Information Security Analysis and Audit

Project Review 2

Name: Satrunjay Kumar
Registration no: 18BIT0040
Slot: G1

Topic: DATA SECURE SMART HOME AUTOMATION SYSTEM

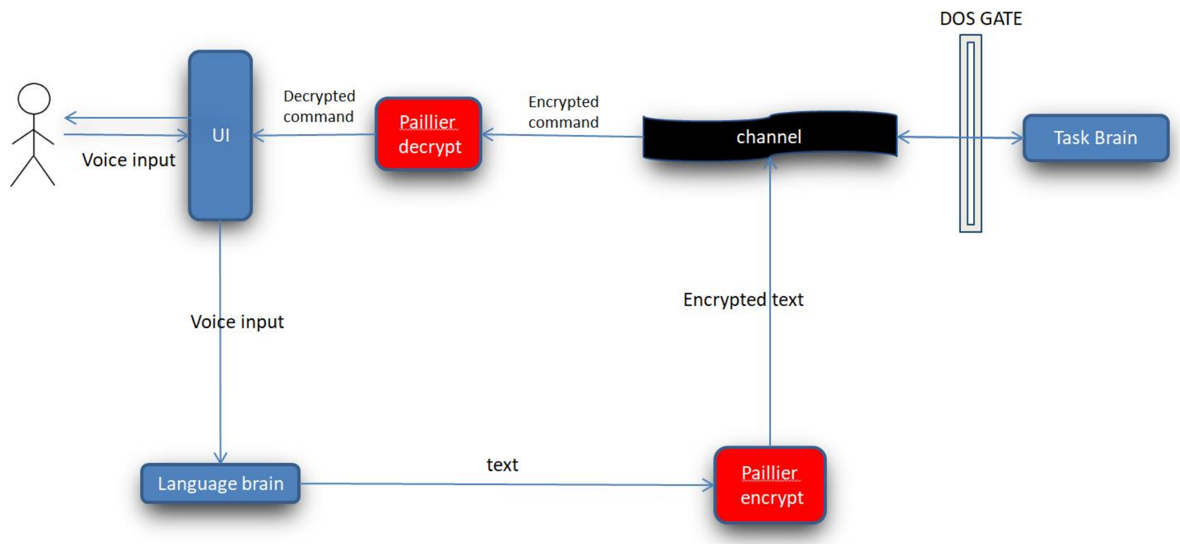
My task: Task Brain using deep learning

1. Design and Description of system – 10 marks (common for the team)

Ans:

DESIGN:

Entire model :



a) Langauge brain:

