Open in Colab

Question No.2. Dataset: data1_0.5-1.csv Download data1_0.5-1.csv

Train two single/double hidden layer deep networks by varying the number of hidden nodes (4, 8, 12, 16) in each layer with 70% training and 30% validation data. Use appropriate learning rate, activation, and loss functions and also mention the reason for choosing the same. Report, compare, and explain the observed accuracy and minimum loss achieved. [0.5+1 mark]

In [2]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
#from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import accuracy_score, confusion_matrix #, precision_score, recall_score, f1_score, precis
import matplotlib.pyplot as plt
from tensorflow.keras import optimizers
from sklearn.decomposition import PCA
import seaborn as sns
#from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam

#google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
print(tf.__version__)
```

```
2.7.0
```

1. Load the attached csv file in python. Each row consists of feature 1, feature 2, feature 3 & class label.

In [16]:
```
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data1_0.5.csv')
data
```

Out[16]:

	0.045999539	-0.18176747	0.187497655	0	0.1
0	0.102303	0.116327	0.154913	0	0
1	-0.131546	-0.038680	0.137115	0	0
2	0.007224	-0.067146	0.067533	0	0
3	0.112290	0.040584	0.119399	0	0
4	-0.178326	-0.023989	0.179932	0	0
...	...	...	...	...	...
938	0.649913	0.079710	0.654783	1	1
939	-0.290557	0.569978	0.639764	1	1
940	-0.093805	0.601935	0.609200	1	1
941	0.633067	-0.111023	0.642729	1	1
942	-0.363217	-0.479049	0.601178	1	1

943 rows × 5 columns

In [17]:
```
data.iloc[:,3].equals(data.iloc[:,4])
```

Out[17]: True

In [23]:
```
data.drop(data.columns[[4]], axis = 1, inplace = True)
```

In [24]:
```
data.columns= ['c1','c2','c3','target']
```

In [33]:
```
data.head()
```

Out[33]:

	c1	c2	c3	target
0	0.102303	0.116327	0.154913	0
1	-0.131546	-0.038680	0.137115	0
2	0.007224	-0.067146	0.067533	0
3	0.112290	0.040584	0.119399	0
4	-0.178326	-0.023989	0.179932	0

In [27]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 943 entries, 0 to 942
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   c1      943 non-null    float64
 1   c2      943 non-null    float64
 2   c3      943 non-null    float64
 3   target  943 non-null    int64
dtypes: float64(3), int64(1)
memory usage: 29.6 KB
```

In [32]:
```
data.shape
```

Out[32]: (943, 4)

In [29]:
```
data.head().T
```

Out[29]:

	0	1	2	3	4
c1	0.102303	-0.131546	0.007224	0.112290	-0.178326
c2	0.116327	-0.038680	-0.067146	0.040584	-0.023989
c3	0.154913	0.137115	0.067533	0.119399	0.179932
target	0.000000	0.000000	0.000000	0.000000	0.000000

In [34]:
```
#Number of distinct categories or classes
data.iloc[:,3].unique()
```

Out[34]: array([0, 1])

In [35]:
```
#checking the percentage of each class in the dataset
(data.iloc[:,3].value_counts(normalize=True))#/(data.iloc[:,3].count())
```

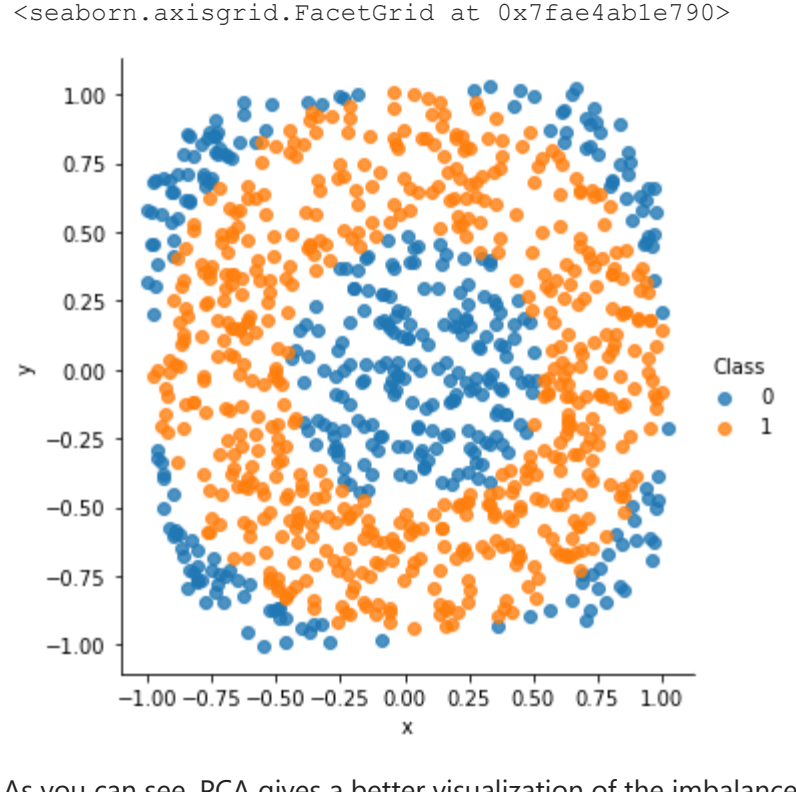
Out[35]:
```
1    0.603393
0    0.396607
Name: target, dtype: float64
```

This shows a complete imbalance of classes. There are 60.33% '1' instances and only 39.66% '0' instances. This means that we are aiming to predict anomalous events.

In [36]:
```
#PCA is performed for visualization only

pca= PCA(n_components=2)
viz_2d= pd.DataFrame(pca.fit_transform(data.iloc[:,0:3]))
viz_2d= pd.concat([viz_2d, data.iloc[:,3]], axis=1)
viz_2d.columns= ['x', 'y', 'Class']
sns.lmplot(x='x', y='y', data=viz_2d, fit_reg=False, hue='Class')
```

Out[36]: <seaborn.axisgrid.FacetGrid at 0x7fae4able790>



As you can see, PCA gives a better visualization of the imbalance in the datasets.

Fetaure transformation

In [37]:
```
df_transformed=pd.DataFrame(pca.fit_transform(data.iloc[:,0:3]))
df_transformed.columns = ['pca1','pca2']
```

In [38]:
```
X_data =df_transformed# data.iloc[:,0:3]
y_data = data.iloc[:, -1]
```

In [39]:
```
#printing the shape of the data
print(X_data.shape)
print(y_data.shape)
```

```
(943, 2)
(943,)
```

In [40]:
```
X_data.describe()
```

Out[40]:

	pca1	pca2
count	9.430000e+02	9.430000e+02
mean	-1.306837e-17	-1.895503e-17
std	5.645451e-01	5.412421e-01
min	-1.002615e+00	-1.003604e+00
25%	-4.936914e-01	-4.581877e-01
50%	-3.613059e-03	-2.077153e-02
75%	4.857837e-01	4.473309e-01
max	1.023938e+00	1.028275e+00

Splitting the Data into train and test set

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size = 0.3, random_state = 7)
```

In [42]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(660, 2)
(283, 2)
(660,)
(283,)
```

