In [1]:

```python
# Import required packages here (after they are installed)
import numpy as np
import pandas as pd
import matplotlib.pyplot as mp
from pylab import show
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
import time
```

BLOOD TRANSFUSION UCI DATA - NEURAL NETWORKS

In [2]:

```python
# Load data
blood_data = pd.read_csv("blood_transfusion_data.csv")
blood_data.head()
```

Out[2]:

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007
0	2	50	12500	98	1
1	0	13	3250	28	1
2	1	16	4000	35	1
3	2	20	5000	45	1
4	1	24	6000	77	0

In [3]:

```python
#Seggregate features and danation result
blood_features = []
donation = []

blood_features = blood_data.iloc[:, 0:4]
donation = blood_data.iloc[:, 4:5]

blood_features = np.array(blood_features)
donation = np.array(donation)
donation = [str(item) for item in donation]
```

In [4]:

```python
#blood_features
#donation
#len(blood_features)
#len(donation)
```

BLOOD TRANSFUSION FEATURES

In [5]:

```python
layers_b = [1,2,5]
nodes_b = [2,5,10]
```

1 - LAYER

In [7]:
```
cv_scores_b1 = []
cv_errors_b1 = []
run_times_b1 = []
for n in nodes_b:
    start_time = time.time()*1000
    #Neural Network Model
    print('LAYER-1; NODES-{}'.format(n))
    perceptron = MLPClassifier(hidden_layer_sizes=(n), activation='relu', solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
    print(perceptron)

    #Cross-validation scores
    cv_score = cross_val_score(perceptron, blood_features, donation, cv=10)
    cv_mean = np.mean(cv_score)
    print(cv_score)
    cv_scores_b1.append(cv_mean)
    print("Cross validation Mean:", cv_mean)

    #Cross-validation errors
    cv_error = 1-cv_mean
    cv_errors_b1.append(cv_error)
    print('Cross validation error:', cv_error)
    end_time = time.time()*1000
    runtime = end_time - start_time
    run_times_b1.append(runtime)
    print('Run Time:', runtime)
```

```
LAYER-1; NODES-2
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=2, learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.73333333 0.76       0.76       0.76       0.76       0.76
 0.76       0.76       0.77027027 0.77027027]
Cross validation Mean: 0.7593873873873873
Cross validation error: 0.2406126126126127
Run Time: 6732.794677734375
LAYER-1; NODES-5
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=5, learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.33333333 0.76       0.72       0.81333333 0.70666667 0.76
 0.76       0.76       0.77027027 0.77027027]
Cross validation Mean: 0.7153873873873874
Cross validation error: 0.2846126126126126
Run Time: 3358.56396484375
LAYER-1; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=10, learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.65333333 0.76       0.76       0.78666667 0.76       0.76
 0.65333333 0.98666667 0.7972973  0.82432432]
Cross validation Mean: 0.7741621621621622
Cross validation error: 0.22583783783783784
Run Time: 1749.54638671875
```

2 - LAYER

In [8]:

```
cv_scores_b2 = []
cv_errors_b2 = []
run_times_b2 = []
for n in nodes_b:
    start_time = time.time()*1000
    #Neural Network Model
    print('LAYER-2; NODES-{}'.format(n))
    perceptron = MLPClassifier(hidden_layer_sizes=(n, n), activation='relu', solver
='adam', alpha=0, max_iter=10000, epsilon=0.001)
    print(perceptron)

    #Cross-validation scores
    cv_score = cross_val_score(perceptron, blood_features, donation, cv=10)
    cv_mean = np.mean(cv_score)
    print(cv_score)
    cv_scores_b2.append(cv_mean)
    print("Cross validation Mean:", cv_mean)

    #Cross-validation errors
    cv_error = 1-cv_mean
    cv_errors_b2.append(cv_error)
    print('Cross validation error:', cv_error)
    end_time = time.time()*1000
    runtime = end_time - start_time
    run_times_b2.append(runtime)
    print('Run Time:', runtime)
```

```
LAYER-2; NODES-2
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(2, 2), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.76       0.76       0.76       0.76       0.76       0.76
 0.36       0.76       0.77027027 0.77027027]
Cross validation Mean: 0.722054054054054
Cross validation error: 0.277945945945946
Run Time: 7107.65576171875
LAYER-2; NODES-5
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(5, 5), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.65333333 0.82666667 0.76       0.76       0.73333333 0.76
 0.76       0.76       0.78378378 0.77027027]
Cross validation Mean: 0.7567387387387388
Cross validation error: 0.24326126126126124
Run Time: 4811.374755859375
LAYER-2; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(10, 10), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.8        0.97333333 0.74666667 0.76       0.81333333 0.76
 0.34666667 0.88       0.77027027 0.75675676]
Cross validation Mean: 0.7607027027027027
Cross validation error: 0.23929729729729732
Run Time: 2155.74072265625
```

