DATA EXPLORATION

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

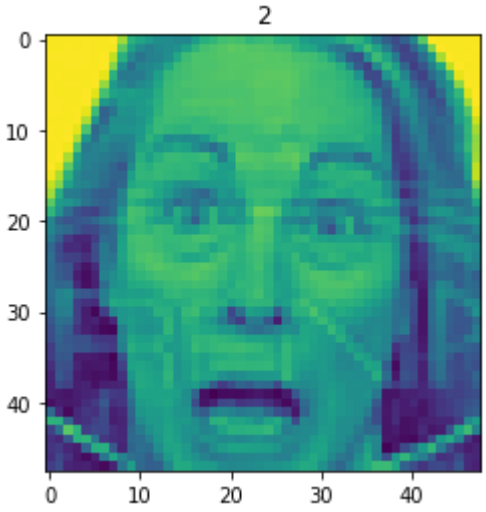
```
#IMPORTING THE DATA FROM THE DATASETS.
import pandas as pd
import numpy as np
import csv
import matplotlib.pyplot as plt
from skimage import data, io, filters

path = "/content/drive/My Drive/Homework3_Data/Train_Data.csv"

path3 = "/content/drive/My Drive/Homework3_Data/Validation_Data.csv"
train_data_df = pd.read_csv(path)

validation_data_df = pd.read_csv(path3)
```

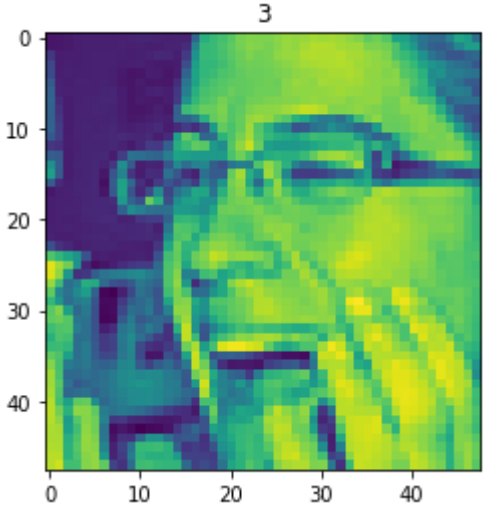
```
#Creating data arrays
train_labels = train_data_df['emotion']
valid_labels = validation_data_df['emotion']

train_pixels = train_data_df.drop('emotion',axis=1)
train_pixels = train_pixels.to_numpy()

valid_pixels = validation_data_df.drop('emotion',axis=1)
valid_pixels = valid_pixels.to_numpy()

def image_display(i):
  image = train_pixels[i][0]
  shape = (48,48)
  image = [int(k) for k in image.split(' ')]
  image = np.array(image)
  image = image.reshape(shape)
  plt.title(train_labels[i])
  plt.imshow(image)
  plt.show()
```

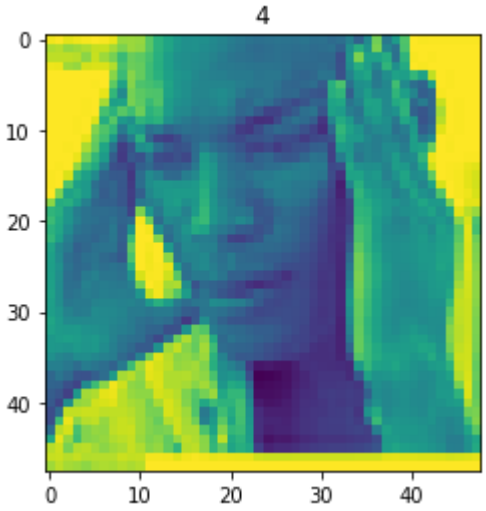
```
#----------------------------TESTTING STUFF
path2 = "/content/drive/My Drive/Homework3_Data/Test_Data.csv"
test_data_df = pd.read_csv(path2)
test_labels = test_data_df['emotion']

test_pixels = test_data_df.drop('emotion',axis=1)
test_pixels = test_pixels.to_numpy()
test_image_array = []
for i in range (len(test_pixels)):
```

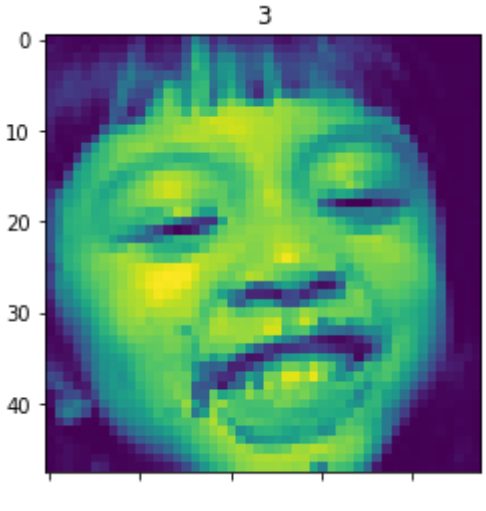
```
  temp = test_pixels[i][0]
  temp = [int(k) for k in temp.split(' ')]
  temp = np.array(temp)
  test_image_array.append(temp)

test_image_array = np.array(test_image_array)

```

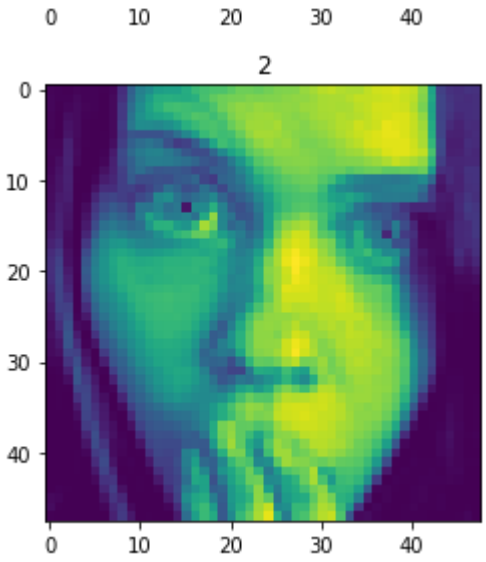
```
import random
my_randoms=[]
for i in range (10):
    my_randoms.append(random.randrange(1,len(train_pixels),1))
for i in my_randoms:
  image_display(i)
```

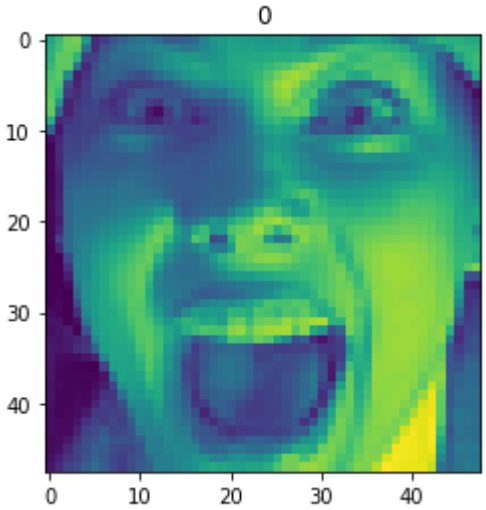

2
0
10
20
30
40
0
10
20
30
40

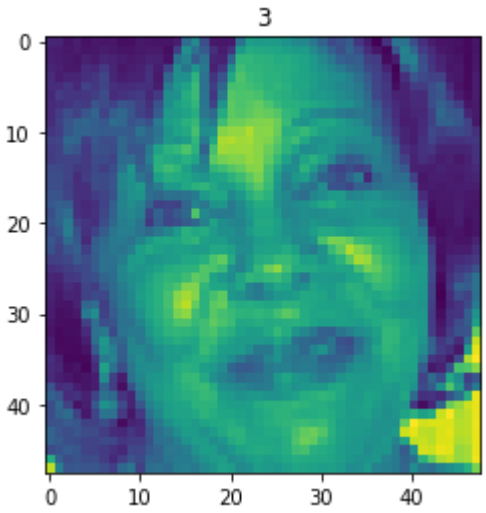

3
0
10
20
30
40
0
10
20
30
40

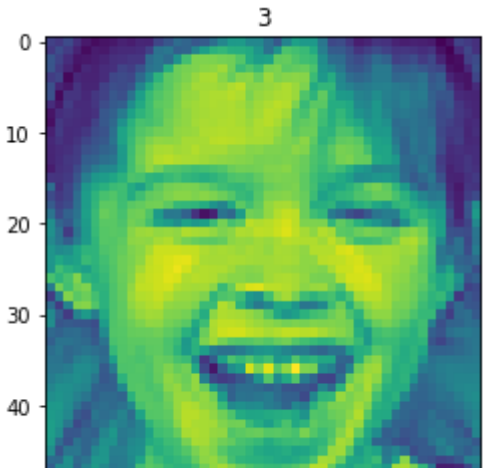

4
0
10
20
30
40
0
10
20
30
40

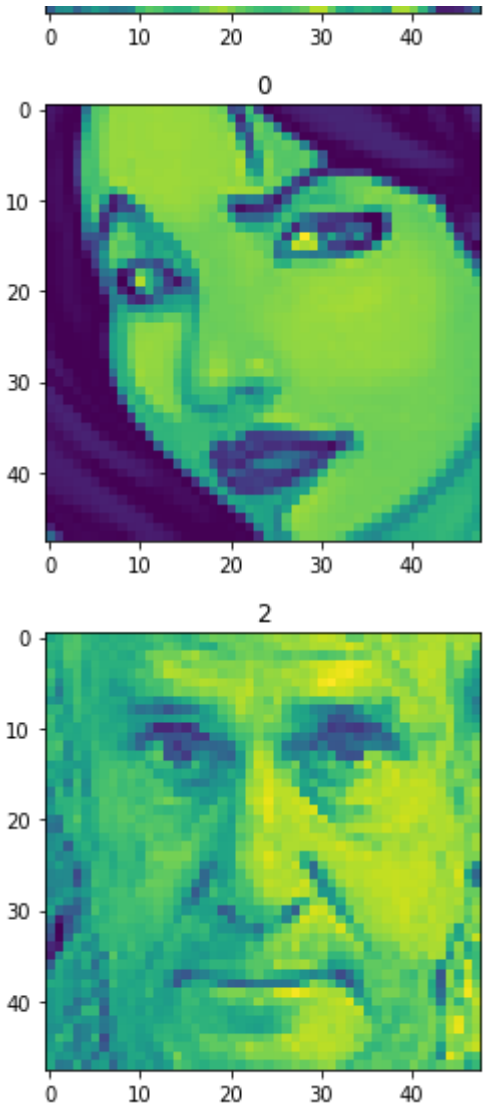

3
0
10
20
30
40


2


0


3


3




```
count0 = 0
count1 = 0
count2 = 0
count3 = 0
count4 = 0
count5 = 0
count6 = 0
for i in range(len(train_labels)):
  if (train_labels[i]==0):
    count0 += 1
  elif(train_labels[i]==1):
    count1 += 1
  elif(train_labels[i]==2):
    count2 += 1
  elif(train_labels[i]==3):
    count3 += 1
  elif(train_labels[i]==4):
    count4 += 1
  elif(train_labels[i]==5):
    count5 += 1
  elif(train_labels[i]==6):
    count6 += 1
  else:
    print("Oops ")
```

```
    print( "Oops." )
total = count0+count1+count2+count3+count4+\
        count5+count6
print("\nAngry:   ",count0,"\nDisgust:  ", count1,"\nFear:    ",count2,\
      "\nHappy:   ",count3,"\nSad:     ",count4,"\nSurprise:" \
      , count5,"\nNeutral: " ,count6,"\n_______\nTotal:  ",total)
```

```
Angry:    3995
Disgust:   436
Fear:     4097
Happy:    7215
Sad:      4830
Surprise: 3171
Neutral:  4965
_______
Total:   28709
```

FNN

```
train_image_array = []
for i in range (len(train_pixels)):
  temp = train_pixels[i][0]
  temp = [int(k) for k in temp.split(' ')]
  temp = np.array(temp)
  train_image_array.append(temp)
```

```
valid_image_array = []
for i in range (len(valid_pixels)):
  temp = valid_pixels[i][0]
  temp = [int(k) for k in temp.split(' ')]
  temp = np.array(temp)
  valid_image_array.append(temp)
```

```
train_image_array = np.array(train_image_array)
valid_image_array = np.array(valid_image_array)
```

```
from skimage import data, io, filters
#noramlise all the images
train_image_array = (train_image_array / 255) - 0.5
valid_image_array = (valid_image_array / 255) -0.5
test_image_array = (test_image_array / 255) -0.5

# train_image_array = filters.sobel(train_image_array)
# valid_image_array = filters.sobel(valid_image_array)
# test_image_array = filters.sobel(test_image_array)
```

```
import warnings
import keras
warnings.filterwarnings("ignore")
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

```
from keras.utils import to_categorical
from keras import regularizers


# # Flatten the images into vectors (1D) for feed forward network
# flatten_train_images = train_image_array.reshape((-1, 48*48))
# flatten_test_images = test_image_array.reshape((-1, 48*48))
# flatten_valid_images = valid_image_array.reshape((-1, 48*48))
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %te

```
import time
class TimeHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.times = []

    def on_epoch_begin(self, batch, logs={}):
        self.epoch_time_start = time.time()

    def on_epoch_end(self, batch, logs={}):
        self.times.append(time.time() - self.epoch_time_start)
```

```
# Evaluate your model's performance on the test data
def model_test(model):
  performance = model.evaluate(valid_image_array, to_categorical(valid_labels))
  print("\nAccuracy on Test samples: {0}".format(performance[1]))
```

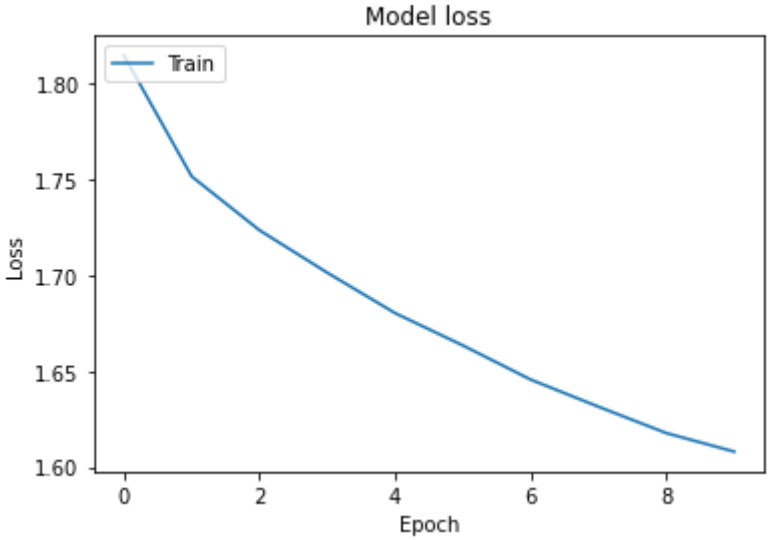
```
# Compiling the model
def model_compile(model):
  model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'],
# Train model
  time_callback = TimeHistory()
  hist = model.fit(train_image_array, to_categorical(train_labels), epochs=10, batch_si

  plt.plot(hist.history['loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train'], loc='upper left')
  plt.show()
  print("\nTotal Training Time is: ", sum(time_callback.times),'s.\n')
```

```
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials

#MODEL 1
#Define a Feed-Forward Model with 2 hidden layers with dimensions 392 and 196 Neurons
def optimize_fnn(hyperparameter):
  model = Sequential([
    Dense(2304, activation=hyperparameter['activation_fn'], input_shape=(48*48,), name=
    Dense(2304//2, activation=hyperparameter['activation_fn'], name="second_hidden_laye
    Dense(7, activation='softmax'),
```

