FacialEmotionRecognition

June 2, 2022

[1]:
```python
# Basics
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# 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
```

[4]:
```python
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

[5]:
```
face = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/face.csv")
```

[6]:
```
face.shape
```

[6]:
```
(28709, 2)
```

[7]:
```
face["pixels"] = face.pixels.apply(lambda x: np.array(tuple(map(int, x.
↪split()))))
```

[8]:
```
face.head()
```

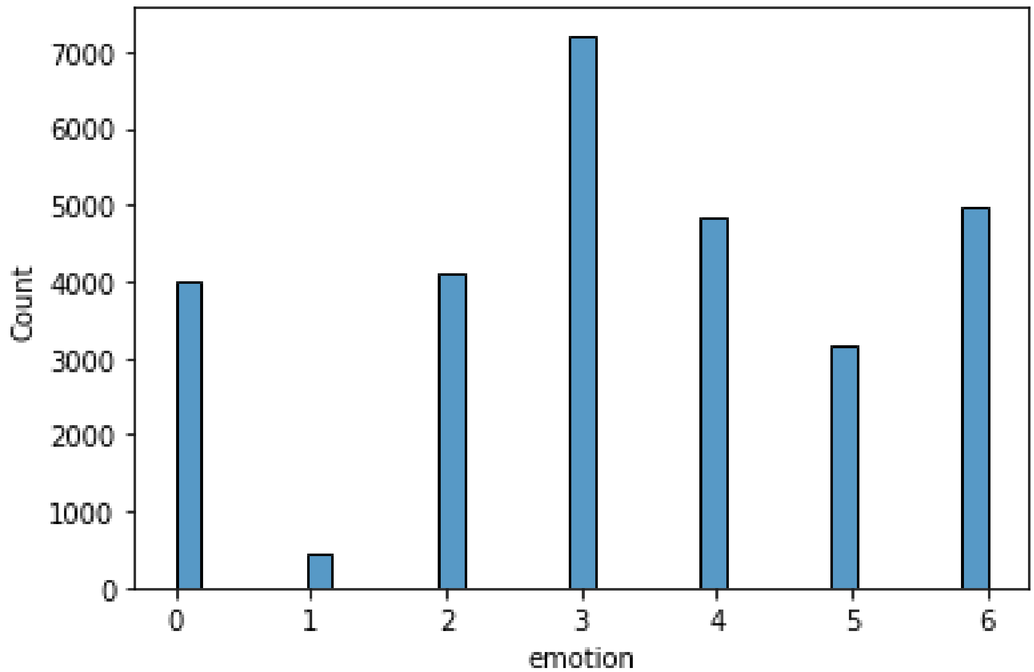
[8]:
```
   emotion                                             pixels
0        0  [70, 80, 82, 72, 58, 58, 60, 63, 54, 58, 60, 4…
1        0  [151, 150, 147, 155, 148, 133, 111, 140, 170, …
2        2  [231, 212, 156, 164, 174, 138, 161, 173, 182, …
3        4  [24, 32, 36, 30, 32, 23, 19, 20, 30, 41, 21, 2…
4        6  [4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 15, 23…
```

[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 0x7f7e8769b4d0>
```



[11]:
```python
emo_di = {0: "Angry", 1:"Disgust", 2: "Fear", 3: "Happy", 4: "Sad", 5: "Surprise", 6: "Neutral"}
```

[12]:
```python
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))
    plt.xlabel("{} = {}".format(i, emo_di[i]))
```

For each one, consider using Cross Validation and/or PCA

[13]:
```python
train_x, test_x, train_y, test_y = train_test_split(np.concatenate(np.asarray(face["pixels"])).reshape(-1, 48 * 48),
```

```
                                                    face.emotion,
                                                    train_size=0.7,
                                                    random_state=1)
val_x, test_x, val_y, test_y = train_test_split(test_x, test_y, test_size=0.5,␣
 ↪random_state=1)


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
```

[92]:
```
val_y.value_counts()
```

[92]:
```
3    1111
6     803
4     708
0     601
2     588
5     445
1      50
Name: emotion, dtype: int64
```

[91]:
```
test_y.value_counts()
```

[91]:
```
3    1094
6     735
4     716
2     628
0     567
5     502
1      65
Name: emotion, dtype: int64
```

2 Modeling

2.1 Naive Bayes

[14]:
```
nb = GaussianNB()
nb.fit(train_x, train_y)
```

[14]:
```
GaussianNB()
```

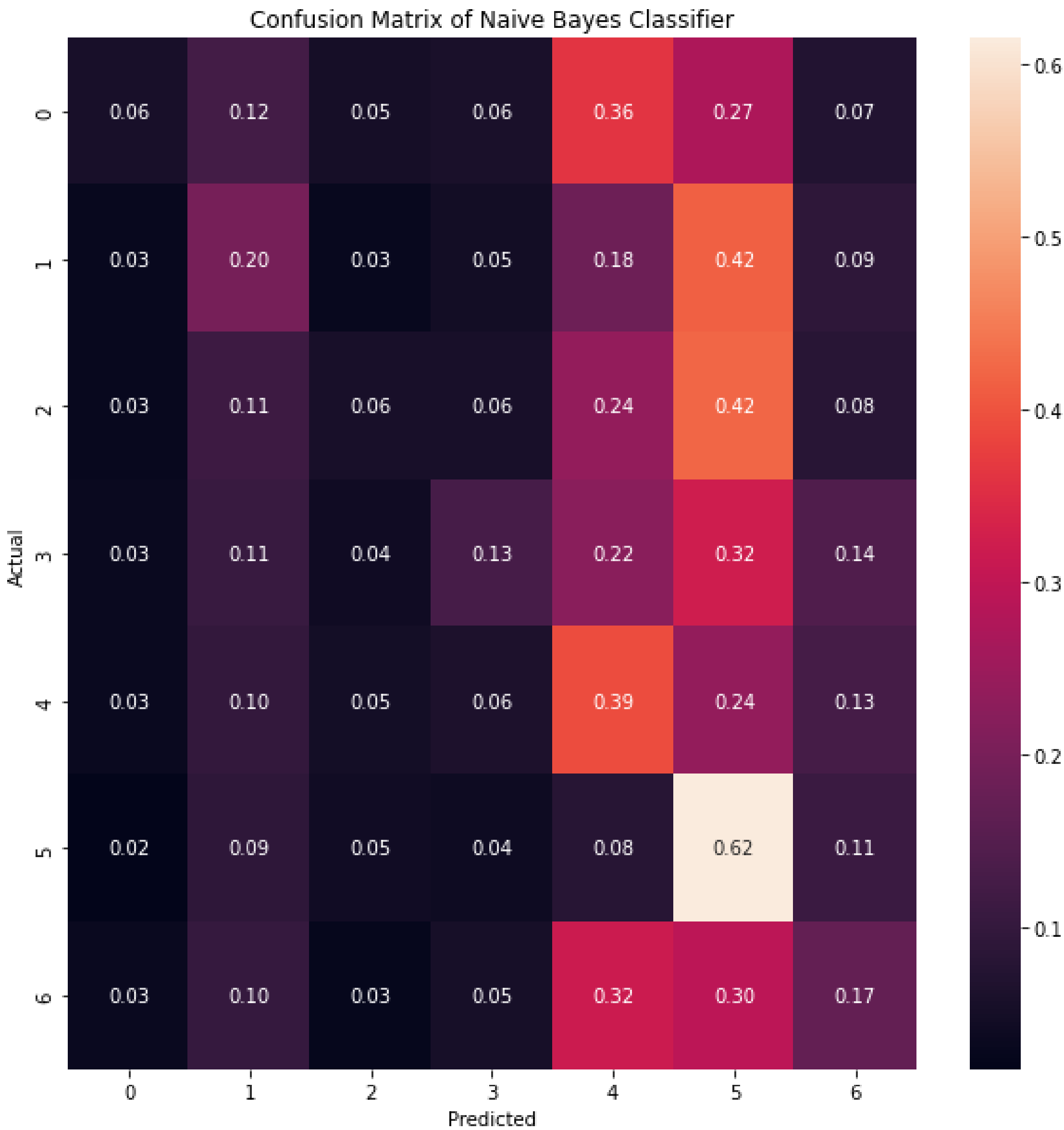
[15]:
```
nb.score(train_x, train_y)
```