5 - LAYERS

In [9]:

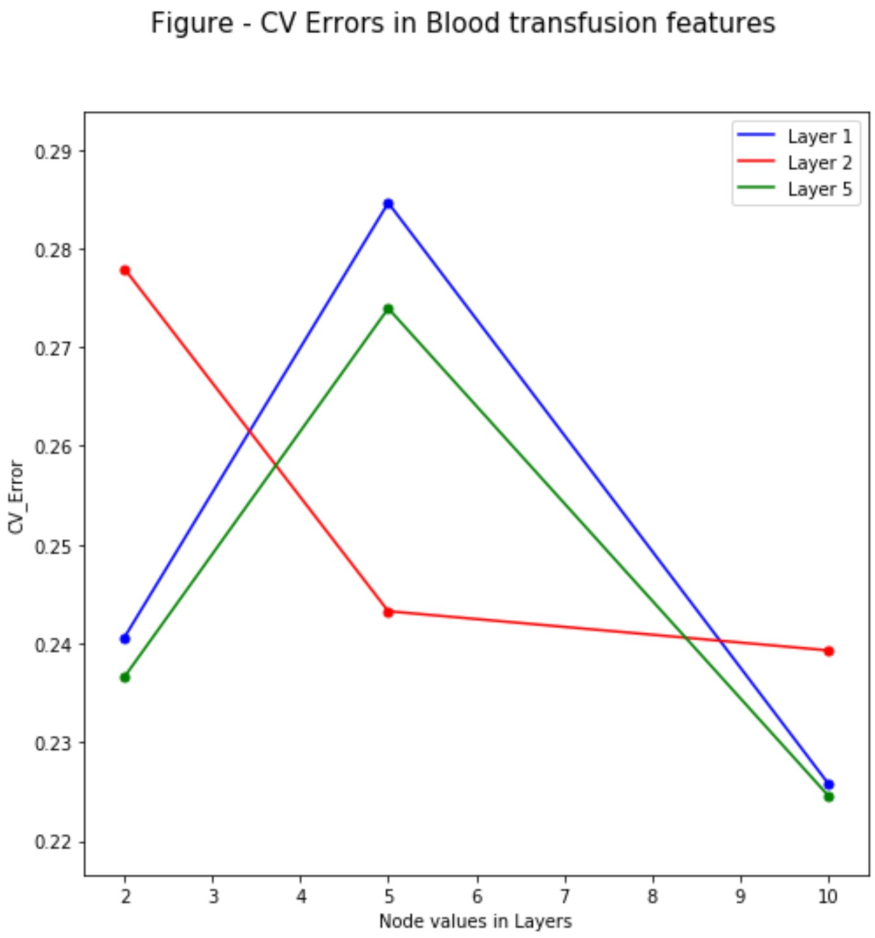
```
cv_scores_b5 = []
cv_errors_b5 = []
run_times_b5 = []
for n in nodes_b:
    start_time = time.time()*1000
    #Neural Network Model
    print('LAYER-5; NODES-{}'.format(n))
    perceptron = MLPClassifier(hidden_layer_sizes=(n, n, n, n, n), activation='relu', solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
    print(perceptron)

    #Cross-validation scores
    cv_score = cross_val_score(perceptron, blood_features, donation, cv=10)
    cv_mean = np.mean(cv_score)
    print(cv_score)
    cv_scores_b5.append(cv_mean)
    print("Cross validation Mean:", cv_mean)

    #Cross-validation errors
    cv_error = 1-cv_mean
    cv_errors_b5.append(cv_error)
    print('Cross validation error:', cv_error)
    end_time = time.time()*1000
    runtime = end_time - start_time
    run_times_b5.append(runtime)
    print('Run Time:', runtime)
```

```
LAYER-5; NODES-2
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(2, 2, 2, 2, 2), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.76       0.76       0.76       0.76       0.76       0.76
 0.76       0.77333333 0.77027027 0.77027027]
Cross validation Mean: 0.7633873873873874
Cross validation error: 0.23661261261261257
Run Time: 6045.41552734375
LAYER-5; NODES-5
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(5, 5, 5, 5, 5), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.78666667 0.76       0.24       0.76       0.76       0.76
 0.76       0.88       0.78378378 0.77027027]
Cross validation Mean: 0.7260720720720721
Cross validation error: 0.27392792792792786
Run Time: 7638.830322265625
LAYER-5; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(10, 10, 10, 10, 10), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.8        0.84       0.76       0.76       0.74666667 0.77333333
 0.77333333 0.76       0.77027027 0.77027027]
Cross validation Mean: 0.7753873873873873
Cross validation error: 0.22461261261261267
Run Time: 5404.9794921875
```