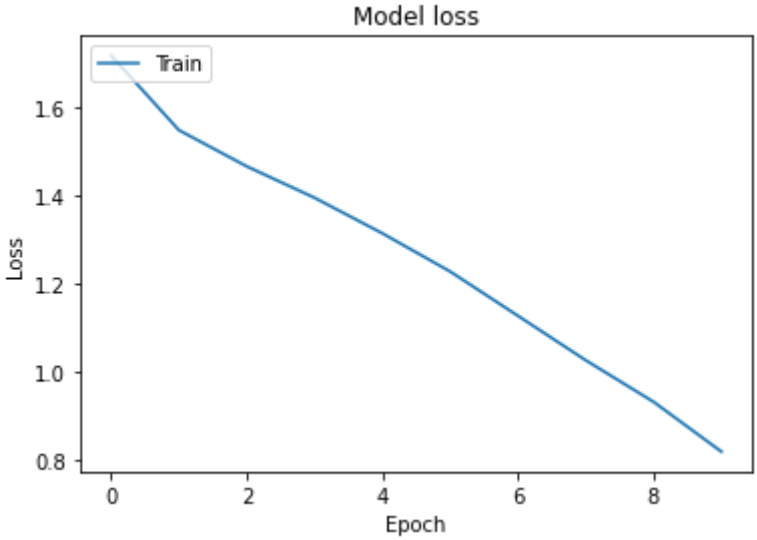
```
    Dense(7, activation='softmax'),
  ])
  #  Validate your Model Architecture
  #print(model1.summary())
  model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentropy',
  time_callback = TimeHistory()
  hist = model.fit(train_image_array, to_categorical(train_labels), epochs=10, batch_si
  plt.plot(hist.history['loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train'], loc='upper left')
  plt.show()
  print("Total Training Time is (s): ", sum(time_callback.times))

  performance = model.evaluate(valid_image_array, to_categorical(valid_labels), verbose

  print("Hyperparameters: ", hyperparameter, "Accuracy: ", performance[1])
  print("------------------------------------------------------")
  # We want to minimize loss i.e. negative of accuracy
  return({"status": STATUS_OK, "loss": -1*performance[1], "model":model})

# Define search space for hyper-parameters
space = {
    # The activation_fn choices:
    'activation_fn':hp.choice('activation_fn',['relu']),
    # Uniform distribution in finding appropriate dropout values
    'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.5),
    # Choice of optimizer
    'optimizer': hp.choice('optimizer', ['Adam', 'sgd']),
}

trials = Trials()

best = fmin(
        optimize_fnn,
        space,
        algo=tpe.suggest,
        trials=trials,
        max_evals=25,
    )

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

# You can retrain the final model with optimal hyperparameters on train+validation data

# Or you can use the model returned directly
# Find trial which has minimum loss value and use that model to perform evaluation on t
test_model = trials.results[np.argmin([r['loss'] for r in trials.results])]['model']

performance = test_model.evaluate(test_image_array, to_categorical(test_labels))

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

```

65

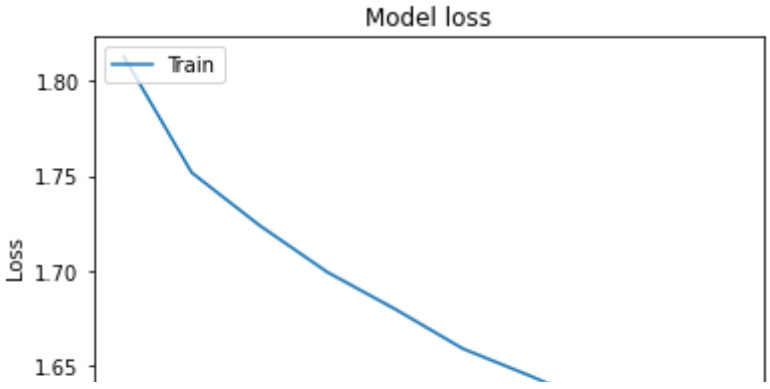
```
  0%|          | 0/25 [00:00<?, ?it/s, best loss: ?]
```

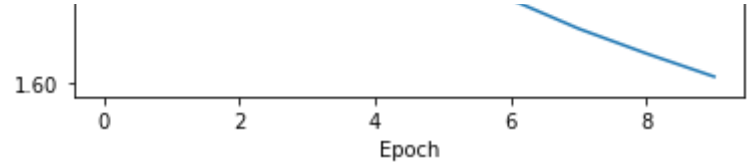



```
Total Training Time is (s):
7.4364213943481445
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.25336800158266903, 'optimizer': 'sgd'}
Accuracy:
0.39091668988991396
----------------------------------------------------------
  4%|▍          | 1/25 [00:07<03:11,  8.00s/it, best loss: -0.39091668988991396]
```

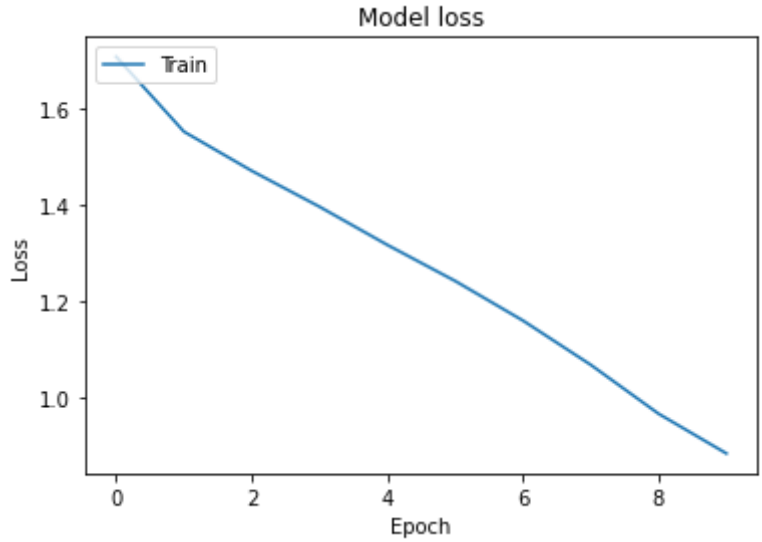



```
Total Training Time is (s):
9.509779930114746
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.10552055620174437, 'optimizer': 'Adam'}
Accuracy:
0.4488715519726393
----------------------------------------------------------
  8%|▊          | 2/25 [00:18<03:19,  8.66s/it, best loss: -0.4488715519726393]
```

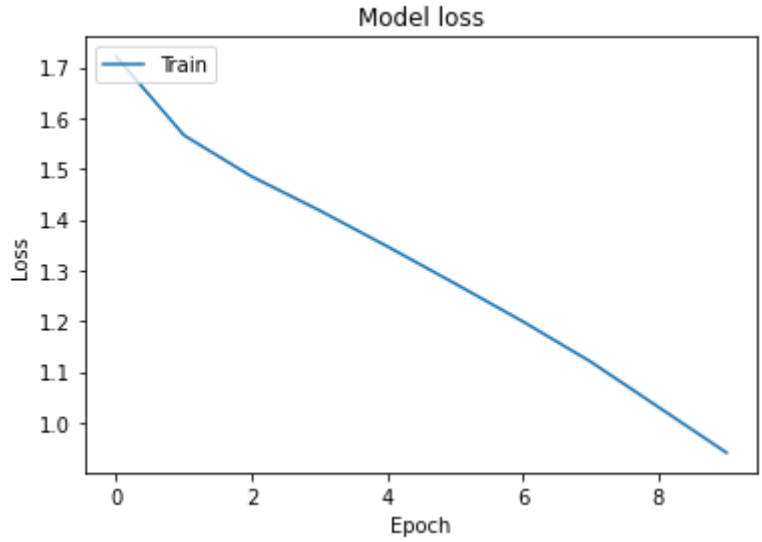




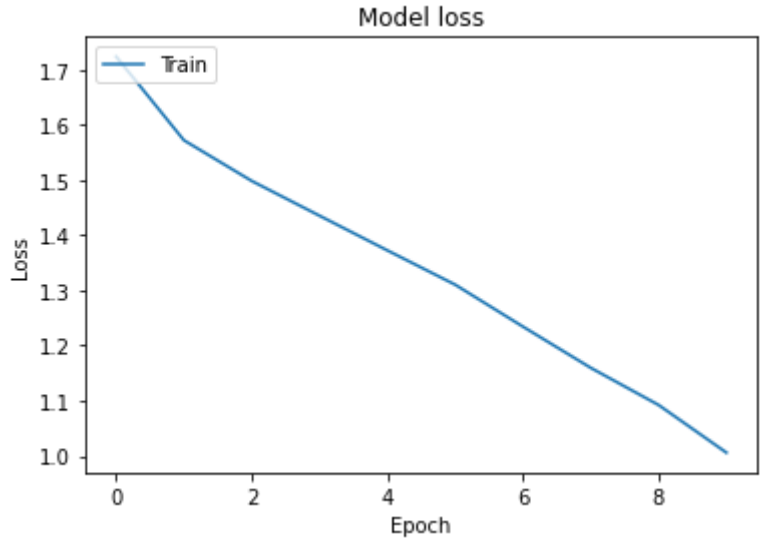

```
Total Training Time is (s):
7.328493356704712
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.1759849523388486, 'optimizer': 'sgd'}
Accuracy:
0.38311507384502713
----------------------------------------------------
 12%|          | 3/25 [00:26<03:05,  8.44s/it, best loss: -0.4488715519726393]
```



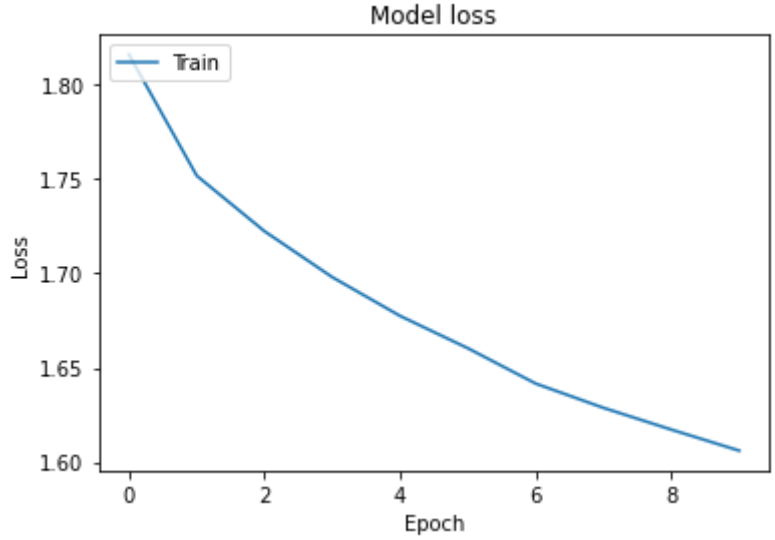

```
Total Training Time is (s):
9.538674116134644
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.18842808735315886, 'optimizer': 'Adam'}
Accuracy:
0.4555586514681553
----------------------------------------------------
 16%|          | 4/25 [00:36<03:08,  8.98s/it, best loss: -0.4555586514681553]
```



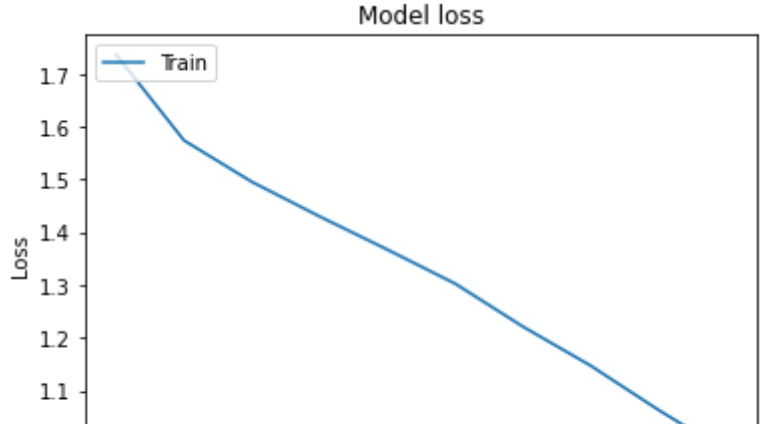

```
Total Training Time is (s):
9.489769220352173
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.2476540471215125, 'optimizer': 'Adam'}
Accuracy:
0.4636388966368912
----------------------------------------------------
 20%|          | 5/25 [00:46<03:07,  9.35s/it, best loss: -0.4636388966368912]
```

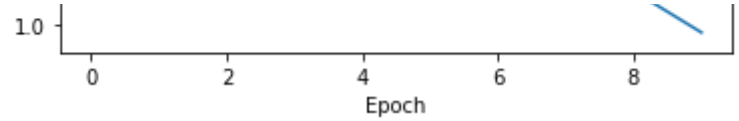



```
Total Training Time is (s):
9.495479583740234
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.3644742571285925, 'optimizer': 'Adam'}
Accuracy:
0.44859292282803076
----------------------------------------------------
 24%|██▍       | 6/25 [00:56<03:02,  9.62s/it, best loss: -0.4636388966368912]
```

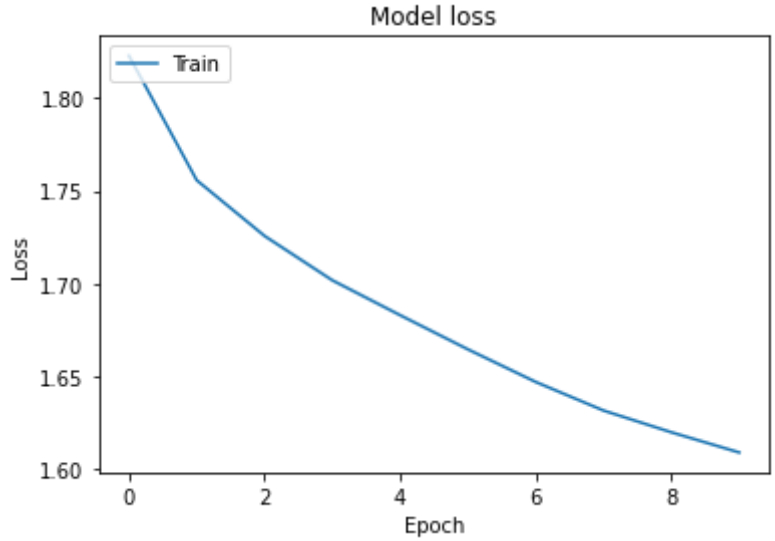



```
Total Training Time is (s):
7.4017863273620605
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.2555559787727395, 'optimizer': 'sgd'}
Accuracy:
0.3842295904234612
----------------------------------------------------
 28%|██▊       | 7/25 [01:04<02:44,  9.15s/it, best loss: -0.4636388966368912]
```

