Introduction:

Once the basics of neural networks have been understood it is time to develop much more powerful models that can be used in real world applications. In this project the goal will be to develop a model that can obtain numerical chemical attributes pertaining to a given wine and output value(s) that will represent the quality of a wine. In order to begin developing a network that can perform such tasks, it is necessary to decide on what model to implement. The models that will be used are a regression model and a classification model. The comparison of these two models should offer insight into which technique is better for the given data set. In addition, it serves as a learning experience when dealing with similar data sets. The data used to train the model will be the following (obtained from the Machine Learning Repository of the University of California) :

Red wine document is comprised of 4898 samples "http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
White wine document is comprised of 1599 samples
"http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv"

Procedures:

Adapting the data for training:

- Combine the red and white wine samples into one pool totaling 6497 samples
- Add attribute to data known as type:
    - type = 0 (red wine)
    - type = 1 (white wine)
- Separate the data into:
    - Xtrain & y_train
    - Xval & y_val
    - Xtst & y_tst (the last 10 patterns of val)

Regression model:

- First obtain the separated data (Xtrain, Xval & Xtst) and normalize the data:
    - Xtrain_N
    - Xval_N
    - Xtst_N
- Develop the regression models:
    - regmodl1
    - regmodl2 (capable of overfitting)
    - regmodl3 (best model)
- Finally graph and display the following results:
    - Loss vs Epochs graph (training and validation)
    - Final Loss and Final Mae (training and validation)
    - Using regmodl3 predict the output and calculate error

Classification mode:

- Use the separated data as it is
- Perform a random classification on the data set
- Develop classification models:
    - clasmodl1
    - clasmodl2 (capable of overfitting)
    - clasmodl3 (best model)
- Finally graph and display the following results:
    - Loss vs Epochs graph (training and validation)
    - Final Loss and Final Accuracy
    - Using regmodl3 predict the output and demonstrate hits

Results:

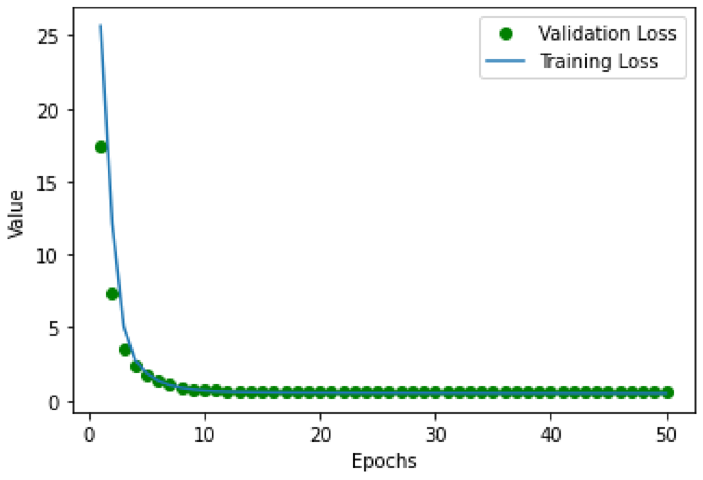
In this section the plots and final values pertaining to each model will be presented. In addition the structure of the network alongside its complete description (i.e number of layers, activation function, processing elements, etc.) will be encompassed in this section. Finally a brief description will be provided regarding the changes that were made between from one model to the next.

II.2 Regression Model 1 (regmodl1)

The first regression model that was built was a network that consisted of a hidden layer containing eight processing elements and a final output layer containing only one processing element. The activation function of the hidden layer is "relu" while the output layer uses a linear activation function. In this network the loss function used was the mse (Mean Square error) function and the metrics obtained from it was the mean (Mean Absolute Error). This was accomplished using the "rmsprop" optimizer. The figure below shows how this model was implemented in keras. The model was built and it was trained for a total of 50 epochs The batch size that was used to train the model was of size 32. The image shown in figure 2 shows the validation vs training loss, as well as the final values pertaining to loss and mae.

```
def build_regmodl1():
  regmodl1 = keras.Sequential(
      [
        layers.Dense(8, activation = 'relu'),
        layers.Dense(1)
      ]
)
  regmodl1.compile(optimizer = "rmsprop", loss = "mse", metrics =
["mae"])
  return regmodl1
```

Figure 1: Regression Model 1 structure and Parameters



```
Final Training loss:
0.48152193427085876
Final Training MAE:
0.5442901849746704
Final Validation loss:
0.5412675738334656
Final Validation MAE:
0.5635526180267334
```