CV-ERRORS FOR BLOOD TRANSFUSION

In [10]:

```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
figb = mp.figure(figsize=(8, 8))

mp.scatter(nodes_b,cv_errors_b1,s=25, c='blue')
lineb1 = mp.plot(nodes_b,cv_errors_b1, label='Layer 1', color='blue')

mp.scatter(nodes_b,cv_errors_b2,s=25, c='red')
lineb2 = mp.plot(nodes_b,cv_errors_b2, label='Layer 2', color='red')

mp.scatter(nodes_b,cv_errors_b5,s=25, c='green')
lineb3 = mp.plot(nodes_b,cv_errors_b5, label='Layer 5', color='green')

mp.legend(prop={'size': 10})

#specify the axes
mp.xlabel("Node values in Layers")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
figb.suptitle('Figure - CV Errors in Blood transfusion features', fontsize=15)


#table
figb_t = mp.figure()
table_vals_b = []

np_array = np.array(cv_errors_b1)
np_round_to_tenths = np.around(np_array, 5)
round_to_tenths = list(np_round_to_tenths)
table_vals_b.append(round_to_tenths)

np_array = np.array(cv_errors_b2)
np_round_to_tenths = np.around(np_array, 5)
round_to_tenths = list(np_round_to_tenths)
table_vals_b.append(round_to_tenths)

np_array = np.array(cv_errors_b5)
np_round_to_tenths = np.around(np_array, 5)
round_to_tenths = list(np_round_to_tenths)
table_vals_b.append(round_to_tenths)

row_labels = ['1-Hidden Layer', '2-Hidden Layer', '5-Hidden Layer']
col_labels = ['2-Nodes', '5-Nodes', '10-Nodes']
the_table = mp.table(cellText=table_vals_b,
                      rowLabels=row_labels,
                      colLabels=col_labels,
                      loc='center')
the_table.auto_set_font_size(False)
the_table.set_fontsize(14)
the_table.scale(2, 2)

# Removing ticks and spines enables you to get the figure only with table
mp.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
mp.tick_params(axis='y', which='both', right=False, left=False, labelleft=False)
for pos in ['right','top','bottom','left']:
    mp.gca().spines[pos].set_visible(False)
mp.savefig('matplotlib-table.png', bbox_inches='tight', pad_inches=0.05)


#display the current graph
mp.show()
```