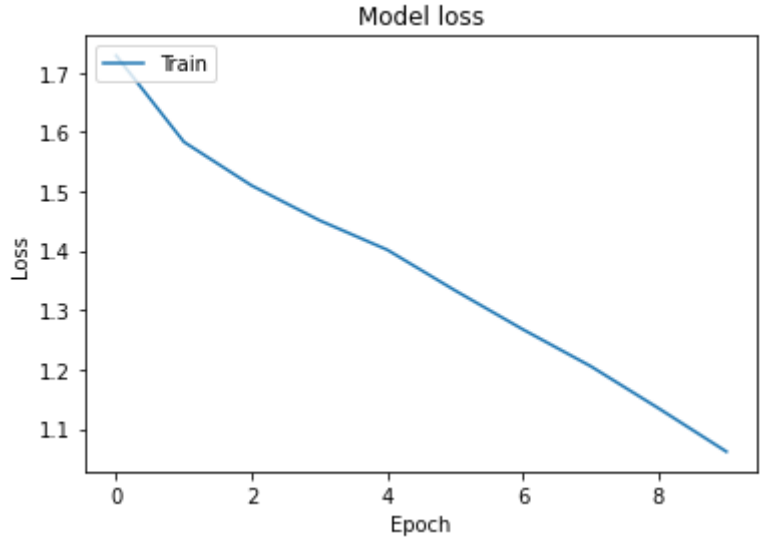




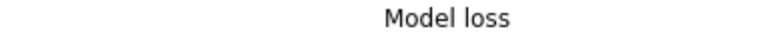

```
Total Training Time is (s):
9.604174613952637
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.32319345246869036, 'optimizer': 'Adam'}
Accuracy:
0.45277236002207
----------------------------------------------------
 32%|███▏      | 8/25 [01:15<02:42,  9.57s/it, best loss: -0.4636388966368912]
```

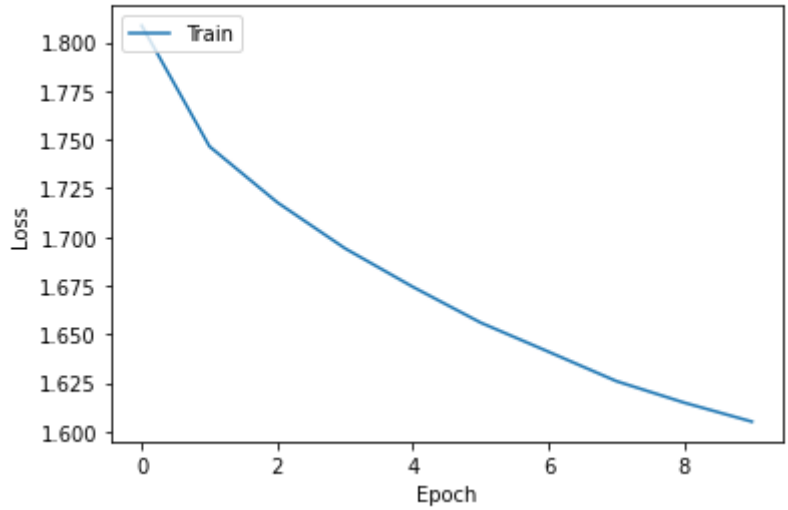



```
Total Training Time is (s):
7.4649622440338135
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.3406336734941001, 'optimizer': 'sgd'}
Accuracy:
0.38255781556411383
----------------------------------------------------
 36%|███▌      | 9/25 [01:23<02:26,  9.15s/it, best loss: -0.4636388966368912]
```

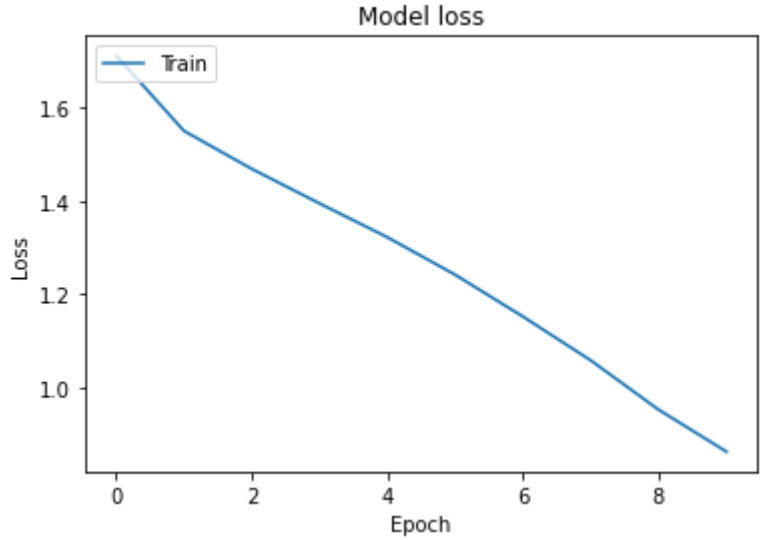



```
Total Training Time is (s):
9.565871000289917
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.4370333627480284, 'optimizer': 'Adam'}
Accuracy:
0.44998606855107337
----------------------------------------------------
 40%|███▉      | 10/25 [01:34<02:22,  9.52s/it, best loss: -0.4636388966368912]
```

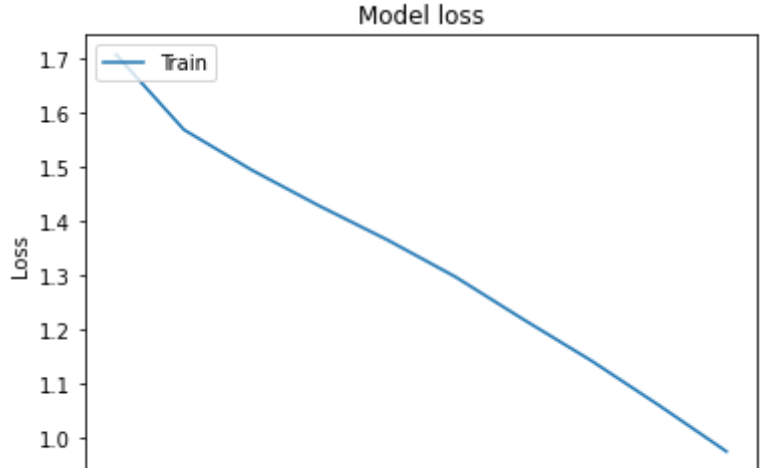


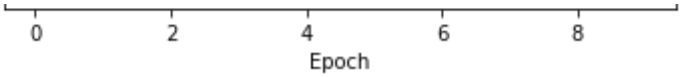



```
Total Training Time is (s):
7.352118968963623
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.2464429141873418, 'optimizer': 'sgd'}
Accuracy:
0.3836723321342442
----------------------------------------------------------
 44%|████▍       | 11/25 [01:42<02:07,  9.10s/it, best loss: -0.4636388966368912]
```

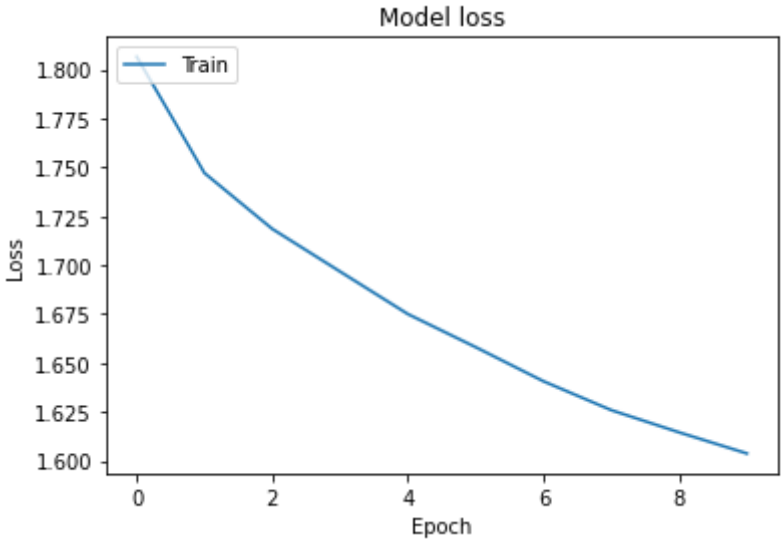



```
Total Training Time is (s):
9.775615215301514
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.1675727368174647, 'optimizer': 'Adam'}
Accuracy:
0.4533296182863757
----------------------------------------------------------
 48%|████▊       | 12/25 [01:52<02:04,  9.56s/it, best loss: -0.4636388966368912]
```

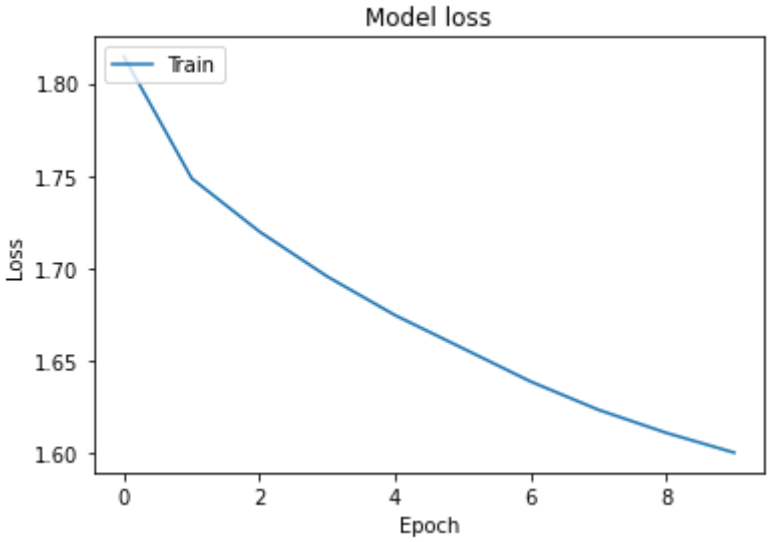




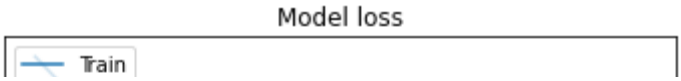

```
Total Training Time is (s):
9.897839069366455
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.3479228659873571, 'optimizer': 'Adam'}
Accuracy:
0.4488715519726393
----------------------------------------------------
 52%|█████      | 13/25 [02:03<01:59,  9.93s/it, best loss: -0.4636388966368912]
```

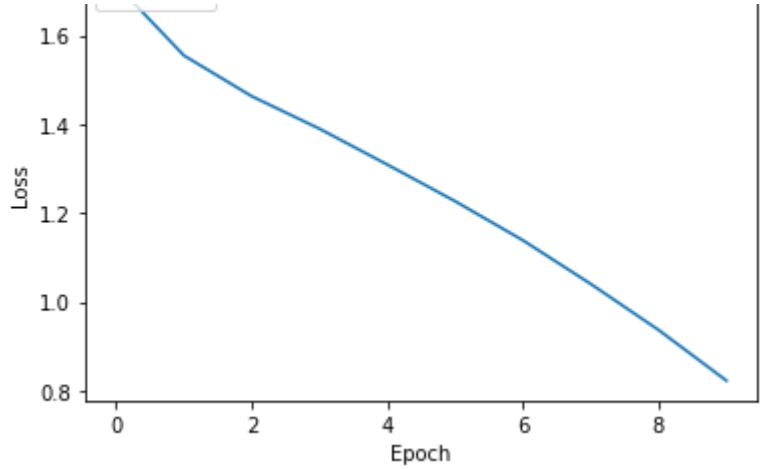



```
Total Training Time is (s):
7.853139162063599
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.23118676879866445, 'optimizer': 'sgd'}
Accuracy:
0.3898021733156317
----------------------------------------------------
 56%|█████      | 14/25 [02:12<01:45,  9.55s/it, best loss: -0.4636388966368912]
```

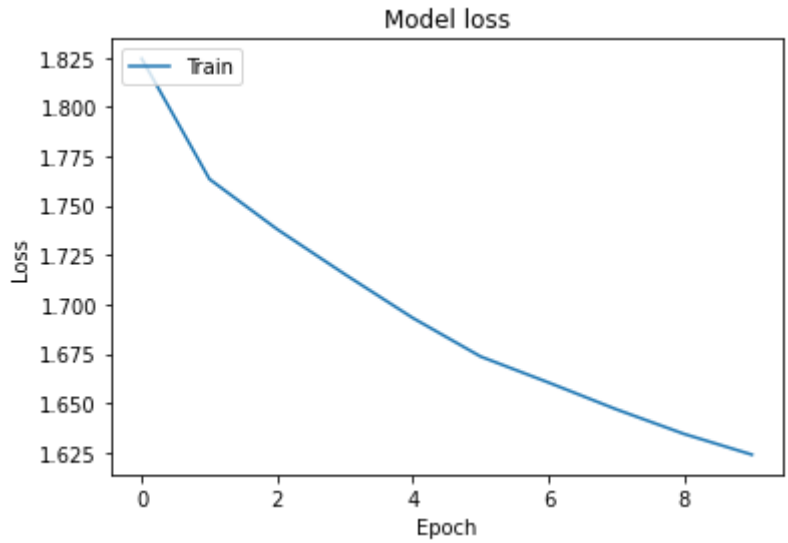



```
Total Training Time is (s):
7.92191481590271
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.11349796190538407, 'optimizer': 'sgd'}
Accuracy:
0.3870158818695465
----------------------------------------------------
 60%|██████     | 15/25 [02:20<01:33,  9.31s/it, best loss: -0.4636388966368912]
```

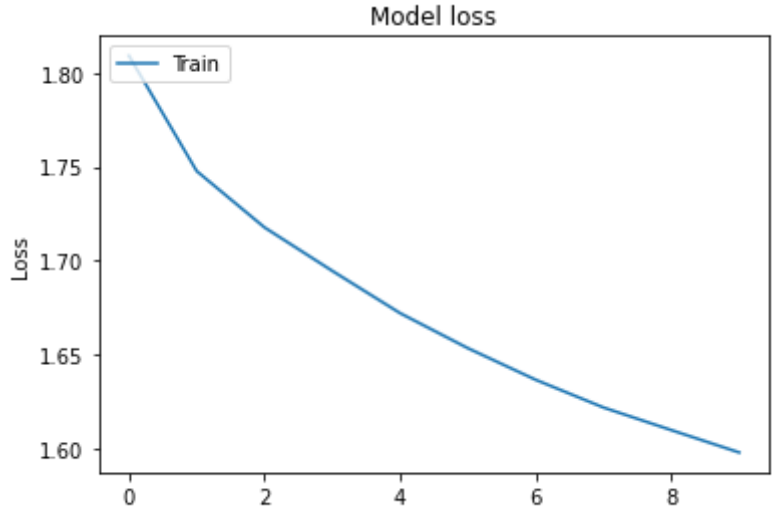





```
Total Training Time is (s):
10.156506061553955
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.10106341569977983, 'optimizer': 'Adam'}
Accuracy:
0.44051267765929486
----------------------------------------------------
 64%|██████    | 16/25 [02:32<01:28,  9.83s/it, best loss: -0.4636388966368912]
```