[15]: 0.2174562101910828

[16]:
```python
nb.score(test_x, test_y)
```

[16]: 0.2173206408172742

[19]:
```python
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 K-Means + Logistic Regression

[20]:
```
pl = Pipeline([
    ("kmeans", KMeans(n_clusters=35)),
    ("logisreg", LogisticRegression(max_iter=10000))
])
```

[21]:
```
pl.fit(train_x, train_y)
```

[21]:
```
Pipeline(steps=[('kmeans', KMeans(n_clusters=35)),
                ('logisreg', LogisticRegression(max_iter=10000))])
```

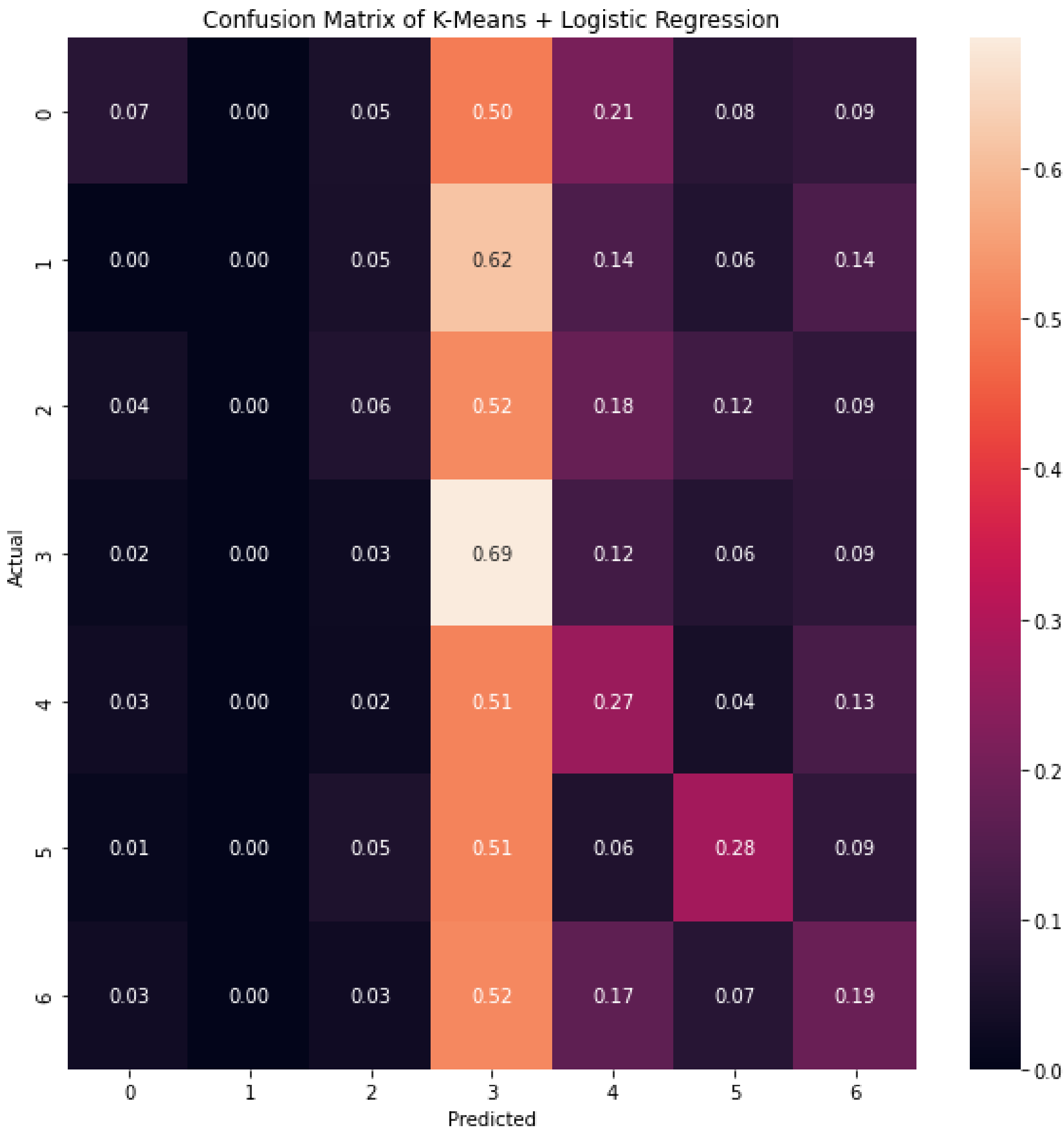
[22]:
```
pl.score(train_x, train_y)
```

[22]: 0.2910031847133758

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

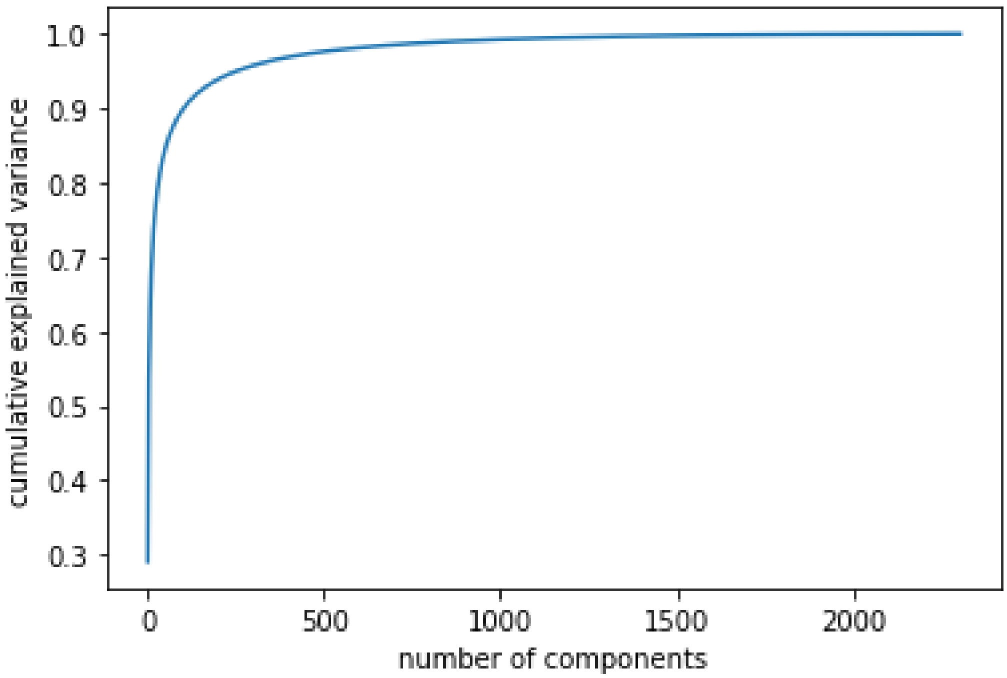
[23]: 0.3018342233573253

[24]:
```
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)
```



2.3 KNN with PCA + CV

[25]:
```
pca = PCA().fit(train_x)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