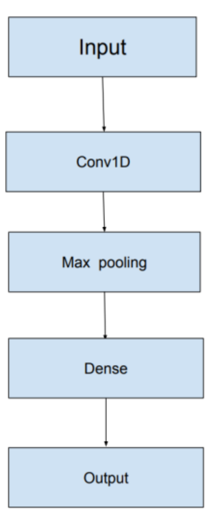


Figure 1: Model

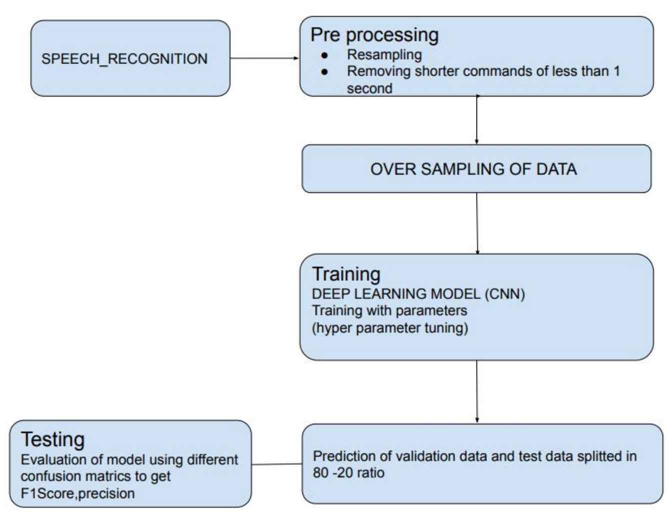


Figure 2: Architecture

b) Encryption

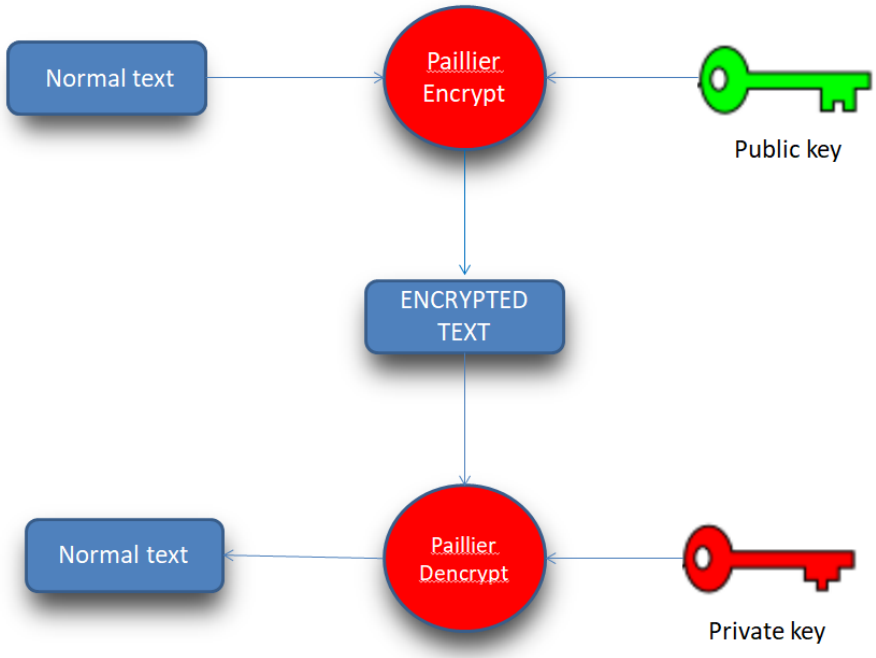


Figure 3: Architecture

c) Task Brain:

