import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from matplotlib import image  
from PIL import Image  
import itertools  
  
import tensorflow as tf  
  
from keras.utils.np\_utils import to\_categorical  
from keras.models import Sequential  
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D  
from keras.optimizers import RMSprop  
from keras.preprocessing.image import ImageDataGenerator  
from keras.callbacks import ReduceLROnPlateau  
  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix

gimage = []  
for i in range(1, 20):  
 im = Image.open('./Images/image ('+ str(i) +').jpg').convert('L').resize((28, 28)) # Loading Images and Converting to Grayscale  
 gimage.append(np.asarray(im)) # Appending in List after converting to np.array

# Plotting images from arrays in gimage list  
for j in range(len(gimage)):  
 plt.imshow(gimage[j], cmap = 'gray', vmin = 0, vmax = 255)  
 plt.show()

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

png

# flattening the arrays in gimage list to convert to dataframe  
fimage = []  
for k in range(len(gimage)):  
 fimage.append(gimage[k].flatten())

# converting to dataframe  
dataset = pd.DataFrame(fimage)

dataset.head(5)

0

1

2

3

4

5

6

7

8

9

...

774

775

776

777

778

779

780

781

782

783

0

207

208

210

212

213

213

213

210

202

202

...

221

225

232

236

236

235

234

231

229

230

1

237

164

146

185

200

198

198

198

198

196

...

244

244

244

244

244

244

244

244

244

244

2

208

226

224

216

232

242

244

241

239

235

...

185

140

149

162

142

83

84

113

120

106

3

134

155

154

154

164

166

165

166

163

163

...

176

165

163

164

164

160

163

171

165

162

4

61

62

79

88

106

114

109

97

140

155

...

212

209

201

201

203

208

210

200

184

176

5 rows × 784 columns

# Plotting images from the flattened fimage list for verification  
for m in range(5):  
 plt.imshow(fimage[m].reshape(28, 28), cmap = 'gray', vmin = 0, vmax = 255)  
 plt.show()

png

png

png

png

png

png

png

png

png

png

# Adding labels to the dataset {0: null, 1: happy, 2: surprised}  
# label = [[0.,1.,0.], [0.,0.,1.], [1.,0.,0.], [0.,0.,1.], [0.,0.,1.], [0.,1.,0.], [0.,1.,0.], [1.,0.,0.], [0.,0.,1.], [0.,0.,1.], [0.,0.,1.], [1.,0.,0.], [1.,0.,0.], [0.,0.,1.], [0.,1.,0.], [0.,0.,1.], [0.,1.,0.], [0.,0.,1.], [0.,1.,0.]]  
label = [0, 1, 2, 1, 1, 0, 0, 2, 1, 1, 1, 2, 2, 1, 0, 1, 0, 1, 0]  
dataset['label'] = label

# Saving dataframe to directory for further use  
dataset.to\_csv(r'dataset.csv')

dataset = pd.read\_csv('dataset.csv').drop(columns = 'Unnamed: 0')  
dataset.head(5)

0

1

2

3

4

5

6

7

8

9

...

775

776

777

778

779

780

781

782

783

label

0

207

208

210

212

213

213

213

210

202

202

...

225

232

236

236

235

234

231

229

230

0

1

237

164

146

185

200

198

198

198

198

196

...

244

244

244

244

244

244

244

244

244

1

2

208

226

224

216

232

242

244

241

239

235

...

140

149

162

142

83

84

113

120

106

2

3

134

155

154

154

164

166

165

166

163

163

...

165

163

164

164

160

163

171

165

162

1

4

61

62

79

88

106

114

109

97

140

155

...

