

FacialEmotionRecognition

June 5, 2022

```
[1]: # Basics
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.core.common import random_state

# Sklearn
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Tensorflow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow.keras.layers as layers
from tensorflow.keras.utils import to_categorical

# Graphing Style
%matplotlib inline
```

```
[2]: from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

1 Data Cleaning and Exploratory Data Analysis

```
[3]: face = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/face.csv")
```

```
[4]: face.shape
```

```
[4]: (28709, 2)
```

```
[5]: face["pixels"] = face.pixels.apply(lambda x: np.array(tuple(map(int, x.  
↳split()))))
```

```
[6]: face.head()
```

```
[6]:      emotion                                pixels  
0         0  [70, 80, 82, 72, 58, 58, 60, 63, 54, 58, 60, 4...  
1         0  [151, 150, 147, 155, 148, 133, 111, 140, 170, ...  
2         2  [231, 212, 156, 164, 174, 138, 161, 173, 182, ...  
3         4  [24, 32, 36, 30, 32, 23, 19, 20, 30, 41, 21, 2...  
4         6  [4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 15, 23...
```

```
[7]: emo_di = {0: "Angry", 1: "Disgust", 2: "Fear", 3: "Happy", 4: "Sad", 5: "  
↳"Surprise", 6: "Neutral"}
```

```
[8]: plt.figure(figsize = (20,20))  
start_index = 0  
for i in range(7):  
    plt.subplot(1,7,i+1)  
    plt.grid(False)  
    plt.xticks([])  
    plt.yticks([])  
    plt.imshow(face[face.emotion == i].pixels.iloc[20].reshape(48,48),  
↳cmap="gray")  
    plt.xlabel("{} = {}".format(i, emo_di[i]))
```



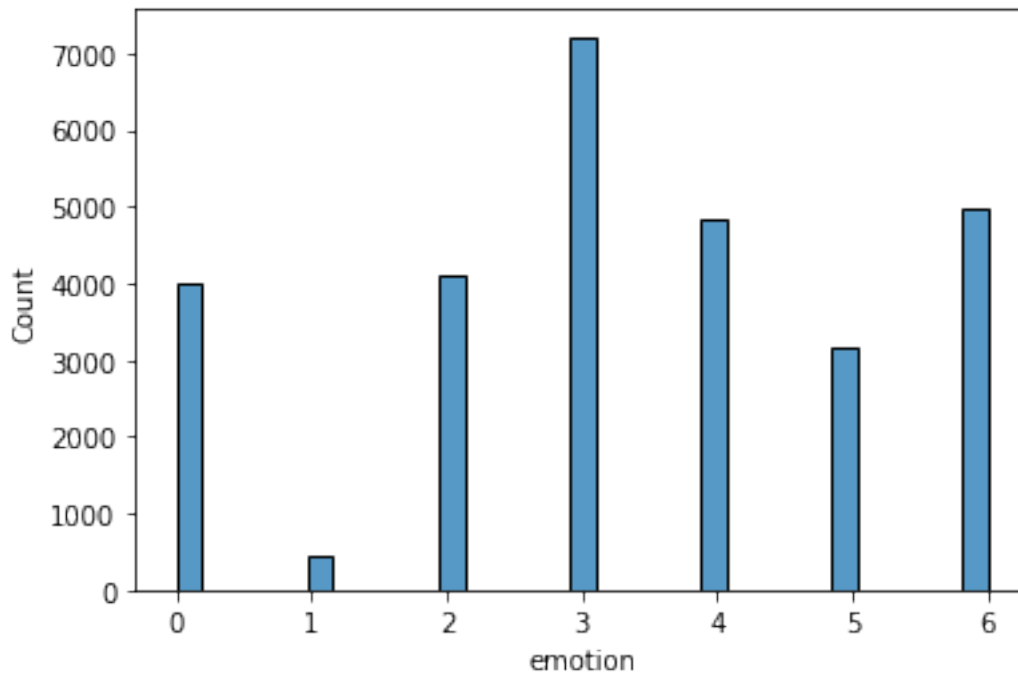
```
[9]: face.emotion.value_counts()
```

```
[9]: 3    7215  
     6    4965  
     4    4830
```

```
2    4097
0    3995
5    3171
1     436
Name: emotion, dtype: int64
```

```
[10]: sns.histplot(face.emotion)
```

```
[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f43872e6ed0>
```



```
[11]: train, test = train_test_split(face, train_size=0.6, random_state=1)
      val, test = train_test_split(test, test_size=0.5, random_state=1)

      # Oversample and subsample
      faces_balanced = None
      for i in range(7):
          if faces_balanced is None:
              faces_balanced = face[face.emotion == i].sample(n=3000, replace=True,
↳random_state=100)
          else:
              faces_balanced = pd.concat([faces_balanced, face[face.emotion == i].
↳sample(n=3000, replace=True, random_state=100)], sort = False)
      faces_balanced = faces_balanced.sample(frac=1, random_state=49)
```

```

train_x = np.concatenate(np.asarray(faces_balanced["pixels"])).reshape(-1, 48 * 48)
val_x = np.concatenate(np.asarray(val["pixels"])).reshape(-1, 48 * 48)
test_x = np.concatenate(np.asarray(test["pixels"])).reshape(-1, 48 * 48)
train_y = faces_balanced.emotion
val_y = val.emotion
test_y = test.emotion

x_mean = np.mean(train_x)
x_std = np.std(train_x) + 1e-10
train_x = (train_x - x_mean) / x_std
val_x = (val_x - x_mean) / x_std
test_x = (test_x - x_mean) / x_std

```

```
[12]: train_y.value_counts()
```

```

[12]: 4    3000
      5    3000
      2    3000
      0    3000
      1    3000
      3    3000
      6    3000
      Name: emotion, dtype: int64

```

```
[13]: val_y.value_counts()
```

```

[13]: 3    1420
      6    1035
      4     963
      0     811
      2     796
      5     625
      1      92
      Name: emotion, dtype: int64

```

```
[14]: test_y.value_counts()
```

```

[14]: 3    1494
      6     976
      4     946
      2     833
      0     792
      5     627
      1      74
      Name: emotion, dtype: int64

```

2 Modeling

2.1 Naive Bayes