	2-Nodes	5-Nodes	10-Nodes
1-Hidden Layer	0.24061	0.28461	0.22584
2-Hidden Layer	0.27795	0.24326	0.2393
5-Hidden Layer	0.23661	0.27393	0.22461

The highest accuracy is obtained for 5-Hidden layers with 10-Nodes and then the next model very close is 1-Hidden layer with 10-Nodes as these models have the lowest cross validation errors. We cannot say that having high number of nodes wrt that particular layer is more desirable. i.e., for 1-Hidden Layer, highest accuracy is with 10-nodes then it is for 2-nodes and least is for 5-nodes for 2-Hidden Layers, highest accuracy is with 10-nodes then it is for 5-nodes and least is for 2-nodes for 5-Hidden Layers, the trend is same as 1-Hidden Layer

The highest accuracy is obtained for 5-Hidden layers with 10-Node with 0.22461 as the cross validation error.

HW1 FEATURES -NEURAL NETWORKS

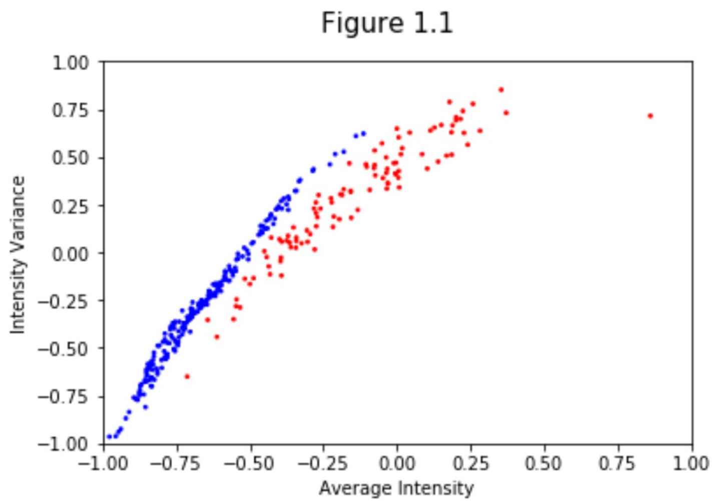
In [13]:
```
# Load data. csv file should be in the same folder as the notebook for this to wor
k, otherwise
# give data path.
data = np.loadtxt("data1.csv")
```

In [14]:

```
#shuffle the data and select training and test data
np.random.seed(100)
np.random.shuffle(data)


features = []
digits = []


for row in data:
    #import the data and select only the 1's and 5's
    if(row[0]==1 or row[0]==5):
        features.append(row[1:])
        digits.append(str(row[0]))


#Select the proportion of data to use for training.
#Notice that we have set aside 80% of the data for testing
numTrain = int(len(features)*.2)

trainFeatures = features[:numTrain]
testFeatures = features[numTrain:]
trainDigits = digits[:numTrain]
testDigits = digits[numTrain:]

#print(trainFeatures)
#trainFeatures[0]
```

In [15]:

```
#Convert the 256D data (trainFeatures) to 2D data
#We need X and Y for plotting and simpleTrain for building the model.
#They contain the same points in a different arrangement

X = []
Y = []
simpleTrain = []

#Colors will be passed to the graphing library to color the points.
#1's are blue: "b" and 5's are red: "r"
colors = []
#legends = []
for index in range(len(trainFeatures)):
    #print(index)
    #break
    #produce the 2D dataset for graphing/training and scale the data so it is in the [-1,1] square
    xNew = 2*np.average(trainFeatures[index])+.75
    yNew = 3*np.var(trainFeatures[index])-1.5
    X.append(xNew)
    Y.append(yNew)
    simpleTrain.append([xNew,yNew])
    #trainDigits will still be the value we try to classify. Here it is the string "1.0" or "5.0"
    if(trainDigits[index]=="1.0"):
        colors.append("b")
        #legends.append("1")
    else:
        colors.append("r")
        #legends.append("5")

#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#specify the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Figure 1.1', fontsize=15)

#display the current graph
show()
```