[26]:
```
pl = Pipeline([
    ("PCA", PCA(n_components=100)),
    ("knn", KNeighborsClassifier())
])
params = {"knn__n_neighbors": [1, 3, 5, 7, 12, 15, 20]}
grids = GridSearchCV(pl, params, cv=5)
grids.fit(train_x, train_y)
```

[26]:
```
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]})
```

[27]:
```
grids.best_params_
```

[27]:
```
{'knn__n_neighbors': 1}
```

[28]:
```
grids.best_score_
```

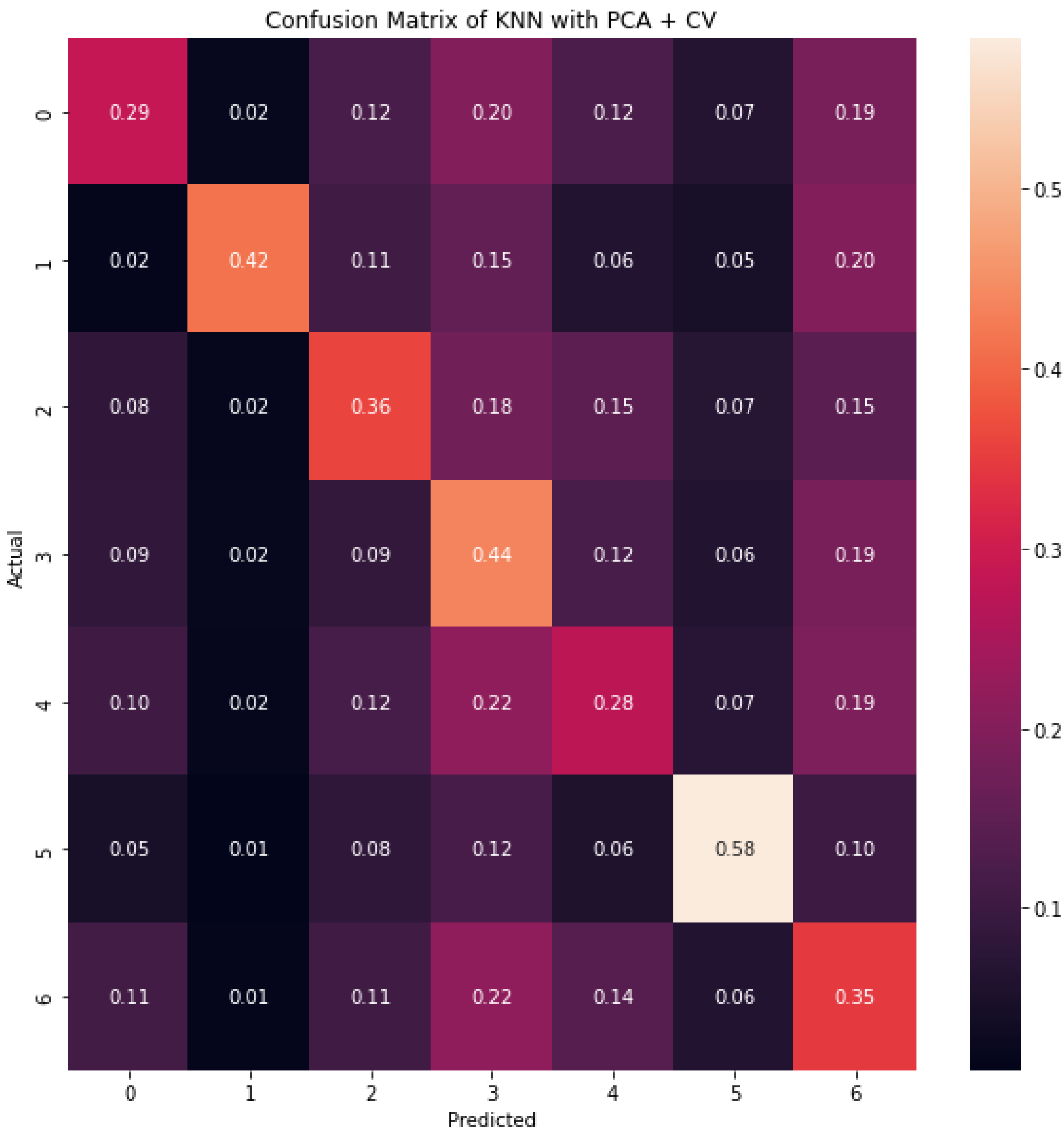
[28]:
```
0.3485272195875561
```

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

[29]:
```
0.3805433016020432
```

[30]:

```
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)
```



2.4 Random Forest

[31]:
```
rfc = RandomForestClassifier(max_depth=5)
rfc.fit(train_x, train_y)
```