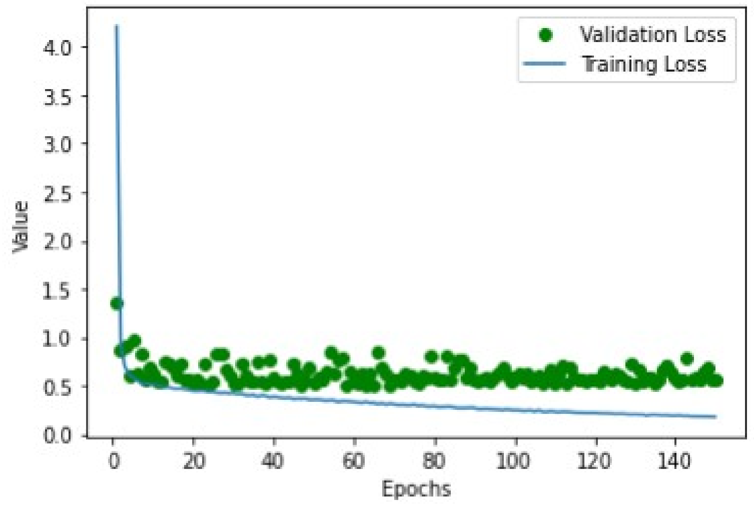
Figure 2: Regression Model 1 MSE vs Epochs

II.2 Regression Model 2 (regmodl2)

The second regression model was improved by adding more layers and processing elements per layer. In this case regmodl2 consists of four hidden layers and an output layer. The four hidden layers contain the following number of processing elements (100, 50, 80, 20) and the output layer contains only one processing element. This can be observed in the figure below. The model was built and it was trained for a total of 150 epochs The batch size that was used to train the model was of size 64. The image shown in figure 4 shows the validation vs training loss, as well as the final values pertaining to loss and mae.

```
def build_regmodl2():
 regmodl2 = keras.Sequential(
     [
       layers.Dense(100, activation = 'relu'),
       layers.Dense(50, activation = 'relu'),
       layers.Dense(80, activation = 'relu'),
       layers.Dense(20, activation='relu'),
       layers.Dense(1)
     ]
 )
 regmodl2.compile(optimizer = "rmsprop", loss = "mse", metrics =
["mae"])
 return regmodl2
```

Figure 3: Regression Model 2 Structure and Parameters



```
Final   Training   loss:
0.17591653764247894
Final   Training   MAE:
0.3224025368690491
Final  Validation  loss:
0.5634228587150574
Final  Validation  MAE:
0.5464485883712769
```

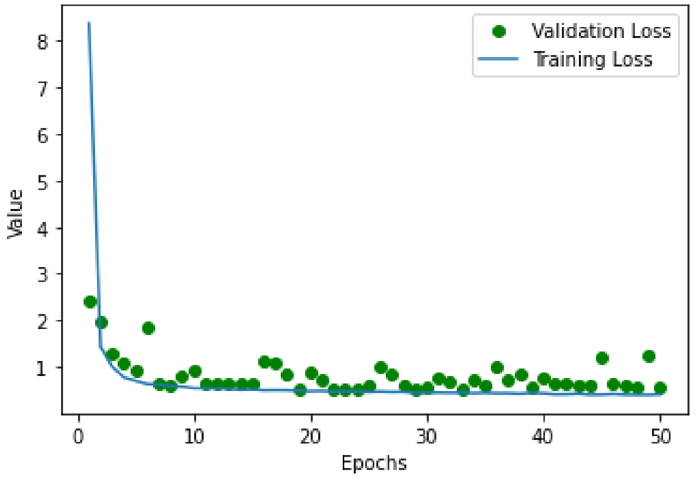
Figure 4: Regression Model 2 MSE vs Epochs

<u>II.3 Regression Model 3 (regmodl3)</u>

The final regression model was kept the same as the regression model 2. Specifically, the number of layers and processing elements pertaining to each layer was kept the same. The figure below shows how the regmodl3 was built and compiled in keras. However, the hyperparameters used for training the model were changed. The regression model 3 was trained for a total of 50 epochs and used a batch size of 128. The image shown in figure 6 shows the validation vs training loss, as well as the final values pertaining to loss and mae.

```
def build_regmodl3():
 regmodl3 = keras.Sequential(
     [
       layers.Dense(100, activation = 'relu'),
       layers.Dense(50, activation = 'relu'),
       layers.Dense(80, activation = 'relu'),
       layers.Dense(20, activation='relu'),
       layers.Dense(1)
     ]
 )
 regmodl3.compile(optimizer = "rmsprop", loss = "mse", metrics =
["mae"])
 return regmodl3
```

Figure 5: Regression Model 3 Structure and Parameters



```
Final   Training   loss:
0.4100159704685211
Final    Training   MAE:
0.4969372749328613З
Final  Validation  loss:
0.5416620373725891
Final   Validation   MAE:
0.5587565898895264
```

Figure 6: Regression Model 3 MSE vs Epochs

Once the training and validation of the best regression model (regmodl3) is done, it's time to test how this model does with the test set. In the table below we can see what the regression model predicted and the error with respect to its target. In the table below we can see that the predicted values are not too far off from their corresponding targets. Specifically, patterns 9,5 and 7 are patterns that are clearly too far away from the target. Aside from these patterns, the model can produce an activation that is very close to their respective target with maybe the exception being pattern 4, whose error is at a halfway point (~.5).