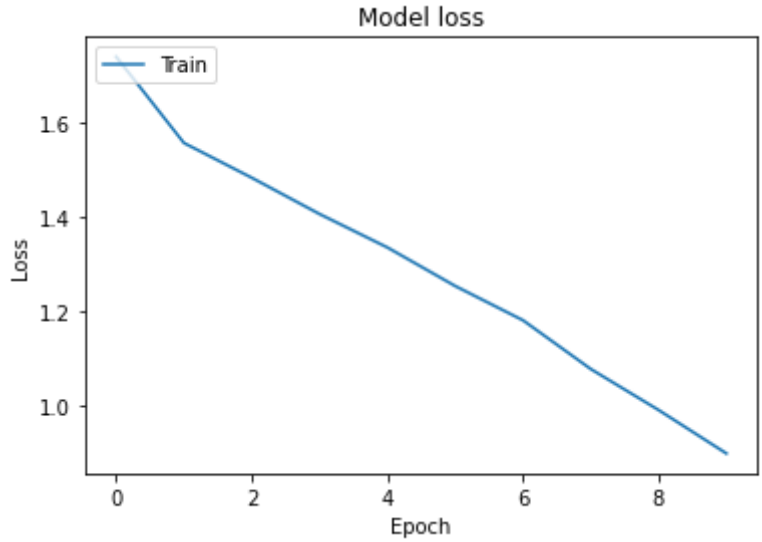



```
Total Training Time is (s):
8.006651639938354
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.49276434949032477, 'optimizer': 'sgd'}
Accuracy:
0.38478684870852636
----------------------------------------------------
 68%|██████▊    | 17/25 [02:40<01:16,  9.55s/it, best loss: -0.4636388966368912]
```

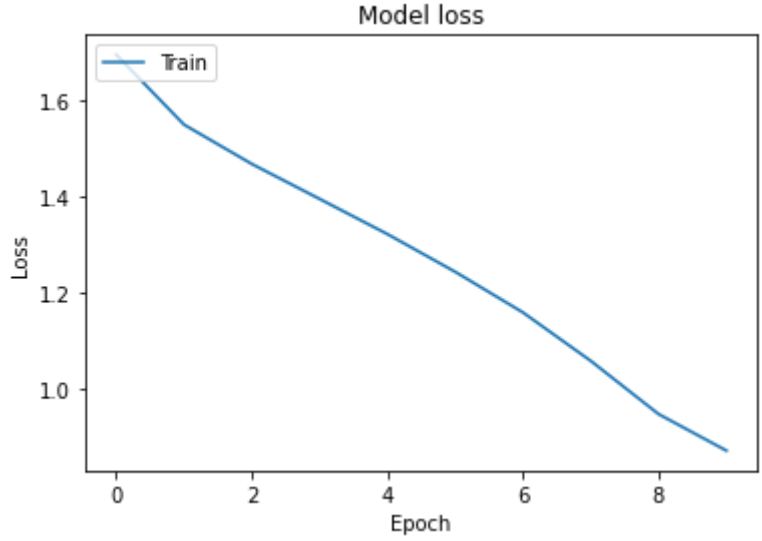


Epoch

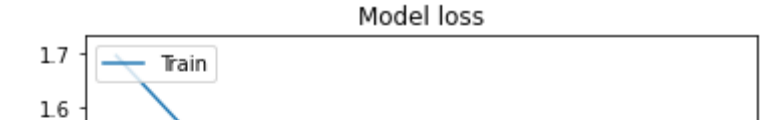
```
Total Training Time is (s):
8.145221471786499
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.11279906259430761, 'optimizer': 'sgd'}
Accuracy:
0.3825578155516582
--------------------------------------------------
 72%|███████   | 18/25 [02:50<01:06,  9.44s/it, best loss: -0.4636388966368912]
```

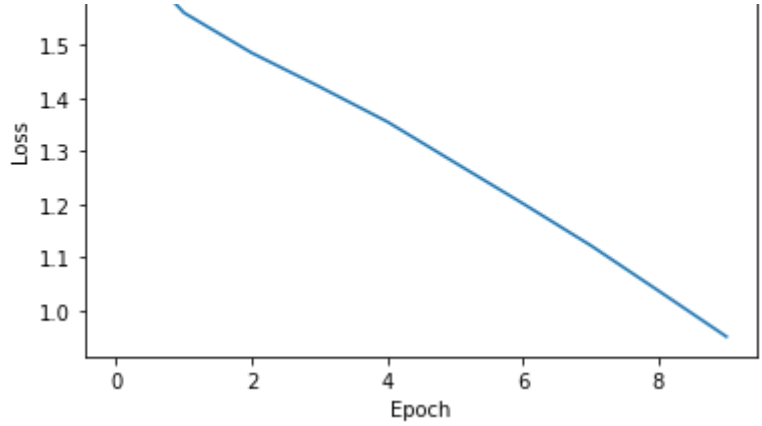



```
Total Training Time is (s):
10.069785594940186
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.21733738187782758, 'optimizer': 'Adam'}
Accuracy:
0.45834494288517724
--------------------------------------------------
 76%|███████   | 19/25 [03:01<00:59,  9.93s/it, best loss: -0.4636388966368912]
```

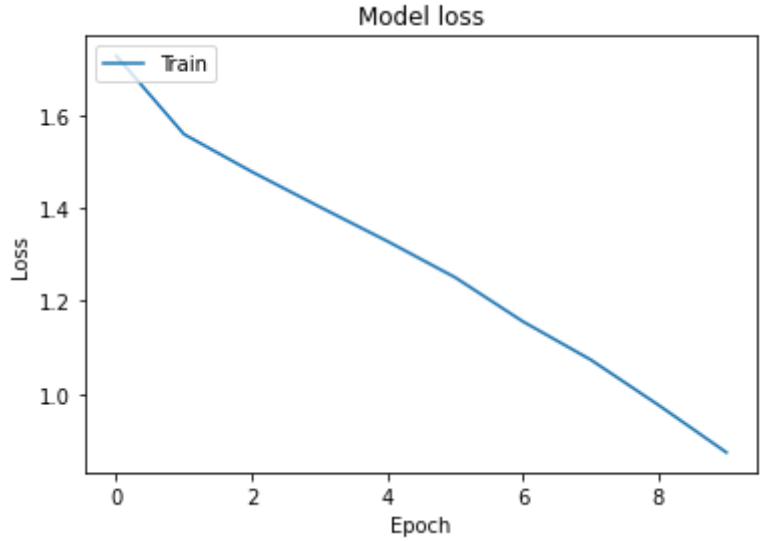



```
Total Training Time is (s):
10.192638874053955
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.14335957639670258, 'optimizer': 'Adam'}
Accuracy:
0.4589022011785462
--------------------------------------------------
 80%|████████  | 20/25 [03:12<00:51, 10.32s/it, best loss: -0.4636388966368912]
```

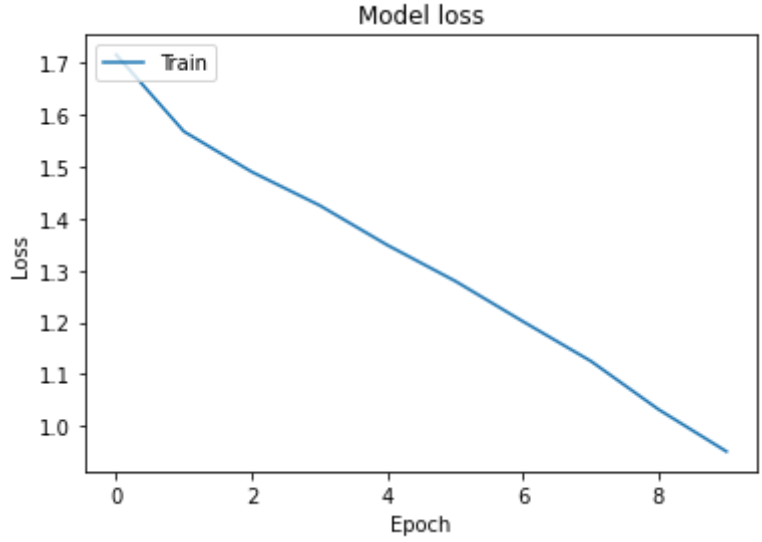




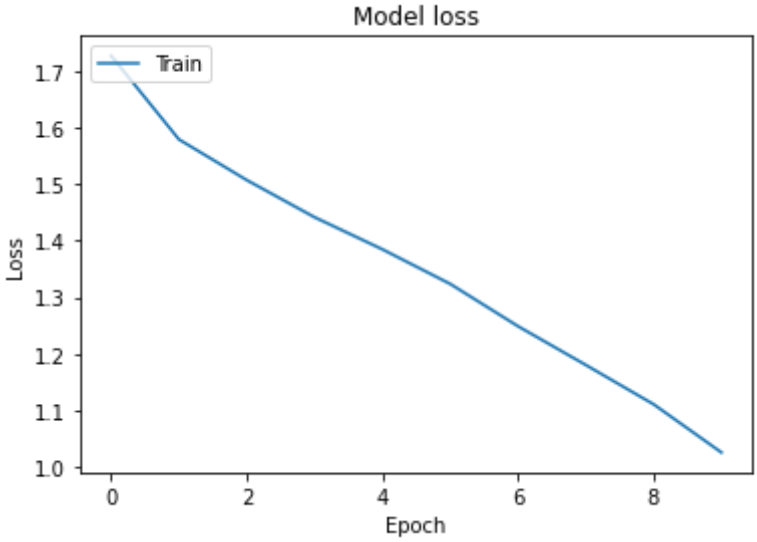

```
Total Training Time is (s):
10.345960855484009
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.2936624860958402, 'optimizer': 'Adam'}
Accuracy:
0.45110058512950746
----------------------------------------------------
 84%|████████▍  | 21/25 [03:23<00:42, 10.64s/it, best loss: -0.4636388966368912]
```



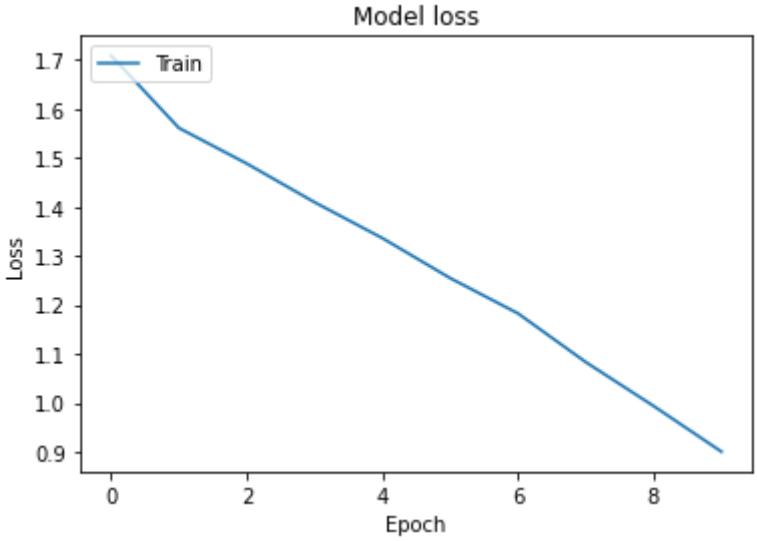

```
Total Training Time is (s):
10.243130207061768
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.15123480999583044, 'optimizer': 'Adam'}
Accuracy:
0.4494288102618563
----------------------------------------------------
 88%|████████▊  | 22/25 [03:35<00:32, 10.84s/it, best loss: -0.4636388966368912]
```



```
Total Training Time is (s):
10.352617740631104
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.28900076526664475, 'optimizer': 'Adam'}
Accuracy:
0.44524937309272844
--------------------------------------------------
 92%|████████▏ | 23/25 [03:46<00:22, 11.03s/it, best loss: -0.4636388966368912]
```




```
Total Training Time is (s):
10.280369758605957
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.40388349489252084, 'optimizer': 'Adam'}
Accuracy:
0.45082195600981034
--------------------------------------------------
 96%|████████▋ | 24/25 [03:57<00:11, 11.13s/it, best loss: -0.4636388966368912]
```




```
Total Training Time is (s):
10.576332807540894
Hyperparameters:
{'activation_fn': 'relu', 'dropout_prob': 0.2051775684138893, 'optimizer': 'Adam'}
Accuracy:
0.45277236002207
--------------------------------------------------
100%|█████████| 25/25 [04:09<00:00,  9.98s/it, best loss: -0.4636388966368912]
================================================================================
Best Hyperparameters {'activation_fn': 0, 'dropout_prob': 0.2476540471215125, 'optimi
3589/3589 [==============================] - 0s 49us/step
==================================
Test Accuracy:  0.45528002229863535
```

