.0013
Epoch 64/100
106/106 [==============================] - 0s 3ms/step - loss: 0.9905 -
accuracy: 0.6494 - val_loss: 1.0670 - val_accuracy: 0.6205 - lr: 0.0014
Epoch 65/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9399 -
accuracy: 0.6603 - val_loss: 0.8432 - val_accuracy: 0.6921 - lr: 0.0016
Epoch 66/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0130 -
```

```
accuracy: 0.6343 - val_loss: 1.0063 - val_accuracy: 0.6253 - lr: 0.0018
Epoch 67/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0176 -
accuracy: 0.6440 - val_loss: 0.9137 - val_accuracy: 0.6563 - lr: 0.0020
Epoch 68/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9970 -
accuracy: 0.6352 - val_loss: 1.0250 - val_accuracy: 0.6205 - lr: 0.0022
Epoch 69/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0229 -
accuracy: 0.6337 - val_loss: 0.8892 - val_accuracy: 0.6492 - lr: 0.0025
Epoch 70/100
106/106 [==============================] - 0s 3ms/step - loss: 0.9997 -
accuracy: 0.6476 - val_loss: 0.9612 - val_accuracy: 0.6826 - lr: 0.0028
Epoch 71/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0664 -
accuracy: 0.6156 - val_loss: 1.0241 - val_accuracy: 0.6611 - lr: 0.0032
Epoch 72/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1392 -
accuracy: 0.5862 - val_loss: 1.1220 - val_accuracy: 0.5895 - lr: 0.0035
Epoch 73/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0473 -
accuracy: 0.6191 - val_loss: 1.1244 - val_accuracy: 0.5919 - lr: 0.0040
Epoch 74/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0536 -
accuracy: 0.6153 - val_loss: 1.3525 - val_accuracy: 0.5632 - lr: 0.0045
Epoch 75/100
106/106 [==============================] - 0s 2ms/step - loss: 1.2240 -
accuracy: 0.5682 - val_loss: 1.2083 - val_accuracy: 0.5394 - lr: 0.0050
Epoch 76/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1495 -
accuracy: 0.5735 - val_loss: 1.0092 - val_accuracy: 0.6492 - lr: 0.0056
Epoch 77/100
106/106 [==============================] - 0s 2ms/step - loss: 1.2518 -
accuracy: 0.5427 - val_loss: 1.1160 - val_accuracy: 0.5943 - lr: 0.0063
Epoch 78/100
106/106 [==============================] - 0s 2ms/step - loss: 1.2297 -
accuracy: 0.5563 - val_loss: 1.1577 - val_accuracy: 0.5632 - lr: 0.0071
Epoch 79/100
106/106 [==============================] - 0s 2ms/step - loss: 1.3262 -
accuracy: 0.5317 - val_loss: 1.4611 - val_accuracy: 0.5012 - lr: 0.0079
Epoch 80/100
106/106 [==============================] - 0s 2ms/step - loss: 1.3472 -
accuracy: 0.5124 - val_loss: 1.8462 - val_accuracy: 0.4081 - lr: 0.0089
Epoch 81/100
106/106 [==============================] - 0s 2ms/step - loss: 1.5481 -
accuracy: 0.4520 - val_loss: 1.6806 - val_accuracy: 0.3508 - lr: 0.0100
Epoch 82/100
106/106 [==============================] - 0s 2ms/step - loss: 1.7034 -
```