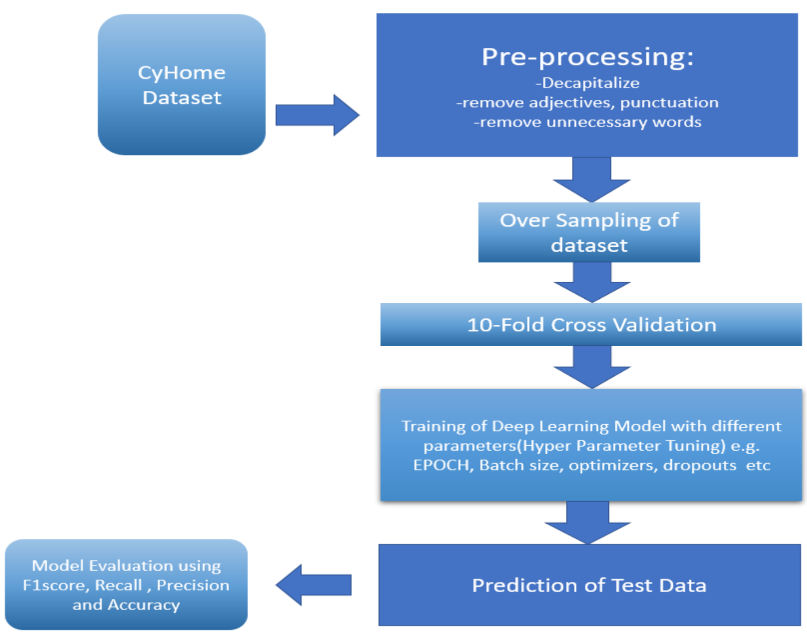


Figure 4: Architecture

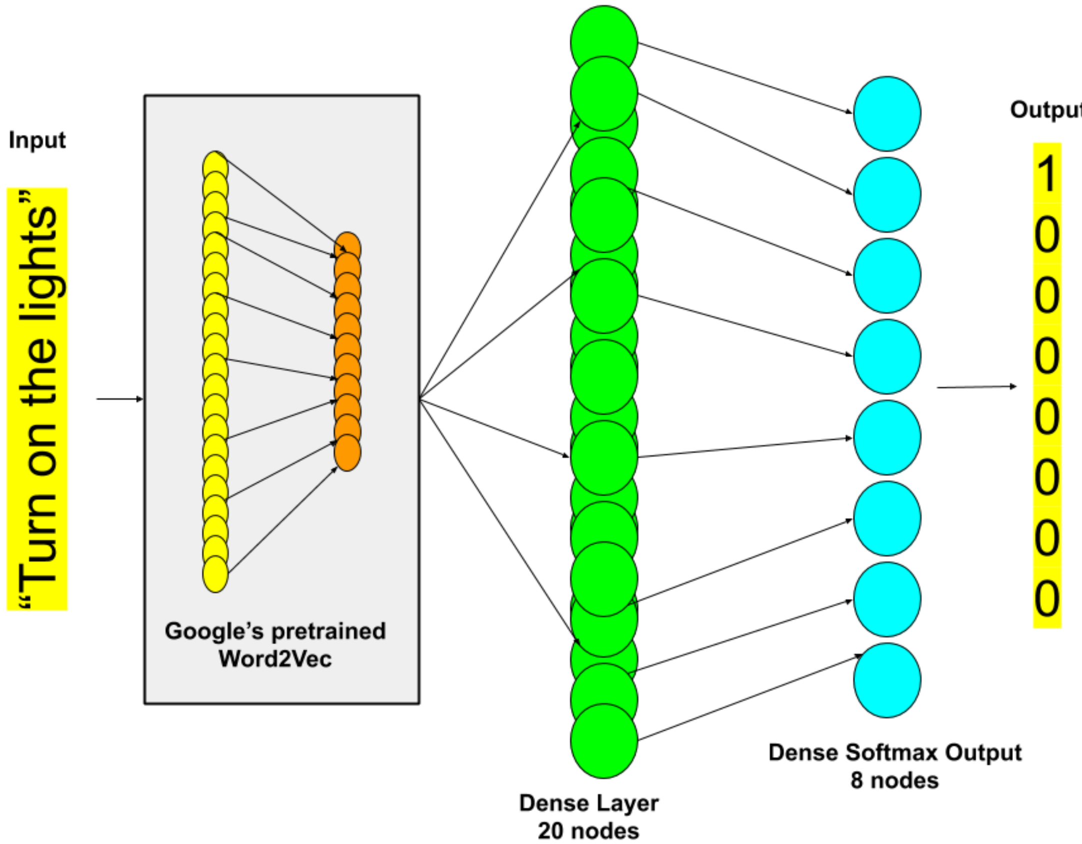


Figure 5: Deep learning model

d) **DOS :**

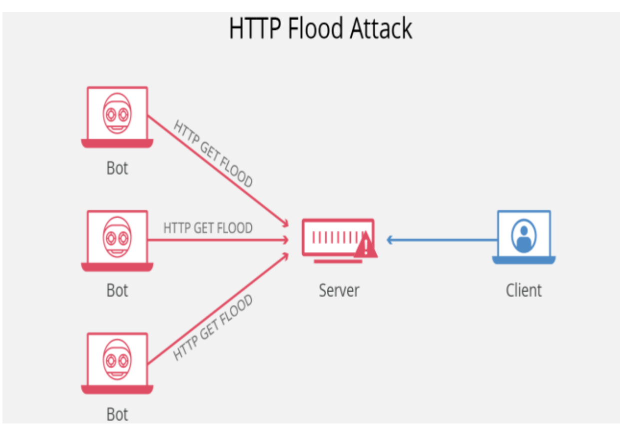


Figure 6: Http Flood Attack

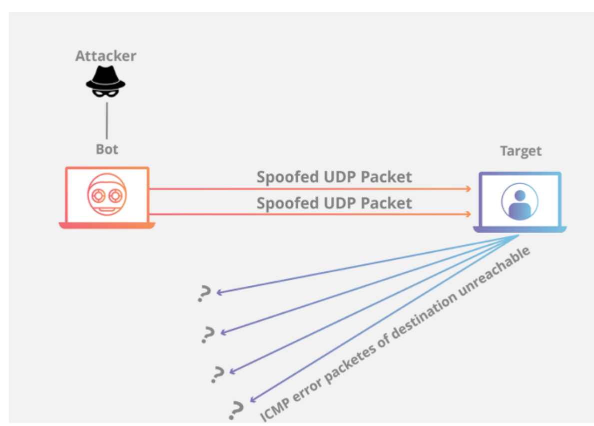


Figure 7: UDP Flood attack

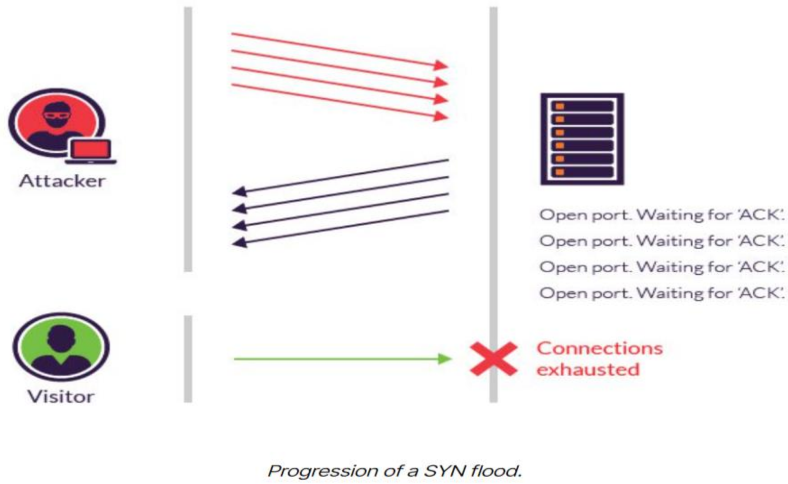


Figure 8: SYN flood attack

DESCRIPTION:

1. Language Brain:

- The user feeds in his voice in the model.
- Preprocessing is done on the voice input in two steps :
    1. Resampling
    2. Removing shorter commands of less than 1 second
- Oversampling of data is done to get 70 percent balanced data
- Training of CNN model is done with different hyper tuning parameters and evaluated many times with different parameters to get better accuracy.
- After that prediction is done with many different parameter and best model is selected
- Last the model is tested and generation of confusion matrics are done to evaluate the model.
- The converted text from the voice input is obtained as the output.

2.Encryption and Task brain:

- The text is then then preprocessed at the client side to convert it to a vector of length 186 by:
    - Decapitalizing each sentence
    - Removing Adjectives and punctuations
    - Removing unnecessary words
- This vector of integers is then encrypted using public key and sent to the server where the task brain resides.

- After this the pre-processed encrypted dataset is oversampled to to get minimum of 70% balanced dataset.
- Then 10-Fold cross validation is used to get max accuracy with a fixed parameter.
- Also parameters are changed (Several times)and 10 fold cross validation is done again.
- At last the best model among all the models is taken and testing is done in that model.
- Finally, to evaluate the model F1-score, Precision, Recall and Accuracy along with confusion matrix is used.
- The trained model processes the input encrypted vector and returns an array of length 8.
- This new array is sent back to the client side.
- It is decrypted using private key and then the values are compared with a predefined dictionary of commands.
- The command with the highest probability is executed in the client side.

3) DOS attack prevention:

- We will be preventing some of the DOS attack on the server where our task brain is located so that the channel between the client and the server is secure.

