Model Definition and Training

In this notebook the data is put into a keras deep learning model. The goal is to find an effective model for testing the recovery/broken state of the water pump. In order to refine the deep learning model, RandomizedSearchCV is used and epochs, batch_size, learning_rate, and dropout_rate is tested. Another RandomizedSearchCV is not run using the PCA refined data from the Feature Creation Notebook due to time restraints. A new model is trained using the PCA data and its F1-Score and Matthews Correlation Coefficient is found.

The following hidden cell reads in the data from the Feature Creation Notebook and puts it into a DataFrame. The second hidden cell reads in the PCA data from the second part of the Feature Creation Notebook.

In [1]:
```
#All of the imports required for the model
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import f1_score
from sklearn.metrics import matthews_corrcoef
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
from sklearn.model_selection import train_test_split
import os, types
import pandas as pd
pd.set_option('display.max_columns', None)
from botocore.client import Config
import ibm_boto3
import matplotlib.pyplot as plt
```

In [2]:
```
# The code was removed by Watson Studio for sharing.
```

Out[2]:

	Id	timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_05	sensor_06	sensor_07	sensor_08	sensor_09	sensor_10	s
0	0	2018-04-01 00:00:00	0.231450	-0.151675	0.639386	1.057675	0.303443	0.177097	-0.042091	0.132586	0.181964	0.122858	-0.350860	
1	1	2018-04-01 00:01:00	0.231450	-0.151675	0.639386	1.057675	0.303443	0.177097	-0.042091	0.132586	0.181964	0.122858	-0.350860	
2	2	2018-04-01 00:02:00	0.180129	-0.072613	0.639386	1.093565	0.334786	0.008647	-0.082656	0.089329	0.207112	0.101892	-0.297906	
3	3	2018-04-01 00:03:00	0.219228	-0.151675	0.627550	1.093564	0.260045	0.207693	-0.086035	0.185835	0.246628	0.136839	-0.239029	
4	4	2018-04-01 00:04:00	0.182573	-0.138499	0.639386	1.093564	0.317909	0.184568	-0.069133	0.169195	0.246628	0.136839	-0.163810	

In [3]:
```
# The code was removed by Watson Studio for sharing.
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	
0	-0.010524	0.776836	-0.522604	-0.782975	-1.943597	1.535913	-1.132301	-0.175648	-1.079528	-0.922840	-0.261418	-0.087582	0.276406	-0.1
1	-0.010524	0.776836	-0.522604	-0.782975	-1.943597	1.535913	-1.132301	-0.175648	-1.079528	-0.922840	-0.261418	-0.087582	0.276406	-0.1
2	-0.151425	0.782444	-0.493743	-0.843039	-2.105161	1.502969	-1.128054	-0.031836	-0.955613	-0.787217	-0.285881	-0.095448	0.120880	-0.1
3	-0.151886	0.816479	-0.541572	-0.971684	-2.012970	1.586361	-1.169678	-0.009723	-0.820260	-0.656993	-0.279351	-0.077121	0.195787	-0.2
4	-0.106250	0.929110	-0.407707	-1.025532	-1.989539	1.511101	-1.171104	-0.146129	-1.093221	-0.809584	-0.145487	-0.011419	0.105777	-0.1

Split the data

In order to prevent overfitting in the model, the data set is split into train and test sets.

In [4]:
```
#Regular train_test_split
X = data.loc[:,'sensor_00':'sensor_51']
y = data.machine_status

#Healthy Split Data for training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

In [5]:
```
#PCA train_test_split
X_PCA = PCA_data.loc[:,'0':'24']
y_PCA = PCA_data.machine_status


#Healthy PCA Split Data for training
PCA_X_train, PCA_X_test, PCA_y_train, PCA_y_test = train_test_split(X_PCA, y_PCA, test_size = 0.25)
```

In [6]:
```
#Printouts to show training split and n_inputs for the build_model function
print(X.shape)
n_input = X.shape[1]
n_input2 = PCA_X_train.shape[1]
```

```
(220320, 51)
```

Creating the Model and the Hyperparamater Tuner

The following code block creates a function that will allow for a new model to be produced for each new hyperparameter test. The four hyperparameters tested are epochs, batch size, Adam optimizer learning rate, and dropout rate for the Dropout layer. The final block in this section will allow for the Keras model to work with the RandomSearchCV and creates an early stop for the model to reduce computation time and to help the model fit better to the data.