```
[15]: nb = GaussianNB()  
      nb.fit(train_x, train_y)
```

```
[15]: GaussianNB()
```

```
[16]: nb.score(train_x, train_y)
```

```
[16]: 0.23157142857142857
```

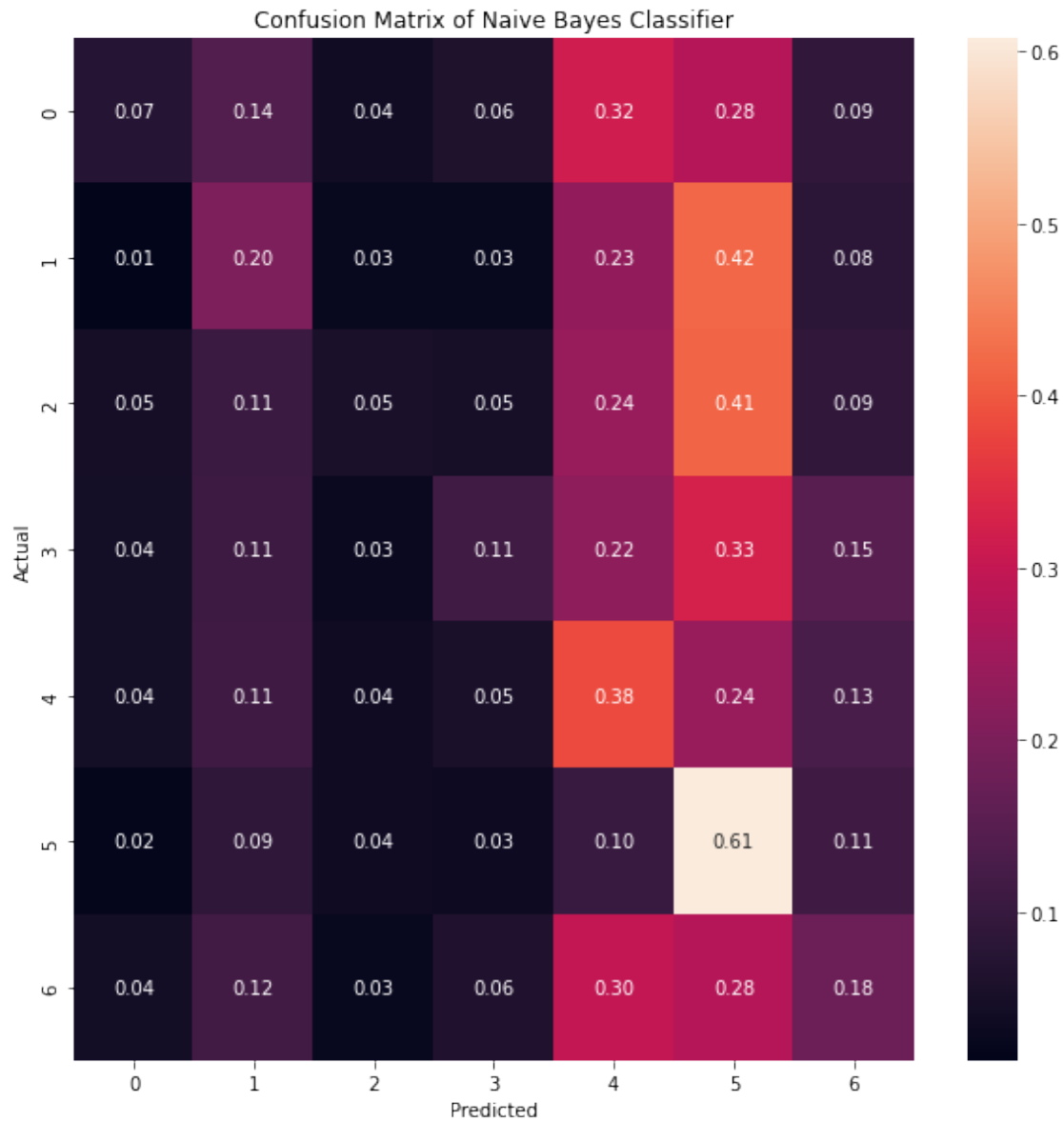
```
[17]: nb.score(val_x, val_y)
```

```
[17]: 0.20358760013932428
```

```
[18]: nb.score(test_x, test_y)
```

```
[18]: 0.20968303726924417
```

```
[19]: cm = confusion_matrix(test_y, nb.predict(test_x))  
      cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
      fig, ax = plt.subplots(figsize=(10,10))  
      sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=nb.classes_, yticklabels=nb.  
                  ↪classes_)  
      plt.title('Confusion Matrix of Naive Bayes Classifier')  
      plt.ylabel('Actual')  
      plt.xlabel('Predicted')  
      plt.show(block=False)
```



2.2 Logistic Regression

```
[20]: lr = LogisticRegression(penalty='l2', tol=0.1, solver='saga')
lr.fit(train_x, train_y)
```

```
[20]: LogisticRegression(solver='saga', tol=0.1)
```

```
[21]: lr.score(train_x, train_y)
```

```
[21]: 0.4888571428571429
```

```
[22]: lr.score(val_x, val_y)
```

```
[22]: 0.36363636363636365
```

```
[23]: lr.score(test_x, test_y)
```

```
[23]: 0.36450714036920934
```

```
[24]: cm = confusion_matrix(test_y, lr.predict(test_x))
      cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
      fig, ax = plt.subplots(figsize=(10,10))
      sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=nb.classes_, yticklabels=nb.
        ↳classes_)
      plt.title('Confusion Matrix of Logistic Regression')
      plt.ylabel('Actual')
      plt.xlabel('Predicted')
      plt.show(block=False)
```



2.3 K-Means + Logistic Regression

```
[25]: pl = Pipeline([
    ("kmeans", KMeans(n_clusters=35)),
    ("logisreg", LogisticRegression(penalty='none', tol=0.1, solver='saga'))
])
```

```
[26]: pl.fit(train_x, train_y)
```



```
[26]: Pipeline(steps=[('kmeans', KMeans(n_clusters=35)),  
                      ('logisreg',  
                       LogisticRegression(penalty='none', solver='saga', tol=0.1))])
```

```
[27]: pl.score(train_x, train_y)
```

```
[27]: 0.2277142857142857
```

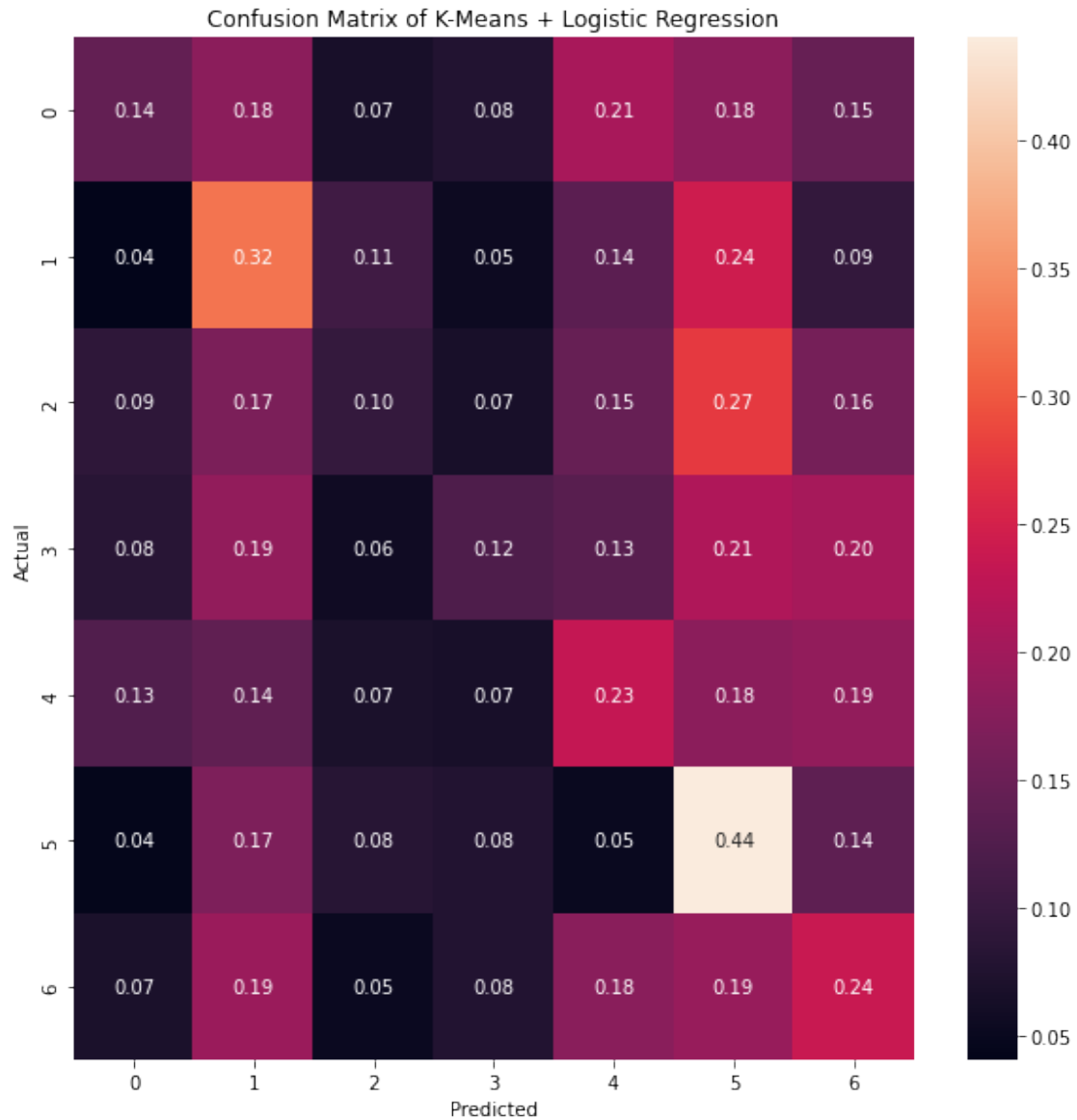
```
[28]: pl.score(val_x, val_y)
```