Table I. Comparison of Activation vs Target

Pattern #	**f1**	**f2**	**f3**	**f4**	**f5**	**f6**	**f7**	**f8**	**f9**	**f10**	**f11**	**f12**	**activation**	**target**	**error**
1	7.4	0.44	0.2	11.5	0.049	44	157	0.998	3.27	0.44	9	0	4.817978382	5	0.1820216179
2	6.5	0.23	0.36	16.3	0.038	43	133	0.99924	3.26	0.41	8.8	0	5.077382565	5	-0.07738256454
3	5.6	0.41	0.24	1.9	0.034	10	53	0.98815	3.32	0.5	13.5	0	6.584134579	7	0.4158654213
4	6.4	0.67	0.08	2.1	0.045	19	48	0.9949	3.49	0.49	11.4	1	5.471700191	6	0.5282998085
5	6.8	0.18	0.37	1.6	0.055	47	154	0.9934	3.08	0.45	9.1	0	6.093779087	5	-1.093779087
6	6.9	0.41	0.33	10.1	0.043	28	152	0.9968	3.2	0.52	9.4	0	5.068320274	5	-0.06832027435
7	5.9	0.32	0.33	2.1	0.027	35	138	0.98945	3.37	0.42	12.7	0	6.789160728	6	-0.7891607285
8	6	0.24	0.41	1.3	0.036	42	118	0.99018	3.04	0.64	11.7333	0	6.101869106	6	-0.1018691063
9	7.3	0.48	0.32	2.1	0.062	31	54	0.99728	3.3	0.65	10	1	5.896887779	7	1.103112221
10	7.4	0.24	0.22	10.7	0.042	26	81	0.9954	2.86	0.36	9.7	0	6.140838623	6	-0.140838623

Once all the regression models have been completed it's time to start testing out the classification models and determining how they perform with the respected data. Before starting to develop the model a random classifier will be implemented to determine what accuracy is to obtain a baseline and determine the effectiveness of the classification models.

III.0 Random Classifier (Estimating)

Using the random classifier presented in figure 7 produces an output of around .31641, which means that this random classifier's accuracy is about 31%.

```
y_val_copy = copy.copy(y_val)
np.random.shuffle(y_val)
hits_array = np.array(y_val) == np.array(y_val_copy)
print("Accuracy using random classifier ",hits_array.mean())
```

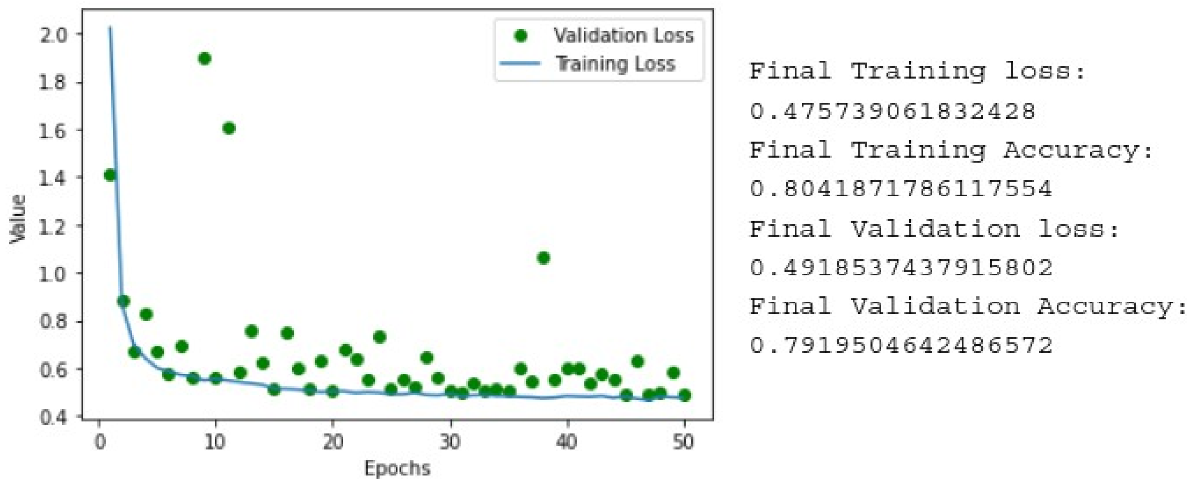
Figure 7: Random Classifier

III.1 Classification model 1(clasmodl1)