In [43]:
```
def make_confusion_matrix(cf,
                          group_names=None,
                          categories='auto',
                          count=True,
                          percent=True,
                          cbar=True,
                          xyticks=True,
                          xyplotlabels=True,
                          sum_stats=True,
                          figsize=None,
                          cmap='Blues',
                          title=None):
    '''
    This function will make a pretty plot of an sklearn Confusion Matrix cm using a Seaborn heatmap visualizati
    Arguments
    '''


    # CODE TO GENERATE TEXT INSIDE EACH SQUARE
    blanks = ['' for i in range(cf.size)]

    if group_names and len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
    else:
        group_labels = blanks

    if count:
        group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
    else:
        group_counts = blanks

    if percent:
        group_percentages = ["{0:.2%}".format(value) for value in cf.flatten()/np.sum(cf)]
    else:
        group_percentages = blanks

    box_labels = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in zip(group_labels,group_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])


    # CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
    if sum_stats:
        #Accuracy is sum of diagonal divided by total observations
        accuracy  = np.trace(cf) / float(np.sum(cf))

        #if it is a binary confusion matrix, show some more stats
        if len(cf)==2:
            #Metrics for Binary Confusion Matrices
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.3f}\nF1 Score={:0.3f}".format(
                accuracy,precision,recall,f1_score)
        else:
            stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    else:
        stats_text = ""


    # SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
    if figsize==None:
        #Get default figure size if not set
        figsize = plt.rcParams.get('figure.figsize')

    if xyticks==False:
        #Do not show categories if xyticks is False
        categories=False


    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)
    sns.heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=categories,yticklabels=categories)

    if xyplotlabels:
        plt.ylabel('True label')
        plt.xlabel('Predicted label' + stats_text)
    else:
        plt.xlabel(stats_text)

    if title:
        plt.title(title)
```

Model Building

Here two hidden layer with hidden node as "4" and learnign rate is 0.01

In [44]:
```
#initialize the modell
model1 = Sequential()


# This adds the input layer (by specifying input dimension) AND the first hidden layer (units)
model1.add(Dense(units=4, input_dim = 2,activation='relu'))
# hidden layer
model1.add(Dense(units=4,activation='relu'))
#Adding Dropout to prevent overfitting
model1.add(Dropout(0.5))
#model1.add(Dense(24,activation='relu'))
model1.add(Dense(4,activation='relu'))
# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output
# We use the sigmoid because we want probability outcomes
model1.add(Dense(1,activation='sigmoid'))

opt = optimizers.Adam(learning_rate=0.01)
# Compile the modell

model1.compile(optimizer=opt,loss='binary_crossentropy',metrics=['accuracy'])

model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4)                 12

 dense_1 (Dense)             (None, 4)                 20

 dropout (Dropout)           (None, 4)                 0

 dense_2 (Dense)             (None, 4)                 20

 dense_3 (Dense)             (None, 1)                 5

=================================================================
Total params: 57
Trainable params: 57
Non-trainable params: 0
_________________________________________________________________
```

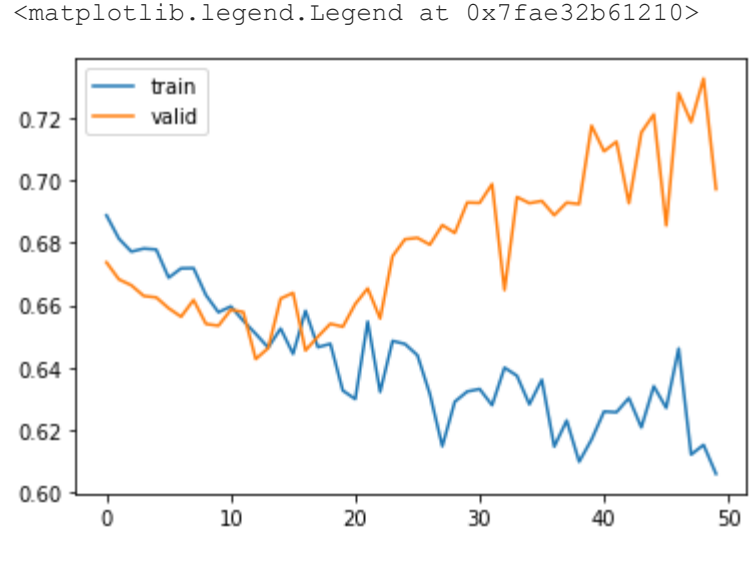
In [45]:
```
#fitting the model
history=model1.fit(X_train,y_train,batch_size=15,epochs=50,validation_split=0.2)
```

```
Epoch 1/50
36/36 [==============================] - 2s 12ms/step - loss: 0.6888 - accuracy: 0.5417 - val_loss: 0.6737 - va
l_accuracy: 0.6136
Epoch 2/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6814 - accuracy: 0.5777 - val_loss: 0.6683 - val
_accuracy: 0.6136
Epoch 3/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6772 - accuracy: 0.5777 - val_loss: 0.6664 - val
_accuracy: 0.6136
Epoch 4/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6782 - accuracy: 0.5777 - val_loss: 0.6630 - val
_accuracy: 0.6136
Epoch 5/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6779 - accuracy: 0.5777 - val_loss: 0.6625 - val
_accuracy: 0.6136
Epoch 6/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6688 - accuracy: 0.5890 - val_loss: 0.6590 - val
_accuracy: 0.6136
Epoch 7/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6718 - accuracy: 0.5985 - val_loss: 0.6563 - val
_accuracy: 0.6136
Epoch 8/50
36/36 [==============================] - 0s 5ms/step - loss: 0.6719 - accuracy: 0.5758 - val_loss: 0.6617 - val
_accuracy: 0.6136
Epoch 9/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6632 - accuracy: 0.5909 - val_loss: 0.6540 - val
_accuracy: 0.6136
Epoch 10/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6577 - accuracy: 0.5966 - val_loss: 0.6534 - val
_accuracy: 0.6136
Epoch 11/50
36/36 [==============================] - 0s 7ms/step - loss: 0.6596 - accuracy: 0.6004 - val_loss: 0.6585 - val
_accuracy: 0.6136
Epoch 12/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6550 - accuracy: 0.6004 - val_loss: 0.6578 - val
_accuracy: 0.6136
Epoch 13/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6508 - accuracy: 0.6004 - val_loss: 0.6427 - val
_accuracy: 0.6364
Epoch 14/50
36/36 [==============================] - 0s 5ms/step - loss: 0.6464 - accuracy: 0.6117 - val_loss: 0.6462 - val
_accuracy: 0.6136
Epoch 15/50
36/36 [==============================] - 0s 6ms/step - loss: 0.6525 - accuracy: 0.5947 - val_loss: 0.6621 - val
_accuracy: 0.6136
Epoch 16/50
36/36 [==============================] - 0s 5ms/step - loss: 0.6444 - accuracy: 0.6174 - val_loss: 0.6640 - val
_accuracy: 0.6136
Epoch 17/50
36/36 [==============================] - 0s 5ms/step - loss: 0.6582 - accuracy: 0.5909 - val_loss: 0.6455 - val
_accuracy: 0.6212
Epoch 18/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6465 - accuracy: 0.6004 - val_loss: 0.6499 - val
_accuracy: 0.6212
Epoch 19/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6476 - accuracy: 0.6023 - val_loss: 0.6541 - val
_accuracy: 0.6136
Epoch 20/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6326 - accuracy: 0.6174 - val_loss: 0.6531 - val
_accuracy: 0.6212
Epoch 21/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6299 - accuracy: 0.6080 - val_loss: 0.6604 - val
_accuracy: 0.6136
Epoch 22/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6547 - accuracy: 0.5947 - val_loss: 0.6654 - val
_accuracy: 0.6136
Epoch 23/50
36/36 [==============================] - 0s 5ms/step - loss: 0.6322 - accuracy: 0.6193 - val_loss: 0.6558 - val
_accuracy: 0.6136
Epoch 24/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6485 - accuracy: 0.5795 - val_loss: 0.6757 - val
_accuracy: 0.6136
Epoch 25/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6476 - accuracy: 0.6061 - val_loss: 0.6812 - val
_accuracy: 0.6136
Epoch 26/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6439 - accuracy: 0.6042 - val_loss: 0.6816 - val
_accuracy: 0.6136
Epoch 27/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6315 - accuracy: 0.6174 - val_loss: 0.6794 - val
_accuracy: 0.6136
Epoch 28/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6148 - accuracy: 0.6174 - val_loss: 0.6857 - val
_accuracy: 0.6136
Epoch 29/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6290 - accuracy: 0.6136 - val_loss: 0.6832 - val
_accuracy: 0.6136
Epoch 30/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6323 - accuracy: 0.5985 - val_loss: 0.6929 - val
_accuracy: 0.6136
Epoch 31/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6331 - accuracy: 0.6042 - val_loss: 0.6928 - val
_accuracy: 0.6136
Epoch 32/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6279 - accuracy: 0.6193 - val_loss: 0.6989 - val
_accuracy: 0.6136
Epoch 33/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6400 - accuracy: 0.5947 - val_loss: 0.6649 - val
_accuracy: 0.6136
Epoch 34/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6374 - accuracy: 0.6004 - val_loss: 0.6947 - val
_accuracy: 0.6136
Epoch 35/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6282 - accuracy: 0.6136 - val_loss: 0.6927 - val
_accuracy: 0.6136
Epoch 36/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6361 - accuracy: 0.5966 - val_loss: 0.6934 - val
_accuracy: 0.6136
Epoch 37/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6146 - accuracy: 0.6250 - val_loss: 0.6889 - val
_accuracy: 0.6136
Epoch 38/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6230 - accuracy: 0.6174 - val_loss: 0.6929 - val
_accuracy: 0.6136
Epoch 39/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6098 - accuracy: 0.6193 - val_loss: 0.6925 - val
_accuracy: 0.6136
Epoch 40/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6169 - accuracy: 0.6136 - val_loss: 0.7176 - val
_accuracy: 0.6136
Epoch 41/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6259 - accuracy: 0.6117 - val_loss: 0.7094 - val
_accuracy: 0.6136
Epoch 42/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6257 - accuracy: 0.6080 - val_loss: 0.7125 - val
_accuracy: 0.6136
Epoch 43/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6302 - accuracy: 0.6098 - val_loss: 0.6927 - val
_accuracy: 0.6136
Epoch 44/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6208 - accuracy: 0.6098 - val_loss: 0.7154 - val
_accuracy: 0.6136
Epoch 45/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6340 - accuracy: 0.6061 - val_loss: 0.7212 - val
_accuracy: 0.6136
Epoch 46/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6271 - accuracy: 0.6155 - val_loss: 0.6856 - val
_accuracy: 0.6136
Epoch 47/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6461 - accuracy: 0.5852 - val_loss: 0.7281 - val
_accuracy: 0.6136
Epoch 48/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6121 - accuracy: 0.6212 - val_loss: 0.7187 - val
_accuracy: 0.6136
Epoch 49/50
36/36 [==============================] - 0s 2ms/step - loss: 0.6152 - accuracy: 0.6212 - val_loss: 0.7327 - val
_accuracy: 0.6136
Epoch 50/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6059 - accuracy: 0.6250 - val_loss: 0.6972 - val
_accuracy: 0.6136
```