HW1 FEATURES

In [28]:
```
layers = [1,2,5,10]
nodes = [2,5,10,50,100]
```

LAYER 1

In [100]:

```
cv_scores_1 = []
cv_errors_1 = []
run_times_1 = []
for n in nodes:
    start_time = time.time()*1000
    #Neural Network Model
    print('LAYER-1; NODES-{}'.format(n))
    perceptron = MLPClassifier(hidden_layer_sizes=(n), activation='relu', solver='
adam', alpha=0, max_iter=10000, epsilon=0.001)
    print(perceptron)

    #Cross-validation scores
    cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
    cv_mean = np.mean(cv_score)
    print(cv_score)
    cv_scores_1.append(cv_mean)
    print("Cross validation Mean:", cv_mean)

    #Cross-validation errors
    cv_error = 1-cv_mean
    cv_errors_1.append(cv_error)
    print('Cross validation error:', cv_error)
    end_time = time.time()*1000
    runtime = end_time - start_time
    run_times_1.append(runtime)
    print('Run Time:', runtime)
```

```
LAYER-1; NODES-2
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=2, learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.65625    0.78125    0.65625    0.96774194 0.67741935 0.67741935
 0.93548387 1.         0.80645161 0.66666667]
Cross validation Mean: 0.7824932795698925
Cross validation error: 0.21750672043010755
Run Time: 5654.78173828125
LAYER-1; NODES-5
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=5, learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.875      1.         0.78125    1.         0.77419355 0.87096774
 1.         1.         0.77419355 0.83333333]
Cross validation Mean: 0.890893817204301
Cross validation error: 0.10910618279569895
Run Time: 9603.447021484375
LAYER-1; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=10, learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         0.8125     1.         0.90322581 0.83870968
 0.93548387 1.         0.96774194 0.9       ]
Cross validation Mean: 0.9357661290322581
Cross validation error: 0.06423387096774191
Run Time: 13999.67822265625
LAYER-1; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=50, learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.90322581 0.93548387
 1.         1.         0.96774194 0.96666667]
Cross validation Mean: 0.9773118279569892
Cross validation error: 0.022688172043010768
Run Time: 17486.692138671875
LAYER-1; NODES-100
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=100, learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.90322581 0.93548387
 1.         1.         0.96774194 0.96666667]
Cross validation Mean: 0.9773118279569892
Cross validation error: 0.022688172043010768
```

LAYER 2

In [101]:

```
cv_scores_2 = []
cv_errors_2 = []
run_times_2 = []
for n in nodes:
    start_time = time.time()*1000
    #Neural Network Model
    print('LAYER-2; NODES-{}'.format(n))
    perceptron = MLPClassifier(hidden_layer_sizes=(n, n), activation='relu', solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
    print(perceptron)

    #Cross-validation scores
    cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
    cv_mean = np.mean(cv_score)
    print(cv_score)
    cv_scores_2.append(cv_mean)
    print("Cross validation Mean:", cv_mean)

    #Cross-validation errors
    cv_error = 1-cv_mean
    cv_errors_2.append(cv_error)
    print('Cross validation error:', cv_error)
    end_time = time.time()*1000
    runtime = end_time - start_time
    run_times_2.append(runtime)
    print('Run Time:', runtime)
```

```
LAYER-2; NODES-2
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(2, 2), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.65625    0.65625    0.65625    1.         0.67741935 0.67741935
 0.67741935 0.67741935 0.74193548 0.96666667]
Cross validation Mean: 0.7387029569892473
Cross validation error: 0.26129704301075274
Run Time: 6703.74365234375
LAYER-2; NODES-5
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(5, 5), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.90322581 0.90322581
 1.         1.         0.96774194 0.96666667]
Cross validation Mean: 0.9740860215053763
Cross validation error: 0.025913978494623735
Run Time: 16056.54638671875
LAYER-2; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(10, 10), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 0.93548387
 1.         1.         0.96774194 0.96666667]
Cross validation Mean: 0.983763440860215
Cross validation error: 0.016236559139784945
Run Time: 13604.873046875
LAYER-2; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(50, 50), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         0.96774194 1.        ]
Cross validation Mean: 0.9935483870967742
Cross validation error: 0.006451612903225823
Run Time: 11520.964111328125
LAYER-2; NODES-100
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(100, 100), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         0.96774194 1.        ]
Cross validation Mean: 0.9935483870967742
Cross validation error: 0.006451612903225823
```

