CNN

November 13, 2021

[1]:
```
import numpy as np
from matplotlib import pyplot as plt
import csv
import tensorflow as tf
import time
```

[2]:
```
tf.keras.backend.clear_session()
```

[3]:
```
Image = np.loadtxt('Q1_Train_Data.csv', delimiter = ',', dtype='str')
```

[4]:
```
Image = Image[1:]
```

[5]:
```
X = [] #All Images
Y = [] #All Emotions corresponding to the images
for i in range(28709):
    H = np.reshape(np.asarray(Image[i][1].split(' '), dtype="float"), (48,48))
    #Image normalization
    max_ele = H.max()
    min_ele = H.min()
    H = (H-min_ele)/(max_ele - min_ele + 0.000000000001)
    X.append(H)
    Y.append(int(Image[i][0]))
```

[6]:
```
Image_valid = np.loadtxt('Q1_Test_Data.csv', delimiter = ',', dtype='str')
Image_valid = Image_valid[1:]
Emotions_valid = Image_valid[:,0]
```

[7]:
```
X_valid = []
Y_valid = []
for i in range(3588):
    H = np.reshape(np.asarray(Image_valid[i][1].split(' '), dtype="float"),␣
 ↪(48,48))
    max_ele = H.max()
    min_ele = H.min()
    H = (H-min_ele)/(max_ele - min_ele + 0.000000000001)
    X_valid.append(H)
    Y_valid.append(Image_valid[i][0])
```

```
X_valid = np.array(X_valid)
X_valid = X_valid.reshape(X_valid.shape[0], 48, 48, 1)
Y_valid = tf.keras.utils.to_categorical(Y_valid, 7)
```

[8]:
```
Image_test = np.loadtxt('Q1_Validation_Data.csv', delimiter = ',', dtype='str')
Image_test = Image_test[1:]
Emotions_test = Image_test[:,0]
```

[47]:
```
X_test = []
Y_test = []
for i in range(3588):
    H = np.reshape(np.asarray(Image_test[i][1].split(' '), dtype="float"),␣
 ↪(48,48))
    max_ele = H.max()
    min_ele = H.min()
    H = (H-min_ele)/(max_ele - min_ele + 0.000000000001)
    X_test.append(H)
    Y_test.append(Image_test[i][0])

X_test = np.array(X_test)
X_test = X_test.reshape(X_test.shape[0], 48, 48, 1)
Y_test = tf.keras.utils.to_categorical(Y_test, 7)
```

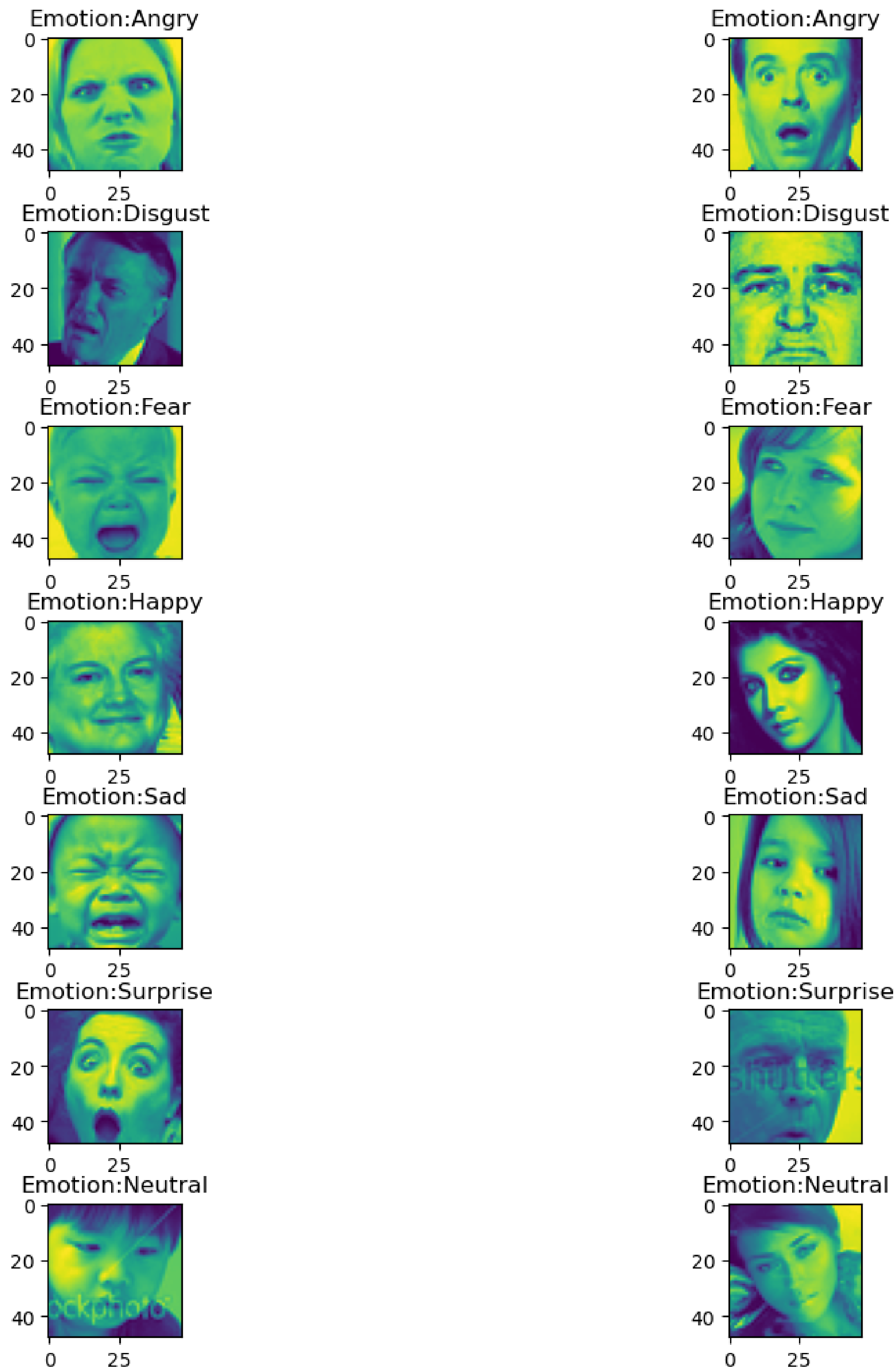
0.0.1 (a) (1 points) Visualization:

Randomly select and visualize 1-2 images per emotion.

[10]:
```
Emotions = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Sad', 5:'Surprise',␣
 ↪6:'Neutral'}
Temp = {}
for i in range(7):
    Temp[Emotions[i]] = [X[x] for x in range(len(X)) if int(Y[x])==i]
plt.figure()
fig, ax = plt.subplots(7,2, figsize=(10,10))
fig.tight_layout()
for i in range(7):
    ax[i, 0].set_title("Emotion:{}".format(Emotions[i]))
    ax[i, 0].imshow(Temp[Emotions[i]][20])
    ax[i, 1].set_title("Emotion:{}".format(Emotions[i]))
    ax[i, 1].imshow(Temp[Emotions[i]][25])

plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```


Emotion:Angry
Emotion:Angry
Emotion:Disgust
Emotion:Disgust
Emotion:Fear
Emotion:Fear
Emotion:Happy
Emotion:Happy
Emotion:Sad
Emotion:Sad
Emotion:Surprise
Emotion:Surprise
Emotion:Neutral
Emotion:Neutral
0
20
40
0
25

0.0.2 (b) (1 points) Data exploration:

Count the number of samples per emotion in the training data.

[11]:
```
import collections
C = collections.Counter(Y)
print(C)
```

```
Counter({3: 7215, 6: 4965, 4: 4830, 2: 4097, 0: 3995, 5: 3171, 1: 436})
```

[12]:
```
X = np.array(X)
X = X.reshape(X.shape[0], 48, 48, 1)
Y = tf.keras.utils.to_categorical(Y, 7)
```

0.0.3 (c) (4 points) Image classification with FNNs:

In this part, you will use a feedforward neural network (FNN) (also called "multilayer perceptron") to perform the emotion classification task. The input of the FNN comprises of all the pixels of the image.

0.0.4 (c.i) (3 points)

Experiment on the validation set with dierent FNN hyper-parameters, e.g. #layers, #nodes per layer, activation function, dropout, weight regularization, etc. For each hyper-parameter combination that you have used, please report the following: (1) emotion classification accuracy on the training and validation sets; (2) running time for training the FNN; (3) # parameters for each FNN. For 2-3 hyper-parameter combinations, please also plot the cross-entropy loss over the number of iterations during training.

Note: If running the FNN takes a long time, you can subsample the input images to a smaller size (e.g., 24 X 24).

0.0.5 Note:

Thoughout the next few FNN models, the total number of parameters have been kept constant ~1.3 million so that it fits in my laptop memory and is trained within a reasonable amount of time.

0.0.6 Model 1:

2 Hidden layers with 500 and 400 neurons and Relu as activation function in each hidden layer. Loss function is cross-entropy and last layers uses softmax for activation and classification. A callback function for early stopping is added (it would stop the execution once accuracy doesnt change for more than 0.002 for atmost 5 iterations)

[13]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=5,␣
↪min_delta=0.002)
```

```python
model1 = tf.keras.Sequential([tf.keras.layers.InputLayer(input_shape=(48,48,1),␣
↪batch_size=None),
                              tf.keras.layers.Flatten(),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Dropout(0.15),
                             tf.keras.layers.Dense(500, activation=tf.nn.relu),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Dropout(0.15),
                             tf.keras.layers.Dense(400, activation=tf.nn.relu),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Dropout(0.15)])
model1.add(tf.keras.layers.Dense(len(C.keys()), activation='softmax'))
model1.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True)
              , optimizer='Adam', metrics=['accuracy'])
print(model1.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 2304)              9216
_________________________________________________________________
dropout (Dropout)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 500)               1152500
_________________________________________________________________
batch_normalization_1 (Batch (None, 500)               2000
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 400)               200400
_________________________________________________________________
batch_normalization_2 (Batch (None, 400)               1600
_________________________________________________________________
dropout_2 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 2807
=================================================================
Total params: 1,368,523
Trainable params: 1,362,115
Non-trainable params: 6,408
_________________________________________________________________
None
```