In [46]:
```
# Capturing learning history per epoch
hist  = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

# Plotting accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)
```

Out[46]: <matplotlib.legend.Legend at 0x7fae32b61210>



Evaluation

In [47]:
```
score = model1.evaluate(X_test, y_test)
```

```
9/9 [==============================] - 0s 3ms/step - loss: 0.6712 - accuracy: 0.6466
```

In [48]:
```
print(score)
```

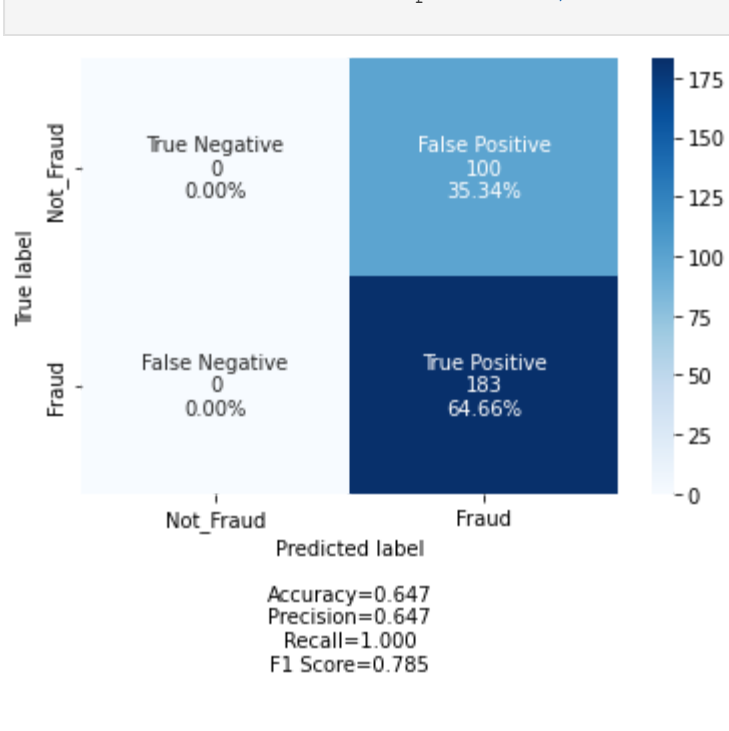
```
[0.67124342918396, 0.6466431021690369]
```

Let's Print confusion matrix

In [49]:
```
## Confusion Matrix on unsee test set
import seaborn as sn
y_pred1 = model1.predict(X_test)
for i in range(len(y_test)):
    if y_pred1[i]>0.5:
        y_pred1[i]=1
    else:
        y_pred1[i]=0



cm2=confusion_matrix(y_test, y_pred1)
labels = ['True Negative','False Positive','False Negative','True Positive']
categories = [ 'Not_Fraud','Fraud']
make_confusion_matrix(cm2,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



Model Tuning

Here two hidden layer with hidden node as "8" and learnign rate is 0.01

In [50]:
```
#initialize the model
model2 = Sequential()


# This adds the input layer (by specifying input dimension) AND the first hidden layer (units)
model2.add(Dense(units=8, input_dim = 2,activation='relu'))
# hidden layer
model2.add(Dense(units=8,activation='relu'))
#Adding Dropout to prevent overfitting
model2.add(Dropout(0.5))
#model.add(Dense(24,activation='relu'))
model2.add(Dense(8,activation='relu'))
# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output
# We use the sigmoid because we want probability outcomes
model2.add(Dense(1,activation='sigmoid'))

# Compile the model
opt = optimizers.Adam(learning_rate=0.01)
# Compile the modell

model2.compile(optimizer=opt,loss='binary_crossentropy',metrics=['accuracy'])
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 8)                 24

 dense_5 (Dense)             (None, 8)                 72

 dropout_1 (Dropout)         (None, 8)                 0

 dense_6 (Dense)             (None, 8)                 72

 dense_7 (Dense)             (None, 1)                 9

=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
_________________________________________________________________
```

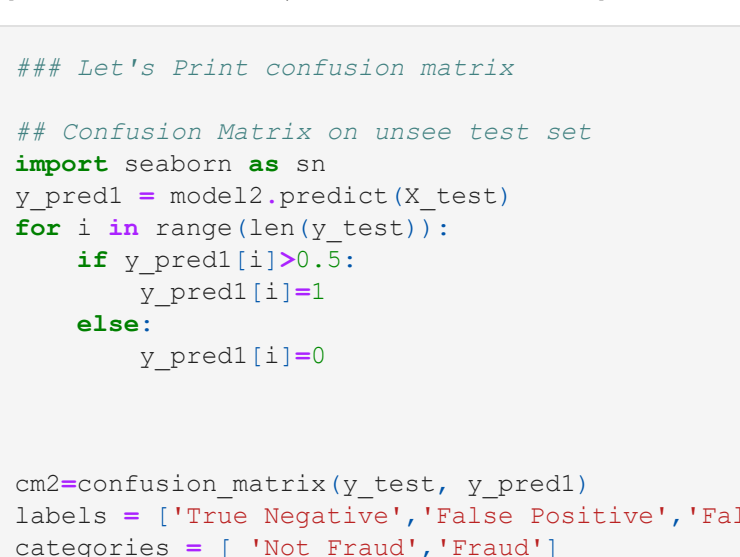
In [51]:
```
#fitting the model
history=model2.fit(X_train,y_train,batch_size=15,epochs=50,validation_split=0.2)
```

```
Epoch 1/50
36/36 [==============================] - 1s 9ms/step - loss: 0.6887 - accuracy: 0.5436 - val_loss: 0.6651 - val
_accuracy: 0.6136
Epoch 2/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6807 - accuracy: 0.5777 - val_loss: 0.6655 - val
_accuracy: 0.6136
Epoch 3/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6734 - accuracy: 0.5777 - val_loss: 0.6569 - val
_accuracy: 0.6136
Epoch 4/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6707 - accuracy: 0.5777 - val_loss: 0.6505 - val
_accuracy: 0.6136
Epoch 5/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6682 - accuracy: 0.5777 - val_loss: 0.6517 - val
_accuracy: 0.6136
Epoch 6/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6758 - accuracy: 0.5777 - val_loss: 0.6546 - val
_accuracy: 0.6136
Epoch 7/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6634 - accuracy: 0.5777 - val_loss: 0.6607 - val
_accuracy: 0.6136
Epoch 8/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6561 - accuracy: 0.5777 - val_loss: 0.6452 - val
_accuracy: 0.6136
Epoch 9/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6656 - accuracy: 0.5871 - val_loss: 0.6506 - val
_accuracy: 0.6061
Epoch 10/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6595 - accuracy: 0.6004 - val_loss: 0.6445 - val
_accuracy: 0.6439
Epoch 11/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6599 - accuracy: 0.5985 - val_loss: 0.6502 - val
_accuracy: 0.6667
Epoch 12/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6514 - accuracy: 0.6326 - val_loss: 0.6483 - val
_accuracy: 0.6364
Epoch 13/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6523 - accuracy: 0.6364 - val_loss: 0.6384 - val
_accuracy: 0.6970
Epoch 14/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6468 - accuracy: 0.6383 - val_loss: 0.6496 - val
_accuracy: 0.7424
Epoch 15/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6298 - accuracy: 0.6818 - val_loss: 0.6266 - val
_accuracy: 0.7197
Epoch 16/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6294 - accuracy: 0.7102 - val_loss: 0.6261 - val
_accuracy: 0.7500
Epoch 17/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6182 - accuracy: 0.7045 - val_loss: 0.6333 - val
_accuracy: 0.6970
Epoch 18/50
36/36 [==============================] - 0s 3ms/step - loss: 0.6027 - accuracy: 0.7083 - val_loss: 0.6104 - val
_accuracy: 0.7652
Epoch 19/50
36/36 [==============================] - 0s 4ms/step - loss: 0.6084 - accuracy: 0.7102 - val_loss: 0.6034 - val
_accuracy: 0.7576
Epoch 20/50
36/36 [==============================] - 0s 8ms/step - loss: 0.5930 - accuracy: 0.7121 - val_loss: 0.6157 - val
_accuracy: 0.6818
Epoch 21/50
36/36 [==============================] - 0s 7ms/step - loss: 0.5846 - accuracy: 0.7367 - val_loss: 0.6018 - val
_accuracy: 0.7121
Epoch 22/50
36/36 [==============================] - 0s 7ms/step - loss: 0.5763 - accuracy: 0.7330 - val_loss: 0.5670 - val
_accuracy: 0.7576
Epoch 23/50
36/36 [==============================] - 0s 7ms/step - loss: 0.5818 - accuracy: 0.7121 - val_loss: 0.6060 - val
_accuracy: 0.6894
Epoch 24/50
36/36 [==============================] - 0s 7ms/step - loss: 0.5688 - accuracy: 0.7462 - val_loss: 0.5845 - val
_accuracy: 0.7348
Epoch 25/50
36/36 [==============================] - 0s 3ms/step - loss: 0.5649 - accuracy: 0.7348 - val_loss: 0.5819 - val
_accuracy: 0.7652
Epoch 26/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5283 - accuracy: 0.7557 - val_loss: 0.5773 - val
_accuracy: 0.7197
Epoch 27/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5746 - accuracy: 0.7348 - val_loss: 0.5867 - val
_accuracy: 0.7197
Epoch 28/50
36/36 [==============================] - 0s 3ms/step - loss: 0.5472 - accuracy: 0.7311 - val_loss: 0.5720 - val
_accuracy: 0.7348
Epoch 29/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5411 - accuracy: 0.7443 - val_loss: 0.5584 - val
_accuracy: 0.7652
Epoch 30/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5144 - accuracy: 0.7576 - val_loss: 0.5824 - val
_accuracy: 0.7348
Epoch 31/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5420 - accuracy: 0.7311 - val_loss: 0.5699 - val
_accuracy: 0.7348
Epoch 32/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5279 - accuracy: 0.7557 - val_loss: 0.6035 - val
_accuracy: 0.6439
Epoch 33/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5314 - accuracy: 0.7481 - val_loss: 0.5443 - val
_accuracy: 0.7273
Epoch 34/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5595 - accuracy: 0.7405 - val_loss: 0.5884 - val
_accuracy: 0.7121
Epoch 35/50
36/36 [==============================] - 0s 2ms/step - loss: 0.4964 - accuracy: 0.7689 - val_loss: 0.4884 - val
_accuracy: 0.7955
Epoch 36/50
36/36 [==============================] - 0s 2ms/step - loss: 0.4879 - accuracy: 0.7822 - val_loss: 0.4996 - val
_accuracy: 0.8030
Epoch 37/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5093 - accuracy: 0.7652 - val_loss: 0.5583 - val
_accuracy: 0.7576
Epoch 38/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5195 - accuracy: 0.7519 - val_loss: 0.5593 - val
_accuracy: 0.7273
Epoch 39/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5295 - accuracy: 0.7652 - val_loss: 0.5694 - val
_accuracy: 0.6970
Epoch 40/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5042 - accuracy: 0.7860 - val_loss: 0.5485 - val
_accuracy: 0.7348
Epoch 41/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5243 - accuracy: 0.7576 - val_loss: 0.5283 - val
_accuracy: 0.7500
Epoch 42/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5133 - accuracy: 0.7670 - val_loss: 0.5730 - val
_accuracy: 0.7045
Epoch 43/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5036 - accuracy: 0.7879 - val_loss: 0.5469 - val
_accuracy: 0.7348
Epoch 44/50
36/36 [==============================] - 0s 2ms/step - loss: 0.4699 - accuracy: 0.7936 - val_loss: 0.5158 - val
_accuracy: 0.7879
Epoch 45/50
36/36 [==============================] - 0s 2ms/step - loss: 0.4928 - accuracy: 0.7765 - val_loss: 0.5268 - val
_accuracy: 0.7500
Epoch 46/50
36/36 [==============================] - 0s 2ms/step - loss: 0.4644 - accuracy: 0.8049 - val_loss: 0.5422 - val
_accuracy: 0.7197
Epoch 47/50
36/36 [==============================] - 0s 2ms/step - loss: 0.4884 - accuracy: 0.7822 - val_loss: 0.5286 - val
_accuracy: 0.7424
Epoch 48/50
36/36 [==============================] - 0s 2ms/step - loss: 0.4816 - accuracy: 0.7727 - val_loss: 0.5814 - val
_accuracy: 0.6970
Epoch 49/50
36/36 [==============================] - 0s 2ms/step - loss: 0.5280 - accuracy: 0.7689 - val_loss: 0.5207 - val
_accuracy: 0.7424
Epoch 50/50
36/36 [==============================] - 0s 2ms/step - loss: 0.4875 - accuracy: 0.7727 - val_loss: 0.5786 - val
_accuracy: 0.7197
```

In [52]:
```
# Capturing learning history per epoch
hist  = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

# Plotting accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)
```

Out[52]: <matplotlib.legend.Legend at 0x7fae38b1bb50>



In [53]:
```
score = model2.evaluate(X_test, y_test)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 0.6086 - accuracy: 0.6572
```

In [54]:
```
print(score)
```

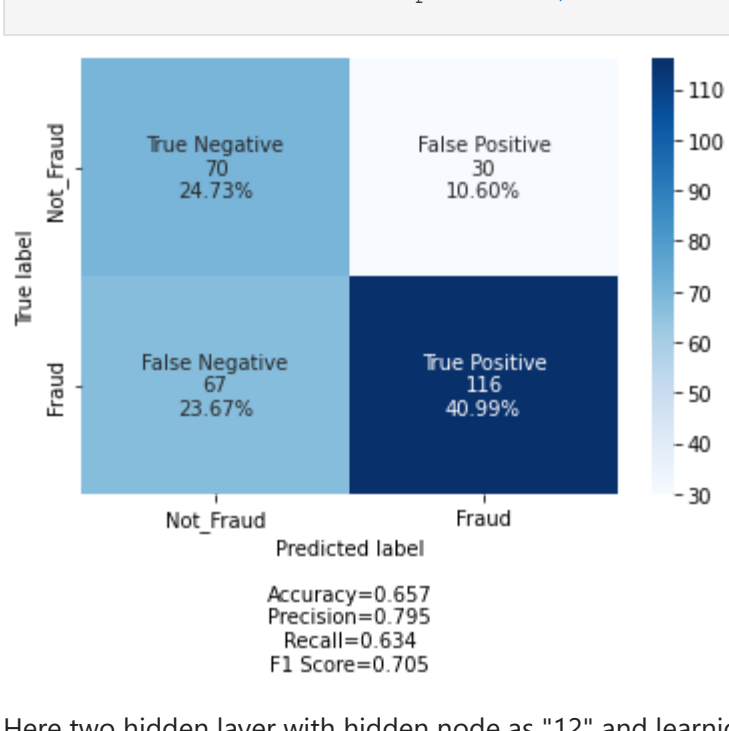
```
[0.608582079410553, 0.6572437882423401]
```

In [55]:
```
### Let's Print confusion matrix

## Confusion Matrix on unsee test set
import seaborn as sn
y_pred1 = model2.predict(X_test)
for i in range(len(y_test)):
    if y_pred1[i]>0.5:
        y_pred1[i]=1
    else:
        y_pred1[i]=0



cm2=confusion_matrix(y_test, y_pred1)
labels = ['True Negative','False Positive','False Negative','True Positive']
categories = [ 'Not_Fraud','Fraud']
make_confusion_matrix(cm2,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



Here two hidden layer with hidden node as "12" and learnign rate is 0.01

In [56]:
```
#initialize the model
model3 = Sequential()
# This adds the input layer (by specifying input dimension) AND the first hidden layer (units)
model3.add(Dense(units=4, input_dim = 2,activation='relu'))
# hidden layer
model3.add(Dense(units=12,activation='relu'))
#Adding Dropout to prevent overfitting
model3.add(Dropout(0.5))
#model.add(Dense(24,activation='relu'))
model3.add(Dense(12,activation='relu'))
# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output
# We use the sigmoid because we want probability outcomes
model3.add(Dense(1,activation='sigmoid'))
# Create optimizer with default learning rate
opt = optimizers.Adam(learning_rate=0.01)
# Compile the model3

model3.compile(optimizer=opt,loss='binary_crossentropy',metrics=['accuracy'])
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 4)                 12

 dense_9 (Dense)             (None, 12)                60

 dropout_2 (Dropout)         (None, 12)                0

 dense_10 (Dense)            (None, 12)                156

 dense_11 (Dense)            (None, 1)                 13

=================================================================
Total params: 241
Trainable params: 241
Non-trainable params: 0
_________________________________________________________________
```

In [57]:

```
#fitting the model
history=model3.fit(X_train,y_train,batch_size=25,epochs=50,validation_split=0.2)
```

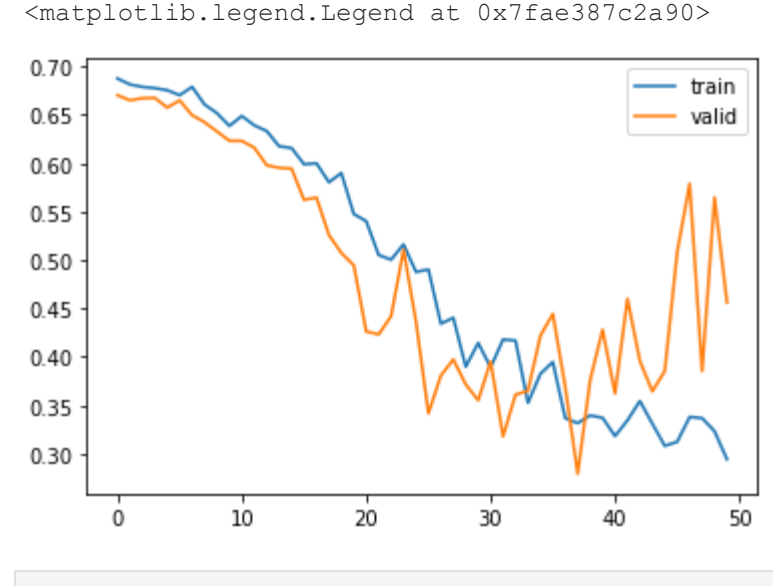
```
Epoch 1/50
22/22 [==============================] - 1s 13ms/step - loss: 0.6876 - accuracy: 0.5568 - val_loss: 0.6704 - va
l_accuracy: 0.6136
Epoch 2/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6815 - accuracy: 0.5777 - val_loss: 0.6651 - val
_accuracy: 0.6136
Epoch 3/50
22/22 [==============================] - 0s 7ms/step - loss: 0.6790 - accuracy: 0.5777 - val_loss: 0.6672 - val
_accuracy: 0.6136
Epoch 4/50
22/22 [==============================] - 0s 6ms/step - loss: 0.6777 - accuracy: 0.5701 - val_loss: 0.6676 - val
_accuracy: 0.6136
Epoch 5/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6756 - accuracy: 0.5833 - val_loss: 0.6576 - val
_accuracy: 0.6136
Epoch 6/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6704 - accuracy: 0.6098 - val_loss: 0.6650 - val
_accuracy: 0.6288
Epoch 7/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6790 - accuracy: 0.5701 - val_loss: 0.6498 - val
_accuracy: 0.6364
Epoch 8/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6608 - accuracy: 0.6345 - val_loss: 0.6425 - val
_accuracy: 0.6591
Epoch 9/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6518 - accuracy: 0.6364 - val_loss: 0.6329 - val
_accuracy: 0.6515
Epoch 10/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6387 - accuracy: 0.6572 - val_loss: 0.6232 - val
_accuracy: 0.6439
Epoch 11/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6488 - accuracy: 0.6420 - val_loss: 0.6231 - val
_accuracy: 0.6667
Epoch 12/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6392 - accuracy: 0.6402 - val_loss: 0.6161 - val
_accuracy: 0.6439
Epoch 13/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6333 - accuracy: 0.6629 - val_loss: 0.5980 - val
_accuracy: 0.6894
Epoch 14/50
22/22 [==============================] - 0s 3ms/step - loss: 0.6176 - accuracy: 0.6723 - val_loss: 0.5953 - val
_accuracy: 0.6894
Epoch 15/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6158 - accuracy: 0.6667 - val_loss: 0.5945 - val
_accuracy: 0.6818
Epoch 16/50
22/22 [==============================] - 0s 4ms/step - loss: 0.5990 - accuracy: 0.6686 - val_loss: 0.5622 - val
_accuracy: 0.6894
Epoch 17/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6000 - accuracy: 0.6780 - val_loss: 0.5646 - val
_accuracy: 0.6970
Epoch 18/50
22/22 [==============================] - 0s 4ms/step - loss: 0.5804 - accuracy: 0.7027 - val_loss: 0.5260 - val
_accuracy: 0.6970
Epoch 19/50
22/22 [==============================] - 0s 4ms/step - loss: 0.5899 - accuracy: 0.6875 - val_loss: 0.5073 - val
_accuracy: 0.7803
Epoch 20/50
22/22 [==============================] - 0s 4ms/step - loss: 0.5476 - accuracy: 0.7159 - val_loss: 0.4945 - val
_accuracy: 0.7727
Epoch 21/50
22/22 [==============================] - 0s 6ms/step - loss: 0.5401 - accuracy: 0.7254 - val_loss: 0.4262 - val
_accuracy: 0.8333
Epoch 22/50
22/22 [==============================] - 0s 5ms/step - loss: 0.5052 - accuracy: 0.7500 - val_loss: 0.4232 - val
_accuracy: 0.8106
Epoch 23/50
22/22 [==============================] - 0s 5ms/step - loss: 0.5004 - accuracy: 0.7557 - val_loss: 0.4421 - val
_accuracy: 0.7955
Epoch 24/50
22/22 [==============================] - 0s 5ms/step - loss: 0.5162 - accuracy: 0.7254 - val_loss: 0.5108 - val
_accuracy: 0.7348
Epoch 25/50
22/22 [==============================] - 0s 4ms/step - loss: 0.4878 - accuracy: 0.7330 - val_loss: 0.4364 - val
_accuracy: 0.7576
Epoch 26/50
22/22 [==============================] - 0s 4ms/step - loss: 0.4902 - accuracy: 0.7670 - val_loss: 0.3416 - val
_accuracy: 0.8636
Epoch 27/50
22/22 [==============================] - 0s 3ms/step - loss: 0.4342 - accuracy: 0.7992 - val_loss: 0.3805 - val
_accuracy: 0.7803
Epoch 28/50
22/22 [==============================] - 0s 3ms/step - loss: 0.4403 - accuracy: 0.7841 - val_loss: 0.3973 - val
_accuracy: 0.8106
Epoch 29/50
22/22 [==============================] - 0s 4ms/step - loss: 0.3898 - accuracy: 0.8314 - val_loss: 0.3715 - val
_accuracy: 0.7955
Epoch 30/50
22/22 [==============================] - 0s 2ms/step - loss: 0.4144 - accuracy: 0.7860 - val_loss: 0.3550 - val
_accuracy: 0.8333
Epoch 31/50
22/22 [==============================] - 0s 2ms/step - loss: 0.3891 - accuracy: 0.8068 - val_loss: 0.3952 - val
_accuracy: 0.8182
Epoch 32/50
22/22 [==============================] - 0s 3ms/step - loss: 0.4180 - accuracy: 0.7841 - val_loss: 0.3177 - val
_accuracy: 0.8409
Epoch 33/50
22/22 [==============================] - 0s 3ms/step - loss: 0.4168 - accuracy: 0.7973 - val_loss: 0.3609 - val
_accuracy: 0.8258
Epoch 34/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3524 - accuracy: 0.8352 - val_loss: 0.3648 - val
_accuracy: 0.8106
Epoch 35/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3824 - accuracy: 0.8049 - val_loss: 0.4217 - val
_accuracy: 0.8030
Epoch 36/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3946 - accuracy: 0.7860 - val_loss: 0.4444 - val
_accuracy: 0.8106
Epoch 37/50
22/22 [==============================] - 0s 2ms/step - loss: 0.3366 - accuracy: 0.8258 - val_loss: 0.3689 - val
_accuracy: 0.8106
Epoch 38/50
22/22 [==============================] - 0s 2ms/step - loss: 0.3314 - accuracy: 0.8371 - val_loss: 0.2791 - val
_accuracy: 0.8788
Epoch 39/50
22/22 [==============================] - 0s 4ms/step - loss: 0.3394 - accuracy: 0.8201 - val_loss: 0.3759 - val
_accuracy: 0.8030
Epoch 40/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3370 - accuracy: 0.8144 - val_loss: 0.4281 - val
_accuracy: 0.8030
Epoch 41/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3181 - accuracy: 0.8390 - val_loss: 0.3620 - val
_accuracy: 0.8182
Epoch 42/50
22/22 [==============================] - 0s 2ms/step - loss: 0.3344 - accuracy: 0.8371 - val_loss: 0.4601 - val
_accuracy: 0.7955
Epoch 43/50
22/22 [==============================] - 0s 2ms/step - loss: 0.3543 - accuracy: 0.8371 - val_loss: 0.3956 - val
_accuracy: 0.8333
Epoch 44/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3309 - accuracy: 0.8239 - val_loss: 0.3643 - val
_accuracy: 0.8333
Epoch 45/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3080 - accuracy: 0.8504 - val_loss: 0.3854 - val
_accuracy: 0.8182
Epoch 46/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3122 - accuracy: 0.8390 - val_loss: 0.5083 - val
_accuracy: 0.7879
Epoch 47/50
22/22 [==============================] - 0s 5ms/step - loss: 0.3379 - accuracy: 0.8125 - val_loss: 0.5790 - val
_accuracy: 0.7879
Epoch 48/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3367 - accuracy: 0.8239 - val_loss: 0.3851 - val
_accuracy: 0.8258
Epoch 49/50
22/22 [==============================] - 0s 3ms/step - loss: 0.3231 - accuracy: 0.8409 - val_loss: 0.5647 - val
_accuracy: 0.7955
Epoch 50/50
22/22 [==============================] - 0s 3ms/step - loss: 0.2941 - accuracy: 0.8561 - val_loss: 0.4561 - val
_accuracy: 0.8030
```

In [58]:

```
# Capturing learning history per epoch
hist  = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

# Plotting accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)
```

Out[58]: `<matplotlib.legend.Legend at 0x7fae387c2a90>`



In [59]:

```
score = model3.evaluate(X_test, y_test)
```

```
9/9 [==============================] - 0s 3ms/step - loss: 0.4665 - accuracy: 0.7951
```

In [60]:

```
print(score)
```

```
[0.46649616956710815, 0.7950530052185059]
```

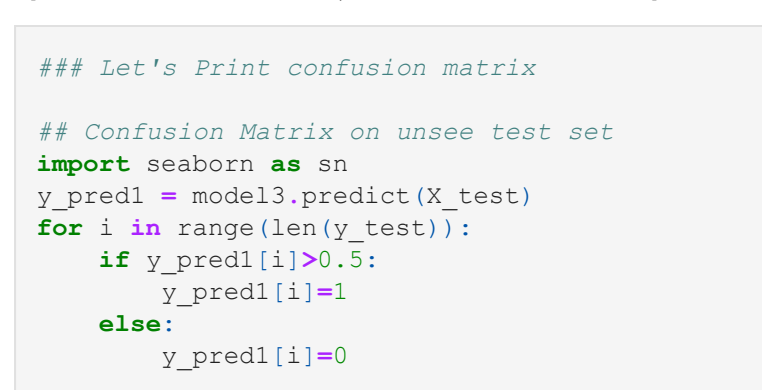
In [61]:

```
### Let's Print confusion matrix

## Confusion Matrix on unsee test set
import seaborn as sn
y_pred1 = model3.predict(X_test)
for i in range(len(y_test)):
    if y_pred1[i]>0.5:
        y_pred1[i]=1
    else:
        y_pred1[i]=0



cm2=confusion_matrix(y_test, y_pred1)
labels = ['True Negative','False Positive','False Negative','True Positive']
categories = [ 'Not_Fraud','Fraud']
make_confusion_matrix(cm2,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



Accuracy=0.795
Precision=0.771
Recall=0.973
F1 Score=0.860

Here two hidden layer with hidden node as "16" and learnign rate is 0.01

In [62]:

```
#initialize the model
model4 = Sequential()
# This adds the input layer (by specifying input dimension) AND the first hidden layer (units)
model4.add(Dense(units=8, input_dim = 2,activation='relu'))
# hidden layer
model4.add(Dense(units=16,activation='relu'))
#Adding Dropout to prevent overfitting
model4.add(Dropout(0.85))
#model.add(Dense(24,activation='relu'))
model4.add(Dense(16,activation='relu'))
# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output (fraud or not)
# We use the sigmoid because we want probability outcomes
model4.add(Dense(1,activation='sigmoid'))
# Create optimizer with default learning rate
# Compile the model
model4.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 8)                 24

 dense_13 (Dense)            (None, 16)                144

 dropout_3 (Dropout)         (None, 16)                0

 dense_14 (Dense)            (None, 16)                272

 dense_15 (Dense)            (None, 1)                 17

=================================================================
Total params: 457
Trainable params: 457
Non-trainable params: 0
_________________________________________________________________
```

In [63]:

```
#fitting the model
history=model4.fit(X_train,y_train,batch_size=25,epochs=50,validation_split=0.2)
```

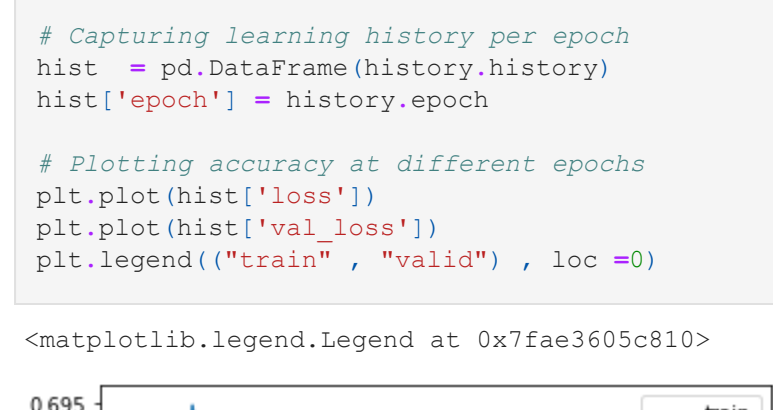
```
Epoch 1/50
22/22 [==============================] - 1s 13ms/step - loss: 0.6925 - accuracy: 0.5758 - val_loss: 0.6793 - va
l_accuracy: 0.6136
Epoch 2/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6886 - accuracy: 0.5701 - val_loss: 0.6799 - val
_accuracy: 0.6136
Epoch 3/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6775 - accuracy: 0.5701 - val_loss: 0.6795 - val
_accuracy: 0.6136
Epoch 4/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6803 - accuracy: 0.5795 - val_loss: 0.6773 - val
_accuracy: 0.6136
Epoch 5/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6801 - accuracy: 0.5795 - val_loss: 0.6764 - val
_accuracy: 0.6136
Epoch 6/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6949 - accuracy: 0.5720 - val_loss: 0.6777 - val
_accuracy: 0.6136
Epoch 7/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6880 - accuracy: 0.5701 - val_loss: 0.6778 - val
_accuracy: 0.6136
Epoch 8/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6883 - accuracy: 0.5795 - val_loss: 0.6783 - val
_accuracy: 0.6136
Epoch 9/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6799 - accuracy: 0.5758 - val_loss: 0.6764 - val
_accuracy: 0.6136
Epoch 10/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6810 - accuracy: 0.5758 - val_loss: 0.6758 - val
_accuracy: 0.6136
Epoch 11/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6781 - accuracy: 0.5758 - val_loss: 0.6751 - val
_accuracy: 0.6136
Epoch 12/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6787 - accuracy: 0.5758 - val_loss: 0.6744 - val
_accuracy: 0.6136
Epoch 13/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6852 - accuracy: 0.5758 - val_loss: 0.6753 - val
_accuracy: 0.6136
Epoch 14/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6881 - accuracy: 0.5758 - val_loss: 0.6763 - val
_accuracy: 0.6136
Epoch 15/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6841 - accuracy: 0.5758 - val_loss: 0.6764 - val
_accuracy: 0.6136
Epoch 16/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6801 - accuracy: 0.5739 - val_loss: 0.6764 - val
_accuracy: 0.6136
Epoch 17/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6795 - accuracy: 0.5777 - val_loss: 0.6744 - val
_accuracy: 0.6136
Epoch 18/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6811 - accuracy: 0.5777 - val_loss: 0.6736 - val
_accuracy: 0.6136
Epoch 19/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6724 - accuracy: 0.5777 - val_loss: 0.6736 - val
_accuracy: 0.6136
Epoch 20/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6794 - accuracy: 0.5777 - val_loss: 0.6717 - val
_accuracy: 0.6136
Epoch 21/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6793 - accuracy: 0.5777 - val_loss: 0.6720 - val
_accuracy: 0.6136
Epoch 22/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6762 - accuracy: 0.5777 - val_loss: 0.6720 - val
_accuracy: 0.6136
Epoch 23/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6796 - accuracy: 0.5777 - val_loss: 0.6719 - val
_accuracy: 0.6136
Epoch 24/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6794 - accuracy: 0.5758 - val_loss: 0.6716 - val
_accuracy: 0.6136
Epoch 25/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6788 - accuracy: 0.5777 - val_loss: 0.6720 - val
_accuracy: 0.6136
Epoch 26/50
22/22 [==============================] - 0s 9ms/step - loss: 0.6796 - accuracy: 0.5777 - val_loss: 0.6723 - val
_accuracy: 0.6136
Epoch 27/50
22/22 [==============================] - 0s 9ms/step - loss: 0.6792 - accuracy: 0.5795 - val_loss: 0.6715 - val
_accuracy: 0.6136
Epoch 28/50
22/22 [==============================] - 0s 11ms/step - loss: 0.6781 - accuracy: 0.5777 - val_loss: 0.6722 - va
l_accuracy: 0.6136
Epoch 29/50
22/22 [==============================] - 0s 9ms/step - loss: 0.6742 - accuracy: 0.5814 - val_loss: 0.6724 - val
_accuracy: 0.6136
Epoch 30/50
22/22 [==============================] - 0s 10ms/step - loss: 0.6721 - accuracy: 0.5758 - val_loss: 0.6709 - va
l_accuracy: 0.6136
Epoch 31/50
22/22 [==============================] - 0s 8ms/step - loss: 0.6772 - accuracy: 0.5795 - val_loss: 0.6717 - val
_accuracy: 0.6136
Epoch 32/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6762 - accuracy: 0.5795 - val_loss: 0.6713 - val
_accuracy: 0.6136
Epoch 33/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6769 - accuracy: 0.5777 - val_loss: 0.6719 - val
_accuracy: 0.6136
Epoch 34/50
22/22 [==============================] - 0s 3ms/step - loss: 0.6751 - accuracy: 0.5777 - val_loss: 0.6721 - val
_accuracy: 0.6136
Epoch 35/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6838 - accuracy: 0.5777 - val_loss: 0.6708 - val
_accuracy: 0.6136
Epoch 36/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6745 - accuracy: 0.5777 - val_loss: 0.6717 - val
_accuracy: 0.6136
Epoch 37/50
22/22 [==============================] - 0s 4ms/step - loss: 0.6784 - accuracy: 0.5777 - val_loss: 0.6706 - val
_accuracy: 0.6136
Epoch 38/50
22/22 [==============================] - 0s 3ms/step - loss: 0.6750 - accuracy: 0.5777 - val_loss: 0.6721 - val
_accuracy: 0.6136
Epoch 39/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6794 - accuracy: 0.5777 - val_loss: 0.6721 - val
_accuracy: 0.6136
Epoch 40/50
22/22 [==============================] - 0s 3ms/step - loss: 0.6768 - accuracy: 0.5777 - val_loss: 0.6712 - val
_accuracy: 0.6136
Epoch 41/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6763 - accuracy: 0.5777 - val_loss: 0.6709 - val
_accuracy: 0.6136
Epoch 42/50
22/22 [==============================] - 0s 3ms/step - loss: 0.6747 - accuracy: 0.5777 - val_loss: 0.6711 - val
_accuracy: 0.6136
Epoch 43/50
22/22 [==============================] - 0s 3ms/step - loss: 0.6766 - accuracy: 0.5833 - val_loss: 0.6698 - val
_accuracy: 0.6136
Epoch 44/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6754 - accuracy: 0.5795 - val_loss: 0.6686 - val
_accuracy: 0.6136
Epoch 45/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6763 - accuracy: 0.5758 - val_loss: 0.6690 - val
_accuracy: 0.6136
Epoch 46/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6761 - accuracy: 0.5777 - val_loss: 0.6698 - val
_accuracy: 0.6136
Epoch 47/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6698 - accuracy: 0.5777 - val_loss: 0.6706 - val
_accuracy: 0.6136
Epoch 48/50
22/22 [==============================] - 0s 3ms/step - loss: 0.6782 - accuracy: 0.5777 - val_loss: 0.6710 - val
_accuracy: 0.6136
Epoch 49/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6782 - accuracy: 0.5777 - val_loss: 0.6707 - val
_accuracy: 0.6136
Epoch 50/50
22/22 [==============================] - 0s 2ms/step - loss: 0.6788 - accuracy: 0.5777 - val_loss: 0.6706 - val
_accuracy: 0.6136
```

In [64]:

```
# Capturing learning history per epoch
hist  = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

# Plotting accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)
```

Out[64]: `<matplotlib.legend.Legend at 0x7fae3605c810>`



In [65]:

```
score = model4.evaluate(X_test, y_test)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 0.6650 - accuracy: 0.6466
```

In [66]:

```
print(score)
```

```
[0.6650264263153076, 0.6466431021690369]
```

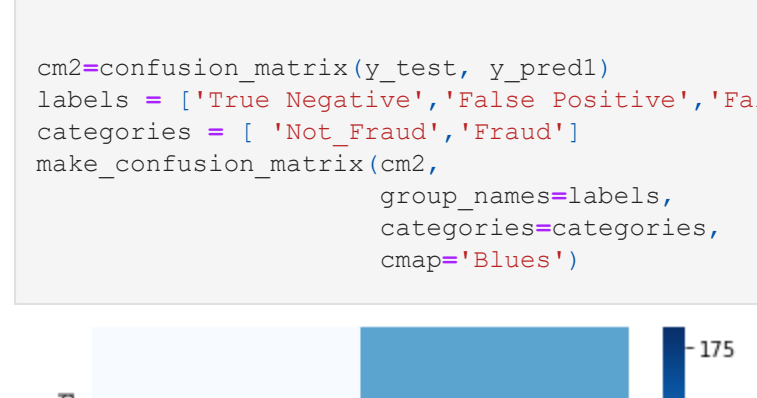
In [67]:

```
### Let's Print confusion matrix

## Confusion Matrix on unsee test set
import seaborn as sn
y_pred1 = model4.predict(X_test)
for i in range(len(y_test)):
    if y_pred1[i]>0.5:
        y_pred1[i]=1
    else:
        y_pred1[i]=0



cm2=confusion_matrix(y_test, y_pred1)
labels = ['True Negative','False Positive','False Negative','True Positive']
categories = [ 'Not_Fraud','Fraud']
make_confusion_matrix(cm2,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



Accuracy=0.647
Precision=0.647
Recall=1.000
F1 Score=0.785

Conclusion

- From the above evaluation using double hidden layer deep networks by varying the number of hidden nodes (4, 8, 12, 16) in each layer, Following things are observed:-
  1. with increase in the number of hidden nodes, accuracy got increased.
  2. increasing the epoc improve the result (make sure that over fitting does not happen)