LAYER 5

In [102]:

```
cv_scores_5 = []
cv_errors_5 = []
run_times_5 = []
for n in nodes:
    start_time = time.time()*1000
    #Neural Network Model
    print('LAYER-5; NODES-{}'.format(n))
    perceptron = MLPClassifier(hidden_layer_sizes=(n, n, n, n, n), activation='relu', solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
    print(perceptron)

    #Cross-validation scores
    cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
    cv_mean = np.mean(cv_score)
    print(cv_score)
    cv_scores_5.append(cv_mean)
    print("Cross validation Mean:", cv_mean)

    #Cross-validation errors
    cv_error = 1-cv_mean
    cv_errors_5.append(cv_error)
    print('Cross validation error:', cv_error)
    end_time = time.time()*1000
    runtime = end_time - start_time
    run_times_5.append(runtime)
    print('Run Time:', runtime)
```

```
LAYER-5; NODES-2
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(2, 2, 2, 2, 2), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         0.65625    1.         0.67741935 0.67741935 0.67741935
 0.67741935 1.         0.67741935 0.66666667]
Cross validation Mean: 0.7710013440860215
Cross validation error: 0.2289986559139785
Run Time: 9167.20263671875
LAYER-5; NODES-5
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(5, 5, 5, 5, 5), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.67741935 0.93548387
 1.         1.         1.         0.66666667]
Cross validation Mean: 0.9279569892473118
Cross validation error: 0.07204301075268815
Run Time: 16507.373291015625
LAYER-5; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(10, 10, 10, 10, 10), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         0.96774194 0.96666667]
Cross validation Mean: 0.9902150537634409
Cross validation error: 0.009784946236559122
Run Time: 9871.62939453125
LAYER-5; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(50, 50, 50, 50, 50), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         1.         1.        ]
Cross validation Mean: 0.9967741935483871
Cross validation error: 0.003225806451612856
Run Time: 11055.3046875
LAYER-5; NODES-100
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(100, 100, 100, 100, 100),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         0.96774194 1.        ]
Cross validation Mean: 0.9935483870967742
```