```
[28]: 0.19696969696969696
```

```
[29]: pl.score(test_x, test_y)
```

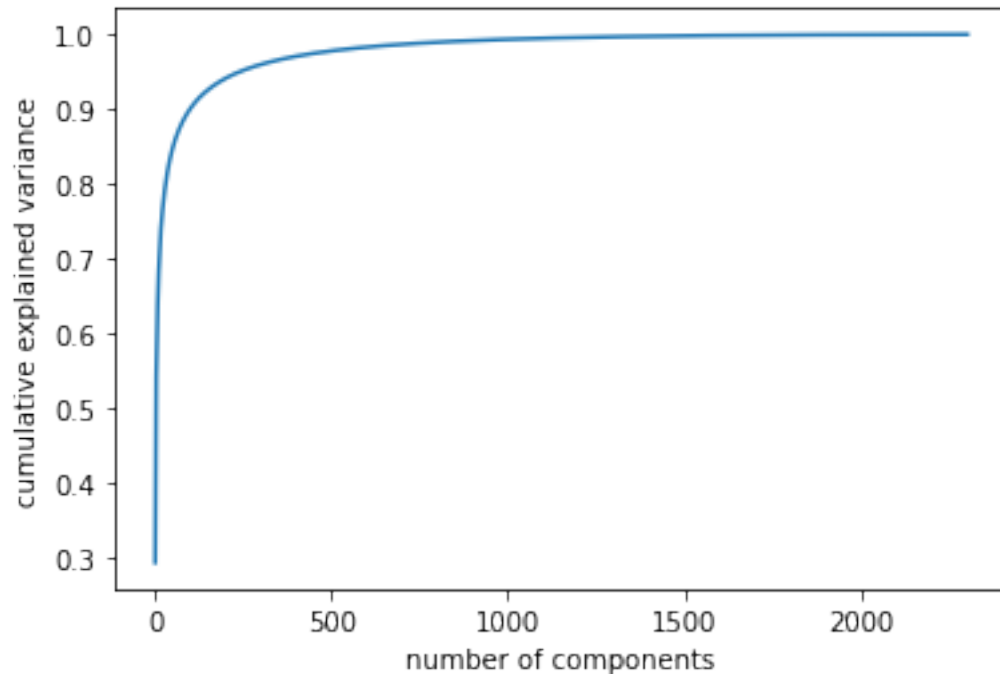
```
[29]: 0.1966213862765587
```

```
[30]: cm = confusion_matrix(test_y, pl.predict(test_x))  
      cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
      fig, ax = plt.subplots(figsize=(10,10))  
      sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=nb.classes_, yticklabels=nb.  
                  ↪classes_)  
      plt.title('Confusion Matrix of K-Means + Logistic Regression')  
      plt.ylabel('Actual')  
      plt.xlabel('Predicted')  
      plt.show(block=False)
```



2.4 KNN with PCA + CV

```
[31]: pca = PCA().fit(train_x)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



```
[32]: pl1 = Pipeline([
        ("PCA", PCA(n_components=100)),
        ("knn", KNeighborsClassifier())
    ])
    params = {"knn__n_neighbors": [1, 3, 5, 7, 12, 15, 20]}
    grids = GridSearchCV(pl1, params, cv=5)
    grids.fit(train_x, train_y)
```

```
[32]: GridSearchCV(cv=5,
                  estimator=Pipeline(steps=[('PCA', PCA(n_components=100)),
                                             ('knn', KNeighborsClassifier())]),
                  param_grid={'knn__n_neighbors': [1, 3, 5, 7, 12, 15, 20]})
```

```
[33]: grids.best_params_
```

```
[33]: {'knn__n_neighbors': 1}
```

```
[34]: grids.best_score_
```

```
[34]: 0.6629047619047619
```

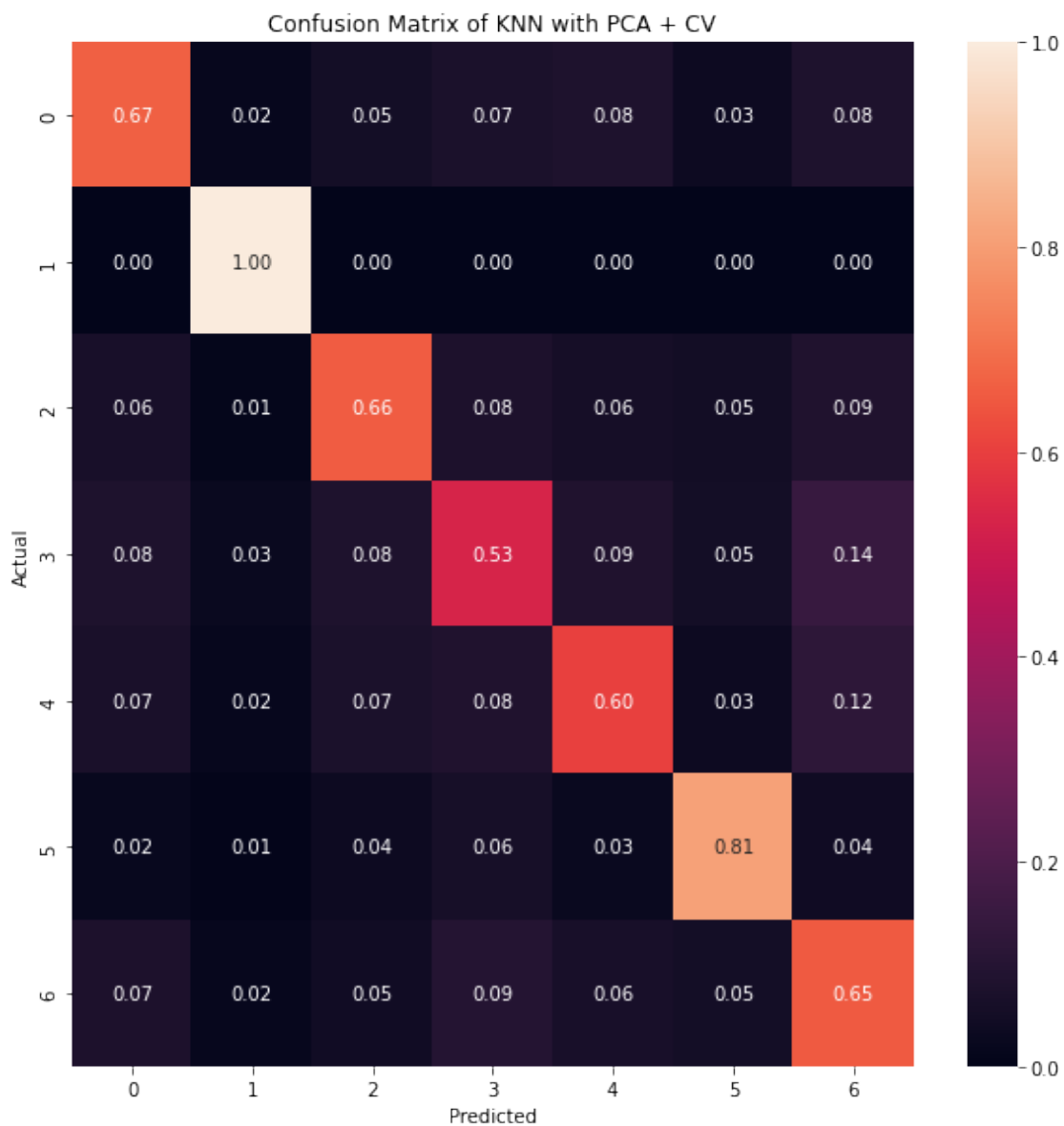
```
[35]: grids.best_estimator_.score(val_x, val_y)
```

```
[35]: 0.6461163357715082
```

```
[36]: grids.best_estimator_.score(test_x, test_y)
```

```
[36]: 0.6379310344827587
```

```
[37]: cm = confusion_matrix(test_y, grids.best_estimator_.predict(test_x))
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=nb.classes_, yticklabels=nb.
    ↪classes_)
plt.title('Confusion Matrix of KNN with PCA + CV')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```



2.5 Random Forest