[31]:
```
RandomForestClassifier(max_depth=5)
```

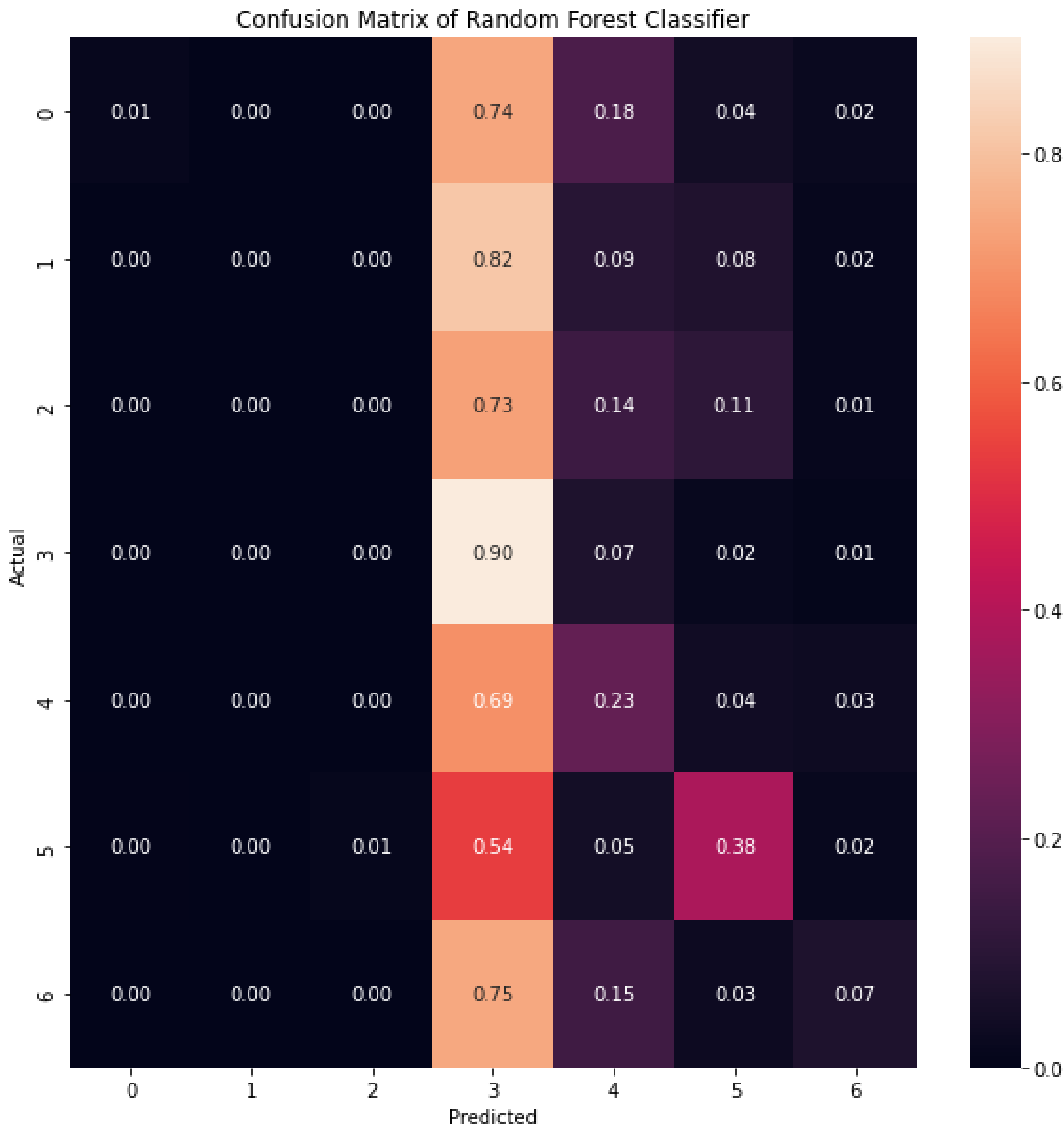
[32]:
```
rfc.score(train_x, train_y)
```

[32]:
```
0.3391222133757962
```

[33]:
```
rfc.score(test_x, test_y)
```

[33]:
```
0.3262131413977246
```

[84]:
```
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)
```



2.5 Support Vector Machine with PCA

[36]:
```
pl = Pipeline([
                ("PCA", PCA(n_components=50)),
                ("svm", SVC())
])
pl.fit(train_x, train_y)
```

[36]:
```
Pipeline(steps=[('PCA', PCA(n_components=50)), ('svm', SVC())])
```

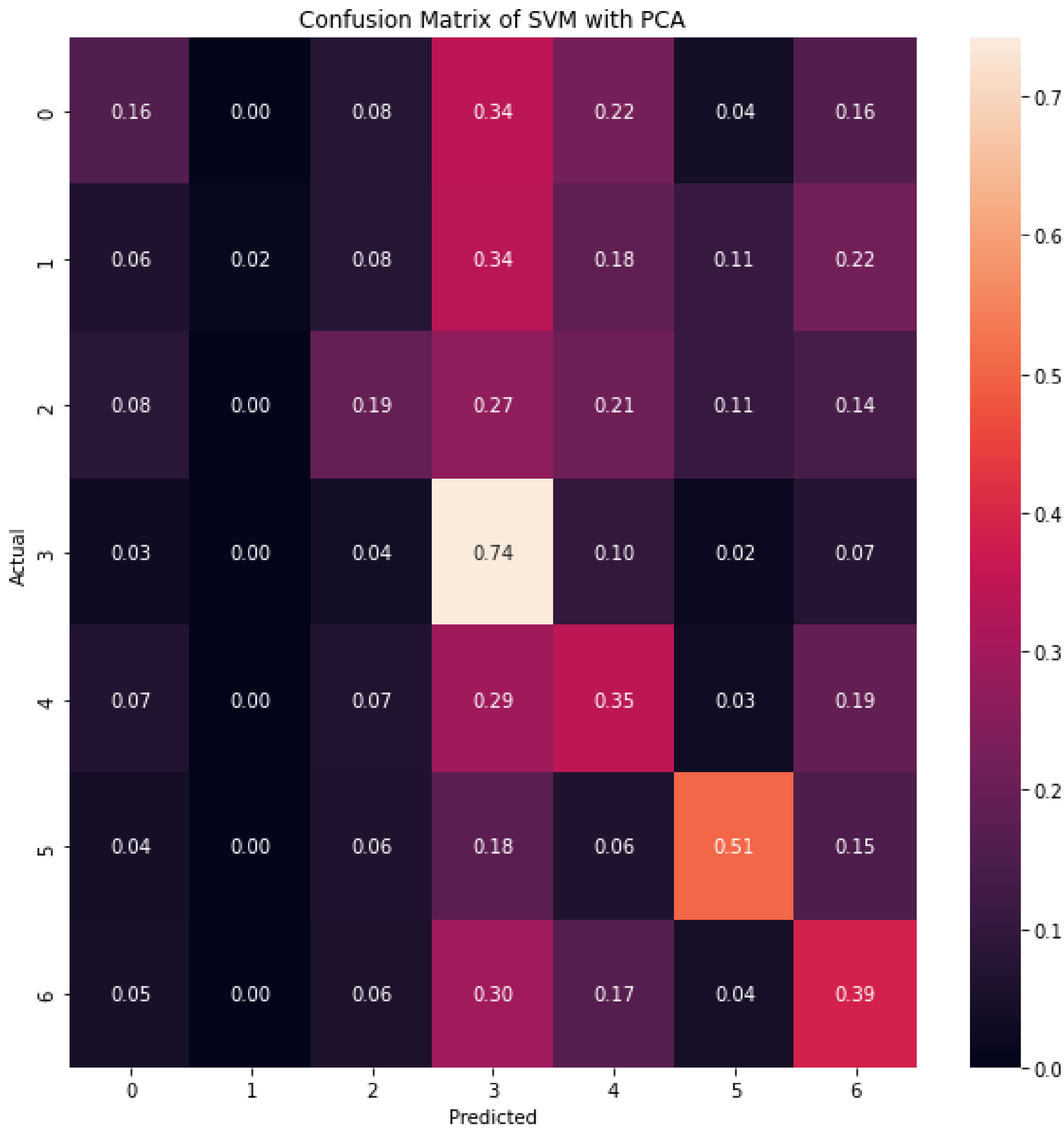
[37]:
```
pl.score(train_x, train_y)
```