CNNs

```
from keras.layers import Conv2D, Flatten, MaxPooling2D

# Flatten the images into vectors (1D) for feed forward network
train_images_3d = train_image_array.reshape((len(train_image_array),48,48,1))
test_images_3d = test_image_array.reshape((len(test_image_array), 48,48,1))
valid_images_3d = valid_image_array.reshape((len(valid_image_array), 48,48,1))

# Define 2 groups of layers: features layer (convolutions) and classification layer
common_features = [Conv2D(64, kernel_size=(3,3), strides=3, activation='relu', input_sh
            Conv2D(64, kernel_size=(3,3), activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.3),
            Conv2D(128, kernel_size=3, activation='relu'),
            Conv2D(128, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.3), Flatten(),]

classifier = [Dense(512, activation='relu'), Dense(7, activation='relu'),]
cnn_model = Sequential(common_features+classifier)
print(cnn_model.summary())
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Model: "sequential_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 16, 16, 64)        640
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_27 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 3, 128)         147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
dropout_28 (Dropout)         (None, 1, 1, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_27 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_28 (Dense)             (None, 7)                 3591
=================================================================
Total params: 328,647
Trainable params: 328,647
Non-trainable params: 0
_________________________________________________________________
None
```

```
 1 from keras.layers import Conv2D, Flatten, MaxPooling2D
 2 def run_cnn_model(kSize, dropout_rate):
 3   # Define 2 groups of layers: features layer (convolutions) and classification layer
 4   common_features = [Conv2D(64, kernel_size=(kSize), strides=3, activation='relu', inpu
 5               Conv2D(64, kernel_size=(kSize), activation='relu'),
 6               MaxPooling2D(pool_size=(2,2)), Dropout(dropout_rate),
 7               Conv2D(128, kernel_size=(kSize), activation='relu'),
 8               Conv2D(128, kernel_size=(kSize), activation='relu'),
 9               MaxPooling2D(pool_size=(2,2)), Dropout(dropout_rate), Flatten(),]
10
11   classifier = [Dense(512, activation='relu'), Dense(7, activation='relu'),]
12   cnn_model = Sequential(common_features+classifier)
13   print("Kernel Size: {0} \nDropout: {1}".format(kSize,dropout_rate))
14   print(cnn_model.summary())
15   cnn_model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accurac
16   # Train model
17   cnn_model.fit(train_images_3d, to_categorical(train_labels), epochs=10, batch_size=25
18
19   performance = cnn_model.evaluate(valid_images_3d, to_categorical(valid_labels))
20   print("Accuracy on Test samples: {0}".format(performance[1]))
21   results.append(performance[1])
```

```
22
```

```
 1 dropout_range = [0.1,0.2,0.3,0.4,0.5]
 2 kernels = [2,3]
 3 featureK = []
 4 featureD = []
 5 # Flatten the images into vectors (1D) for feed forward network
 6 train_images_3d = train_image_array.reshape((len(train_image_array),48,48,1))
 7 test_images_3d = test_image_array.reshape((len(test_image_array), 48,48,1))
 8 valid_images_3d = valid_image_array.reshape((len(valid_image_array), 48,48,1))
 9 results = []
10 for i in dropout_range:
11   for j in kernels:
12     run_cnn_model(j,i)
13     featureK.append(j)
14     featureD.append(i)
15 print(results)
```

```
Kernel Size: 2
Dropout: 0.1
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_113 (Conv2D)          (None, 16, 16, 64)        320
_________________________________________________________________
conv2d_114 (Conv2D)          (None, 15, 15, 64)        16448
_________________________________________________________________
max_pooling2d_57 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_57 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_115 (Conv2D)          (None, 6, 6, 128)         32896
_________________________________________________________________
conv2d_116 (Conv2D)          (None, 5, 5, 128)         65664
_________________________________________________________________
max_pooling2d_58 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
dropout_58 (Dropout)         (None, 2, 2, 128)         0
_________________________________________________________________
flatten_29 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_57 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_58 (Dense)             (None, 7)                 3591
=================================================================
Total params: 381,575
Trainable params: 381,575
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 4s 130us/step - loss: 3.7096 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 44us/step - loss: 1.9558 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 44us/step - loss: 1.9367 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 44us/step - loss: 1.9275 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 44us/step - loss: 1.9317 - acc: 0.2
Epoch 6/10
28709/28709 [==============================] - 1s 43us/step - loss: 1.9115 - acc: 0.2
Epoch 7/10
28709/28709 [==============================] - 1s 43us/step - loss: 1.8791 - acc: 0.3
Epoch 8/10
28709/28709 [==============================] - 1s 44us/step - loss: 1.8719 - acc: 0.3
Epoch 9/10
28709/28709 [==============================] - 1s 44us/step - loss: 1.8369 - acc: 0.3
Epoch 10/10
28709/28709 [==============================] - 1s 43us/step - loss: 1.8920 - acc: 0.3
3589/3589 [==============================] - 1s 334us/step
Accuracy on Test samples: 0.24965171357339125
Kernel Size: 3
Dropout: 0.1
Model: "sequential_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
conv2d_117 (Conv2D)          (None, 16, 16, 64)        640
_________________________________________________________________
conv2d_118 (Conv2D)          (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_59 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_59 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_119 (Conv2D)          (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_120 (Conv2D)          (None, 3, 3, 128)         147584
_________________________________________________________________
max_pooling2d_60 (MaxPooling (None, 1, 1, 128)         0
_________________________________________________________________
dropout_60 (Dropout)         (None, 1, 1, 128)         0
_________________________________________________________________
flatten_30 (Flatten)         (None, 128)               0
_________________________________________________________________
dense_59 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_60 (Dense)             (None, 7)                 3591
=================================================================
Total params: 328,647
Trainable params: 328,647
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 4s 135us/step - loss: 4.3348 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 46us/step - loss: 4.2666 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 46us/step - loss: 4.2487 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 47us/step - loss: 4.2666 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 46us/step - loss: 4.2508 - acc: 0.2
Epoch 6/10
28709/28709 [==============================] - 1s 46us/step - loss: 4.2085 - acc: 0.3
Epoch 7/10
28709/28709 [==============================] - 1s 46us/step - loss: 4.1103 - acc: 0.3
Epoch 8/10
28709/28709 [==============================] - 1s 47us/step - loss: 4.0372 - acc: 0.3
Epoch 9/10
28709/28709 [==============================] - 1s 47us/step - loss: 4.0335 - acc: 0.3
Epoch 10/10
28709/28709 [==============================] - 1s 46us/step - loss: 4.0242 - acc: 0.3
3589/3589 [==============================] - 1s 363us/step
Accuracy on Test samples: 0.3700195040567302
Kernel Size: 2
Dropout: 0.2
Model: "sequential_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_121 (Conv2D)          (None, 16, 16, 64)        320
_________________________________________________________________
conv2d_122 (Conv2D)          (None, 15, 15, 64)        16448
_________________________________________________________________
max_pooling2d_61 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_61 (Dropout)         (None, 7, 7, 64)          0
```

```
dropout_61 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_123 (Conv2D)          (None, 6, 6, 128)         32896
_________________________________________________________________
conv2d_124 (Conv2D)          (None, 5, 5, 128)         65664
_________________________________________________________________
max_pooling2d_62 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
dropout_62 (Dropout)         (None, 2, 2, 128)         0
_________________________________________________________________
flatten_31 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_61 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_62 (Dense)             (None, 7)                 3591
=================================================================
Total params: 381,575
Trainable params: 381,575
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 4s 141us/step - loss: 5.6505 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 45us/step - loss: 5.5264 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 45us/step - loss: 5.5252 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 45us/step - loss: 5.5148 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 45us/step - loss: 5.5090 - acc: 0.2
Epoch 6/10
28709/28709 [==============================] - 1s 44us/step - loss: 5.5137 - acc: 0.2
Epoch 7/10
28709/28709 [==============================] - 1s 44us/step - loss: 5.5278 - acc: 0.2
Epoch 8/10
28709/28709 [==============================] - 1s 45us/step - loss: 5.5024 - acc: 0.2
Epoch 9/10
28709/28709 [==============================] - 1s 45us/step - loss: 5.4945 - acc: 0.2
Epoch 10/10
28709/28709 [==============================] - 1s 44us/step - loss: 5.5209 - acc: 0.2
3589/3589 [==============================] - 1s 363us/step
Accuracy on Test samples: 0.2471440512719145
Kernel Size: 3
Dropout: 0.2
Model: "sequential_32"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_125 (Conv2D)          (None, 16, 16, 64)        640
_________________________________________________________________
conv2d_126 (Conv2D)          (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_63 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_63 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_127 (Conv2D)          (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_128 (Conv2D)          (None, 3, 3, 128)         147584
_________________________________________________________________
max_pooling2d_64 (MaxPooling (None, 1, 1, 128)         0
```

```
_________________________________________________________________
dropout_64 (Dropout)         (None, 1, 1, 128)         0
_________________________________________________________________
flatten_32 (Flatten)         (None, 128)               0
_________________________________________________________________
dense_63 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_64 (Dense)             (None, 7)                 3591
=================================================================
Total params: 328,647
Trainable params: 328,647
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 4s 145us/step - loss: 2.1478 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.9520 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.9378 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9059 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8848 - acc: 0.3
Epoch 6/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8792 - acc: 0.3
Epoch 7/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.9025 - acc: 0.2
Epoch 8/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.9804 - acc: 0.2
Epoch 9/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.9621 - acc: 0.2
Epoch 10/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9537 - acc: 0.2
3589/3589 [==============================] - 1s 399us/step
Accuracy on Test samples: 0.24965171357339125
Kernel Size: 2
Dropout: 0.3
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_129 (Conv2D)          (None, 16, 16, 64)        320
_________________________________________________________________
conv2d_130 (Conv2D)          (None, 15, 15, 64)        16448
_________________________________________________________________
max_pooling2d_65 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_65 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_131 (Conv2D)          (None, 6, 6, 128)         32896
_________________________________________________________________
conv2d_132 (Conv2D)          (None, 5, 5, 128)         65664
_________________________________________________________________
max_pooling2d_66 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
dropout_66 (Dropout)         (None, 2, 2, 128)         0
_________________________________________________________________
flatten_33 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_65 (Dense)             (None, 512)               262656
```

```
_________________________________________________________________
dense_66 (Dense)             (None, 7)                 3591
=================================================================
Total params: 381,575
Trainable params: 381,575
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 4s 146us/step - loss: 2.0020 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.9467 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.9280 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.9140 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.8755 - acc: 0.3
Epoch 6/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.8854 - acc: 0.3
Epoch 7/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.8927 - acc: 0.2
Epoch 8/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.8536 - acc: 0.3
Epoch 9/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.8521 - acc: 0.3
Epoch 10/10
28709/28709 [==============================] - 1s 45us/step - loss: 1.8289 - acc: 0.3
3589/3589 [==============================] - 1s 407us/step
Accuracy on Test samples: 0.3382557815630544
Kernel Size: 3
Dropout: 0.3
Model: "sequential_34"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_133 (Conv2D)          (None, 16, 16, 64)        640
_________________________________________________________________
conv2d_134 (Conv2D)          (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_67 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_67 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_135 (Conv2D)          (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_136 (Conv2D)          (None, 3, 3, 128)         147584
_________________________________________________________________
max_pooling2d_68 (MaxPooling (None, 1, 1, 128)         0
_________________________________________________________________
dropout_68 (Dropout)         (None, 1, 1, 128)         0
_________________________________________________________________
flatten_34 (Flatten)         (None, 128)               0
_________________________________________________________________
dense_67 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_68 (Dense)             (None, 7)                 3591
=================================================================
Total params: 328,647
Trainable params: 328,647
Non-trainable params: 0
_________________________________________________________________
```

```
None
Epoch 1/10
28709/28709 [==============================] - 4s 156us/step - loss: 1.9843 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9349 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.8743 - acc: 0.3
Epoch 4/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8782 - acc: 0.3
Epoch 5/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8071 - acc: 0.3
Epoch 6/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.7780 - acc: 0.3
Epoch 7/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.7639 - acc: 0.3
Epoch 8/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.7149 - acc: 0.4
Epoch 9/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.6695 - acc: 0.4
Epoch 10/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.6500 - acc: 0.4
3589/3589 [==============================] - 2s 420us/step
Accuracy on Test samples: 0.45082195598489894
Kernel Size: 2
Dropout: 0.4
Model: "sequential_35"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_137 (Conv2D)          (None, 16, 16, 64)        320
_________________________________________________________________
conv2d_138 (Conv2D)          (None, 15, 15, 64)        16448
_________________________________________________________________
max_pooling2d_69 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_69 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_139 (Conv2D)          (None, 6, 6, 128)         32896
_________________________________________________________________
conv2d_140 (Conv2D)          (None, 5, 5, 128)         65664
_________________________________________________________________
max_pooling2d_70 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
dropout_70 (Dropout)         (None, 2, 2, 128)         0
_________________________________________________________________
flatten_35 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_69 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_70 (Dense)             (None, 7)                 3591
=================================================================
Total params: 381,575
Trainable params: 381,575
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 4s 157us/step - loss: 3.9552 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 46us/step - loss: 3.8373 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 45us/step - loss: 3.8247 - acc: 0.2
```

```
Epoch 4/10
28709/28709 [==============================] - 1s 45us/step - loss: 3.8246 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 45us/step - loss: 3.8046 - acc: 0.2
Epoch 6/10
28709/28709 [==============================] - 1s 44us/step - loss: 3.7788 - acc: 0.3
Epoch 7/10
28709/28709 [==============================] - 1s 45us/step - loss: 3.7734 - acc: 0.3
Epoch 8/10
28709/28709 [==============================] - 1s 45us/step - loss: 3.7678 - acc: 0.3
Epoch 9/10
28709/28709 [==============================] - 1s 45us/step - loss: 3.7833 - acc: 0.3
Epoch 10/10
28709/28709 [==============================] - 1s 46us/step - loss: 3.7560 - acc: 0.3
3589/3589 [==============================] - 2s 445us/step
Accuracy on Test samples: 0.34717191419052723
Kernel Size: 3
Dropout: 0.4
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_141 (Conv2D)          (None, 16, 16, 64)        640
_________________________________________________________________
conv2d_142 (Conv2D)          (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_71 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_71 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_143 (Conv2D)          (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_144 (Conv2D)          (None, 3, 3, 128)         147584
_________________________________________________________________
max_pooling2d_72 (MaxPooling (None, 1, 1, 128)         0
_________________________________________________________________
dropout_72 (Dropout)         (None, 1, 1, 128)         0
_________________________________________________________________
flatten_36 (Flatten)         (None, 128)               0
_________________________________________________________________
dense_71 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_72 (Dense)             (None, 7)                 3591
=================================================================
Total params: 328,647
Trainable params: 328,647
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 5s 162us/step - loss: 2.0823 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.9556 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9332 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8788 - acc: 0.3
Epoch 5/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8616 - acc: 0.3
Epoch 6/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9176 - acc: 0.2
```

```
Epoch 7/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.8957 - acc: 0.3
Epoch 8/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8530 - acc: 0.3
Epoch 9/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.8071 - acc: 0.3
Epoch 10/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.7736 - acc: 0.3
3589/3589 [==============================] - 2s 444us/step
Accuracy on Test samples: 0.41432153804118205
Kernel Size: 2
Dropout: 0.5
Model: "sequential_37"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_145 (Conv2D)          (None, 16, 16, 64)        320
_________________________________________________________________
conv2d_146 (Conv2D)          (None, 15, 15, 64)        16448
_________________________________________________________________
max_pooling2d_73 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_73 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_147 (Conv2D)          (None, 6, 6, 128)         32896
_________________________________________________________________
conv2d_148 (Conv2D)          (None, 5, 5, 128)         65664
_________________________________________________________________
max_pooling2d_74 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
dropout_74 (Dropout)         (None, 2, 2, 128)         0
_________________________________________________________________
flatten_37 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_73 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_74 (Dense)             (None, 7)                 3591
=================================================================
Total params: 381,575
Trainable params: 381,575
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 5s 169us/step - loss: 2.0850 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.9683 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.9480 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9409 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9328 - acc: 0.2
Epoch 6/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9206 - acc: 0.2
Epoch 7/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8985 - acc: 0.2
Epoch 8/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.8919 - acc: 0.3
Epoch 9/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.8900 - acc: 0.3
Epoch 10/10
```

```
28709/28709 [==============================] - 1s 46us/step - loss: 1.8610 - acc: 0.3
3589/3589 [==============================] - 2s 465us/step
Accuracy on Test samples: 0.35775982171056264
Kernel Size: 3
Dropout: 0.5
Model: "sequential_38"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_149 (Conv2D)          (None, 16, 16, 64)        640
_________________________________________________________________
conv2d_150 (Conv2D)          (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_75 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_75 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_151 (Conv2D)          (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_152 (Conv2D)          (None, 3, 3, 128)         147584
_________________________________________________________________
max_pooling2d_76 (MaxPooling (None, 1, 1, 128)         0
_________________________________________________________________
dropout_76 (Dropout)         (None, 1, 1, 128)         0
_________________________________________________________________
flatten_38 (Flatten)         (None, 128)               0
_________________________________________________________________
dense_75 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_76 (Dense)             (None, 7)                 3591
=================================================================
Total params: 328,647
Trainable params: 328,647
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 5s 174us/step - loss: 2.1674 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9582 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.9515 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.9384 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 46us/step - loss: 1.9252 - acc: 0.2
Epoch 6/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9082 - acc: 0.2
Epoch 7/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8795 - acc: 0.3
Epoch 8/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.8527 - acc: 0.3
Epoch 9/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.9342 - acc: 0.2
Epoch 10/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9389 - acc: 0.2
3589/3589 [==============================] - 2s 493us/step
Accuracy on Test samples: 0.289774310397019
[0.24965171357339125, 0.3700195040567302, 0.2471440512719145, 0.24965171357339125, 0.
```

```
best=(np.argmax(results))
print('Best Feature Pair is: {0} and {1}'.format(featureK[best], featureD[best]))
print("Best performance from training:",results[best])
kSize = featureK[best]
dropout_rate = featureD[best]
common_features = [Conv2D(64, kernel_size=(kSize), strides=3, activation='relu', input_
            Conv2D(64, kernel_size=(kSize), activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(dropout_rate),
            Conv2D(128, kernel_size=(kSize), activation='relu'),
            Conv2D(128, kernel_size=(kSize), activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(dropout_rate), Flatten(),]

classifier = [Dense(512, activation='relu'), Dense(7, activation='relu'),]
cnn_model = Sequential(common_features+classifier)
print("Kernel Size: {0} \nDropout: {1}".format(kSize,dropout_rate))
print(cnn_model.summary())
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'
# Train model
cnn_model.fit(train_images_3d, to_categorical(train_labels), epochs=10, batch_size=256,

performance = cnn_model.evaluate(test_images_3d, to_categorical(test_labels))
print("Accuracy on Test samples: {0}".format(performance[1]))
```

```
Best Feature Pair is: 3 and 0.3
Best performance from training: 0.45082195598489894
Kernel Size: 3
Dropout: 0.3
Model: "sequential_40"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_157 (Conv2D)          (None, 16, 16, 64)        640
_________________________________________________________________
conv2d_158 (Conv2D)          (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_79 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_79 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_159 (Conv2D)          (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_160 (Conv2D)          (None, 3, 3, 128)         147584
_________________________________________________________________
max_pooling2d_80 (MaxPooling (None, 1, 1, 128)         0
_________________________________________________________________
dropout_80 (Dropout)         (None, 1, 1, 128)         0
_________________________________________________________________
flatten_40 (Flatten)         (None, 128)               0
_________________________________________________________________
dense_79 (Dense)             (None, 512)               66048
_________________________________________________________________
dense_80 (Dense)             (None, 7)                 3591
=================================================================
Total params: 328,647
Trainable params: 328,647
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
28709/28709 [==============================] - 5s 182us/step - loss: 1.9924 - acc: 0.
Epoch 2/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.9631 - acc: 0.2
Epoch 3/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9470 - acc: 0.2
Epoch 4/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9074 - acc: 0.2
Epoch 5/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.8598 - acc: 0.3
Epoch 6/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.8313 - acc: 0.3
Epoch 7/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9253 - acc: 0.2
Epoch 8/10
28709/28709 [==============================] - 1s 47us/step - loss: 1.9469 - acc: 0.2
Epoch 9/10
28709/28709 [==============================] - 1s 49us/step - loss: 1.8832 - acc: 0.3
Epoch 10/10
28709/28709 [==============================] - 1s 48us/step - loss: 1.8336 - acc: 0.3
3589/3589 [==============================] - 2s 522us/step
Accuracy on Test samples: 0.2449150181191982
```

```
def optimize_cnn(hyperparameter):

  # Define model using hyperparameters
  cnn_model = Sequential([Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], ac
            Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
            Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
            Flatten(),
            Dense(512, activation='relu'),
            Dense(7, activation='softmax'),])

  cnn_model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentro
  time_callback = TimeHistory()
  hist = cnn_model.fit(train_images_3d, to_categorical(train_labels), epochs=10, batch_
  # print(hist.history.accuracy)
  print("Total Training Time is (s): ", sum(time_callback.times))
  # Evaluate accuracy on validation data
  performance = cnn_model.evaluate(valid_images_3d, to_categorical(valid_labels), verbo

  print("Hyperparameters: ", hyperparameter, "Accuracy: ", performance[1])
  print("------------------------------------------------------")
  # We want to minimize loss i.e. negative of accuracy
  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, 5]),
    # Uniform distribution in finding appropriate dropout values
    'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.35),
    # Choice of optimizer
    'optimizer': hp.choice('optimizer', ['Adam', 'sgd']),
}

trials = Trials()

# Find the best hyperparameters
best = fmin(
        optimize_cnn,
        space,
        algo=tpe.suggest,
        trials=trials,
        max_evals=25,
    )

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

# You can retrain the final model with optimal hyperparameters on train+validation data

# Or you can use the model returned directly
# Find trial which has minimum loss value and use that model to perform evaluation on t
```

```
# Find trial which has minimum loss value and use that model to perform evaluation on t
test_model = trials.results[np.argmin([r['loss'] for r in trials.results])]['model']

performance = test_model.evaluate(test_images_3d, to_categorical(test_labels))

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

```
Total Training Time is (s):
28.2460458278656
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.14668735307891242, 'optimizer': 'sgd'}
Accuracy:
0.3399275564265537
--------------------------------------------------
Total Training Time is (s):
23.3261399269104
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.17386172811548062, 'optimizer': 'sgd'}
Accuracy:
0.3265533575144077
--------------------------------------------------
Total Training Time is (s):
29.898162126541138
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.16029133971157797, 'optimizer': 'Adam'}
Accuracy:
0.5770409585174726
--------------------------------------------------
Total Training Time is (s):
28.340755224227905
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.19075674893547134, 'optimizer': 'sgd'}
Accuracy:
0.3558094176692397
--------------------------------------------------
Total Training Time is (s):
28.23865532875061
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2915313031622959, 'optimizer': 'sgd'}
Accuracy:
0.3349122318236002
--------------------------------------------------
Total Training Time is (s):
29.37062120437622
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.34814906012233493, 'optimizer': 'Adam'}
Accuracy:
0.5884647534464222
--------------------------------------------------
Total Training Time is (s):
29.427371740341187
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.12730390537713165, 'optimizer': 'Adam'}
Accuracy:
0.5795486208189494
--------------------------------------------------
Total Training Time is (s):
23.43778681755066
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.2914162541212033, 'optimizer': 'sgd'}
Accuracy:
0.32683198663410484
--------------------------------------------------
Total Training Time is (s):
23.60374140739441
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.23456208751505514, 'optimizer': 'sgd'}
Accuracy:
```

```
0.3413207021786596
----------------------------------------------------
Total Training Time is (s):
25.58343744277954
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.21816456764498765, 'optimizer': 'Adam'}
Accuracy:
0.4611312343312625
----------------------------------------------------
 40%|████      | 10/25 [04:58<07:27, 29.83s/it, best loss: -0.5884647534464222]
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-18-dd4c7d03b67d> in <module>()
     44         algo=tpe.suggest,
     45         trials=trials,
---> 46         max_evals=25,
     47     )
     48

                                 12 frames
/tensorflow-1.15.0/python3.6/tensorflow_core/python/client/session.py in __call__(sel
   1470         ret = tf_session.TF_SessionRunCallable(self._session._session,
   1471                                                self._handle, args,
-> 1472                                                run_metadata_ptr)
   1473         if run_metadata:
   1474           proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

FINE TUNING

```
1 from keras.datasets import mnist
2 from keras.models import Model
3 from keras.layers import Input
4  #Load MNISt dataset
5 (train_imagesf, train_labelsf), (test_imagesf, test_labelsf) = mnist.load_data()
6 train_imagesf = (train_imagesf / 255) - 0.5
7 test_imagesf = (test_imagesf / 255) - 0.5
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

```
1 train_images_3df = train_imagesf.reshape(60000,28,28,1)
2 test_images_3df = test_imagesf.reshape(10000,28,28,1)
3 train_images_3df = train_images_3df[:28709]
4 train_labelsf = train_labelsf[:28709]
5 test_images_3df = test_images_3df[:3589]
6 test_labelsf = test_labelsf[:3589]
7
```

```
Base_feature_model = Sequential([Conv2D(32, kernel_size=3, activation='relu'),
            Conv2D(32, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25),
            Conv2D(64, kernel_size=3, activation='relu'),
            Conv2D(64, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25),
            Dense(512, activation='relu'), Flatten(),])

# Define task-specific classifiation layers
Classifier_mnist = Dense(10, activation='softmax', name='MNIST')
Classifier_fer = Dense(7, activation='softmax', name='FER')

# Instantiate a Tensor to feed Input (Input Layer)
mnist_input = Input(shape=(28,28,1))
fer_input = Input(shape=(48,48,1))

# Call Base_feature_model over the mnist images
mnist_features = Base_feature_model(mnist_input)

# Call Base_feature_model over the fashion-mnist images
fer_features = Base_feature_model(fer_input)

# Call mnist_prediction layer over the mnist images
# mnist_prediction represents the predicted output for mnist dataset
mnist_prediction = Classifier_mnist(mnist_features)

# Call fashion_mnist_prediction layer over the mnist images
# fashion_mnist_prediction represents the predicted output for fashion-mnist dataset
fer_prediction = Classifier_fer(fer_features)


joint_model = Model(inputs=[mnist_input, fer_input],
                    outputs=[mnist_prediction, fer_prediction])

print(joint_model.summary())

joint_model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'],)

joint_model.fit([train_images_3df, train_images_3d],
                [to_categorical(train_labelsf), to_categorical(train_labels)],
                epochs=2, batch_size=1024,)
performance = joint_model.evaluate([test_images_3df, valid_images_3d],
                                   [to_categorical(test_labelsf),
                                    to_categorical(valid_labels)], verbose=1)

print("===\nMNIST Accuracy: {0}\nFER Accuracy: {1}".format(performance[3], performance[
performance = joint_model.evaluate([test_images_3df, test_images_3d],
                                   [to_categorical(test_labelsf),
                                    to_categorical(test_labels)], verbose=1)

print("===\nMNIST Accuracy: {0}\nFER Accuracy: {1}".format(performance[3], performance[
```

```
Model: "model_1"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_1 (InputLayer)            (None, 28, 28, 1)    0
__________________________________________________________________________________________
input_2 (InputLayer)            (None, 48, 48, 1)    0
__________________________________________________________________________________________
sequential_2 (Sequential)       multiple             98272       input_1[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________
MNIST (Dense)                   (None, 10)           81930       sequential_2[1][0]
__________________________________________________________________________________________
FER (Dense)                     (None, 7)            290311      sequential_2[2][0]
==========================================================================================
Total params: 470,513
Trainable params: 470,513
Non-trainable params: 0
__________________________________________________________________________________________
None
WARNING:tensorflow:From /tensorflow-1.15.0/python3.6/tensorflow_core/python/ops/nn_im
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Epoch 1/2
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

28709/28709 [==============================] - 18s 611us/step - loss: 0.5884 - MNIST_
Epoch 2/2
28709/28709 [==============================] - 3s 116us/step - loss: 0.4026 - MNIST_l
3589/3589 [==============================] - 1s 209us/step
===
MNIST Accuracy: 0.9875731395127282
FER Accuracy: 0.8647852702936644
3589/3589 [==============================] - 1s 145us/step
===
MNIST Accuracy: 0.9875731395127282
FER Accuracy: 0.8625562360739103
```

Fine tuning using the MNIST Model.

```
import cv2
from skimage.transform import resize
# Resizing the images.
flatten_train_images = train_image_array.reshape((-1, 48,48))
flatten_test_images = test_image_array.reshape((-1, 48,48))
flatten_valid_images = valid_image_array.reshape((-1,48,48))
resized_images_train = []
resized_images_test = []
resized_images_valid = []
# for i in range(len(flatten_test_images)):
for i in range(len(train_pixels)):
  img = flatten_train_images[i]
  img=cv2.resize(img, dsize=(28,28),interpolation=cv2.INTER_CUBIC)
  resized_images_train.append(img)
resized_images_train = np.array(resized_images_train)

for i in range(len(valid_pixels)):
  img = flatten_valid_images[i]
  img=cv2.resize(img, dsize=(28,28),interpolation=cv2.INTER_CUBIC)
  resized_images_valid.append(img)
resized_images_valid = np.array(resized_images_valid)


for i in range(len(test_pixels)):
  img = flatten_test_images[i]
  img=cv2.resize(img, dsize=(28,28),interpolation=cv2.INTER_CUBIC)
  resized_images_test.append(img)
resized_images_test = np.array(resized_images_test)

```

```
from keras.layers import Conv2D, Flatten, MaxPooling2D

# Define 2 groups of layers: features layer (convolutions) and classification layer
common_features = [Conv2D(32, kernel_size=3, activation='relu', input_shape=(28,28,1)),
            Conv2D(32, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25),
            Conv2D(64, kernel_size=3, activation='relu'),
            Conv2D(64, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25), Flatten(),]
classifier = [Dense(512, activation='relu'), Dense(10, activation='softmax'),]

cnn_model = Sequential(common_features+classifier)

print(cnn_model.summary())  # Compare number of parameteres against FFN
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'

train_images_3df = train_imagesf.reshape(60000,28,28,1)
test_images_3df = test_imagesf.reshape(10000,28,28,1)

cnn_model.fit(train_images_3df, to_categorical(train_labelsf), epochs=10, batch_size=25
performance = cnn_model.evaluate(test_images_3df, to_categorical(test_labelsf))

print("Accuracy on Test samples: {0}".format(performance[1]))
```

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_31 (Dropout)         (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 64)        18496
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_32 (Dropout)         (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_31 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_32 (Dense)             (None, 10)                5130
=================================================================
Total params: 594,922
Trainable params: 594,922
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
60000/60000 [==============================] - 9s 145us/step - loss: 0.2772 - acc: 0.
Epoch 2/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0598 - acc: 0.9
Epoch 3/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0429 - acc: 0.9
Epoch 4/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0336 - acc: 0.9
Epoch 5/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0272 - acc: 0.9
Epoch 6/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0242 - acc: 0.9
Epoch 7/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0209 - acc: 0.9
Epoch 8/10
60000/60000 [==============================] - 2s 38us/step - loss: 0.0188 - acc: 0.9
Epoch 9/10
60000/60000 [==============================] - 2s 38us/step - loss: 0.0160 - acc: 0.9
Epoch 10/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0148 - acc: 0.9
10000/10000 [==============================] - 1s 124us/step
Accuracy on Test samples: 0.9944
```

```
1 for l in common_features:
2   l.trainable = False
3
```

```
classifier2 = [Dense(512, activation='relu'), Dense(7, activation='softmax'),]

cnn_model.add(Dense(7,activation='softmax'))
print(cnn_model.summary())
resized_images_valid_3d = resized_images_valid.reshape(3589,28,28,1)
resized_images_train_3d = resized_images_train.reshape(28709,28,28,1)
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'

cnn_model.fit(resized_images_train_3d, to_categorical(train_labels), epochs=8, batch_si
performance = cnn_model.evaluate(resized_images_valid_3d, to_categorical(valid_labels))

print("Accuracy on Test samples: {0}".format(performance[1]))
```

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_31 (Dropout)         (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 64)        18496
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_32 (Dropout)         (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_31 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_32 (Dense)             (None, 10)                5130
_________________________________________________________________
dense_35 (Dense)             (None, 7)                 77
=================================================================
Total params: 659,914
Trainable params: 594,922
Non-trainable params: 64,992
_________________________________________________________________
None
Epoch 1/8
28709/28709 [==============================] - 2s 76us/step - loss: 1.8593 - acc: 0.2
Epoch 2/8
28709/28709 [==============================] - 1s 23us/step - loss: 1.7825 - acc: 0.2
Epoch 3/8
28709/28709 [==============================] - 1s 25us/step - loss: 1.7408 - acc: 0.3
Epoch 4/8
28709/28709 [==============================] - 1s 25us/step - loss: 1.7033 - acc: 0.3
Epoch 5/8
28709/28709 [==============================] - 1s 23us/step - loss: 1.6698 - acc: 0.3
Epoch 6/8
28709/28709 [==============================] - 1s 24us/step - loss: 1.6397 - acc: 0.3
Epoch 7/8
28709/28709 [==============================] - 1s 24us/step - loss: 1.6091 - acc: 0.3
Epoch 8/8
28709/28709 [==============================] - 1s 24us/step - loss: 1.5825 - acc: 0.4
3589/3589 [==============================] - 1s 232us/step
Accuracy on Test samples: 0.4093062134299248
```

Data Augmentation

```
1 from keras.preprocessing.image import ImageDataGenerator
```

```
from matplotlib import pyplot
datagen = ImageDataGenerator()

#feature standardization
x_train = (train_image_array)
x_valid = (valid_image_array)
x_test = (test_image_array)

x_train = x_train.reshape((x_train.shape[0], 48, 48, 1))
x_valid = x_valid.reshape((x_valid.shape[0], 48, 48, 1))
x_test = x_test.reshape((x_valid.shape[0], 48, 48, 1))

x_train = x_train.astype('float32')
x_valid = x_valid.astype('float32')
x_test = x_test.astype('float32')

datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=Tru
datagen.fit(x_train)
datagen.fit(x_valid)
datagen.fit(x_test)
```

```
def optimize_cnn(hyperparameter):

  # Define model using hyperparameters
  cnn_model = Sequential([Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], ac
            Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
            Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
            Flatten(),
            Dense(512, activation='relu'),
            Dense(7, activation='softmax'),])

  cnn_model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentro
  time_callback = TimeHistory()
  hist = cnn_model.fit(x_train, to_categorical(train_labels), epochs=10, batch_size=256
  # print(hist.history.accuracy)
  print("Total Training Time is (s): ", sum(time_callback.times))
  # Evaluate accuracy on validation data
  performance = cnn_model.evaluate(x_valid, to_categorical(valid_labels), verbose=0)

  print("Hyperparameters: ", hyperparameter, "Accuracy: ", performance[1])
  print("------------------------------------------------------")
  # We want to minimize loss i.e. negative of accuracy
  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, 5]),
    # Uniform distribution in finding appropriate dropout values
    'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.35),
    # Choice of optimizer
```

```
    # Choice of optimizer
    'optimizer': hp.choice('optimizer', ['Adam', 'sgd']),
}

trials = Trials()

# Find the best hyperparameters
best = fmin(
        optimize_cnn,
        space,
        algo=tpe.suggest,
        trials=trials,
        max_evals=25,
    )

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

# You can retrain the final model with optimal hyperparameters on train+validation data

# Or you can use the model returned directly
# Find trial which has minimum loss value and use that model to perform evaluation on t
test_model = trials.results[np.argmin([r['loss'] for r in trials.results])]['model']

performance = test_model.evaluate(x_test, to_categorical(test_labels))

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

```
Total Training Time is (s):
26.69492268562317
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2542249469349509, 'optimizer': 'Adam'}
Accuracy:
0.5856784619754255
--------------------------------------------------
Total Training Time is (s):
26.36576533317566
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.1339069129108087, 'optimizer': 'Adam'}
Accuracy:
0.5881861242727504
--------------------------------------------------
Total Training Time is (s):
25.207025289535522
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.18646617942102145, 'optimizer': 'sgd'}
Accuracy:
0.3290610197868212
--------------------------------------------------
Total Training Time is (s):
25.25748372077942
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.2422444255685212, 'optimizer': 'sgd'}
Accuracy:
0.33630537754664286
--------------------------------------------------
Total Training Time is (s):
25.461172342300415
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.2448541881379686, 'optimizer': 'sgd'}
Accuracy:
0.31930899972552274
--------------------------------------------------
Total Training Time is (s):
26.484083890914917
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.18503897577787976, 'optimizer': 'Adam'}
Accuracy:
0.5792699916494295
--------------------------------------------------
Total Training Time is (s):
26.643880128860474
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.32063192160086557, 'optimizer': 'Adam'}
Accuracy:
0.590136528289162
--------------------------------------------------
Total Training Time is (s):
27.22426700592041
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.13249574191657534, 'optimizer': 'Adam'}
Accuracy:
0.5859570911200341
--------------------------------------------------
Total Training Time is (s):
26.097631216049194
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.29590396234361205, 'optimizer': 'sgd'}
Accuracy:
```

```
0.33769852327383737
--------------------------------------------------
Total Training Time is (s):
26.8969247341156
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.21040384851779753, 'optimizer': 'Adam'}
Accuracy:
0.5951518529170269
--------------------------------------------------
Total Training Time is (s):
27.601470947265625
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.11880611138425023, 'optimizer': 'Adam'}
Accuracy:
0.5820562830913628
--------------------------------------------------
Total Training Time is (s):
23.013309955596924
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.14665198365052354, 'optimizer': 'Adam'}
Accuracy:
0.46308163833937027
--------------------------------------------------
Total Training Time is (s):
26.464831113815308
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.18015588988489312, 'optimizer': 'sgd'}
Accuracy:
0.3134577876928956
--------------------------------------------------
Total Training Time is (s):
26.698853969573975
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2799730869754218, 'optimizer': 'sgd'}
Accuracy:
0.309835608804681
--------------------------------------------------
Total Training Time is (s):
23.167685508728027
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.160008848531309, 'optimizer': 'Adam'}
Accuracy:
0.4611312343312625
--------------------------------------------------
Total Training Time is (s):
26.600367546081543
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.16467814646642104, 'optimizer': 'sgd'}
Accuracy:
0.30342713848283676
--------------------------------------------------
Total Training Time is (s):
26.676040172576904
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.34563440284244595, 'optimizer': 'sgd'}
Accuracy:
0.30203399275979415
--------------------------------------------------
Total Training Time is (s):
27.901010513305664
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.26596686122277424, 'optimizer': 'Adam'}
```

```
{'conv_kernel_size': 5, 'dropout_prob': 0.2099060861222742, 'optimizer': 'Adam'}
Accuracy:
0.5890220117356393
------------------------------------------------------
Total Training Time is (s):
27.972566843032837
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.1641466973375958, 'optimizer': 'Adam'}
Accuracy:
0.5817776539758176
------------------------------------------------------
Total Training Time is (s):
28.094647645950317
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.3380391316682997, 'optimizer': 'Adam'}
Accuracy:
0.573697408753107
------------------------------------------------------
Total Training Time is (s):
23.941020727157593
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.2156736105684234, 'optimizer': 'Adam'}
Accuracy:
0.45082195598074704
------------------------------------------------------
Total Training Time is (s):
24.138392686843872
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.30311341285267246, 'optimizer': 'Adam'}
Accuracy:
0.4522151017037897
------------------------------------------------------
Total Training Time is (s):
28.54242205619812
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.31859347865493814, 'optimizer': 'Adam'}
Accuracy:
0.584285316248231
------------------------------------------------------
Total Training Time is (s):
28.73721957206726
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.2100158958151668, 'optimizer': 'Adam'}
Accuracy:
0.590136528289162
------------------------------------------------------
Total Training Time is (s):
28.72278118133545
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.2108128237693697, 'optimizer': 'Adam'}
Accuracy:
0.5759264419390385
------------------------------------------------------
100%|██████████| 25/25 [11:48<00:00, 28.32s/it, best loss: -0.5951518529170269]
==================================
Best Hyperparameters {'conv_kernel_size': 2, 'dropout_prob': 0.21040384851779753, 'op
3589/3589 [==============================] - 0s 92us/step
==================================
Test Accuracy:  0.6032320980816108
```

```
# ZCA Whitening
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
datagen = ImageDataGenerator()

#feature standardization
x_train = (train_image_array)
x_valid = (valid_image_array)
x_test = (test_image_array)

x_train = x_train.reshape((x_train.shape[0], 48, 48, 1))
x_valid = x_valid.reshape((x_valid.shape[0], 48, 48, 1))
x_test = x_test.reshape((x_valid.shape[0], 48, 48, 1))

x_train = x_train.astype('float32')
x_valid = x_valid.astype('float32')
x_test = x_test.astype('float32')

datagen = ImageDataGenerator(zca_whitening=True)
datagen.fit(x_train)
datagen.fit(x_valid)
datagen.fit(x_test)
```

```
def optimize_cnn(hyperparameter):

  # Define model using hyperparameters
  cnn_model = Sequential([Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], ac
            Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
            Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
            MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
            Flatten(),
            Dense(512, activation='relu'),
            Dense(7, activation='softmax'),])

  cnn_model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentro
  time_callback = TimeHistory()
  hist = cnn_model.fit(x_train, to_categorical(train_labels), epochs=10, batch_size=256
  # print(hist.history.accuracy)
  print("Total Training Time is (s): ", sum(time_callback.times))
  # Evaluate accuracy on validation data
  performance = cnn_model.evaluate(x_valid, to_categorical(valid_labels), verbose=0)

  print("Hyperparameters: ", hyperparameter, "Accuracy: ", performance[1])
  print("------------------------------------------------------")
  # We want to minimize loss i.e. negative of accuracy
  return({"status": STATUS_OK, "loss": -1*performance[1], "model":cnn_model})

```

```
27
28 # Define search space for hyper-parameters
29 space = {
30     # The kernel_size for convolutions:
31     'conv_kernel_size': hp.choice('conv_kernel_size', [1, 3, 5]),
32     # Uniform distribution in finding appropriate dropout values
33     'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.35),
34     # Choice of optimizer
35     'optimizer': hp.choice('optimizer', ['Adam', 'sgd']),
36 }
37
38 trials = Trials()
39
40 # Find the best hyperparameters
41 best = fmin(
42         optimize_cnn,
43         space,
44         algo=tpe.suggest,
45         trials=trials,
46         max_evals=25,
47     )
48
49 print("==================================")
50 print("Best Hyperparameters", best)
51
52 # You can retrain the final model with optimal hyperparameters on train+validation data
53
54 # Or you can use the model returned directly
55 # Find trial which has minimum loss value and use that model to perform evaluation on t
56 test_model = trials.results[np.argmin([r['loss'] for r in trials.results])]['model']
57
58 performance = test_model.evaluate(x_test, to_categorical(test_labels))
59
60 print("==================================")
61 print("Test Accuracy: ", performance[1])
```

```
Total Training Time is (s):
28.13971781730652
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.24814050988778366, 'optimizer': 'sgd'}
Accuracy:
0.34912231820278694
----------------------------------------------------
Total Training Time is (s):
29.38590717315674
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.25791132899874547, 'optimizer': 'Adam'}
Accuracy:
0.574811925335693
----------------------------------------------------
Total Training Time is (s):
25.396419286727905
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.22730878575841162, 'optimizer': 'Adam'}
Accuracy:
0.4413485650640572
----------------------------------------------------
Total Training Time is (s):
23.52994656562805
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.3356007393482103, 'optimizer': 'sgd'}
Accuracy:
0.3170799665977178
----------------------------------------------------
Total Training Time is (s):
28.708142280578613
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.11560059994968835, 'optimizer': 'sgd'}
Accuracy:
0.3455001393187242
----------------------------------------------------
Total Training Time is (s):
25.343907117843628
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.1569881114299798, 'optimizer': 'Adam'}
Accuracy:
0.45555865143909197
----------------------------------------------------
Total Training Time is (s):
23.794260263442993
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.14529594832766432, 'optimizer': 'sgd'}
Accuracy:
0.3232098077500421
----------------------------------------------------
Total Training Time is (s):
29.731204509735107
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.14865966844802445, 'optimizer': 'Adam'}
Accuracy:
0.5959877403508524
----------------------------------------------------
Total Training Time is (s):
29.16386127471924
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.1668994273753261, 'optimizer': 'sgd'}
Accuracy:
```

```
0.29757592644190584
--------------------------------------------------
Total Training Time is (s):
24.0995192527771
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.34520159231583475, 'optimizer': 'sgd'}
Accuracy:
0.3037057676274453
--------------------------------------------------
Total Training Time is (s):
25.988826990127563
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.11260333076964793, 'optimizer': 'Adam'}
Accuracy:
0.44552800223733696
--------------------------------------------------
Total Training Time is (s):
24.36750626564026
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.10067291427456249, 'optimizer': 'sgd'}
Accuracy:
0.3240456951838677
--------------------------------------------------
Total Training Time is (s):
29.36147117614746
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.3329262177643334, 'optimizer': 'sgd'}
Accuracy:
0.31707996656865456
--------------------------------------------------
Total Training Time is (s):
24.480132341384888
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.2987819312451138, 'optimizer': 'sgd'}
Accuracy:
0.29311786013647323
--------------------------------------------------
Total Training Time is (s):
30.930524110794067
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.1979817749585424, 'optimizer': 'Adam'}
Accuracy:
0.5812203956616891
--------------------------------------------------
Total Training Time is (s):
29.91033911705017
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.1050616079800136, 'optimizer': 'sgd'}
Accuracy:
0.35218723879348074
--------------------------------------------------
Total Training Time is (s):
29.913865327835083
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.1531927927777971, 'optimizer': 'sgd'}
Accuracy:
0.29172471440512676
--------------------------------------------------
Total Training Time is (s):
30.116098642349243
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.16843832400704467, 'optimizer': 'sgd'}
```

{'conv_kernel_size': 5, 'dropout_prob': 0.18049532400704407, 'optimizer': 'sgd'}
Accuracy:
0.3507940930787419

Total Training Time is (s):
30.94815754890442
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.15462614464529809, 'optimizer': 'Adam'}
Accuracy:
0.5876288660125967

Total Training Time is (s):
30.283236026763916
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.11775894597544814, 'optimizer': 'sgd'}
Accuracy:
0.2942323767149073

Total Training Time is (s):
31.25113534927368
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.1959757796377455, 'optimizer': 'Adam'}
Accuracy:
0.5748119253315411

Total Training Time is (s):
31.228559494018555
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.1930187501789366, 'optimizer': 'Adam'}
Accuracy:
0.5912510448634442

Total Training Time is (s):
31.54096746444702
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.19116528768022692, 'optimizer': 'Adam'}
Accuracy:
0.5834494288185573

Total Training Time is (s):
31.804988145828247
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.22138591652886783, 'optimizer': 'Adam'}
Accuracy:
0.577598216790082

Total Training Time is (s):
31.88996720314026
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.13687584560602467, 'optimizer': 'Adam'}
Accuracy:
0.5851212037111199

100%|██████████| 25/25 [13:03<00:00, 31.35s/it, best loss: -0.5959877403508524]
==================================
Best Hyperparameters {'conv_kernel_size': 2, 'dropout_prob': 0.14865966844802445, 'op
3589/3589 [==============================] - 0s 112us/step
==================================
Test Accuracy: 0.5918083031568131

```
# Random Rotations
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
datagen = ImageDataGenerator()

#feature standardization
x_train = (train_image_array)
x_valid = (valid_image_array)
x_test = (test_image_array)

x_train = x_train.reshape((x_train.shape[0], 48, 48, 1))
x_valid = x_valid.reshape((x_valid.shape[0], 48, 48, 1))
x_test = x_test.reshape((x_valid.shape[0], 48, 48, 1))

x_train = x_train.astype('float32')
x_valid = x_valid.astype('float32')
x_test = x_test.astype('float32')

datagen = ImageDataGenerator(rotation_range=90)
datagen.fit(x_train)
datagen.fit(x_valid)
datagen.fit(x_test)
```

```
def optimize_cnn(hyperparameter):

  # Define model using hyperparameters
  cnn_model = Sequential([Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], ac
             Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
             MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
             Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
             Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
             MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
             Flatten(),
             Dense(512, activation='relu'),
             Dense(7, activation='softmax'),])

  cnn_model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentro
  time_callback = TimeHistory()
  hist = cnn_model.fit(x_train, to_categorical(train_labels), epochs=10, batch_size=256
  # print(hist.history.accuracy)
  print("Total Training Time is (s): ", sum(time_callback.times))
  # Evaluate accuracy on validation data
  performance = cnn_model.evaluate(x_valid, to_categorical(valid_labels), verbose=0)

  print("Hyperparameters: ", hyperparameter, "Accuracy: ", performance[1])
  print("------------------------------------------------------")
  # We want to minimize loss i.e. negative of accuracy
  return({"status": STATUS_OK, "loss": -1*performance[1], "model":cnn_model})

```

```
27
28 # Define search space for hyper-parameters
29 space = {
30     # The kernel_size for convolutions:
31     'conv_kernel_size': hp.choice('conv_kernel_size', [1, 3, 5]),
32     # Uniform distribution in finding appropriate dropout values
33     'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.35),
34     # Choice of optimizer
35     'optimizer': hp.choice('optimizer', ['Adam', 'sgd']),
36 }
37
38 trials = Trials()
39
40 # Find the best hyperparameters
41 best = fmin(
42         optimize_cnn,
43         space,
44         algo=tpe.suggest,
45         trials=trials,
46         max_evals=25,
47     )
48
49 print("==================================")
50 print("Best Hyperparameters", best)
51
52 # You can retrain the final model with optimal hyperparameters on train+validation data
53
54 # Or you can use the model returned directly
55 # Find trial which has minimum loss value and use that model to perform evaluation on t
56 test_model = trials.results[np.argmin([r['loss'] for r in trials.results])]['model']
57
58 performance = test_model.evaluate(x_test, to_categorical(test_labels))
59
60 print("==================================")
61 print("Test Accuracy: ", performance[1])
```

```
Total Training Time is (s):
31.095835208892822
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.2429661912776022, 'optimizer': 'sgd'}
Accuracy:
0.337977152414294
--------------------------------------------------
Total Training Time is (s):
31.687371969223022
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.3226159513161642, 'optimizer': 'sgd'}
Accuracy:
0.3482864307689614
--------------------------------------------------
Total Training Time is (s):
28.15788960456848
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.1697117944569278, 'optimizer': 'Adam'}
Accuracy:
0.4446921147993595
--------------------------------------------------
Total Training Time is (s):
32.000370264053345
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.21779164306545729, 'optimizer': 'sgd'}
Accuracy:
0.3307327946586242
--------------------------------------------------
Total Training Time is (s):
32.85355305671692
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.10663941492259457, 'optimizer': 'sgd'}
Accuracy:
0.35135135135550327
--------------------------------------------------
Total Training Time is (s):
32.57329607009888
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.33037373096037836, 'optimizer': 'sgd'}
Accuracy:
0.3379771524433573
--------------------------------------------------
Total Training Time is (s):
32.80647110939026
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.1381682853825744, 'optimizer': 'sgd'}
Accuracy:
0.3296182780760382
--------------------------------------------------
Total Training Time is (s):
28.958810329437256
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.34275056286314465, 'optimizer': 'Adam'}
Accuracy:
0.4419058233823375
--------------------------------------------------
Total Training Time is (s):
32.93373417854309
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.3145589369201526, 'optimizer': 'sgd'}
Accuracy:
```

```
0.3207021454485654
--------------------------------------------------
Total Training Time is (s):
34.28903603553772
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.33845931234106297, 'optimizer': 'Adam'}
Accuracy:
0.5876288660125967
--------------------------------------------------
Total Training Time is (s):
27.753254175186157
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.2607511117740823, 'optimizer': 'sgd'}
Accuracy:
0.3296182780801901
--------------------------------------------------
Total Training Time is (s):
33.09608292579651
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.13697860925216643, 'optimizer': 'sgd'}
Accuracy:
0.34410699362474484
--------------------------------------------------
Total Training Time is (s):
32.954015254974365
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.10060224254979439, 'optimizer': 'sgd'}
Accuracy:
0.3402061855711622
--------------------------------------------------
Total Training Time is (s):
33.843745470047
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.19430122651964926, 'optimizer': 'Adam'}
Accuracy:
0.5984954026274177
--------------------------------------------------
Total Training Time is (s):
27.96817183494568
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.11444065209952078, 'optimizer': 'sgd'}
Accuracy:
0.3324045695262754
--------------------------------------------------
Total Training Time is (s):
33.4222207069397
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.10390735075834195, 'optimizer': 'sgd'}
Accuracy:
0.3482864307689614
--------------------------------------------------
Total Training Time is (s):
33.40985345840454
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.18057138820927132, 'optimizer': 'sgd'}
Accuracy:
0.32794650321253893
--------------------------------------------------
Total Training Time is (s):
33.81111192703247
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.2277213692352072, 'optimizer': 'Adam'}
```

{'conv_kernel_size': 3, 'dropout_prob': 0.22721909259926[illegible], 'optimizer': 'Adam'}
Accuracy:
0.5976595151935922

Total Training Time is (s):
34.45823407173157
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.10651244975702923, 'optimizer': 'Adam'}
Accuracy:
0.5684034550346083

Total Training Time is (s):
34.07155179977417
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.29316343846787873, 'optimizer': 'Adam'}
Accuracy:
0.591529674037116

Total Training Time is (s):
34.26889753341675
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.20573788361981166, 'optimizer': 'Adam'}
Accuracy:
0.5739760379018675

Total Training Time is (s):
34.23374938964844
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.2738452663694213, 'optimizer': 'Adam'}
Accuracy:
0.5982167734828092

Total Training Time is (s):
34.337204694747925
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.27431157338690143, 'optimizer': 'Adam'}
Accuracy:
0.5851212036862085

Total Training Time is (s):
35.21518063545227
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.1917634352204038, 'optimizer': 'Adam'}
Accuracy:
0.5834494288185573

Total Training Time is (s):
35.195539474487305
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.29441982460016747, 'optimizer': 'Adam'}
Accuracy:
0.5639453886959606

100%|██████████| 25/25 [15:16<00:00, 36.66s/it, best loss: -0.5984954026274177]
==================================
Best Hyperparameters {'conv_kernel_size': 2, 'dropout_prob': 0.19430122651964926, 'op
3589/3589 [==============================] - 0s 123us/step
==================================
Test Accuracy: 0.5770409584925612

FEATURE Design

```
from skimage.feature import hog
from skimage import data, exposure

x_train = (train_image_array)
x_valid = (valid_image_array)
x_test = (test_image_array)

x_train = x_train.reshape((x_train.shape[0], 48, 48))
x_valid = x_valid.reshape((x_valid.shape[0], 48, 48))
x_test = x_test.reshape((x_test.shape[0], 48, 48))

hog_train = []
for i in range(len(x_train)):
  image = x_train[i]
  fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                      cells_per_block=(1, 1), visualize=True, multichannel=False)
  hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
  hog_train.append(hog_image_rescaled)

hog_valid = []
for i in range(len(x_valid)):
  image = x_valid[i]
  fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                      cells_per_block=(1, 1), visualize=True, multichannel=False)
  hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
  hog_valid.append(hog_image_rescaled)

hog_test = []
for i in range(len(x_test)):
  image = x_test[i]
  fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                      cells_per_block=(1, 1), visualize=True, multichannel=False)
  hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
  hog_test.append(hog_image_rescaled)

x_train = np.array(hog_train)
x_valid = np.array(hog_valid)
x_test = np.array(hog_test)

x_train = x_train.reshape((x_train.shape[0], 48, 48, 1))
x_valid = x_valid.reshape((x_valid.shape[0], 48, 48, 1))
x_test = x_test.reshape((x_test.shape[0], 48, 48, 1))
```

```

def optimize_cnn(hyperparameter):
```

```
 3
 4   # Define model using hyperparameters
 5   cnn_model = Sequential([Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], ac
 6             Conv2D(32, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
 7             MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
 8             Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
 9             Conv2D(64, kernel_size=hyperparameter['conv_kernel_size'], activation='relu
10             MaxPooling2D(pool_size=(2,2)), Dropout(hyperparameter['dropout_prob']),
11             Flatten(),
12             Dense(512, activation='relu'),
13             Dense(7, activation='softmax'),])
14
15   cnn_model.compile(optimizer=hyperparameter['optimizer'], loss='categorical_crossentro
16   time_callback = TimeHistory()
17   hist = cnn_model.fit(x_train, to_categorical(train_labels), epochs=10, batch_size=256
18   # print(hist.history.accuracy)
19   print("Total Training Time is (s): ", sum(time_callback.times))
20   # Evaluate accuracy on validation data
21   performance = cnn_model.evaluate(x_valid, to_categorical(valid_labels), verbose=0)
22
23   print("Hyperparameters: ", hyperparameter, "Accuracy: ", performance[1])
24   print("------------------------------------------------------")
25   # We want to minimize loss i.e. negative of accuracy
26   return({"status": STATUS_OK, "loss": -1*performance[1], "model":cnn_model})
27
28
29 # Define search space for hyper-parameters
30 space = {
31     # The kernel_size for convolutions:
32     'conv_kernel_size': hp.choice('conv_kernel_size', [1, 3, 5]),
33     # Uniform distribution in finding appropriate dropout values
34     'dropout_prob': hp.uniform('dropout_prob', 0.1, 0.35),
35     # Choice of optimizer
36     'optimizer': hp.choice('optimizer', ['Adam', 'sgd']),
37 }
38
39 trials = Trials()
40
41 # Find the best hyperparameters
42 best = fmin(
43         optimize_cnn,
44         space,
45         algo=tpe.suggest,
46         trials=trials,
47         max_evals=25,
48     )
49
50 print("==================================")
51 print("Best Hyperparameters", best)
52
53 # You can retrain the final model with optimal hyperparameters on train+validation data
54
55 # Or you can use the model returned directly
56 # Find trial which has minimum loss value and use that model to perform evaluation on t
57 test_model = trials.results[np.argmin([r['loss'] for r in trials.results])]['model']
```

```
58
59 performance = test_model.evaluate(x_test, to_categorical(test_labels))
60
61 print("==================================")
62 print("Test Accuracy: ", performance[1])
```

```
Total Training Time is (s):
30.280741214752197
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.15049653462509077, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
26.057782888412476
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.3327359916145542, 'optimizer': 'Adam'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
25.993083477020264
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2399034427313376, 'optimizer': 'Adam'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
24.862027883529663
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.15711783288164408, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
26.145740032196045
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.13095244695573482, 'optimizer': 'Adam'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
21.854387521743774
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.3277228071731925, 'optimizer': 'Adam'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
21.76009750366211
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.12063233345739746, 'optimizer': 'Adam'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
26.186025619506836
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.1300360838648438, 'optimizer': 'Adam'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
25.091683626174927
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.31162659374425034, 'optimizer': 'sgd'}
Accuracy:
```

```
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
26.44800066947937
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.32473622789596435, 'optimizer': 'Adam'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
25.53569984436035
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.25676837994360124, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
25.3596453666687
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.1996249235229065, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
26.942335844039917
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2226626407591457, 'optimizer': 'Adam'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
25.856187105178833
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.2767054261515769, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
25.78175401687622
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.25144256564530276, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
21.05579113960266
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.1795332127981628, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
21.038676261901855
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.3163431451188834, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273
--------------------------------------------------
Total Training Time is (s):
26.15276527404785
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.19093288081246018, 'optimizer': 'sgd'}
```

{'conv_kernel_size': 3, 'dropout_prob': 0.19099280881240018, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273

Total Training Time is (s):
21.135244607925415
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.21503236289103644, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273

Total Training Time is (s):
26.280521392822266
Hyperparameters:
{'conv_kernel_size': 3, 'dropout_prob': 0.18627258090208043, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273

Total Training Time is (s):
26.11707830429077
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.15518590423516235, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273

Total Training Time is (s):
21.610968351364136
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.2137888917775868, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273

Total Training Time is (s):
26.30264401435852
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.1607458205645651, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273

Total Training Time is (s):
26.25912308692932
Hyperparameters:
{'conv_kernel_size': 5, 'dropout_prob': 0.10813003008178762, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273

Total Training Time is (s):
21.570691347122192
Hyperparameters:
{'conv_kernel_size': 1, 'dropout_prob': 0.27555616085888457, 'optimizer': 'sgd'}
Accuracy:
0.24937308442878273

100%|██████████| 25/25 [10:50<00:00, 26.00s/it, best loss: -0.24937308442878273]
==================================
Best Hyperparameters {'conv_kernel_size': 2, 'dropout_prob': 0.15049653462509077, 'op
3589/3589 [==============================] - 0s 87us/step
==================================
Test Accuracy: 0.2449150181191982