The first classification model that was built consists of a two layer network, whereas layer one has eight processing elements and the output layer has four processing elements. In addition the activation function that will be used in the output layer will be 'softmax' because this activation is particularly good when it comes to classification models. In addition, the optimizer that is being used is 'rmsprop', with the loss being 'categorical_crossentropy' and the metric that will be used 'accuracy'. The figure below shows the structure of the first classification model. The model was built and it was trained for a total of 50 epochs The batch size that was used to train the model was of size 64. The image shown in figure 9 shows the validation vs training loss, as well as the final values pertaining to loss and accuracy..

```
def build_clasmodl1():
  clasmodl1 = keras.Sequential(
        [
          layers.Dense(8, activation = 'relu'),
          layers.Dense(4, activation = 'softmax')
        ]
)
  clasmodl1.compile(optimizer = "rmsprop", loss =
"categorical_crossentropy", metrics = ["accuracy"])
  return clasmodl1
```

Figure 8: Classification Model 1 structure and parameters



```
Final Training loss:
0.475739061832428
Final Training Accuracy:
0.8041871786117554
Final Validation loss:
0.4918537437915802
Final Validation Accuracy:
0.7919504642486572
```

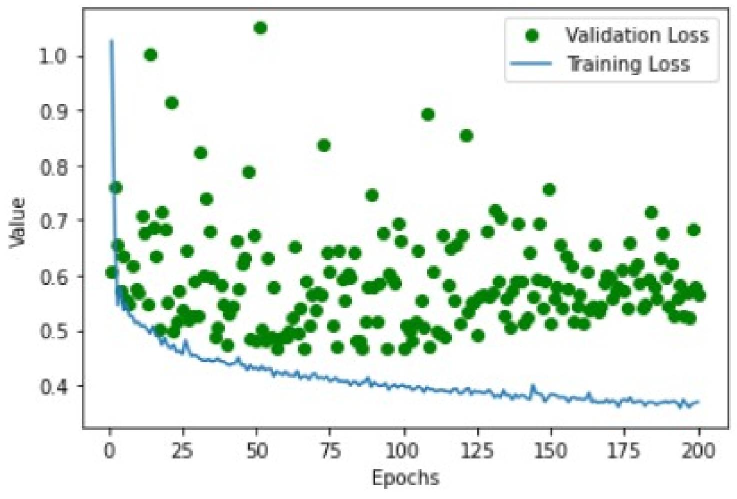
Figure 9: Classification Model 1 MSE vs Epochs

III.2 Classification model 2(clasmodl2)

The second classification model that was built consists of four hidden layers and an output layer. The four hidden layers are composed of 512, 100, 32, 100 processing elements respectively. In the figure below we can see the infrastructure of this model. The model was built and it was trained for a total of 200 epochs The batch size that was used to train the model was of size 128. The image shown in figure 11 shows the validation vs training loss, as well as the final values pertaining to loss and accuracy.

```
def build_clasmodl2():
 clasmodl2 = keras.Sequential(
     [
       layers.Dense(512, activation = 'relu'),
       layers.Dense(100, activation = 'relu'),
       layers.Dense(30, activation = 'relu'),
       layers.Dense(100, activation = 'relu'),
       layers.Dense(4, activation = 'softmax')
     ]
)
 clasmodl2.compile(optimizer = "rmsprop", loss = 
"categorical_crossentropy", metrics = ["accuracy"])
 return clasmodl2
```

Figure 10: Classification Model 2 structure and parameters



```
Final Training loss:
0.3695135712623596
Final Training Accuracy:
0.8362069129943848
Final Validation loss:
0.5642444491386414
Final Validation Accuracy:
0.8229101896286011
```

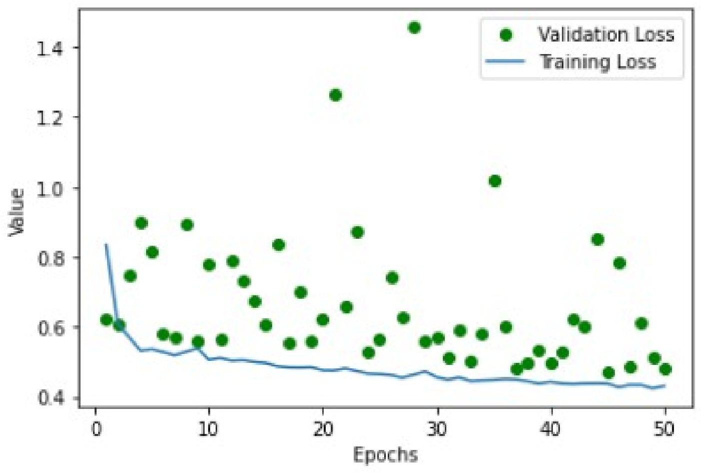
Figure 11: Classification Model 2 MSE vs Epochs

III.3 Classification model 3 (clasmodl3)