```
accuracy: 0.3560 - val_loss: 1.7083 - val_accuracy: 0.3580 - lr: 0.0112
Epoch 83/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8137 -
accuracy: 0.3029 - val_loss: 1.8140 - val_accuracy: 0.2792 - lr: 0.0126
Epoch 84/100
106/106 [==============================] - 0s 3ms/step - loss: 1.7279 -
accuracy: 0.3379 - val_loss: 1.6763 - val_accuracy: 0.3508 - lr: 0.0141
Epoch 85/100
106/106 [==============================] - 0s 3ms/step - loss: 1.7148 -
accuracy: 0.3527 - val_loss: 1.6386 - val_accuracy: 0.3675 - lr: 0.0158
Epoch 86/100
106/106 [==============================] - 0s 3ms/step - loss: 1.7375 -
accuracy: 0.3441 - val_loss: 1.7975 - val_accuracy: 0.2840 - lr: 0.0178
Epoch 87/100
106/106 [==============================] - 0s 3ms/step - loss: 1.6986 -
accuracy: 0.3604 - val_loss: 1.7130 - val_accuracy: 0.3508 - lr: 0.0200
Epoch 88/100
106/106 [==============================] - 0s 2ms/step - loss: 1.6658 -
accuracy: 0.3711 - val_loss: 1.6909 - val_accuracy: 0.3914 - lr: 0.0224
Epoch 89/100
106/106 [==============================] - 0s 2ms/step - loss: 1.6620 -
accuracy: 0.3512 - val_loss: 1.6078 - val_accuracy: 0.3675 - lr: 0.0251
Epoch 90/100
106/106 [==============================] - 0s 3ms/step - loss: 1.6252 -
accuracy: 0.3631 - val_loss: 1.6577 - val_accuracy: 0.3508 - lr: 0.0282
Epoch 91/100
106/106 [==============================] - 0s 3ms/step - loss: 1.6330 -
accuracy: 0.3613 - val_loss: 1.9379 - val_accuracy: 0.2792 - lr: 0.0316
Epoch 92/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8769 -
accuracy: 0.2830 - val_loss: 1.8616 - val_accuracy: 0.2792 - lr: 0.0355
Epoch 93/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8647 -
accuracy: 0.2854 - val_loss: 1.8645 - val_accuracy: 0.2792 - lr: 0.0398
Epoch 94/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8661 -
accuracy: 0.2854 - val_loss: 1.8646 - val_accuracy: 0.2792 - lr: 0.0447
Epoch 95/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8671 -
accuracy: 0.2854 - val_loss: 1.8705 - val_accuracy: 0.2792 - lr: 0.0501
Epoch 96/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8674 -
accuracy: 0.2854 - val_loss: 1.8657 - val_accuracy: 0.2792 - lr: 0.0562
Epoch 97/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8654 -
accuracy: 0.2854 - val_loss: 1.8616 - val_accuracy: 0.2792 - lr: 0.0631
Epoch 98/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8652 -
```

```
accuracy: 0.2813 - val_loss: 1.8639 - val_accuracy: 0.2792 - lr: 0.0708
Epoch 99/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8687 -
accuracy: 0.2854 - val_loss: 1.8628 - val_accuracy: 0.2792 - lr: 0.0794
Epoch 100/100
106/106 [==============================] - 0s 2ms/step - loss: 1.8694 -
accuracy: 0.2854 - val_loss: 1.8638 - val_accuracy: 0.2792 - lr: 0.0891
```

[23]:
```
# Plot the loss against LR
plt.figure(figsize=(10,8))
plt.semilogx(history.history["lr"], history.history["loss"])
plt.xlabel("Learning Rate")
plt.ylabel("Training Loss")
plt.show()
```



Problem

What do you observe about this plot of `training_loss` against `learning_rate`?

YourAnswer :

Obtain the learning rate resulting in the smallest loss. However, we do not directly choose this value because the loss is not stable there. Instead, we choose some distance before where the loss is more stable. In this case, we have chosen a learning_rate of 0.0002.

Retrain your model with the new learning rate.

[24]:
```
learning_rate = 2e-4 # new learning rate

# Retrain model
# Get untrained model
model = create_model(num_classes,
                     input_shape,
                     learning_rate)

# Train the model
history = model.fit(inputs_train, target_train,
                    epochs=EPOCHS,
                    verbose=1,
                    validation_data=(inputs_valid, target_valid))
```