```
[38]: rfc = RandomForestClassifier(max_depth=5)
      rfc.fit(train_x, train_y)
```

```
[38]: RandomForestClassifier(max_depth=5)
```

```
[39]: rfc.score(train_x, train_y)
```

```
[39]: 0.4005714285714286
```

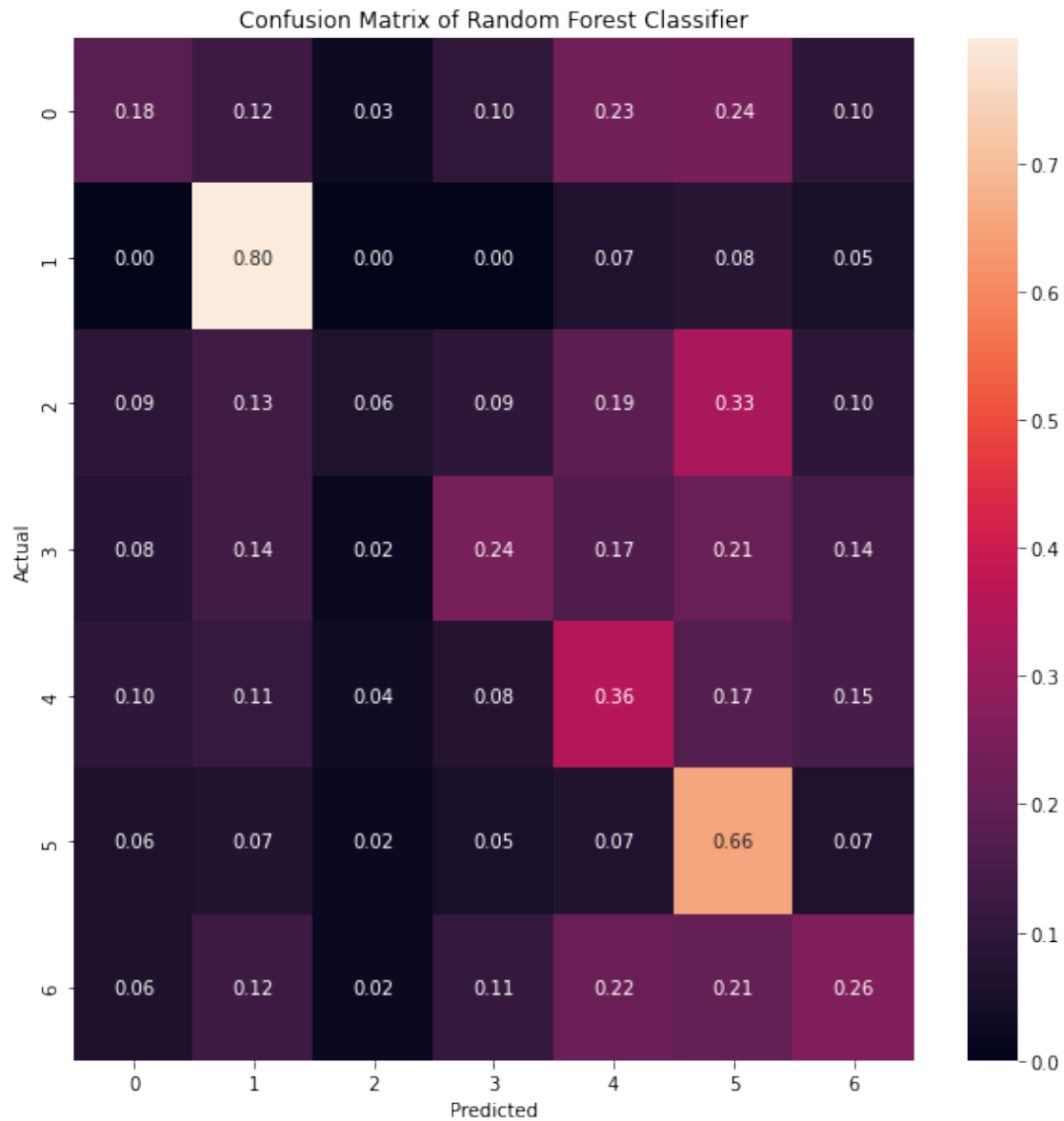
```
[40]: rfc.score(val_x, val_y)
```

```
[40]: 0.2871821664925113
```

```
[41]: rfc.score(test_x, test_y)
```

```
[41]: 0.28126088470916055
```

```
[42]: cm = confusion_matrix(test_y, rfc.predict(test_x))
      cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
      fig, ax = plt.subplots(figsize=(10,10))
      sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=nb.classes_, yticklabels=nb.
        ↳classes_)
      plt.title('Confusion Matrix of Random Forest Classifier')
      plt.ylabel('Actual')
      plt.xlabel('Predicted')
      plt.show(block=False)
```



2.6 Support Vector Machine with PCA

```
[43]: pl2 = Pipeline([
        ("PCA", PCA(n_components=50)),
        ("svm", SVC())
    ])
    pl2.fit(train_x, train_y)
```