The final classification model structured from classification model 2 was used. This means that the number of layers and processing elements corresponding to each layer is the same. The figure below shows the architecture of the final model. The model was built and it was trained for a total of 200 epochs The batch size that was used to train the model was of size 128. The image shown in figure 11 shows the validation vs training loss, as well as the final values pertaining to loss and accuracy.

```
def build_clasmodl3():
 clasmodl3 = keras.Sequential(
     [
       layers.Dense(512, activation = 'relu'),
       layers.Dense(100, activation = 'relu'),
       layers.Dense(30, activation = 'relu'),
       layers.Dense(100, activation = 'relu'),
       layers.Dense(4, activation = 'softmax')
     ]
)
 clasmodl3.compile(optimizer = "rmsprop", loss =
"categorical_crossentropy", metrics = ["accuracy"])
 return clasmodl3
```

Figure 12: Classification Model 3 structure and parameters



```
Final Training loss:
0.4316848814487457
Final Training Accuracy:
0.8128078579902649
Final Validation loss:
0.4793102443218231
Final Validation Accuracy:
0.7882353067398071
```

Figure 13: Classification Model 2 MSE vs Epochs

Once the training and validation of the best classification model (clasmodl3) is done, it's time to test how this model does with the test set. In the table below we can see what the classification model predicted and the accuracy with respect to its target. In the table below we can see that most of the predicted values hit their corresponding target. In this specific model the only targets that were not hit by the classification model were pattern number 3 and 9 which means that the total accuracy of the classification model 3 is eight out of ten (80 %).

Table II. Comparison of Activation vs Target

Pattern #	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	activation	target	accuracy
1	7.4	0.44	0.2	11.5	0.049	44	157	0.998	3.27	0.44	9	0	[1.0848407e-03 9.5730478e-01 4.1609883e-02 6.2462721e-07]	[0. 1. 0. 0.]	Hit
2	6.5	0.23	0.36	16.3	0.038	43	133	0.99924	3.26	0.41	8.8	0	[6.78680908e-06 9.80151594e-01 1.98415183e-02 1.14908275e-07]	[0. 1. 0. 0.]	Hit
3	5.6	0.41	0.24	1.9	0.034	10	53	0.98815	3.32	0.5	13.5	0	[0.00163119 0.5857639 0.4092513 0.00335363]	[0. 0. 1. 0.]	Missed
4	6.4	0.67	0.08	2.1	0.045	19	48	0.9949	3.49	0.49	11.4	1	[1.0622812e-03 9.3931246e-01 5.9619188e-02 6.0928160e-06]	[0. 1. 0. 0.]	Hit
5	6.8	0.18	0.37	1.6	0.055	47	154	0.9934	3.08	0.45	9.1	0	[4.3470034e-04 9.1502351e-01 8.4541321e-02 4.3267636e-07]	[0. 1. 0. 0.]	Hit
6	6.9	0.41	0.33	10.1	0.043	28	152	0.9968	3.2	0.52	9.4	0	[3.7617211e-03	[0. 1. 0. 0.]	Hit

													9.0640402e-01 8.9759447e-02 7.4819094e-05]		
7	5.9	0.32	0.33	2.1	0.027	35	138	0.98945	3.37	0.42	12.7	0	[1.8569552e-04 6.2310886e-01 3.7645802e-01 2.4740153e-04]	[0. 1. 0. 0.]	Hit
8	6	0.24	0.41	1.3	0.036	42	118	0.99018	3.04	0.64	11.733	0	[0.00060205 0.5680213 0.42969406 0.00168259]	[0. 1. 0. 0.]	Hit
9	7.3	0.48	0.32	2.1	0.062	31	54	0.99728	3.3	0.65	10	1	[2.6267761e-04 8.3705467e-01 1.6262574e-01 5.6991110e-05]	[0. 0. 1. 0.]	Missed
10	7.4	0.24	0.22	10.7	0.042	26	81	0.9954	2.86	0.36	9.7	0	[0.00230696 0.86245453 0.13375148 0.00148711]	[0. 1. 0. 0.]	Hit

Conclusion:

In conclusion the regression and classification models have been developed, trained and tested to determine their performance with the test set. For the most part we saw that the models behaved as expected, for example we saw that the regression model managed to be able to predict with a reasonable error for most patterns and only a few of them were completely missed. In addition, the classification model was able to predict eight out of the 10 patterns as hit's which is better than then random classifier. Ways to improve the models could be to try a different optimizer, for example in regards to the regression model using the 'Adam' optimizer since it uses stochastic gradient descent and it is believed to be better for regression models. The classification model can be improved by maybe using a different learning rate or using weight initialization. I also believe that additional pre-processing of the data would be beneficial for the models. Lastly, we learned about the activation functions, loss function, defining a model, effects of batch_size among other things.