In [7]:
```
#This function allows the Keras Model to be built for each RandomizedSearchCV iteration
def build_model(n_neurons = 64, dropout_rate = 0.2, lrat = 0.01, n_input = 10):
    model = Sequential()

    model.add(InputLayer(input_shape = (n_input,)))
    model.add(Dense(n_neurons))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation = 'sigmoid'))

    opt = Adam(learning_rate = lrat)

    model.compile(loss='mse', metrics=['accuracy'], optimizer=opt)
    return model
```

In [8]:
```
#All of the different hyperparameters to be tested with the RandomizedSearchCV
epochs = tuple(int(x) for x in np.linspace(5, 100, num = 12))
batch_size = (32, 64, 72, 128, 256)
lr = (0.1, 0.05, 0.01, 0.005, 0.001)
dr = (0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.6)
```

In [9]:
```
#The hyperparameters that are trained put into a dict for the RandomizedSearchCV method
param_grid = {'epochs': epochs, 'batch_size': batch_size, 'lrat': lr, 'dropout_rate': dr}
```

In [10]:
```
#KerasClassifier wrapper for the RandomizedSearchCV to work
pump_model = KerasClassifier(build_fn = build_model, n_input = n_input)

#EarlyStopping used because if the model converges fast then the training will stop and move on to the next mod
stop = EarlyStopping(monitor='loss', patience=5)
```

Training the Model

The following model is trained using the 51 sensor readings from the Feature Creation Notebook.