```
Epoch 1/100
106/106 [==============================] - 1s 3ms/step - loss: 1.8037 -
accuracy: 0.3094 - val_loss: 1.6992 - val_accuracy: 0.4296
Epoch 2/100
106/106 [==============================] - 0s 3ms/step - loss: 1.6077 -
accuracy: 0.4265 - val_loss: 1.4937 - val_accuracy: 0.4726
Epoch 3/100
106/106 [==============================] - 0s 2ms/step - loss: 1.4324 -
accuracy: 0.4905 - val_loss: 1.3472 - val_accuracy: 0.5107
Epoch 4/100
106/106 [==============================] - 0s 2ms/step - loss: 1.3183 -
accuracy: 0.5406 - val_loss: 1.2550 - val_accuracy: 0.5346
Epoch 5/100
106/106 [==============================] - 0s 2ms/step - loss: 1.2545 -
accuracy: 0.5528 - val_loss: 1.2762 - val_accuracy: 0.5322
Epoch 6/100
106/106 [==============================] - 0s 3ms/step - loss: 1.1778 -
accuracy: 0.5848 - val_loss: 1.1318 - val_accuracy: 0.5752
Epoch 7/100
106/106 [==============================] - 0s 2ms/step - loss: 1.1202 -
accuracy: 0.6043 - val_loss: 1.1296 - val_accuracy: 0.5990
Epoch 8/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0846 -
accuracy: 0.6212 - val_loss: 1.0332 - val_accuracy: 0.6229
```

```
Epoch 9/100
106/106 [==============================] - 0s 3ms/step - loss: 1.0449 -
accuracy: 0.6254 - val_loss: 1.0155 - val_accuracy: 0.6014
Epoch 10/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0336 -
accuracy: 0.6310 - val_loss: 0.9668 - val_accuracy: 0.6277
Epoch 11/100
106/106 [==============================] - 0s 2ms/step - loss: 1.0096 -
accuracy: 0.6343 - val_loss: 1.0514 - val_accuracy: 0.6181
Epoch 12/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9788 -
accuracy: 0.6503 - val_loss: 1.0755 - val_accuracy: 0.6014
Epoch 13/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9528 -
accuracy: 0.6589 - val_loss: 0.9194 - val_accuracy: 0.6754
Epoch 14/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9401 -
accuracy: 0.6672 - val_loss: 0.9615 - val_accuracy: 0.6348
Epoch 15/100
106/106 [==============================] - 0s 2ms/step - loss: 0.9409 -
accuracy: 0.6562 - val_loss: 0.8824 - val_accuracy: 0.6492
Epoch 16/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8896 -
accuracy: 0.6769 - val_loss: 0.8608 - val_accuracy: 0.6563
Epoch 17/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8878 -
accuracy: 0.6885 - val_loss: 0.8843 - val_accuracy: 0.6659
Epoch 18/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8713 -
accuracy: 0.6817 - val_loss: 0.9029 - val_accuracy: 0.6516
Epoch 19/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8398 -
accuracy: 0.7033 - val_loss: 0.8507 - val_accuracy: 0.6874
Epoch 20/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8509 -
accuracy: 0.6980 - val_loss: 0.9350 - val_accuracy: 0.6563
Epoch 21/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8294 -
accuracy: 0.7039 - val_loss: 0.8524 - val_accuracy: 0.6730
Epoch 22/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8214 -
accuracy: 0.7143 - val_loss: 0.8194 - val_accuracy: 0.7064
Epoch 23/100
106/106 [==============================] - 0s 2ms/step - loss: 0.8459 -
accuracy: 0.6941 - val_loss: 0.9308 - val_accuracy: 0.6683
Epoch 24/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7903 -
accuracy: 0.7202 - val_loss: 0.8391 - val_accuracy: 0.7088
```