2. Preprocessing (normalization, sampling – oversampling, undersampling (technique used for sampling – atleast 70%balanced)

➔ Raw dataset:

```
In [2]: #VIEW RAW DATASET
        raw=pd.read_csv('genDS.csv')
        raw.head(10)
```

Out[2]:

	String	Trigger
0	Turn on the lights	#LIGHTS_ON
1	Switch on the light	#LIGHTS_ON
2	Bring the lights on	#LIGHTS_ON
3	Light up the room	#LIGHTS_ON
4	Wake up light	#LIGHTS_ON
5	Get the lights running	#LIGHTS_ON
6	Turn the lights on	#LIGHTS_ON
7	Switch the light on	#LIGHTS_ON
8	Turn on the bulb	#LIGHTS_ON
9	Light up the bulb	#LIGHTS_ON

Preprocessing code for refining the raw string to remove unnecessary words and decaptilize it:

```
def aggressive_fermenter(s):

    s=s.lower()
    stopwords=["i am","i "," the"," is"," and"," could","to "," please"," that","just","my"," some"," goddamn"," can"," would"," a
             " these"," their"," those","will"," should","you"," our"," want",","," me","of "]
    for stopper in stopwords:
        s=s.replace(stopper,'')

    replace_rules=[["ness",""],["es","e"],["sses","ss"],["ies","y"],["s",""]]

    for rule in range(len(replace_rules)):
        regstring=replace_rules[rule][0]+r"\s+"
        endreg=replace_rules[rule][0]+r"$"
        s=re.sub(regstring,replace_rules[rule][1]+" ",s)
        s=re.sub(endreg,replace_rules[rule][1]+" ",s)


    s=re.sub(r"\s\s"," ",s)
    s=re.sub(r"^\s","",s)
    s=re.sub(r"\s$","",s)
    return s
```

Over Sampling(sample code for over sampling but I didn't use it because my dataset is balanced already)

```
In [9]: from imblearn.over_sampling import RandomOverSampler
        oversample = RandomOverSampler(sampling_strategy='minority')
        X_over, y_over = oversample.fit_resample(X, y)
```

3. training and cross-validation with hyper parameter tuning ➔

model(3 models with different parameters)

```
def build_model():
    taskbrain=models.Sequential()
    taskbrain.add(word2vec)
    taskbrain.add(layers.Dense(20))
    taskbrain.add(Dropout(0.2))
    taskbrain.add(layers.Dense(8,activation='softmax',input_shape=(20,)))
    taskbrain.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy','mae','mse'])
    return taskbrain

def build_model1():
    taskbrain=models.Sequential()
    taskbrain.add(word2vec)
    taskbrain.add(layers.Dense(20))
    taskbrain.add(Dropout(0.3))
    taskbrain.add(layers.Dense(8,activation='softmax',input_shape=(20,)))
    taskbrain.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy','mae','mse'])
    return taskbrain

def build_model2():
    taskbrain=models.Sequential()
    taskbrain.add(word2vec)
    taskbrain.add(layers.Dense(20))
    taskbrain.add(Dropout(0.5))
    taskbrain.add(layers.Dense(8,activation='softmax',input_shape=(20,)))
    taskbrain.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy','mae','mse'])
    return taskbrain
```

10 fold cross validation with parameter 1(epoch=100, batch_size=64 with model 1).

```
#parameters 1:
from keras.callbacks import EarlyStopping
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5)
k=10
print('training in progress:...\n\n')
num_val_samples = len(x_train) // k
num_epochs = 100
all_scores = []
all_acc=[]
allmodels=[]
history=[]
for i in range(k):
    print('\nprocessing fold #', i)
    val_data = x_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_x_train = np.concatenate(
        [x_train[:i * num_val_samples],
        x_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_y_train = np.concatenate(
        [y_train[:i * num_val_samples],
        y_train[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model1()
    cur_history=model.fit(partial_x_train, partial_y_train, epochs=num_epochs, batch_size=64, validation_data=(val_data,val_target
    loss,acc,mae,mse=model.evaluate(val_data, val_targets)
    allmodels.append(model);
    history.append(cur_history)
    all_scores.append(mae)
    all_acc.append(acc)
    print('='*100)
```

10 fold cross validation with parameter 2(epoch=10, batch_size=10 with model 2).

