LOVELY PROFESSIONAL UNIVERSITY

A deep learning model for organic chemical compounds prediction using CNN

By

Vikas Kumar (11710782)

B.Tech (Computer Science and Engineering)

A deep learning model for organic chemical compounds prediction using CNN

In this task, we will train a model to predict the compounds are classified as either ‘Musk’ or ‘Non-Musk’ compounds. Our training data consists of 170 variables. 169 variables are predictor variables, with the last being the target variable.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import f1_score, precision_score, recall_score
import pandas_profiling as pp
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

data=pd.read_csv("/content/drive/My Drive/Predicting-musk-non-musk/
musk_csv.csv")
data.head(20)
```

The above code will print the first 20 rows of our imported data. You will see the column names printed along with the data in a tabular format.

After that we check the shape or size and missing values of our data set by using following code.

```
print("Printing shape of the csv file")

data.shape

print("Checking missing values")

data.isna().sum()
```

Droping high correlation columns :

I have a huge data set and prior to machine learning modeling it is always suggested that first you should remove highly correlated descriptors(columns) how can i calculate the column wice correlation and remove the column with a threshold value say remove all the columns or descriptors having >0.92 correlation.

```
corr_matrix = data.corr().abs()

upper=corr_matrix.where(np.triu(np.ones(corr_matrix.shape),k=1).astype(np.bool))
to_drop = [column for column in upper.columns if any(upper[column] > 0.92)]

data = data.drop(columns = to_drop)
```

Next, we will import model_selection from scikit-learn, and use the function train_test_split( ) to split our data into two sets 80:20 ratio :

```
train,test = train_test_split(data, random_state=30, test_size = 0.2)

X_train = train.iloc[:,3:-1]
Y_train = train.iloc[:,-1:]
X_test = test.iloc[:,3:-1]
Y_test = test.iloc[:,-1:]
X_train.shape
```

Making CNN Models :

We are now reading to start building our Neural Network. We will make use of a *Sequential* model.

```
import tensorflow as tf

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

model=Sequential()
```

We can now add layers to our model. We will be creating fully connected layers using model.add().

We need to tell each layer what its output will be, which is the number of neurons it will output. We also need to specify the activation of the layers. In this case, we use the *relu* activation function.

```
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(19,6,1)))
model.add(Conv2D(64,(3,3),activation='relu'))
```

```
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))
```

Next, we need to compile our model. We do this by specifying our *loss function* and our *optimizer* .

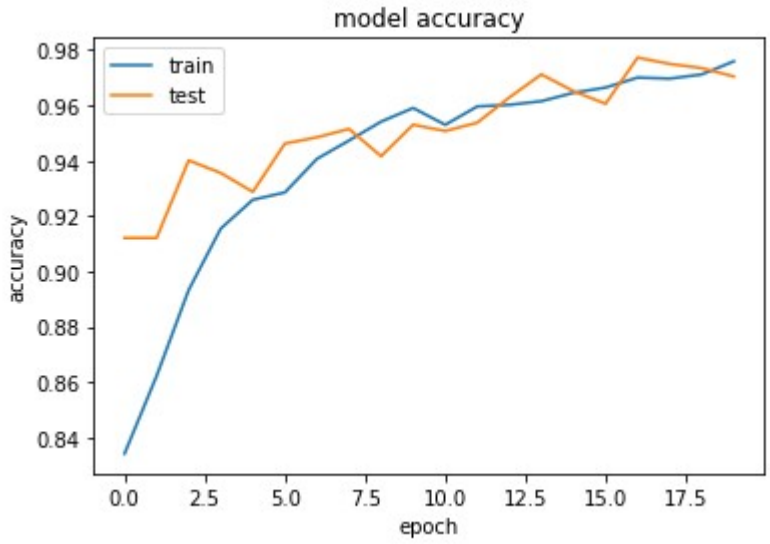
```
model.compile(loss=keras.losses.binary_crossentropy,optimizer=keras.optimizer.Adadelta(),metrics=['accuracy'])
```

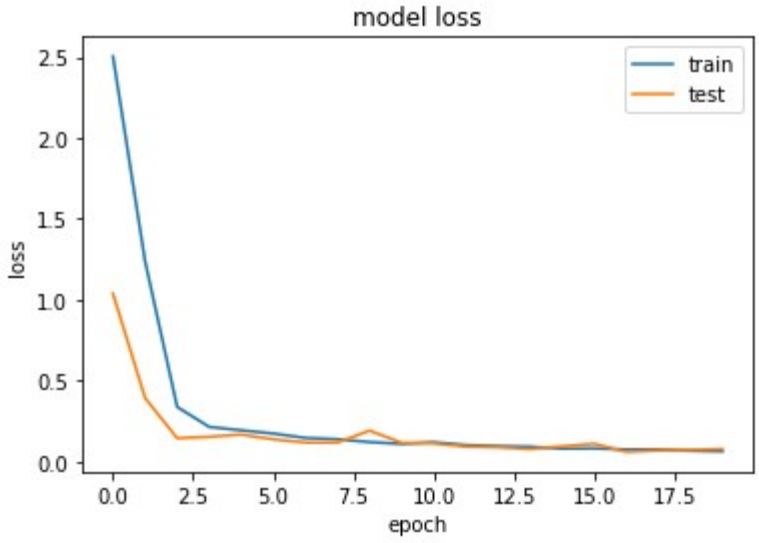
We are now ready to train our model. We will train our model by passing in our training dataset. We also need to specify the number of times we would like to go over our training data. This is called an epoch.

```
history = model.fit(x_train,Y_train,batch_size=128,epochs=20,validation_data=(x_test,Y_test))

score=model.evaluate(x_test,Y_test,verbose=0)
print(score)
```

Model Loss and Accuracy Graph :





Measures of our model including validation accuracy, loss, precision, recall, F1 score.

```
print("f1_score:",f1_score(Y_test,model.predict_classes(x_test),))
print("recall:",recall_score(Y_test,model.predict_classes(x_test),))
print("precision",precision_score(Y_test,model.predict_classes(x_test),))
print("Validation Loss:",score[0])
print("Validation Accuracy:",score[1])
```

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. We have got 0.849 precision which is pretty good.

Precision = TP/TP+FP

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations. We have got recall of 0.932.

Recall = TP/TP+FN

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy.In our case, F1 score is 0.888.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)