209

201

201

203

208

210

200

184

176

1

5 rows × 785 columns

dataset['label']

0 0  
1 1  
2 2  
3 1  
4 1  
5 0  
6 0  
7 2  
8 1  
9 1  
10 1  
11 2  
12 2  
13 1  
14 0  
15 1  
16 0  
17 1  
18 0  
Name: label, dtype: int64

dataset.shape

(19, 785)

x = dataset.drop(columns = 'label')  
x.shape

(19, 784)

X = dataset.drop(columns = 'label')  
y = dataset['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

X\_train = X\_train.values.reshape(-1, 28, 28, 1)  
X\_test = X\_test.values.reshape(-1, 28, 28, 1)

y\_train = tf.one\_hot(y\_train, 3)  
y\_test = tf.one\_hot(y\_test, 3)

print(X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape)

(13, 28, 28, 1) (13, 3) (6, 28, 28, 1) (6, 3)

model = Sequential()

model.add(Conv2D(filters = 32, kernel\_size = (5,5),padding = 'Same', activation ='relu', input\_shape = (28,28,1)))  
model.add(Conv2D(filters = 32, kernel\_size = (5,5),padding = 'Same', activation ='relu'))  
model.add(MaxPool2D(pool\_size=(2,2)))  
model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel\_size = (3,3),padding = 'Same', activation ='relu'))  
model.add(Conv2D(filters = 64, kernel\_size = (3,3),padding = 'Same', activation ='relu'))  
model.add(MaxPool2D(pool\_size=(2,2), strides=(2,2)))  
model.add(Dropout(0.25))

model.add(Flatten())  
model.add(Dense(128, activation = "relu"))  
model.add(Dropout(0.5))  
model.add(Dense(3, activation = "softmax"))

optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

model.compile(optimizer = optimizer , loss = "categorical\_crossentropy", metrics=["accuracy"])

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc', patience=3, verbose=1, factor=0.5, min\_lr=0.00001)

print(X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)

(13, 28, 28, 1) (6, 28, 28, 1) (13, 3) (6, 3)

history = model.fit(X\_train, y\_train, batch\_size = 2, epochs = 20, validation\_data = (X\_test, y\_test), verbose = 2)

Epoch 1/20  
7/7 - 1s - loss: 31.0880 - accuracy: 0.2308 - val\_loss: 2.3177 - val\_accuracy: 0.1667  
Epoch 2/20  
7/7 - 0s - loss: 2.5851 - accuracy: 0.4615 - val\_loss: 1.9106 - val\_accuracy: 0.3333  
Epoch 3/20  
7/7 - 0s - loss: 1.0754 - accuracy: 0.4615 - val\_loss: 1.0973 - val\_accuracy: 0.3333  
Epoch 4/20  
7/7 - 0s - loss: 2.5568 - accuracy: 0.5385 - val\_loss: 1.5742 - val\_accuracy: 0.3333  
Epoch 5/20  
7/7 - 0s - loss: 1.0356 - accuracy: 0.4615 - val\_loss: 1.4359 - val\_accuracy: 0.3333  
Epoch 6/20  
7/7 - 0s - loss: 1.1778 - accuracy: 0.4615 - val\_loss: 1.2673 - val\_accuracy: 0.3333  
Epoch 7/20  
7/7 - 0s - loss: 0.9661 - accuracy: 0.3846 - val\_loss: 1.5159 - val\_accuracy: 0.3333  
Epoch 8/20  
7/7 - 0s - loss: 1.0742 - accuracy: 0.5385 - val\_loss: 1.4009 - val\_accuracy: 0.3333  
Epoch 9/20  
7/7 - 0s - loss: 1.0593 - accuracy: 0.4615 - val\_loss: 1.1706 - val\_accuracy: 0.3333  
Epoch 10/20  
7/7 - 0s - loss: 1.1214 - accuracy: 0.4615 - val\_loss: 2.2162 - val\_accuracy: 0.3333  
Epoch 11/20  
7/7 - 0s - loss: 1.0830 - accuracy: 0.6154 - val\_loss: 2.2801 - val\_accuracy: 0.3333  
Epoch 12/20  
7/7 - 0s - loss: 1.6218 - accuracy: 0.4615 - val\_loss: 1.8752 - val\_accuracy: 0.3333  
Epoch 13/20  
7/7 - 0s - loss: 1.0345 - accuracy: 0.5385 - val\_loss: 1.2430 - val\_accuracy: 0.3333  
Epoch 14/20  
7/7 - 0s - loss: 0.9274 - accuracy: 0.6154 - val\_loss: 1.1948 - val\_accuracy: 0.3333  
Epoch 15/20  
7/7 - 0s - loss: 0.7964 - accuracy: 0.6154 - val\_loss: 1.9358 - val\_accuracy: 0.3333  
Epoch 16/20  
7/7 - 0s - loss: 0.5593 - accuracy: 0.6923 - val\_loss: 1.5384 - val\_accuracy: 0.3333  
Epoch 17/20  
7/7 - 0s - loss: 1.0909 - accuracy: 0.6154 - val\_loss: 2.6862 - val\_accuracy: 0.5000  
Epoch 18/20  
7/7 - 0s - loss: 1.9196 - accuracy: 0.4615 - val\_loss: 1.2563 - val\_accuracy: 0.1667  
Epoch 19/20  
7/7 - 0s - loss: 0.4025 - accuracy: 0.9231 - val\_loss: 1.5625 - val\_accuracy: 0.1667  
Epoch 20/20  
7/7 - 0s - loss: 0.4491 - accuracy: 0.8462 - val\_loss: 2.6482 - val\_accuracy: 0.1667