In [11]:
```
#Creation of the RandomizedSearchCV object and the fit method.
keras_rands = RandomizedSearchCV(pump_model, param_distributions = param_grid, n_iter = 15,verbose = 0, refit=T
keras_rands.fit(X_train, y_train, callbacks = [stop], verbose=0)
```

```
1033/1033 [==============================] - 1s 789us/step - loss: 0.0082 - accuracy: 0.99180s - loss: 0.0082 -
accuracy
1033/1033 [==============================] - 1s 803us/step - loss: 0.0065 - accuracy: 0.9935
1033/1033 [==============================] - 1s 802us/step - loss: 0.0055 - accuracy: 0.9945
1033/1033 [==============================] - 1s 765us/step - loss: 0.0054 - accuracy: 0.9946
1033/1033 [==============================] - 1s 767us/step - loss: 0.0043 - accuracy: 0.9957
517/517 [==============================] - 0s 798us/step - loss: 0.0084 - accuracy: 0.9916
517/517 [==============================] - 1s 816us/step - loss: 0.0058 - accuracy: 0.9942
517/517 [==============================] - 1s 806us/step - loss: 0.0064 - accuracy: 0.9936
517/517 [==============================] - 1s 819us/step - loss: 0.0058 - accuracy: 0.9942
517/517 [==============================] - 0s 800us/step - loss: 0.0044 - accuracy: 0.9955
259/259 [==============================] - 0s 825us/step - loss: 0.0035 - accuracy: 0.9963
259/259 [==============================] - 0s 830us/step - loss: 0.0041 - accuracy: 0.9958
259/259 [==============================] - 0s 767us/step - loss: 0.0037 - accuracy: 0.9962
259/259 [==============================] - 0s 780us/step - loss: 0.0047 - accuracy: 0.9952
259/259 [==============================] - 0s 826us/step - loss: 0.0047 - accuracy: 0.9952
130/130 [==============================] - 0s 1ms/step - loss: 0.0044 - accuracy: 0.9956
130/130 [==============================] - 0s 966us/step - loss: 0.0057 - accuracy: 0.9943
130/130 [==============================] - 0s 1ms/step - loss: 0.0042 - accuracy: 0.9957
130/130 [==============================] - 0s 1ms/step - loss: 0.0038 - accuracy: 0.9961
130/130 [==============================] - 0s 952us/step - loss: 0.0035 - accuracy: 0.9964
517/517 [==============================] - 1s 831us/step - loss: 0.0053 - accuracy: 0.9947
517/517 [==============================] - ETA: 0s - loss: 0.0062 - accuracy: 0.99 - 0s 779us/step - loss: 0.00
61 - accuracy: 0.9939
517/517 [==============================] - 0s 790us/step - loss: 0.0048 - accuracy: 0.9952
517/517 [==============================] - 0s 807us/step - loss: 0.0061 - accuracy: 0.9938
517/517 [==============================] - 1s 820us/step - loss: 0.0048 - accuracy: 0.9952
130/130 [==============================] - 0s 997us/step - loss: 0.0014 - accuracy: 0.9983
130/130 [==============================] - 1s 1ms/step - loss: 0.0017 - accuracy: 0.9978
130/130 [==============================] - 0s 1ms/step - loss: 0.0016 - accuracy: 0.9978
130/130 [==============================] - 0s 1ms/step - loss: 0.0016 - accuracy: 0.9979
130/130 [==============================] - 0s 1ms/step - loss: 0.0013 - accuracy: 0.9984
1033/1033 [==============================] - 1s 750us/step - loss: 0.0192 - accuracy: 0.9808
1033/1033 [==============================] - 1s 764us/step - loss: 0.0199 - accuracy: 0.9801
1033/1033 [==============================] - 1s 845us/step - loss: 0.0179 - accuracy: 0.9821
1033/1033 [==============================] - 1s 775us/step - loss: 0.0175 - accuracy: 0.9825
1033/1033 [==============================] - 1s 767us/step - loss: 0.0211 - accuracy: 0.9789
259/259 [==============================] - 0s 785us/step - loss: 0.0149 - accuracy: 0.9851
259/259 [==============================] - 0s 859us/step - loss: 0.0199 - accuracy: 0.9801
259/259 [==============================] - 0s 808us/step - loss: 0.0133 - accuracy: 0.9867
259/259 [==============================] - 0s 840us/step - loss: 0.0136 - accuracy: 0.9864
259/259 [==============================] - 0s 839us/step - loss: 0.1228 - accuracy: 0.8771
259/259 [==============================] - 0s 840us/step - loss: 0.0015 - accuracy: 0.9979
259/259 [==============================] - 0s 831us/step - loss: 0.0015 - accuracy: 0.9981
259/259 [==============================] - 0s 767us/step - loss: 0.0017 - accuracy: 0.9978
259/259 [==============================] - 0s 805us/step - loss: 0.0015 - accuracy: 0.9980
259/259 [==============================] - 0s 827us/step - loss: 0.0013 - accuracy: 0.9985
1033/1033 [==============================] - 1s 758us/step - loss: 0.0047 - accuracy: 0.9953
1033/1033 [==============================] - 1s 803us/step - loss: 0.0052 - accuracy: 0.9947
1033/1033 [==============================] - 1s 802us/step - loss: 0.0037 - accuracy: 0.9962
1033/1033 [==============================] - 1s 842us/step - loss: 0.0062 - accuracy: 0.9938
1033/1033 [==============================] - 1s 821us/step - loss: 0.0044 - accuracy: 0.9954
459/459 [==============================] - 0s 780us/step - loss: 0.0014 - accuracy: 0.9983
459/459 [==============================] - 0s 756us/step - loss: 0.0014 - accuracy: 0.9983
459/459 [==============================] - 0s 749us/step - loss: 0.0017 - accuracy: 0.9979
459/459 [==============================] - 0s 731us/step - loss: 0.0014 - accuracy: 0.9981
459/459 [==============================] - 0s 752us/step - loss: 0.0011 - accuracy: 0.9988
130/130 [==============================] - 0s 1ms/step - loss: 0.0019 - accuracy: 0.9976
130/130 [==============================] - 0s 975us/step - loss: 0.0018 - accuracy: 0.9979
130/130 [==============================] - 0s 1ms/step - loss: 0.0024 - accuracy: 0.9973
130/130 [==============================] - 0s 953us/step - loss: 0.0022 - accuracy: 0.9974
130/130 [==============================] - 0s 955us/step - loss: 0.0020 - accuracy: 0.9975
259/259 [==============================] - 0s 813us/step - loss: 0.0141 - accuracy: 0.9859
259/259 [==============================] - 0s 767us/step - loss: 0.0098 - accuracy: 0.9902
259/259 [==============================] - 0s 808us/step - loss: 0.0160 - accuracy: 0.9840
259/259 [==============================] - 0s 794us/step - loss: 0.0179 - accuracy: 0.9821
259/259 [==============================] - 0s 812us/step - loss: 0.0145 - accuracy: 0.9855
1033/1033 [==============================] - 1s 808us/step - loss: 0.0349 - accuracy: 0.9651
1033/1033 [==============================] - 1s 814us/step - loss: 0.0241 - accuracy: 0.9759
1033/1033 [==============================] - 1s 735us/step - loss: 0.1204 - accuracy: 0.8796
1033/1033 [==============================] - 1s 821us/step - loss: 0.0206 - accuracy: 0.9794
1033/1033 [==============================] - 1s 790us/step - loss: 0.0899 - accuracy: 0.9101
259/259 [==============================] - 0s 884us/step - loss: 0.0041 - accuracy: 0.9958
259/259 [==============================] - 0s 820us/step - loss: 0.0054 - accuracy: 0.9944
259/259 [==============================] - 0s 788us/step - loss: 0.0045 - accuracy: 0.9955
259/259 [==============================] - 0s 824us/step - loss: 0.0048 - accuracy: 0.9951
259/259 [==============================] - 0s 763us/step - loss: 0.0029 - accuracy: 0.9971
```