Appendix:

```
#@title Data as Numpy Arrays
import matplotlib.pyplot as plt
import numpy as np
# This time we need to also import pandas
import pandas as pd
from io import StringIO

# Read in white wine data
# Uses PANDAS (pd) to create a PANDAS DataFrame Object:
white = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
winequality-white.csv", sep = ';')

# Read in red wine data
# Uses PANDAS (pd) to create a PANDAS DataFrame Object:
red = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-
red.csv", sep =';')

red['type'] = 1
white['type'] = 0

wines = red.append(white, ignore_index = True)

# Import SKLEARN
import sklearn

# Import `train_test_split` from `sklearn.model_selection`
from sklearn.model_selection import train_test_split
# Specify the data -
X1 = wines.iloc[:, 0:11]
X2 = wines.iloc[:,12]
X = pd.concat([X1,X2],axis = 1)

y = np.ravel(wines.quality)

# Splitting the data set for training and validating - Done with SKLEARN
X_train, X_valid, y_train, y_valid = train_test_split(X,y, test_size = 0.25, random_state = 45)

# Converting X_train & X_test DataFrame s to TF sensors
# Will use NumPy, TF, & Keras after this
# import tensorflow as tf

Xtrain = X_train.to_numpy()
X_valid = X_valid.to_numpy()

X_valid
# In reality:
# [1] ALL THE Xtrain patterns (with their y_train targets)
# will be used for TRAINING ([TR]), as Xtrain & y_train
# [2] MOST OF THE X_valid patterns (and their y_valid targets)
# will be used for VALIDATION ([TT]), as X_val & y_val
# BUT WE WILL SET ASIDE THE LAST 10 for "testing" ([TS])
# as X_tst & y_tst

# Retain the first 1615 patterns for validation ([TT])
Xval = X_valid[:1615]
Xval.shape

# and now set aside the last 10 for test
Xtst = X_valid[1615:]
Xtst.shape
# Setting Aside the Patterns to Normalize them
```

```
import copy
Xtrain_N = copy.copy(Xtrain)
Xval_N = copy.copy(Xval)
Xtst_N = copy.copy(Xtst)
mean = Xtrain_N.mean(axis=0)
Xtrain_N -= mean
std = Xtrain_N.std(axis=0)
Xtrain_N /= std
Xval_N -= mean
Xval_N /= std
Xtst_N -=mean
Xtst_N /=std

# Same for the corresponding targets
# Retain the first 1615 for validation ([TT])
y_val = y_valid[:1615]
y_val.shape

y_tst = y_valid[1615:]
y_tst.shape
y_tst

# Now, in addition, create the targets as one-hot-encoded 4 quality levels
# We will track these few targets through the conversion process
y_train[272:283]
```

```
from keras.engine.input_layer import Input
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def build_regmodl1():
 regmodl1 = keras.Sequential(
     [
       layers.Dense(8, activation = 'relu'),
       layers.Dense(1)
     ]
)
 regmodl1.compile(optimizer = "rmsprop", loss = "mse", metrics = ["mae"])
 return regmodl1

regmodl1 = build_regmodl1()
history_regmodl1 = regmodl1.fit(x = Xtrain_N,y = y_train, batch_size = 32, epochs = 50, verbose = 2,
validation_data = (Xval_N,y_val), validation_freq = 1)
```

```
# Plot the validation and training loss
plt.plot(range(1, len(history_regmodl1.history['val_loss']) + 1),
history_regmodl1.history['val_loss'], 'go', label = "Validation Loss")
plt.plot(range(1, len(history_regmodl1.history['loss']) + 1), history_regmodl1.history['loss'],label
= "Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Value")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history_regmodl1.history['loss'][-1],"\nFinal Training MAE: ",
history_regmodl1.history['mae'][-1])
print("Final Validation loss: ",history_regmodl1.history['val_loss'][-1],"\nFinal Validation MAE: ",
history_regmodl1.history['val_mae'][-1])


from keras.engine.input_layer import Input
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def build_regmodl2():
 regmodl2 = keras.Sequential(
     [
       layers.Dense(100, activation = 'relu'),
       layers.Dense(50, activation = 'relu'),
       layers.Dense(80, activation = 'relu'),
       layers.Dense(20, activation='relu'),
       layers.Dense(1)
     ]
 )
 regmodl2.compile(optimizer = "rmsprop", loss = "mse", metrics = ["mae"])
 return regmodl2

regmodl2 = build_regmodl2()
history_regmodl2 = regmodl2.fit(x = Xtrain_N,y = y_train, batch_size = 64, epochs = 150, verbose = 2,
validation_data = (Xval_N,y_val), validation_freq = 1)


# Plot the validation and training loss
plt.plot(range(1, len(history_regmodl2.history['val_loss']) + 1),
history_regmodl2.history['val_loss'], 'go',label = "Validation Loss")
plt.plot(range(1, len(history_regmodl2.history['loss']) + 1), history_regmodl2.history['loss'], label