```
Epoch 25/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7926 -
accuracy: 0.7178 - val_loss: 0.8281 - val_accuracy: 0.7017
Epoch 26/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7696 -
accuracy: 0.7229 - val_loss: 0.8939 - val_accuracy: 0.6683
Epoch 27/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7798 -
accuracy: 0.7232 - val_loss: 0.7928 - val_accuracy: 0.7088
Epoch 28/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7470 -
accuracy: 0.7484 - val_loss: 0.8471 - val_accuracy: 0.6802
Epoch 29/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7194 -
accuracy: 0.7478 - val_loss: 1.0335 - val_accuracy: 0.6396
Epoch 30/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7500 -
accuracy: 0.7359 - val_loss: 0.8066 - val_accuracy: 0.7017
Epoch 31/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7412 -
accuracy: 0.7383 - val_loss: 0.7866 - val_accuracy: 0.7088
Epoch 32/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7032 -
accuracy: 0.7487 - val_loss: 0.8440 - val_accuracy: 0.7184
Epoch 33/100
106/106 [==============================] - 0s 2ms/step - loss: 0.7183 -
accuracy: 0.7487 - val_loss: 0.8362 - val_accuracy: 0.7041
Epoch 34/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6936 -
accuracy: 0.7501 - val_loss: 0.7333 - val_accuracy: 0.7422
Epoch 35/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6869 -
accuracy: 0.7531 - val_loss: 0.7439 - val_accuracy: 0.7208
Epoch 36/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6834 -
accuracy: 0.7570 - val_loss: 0.8807 - val_accuracy: 0.6826
Epoch 37/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6634 -
accuracy: 0.7679 - val_loss: 0.7882 - val_accuracy: 0.7112
Epoch 38/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6525 -
accuracy: 0.7638 - val_loss: 0.7682 - val_accuracy: 0.7184
Epoch 39/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6605 -
accuracy: 0.7682 - val_loss: 0.7572 - val_accuracy: 0.7136
Epoch 40/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6646 -
accuracy: 0.7721 - val_loss: 0.8446 - val_accuracy: 0.6945
```

```
Epoch 41/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6556 -
accuracy: 0.7650 - val_loss: 0.7708 - val_accuracy: 0.7208
Epoch 42/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6149 -
accuracy: 0.7792 - val_loss: 0.8024 - val_accuracy: 0.7136
Epoch 43/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6226 -
accuracy: 0.7795 - val_loss: 0.8187 - val_accuracy: 0.7017
Epoch 44/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6155 -
accuracy: 0.7804 - val_loss: 0.7692 - val_accuracy: 0.7136
Epoch 45/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5937 -
accuracy: 0.7916 - val_loss: 0.7863 - val_accuracy: 0.7160
Epoch 46/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6012 -
accuracy: 0.7762 - val_loss: 0.7488 - val_accuracy: 0.7494
Epoch 47/100
106/106 [==============================] - 0s 2ms/step - loss: 0.6080 -
accuracy: 0.7851 - val_loss: 0.8118 - val_accuracy: 0.7136
Epoch 48/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5767 -
accuracy: 0.8038 - val_loss: 0.7245 - val_accuracy: 0.7422
Epoch 49/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5817 -
accuracy: 0.7905 - val_loss: 0.7551 - val_accuracy: 0.7399
Epoch 50/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5966 -
accuracy: 0.7878 - val_loss: 0.7587 - val_accuracy: 0.7184
Epoch 51/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5658 -
accuracy: 0.8038 - val_loss: 0.7276 - val_accuracy: 0.7494
Epoch 52/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5855 -
accuracy: 0.7896 - val_loss: 0.7341 - val_accuracy: 0.7375
Epoch 53/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5714 -
accuracy: 0.7958 - val_loss: 0.8029 - val_accuracy: 0.7422
Epoch 54/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5552 -
accuracy: 0.8011 - val_loss: 0.7485 - val_accuracy: 0.7351
Epoch 55/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5525 -
accuracy: 0.8011 - val_loss: 0.7582 - val_accuracy: 0.7351
Epoch 56/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5282 -
accuracy: 0.8136 - val_loss: 0.7424 - val_accuracy: 0.7494
```