Out[11]:

```
RandomizedSearchCV(estimator=<tensorflow.python.keras.wrappers.scikit_learn.KerasClassifier object at 0x7fbcdff
f5090>,
                   n_iter=15,
                   param_distributions={'batch_size': (32, 64, 72, 128, 256),
                                        'dropout_rate': (0.1, 0.15, 0.2, 0.25,
                                                         0.3, 0.4, 0.5, 0.6),
                                        'epochs': (5, 13, 22, 30, 39, 48, 56,
                                                   65, 74, 82, 91, 100),
                                        'lrat': (0.1, 0.05, 0.01, 0.005,
                                                 0.001)})
```

In [12]:
```
#In order to predict using the test data the final model is trained using the best parameters from the Randomiz
best = keras_rands.best_params_
best_model = build_model(lrat = best['lrat'], n_input = n_input, dropout_rate = best['dropout_rate'])
best_model_hist = best_model.fit(X_train, y_train, epochs = best['epochs'], batch_size = best['batch_size'], ca
```

In [13]:
```
#Showing off the best hyperparameters
best = keras_rands.best_params_

print('Best Hyperparameters:')
print('Epochs: {}'.format(best['epochs']))
print('Batch Size: {}'.format(best['batch_size']))
print('Learning Rate: {}'.format(best['lrat']))
print('Dropout Rate: {}'.format(best['dropout_rate']))
```

```
Best Hyperparameters:
Epochs: 82
Batch Size: 72
Learning Rate: 0.001
Dropout Rate: 0.25
```

PCA Model Training

To test one other iteration of the Feature Creation, PCA was done on the 51 sensor readings and 25 total new dimensions were made. The hope is that the time of the training will either be reduced or the final model will be more accurate. The original best params are used in order to reduce total training time.

In [14]:
```
#KerasClassifier wrapper for the RandomizedSearchCV to work
pump_model = KerasClassifier(build_fn = build_model, n_input = n_input2)

#EarlyStopping used because if the model converges fast then the training will stop and move on to the next mod
stop = EarlyStopping(monitor='loss', patience=5)
```

In [15]:
```
#In order to predict using the test data the final model is trained using the best parameters from the Randomiz
best_model2 = build_model(lrat = best['lrat'], n_input = n_input2, dropout_rate = best['dropout_rate'])
best_model2_hist = best_model2.fit(PCA_X_train, PCA_y_train, epochs = best['epochs'], batch_size = best['batch_
```

Predictions

The final predictions using the test data is found and the F1-Score and the Matthews Correlation Coefficient is calculated for each iteration type.