= "Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Value")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history_regmodl2.history['loss'][-1],"\nFinal Training MAE: ",
history_regmodl2.history['mae'][-1])
print("Final Validation loss: ",history_regmodl2.history['val_loss'][-1],"\nFinal Validation MAE: ",
history_regmodl2.history['val_mae'][-1])


from keras.engine.input_layer import Input
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def build_regmodl3():
 regmodl3 = keras.Sequential(
     [
       layers.Dense(100, activation = 'relu'),
       layers.Dense(50, activation = 'relu'),
       layers.Dense(80, activation = 'relu'),
       layers.Dense(20, activation='relu'),
       layers.Dense(1)
     ]
 )
 regmodl3.compile(optimizer = "rmsprop", loss = "mse", metrics = ["mae"])
 return regmodl3
```

```
regmodl3 = build_regmodl3()
history_regmodl3 = regmodl3.fit(x = Xtrain_N,y = y_train, batch_size = 128, epochs = 50, verbose = 2,
validation_data = (Xval_N,y_val), validation_freq = 1)



# Plot the validation and training loss
plt.plot(range(1, len(history_regmodl3.history['val_loss']) + 1),
history_regmodl3.history['val_loss'], 'go',label = "Validation Loss")
plt.plot(range(1, len(history_regmodl3.history['loss']) + 1), history_regmodl3.history['loss'],label
= "Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Value")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history_regmodl3.history['loss'][-1],"\nFinal Training MAE: ",
history_regmodl3.history['mae'][-1])
print("Final Validation loss: ",history_regmodl3.history['val_loss'][-1],"\nFinal Validation MAE: ",
history_regmodl3.history['val_mae'][-1])


predict_labels = regmodl3.predict(Xtst_N)
predict_labels_T=predict_labels.T
error = y_tst - predict_labels_T
predict_table = pd.DataFrame(Xtst)
predict_table['activation'] = predict_labels
predict_table['target'] = y_tst
predict_table['error'] = error.T
# Renaming the columns of the table
predict_table.columns.values[0] = "f1"
predict_table.columns.values[1] = "f2"
predict_table.columns.values[2] = "f3"
predict_table.columns.values[3] = "f4"
predict_table.columns.values[4] = "f5"
predict_table.columns.values[5] = "f6"
predict_table.columns.values[6] = "f7"
predict_table.columns.values[7] = "f8"
predict_table.columns.values[8] = "f9"
predict_table.columns.values[9] = "f10"
predict_table.columns.values[10] = "f11"
predict_table.columns.values[11] = "f12"
display(predict_table)


# Function create rank-1 arrays where 3,4,5,6,7,8,9 are mapped to 1 or 2 or 3 or 4
def to_4cs(x):
 lx = len(x)
 results = np.zeros(lx)
 for i in range(lx):
   # print("start")
   xa = x[i];
   if xa <= 3:
     results[i] = 1
   elif xa <= 6:
     results[i] = 2
   elif xa <= 8:
     results[i] = 3
   else:
     results[i] = 4
   # results [i, label] = 1
 results = results.astype(int)
 return results
```

```
train_labels = to_4cs(y_train)
val_labels = to_4cs(y_val)
tst_labels = to_4cs(y_tst)

# Let's verify that the training targets that we are tracking
# were converted to levels (1 = BAD; 2 = Medium; 3 = GOOD; 4- Excellent) correctly:
train_labels[272:283]

# Now, one shot encoding of all 3 target arrays
# define a function to do the

def to_one_hot(labels, dimension = 4):
 results = np.zeros((len(labels), dimension))
 for i, label in enumerate(labels-1):
   results[i, label] = 1.
 return results

one_hot_train_labels = to_one_hot(train_labels)
one_hot_val_labels = to_one_hot(val_labels)
one_hot_tst_labels = to_one_hot(tst_labels)


import copy
y_val_copy = copy.copy(y_val)
np.random.shuffle(y_val)
hits_array = np.array(y_val) == np.array(y_val_copy)
print("Accuracy using random classifier ",hits_array.mean())



from keras.engine.input_layer import Input
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def build_clasmodl1():
 clasmodl1 = keras.Sequential(
     [
       layers.Dense(8, activation = 'relu'),
       layers.Dense(4, activation = 'softmax')
     ]
)
 clasmodl1.compile(optimizer = "rmsprop", loss = "categorical_crossentropy", metrics = ["accuracy"])
 return clasmodl1

clasmodl1 = build_clasmodl1()
history_clasmodl1 = clasmodl1.fit(x = Xtrain,y = one_hot_train_labels, batch_size = 32, epochs = 50,
verbose = 2, validation_data = (Xval,one_hot_val_labels), validation_freq = 1)




# Plot the validation and training loss
plt.plot(range(1, len(history_clasmodl1.history['val_loss']) + 1),
history_clasmodl1.history['val_loss'], 'go', label = "Validation Loss")
plt.plot(range(1, len(history_clasmodl1.history['loss']) + 1),
history_clasmodl1.history['loss'],label = "Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Value")