[14]:
```
startTime = time.time()
history1 = model1.fit(X, Y, epochs = 250, verbose=1, callbacks = [callback])
endTime = time.time()
```

```
Epoch 1/250
898/898 [==============================] - 5s 5ms/step - loss: 1.8384 -
accuracy: 0.3130
Epoch 2/250
898/898 [==============================] - 4s 5ms/step - loss: 1.8021 -
accuracy: 0.3526
Epoch 3/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7859 -
accuracy: 0.3709
Epoch 4/250
898/898 [==============================] - 5s 5ms/step - loss: 1.7749 -
accuracy: 0.3801
Epoch 5/250
898/898 [==============================] - 5s 5ms/step - loss: 1.7653 -
accuracy: 0.3901
Epoch 6/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7569 -
accuracy: 0.3987
Epoch 7/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7465 -
accuracy: 0.4091
Epoch 8/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7355 -
accuracy: 0.4214
Epoch 9/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7292 -
accuracy: 0.4278
Epoch 10/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7214 -
accuracy: 0.4368
Epoch 11/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7137 -
accuracy: 0.4446
Epoch 12/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7070 -
accuracy: 0.4511
Epoch 13/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7038 -
accuracy: 0.4543
Epoch 14/250
898/898 [==============================] - 4s 5ms/step - loss: 1.6978 -
accuracy: 0.4614
Epoch 15/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6914 -
```

```
accuracy: 0.4672
Epoch 16/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6896 -
accuracy: 0.4697
Epoch 17/250
898/898 [==============================] - 4s 5ms/step - loss: 1.6840 -
accuracy: 0.4755
Epoch 18/250
898/898 [==============================] - 4s 5ms/step - loss: 1.6777 -
accuracy: 0.4823
Epoch 19/250
898/898 [==============================] - 4s 5ms/step - loss: 1.6770 -
accuracy: 0.4809
Epoch 20/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6724 -
accuracy: 0.4872
Epoch 21/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6669 -
accuracy: 0.4935
Epoch 22/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6617 -
accuracy: 0.4974
Epoch 23/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6582 -
accuracy: 0.5023
Epoch 24/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6567 -
accuracy: 0.5042
Epoch 25/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6549 -
accuracy: 0.5053
Epoch 26/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6481 -
accuracy: 0.5126
Epoch 27/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6467 -
accuracy: 0.5140
Epoch 28/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6471 -
accuracy: 0.5134
Epoch 29/250
898/898 [==============================] - 4s 5ms/step - loss: 1.6417 -
accuracy: 0.5193
Epoch 30/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6378 -
accuracy: 0.5229
Epoch 31/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6371 -
```

```
accuracy: 0.5241
Epoch 32/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6353 -
accuracy: 0.5253
Epoch 33/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6287 -
accuracy: 0.5320
Epoch 34/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6308 -
accuracy: 0.5302
Epoch 35/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6232 -
accuracy: 0.5386
Epoch 36/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6238 -
accuracy: 0.5365
Epoch 37/250
898/898 [==============================] - 4s 5ms/step - loss: 1.6199 -
accuracy: 0.5407
Epoch 38/250
898/898 [==============================] - 4s 5ms/step - loss: 1.6207 -
accuracy: 0.5404
Epoch 39/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6185 -
accuracy: 0.5419
Epoch 40/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6149 -
accuracy: 0.5462
Epoch 41/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6121 -
accuracy: 0.5492
Epoch 42/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6085 -
accuracy: 0.5534
Epoch 43/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6110 -
accuracy: 0.5496
Epoch 44/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6109 -
accuracy: 0.5502
Epoch 45/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6036 -
accuracy: 0.5582
Epoch 46/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6036 -
accuracy: 0.5582
Epoch 47/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6008 -
```

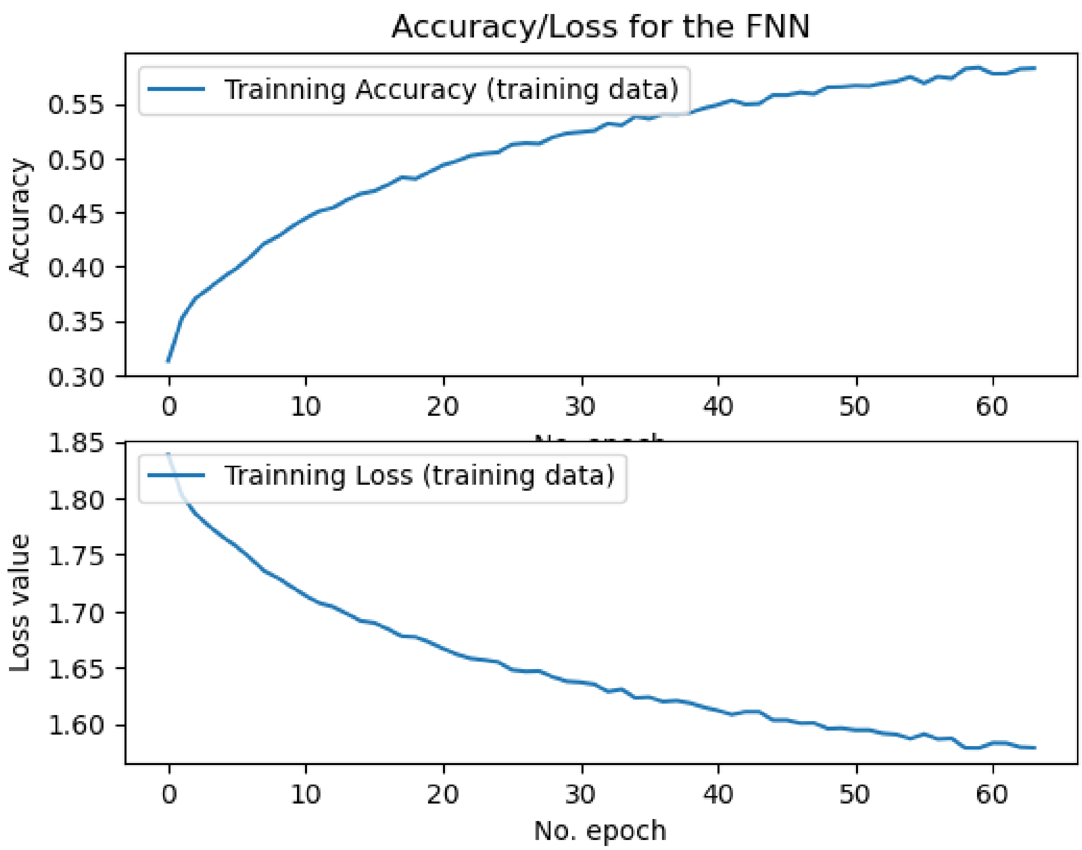
```
accuracy: 0.5606
Epoch 48/250
898/898 [==============================] - 4s 4ms/step - loss: 1.6010 -
accuracy: 0.5593
Epoch 49/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5959 -
accuracy: 0.5654
Epoch 50/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5964 -
accuracy: 0.5657
Epoch 51/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5947 -
accuracy: 0.5668
Epoch 52/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5947 -
accuracy: 0.5664
Epoch 53/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5919 -
accuracy: 0.5691
Epoch 54/250
898/898 [==============================] - 4s 5ms/step - loss: 1.5910 -
accuracy: 0.5708
Epoch 55/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5871 -
accuracy: 0.5749
Epoch 56/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5913 -
accuracy: 0.5692
Epoch 57/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5868 -
accuracy: 0.5751
Epoch 58/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5875 -
accuracy: 0.5736
Epoch 59/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5790 -
accuracy: 0.5825
Epoch 60/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5789 -
accuracy: 0.5836
Epoch 61/250
898/898 [==============================] - 4s 5ms/step - loss: 1.5834 -
accuracy: 0.5778
Epoch 62/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5832 -
accuracy: 0.5780
Epoch 63/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5797 -
```

```
accuracy: 0.5824
Epoch 64/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5791 -
accuracy: 0.5830
```

[15]:

```
# Plot history
plt.figure()
fig, ax = plt.subplots(2,1)
ax[0].plot(history1.history['accuracy'], label='Trainning Accuracy (training␣
↪data)')
ax[1].plot(history1.history['loss'], label='Trainning Loss (training data)')
# ax[0].plot(history1.history['val_accuracy'], label='Validation Accuracy␣
↪(training data)')
# ax[1].plot(history1.history['val_loss'], label='Validation Loss (training␣
↪data)')
ax[0].set_title('Accuracy/Loss for the FNN')
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('No. epoch')
ax[0].legend(loc="upper left")
ax[1].set_ylabel('Loss value')
ax[1].set_xlabel('No. epoch')
ax[1].legend(loc="upper left")
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



[16]:
```
results = model1.evaluate(X_valid, Y_valid)
print("test loss, test acc:", results)
print("Total trainning time: ", (endTime - startTime), "seconds")
```

```
113/113 [==============================] - 0s 2ms/step - loss: 1.6994 -
accuracy: 0.4588
test loss, test acc: [1.6994346380233765, 0.458751380443573]
Total trainning time:  262.07589650154114 seconds
```

0.0.7 Model 2 (FNN) :

Same layers were same as before but activation are now sigmoid functions. Same callback function is utilized from FNN model 1

[17]:
```
model2 = tf.keras.Sequential([tf.keras.layers.InputLayer(input_shape=(48,48,1),
 ↪batch_size=None),
                              tf.keras.layers.Flatten(),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Dropout(0.15),
```

```
                                tf.keras.layers.Dense(500, activation=tf.nn.
↪sigmoid),
                                tf.keras.layers.BatchNormalization(),
                                tf.keras.layers.Dropout(0.15),
                                tf.keras.layers.Dense(400, activation=tf.nn.
↪sigmoid),
                                tf.keras.layers.BatchNormalization(),
                                tf.keras.layers.Dropout(0.15)])
model2.add(tf.keras.layers.Dense(len(C.keys()), activation='softmax'))
model2.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True)
              , optimizer='Adam', metrics=['accuracy'])
print(model2.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
batch_normalization_3 (Batch (None, 2304)              9216
_________________________________________________________________
dropout_3 (Dropout)          (None, 2304)              0
_________________________________________________________________
dense_3 (Dense)              (None, 500)               1152500
_________________________________________________________________
batch_normalization_4 (Batch (None, 500)               2000
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_4 (Dense)              (None, 400)               200400
_________________________________________________________________
batch_normalization_5 (Batch (None, 400)               1600
_________________________________________________________________
dropout_5 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 2807
=================================================================
Total params: 1,368,523
Trainable params: 1,362,115
Non-trainable params: 6,408
_________________________________________________________________
None
```

[18]:
```
startTime = time.time()
history2 = model2.fit(X, Y, epochs = 250, verbose=1, callbacks = [callback])
endTime = time.time()
```

```
Epoch 1/250
```

```
898/898 [==============================] - 4s 5ms/step - loss: 1.8687 -
accuracy: 0.2847
Epoch 2/250
898/898 [==============================] - 4s 4ms/step - loss: 1.8231 -
accuracy: 0.3326
Epoch 3/250
898/898 [==============================] - 4s 4ms/step - loss: 1.8087 -
accuracy: 0.3476
Epoch 4/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7982 -
accuracy: 0.3577
Epoch 5/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7925 -
accuracy: 0.3641
Epoch 6/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7894 -
accuracy: 0.3672
Epoch 7/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7883 -
accuracy: 0.3674
Epoch 8/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7774 -
accuracy: 0.3784
Epoch 9/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7752 -
accuracy: 0.3796
Epoch 10/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7706 -
accuracy: 0.3855
Epoch 11/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7702 -
accuracy: 0.3869
Epoch 12/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7653 -
accuracy: 0.3907
Epoch 13/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7641 -
accuracy: 0.3931
Epoch 14/250
898/898 [==============================] - 5s 6ms/step - loss: 1.7621 -
accuracy: 0.3953
Epoch 15/250
898/898 [==============================] - 6s 6ms/step - loss: 1.7574 -
accuracy: 0.4004
Epoch 16/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7567 -
accuracy: 0.3998
Epoch 17/250
```

```
898/898 [==============================] - 5s 5ms/step - loss: 1.7543 -
accuracy: 0.4020
Epoch 18/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7481 -
accuracy: 0.4099
Epoch 19/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7497 -
accuracy: 0.4075
Epoch 20/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7435 -
accuracy: 0.4157
Epoch 21/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7466 -
accuracy: 0.4107
Epoch 22/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7456 -
accuracy: 0.4118
Epoch 23/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7412 -
accuracy: 0.4174
Epoch 24/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7401 -
accuracy: 0.4183
Epoch 25/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7396 -
accuracy: 0.4174
Epoch 26/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7376 -
accuracy: 0.4209
Epoch 27/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7356 -
accuracy: 0.4215
Epoch 28/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7330 -
accuracy: 0.4245
Epoch 29/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7331 -
accuracy: 0.4226
Epoch 30/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7309 -
accuracy: 0.4254
Epoch 31/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7283 -
accuracy: 0.4282
Epoch 32/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7284 -
accuracy: 0.4296
Epoch 33/250
```

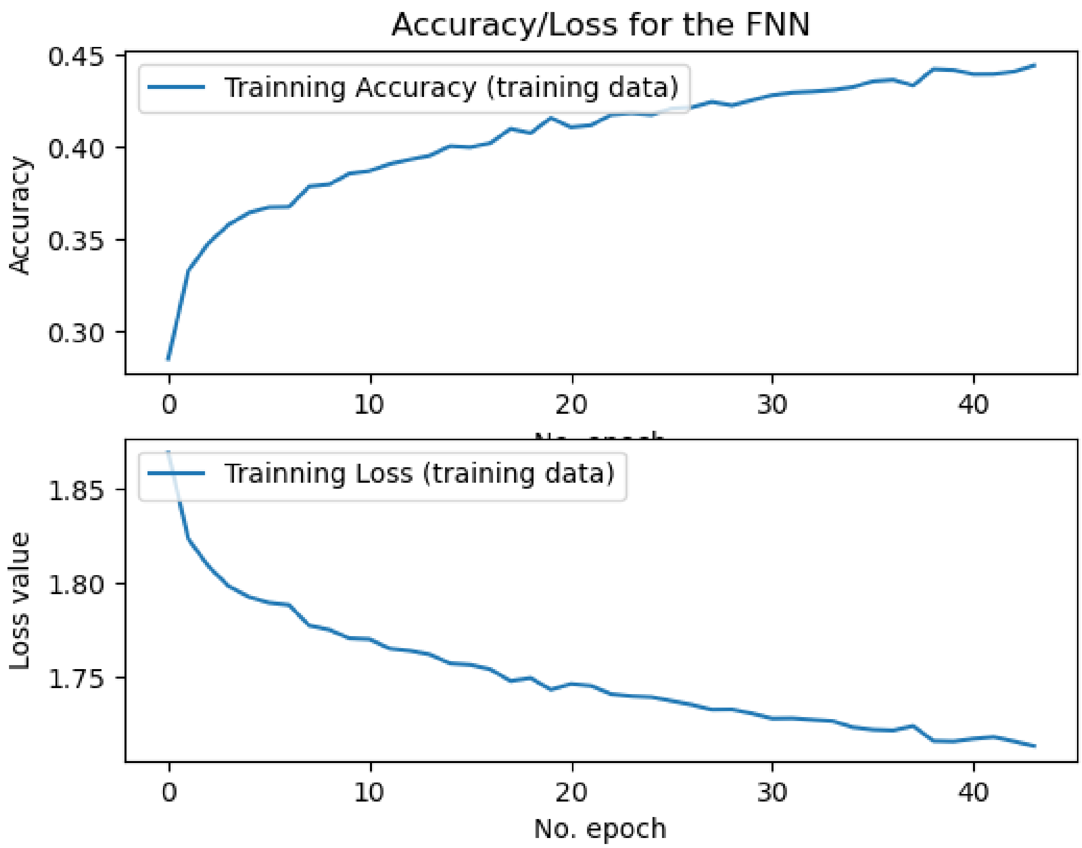
```
898/898 [==============================] - 4s 4ms/step - loss: 1.7276 -
accuracy: 0.4301
Epoch 34/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7269 -
accuracy: 0.4309
Epoch 35/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7237 -
accuracy: 0.4325
Epoch 36/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7224 -
accuracy: 0.4357
Epoch 37/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7220 -
accuracy: 0.4365
Epoch 38/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7243 -
accuracy: 0.4334
Epoch 39/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7165 -
accuracy: 0.4423
Epoch 40/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7162 -
accuracy: 0.4418
Epoch 41/250
898/898 [==============================] - 4s 4ms/step - loss: 1.7176 -
accuracy: 0.4395
Epoch 42/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7186 -
accuracy: 0.4396
Epoch 43/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7163 -
accuracy: 0.4409
Epoch 44/250
898/898 [==============================] - 4s 5ms/step - loss: 1.7138 -
accuracy: 0.4443
```

[19]:

```
# Plot history
plt.figure()
fig, ax = plt.subplots(2,1)
ax[0].plot(history2.history['accuracy'], label='Trainning Accuracy (training␣
↪data)')
ax[1].plot(history2.history['loss'], label='Trainning Loss (training data)')
# ax[0].plot(history1.history['val_accuracy'], label='Validation Accuracy␣
↪(training data)')
# ax[1].plot(history1.history['val_loss'], label='Validation Loss (training␣
↪data)')
ax[0].set_title('Accuracy/Loss for the FNN')
```

```
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('No. epoch')
ax[0].legend(loc="upper left")
ax[1].set_ylabel('Loss value')
ax[1].set_xlabel('No. epoch')
ax[1].legend(loc="upper left")
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



[20]:
```
results = model2.evaluate(X_valid, Y_valid)
print("test loss, test acc:", results)
print("Total trainning time: ", (endTime - startTime), "seconds")
```

```
113/113 [==============================] - 0s 2ms/step - loss: 1.7494 -
accuracy: 0.4044
test loss, test acc: [1.7493516206741333, 0.40440356731414795]
Total trainning time:  184.17514038085938 seconds
```

0.0.8 Model 3 (FNN) :

This time the model has only 1 hidden layer but has increased number of neurons in the hidden layer (keeping the total number of parameters approximately same ~ 1.3 million). Again the same callback functions is utilized for early stopping

[21]:
```
model3 = tf.keras.Sequential([tf.keras.layers.InputLayer(input_shape=(48,48,1),␣
↪batch_size=None),
                              tf.keras.layers.Flatten(),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Dropout(0.15),
                             tf.keras.layers.Dense(600, activation=tf.nn.relu),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Dropout(0.15)])
model3.add(tf.keras.layers.Dense(len(C.keys()), activation='softmax'))
model3.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True)
              , optimizer='Adam', metrics=['accuracy'])
print(model3.summary())
```

```
Model: "sequential_2"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 2304)              0

_________________________________________________________________
batch_normalization_6 (Batch (None, 2304)              9216

_________________________________________________________________
dropout_6 (Dropout)          (None, 2304)              0

_________________________________________________________________
dense_6 (Dense)              (None, 600)               1383000

_________________________________________________________________
batch_normalization_7 (Batch (None, 600)               2400

_________________________________________________________________
dropout_7 (Dropout)          (None, 600)               0

_________________________________________________________________
dense_7 (Dense)              (None, 7)                 4207
=================================================================
Total params: 1,398,823
Trainable params: 1,393,015
Non-trainable params: 5,808

_________________________________________________________________
None
```

[22]:
```
startTime = time.time()
history3 = model3.fit(X, Y, epochs = 250, verbose=1, callbacks = [callback])
endTime = time.time()
```

```
Epoch 1/250
898/898 [==============================] - 3s 4ms/step - loss: 1.8372 -
```

```
accuracy: 0.3158
Epoch 2/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7895 -
accuracy: 0.3661
Epoch 3/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7750 -
accuracy: 0.3817
Epoch 4/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7649 -
accuracy: 0.3915
Epoch 5/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7580 -
accuracy: 0.3990
Epoch 6/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7471 -
accuracy: 0.4093
Epoch 7/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7403 -
accuracy: 0.4182
Epoch 8/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7308 -
accuracy: 0.4267
Epoch 9/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7245 -
accuracy: 0.4350
Epoch 10/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7178 -
accuracy: 0.4400
Epoch 11/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7092 -
accuracy: 0.4507
Epoch 12/250
898/898 [==============================] - 3s 4ms/step - loss: 1.7035 -
accuracy: 0.4558
Epoch 13/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6951 -
accuracy: 0.4630
Epoch 14/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6894 -
accuracy: 0.4699
Epoch 15/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6854 -
accuracy: 0.4736
Epoch 16/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6771 -
accuracy: 0.4830
Epoch 17/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6749 -
```

```
accuracy: 0.4853
Epoch 18/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6700 -
accuracy: 0.4905
Epoch 19/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6659 -
accuracy: 0.4947
Epoch 20/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6645 -
accuracy: 0.4974
Epoch 21/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6581 -
accuracy: 0.5024
Epoch 22/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6562 -
accuracy: 0.5051
Epoch 23/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6512 -
accuracy: 0.5103
Epoch 24/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6453 -
accuracy: 0.5163
Epoch 25/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6443 -
accuracy: 0.5165
Epoch 26/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6407 -
accuracy: 0.5212
Epoch 27/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6371 -
accuracy: 0.5249
Epoch 28/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6361 -
accuracy: 0.5261
Epoch 29/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6348 -
accuracy: 0.5261
Epoch 30/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6324 -
accuracy: 0.5289
Epoch 31/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6306 -
accuracy: 0.5301
Epoch 32/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6279 -
accuracy: 0.5336
Epoch 33/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6254 -
```

```
accuracy: 0.5362
Epoch 34/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6232 -
accuracy: 0.5378
Epoch 35/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6175 -
accuracy: 0.5444
Epoch 36/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6172 -
accuracy: 0.5448
Epoch 37/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6157 -
accuracy: 0.5468
Epoch 38/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6156 -
accuracy: 0.5457
Epoch 39/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6106 -
accuracy: 0.5517
Epoch 40/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6075 -
accuracy: 0.5551
Epoch 41/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6104 -
accuracy: 0.5513
Epoch 42/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6053 -
accuracy: 0.5559
Epoch 43/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6052 -
accuracy: 0.5565
Epoch 44/250
898/898 [==============================] - 3s 4ms/step - loss: 1.6067 -
accuracy: 0.5548
Epoch 45/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5995 -
accuracy: 0.5637
Epoch 46/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5972 -
accuracy: 0.5646
Epoch 47/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5945 -
accuracy: 0.5682
Epoch 48/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5960 -
accuracy: 0.5653
Epoch 49/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5918 -
```

```
accuracy: 0.5703
Epoch 50/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5936 -
accuracy: 0.5682
Epoch 51/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5907 -
accuracy: 0.5717
Epoch 52/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5914 -
accuracy: 0.5707
Epoch 53/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5907 -
accuracy: 0.5714
Epoch 54/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5878 -
accuracy: 0.5730
Epoch 55/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5846 -
accuracy: 0.5773
Epoch 56/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5838 -
accuracy: 0.5791
Epoch 57/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5830 -
accuracy: 0.5795
Epoch 58/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5795 -
accuracy: 0.5822
Epoch 59/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5773 -
accuracy: 0.5850
Epoch 60/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5801 -
accuracy: 0.5824
Epoch 61/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5757 -
accuracy: 0.5862
Epoch 62/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5754 -
accuracy: 0.5864
Epoch 63/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5777 -
accuracy: 0.5846
Epoch 64/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5731 -
accuracy: 0.5892
Epoch 65/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5710 -
```

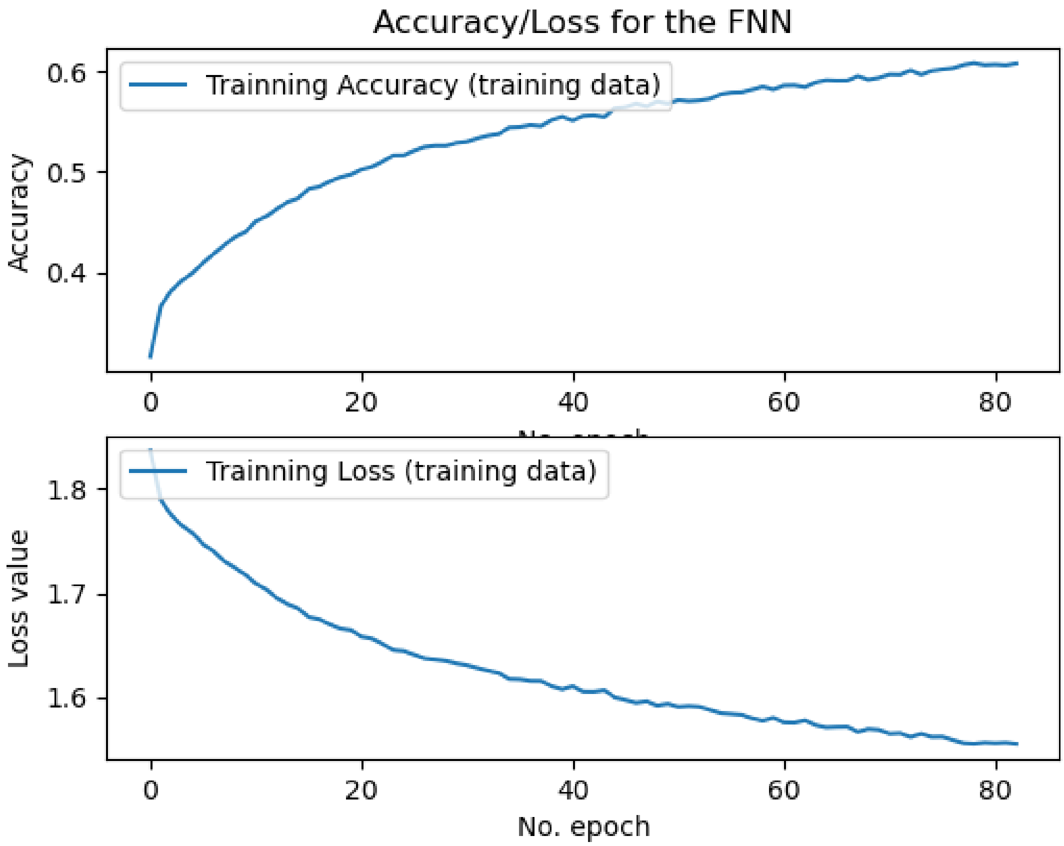
```
accuracy: 0.5913
Epoch 66/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5714 -
accuracy: 0.5909
Epoch 67/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5716 -
accuracy: 0.5910
Epoch 68/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5666 -
accuracy: 0.5956
Epoch 69/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5694 -
accuracy: 0.5919
Epoch 70/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5686 -
accuracy: 0.5939
Epoch 71/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5649 -
accuracy: 0.5972
Epoch 72/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5654 -
accuracy: 0.5970
Epoch 73/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5617 -
accuracy: 0.6014
Epoch 74/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5649 -
accuracy: 0.5971
Epoch 75/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5617 -
accuracy: 0.6009
Epoch 76/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5619 -
accuracy: 0.6023
Epoch 77/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5588 -
accuracy: 0.6034
Epoch 78/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5557 -
accuracy: 0.6067
Epoch 79/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5550 -
accuracy: 0.6086
Epoch 80/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5562 -
accuracy: 0.6066
Epoch 81/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5556 -
```

```
accuracy: 0.6071
Epoch 82/250
898/898 [==============================] - 4s 4ms/step - loss: 1.5561 -
accuracy: 0.6064
Epoch 83/250
898/898 [==============================] - 3s 4ms/step - loss: 1.5550 -
accuracy: 0.6082
```

[23]:

```
# Plot history
plt.figure()
fig, ax = plt.subplots(2,1)
ax[0].plot(history3.history['accuracy'], label='Trainning Accuracy (training␣
↪data)')
ax[1].plot(history3.history['loss'], label='Trainning Loss (training data)')
# ax[0].plot(history1.history['val_accuracy'], label='Validation Accuracy␣
↪(training data)')
# ax[1].plot(history1.history['val_loss'], label='Validation Loss (training␣
↪data)')
ax[0].set_title('Accuracy/Loss for the FNN')
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('No. epoch')
ax[0].legend(loc="upper left")
ax[1].set_ylabel('Loss value')
ax[1].set_xlabel('No. epoch')
ax[1].legend(loc="upper left")
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



[24]:
```
results = model3.evaluate(X_valid, Y_valid)
print("test loss, test acc:", results)
print("Total trainning time: ", (endTime - startTime), "seconds")
```

```
113/113 [==============================] - 0s 2ms/step - loss: 1.6965 -
accuracy: 0.4657
test loss, test acc: [1.6964530944824219, 0.4657190740108490]
Total trainning time:  271.9993920326233 seconds
```

Thus out of the three models, the first model with 1 hidden layers but increased neurons, performed the best on validation set accuracy of 46.57%.

0.0.9 (c.ii) (1 point)

Run the best model that was found based on the validation set from question (c.i) on the testing set. Report the emotion classification accuracy on the testing set.

[25]:
```
result1_final = model3.evaluate(X_test, Y_test)
print("test loss, test acc:", result1_final)
```

```
113/113 [==============================] - 0s 2ms/step - loss: 1.7050 -
accuracy: 0.4543
```

```
test loss, test acc: [1.705011248588562, 0.45429208874702454]
```

0.0.10 (d) (4 points) Image classification with CNNs:

In this part, you will use a convolutional neural network (CNN) to perform the emotion classification task.

0.0.11 (d.i) (3 points)

Experiment on the validation set with different CNN hyper-parameters, e.g. #layers, filter size, stride size, activation function, dropout, weight regularization, etc. For each hyper-parameter combination that you have used, please report the following: (1) emotion classification accuracy on the training and validation sets; (2) running time for training the FNN; (3) # parameters for each CNN. How do these metrics compare to the FNN?

0.0.12 Note:

In the next few CNN, the number of parameters are kept approximately same as what was for FNNs (1.3 million). Along with the easy of execution, this was also done to compare the accuracies directly, keeping the total number of parameters constant.

0.0.13 Model 1:

The first CNN model has 64 filter Conv layer of kernel size 3, then a max pooling layer of size 2 followed by another Conv layer of 32 filters of kernel size 3 and max pooling layer of size 2. Both the conv layers had activation function of Relu. Then comes the fully connected layer with two hidden layers 375 and 350 neurons respectively, with activation function of Relu.

[26]:
```
model4 = tf.keras.Sequential([tf.keras.layers.Conv2D(64, (3, 3),␣
 ↪activation='relu', input_shape=(48,48,1)),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),
                              tf.keras.layers.MaxPooling2D((2, 2)),
                              tf.keras.layers.Conv2D(32, (3, 3),␣
 ↪activation='relu'),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),
                              tf.keras.layers.MaxPooling2D((2, 2)),
                              tf.keras.layers.Flatten(),
                              tf.keras.layers.Dense(375, activation=tf.nn.relu),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),
                              tf.keras.layers.Dense(350, activation=tf.nn.relu),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15)])
model4.add(tf.keras.layers.Dense(len(C.keys()), activation='softmax'))
model4.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True)
               , optimizer='Adam', metrics=['accuracy'])
print(model4.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 64)        640
_________________________________________________________________
batch_normalization_8 (Batch (None, 46, 46, 64)        256
_________________________________________________________________
dropout_8 (Dropout)          (None, 46, 46, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 32)        18464
_________________________________________________________________
batch_normalization_9 (Batch (None, 21, 21, 32)        128
_________________________________________________________________
dropout_9 (Dropout)          (None, 21, 21, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_8 (Dense)              (None, 375)               1200375
_________________________________________________________________
batch_normalization_10 (Batc (None, 375)               1500
_________________________________________________________________
dropout_10 (Dropout)         (None, 375)               0
_________________________________________________________________
dense_9 (Dense)              (None, 350)               131600
_________________________________________________________________
batch_normalization_11 (Batc (None, 350)               1400
_________________________________________________________________
dropout_11 (Dropout)         (None, 350)               0
_________________________________________________________________
dense_10 (Dense)             (None, 7)                 2457
=================================================================
Total params: 1,356,820
Trainable params: 1,355,178
Non-trainable params: 1,642
_________________________________________________________________
None
```

[27]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=7,␣
 ↪min_delta=0.002)
startTime = time.time()
history4 = model4.fit(X, Y, epochs = 250, verbose=1, callbacks = [callback])
endTime = time.time()
```

```
Epoch 1/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7995 -
accuracy: 0.3548
Epoch 2/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7378 -
accuracy: 0.4185
Epoch 3/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7092 -
accuracy: 0.4470
Epoch 4/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6868 -
accuracy: 0.4713
Epoch 5/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6698 -
accuracy: 0.4890
Epoch 6/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6541 -
accuracy: 0.5060
Epoch 7/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6447 -
accuracy: 0.5148
Epoch 8/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6404 -
accuracy: 0.5188
Epoch 9/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6290 -
accuracy: 0.5307
Epoch 10/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6087 -
accuracy: 0.5524
Epoch 11/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5992 -
accuracy: 0.5611
Epoch 12/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5938 -
accuracy: 0.5678
Epoch 13/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5827 -
accuracy: 0.5782
Epoch 14/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5735 -
accuracy: 0.5881
Epoch 15/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5635 -
accuracy: 0.5974
Epoch 16/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5534 -
accuracy: 0.6072: 0s - loss: 1.5538 - accuracy:
```

```
Epoch 17/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5475 -
accuracy: 0.6147
Epoch 18/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5460 -
accuracy: 0.6159
Epoch 19/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5316 -
accuracy: 0.6302
Epoch 20/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5242 -
accuracy: 0.6378
Epoch 21/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5155 -
accuracy: 0.6462
Epoch 22/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5177 -
accuracy: 0.6444
Epoch 23/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5103 -
accuracy: 0.6521
Epoch 24/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4959 -
accuracy: 0.6667
Epoch 25/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4937 -
accuracy: 0.6700
Epoch 26/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4921 -
accuracy: 0.6703
Epoch 27/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4821 -
accuracy: 0.6817
Epoch 28/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4798 -
accuracy: 0.6831
Epoch 29/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4803 -
accuracy: 0.6817
Epoch 30/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4788 -
accuracy: 0.6841
Epoch 31/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4743 -
accuracy: 0.6892
Epoch 32/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4709 -
accuracy: 0.6925
```

```
Epoch 33/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4657 -
accuracy: 0.6979
Epoch 34/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4722 -
accuracy: 0.6901
Epoch 35/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4643 -
accuracy: 0.6991
Epoch 36/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4570 -
accuracy: 0.7057
Epoch 37/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4530 -
accuracy: 0.7102
Epoch 38/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4468 -
accuracy: 0.7165
Epoch 39/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4440 -
accuracy: 0.7195
Epoch 40/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4489 -
accuracy: 0.7146
Epoch 41/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4503 -
accuracy: 0.7135
Epoch 42/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4457 -
accuracy: 0.7184
Epoch 43/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4342 -
accuracy: 0.7290
Epoch 44/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4289 -
accuracy: 0.7344
Epoch 45/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4288 -
accuracy: 0.7345
Epoch 46/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4299 -
accuracy: 0.7331
Epoch 47/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4204 -
accuracy: 0.7435
Epoch 48/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4276 -
accuracy: 0.7358
```

```
Epoch 49/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4226 -
accuracy: 0.7410
Epoch 50/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4134 -
accuracy: 0.7504
Epoch 51/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4136 -
accuracy: 0.7500
Epoch 52/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4147 -
accuracy: 0.7489
Epoch 53/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4170 -
accuracy: 0.7459
Epoch 54/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4051 -
accuracy: 0.7588
Epoch 55/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4075 -
accuracy: 0.7563
Epoch 56/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3996 -
accuracy: 0.7640
Epoch 57/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4005 -
accuracy: 0.7628
Epoch 58/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4032 -
accuracy: 0.7606
Epoch 59/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3989 -
accuracy: 0.7652
Epoch 60/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3987 -
accuracy: 0.7652
Epoch 61/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4017 -
accuracy: 0.7625
Epoch 62/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4012 -
accuracy: 0.7623
Epoch 63/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3914 -
accuracy: 0.7730
Epoch 64/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3879 -
accuracy: 0.7770
```

```
Epoch 65/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3902 -
accuracy: 0.7738
Epoch 66/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3840 -
accuracy: 0.7798
Epoch 67/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3884 -
accuracy: 0.7754
Epoch 68/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3917 -
accuracy: 0.7727
Epoch 69/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3846 -
accuracy: 0.7794
Epoch 70/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3953 -
accuracy: 0.7687
Epoch 71/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3890 -
accuracy: 0.7747
Epoch 72/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3861 -
accuracy: 0.7777
Epoch 73/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3800 -
accuracy: 0.7833
Epoch 74/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3749 -
accuracy: 0.7896
Epoch 75/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3875 -
accuracy: 0.7757
Epoch 76/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3701 -
accuracy: 0.7940
Epoch 77/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3749 -
accuracy: 0.7892
Epoch 78/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3754 -
accuracy: 0.7887
Epoch 79/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3678 -
accuracy: 0.7964
Epoch 80/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3652 -
accuracy: 0.7990
```

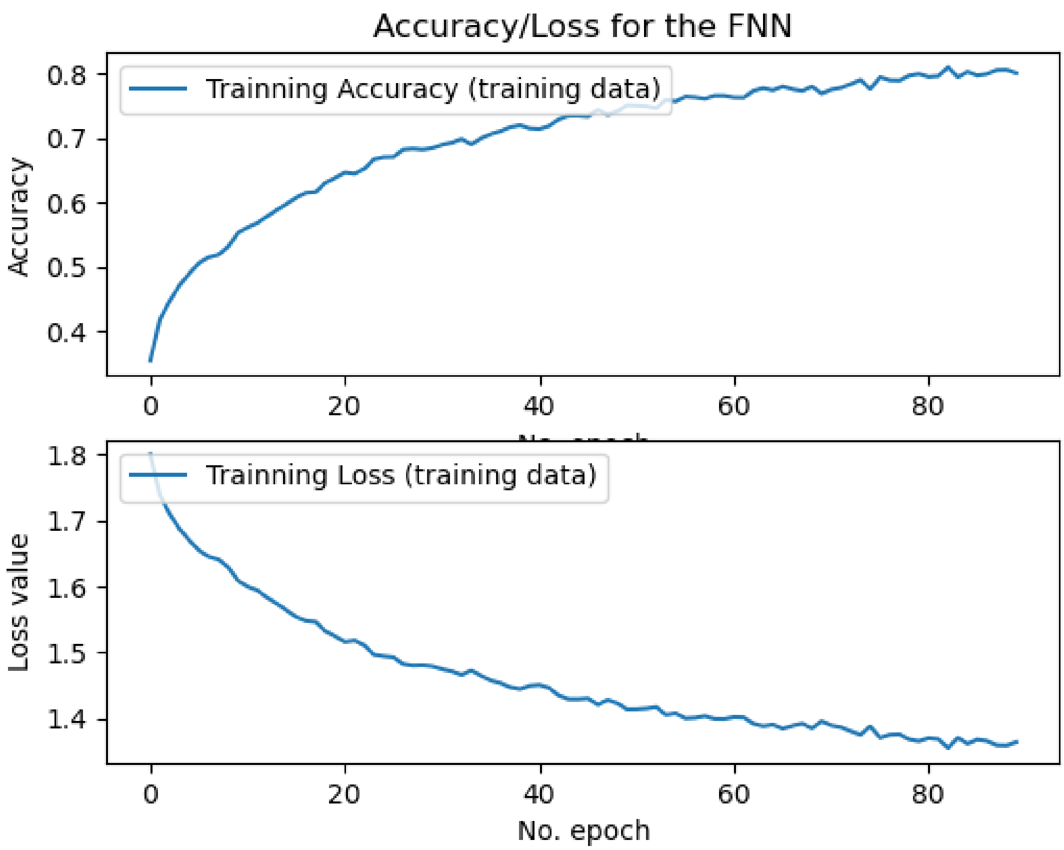
```
Epoch 81/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3699 -
accuracy: 0.7944
Epoch 82/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3682 -
accuracy: 0.7956
Epoch 83/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3549 -
accuracy: 0.8097
Epoch 84/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3701 -
accuracy: 0.7939
Epoch 85/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3613 -
accuracy: 0.8025
Epoch 86/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3676 -
accuracy: 0.7965
Epoch 87/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3653 -
accuracy: 0.7990
Epoch 88/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3589 -
accuracy: 0.8051
Epoch 89/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3586 -
accuracy: 0.8054
Epoch 90/250
898/898 [==============================] - 7s 8ms/step - loss: 1.3636 -
accuracy: 0.8004
```

[28]:

```
# Plot history
plt.figure()
fig, ax = plt.subplots(2,1)
ax[0].plot(history4.history['accuracy'], label='Trainning Accuracy (training␣
↪data)')
ax[1].plot(history4.history['loss'], label='Trainning Loss (training data)')
# ax[0].plot(history1.history['val_accuracy'], label='Validation Accuracy␣
↪(training data)')
# ax[1].plot(history1.history['val_loss'], label='Validation Loss (training␣
↪data)')
ax[0].set_title('Accuracy/Loss for the FNN')
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('No. epoch')
ax[0].legend(loc="upper left")
ax[1].set_ylabel('Loss value')
ax[1].set_xlabel('No. epoch')
```

```
ax[1].legend(loc="upper left")
plt.show()
```

<Figure size 640x480 with 0 Axes>



[29]:
```
results = model4.evaluate(X_valid, Y_valid)
print("test loss, test acc:", results)
print("Total trainning time: ", (endTime - startTime), "seconds")
```

```
113/113 [==============================] - 0s 3ms/step - loss: 1.6243 -
accuracy: 0.5340
test loss, test acc: [1.6243261098861694, 0.5340022444725037]
Total trainning time:  649.8445291519165 seconds
```

0.0.14 Model 2 (CNN) :

The second CNN model has 64 filter Conv layer of kernel size 3, then a max pooling layer of size 2 followed by another Conv layer of 32 filters of kernel size 3 and max pooling layer of size 2. Both the conv layers had activation function of Relu. Then comes the fully connected layer with one hidden layers 425 neurons, with activation function of sigmoid.

[30]:
```
model5 = tf.keras.Sequential([tf.keras.layers.Conv2D(64, (3, 3),␣
↪activation='relu', input_shape=(48,48,1)),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),
                              tf.keras.layers.MaxPooling2D((2, 2)),
                              tf.keras.layers.Conv2D(32, (3, 3),␣
↪activation='relu'),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),
                              tf.keras.layers.MaxPooling2D((2, 2)),
                              tf.keras.layers.Flatten(),
                              tf.keras.layers.Dense(425, activation=tf.nn.
↪sigmoid),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),])
model5.add(tf.keras.layers.Dense(len(C.keys()), activation='softmax'))
model5.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True)
              , optimizer='Adam', metrics=['accuracy'])
print(model5.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 46, 46, 64)        640
_________________________________________________________________
batch_normalization_12 (Batc (None, 46, 46, 64)        256
_________________________________________________________________
dropout_12 (Dropout)         (None, 46, 46, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 21, 32)        18464
_________________________________________________________________
batch_normalization_13 (Batc (None, 21, 21, 32)        128
_________________________________________________________________
dropout_13 (Dropout)         (None, 21, 21, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_11 (Dense)             (None, 425)               1360425
_________________________________________________________________
batch_normalization_14 (Batc (None, 425)               1700
_________________________________________________________________
dropout_14 (Dropout)         (None, 425)               0
```

```
_________________________________________________________________
dense_12 (Dense)             (None, 7)                 2982
=================================================================
Total params: 1,384,595
Trainable params: 1,383,553
Non-trainable params: 1,042
_________________________________________________________________
None
```

[31]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=7,␣
 ↪min_delta=0.002)
startTime = time.time()
history5 = model5.fit(X, Y, epochs = 250, verbose=1, callbacks = [callback])
endTime = time.time()
```

```
Epoch 1/250
898/898 [==============================] - 7s 7ms/step - loss: 1.7729 -
accuracy: 0.3818
Epoch 2/250
898/898 [==============================] - 7s 7ms/step - loss: 1.7016 -
accuracy: 0.4537
Epoch 3/250
898/898 [==============================] - 7s 7ms/step - loss: 1.6736 -
accuracy: 0.4849
Epoch 4/250
898/898 [==============================] - 7s 7ms/step - loss: 1.6491 -
accuracy: 0.5105
Epoch 5/250
898/898 [==============================] - 7s 7ms/step - loss: 1.6297 -
accuracy: 0.5311
Epoch 6/250
898/898 [==============================] - 7s 7ms/step - loss: 1.6122 -
accuracy: 0.5482
Epoch 7/250
898/898 [==============================] - 7s 7ms/step - loss: 1.5938 -
accuracy: 0.5678: 0s - loss: 1.5928
Epoch 8/250
898/898 [==============================] - 7s 7ms/step - loss: 1.5762 -
accuracy: 0.5867
Epoch 9/250
898/898 [==============================] - 7s 7ms/step - loss: 1.5625 -
accuracy: 0.6010
Epoch 10/250
898/898 [==============================] - 7s 7ms/step - loss: 1.5494 -
accuracy: 0.6142
Epoch 11/250
898/898 [==============================] - 7s 7ms/step - loss: 1.5403 -
accuracy: 0.6239
```

```
Epoch 12/250
898/898 [==============================] - 7s 7ms/step - loss: 1.5261 -
accuracy: 0.6381
Epoch 13/250
898/898 [==============================] - 7s 7ms/step - loss: 1.5187 -
accuracy: 0.6457
Epoch 14/250
898/898 [==============================] - 7s 7ms/step - loss: 1.5095 -
accuracy: 0.6552
Epoch 15/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4972 -
accuracy: 0.6673
Epoch 16/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4918 -
accuracy: 0.6725
Epoch 17/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4844 -
accuracy: 0.6802
Epoch 18/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4757 -
accuracy: 0.6890
Epoch 19/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4701 -
accuracy: 0.6953
Epoch 20/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4679 -
accuracy: 0.6976
Epoch 21/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4583 -
accuracy: 0.7069
Epoch 22/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4542 -
accuracy: 0.7111
Epoch 23/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4526 -
accuracy: 0.7120
Epoch 24/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4432 -
accuracy: 0.7206: 0s - l
Epoch 25/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4405 -
accuracy: 0.7252:
Epoch 26/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4355 -
accuracy: 0.7300
Epoch 27/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4300 -
accuracy: 0.7358
```

```
Epoch 28/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4238 -
accuracy: 0.7419
Epoch 29/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4219 -
accuracy: 0.7447
Epoch 30/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4252 -
accuracy: 0.7402
Epoch 31/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4203 -
accuracy: 0.7450
Epoch 32/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4153 -
accuracy: 0.7493
Epoch 33/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4150 -
accuracy: 0.7504
Epoch 34/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4117 -
accuracy: 0.7532
Epoch 35/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4057 -
accuracy: 0.7584
Epoch 36/250
898/898 [==============================] - 7s 7ms/step - loss: 1.4029 -
accuracy: 0.7612
Epoch 37/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3998 -
accuracy: 0.7654
Epoch 38/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3949 -
accuracy: 0.7701
Epoch 39/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3918 -
accuracy: 0.7728
Epoch 40/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3934 -
accuracy: 0.7723
Epoch 41/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3901 -
accuracy: 0.7748
Epoch 42/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3895 -
accuracy: 0.7759
Epoch 43/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3865 -
accuracy: 0.7784
```

```
Epoch 44/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3850 -
accuracy: 0.7791
Epoch 45/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3834 -
accuracy: 0.7812
Epoch 46/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3783 -
accuracy: 0.7865
Epoch 47/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3774 -
accuracy: 0.7872
Epoch 48/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3732 -
accuracy: 0.7924
Epoch 49/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3734 -
accuracy: 0.7916
Epoch 50/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3711 -
accuracy: 0.7942
Epoch 51/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3711 -
accuracy: 0.7938
Epoch 52/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3681 -
accuracy: 0.7968
Epoch 53/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3660 -
accuracy: 0.7999
Epoch 54/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3693 -
accuracy: 0.7952
Epoch 55/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3653 -
accuracy: 0.8002
Epoch 56/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3594 -
accuracy: 0.8051
Epoch 57/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3611 -
accuracy: 0.8047
Epoch 58/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3649 -
accuracy: 0.7999
Epoch 59/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3581 -
accuracy: 0.8061
```

```
Epoch 60/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3550 -
accuracy: 0.8097
Epoch 61/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3530 -
accuracy: 0.8121
Epoch 62/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3488 -
accuracy: 0.8173
Epoch 63/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3497 -
accuracy: 0.8147
Epoch 64/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3506 -
accuracy: 0.8148
Epoch 65/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3512 -
accuracy: 0.8137
Epoch 66/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3483 -
accuracy: 0.8172
Epoch 67/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3488 -
accuracy: 0.8158
Epoch 68/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3419 -
accuracy: 0.8225
Epoch 69/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3479 -
accuracy: 0.8173
Epoch 70/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3429 -
accuracy: 0.8215
Epoch 71/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3432 -
accuracy: 0.8226
Epoch 72/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3441 -
accuracy: 0.8211
Epoch 73/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3419 -
accuracy: 0.8225
Epoch 74/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3389 -
accuracy: 0.8261
Epoch 75/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3358 -
accuracy: 0.8293
```

```
Epoch 76/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3376 -
accuracy: 0.8264
Epoch 77/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3320 -
accuracy: 0.8328
Epoch 78/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3302 -
accuracy: 0.8347
Epoch 79/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3312 -
accuracy: 0.8339
Epoch 80/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3324 -
accuracy: 0.8322
Epoch 81/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3330 -
accuracy: 0.8319
Epoch 82/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3306 -
accuracy: 0.8341
Epoch 83/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3271 -
accuracy: 0.8381
Epoch 84/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3281 -
accuracy: 0.8363
Epoch 85/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3269 -
accuracy: 0.8378
Epoch 86/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3229 -
accuracy: 0.8417
Epoch 87/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3207 -
accuracy: 0.8441
Epoch 88/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3219 -
accuracy: 0.8425
Epoch 89/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3233 -
accuracy: 0.8410
Epoch 90/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3232 -
accuracy: 0.8418
Epoch 91/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3183 -
accuracy: 0.8460
```

```
Epoch 92/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3202 -
accuracy: 0.8447
Epoch 93/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3165 -
accuracy: 0.8486
Epoch 94/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3166 -
accuracy: 0.8482
Epoch 95/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3162 -
accuracy: 0.8485
Epoch 96/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3146 -
accuracy: 0.8499
Epoch 97/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3167 -
accuracy: 0.8483
Epoch 98/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3139 -
accuracy: 0.8517
Epoch 99/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3128 -
accuracy: 0.8518
Epoch 100/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3091 -
accuracy: 0.8555
Epoch 101/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3113 -
accuracy: 0.8530
Epoch 102/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3109 -
accuracy: 0.8537
Epoch 103/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3079 -
accuracy: 0.8571
Epoch 104/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3091 -
accuracy: 0.8553
Epoch 105/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3127 -
accuracy: 0.8514
Epoch 106/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3079 -
accuracy: 0.8565
Epoch 107/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3036 -
accuracy: 0.8618
```

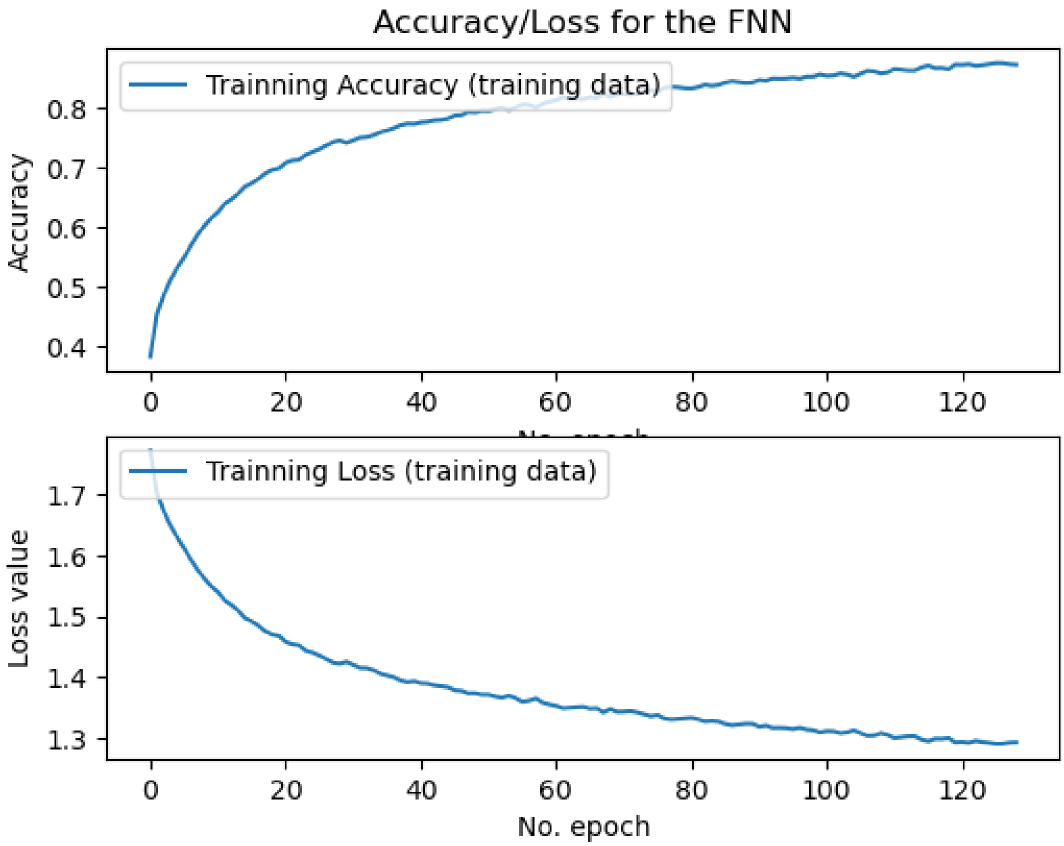
```
Epoch 108/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3041 -
accuracy: 0.8606
Epoch 109/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3079 -
accuracy: 0.8571
Epoch 110/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3049 -
accuracy: 0.8590
Epoch 111/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2999 -
accuracy: 0.8647
Epoch 112/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3012 -
accuracy: 0.8633
Epoch 113/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3026 -
accuracy: 0.8622
Epoch 114/250
898/898 [==============================] - 7s 7ms/step - loss: 1.3029 -
accuracy: 0.8620
Epoch 115/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2973 -
accuracy: 0.8675
Epoch 116/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2945 -
accuracy: 0.8705
Epoch 117/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2986 -
accuracy: 0.8661
Epoch 118/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2984 -
accuracy: 0.8663
Epoch 119/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2998 -
accuracy: 0.8643
Epoch 120/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2927 -
accuracy: 0.8720
Epoch 121/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2933 -
accuracy: 0.8715
Epoch 122/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2921 -
accuracy: 0.8727
Epoch 123/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2954 -
accuracy: 0.8698
```

```
Epoch 124/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2931 -
accuracy: 0.8711
Epoch 125/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2921 -
accuracy: 0.8728
Epoch 126/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2901 -
accuracy: 0.8745
Epoch 127/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2905 -
accuracy: 0.8744
Epoch 128/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2923 -
accuracy: 0.8726
Epoch 129/250
898/898 [==============================] - 7s 7ms/step - loss: 1.2927 -
accuracy: 0.8719
```

[32]:

```
# Plot history
plt.figure()
fig, ax = plt.subplots(2,1)
ax[0].plot(history5.history['accuracy'], label='Trainning Accuracy (training␣
↪data)')
ax[1].plot(history5.history['loss'], label='Trainning Loss (training data)')
# ax[0].plot(history1.history['val_accuracy'], label='Validation Accuracy␣
↪(training data)')
# ax[1].plot(history1.history['val_loss'], label='Validation Loss (training␣
↪data)')
ax[0].set_title('Accuracy/Loss for the FNN')
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('No. epoch')
ax[0].legend(loc="upper left")
ax[1].set_ylabel('Loss value')
ax[1].set_xlabel('No. epoch')
ax[1].legend(loc="upper left")
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



[33]:
```
results = model5.evaluate(X_valid, Y_valid)
print("test loss, test acc:", results)
print("Total trainning time: ", (endTime - startTime), "seconds")
```

```
113/113 [==============================] - 0s 3ms/step - loss: 1.6244 -
accuracy: 0.5354
test loss, test acc: [1.6244301795959473, 0.5353957414627075]
Total trainning time:  849.2923784255981 seconds
```

0.0.15 Model 3 (CNN) :

The third CNN model has 64 filter Conv layer of kernel size 3, then a max pooling layer of size 2 followed by another Conv layer of 32 filters of kernel size 3 and max pooling layer of size 2. Both the conv layers had activation function of Relu. Then comes the fully connected layer with two hidden layers 450 and 350 neurons respectively, with activation function of sigmoid.)

[34]:
```
model6 = tf.keras.Sequential([tf.keras.layers.Conv2D(64, (5, 5),␣
↪activation='relu', input_shape=(48,48,1)),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),
```

```
                              tf.keras.layers.MaxPooling2D((2, 2)),
                              tf.keras.layers.Conv2D(32, (5, 5),␣
 ↪activation='relu'),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),
                              tf.keras.layers.MaxPooling2D((2, 2)),
                              tf.keras.layers.Flatten(),
                              tf.keras.layers.Dense(450, activation=tf.nn.
 ↪sigmoid),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15),
                              tf.keras.layers.Dense(350, activation=tf.nn.
 ↪sigmoid),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Dropout(0.15)])
model6.add(tf.keras.layers.Dense(len(C.keys()), activation='softmax'))
model6.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True)
              , optimizer='Adam', metrics=['accuracy'])
print(model6.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 44, 44, 64)        1664
_________________________________________________________________
batch_normalization_15 (Batc (None, 44, 44, 64)        256
_________________________________________________________________
dropout_15 (Dropout)         (None, 44, 44, 64)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 18, 18, 32)        51232
_________________________________________________________________
batch_normalization_16 (Batc (None, 18, 18, 32)        128
_________________________________________________________________
dropout_16 (Dropout)         (None, 18, 18, 32)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 9, 9, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 2592)              0
_________________________________________________________________
dense_13 (Dense)             (None, 450)               1166850
_________________________________________________________________
batch_normalization_17 (Batc (None, 450)               1800
_________________________________________________________________
dropout_17 (Dropout)         (None, 450)               0
```

```
_________________________________________________________________
dense_14 (Dense)             (None, 350)               157850
_________________________________________________________________
batch_normalization_18 (Batc (None, 350)               1400
_________________________________________________________________
dropout_18 (Dropout)         (None, 350)               0
_________________________________________________________________
dense_15 (Dense)             (None, 7)                 2457
=================================================================
Total params: 1,383,637
Trainable params: 1,381,845
Non-trainable params: 1,792
_________________________________________________________________
None
```

[35]:

```
callback = tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=7,␣
 ↪min_delta=0.002)
startTime = time.time()
history6 = model6.fit(X, Y, epochs = 250, verbose=1, callbacks = [callback])
endTime = time.time()
```

```
Epoch 1/250
898/898 [==============================] - 7s 8ms/step - loss: 1.8277 -
accuracy: 0.3281
Epoch 2/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7813 -
accuracy: 0.3750
Epoch 3/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7630 -
accuracy: 0.3946
Epoch 4/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7567 -
accuracy: 0.4002
Epoch 5/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7425 -
accuracy: 0.4139
Epoch 6/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7258 -
accuracy: 0.4322
Epoch 7/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7191 -
accuracy: 0.4375
Epoch 8/250
898/898 [==============================] - 7s 8ms/step - loss: 1.7021 -
accuracy: 0.4566
Epoch 9/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6894 -
accuracy: 0.4691
```

```
Epoch 10/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6834 -
accuracy: 0.4754
Epoch 11/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6740 -
accuracy: 0.4850
Epoch 12/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6629 -
accuracy: 0.4955
Epoch 13/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6542 -
accuracy: 0.5044
Epoch 14/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6502 -
accuracy: 0.5086
Epoch 15/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6426 -
accuracy: 0.5173
Epoch 16/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6392 -
accuracy: 0.5208
Epoch 17/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6251 -
accuracy: 0.5342
Epoch 18/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6251 -
accuracy: 0.5353
Epoch 19/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6211 -
accuracy: 0.5393
Epoch 20/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6106 -
accuracy: 0.5491
Epoch 21/250
898/898 [==============================] - 7s 8ms/step - loss: 1.6008 -
accuracy: 0.5604
Epoch 22/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5956 -
accuracy: 0.5655
Epoch 23/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5935 -
accuracy: 0.5677
Epoch 24/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5911 -
accuracy: 0.5704
Epoch 25/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5818 -
accuracy: 0.5793
```

```
Epoch 26/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5725 -
accuracy: 0.5898
Epoch 27/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5682 -
accuracy: 0.5936
Epoch 28/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5658 -
accuracy: 0.5961
Epoch 29/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5596 -
accuracy: 0.6024
Epoch 30/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5649 -
accuracy: 0.5970
Epoch 31/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5561 -
accuracy: 0.6065
Epoch 32/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5527 -
accuracy: 0.6083
Epoch 33/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5540 -
accuracy: 0.6082
Epoch 34/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5504 -
accuracy: 0.6120
Epoch 35/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5541 -
accuracy: 0.6081
Epoch 36/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5424 -
accuracy: 0.6194
Epoch 37/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5419 -
accuracy: 0.6210
Epoch 38/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5382 -
accuracy: 0.6244
Epoch 39/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5376 -
accuracy: 0.6251
Epoch 40/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5343 -
accuracy: 0.6270
Epoch 41/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5334 -
accuracy: 0.6303
```

```
Epoch 42/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5291 -
accuracy: 0.6333
Epoch 43/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5258 -
accuracy: 0.6367
Epoch 44/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5255 -
accuracy: 0.6372
Epoch 45/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5208 -
accuracy: 0.6419
Epoch 46/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5188 -
accuracy: 0.6439
Epoch 47/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5167 -
accuracy: 0.6465
Epoch 48/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5207 -
accuracy: 0.6418
Epoch 49/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5198 -
accuracy: 0.6427
Epoch 50/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5144 -
accuracy: 0.6487
Epoch 51/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5129 -
accuracy: 0.6507
Epoch 52/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5103 -
accuracy: 0.6522
Epoch 53/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5072 -
accuracy: 0.6554
Epoch 54/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5108 -
accuracy: 0.6511
Epoch 55/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5088 -
accuracy: 0.6548
Epoch 56/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5010 -
accuracy: 0.6615
Epoch 57/250
898/898 [==============================] - 7s 8ms/step - loss: 1.5017 -
accuracy: 0.6613
```

```
Epoch 58/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4994 -
accuracy: 0.6643
Epoch 59/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4966 -
accuracy: 0.6663
Epoch 60/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4933 -
accuracy: 0.6691
Epoch 61/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4877 -
accuracy: 0.6757
Epoch 62/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4920 -
accuracy: 0.6705
Epoch 63/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4874 -
accuracy: 0.6757
Epoch 64/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4914 -
accuracy: 0.6718
Epoch 65/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4901 -
accuracy: 0.6724
Epoch 66/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4838 -
accuracy: 0.6789
Epoch 67/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4826 -
accuracy: 0.6806
Epoch 68/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4842 -
accuracy: 0.6792
Epoch 69/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4866 -
accuracy: 0.6768
Epoch 70/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4897 -
accuracy: 0.6733
Epoch 71/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4843 -
accuracy: 0.6785
Epoch 72/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4800 -
accuracy: 0.6819
Epoch 73/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4745 -
accuracy: 0.6888
```

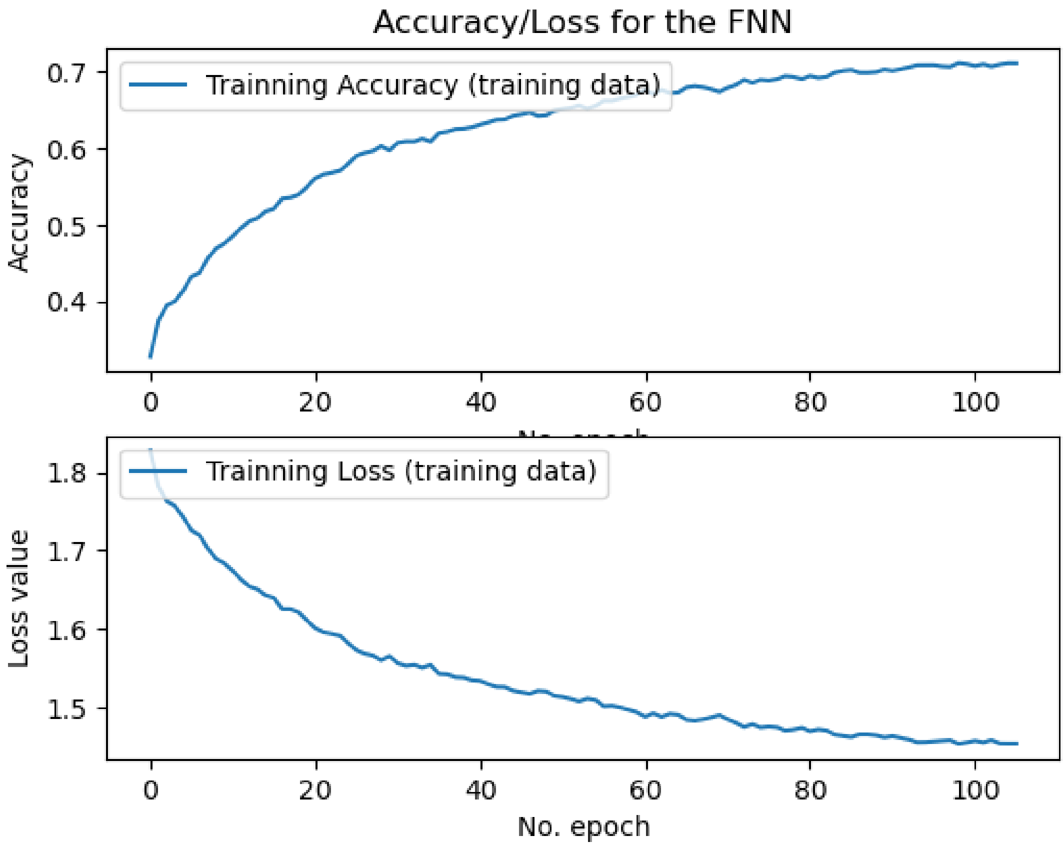
```
Epoch 74/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4781 -
accuracy: 0.6848
Epoch 75/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4740 -
accuracy: 0.6887
Epoch 76/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4752 -
accuracy: 0.6876
Epoch 77/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4742 -
accuracy: 0.6896
Epoch 78/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4700 -
accuracy: 0.6936
Epoch 79/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4712 -
accuracy: 0.6926
Epoch 80/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4735 -
accuracy: 0.6898
Epoch 81/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4692 -
accuracy: 0.6938
Epoch 82/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4715 -
accuracy: 0.6914
Epoch 83/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4700 -
accuracy: 0.6928
Epoch 84/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4649 -
accuracy: 0.6983
Epoch 85/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4632 -
accuracy: 0.7006
Epoch 86/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4617 -
accuracy: 0.7017
Epoch 87/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4653 -
accuracy: 0.6982
Epoch 88/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4653 -
accuracy: 0.6981
Epoch 89/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4640 -
accuracy: 0.6989
```

```
Epoch 90/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4613 -
accuracy: 0.7022
Epoch 91/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4630 -
accuracy: 0.7005
Epoch 92/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4607 -
accuracy: 0.7027
Epoch 93/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4585 -
accuracy: 0.7051
Epoch 94/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4549 -
accuracy: 0.7078
Epoch 95/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4551 -
accuracy: 0.7078
Epoch 96/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4560 -
accuracy: 0.7079
Epoch 97/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4569 -
accuracy: 0.7063
Epoch 98/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4577 -
accuracy: 0.7057
Epoch 99/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4529 -
accuracy: 0.7106
Epoch 100/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4546 -
accuracy: 0.7094
Epoch 101/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4569 -
accuracy: 0.7067
Epoch 102/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4547 -
accuracy: 0.7091
Epoch 103/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4578 -
accuracy: 0.7062
Epoch 104/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4534 -
accuracy: 0.7088
Epoch 105/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4532 -
accuracy: 0.7104
```

```
Epoch 106/250
898/898 [==============================] - 7s 8ms/step - loss: 1.4532 -
accuracy: 0.7103
```

[36]:
```
# Plot history
plt.figure()
fig, ax = plt.subplots(2,1)
ax[0].plot(history6.history['accuracy'], label='Trainning Accuracy (training␣
 ↪data)')
ax[1].plot(history6.history['loss'], label='Trainning Loss (training data)')
# ax[0].plot(history1.history['val_accuracy'], label='Validation Accuracy␣
 ↪(training data)')
# ax[1].plot(history1.history['val_loss'], label='Validation Loss (training␣
 ↪data)')
ax[0].set_title('Accuracy/Loss for the FNN')
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('No. epoch')
ax[0].legend(loc="upper left")
ax[1].set_ylabel('Loss value')
ax[1].set_xlabel('No. epoch')
ax[1].legend(loc="upper left")
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



[37]:
```
results = model6.evaluate(X_valid, Y_valid)
print("test loss, test acc:", results)
print("Total trainning time: ", (endTime - startTime), "seconds")
```

```
113/113 [==============================] - 0s 3ms/step - loss: 1.6471 -
accuracy: 0.5128
test loss, test acc: [1.647139072418213, 0.5128205418586731]
Total trainning time:  727.9486835002899 seconds
```

0.0.16 (d.ii) (1 point)

Run the best model that was found based on the validation set from question (d.i) on the testing set. Report the emotion classification accuracy on the testing set. How does this metric compare to the FNN?

[38]:
```
result2_final = model5.evaluate(X_test, Y_test)
print("test loss, test acc:", result1_final)
```

```
113/113 [==============================] - 0s 3ms/step - loss: 1.6180 -
accuracy: 0.5460
test loss, test acc: [1.705011248588562, 0.45429208874702454]
```

0.0.17 (g) (Bonus - 1 point) Data augmentation:

Data augmentation is a way to increase the size of our dataset and reduce overfitting, especially when we use complicated models with many parameters to learn. Using any available toolbox or your own code, implement some of these techniques and augment the original FER data.

0.0.18 Augmented Data set:

The augmented set below generates a new image by rotation (from -30 degrees to +30 degrees), Shifting horizontally or vertically and completely flipping the image). A total of 1000 new images have been generated from a sample of 5000 original images. Only 5000 were chosen, since my computer didnt have enough resources to use the complete dataset.

[39]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[40]:
```
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
datagen.fit(X)
```

[41]:
```
X_aug  = []
Y_aug = []
for X_batch, y_batch in datagen.flow(X[:5000], Y[:5000], batch_size=1000):
    for i in range(1000):
        max_ele = X_batch[i].max()
        min_ele = X_batch[i].min()
        X_batch[i] = (X_batch[i]-min_ele)/(max_ele - min_ele + 0.000000000001)
        X_aug.append(X_batch[i])
        Y_aug.append(y_batch[i])
    break
```

[42]:
```
X_aug = np.array(X_aug)
X_aug = X_aug.reshape(X_aug.shape[0], 48, 48, 1)
Y_aug = tf.keras.utils.to_categorical(Y_aug, 7)
```

[43]:
```
X_aug.shape
```

[43]:
```
(1000, 48, 48, 1)
```

0.0.19 (e) (1 point) Bayesian optimization for hyper-parameter tuning:

Instead of performing grid or random search to tune the hyper-parameters of the CNN, we can also try a model-based method for finding the optimal hyper-parameters through Bayesian optimization. This method performs a more intelligent search on the hyper-parameter space in order to estimate the best set of hyper-parameters for the data. Use publicly available libraries (e.g., hyperopt in

Python) to perform a Bayesian optimization on the hyper-parameter space using the validation set. Re-port the emotion classification accuracy on the testing set.

0.0.20 Soln:

For the code below, the first model of CNN has been taken. The parameters searched are dropuout probablities (ranging from 0.1 to 0.35) and kernel size for convolutions (ranging from 1x3 or 3x3).

[46]:
```
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,␣
 ↪Dropout
from tensorflow.keras.utils import to_categorical

def optimize_cnn(hyperparameter):

  # Define model using hyperparameters
    cnn_model = tf.keras.Sequential([tf.keras.layers.Conv2D(64,␣
 ↪kernel_size=hyperparameter['conv_kernel_size'], activation='relu',␣
 ↪input_shape=(48,48,1)),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Dropout(rate =␣
 ↪hyperparameter['dropout_prob']),
                             tf.keras.layers.
 ↪MaxPooling2D(pool_size=hyperparameter['conv_kernel_size']),
                             tf.keras.layers.Conv2D(32,␣
 ↪kernel_size=hyperparameter['conv_kernel_size'], activation='relu'),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.
 ↪Dropout(rate=hyperparameter['dropout_prob']),
                             tf.keras.layers.
 ↪MaxPooling2D(pool_size=hyperparameter['conv_kernel_size']),
                             tf.keras.layers.Flatten(),
                             tf.keras.layers.Dense(425, activation=tf.nn.
 ↪sigmoid),
                             tf.keras.layers.BatchNormalization(),
                             tf.keras.layers.Dropout(rate =␣
 ↪hyperparameter['dropout_prob']),])
    cnn_model.add(tf.keras.layers.Dense(7, activation='softmax'))

    cnn_model.compile(optimizer='Adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'],)

    train_X, train_y = X, Y
    valid_X, valid_y = X_valid, Y_valid

    cnn_model.fit(train_X, train_y, epochs=100, batch_size=256, verbose=0)
```

```
  # Evaluate accuracy on validation data
    performance = cnn_model.evaluate(valid_X, valid_y, verbose=0)

    print("Hyperparameters: ", hyperparameter, "Accuracy: ", performance[1])
    print("----------------------------------------------------")
    return({"status": STATUS_OK, "loss": -1*performance[1], "model":cnn_model})


# Define search space for hyper-parameters
space = {
    # The kernel_size for convolutions:
    'conv_kernel_size': hp.choice('conv_kernel_size', [1,3]),
    # Uniform distribution in finding appropriate dropout values
    'dropout_prob': hp.choice('dropout_prob', [0.1, 0.2, 0.3]),
}

trials = Trials()

# Find the best hyperparameters
best = fmin(
        optimize_cnn,
        space,
        algo=tpe.suggest,
        trials=trials,
        max_evals=5,
    )

print("==================================")
print("Best Hyperparameters", best)

# Find trial which has minimum loss value and use that model to perform␣
 ↪evaluation on the test data
test_model = trials.results[np.argmin([r['loss'] for r in trials.
 ↪results])]['model']

performance = test_model.evaluate(X_test, Y_test)

print("==================================")
print("Test Accuracy: ", performance[1])
```

```
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.1}
Accuracy:
0.39966556429862976
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.1}
```

```
Accuracy:
0.5490524172782898
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2}
Accuracy:
0.5618728995323181
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.3}
Accuracy:
0.3940914273262024
----------------------------------------------------
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2}
Accuracy:
0.5515607595443726
----------------------------------------------------
100%|      | 5/5 [42:08<00:00, 505.70s/trial, best loss:
-0.5618728995323181]
==================================
Best Hyperparameters {'conv_kernel_size': 1, 'dropout_prob': 1}


       ␣
 ↪---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call␣
 ↪last)

        <ipython-input-46-1898a14f5022> in <module>
         60 test_model = trials.results[np.argmin([r['loss'] for r in trials.
 ↪results])]['model']
         61
    ---> 62 performance = test_model.evaluate(X_test, Y_test)
         63
         64 print("==================================")


        ~\.
 ↪conda\envs\gputensorflow\lib\site-packages\tensorflow\python\keras\engine\training.
 ↪py in _method_wrapper(self, *args, **kwargs)
        106   def _method_wrapper(self, *args, **kwargs):
        107     if not self._in_multi_worker_mode():  # pylint:␣
 ↪disable=protected-access
    --> 108       return method(self, *args, **kwargs)
        109
        110     # Running inside `run_distribute_coordinator` already.
```

```
    ~\.
↪conda\envs\gputensorflow\lib\site-packages\tensorflow\python\keras\engine\training.
↪py in evaluate(self, x, y, batch_size, verbose, sample_weight, steps,␣
↪callbacks, max_queue_size, workers, use_multiprocessing, return_dict)
   1354                 use_multiprocessing=use_multiprocessing,
   1355                 model=self,
-> 1356                 steps_per_execution=self._steps_per_execution)
   1357
   1358         # Container that configures and calls `tf.keras.Callback`s.


    ~\.
↪conda\envs\gputensorflow\lib\site-packages\tensorflow\python\keras\engine\data_adapter.
↪py in __init__(self, x, y, sample_weight, batch_size, steps_per_epoch,␣
↪initial_epoch, epochs, shuffle, class_weight, max_queue_size, workers,␣
↪use_multiprocessing, model, steps_per_execution)
   1115             use_multiprocessing=use_multiprocessing,
   1116             distribution_strategy=ds_context.get_strategy(),
-> 1117             model=model)
   1118
   1119       strategy = ds_context.get_strategy()


    ~\.
↪conda\envs\gputensorflow\lib\site-packages\tensorflow\python\keras\engine\data_adapter.
↪py in __init__(self, x, y, sample_weights, sample_weight_modes, batch_size,␣
↪epochs, steps, shuffle, **kwargs)
    280                 label, ", ".join(str(i.shape[0]) for i in nest.
↪flatten(data)))
    281         msg += "Please provide data which shares the same first␣
↪dimension."
--> 282         raise ValueError(msg)
    283       num_samples = num_samples.pop()
    284


    ValueError: Data cardinality is ambiguous:
  x sizes: 3588
  y sizes: 1000
Please provide data which shares the same first dimension.
```

[48]:
```
performance = test_model.evaluate(X_test, Y_test)

print("===================================")
```

```python
print("Test Accuracy: ", performance[1])
```

```
113/113 [==============================] - 1s 5ms/step - loss: 1.9133 -
accuracy: 0.5510
==================================
Test Accuracy:  0.5510033369064331
```

[ ]:

```python

```