In [16]:
```
#Predictions of the data using the two types of test data
y_pred = (best_model.predict(X_test) > 0.5).astype('int32')

#PCA Predictions
y_pred_PCA = (best_model2.predict(PCA_X_test) > 0.5).astype('int32')
```

In [17]:
```
#The best model is tested using f1_score and matthews correlation coefficient
f1 = f1_score(y_test, y_pred)
phi = matthews_corrcoef(y_test, y_pred)

#The best model is tested using f1_score and matthews correlation coefficient
f1_PCA = f1_score(PCA_y_test, y_pred_PCA)
phi_PCA = matthews_corrcoef(PCA_y_test, y_pred_PCA)
```

In [18]:
```
#Printouts of the score
print('Original Iteration Scores:')
print('F1 Score: {}'.format(f1))
print('Matthews Corr Coeficient: {} \n'.format(phi))

print('PCA Iteration Scores: ')
print('F1 Score: {}'.format(f1_PCA))
print('Matthews Corr Coeficient: {} \n'.format(phi_PCA))
```

```
Original Iteration Scores:
F1 Score: 0.9988252769336816
Matthews Corr Coeficient: 0.9819256191529896

PCA Iteration Scores:
F1 Score: 0.9976837884656559
Matthews Corr Coeficient: 0.9657194530936922
```

Graphing Loss During Training

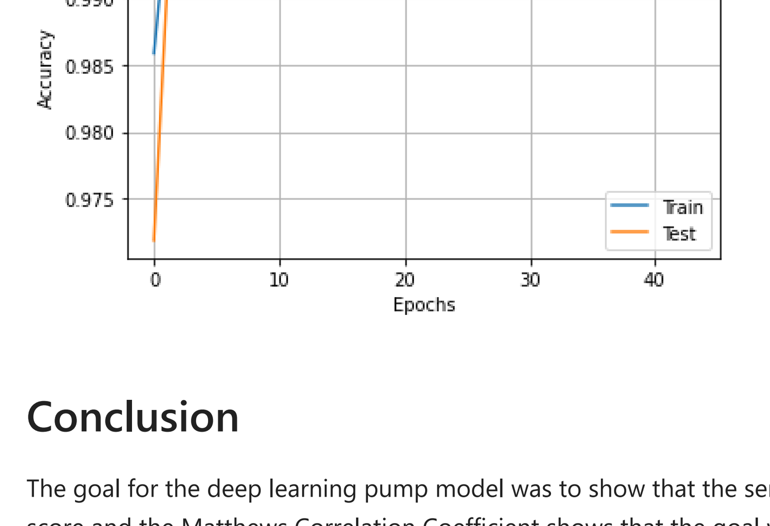
To check the progress of the training, plots of the loss over each epoch is made.

In [19]:
```
#Train with the test data and track the losses over epochs
test_model = build_model(lrat = best['lrat'], n_input = n_input, dropout_rate = best['dropout_rate'])
testHist = test_model.fit(X_test, y_test, epochs = best['epochs'], batch_size = best['batch_size'], callbacks =
```

In [20]:
```
#Losses during training model epochs
best_losses = best_model_hist.history['loss']
best_acc = best_model_hist.history['accuracy']

#Losses during test model epochs
test_losses = testHist.history['loss']
test_acc = testHist.history['accuracy']
```

In [21]:
```
#Plot of the losses over all epochs with train and test data in each line
%matplotlib inline

plt.plot(best_losses)
plt.plot(test_losses)
plt.title('Losses over Training Epochs')
plt.ylabel('Total Losses')
plt.xlabel('Epochs')
plt.grid()
plt.legend(['Train', 'Test'])
plt.show()
```

In [22]:
```
#Plot of the losses over all epochs with train and test data in each line
%matplotlib inline

plt.plot(best_acc)
plt.plot(test_acc)
plt.title('Accuracy over Training Epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.grid()
plt.legend(['Train', 'Test'])
plt.show()
```



Conclusion

The goal for the deep learning pump model was to show that the sensor data can be used to check the status of a water pump. The F1-score and the Matthews Correlation Coefficient shows that the goal was met. With both scores being around 0.99, the model seems to predict the pump accurately. The reason both scores are used in this case is that the F1-score is a well known accuracy measurement and the Matthews Correlation Coefficient also takes the true negatives into account. Taking true negatives into account is very important in this case as we want to be very accurate in predicting the broken state. The second iteration using PCA did not provide any benefit in this case. If a second hyperparameter tuning was run with the PCA data the accuracy may have been better. With further PCA reduction the model may have been able to train faster and be more accurate, but the current PCA was slower and about the same accuracy. To conclude, the model was accurate and did not take long to train, therefore our goal was met.