datagen = ImageDataGenerator(  
 featurewise\_center=False, # set input mean to 0 over the dataset  
 samplewise\_center=False, # set each sample mean to 0  
 featurewise\_std\_normalization=False, # divide inputs by std of the dataset  
 samplewise\_std\_normalization=False, # divide each input by its std  
 zca\_whitening=False, # apply ZCA whitening  
 rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)  
 zoom\_range = 0.1, # Randomly zoom image   
 width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)  
 height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)  
 horizontal\_flip=False, # randomly flip images  
 vertical\_flip=False) # randomly flip images

datagen.fit(X\_train)

print(history.history)

{'loss': [31.087982177734375, 2.585083484649658, 1.0753746032714844, 2.556833028793335, 1.0356119871139526, 1.177750825881958, 0.9661179780960083, 1.0741758346557617, 1.0592682361602783, 1.1213538646697998, 1.0829664468765259, 1.6217753887176514, 1.0344955921173096, 0.9274314641952515, 0.7963706254959106, 0.5592663884162903, 1.0908727645874023, 1.919561505317688, 0.4025065302848816, 0.4491156339645386], 'accuracy': [0.23076923191547394, 0.4615384638309479, 0.4615384638309479, 0.5384615659713745, 0.4615384638309479, 0.4615384638309479, 0.38461539149284363, 0.5384615659713745, 0.4615384638309479, 0.4615384638309479, 0.6153846383094788, 0.4615384638309479, 0.5384615659713745, 0.6153846383094788, 0.6153846383094788, 0.692307710647583, 0.6153846383094788, 0.4615384638309479, 0.9230769276618958, 0.8461538553237915], 'val\_loss': [2.317700147628784, 1.9105762243270874, 1.097281575202942, 1.5742197036743164, 1.4358612298965454, 1.2672605514526367, 1.5158854722976685, 1.4009289741516113, 1.170578956604004, 2.216231346130371, 2.2801029682159424, 1.8752355575561523, 1.2429604530334473, 1.1948405504226685, 1.9358230829238892, 1.5383585691452026, 2.6862475872039795, 1.2562830448150635, 1.5624996423721313, 2.6482338905334473], 'val\_accuracy': [0.1666666716337204, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.3333333432674408, 0.5, 0.1666666716337204, 0.1666666716337204, 0.1666666716337204]}

fig, ax = plt.subplots(2,1)  
ax[0].plot(history.history['loss'], color='b', label="Training loss")  
ax[0].plot(history.history['val\_loss'], color='r', label="validation loss",axes =ax[0])  
legend = ax[0].legend(loc='best', shadow=True)  
  
ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")  
ax[1].plot(history.history['val\_accuracy'], color='r',label="Validation accuracy")  
legend = ax[1].legend(loc='best', shadow=True)

png

png

def plot\_confusion\_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):  
  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, cm[i, j],  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.tight\_layout()  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')

Y\_pred = model.predict(X\_test)  
Y\_pred\_classes = np.argmax(Y\_pred, axis = 1)   
Y\_true = np.argmax(y\_test, axis = 1)   
confusion\_mtx = confusion\_matrix(Y\_true, Y\_pred\_classes)   
plot\_confusion\_matrix(confusion\_mtx, classes = range(10))

png

png

results = model.predict(X\_test)  
results = np.argmax(results, axis = 1)  
results = pd.Series(results, name="Label")