[37]: 0.528015525477707

[38]:
```python
pl.score(test_x, test_y)
```

[38]: 0.4211748316693754

[39]:
```python
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 SVM with PCA')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```



2.6 Artificial Neural Network

[40]:
```
train_y_cat = to_categorical(train_y)
val_y_cat = to_categorical(val_y)
test_y_cat = to_categorical(test_y)
```

[41]:
```
model = Sequential()
model.add(Dense(256, activation = 'relu', input_shape = (48 * 48,)))
model.add(layers.Dropout(0.2))
model.add(Dense(256, activation = 'relu'))
model.add(layers.Dropout(0.5))
```

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               590080

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 7)                 1799

=================================================================
Total params: 657,671
Trainable params: 657,671
Non-trainable params: 0
_________________________________________________________________
```

[42]:
```
history = model.fit(train_x, train_y_cat, epochs = 20, validation_data =
 ↪(val_x, val_y_cat))
```

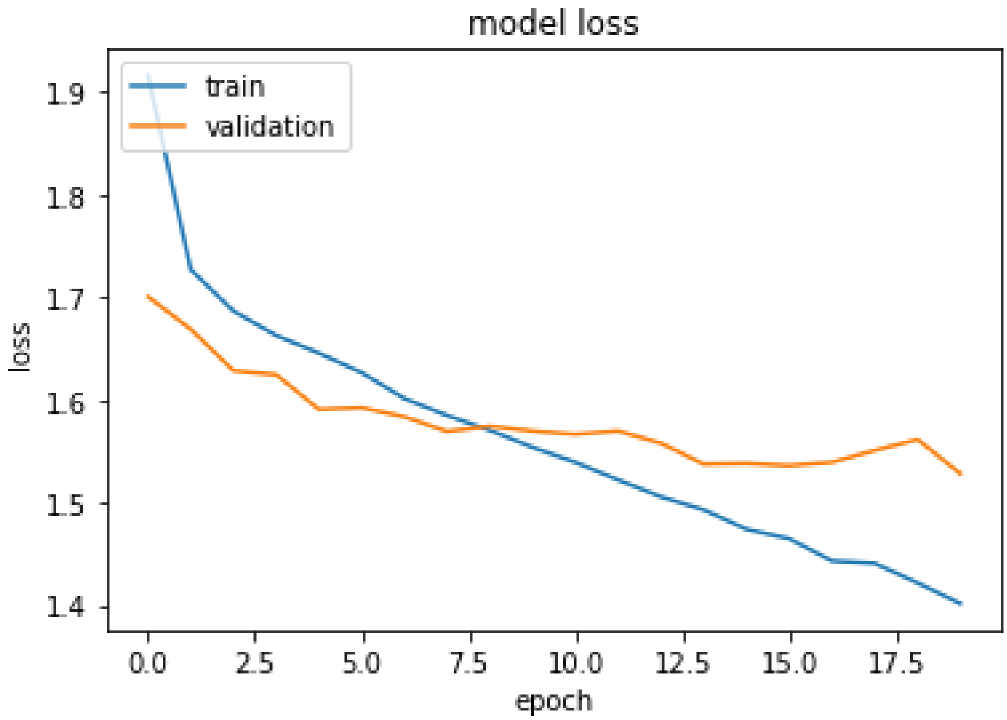
```
Epoch 1/20
628/628 [==============================] - 3s 3ms/step - loss: 1.9157 -
accuracy: 0.2620 - val_loss: 1.7005 - val_accuracy: 0.3281
Epoch 2/20
628/628 [==============================] - 2s 3ms/step - loss: 1.7268 -
accuracy: 0.3102 - val_loss: 1.6687 - val_accuracy: 0.3416
Epoch 3/20
628/628 [==============================] - 2s 3ms/step - loss: 1.6865 -
accuracy: 0.3293 - val_loss: 1.6284 - val_accuracy: 0.3595
Epoch 4/20
628/628 [==============================] - 2s 3ms/step - loss: 1.6627 -
accuracy: 0.3350 - val_loss: 1.6245 - val_accuracy: 0.3500
Epoch 5/20
628/628 [==============================] - 2s 3ms/step - loss: 1.6454 -
accuracy: 0.3452 - val_loss: 1.5907 - val_accuracy: 0.3739
Epoch 6/20
628/628 [==============================] - 2s 3ms/step - loss: 1.6265 -
accuracy: 0.3529 - val_loss: 1.5924 - val_accuracy: 0.3688
```

```
Epoch 7/20
628/628 [==============================] - 2s 3ms/step - loss: 1.6011 -
accuracy: 0.3652 - val_loss: 1.5835 - val_accuracy: 0.3716
Epoch 8/20
628/628 [==============================] - 2s 3ms/step - loss: 1.5847 -
accuracy: 0.3729 - val_loss: 1.5692 - val_accuracy: 0.3908
Epoch 9/20
628/628 [==============================] - 2s 3ms/step - loss: 1.5703 -
accuracy: 0.3796 - val_loss: 1.5744 - val_accuracy: 0.3804
Epoch 10/20
628/628 [==============================] - 2s 3ms/step - loss: 1.5537 -
accuracy: 0.3880 - val_loss: 1.5695 - val_accuracy: 0.3864
Epoch 11/20
628/628 [==============================] - 2s 3ms/step - loss: 1.5394 -
accuracy: 0.3885 - val_loss: 1.5664 - val_accuracy: 0.3790
Epoch 12/20
628/628 [==============================] - 2s 3ms/step - loss: 1.5223 -
accuracy: 0.3957 - val_loss: 1.5696 - val_accuracy: 0.3929
Epoch 13/20
628/628 [==============================] - 2s 3ms/step - loss: 1.5058 -
accuracy: 0.4062 - val_loss: 1.5577 - val_accuracy: 0.3878
Epoch 14/20
628/628 [==============================] - 2s 3ms/step - loss: 1.4931 -
accuracy: 0.4085 - val_loss: 1.5376 - val_accuracy: 0.3990
Epoch 15/20
628/628 [==============================] - 2s 3ms/step - loss: 1.4742 -
accuracy: 0.4136 - val_loss: 1.5383 - val_accuracy: 0.3950
Epoch 16/20
628/628 [==============================] - 2s 3ms/step - loss: 1.4651 -
accuracy: 0.4182 - val_loss: 1.5363 - val_accuracy: 0.3990
Epoch 17/20
628/628 [==============================] - 2s 3ms/step - loss: 1.4434 -
accuracy: 0.4314 - val_loss: 1.5395 - val_accuracy: 0.3890
Epoch 18/20
628/628 [==============================] - 2s 3ms/step - loss: 1.4410 -
accuracy: 0.4343 - val_loss: 1.5507 - val_accuracy: 0.3946
Epoch 19/20
628/628 [==============================] - 2s 3ms/step - loss: 1.4220 -
accuracy: 0.4378 - val_loss: 1.5614 - val_accuracy: 0.3943
Epoch 20/20
628/628 [==============================] - 2s 3ms/step - loss: 1.4018 -
accuracy: 0.4449 - val_loss: 1.5286 - val_accuracy: 0.4129
```

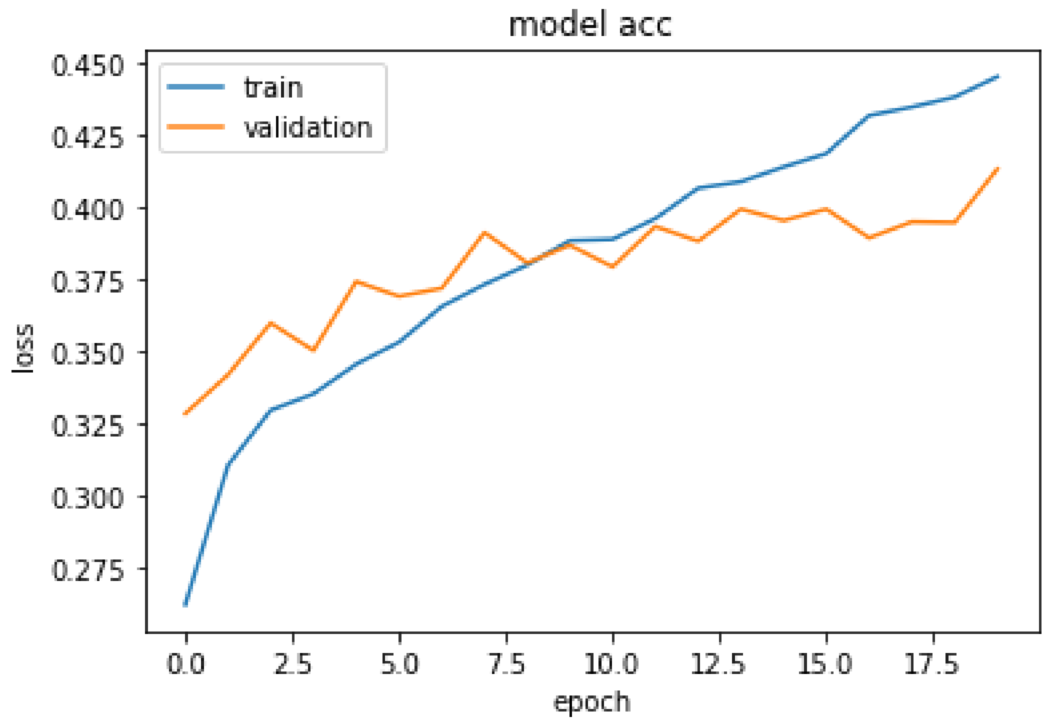
[44]:

```
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()
```



[45]:
```
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()
```

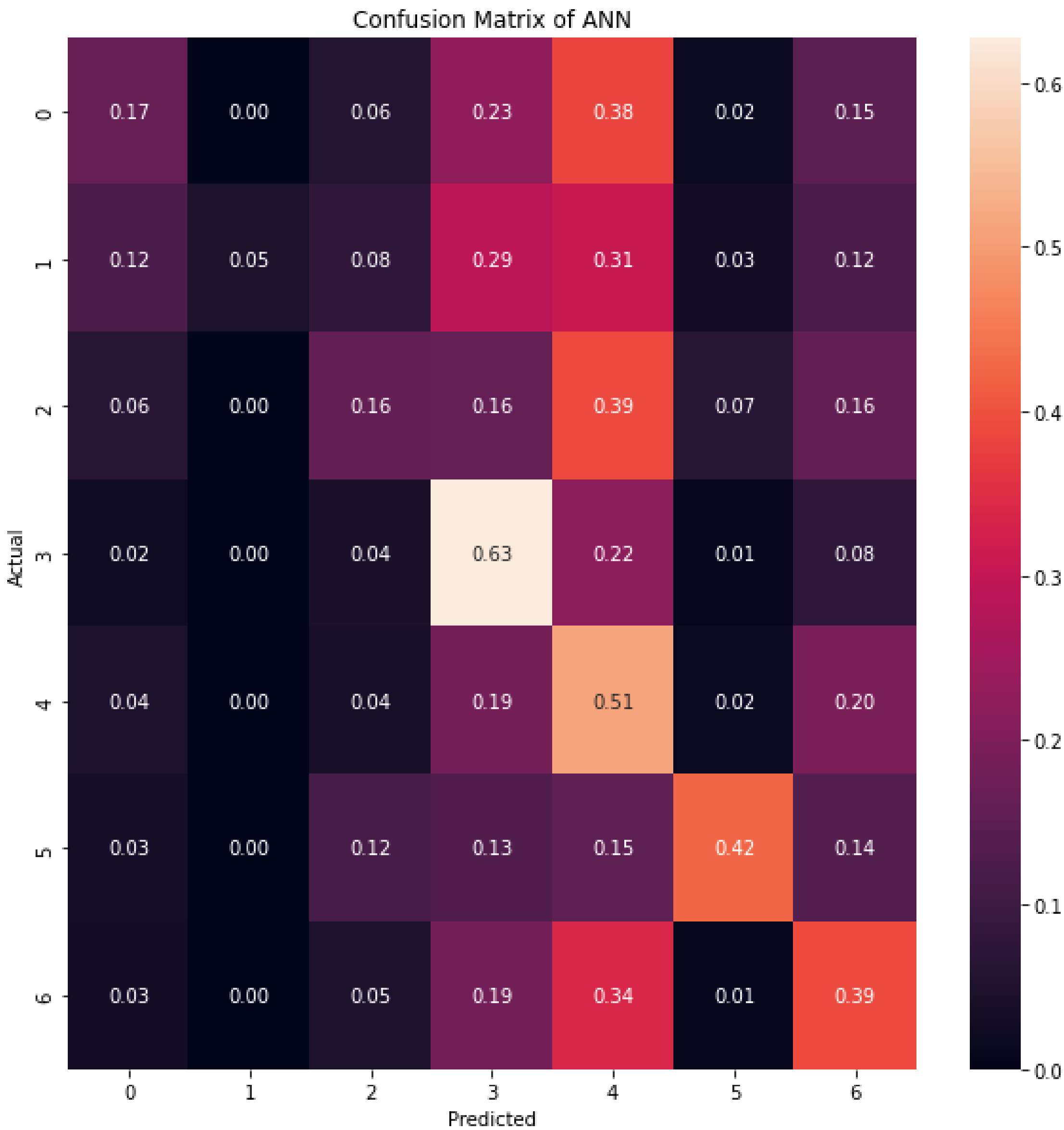


[43]:
```
model.evaluate(test_x, test_y_cat)
```

```
135/135 [==============================] - 0s 2ms/step - loss: 1.5302 -
accuracy: 0.4065
```

[43]: [1.5301593542099, 0.406547486782074]

[50]:
```
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)
```



2.7 Convolutional Neural Network

[51]:
```
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)
```

[52]:
```
model2 = Sequential()
model2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48,␣
 ↪1)))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
model2.add(layers.Dropout(0.5))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Dropout(0.5))
model2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model2.add(layers.Flatten())
model2.add(layers.Dropout(0.2))
model2.add(layers.Dense(64, activation='relu'))
model2.add(layers.Dropout(0.5))
model2.add(layers.Dense(7))
```

[53]:
```
model2.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
model2.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 46, 46, 32)        320

 max_pooling2d (MaxPooling2D  (None, 23, 23, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 21, 21, 64)        18496

 dropout_2 (Dropout)         (None, 21, 21, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 10, 10, 64)       0
 2D)

 dropout_3 (Dropout)         (None, 10, 10, 64)        0

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          36928

 flatten (Flatten)           (None, 4096)              0

 dropout_4 (Dropout)         (None, 4096)              0

 dense_3 (Dense)             (None, 64)                262208

 dropout_5 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 7)                 455

=================================================================
```

```
Total params: 318,407
Trainable params: 318,407
Non-trainable params: 0
_________________________________________________________________
```

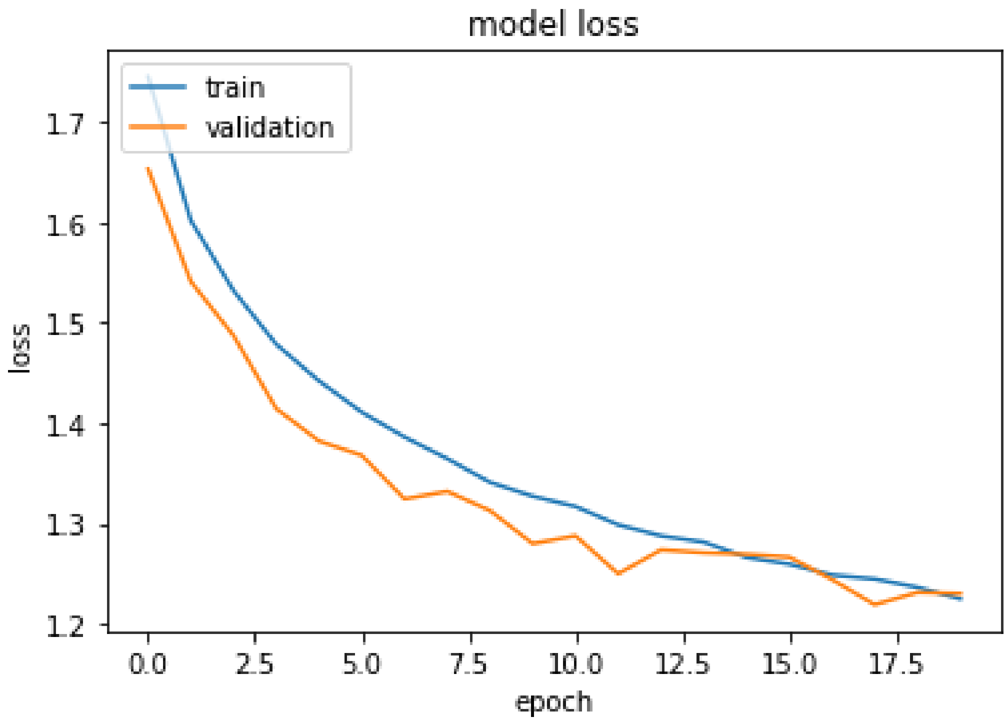
[54]:
```
history2 = model2.fit(train_x_cnn, train_y, epochs = 20, validation_data =␣
 ↪(val_x_cnn, val_y))
```

```
Epoch 1/20
628/628 [==============================] - 12s 18ms/step - loss: 1.7455 -
accuracy: 0.2952 - val_loss: 1.6535 - val_accuracy: 0.3994
Epoch 2/20
628/628 [==============================] - 10s 16ms/step - loss: 1.6020 -
accuracy: 0.3716 - val_loss: 1.5413 - val_accuracy: 0.4371
Epoch 3/20
628/628 [==============================] - 10s 17ms/step - loss: 1.5322 -
accuracy: 0.4031 - val_loss: 1.4873 - val_accuracy: 0.4603
Epoch 4/20
628/628 [==============================] - 11s 17ms/step - loss: 1.4787 -
accuracy: 0.4273 - val_loss: 1.4146 - val_accuracy: 0.4807
Epoch 5/20
628/628 [==============================] - 10s 16ms/step - loss: 1.4423 -
accuracy: 0.4407 - val_loss: 1.3823 - val_accuracy: 0.4868
Epoch 6/20
628/628 [==============================] - 10s 16ms/step - loss: 1.4111 -
accuracy: 0.4538 - val_loss: 1.3680 - val_accuracy: 0.4998
Epoch 7/20
628/628 [==============================] - 10s 16ms/step - loss: 1.3862 -
accuracy: 0.4660 - val_loss: 1.3246 - val_accuracy: 0.5109
Epoch 8/20
628/628 [==============================] - 10s 17ms/step - loss: 1.3646 -
accuracy: 0.4727 - val_loss: 1.3322 - val_accuracy: 0.5193
Epoch 9/20
628/628 [==============================] - 10s 16ms/step - loss: 1.3412 -
accuracy: 0.4868 - val_loss: 1.3130 - val_accuracy: 0.5128
Epoch 10/20
628/628 [==============================] - 10s 16ms/step - loss: 1.3273 -
accuracy: 0.4917 - val_loss: 1.2800 - val_accuracy: 0.5260
Epoch 11/20
628/628 [==============================] - 10s 16ms/step - loss: 1.3171 -
accuracy: 0.4952 - val_loss: 1.2878 - val_accuracy: 0.5251
Epoch 12/20
628/628 [==============================] - 10s 16ms/step - loss: 1.2988 -
accuracy: 0.4991 - val_loss: 1.2500 - val_accuracy: 0.5307
Epoch 13/20
628/628 [==============================] - 10s 16ms/step - loss: 1.2877 -
accuracy: 0.5042 - val_loss: 1.2738 - val_accuracy: 0.5381
Epoch 14/20
```

```
628/628 [==============================] - 10s 16ms/step - loss: 1.2815 -
accuracy: 0.5094 - val_loss: 1.2709 - val_accuracy: 0.5390
Epoch 15/20
628/628 [==============================] - 10s 16ms/step - loss: 1.2663 -
accuracy: 0.5157 - val_loss: 1.2694 - val_accuracy: 0.5437
Epoch 16/20
628/628 [==============================] - 10s 16ms/step - loss: 1.2596 -
accuracy: 0.5146 - val_loss: 1.2666 - val_accuracy: 0.5439
Epoch 17/20
628/628 [==============================] - 10s 16ms/step - loss: 1.2487 -
accuracy: 0.5222 - val_loss: 1.2444 - val_accuracy: 0.5444
Epoch 18/20
628/628 [==============================] - 10s 16ms/step - loss: 1.2448 -
accuracy: 0.5208 - val_loss: 1.2193 - val_accuracy: 0.5462
Epoch 19/20
628/628 [==============================] - 10s 16ms/step - loss: 1.2365 -
accuracy: 0.5246 - val_loss: 1.2314 - val_accuracy: 0.5467
Epoch 20/20
628/628 [==============================] - 10s 17ms/step - loss: 1.2251 -
accuracy: 0.5249 - val_loss: 1.2304 - val_accuracy: 0.5476
```

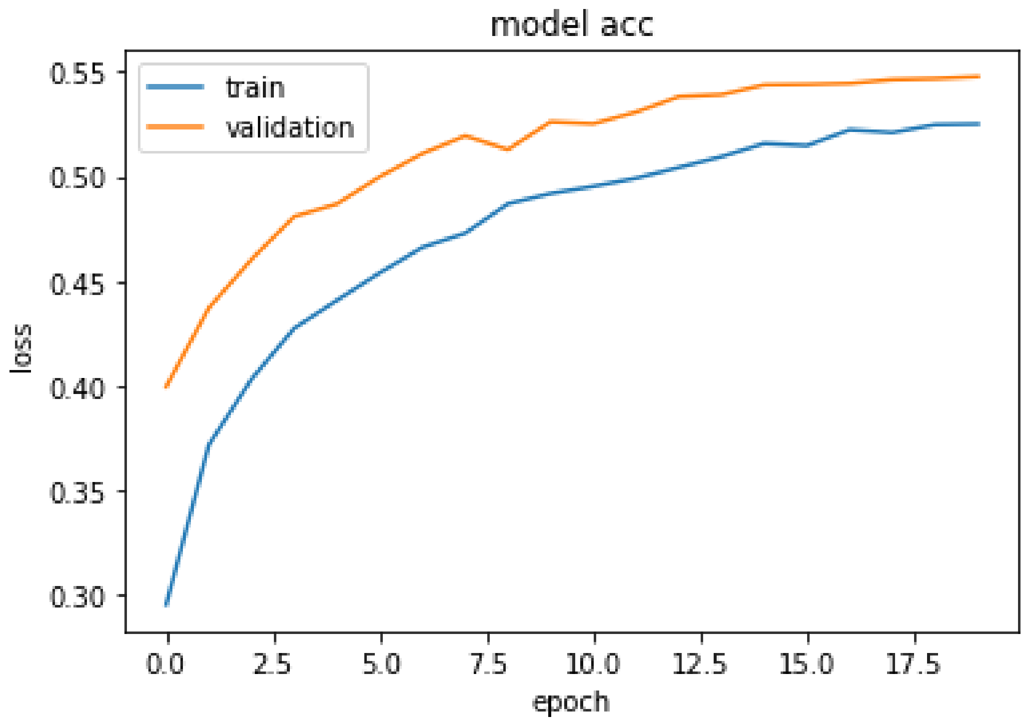
[55]:

```
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



[56]:

```
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('model acc')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

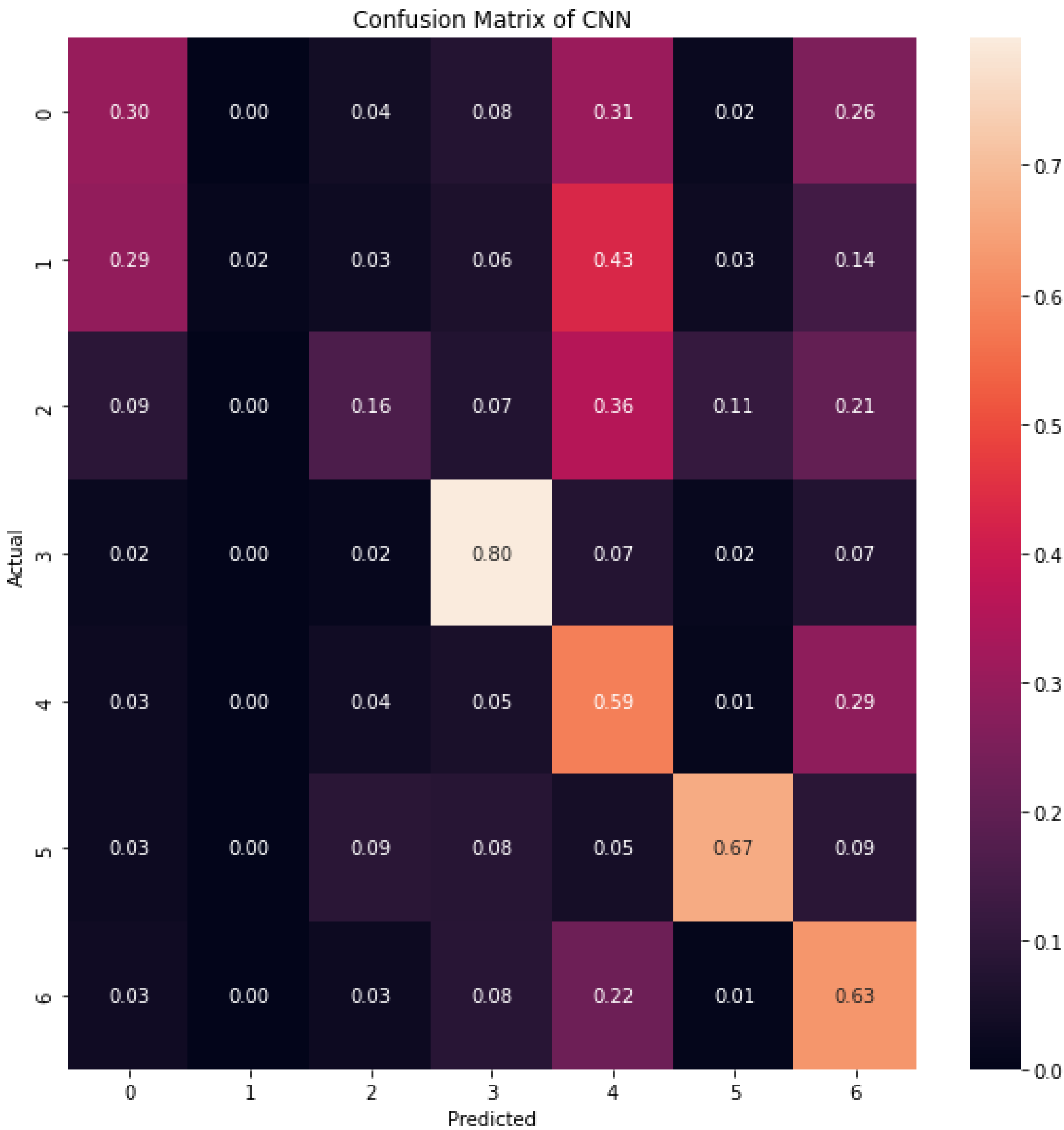


[57]:
```
model2.evaluate(test_x_cnn, test_y)
```

```
135/135 [==============================] - 1s 5ms/step - loss: 1.2294 -
accuracy: 0.5482
```

[57]:
```
[1.229351282119751, 0.5481773614883423]
```

[81]:
```
cm = confusion_matrix(test_y, np.apply_along_axis(np.argmax, 1, model2.
 ↪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)
```



[83]:

```
model.save('saved_model/ann')
model2.save('saved_model/cnn')
```

```
INFO:tensorflow:Assets written to: saved_model/ann/assets
INFO:tensorflow:Assets written to: saved_model/cnn/assets
```