```
[43]: Pipeline(steps=[('PCA', PCA(n_components=50)), ('svm', SVC())])
```

```
[44]: pl2.score(train_x, train_y)
```

[44]: 0.6312857142857143

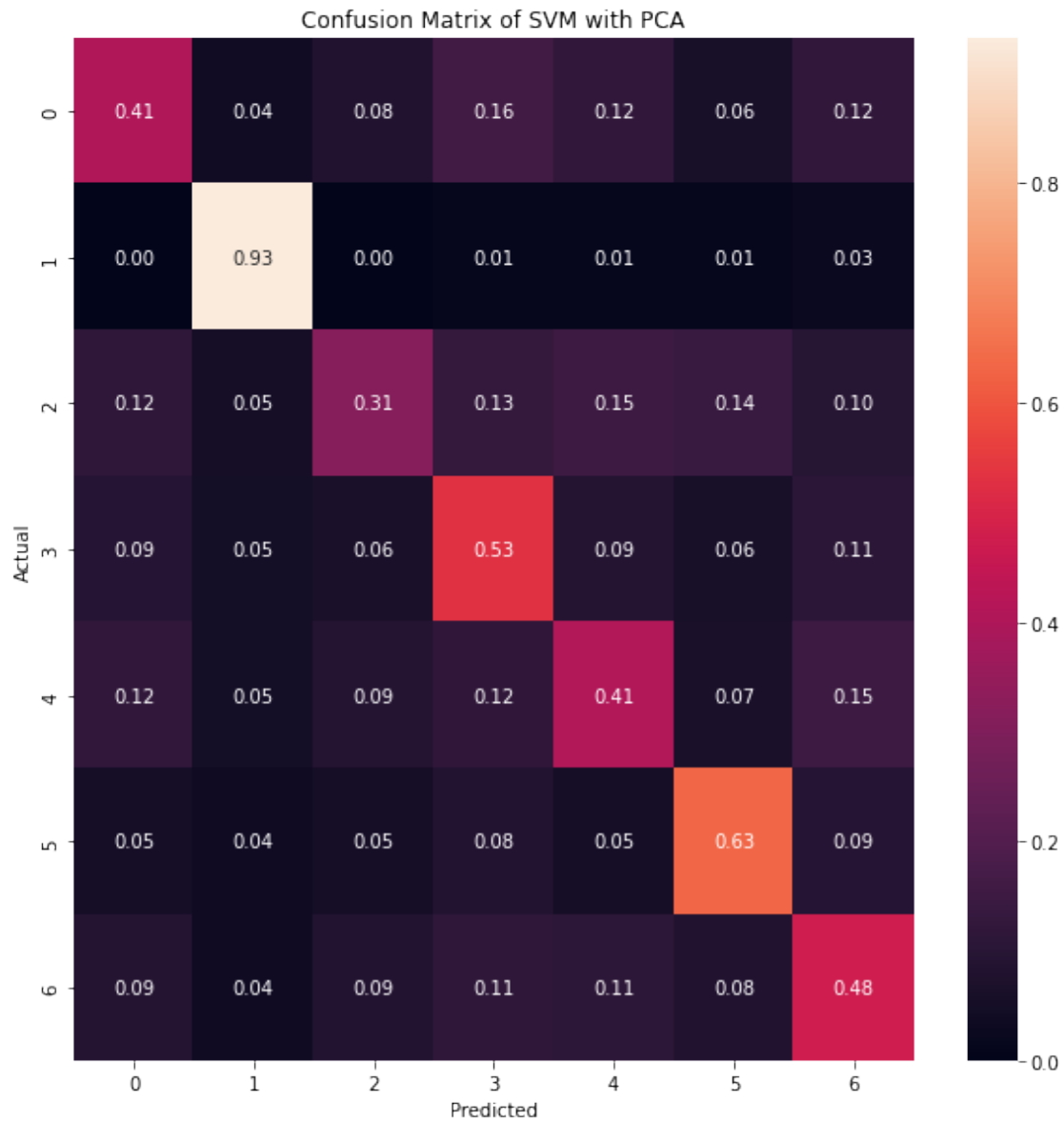
```
[45]: pl2.score(val_x, val_y)
```

[45]: 0.45942180424939044

```
[46]: pl2.score(test_x, test_y)
```

[46]: 0.46900034831069315

```
[47]: cm = confusion_matrix(test_y, pl2.predict(test_x))
      cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
      fig, ax = plt.subplots(figsize=(10,10))
      sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=nb.classes_, yticklabels=nb.
        ↪classes_)
      plt.title('Confusion Matrix of SVM with PCA')
      plt.ylabel('Actual')
      plt.xlabel('Predicted')
      plt.show(block=False)
```



2.7 Artificial Neural Network

```
[99]: train_y_cat = to_categorical(train_y)
      val_y_cat = to_categorical(val_y)
      test_y_cat = to_categorical(test_y)
```

```
[100]: model = Sequential()
        model.add(Dense(256, activation = 'relu', input_shape = (48 * 48,)))
        model.add(Dense(256, activation = 'relu'))
        model.add(Dense(256, activation = 'relu')),
        model.add(Dense(256, activation = 'relu')),
```



```

model.add(Dense(7, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = [
    ↪ 'accuracy'])
model.summary()

```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_22 (Dense)	(None, 256)	590080
dense_23 (Dense)	(None, 256)	65792
dense_24 (Dense)	(None, 256)	65792
dense_25 (Dense)	(None, 256)	65792
dense_26 (Dense)	(None, 7)	1799

Total params: 789,255
 Trainable params: 789,255
 Non-trainable params: 0

```

[101]: history = model.fit(train_x, train_y_cat, epochs = 20, validation_data = (
    ↪ val_x, val_y_cat))

```

```

Epoch 1/20
657/657 [=====] - 4s 5ms/step - loss: 1.7671 -
accuracy: 0.3019 - val_loss: 1.7173 - val_accuracy: 0.3022
Epoch 2/20
657/657 [=====] - 3s 5ms/step - loss: 1.5181 -
accuracy: 0.4083 - val_loss: 1.5960 - val_accuracy: 0.3661
Epoch 3/20
657/657 [=====] - 3s 5ms/step - loss: 1.3385 -
accuracy: 0.4810 - val_loss: 1.5412 - val_accuracy: 0.3931
Epoch 4/20
657/657 [=====] - 3s 5ms/step - loss: 1.2176 -
accuracy: 0.5320 - val_loss: 1.5267 - val_accuracy: 0.4072
Epoch 5/20
657/657 [=====] - 3s 4ms/step - loss: 1.1043 -
accuracy: 0.5774 - val_loss: 1.4307 - val_accuracy: 0.4702
Epoch 6/20
657/657 [=====] - 3s 5ms/step - loss: 0.9904 -
accuracy: 0.6308 - val_loss: 1.4433 - val_accuracy: 0.4805

```

```

Epoch 7/20
657/657 [=====] - 3s 5ms/step - loss: 0.8841 -
accuracy: 0.6672 - val_loss: 1.4089 - val_accuracy: 0.5158
Epoch 8/20
657/657 [=====] - 3s 5ms/step - loss: 0.7984 -
accuracy: 0.7047 - val_loss: 1.4509 - val_accuracy: 0.5270
Epoch 9/20
657/657 [=====] - 3s 5ms/step - loss: 0.7165 -
accuracy: 0.7427 - val_loss: 1.4845 - val_accuracy: 0.5420
Epoch 10/20
657/657 [=====] - 3s 5ms/step - loss: 0.6224 -
accuracy: 0.7739 - val_loss: 1.5038 - val_accuracy: 0.5594
Epoch 11/20
657/657 [=====] - 3s 5ms/step - loss: 0.5644 -
accuracy: 0.7979 - val_loss: 1.5244 - val_accuracy: 0.5610
Epoch 12/20
657/657 [=====] - 3s 5ms/step - loss: 0.5080 -
accuracy: 0.8174 - val_loss: 1.6178 - val_accuracy: 0.5845
Epoch 13/20
657/657 [=====] - 3s 5ms/step - loss: 0.4697 -
accuracy: 0.8347 - val_loss: 1.5725 - val_accuracy: 0.5939
Epoch 14/20
657/657 [=====] - 3s 5ms/step - loss: 0.4167 -
accuracy: 0.8520 - val_loss: 1.7744 - val_accuracy: 0.5951
Epoch 15/20
657/657 [=====] - 3s 5ms/step - loss: 0.3934 -
accuracy: 0.8652 - val_loss: 1.7112 - val_accuracy: 0.6021
Epoch 16/20
657/657 [=====] - 3s 5ms/step - loss: 0.3665 -
accuracy: 0.8714 - val_loss: 1.7708 - val_accuracy: 0.6125
Epoch 17/20
657/657 [=====] - 3s 5ms/step - loss: 0.3258 -
accuracy: 0.8890 - val_loss: 1.8759 - val_accuracy: 0.6217
Epoch 18/20
657/657 [=====] - 3s 5ms/step - loss: 0.3235 -
accuracy: 0.8909 - val_loss: 1.7949 - val_accuracy: 0.6172
Epoch 19/20
657/657 [=====] - 3s 5ms/step - loss: 0.3030 -
accuracy: 0.8960 - val_loss: 1.9177 - val_accuracy: 0.6193
Epoch 20/20
657/657 [=====] - 3s 5ms/step - loss: 0.2823 -
accuracy: 0.9033 - val_loss: 2.0023 - val_accuracy: 0.6290