```
Epoch 57/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5444 -
accuracy: 0.8044 - val_loss: 0.7507 - val_accuracy: 0.7375
Epoch 58/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5290 -
accuracy: 0.8151 - val_loss: 0.7238 - val_accuracy: 0.7446
Epoch 59/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5257 -
accuracy: 0.8103 - val_loss: 0.7366 - val_accuracy: 0.7494
Epoch 60/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5041 -
accuracy: 0.8210 - val_loss: 0.7469 - val_accuracy: 0.7470
Epoch 61/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4868 -
accuracy: 0.8248 - val_loss: 0.7079 - val_accuracy: 0.7589
Epoch 62/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4994 -
accuracy: 0.8204 - val_loss: 0.7511 - val_accuracy: 0.7542
Epoch 63/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5210 -
accuracy: 0.8106 - val_loss: 0.8046 - val_accuracy: 0.7375
Epoch 64/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5181 -
accuracy: 0.8180 - val_loss: 0.7112 - val_accuracy: 0.7613
Epoch 65/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4782 -
accuracy: 0.8337 - val_loss: 0.7532 - val_accuracy: 0.7446
Epoch 66/100
106/106 [==============================] - 0s 2ms/step - loss: 0.5144 -
accuracy: 0.8133 - val_loss: 0.7400 - val_accuracy: 0.7327
Epoch 67/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4964 -
accuracy: 0.8225 - val_loss: 0.7260 - val_accuracy: 0.7470
Epoch 68/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4760 -
accuracy: 0.8334 - val_loss: 0.8091 - val_accuracy: 0.7399
Epoch 69/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4785 -
accuracy: 0.8266 - val_loss: 0.8328 - val_accuracy: 0.7184
Epoch 70/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4559 -
accuracy: 0.8417 - val_loss: 0.7291 - val_accuracy: 0.7399
Epoch 71/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4510 -
accuracy: 0.8376 - val_loss: 0.7749 - val_accuracy: 0.7327
Epoch 72/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4790 -
accuracy: 0.8266 - val_loss: 0.8071 - val_accuracy: 0.7255
```

```
Epoch 73/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4748 -
accuracy: 0.8308 - val_loss: 0.7697 - val_accuracy: 0.7470
Epoch 74/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4415 -
accuracy: 0.8379 - val_loss: 0.7996 - val_accuracy: 0.7351
Epoch 75/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4466 -
accuracy: 0.8414 - val_loss: 0.7536 - val_accuracy: 0.7566
Epoch 76/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4412 -
accuracy: 0.8411 - val_loss: 0.8236 - val_accuracy: 0.7351
Epoch 77/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4497 -
accuracy: 0.8408 - val_loss: 0.8416 - val_accuracy: 0.7422
Epoch 78/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4169 -
accuracy: 0.8530 - val_loss: 0.7746 - val_accuracy: 0.7566
Epoch 79/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4371 -
accuracy: 0.8444 - val_loss: 0.7997 - val_accuracy: 0.7566
Epoch 80/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4350 -
accuracy: 0.8518 - val_loss: 0.7692 - val_accuracy: 0.7518
Epoch 81/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4293 -
accuracy: 0.8456 - val_loss: 0.7435 - val_accuracy: 0.7589
Epoch 82/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4141 -
accuracy: 0.8459 - val_loss: 0.7873 - val_accuracy: 0.7613
Epoch 83/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4216 -
accuracy: 0.8477 - val_loss: 0.8219 - val_accuracy: 0.7351
Epoch 84/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4144 -
accuracy: 0.8560 - val_loss: 0.8143 - val_accuracy: 0.7255
Epoch 85/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4057 -
accuracy: 0.8563 - val_loss: 0.8378 - val_accuracy: 0.7160
Epoch 86/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3977 -
accuracy: 0.8551 - val_loss: 0.8041 - val_accuracy: 0.7375
Epoch 87/100
106/106 [==============================] - 0s 3ms/step - loss: 0.4244 -
accuracy: 0.8438 - val_loss: 0.8381 - val_accuracy: 0.7232
Epoch 88/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4075 -
accuracy: 0.8563 - val_loss: 0.7669 - val_accuracy: 0.7518
```