In [125]:

```
#parameters 2:
from keras.callbacks import EarlyStopping
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
k=10
print('training in progress:...\n\n')
num_val_samples = len(x_train) // k
num_epochs = 100
all_scores = []
all_acc=[]
history=[]
allmodels=[]
for i in range(k):
    print('\nprocessing fold #', i)
    val_data = x_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_x_train = np.concatenate(
        [x_train[:i * num_val_samples],
        x_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_y_train = np.concatenate(
        [y_train[:i * num_val_samples],
        y_train[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model2()
    cur_history=model.fit(partial_x_train, partial_y_train, epochs=num_epochs, batch_size=10, validation_data=(val_data,val_targe
    loss,acc,mae,mse=model.evaluate(val_data, val_targets)
    history.append(cur_history)
    allmodels.append(model);
    all_scores.append(mae)
    all_acc.append(acc)
    print('='*100)
```

10 fold cross validation with parameter 3(epoch=30, batch_size=10 with model 1).

```
#parameters 3:
k=10
print('training in progress:...\n\n')
num_val_samples = len(x_train) // k
num_epochs = 30
all_scores = []
all_acc=[]
history=[]
for i in range(k):
    print('\nprocessing fold #', i)
    val_data = x_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_x_train = np.concatenate(
        [x_train[:i * num_val_samples],
        x_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_y_train = np.concatenate(
        [y_train[:i * num_val_samples],
        y_train[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    cur_history=model.fit(partial_x_train, partial_y_train, epochs=num_epochs, batch_size=10, validation_data=(val_data,val_target
    loss,acc,mae,mse=model.evaluate(val_data, val_targets)
    history.append(cur_history)
    all_scores.append(mae)
    all_acc.append(acc)
    print('='*100)
```

10 fold cross validation with parameter 4(epoch=30, batch_size=32 with model 3).

```
#parameters 4:
k=10
print('training in progress:...\n\n')
num_val_samples = len(x_train) // k
num_epochs = 30
all_scores = []
all_acc=[]
history=[]
for i in range(k):
    print('\nprocessing fold #', i)
    val_data = x_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_x_train = np.concatenate(
        [x_train[:i * num_val_samples],
        x_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_y_train = np.concatenate(
        [y_train[:i * num_val_samples],
        y_train[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    cur_history=model.fit(partial_x_train, partial_y_train, epochs=num_epochs, batch_size=32, validation_data=(val_data,val_target
    loss,acc,mae,mse=model.evaluate(val_data, val_targets)
    history.append(cur_history)
    all_scores.append(mae)
    all_acc.append(acc)
    print('='*100)
```

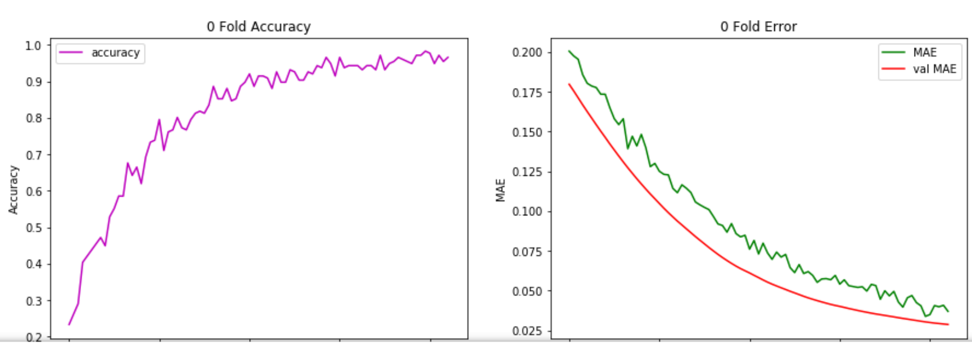
10 fold cross validation with parameter 5(epoch=100, batch_size=64 with model 2).

```
#parameters 5:

from keras.callbacks import EarlyStopping
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
k=10
print('training in progress:...\n\n')
num_val_samples = len(x_train) // k
num_epochs = 100
all_scores = []
all_acc=[]
history=[]
allmodels=[]
for i in range(k):
    print('\nprocessing fold #', i)
    val_data = x_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]
    partial_x_train = np.concatenate(
        [x_train[:i * num_val_samples],
        x_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_y_train = np.concatenate(
        [y_train[:i * num_val_samples],
        y_train[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    cur_history=model.fit(partial_x_train, partial_y_train, epochs=num_epochs, batch_size=64, validation_data=(val_data,val_target
    allmodels.append(model);
    loss,acc,mae,mse=model.evaluate(val_data, val_targets)
    history.append(cur_history)
    all_scores.append(mae)
    all_acc.append(acc)
    print('='*100)
```

Accuracy vs epoch, mae and val mae vs epoch graph plotting for training

```
In [11]: #PLOTTING LOSS AND ACCURACY GRAPHS AFTER TRAINING THE MODEL WITH K=10 FOLDS:
print('Cross validation''s acc and mae graphs')
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(15,120))
for i in range(k):
    ax=fig.add_subplot(20,2,2*i+1)
    ax.plot(history[i].history['accuracy'],'m-',label='accuracy')
    ax.set_xlabel('Epochs')
    ax.set_ylabel('Accuracy')
    ax.legend()
    #ax.title(' Fold accuracy')
    ax.set_title(str(i)+' Fold Accuracy')

    ax2=fig.add_subplot(20,2,2*i+2)
    ax2.plot(history[i].history['mae'],'g-',label='MAE')
    ax2.plot(history[i].history['val_mae'],'r-',label='val MAE')
    ax2.set_xlabel('Epochs')
    ax2.set_ylabel('MAE')
    ax2.legend()
    ax2.set_title(str(i)+' Fold Error')
    #axs[1, 0].set_title('Axis [1, 0]')
```

```
Cross validations acc and mae graphs
```



4. Testing:

→prediction

```
#model7=allmodels[6]
y_pred=model7.predict(x_test)
for i in range(0,len(y_pred)):
    mval=max(y_pred[i]);
    for j in range(0,8):
        if (y_pred[i][j]==mval):
            y_pred[i][j]=1
        else:
            y_pred[i][j]=0
y_test
```

```
ytest=np.zeros(len(y_test),dtype=int)
ypred=np.zeros(len(y_test),dtype=int)
for i in range(0,len(y_test)):
    for j in range(0,8):
        if (y_test[i][j]==1):
            ytest[i]=j+1
            break

for i in range(0,len(y_test)):
    for j in range(0,8):
        if (y_pred[i][j]==1):
            ypred[i]=j+1
            break
ypred
```

```
Out[200]: array([3, 8, 7, 5, 3, 8, 4, 7, 6, 1, 7, 7, 5, 4, 6, 8, 7, 2, 1, 1, 8, 1,
       4, 7, 5, 6, 4, 3, 2, 7, 7, 1, 1, 7, 3, 8, 2, 1, 1, 6, 7, 4, 4, 3,
       2, 4, 1, 1, 1])
```

Confusion matrix:

```
In [193]: from sklearn.metrics import confusion_matrix
          confusion_matrix(ytest, ypred)

Out[193]: array([[6, 0, 0, 0, 0, 0, 0, 1],
                 [2, 7, 0, 0, 0, 0, 0, 0],
                 [0, 0, 5, 0, 0, 0, 0, 0],
                 [0, 0, 0, 5, 0, 0, 0, 0],
                 [0, 0, 0, 0, 4, 1, 0, 1],
                 [0, 0, 0, 0, 0, 2, 0, 0],
                 [0, 0, 0, 0, 0, 0, 6, 0],
                 [2, 0, 0, 0, 0, 0, 5, 2]], dtype=int64)
```

F1score, precision ,recall and accuracy

```
In [194]: from sklearn.metrics import f1_score
          f1_score(ytest, ypred, average='weighted')

Out[194]: 0.7391979868870625

In [201]: from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
          precision_score(ytest, ypred, average='weighted')

Out[201]: 0.836522661012457

In [196]: recall_score(ytest, ypred,average='weighted')

Out[196]: 0.7551020408163265

In [197]: accuracy_score(ytest, ypred)

Out[197]: 0.7551020408163265

In [ ]:
```