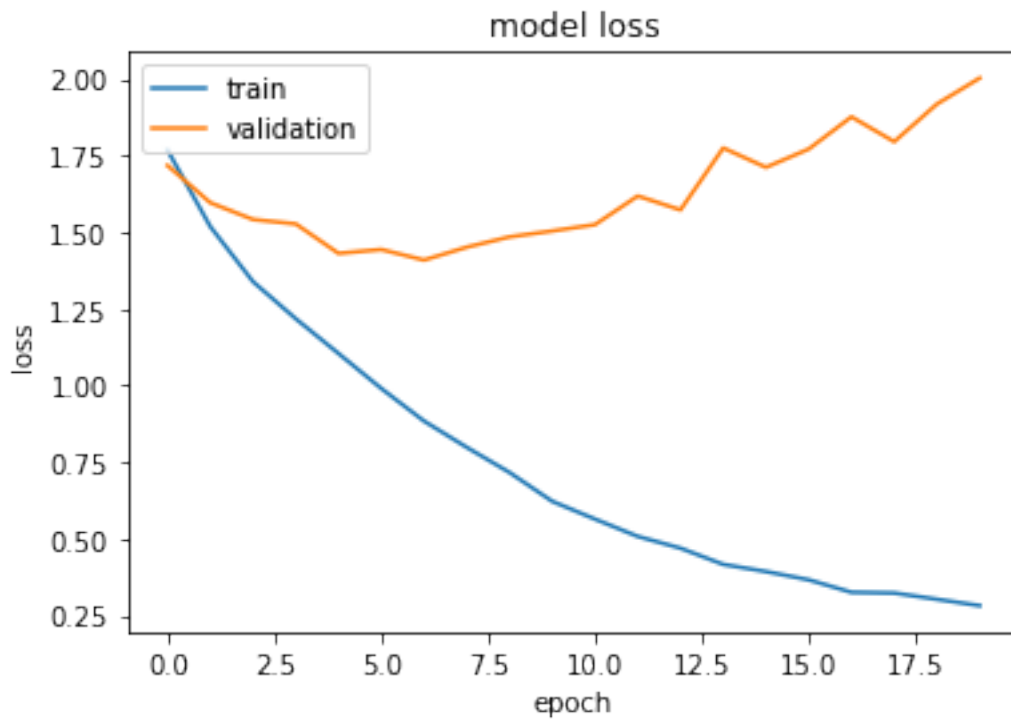
```

```

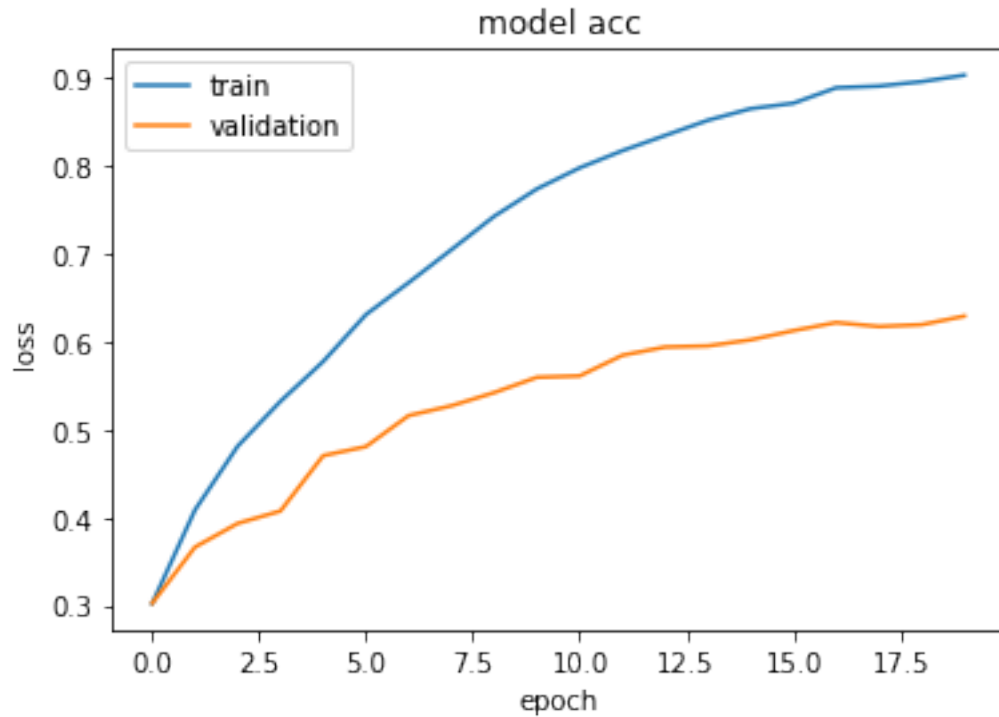
[102]: plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')

```

```
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



```
[103]: plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model acc')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

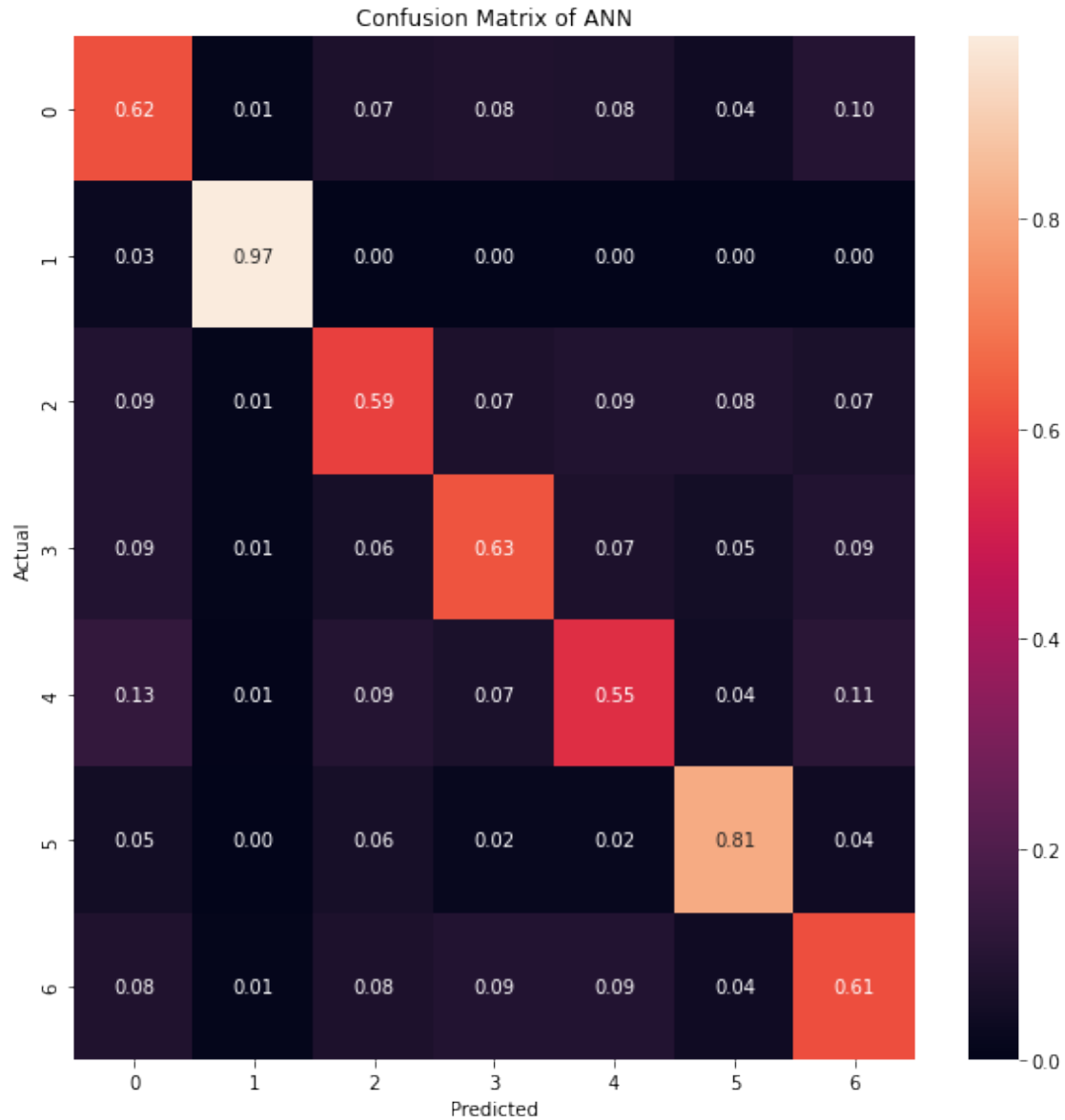