```
Epoch 89/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3863 -
accuracy: 0.8708 - val_loss: 0.8706 - val_accuracy: 0.7208
Epoch 90/100
106/106 [==============================] - 0s 3ms/step - loss: 0.3755 -
accuracy: 0.8672 - val_loss: 0.7284 - val_accuracy: 0.7804
Epoch 91/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3750 -
accuracy: 0.8717 - val_loss: 0.7706 - val_accuracy: 0.7518
Epoch 92/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3632 -
accuracy: 0.8731 - val_loss: 0.7444 - val_accuracy: 0.7613
Epoch 93/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3641 -
accuracy: 0.8788 - val_loss: 0.8321 - val_accuracy: 0.7327
Epoch 94/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3795 -
accuracy: 0.8651 - val_loss: 0.7839 - val_accuracy: 0.7494
Epoch 95/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4197 -
accuracy: 0.8483 - val_loss: 0.8585 - val_accuracy: 0.7088
Epoch 96/100
106/106 [==============================] - 0s 3ms/step - loss: 0.3604 -
accuracy: 0.8687 - val_loss: 0.7638 - val_accuracy: 0.7494
Epoch 97/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3416 -
accuracy: 0.8803 - val_loss: 0.7728 - val_accuracy: 0.7446
Epoch 98/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3350 -
accuracy: 0.8809 - val_loss: 0.7468 - val_accuracy: 0.7637
Epoch 99/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3377 -
accuracy: 0.8811 - val_loss: 0.8796 - val_accuracy: 0.7327
Epoch 100/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3549 -
accuracy: 0.8699 - val_loss: 0.7957 - val_accuracy: 0.7375
```

Use `model.predict` again and obtain the accuracy value for the learning rate you have chosen.

[27]:
```python
from sklearn.metrics import accuracy_score

# Conduct prediction using new model
predicted_ytest = model.predict(inputs_test)
predicted_test = np.argmax(predicted_ytest, axis=1)