plt.legend()
plt.show()
# Final Values
```

```
print("Final Training loss: ",history_clasmodl1.history['loss'][-1],"\nFinal Training Accuracy: ",
history_clasmodl1.history['accuracy'][-1])
print("Final Validation loss: ",history_clasmodl1.history['val_loss'][-1],"\nFinal Validation
Accuracy: ", history_clasmodl1.history['val_accuracy'][-1])


from keras.engine.input_layer import Input
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def build_clasmodl2():
 clasmodl2 = keras.Sequential(
     [
       layers.Dense(512, activation = 'relu'),
       layers.Dense(100, activation = 'relu'),
       layers.Dense(30, activation = 'relu'),
       layers.Dense(100, activation = 'relu'),
       layers.Dense(4, activation = 'softmax')
     ]
)
 clasmodl2.compile(optimizer = "rmsprop", loss = "categorical_crossentropy", metrics = ["accuracy"])
 return clasmodl2

clasmodl2 = build_clasmodl2()
history_clasmodl2 = clasmodl2.fit(x = Xtrain,y = one_hot_train_labels, batch_size = 128, epochs =
200, verbose = 2, validation_data = (Xval,one_hot_val_labels), validation_freq = 1)


# Plot the validation and training loss
plt.plot(range(1, len(history_clasmodl2.history['val_loss']) + 1),
history_clasmodl2.history['val_loss'], 'go', label = "Validation Loss")
plt.plot(range(1, len(history_clasmodl2.history['loss']) + 1),
history_clasmodl2.history['loss'],label = "Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Value")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history_clasmodl2.history['loss'][-1],"\nFinal Training Accuracy: ",
history_clasmodl2.history['accuracy'][-1])
print("Final Validation loss: ",history_clasmodl2.history['val_loss'][-1],"\nFinal Validation
Accuracy: ", history_clasmodl2.history['val_accuracy'][-1])


from keras.engine.input_layer import Input
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

def build_clasmodl3():
 clasmodl3 = keras.Sequential(
     [
       layers.Dense(512, activation = 'relu'),
       layers.Dense(100, activation = 'relu'),
       layers.Dense(30, activation = 'relu'),
       layers.Dense(100, activation = 'relu'),
       layers.Dense(4, activation = 'softmax')
     ]
)
 clasmodl3.compile(optimizer = "rmsprop", loss = "categorical_crossentropy", metrics = ["accuracy"])
 return clasmodl3
```

```
clasmodl3 = build_clasmodl3()
history_clasmodl3 = clasmodl3.fit(x = Xtrain,y = one_hot_train_labels, batch_size = 128, epochs = 50,
verbose = 2, validation_data = (Xval,one_hot_val_labels), validation_freq = 1)
clasmodl3.save('my_clasmodl3.h5')


# Plot the validation and training loss
plt.plot(range(1, len(history_clasmodl3.history['val_loss']) + 1),
history_clasmodl3.history['val_loss'], 'go', label = "Validation Loss")
plt.plot(range(1, len(history_clasmodl3.history['loss']) + 1),
history_clasmodl3.history['loss'],label = "Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Value")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history_clasmodl3.history['loss'][-1],"\nFinal Training Accuracy: ",
history_clasmodl3.history['accuracy'][-1])
print("Final Validation loss: ",history_clasmodl3.history['val_loss'][-1],"\nFinal Validation
Accuracy: ", history_clasmodl3.history['val_accuracy'][-1])


# Predicting the activation and inputting it alongside the targets
# into the predict_table
from pandas.core.arrays.sparse import dtype
predict_labels = clasmodl3.predict(Xtst)
predict_table = pd.DataFrame(Xtst)
predict_table["activation"]=[predict_labels[x,:] for x in range(len(predict_labels))]
predict_table["target"]=[one_hot_tst_labels[x,:] for x in range(len(one_hot_tst_labels))]
predict_labels = np.around(predict_labels)
# Checking which activations match their target i.e accuracy
check = np.empty((10,4), dtype=bool)
hits = []
for i in range(len(predict_labels)):
 check[i,:] = predict_labels[i,:] == one_hot_tst_labels[i,:]
 if (np.count_nonzero(check[i,:]) == 4):
   hits.append("Hit")
 else:
   hits.append("Missed")
predict_table["accuracy"] = hits

# Renaming the columns of the table
predict_table.columns.values[0] = "f1"
predict_table.columns.values[1] = "f2"
predict_table.columns.values[2] = "f3"
predict_table.columns.values[3] = "f4"
predict_table.columns.values[4] = "f5"
predict_table.columns.values[5] = "f6"
predict_table.columns.values[6] = "f7"
predict_table.columns.values[7] = "f8"
predict_table.columns.values[8] = "f9"
predict_table.columns.values[9] = "f10"
predict_table.columns.values[10] = "f11"
predict_table.columns.values[11] = "f12"
display(predict_table)
```