```
[104]: model.evaluate(test_x, test_y_cat)
```

```
180/180 [=====] - 0s 2ms/step - loss: 1.9723 -
accuracy: 0.6287
```

```
[104]: [1.972302794456482, 0.6287007927894592]
```

```
[105]: cm = confusion_matrix(test_y, np.apply_along_axis(np.argmax, 1, model.
    ↪predict(test_x)))
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=nb.classes_, yticklabels=nb.
    ↪classes_)
plt.title('Confusion Matrix of ANN')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```



2.8 Convolutional Neural Network

```
[86]: train_x_cnn = train_x.reshape(-1, 48, 48, 1)
      val_x_cnn = val_x.reshape(-1, 48, 48, 1)
      test_x_cnn = test_x.reshape(-1, 48, 48, 1)
```

```
[93]: model3 = Sequential([
      layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
      layers.Conv2D(16, 3, padding='same', activation='relu'),
      layers.MaxPooling2D(),
      layers.Conv2D(32, 3, padding='same', activation='relu'),
```

```

layers.MaxPooling2D(),
layers.Conv2D(64, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(7)
])

```

```

[94]: model3.compile(optimizer='adam',
                    loss=tf.keras.losses.
                        SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])

```

```

[95]: history3 = model3.fit(train_x_cnn, train_y, epochs = 20, validation_data =
    (val_x_cnn, val_y))

```

```

Epoch 1/20
657/657 [=====] - 16s 24ms/step - loss: 1.6371 -
accuracy: 0.3590 - val_loss: 1.4592 - val_accuracy: 0.4352
Epoch 2/20
657/657 [=====] - 16s 24ms/step - loss: 1.1933 -
accuracy: 0.5517 - val_loss: 1.2526 - val_accuracy: 0.5219
Epoch 3/20
657/657 [=====] - 16s 24ms/step - loss: 0.9327 -
accuracy: 0.6554 - val_loss: 1.1211 - val_accuracy: 0.5920
Epoch 4/20
657/657 [=====] - 16s 24ms/step - loss: 0.7309 -
accuracy: 0.7345 - val_loss: 1.1220 - val_accuracy: 0.6172
Epoch 5/20
657/657 [=====] - 16s 24ms/step - loss: 0.5627 -
accuracy: 0.8034 - val_loss: 1.0846 - val_accuracy: 0.6641
Epoch 6/20
657/657 [=====] - 16s 24ms/step - loss: 0.4200 -
accuracy: 0.8506 - val_loss: 1.2082 - val_accuracy: 0.6628
Epoch 7/20
657/657 [=====] - 16s 24ms/step - loss: 0.3095 -
accuracy: 0.8928 - val_loss: 1.2962 - val_accuracy: 0.6886
Epoch 8/20
657/657 [=====] - 16s 24ms/step - loss: 0.2275 -
accuracy: 0.9235 - val_loss: 1.3900 - val_accuracy: 0.7055
Epoch 9/20
657/657 [=====] - 16s 24ms/step - loss: 0.1819 -
accuracy: 0.9378 - val_loss: 1.5935 - val_accuracy: 0.6998
Epoch 10/20
657/657 [=====] - 16s 24ms/step - loss: 0.1518 -
accuracy: 0.9492 - val_loss: 1.6249 - val_accuracy: 0.7062
Epoch 11/20

```

```

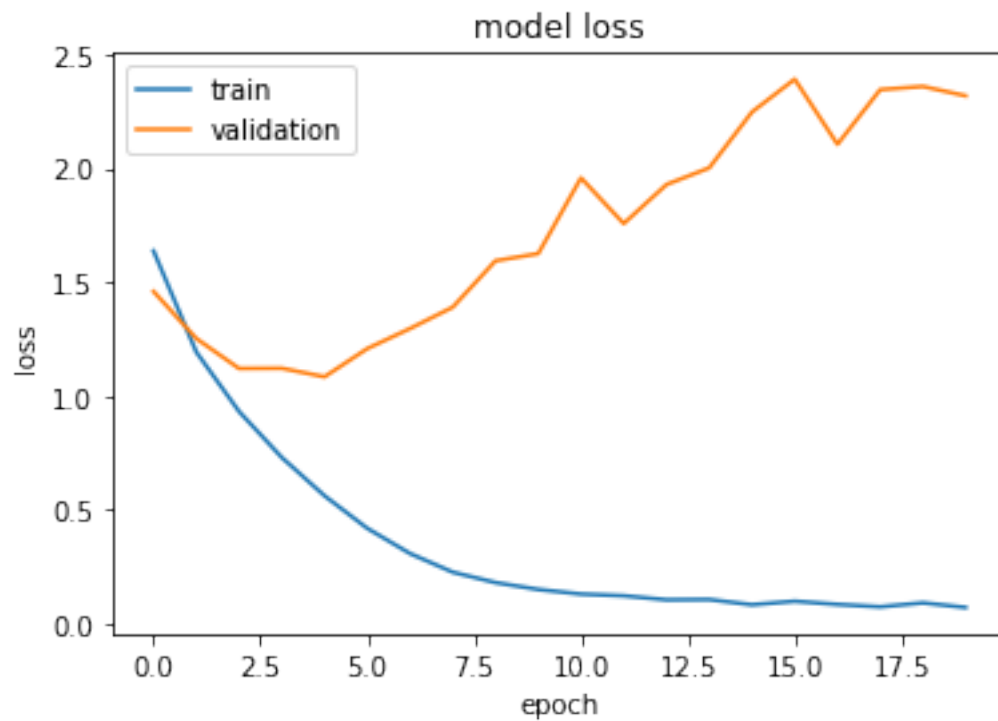
657/657 [=====] - 16s 24ms/step - loss: 0.1315 -
accuracy: 0.9568 - val_loss: 1.9570 - val_accuracy: 0.6846
Epoch 12/20
657/657 [=====] - 16s 24ms/step - loss: 0.1240 -
accuracy: 0.9598 - val_loss: 1.7568 - val_accuracy: 0.7156
Epoch 13/20
657/657 [=====] - 16s 24ms/step - loss: 0.1064 -
accuracy: 0.9652 - val_loss: 1.9280 - val_accuracy: 0.7160
Epoch 14/20
657/657 [=====] - 16s 24ms/step - loss: 0.1075 -
accuracy: 0.9651 - val_loss: 2.0010 - val_accuracy: 0.7147
Epoch 15/20
657/657 [=====] - 16s 24ms/step - loss: 0.0847 -
accuracy: 0.9733 - val_loss: 2.2462 - val_accuracy: 0.7095
Epoch 16/20
657/657 [=====] - 16s 24ms/step - loss: 0.1002 -
accuracy: 0.9679 - val_loss: 2.3902 - val_accuracy: 0.7046
Epoch 17/20
657/657 [=====] - 16s 24ms/step - loss: 0.0861 -
accuracy: 0.9738 - val_loss: 2.1045 - val_accuracy: 0.7173
Epoch 18/20
657/657 [=====] - 16s 24ms/step - loss: 0.0756 -
accuracy: 0.9760 - val_loss: 2.3444 - val_accuracy: 0.7161
Epoch 19/20
657/657 [=====] - 16s 24ms/step - loss: 0.0930 -
accuracy: 0.9715 - val_loss: 2.3581 - val_accuracy: 0.7229
Epoch 20/20
657/657 [=====] - 16s 24ms/step - loss: 0.0727 -
accuracy: 0.9768 - val_loss: 2.3171 - val_accuracy: 0.7276

```

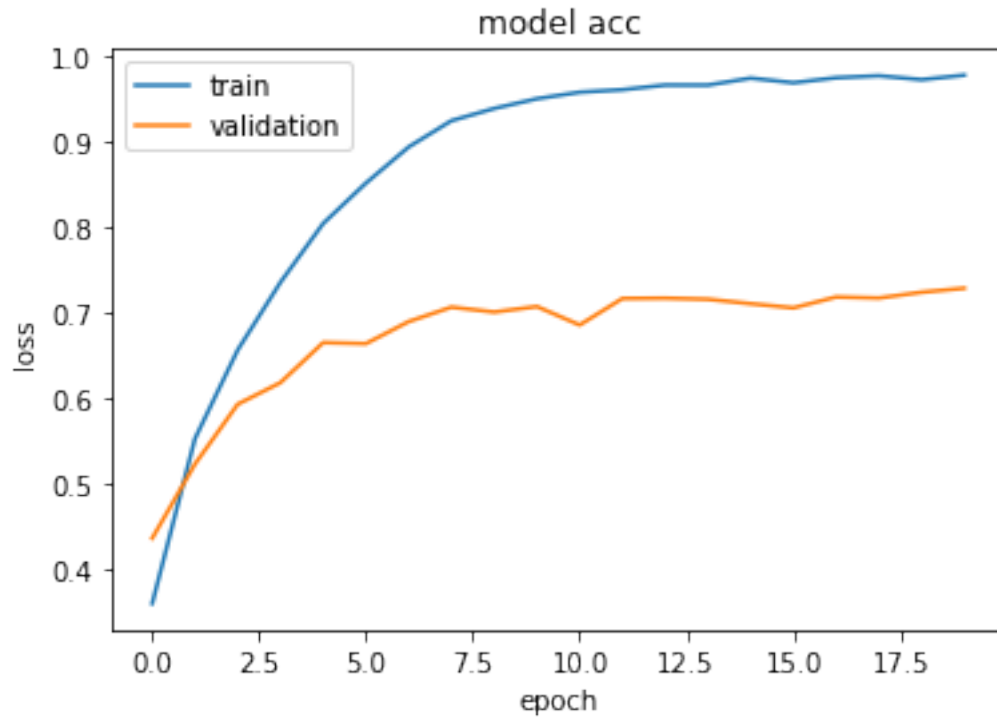
```

[106]: plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

```



```
[107]: plt.plot(history3.history['accuracy'])
plt.plot(history3.history['val_accuracy'])
plt.title('model acc')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

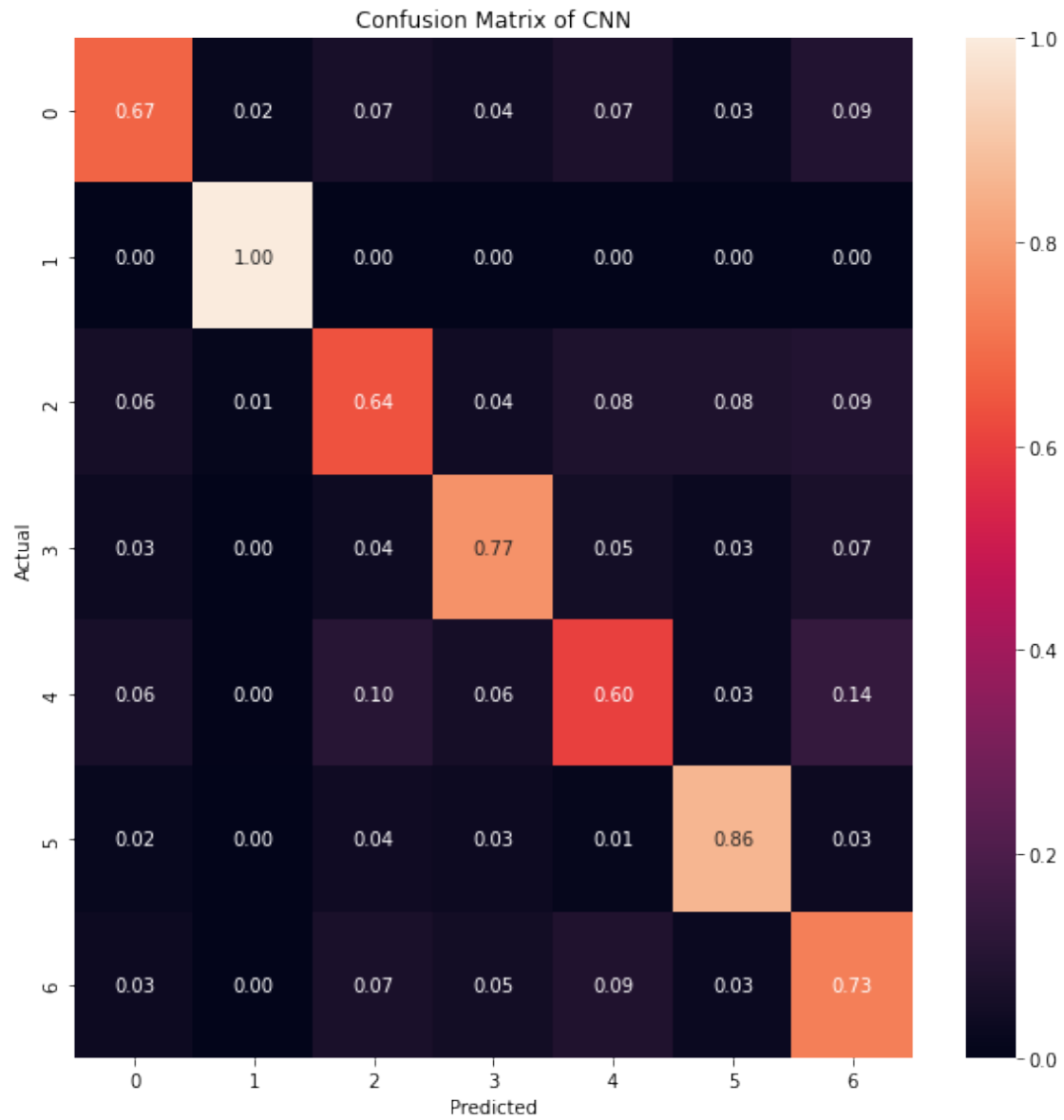



```
[96]: model3.evaluate(test_x_cnn, test_y)
```

```
180/180 [=====] - 1s 7ms/step - loss: 2.3736 -
accuracy: 0.7177
```

```
[96]: [2.3736231327056885, 0.7176941633224487]
```

```
[97]: cm = confusion_matrix(test_y, np.apply_along_axis(np.argmax, 1, model3.
    ↪predict(test_x_cnn)))
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=nb.classes_, yticklabels=nb.
    ↪classes_)
plt.title('Confusion Matrix of CNN')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```



```
[108]: model.save('saved_model/ann.h5')
model3.save('saved_model/cnn.h5')
```