# Print out the new accuracy percentage
accuracy_value = accuracy_score(predicted_test, target_test) * 100
print(f"Test Accuracy = {accuracy_value}")
```

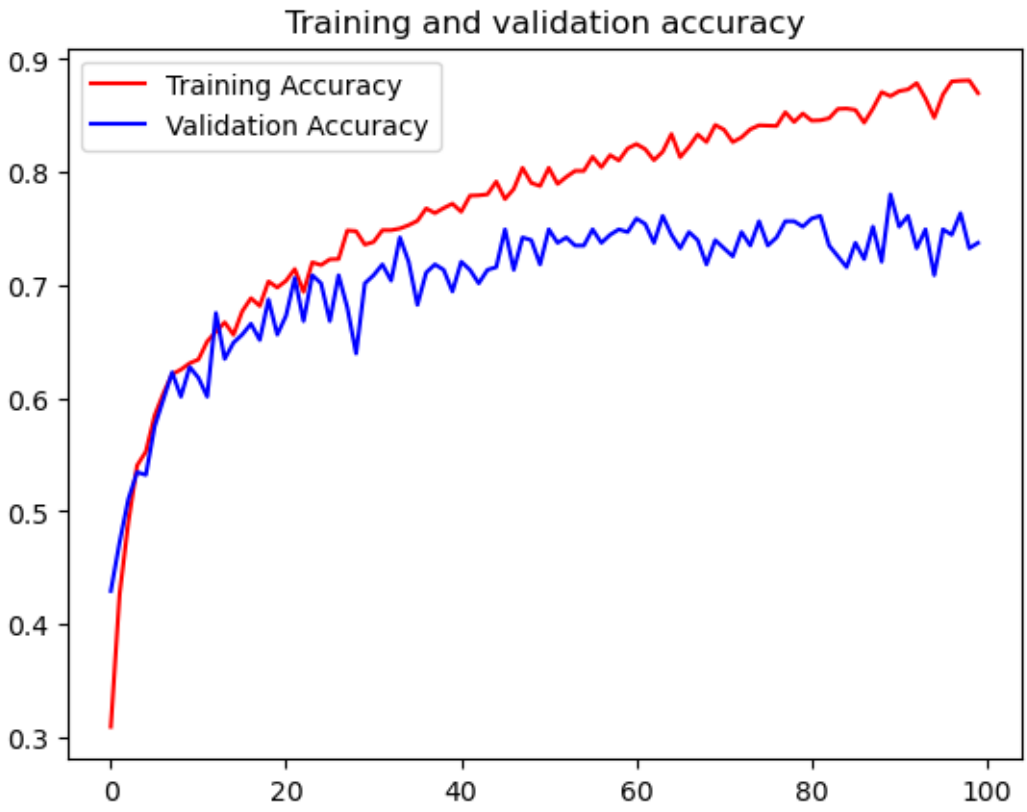
```
13/13 [==============================] - 0s 1ms/step
Test Accuracy = 76.1038961038961
```

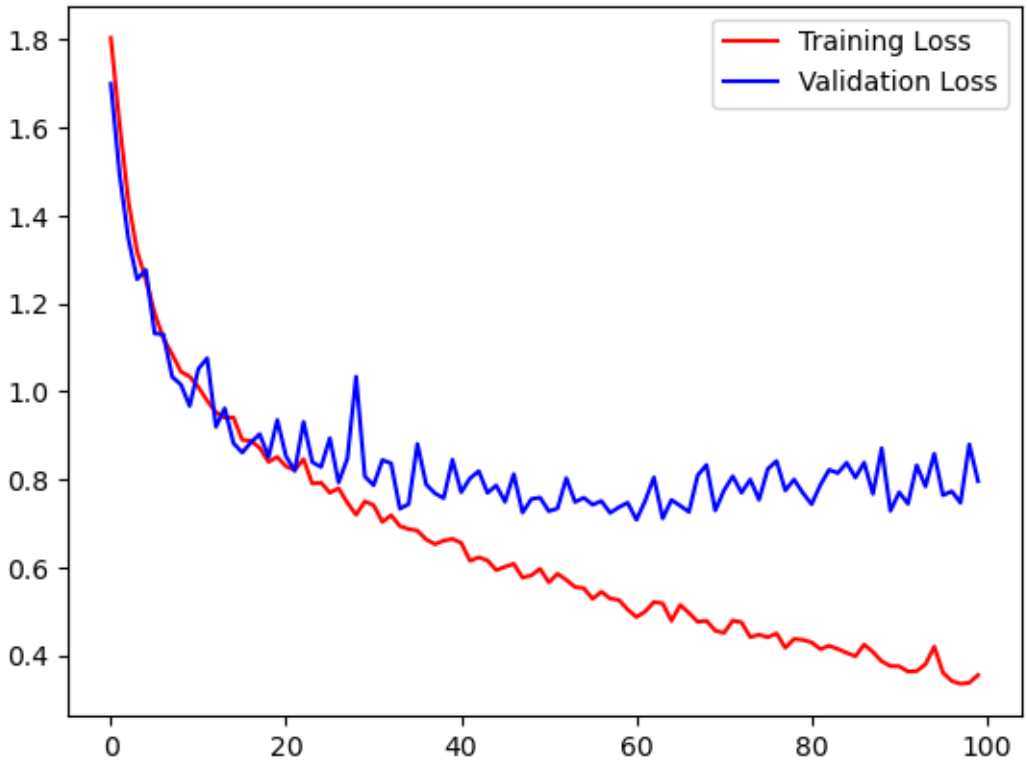
Problem

How does this compare to the accuracy level for the model created in Problem 2 using a learning rate of 0.001? How do you think choosing the learning rate impacts your model?

YourAnswer :

[28]:
```
#-----------------------------------------------------------
# Retrieve results on training and validation datasets for each training epoch
#-----------------------------------------------------------
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs=range(len(acc)) # Get number of epochs

#------------------------------------------------
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot(epochs, acc, 'r', label="Training Accuracy")
plt.plot(epochs, val_acc, 'b', label="Validation Accuracy")
plt.title('Training and validation accuracy')
plt.legend()
plt.show()
print("")

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r', label="Training Loss")
plt.plot(epochs, val_loss, 'b', label="Validation Loss")
plt.legend()
plt.show()
```


Training and validation accuracy
Training Accuracy
Validation Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100


Training Loss
Validation Loss
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
20
40
60
80
100

[ ]:

[ ]: