# Neural Network Approach

**In this file, we trying to explore the performance of Neural Network. The performance summary is in the end of the file.**

```
# Import Libraries
import os, sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
import time
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Activation, Flatten, Input, Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras import initializers
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score
import warnings
```

```
# Set working directories
os.chdir('../data/train_set')
root = os.getcwd()
train_dir = root
```

```
# Load the pre-processed Data
train = pd.read_csv('dat_train.csv')
test = pd.read_csv('dat_test.csv')
```

```
# Split and transform the Data
X = train.iloc[:, 0:6006]
Y = train.iloc[:, 6006:6007]
Y = tensorflow.keras.utils.to_categorical(Y)
Y= Y[:,1:]
X_test = test.iloc[:, 0:6006]
Y_test = test.iloc[:, 6006:6007]
Y_test = tensorflow.keras.utils.to_categorical(Y_test)
Y_test = Y_test[:,1:]
```

```
# Build the Network
warnings.filterwarnings('ignore')
input_shape = [6006]
input_layer = Input(input_shape)
x = BatchNormalization()(input_layer)
x = Dense(22*12,activation='relu',kernel_initializer=initializers.glorot_normal(seed=4))(x)
x = Dropout(0.25)(x)
x = BatchNormalization()(x)
x = Dense(22*8,activation='relu',kernel_initializer=initializers.glorot_normal(seed=4))(x)
x = Dropout(0.2)(x)
x = Dense(22*4,activation='sigmoid',kernel_initializer=initializers.glorot_normal(seed=4))(x)
```

```
x = Dense(22*2,activation='tanh',kernel_initializer=initializers.glorot_normal(seed=4))(x)
output_layer = Dense(22,activation='softmax',kernel_initializer=initializers.glorot_normal(seed=4))(x)
model = Model(input_layer,output_layer)
```

```
# Training the network and record training time
warnings.filterwarnings('ignore')
start_time = time.time()
model.compile(loss='categorical_crossentropy',optimizer = Adam(lr=0.001),metrics=['accuracy'])
model_his = model.fit(X,Y,epochs=20)
print('training model takes %s seconds' % round ((time.time() - start_time), 3))
```

```
WARNING:tensorflow:From C:\Users\Dr.FlyOnBeD\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops
Instructions for updating:
Use tf.cast instead.
Epoch 1/20
2000/2000 [==============================] - 3s 1ms/sample - loss: 2.8331 - acc: 0.1625 0s - loss: 2.888
Epoch 2/20
2000/2000 [==============================] - 2s 794us/sample - loss: 2.2803 - acc: 0.2965
Epoch 3/20
2000/2000 [==============================] - 2s 808us/sample - loss: 1.9669 - acc: 0.3765
Epoch 4/20
2000/2000 [==============================] - 2s 771us/sample - loss: 1.8103 - acc: 0.4020
Epoch 5/20
2000/2000 [==============================] - 2s 770us/sample - loss: 1.6740 - acc: 0.4570
Epoch 6/20
2000/2000 [==============================] - 2s 783us/sample - loss: 1.6031 - acc: 0.4820
Epoch 7/20
2000/2000 [==============================] - 2s 779us/sample - loss: 1.4981 - acc: 0.5015
Epoch 8/20
2000/2000 [==============================] - 2s 757us/sample - loss: 1.4491 - acc: 0.5280
Epoch 9/20
2000/2000 [==============================] - 2s 758us/sample - loss: 1.3621 - acc: 0.5480
Epoch 10/20
2000/2000 [==============================] - 2s 778us/sample - loss: 1.3082 - acc: 0.5650s - loss: 1.261
Epoch 11/20
2000/2000 [==============================] - 2s 816us/sample - loss: 1.2426 - acc: 0.5865
Epoch 12/20
2000/2000 [==============================] - 2s 821us/sample - loss: 1.2002 - acc: 0.6085
Epoch 13/20
2000/2000 [==============================] - 2s 773us/sample - loss: 1.1668 - acc: 0.6085
Epoch 14/20
2000/2000 [==============================] - 2s 799us/sample - loss: 1.1437 - acc: 0.6195
Epoch 15/20
2000/2000 [==============================] - 2s 833us/sample - loss: 1.0951 - acc: 0.6295
Epoch 16/20
2000/2000 [==============================] - 2s 765us/sample - loss: 1.0252 - acc: 0.6620s - loss: 1.030
Epoch 17/20
2000/2000 [==============================] - 2s 764us/sample - loss: 1.0153 - acc: 0.6600
Epoch 18/20
2000/2000 [==============================] - 2s 785us/sample - loss: 1.0050 - acc: 0.6505s - loss: 1.000
Epoch 19/20
2000/2000 [==============================] - 2s 781us/sample - loss: 0.9700 - acc: 0.6605
Epoch 20/20
```

```
2000/2000 [==============================] - 2s 793us/sample - loss: 0.9667 - acc: 0.6650
training model takes 34.652 seconds
```

```
# Plot the training process with accuracy as our metric
check = model_his.history
plt.figure(figsize=[8, 6])
plt.plot(check['acc'], 'r', linewidth = 1.0)
plt.xlabel('Epoches', fontsize = 16)
plt.ylabel('Accuracy', fontsize = 16)
plt.title('Accuracy Curve', fontsize = 16)
```

```
Text(0.5, 1.0, 'Accuracy Curve')
```

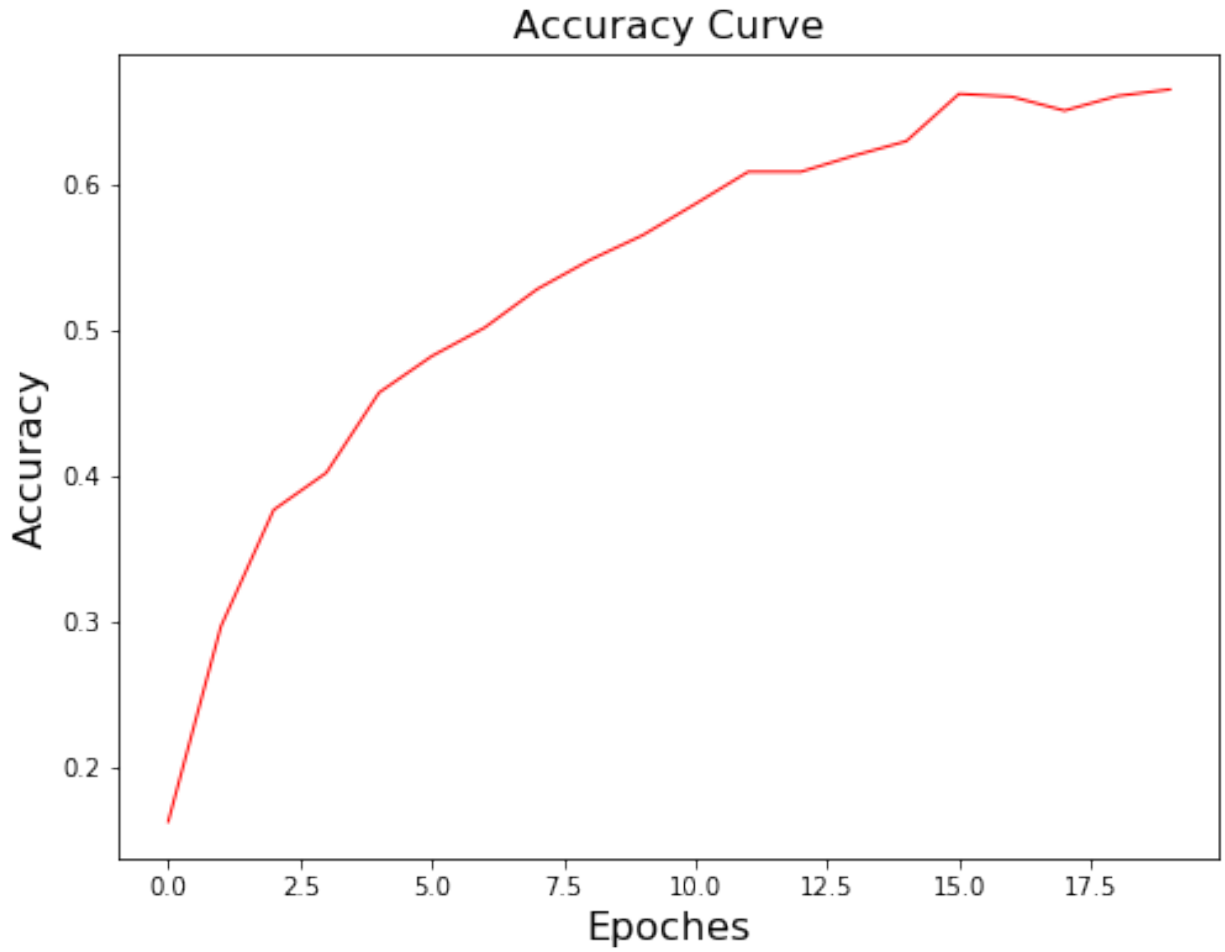


Figure 1: png

```
# Predict the test set and record the test time
t0 = time.time()
pred = model.predict(X_test)
pred_lst = []
for i in range(len(pred)):
    arr = pred[i]
```

```
    idx = np.argwhere(arr == np.max(arr))
    pred_lst.append(idx[0][0])
tst_labl = np.argmax(Y_test, axis=-1)
acc = accuracy_score(pred_lst, tst_labl)
print("Test accuracy is %s percent" %(acc*100))
print("testing model takes %s seconds" % round((time.time() - t0),3))
```

```
Test accuracy is 56.2 percent
testing model takes 0.274 seconds
```

**In summary, the Neural Network did a good job. After 20 epoches of training, the training accuracy is 66.5% and the corresponding test accuracy is 56.2%. Speak of the time cost, Neural Network did a excellent job. The training process only takes 35 seconds and the testing process only takes 0.274 seconds. Overall, the Neural Network gives us a good prediction accuracy within a very short time. Since the most concern of this project is time cost, we consider Neural Network as our advanced model.**