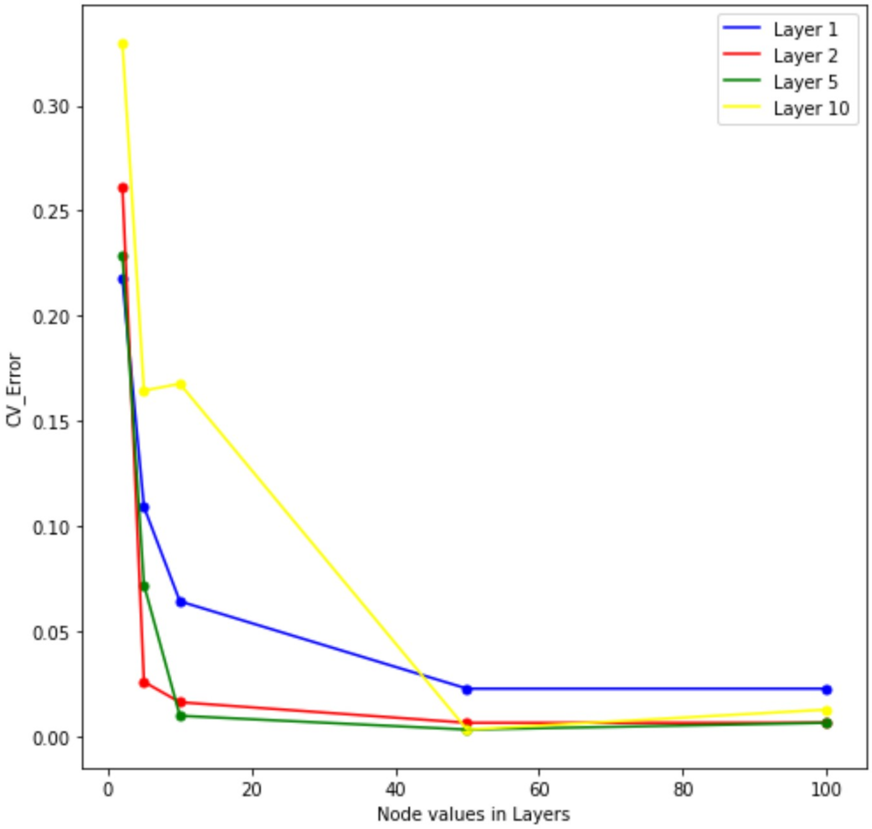
LAYER 10

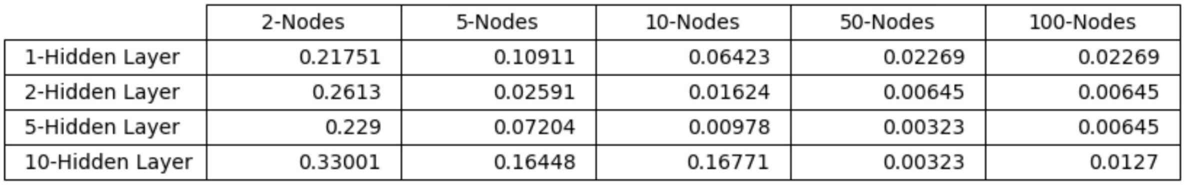
In [103]:

```
cv_scores_10 = []
cv_errors_10 = []
run_times_10 = []
for n in nodes:
    start_time = time.time()*1000
    #Neural Network Model
    print('LAYER-10; NODES-{}'.format(n))
    perceptron = MLPClassifier(hidden_layer_sizes=(n, n, n, n, n, n, n, n, n, n), 
activation='relu', solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
    print(perceptron)

    #Cross-validation scores
    cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
    cv_mean = np.mean(cv_score)
    print(cv_score)
    cv_scores_10.append(cv_mean)
    print("Cross validation Mean:", cv_mean)

    #Cross-validation errors
    cv_error = 1-cv_mean
    cv_errors_10.append(cv_error)
    print('Cross validation error:', cv_error)
    end_time = time.time()*1000
    runtime = end_time - start_time
    run_times_10.append(runtime)
    print('Run Time:', runtime)
```

```
LAYER-10; NODES-2
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(2, 2, 2, 2, 2, 2, 2, 2, 2, 2),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[0.65625    0.65625    0.65625    0.67741935 0.67741935 0.67741935
 0.67741935 0.67741935 0.67741935 0.66666667]
Cross validation Mean: 0.6699932795698925
Cross validation error: 0.3300067204301075
Run Time: 6233.559326171875
LAYER-10; NODES-5
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(5, 5, 5, 5, 5, 5, 5, 5, 5, 5),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[1.         1.         0.65625    1.         0.67741935 0.67741935
 1.         0.67741935 1.         0.66666667]
Cross validation Mean: 0.8355174731182796
Cross validation error: 0.16448252688172038
Run Time: 9835.778076171875
LAYER-10; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(10, 10, 10, 10, 10, 10, 10, 10, 10, 10),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[1.         1.         0.65625    0.67741935 0.96774194 0.67741935
 1.         1.         0.67741935 0.66666667]
Cross validation Mean: 0.8322916666666667
Cross validation error: 0.16770833333333335
Run Time: 7737.563720703125
LAYER-10; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(50, 50, 50, 50, 50, 50, 50, 50, 50, 50),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         1.         1.        ]
Cross validation Mean: 0.9967741935483871
Cross validation error: 0.003225806451612856
Run Time: 12460.551025390625
LAYER-10; NODES-100
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(100, 100, 100, 100, 100, 100, 100, 100, 100,
                                  100),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
In [104]: #plot the data points
          ### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
          fig1 = mp.figure(figsize=(8, 8))

          mp.scatter(nodes,cv_errors_1,s=25, c='blue')
          line1 = mp.plot(nodes,cv_errors_1, label='Layer 1', color='blue')

          mp.scatter(nodes,cv_errors_2,s=25, c='red')
          line2 = mp.plot(nodes,cv_errors_2, label='Layer 2', color='red')

          mp.scatter(nodes,cv_errors_5,s=25, c='green')
          line3 = mp.plot(nodes,cv_errors_5, label='Layer 5', color='green')

          mp.scatter(nodes,cv_errors_10,s=25, c='yellow')
          line4 = mp.plot(nodes,cv_errors_10, label='Layer 10', color='yellow')

          mp.legend(prop={'size': 10})

          #specify the axes
          mp.xlabel("Node values in Layers")
          mp.ylabel("CV_Error")

          #Labeling the plot
          #mp.legend(['1'])
          #mp.legend(legends)
          fig1.suptitle('Figure - CV Errors in HW1 features', fontsize=15)


          #table
          fig = mp.figure()
          table_vals = []

          np_array = np.array(cv_errors_1)
          np_round_to_tenths = np.around(np_array, 5)
          round_to_tenths = list(np_round_to_tenths)
          table_vals.append(round_to_tenths)

          np_array = np.array(cv_errors_2)
          np_round_to_tenths = np.around(np_array, 5)
          round_to_tenths = list(np_round_to_tenths)
          table_vals.append(round_to_tenths)

          np_array = np.array(cv_errors_5)
          np_round_to_tenths = np.around(np_array, 5)
          round_to_tenths = list(np_round_to_tenths)
          table_vals.append(round_to_tenths)

          np_array = np.array(cv_errors_10)
          np_round_to_tenths = np.around(np_array, 5)
          round_to_tenths = list(np_round_to_tenths)
          table_vals.append(round_to_tenths)

          row_labels = ['1-Hidden Layer', '2-Hidden Layer', '5-Hidden Layer', '10-Hidden Lay
          er']
          col_labels = ['2-Nodes', '5-Nodes', '10-Nodes', '50-Nodes', '100-Nodes']
          the_table = mp.table(cellText=table_vals,
                                rowLabels=row_labels,
                                colLabels=col_labels,
                                loc='center')
          the_table.auto_set_font_size(False)
          the_table.set_fontsize(14)
          the_table.scale(2, 2)

          # Removing ticks and spines enables you to get the figure only with table
```

Figure - CV Errors in HW1 features





	2-Nodes	5-Nodes	10-Nodes	50-Nodes	100-Nodes
1-Hidden Layer	0.21751	0.10911	0.06423	0.02269	0.02269
2-Hidden Layer	0.2613	0.02591	0.01624	0.00645	0.00645
5-Hidden Layer	0.229	0.07204	0.00978	0.00323	0.00645
10-Hidden Layer	0.33001	0.16448	0.16771	0.00323	0.0127

In [105]:

```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig1 = mp.figure(figsize=(8, 8))

mp.scatter(nodes,run_times_1,s=25, c='blue')
line1 = mp.plot(nodes,run_times_1, label='Layer 1', color='blue')

mp.scatter(nodes,run_times_2,s=25, c='red')
line2 = mp.plot(nodes,run_times_2, label='Layer 2', color='red')

mp.scatter(nodes,run_times_5,s=25, c='green')
line3 = mp.plot(nodes,run_times_5, label='Layer 5', color='green')

mp.scatter(nodes,run_times_10,s=25, c='yellow')
line4 = mp.plot(nodes,run_times_10, label='Layer 10', color='yellow')

mp.legend(prop={'size': 10})

#specify the axes
mp.xlabel("Node values in Layers")
mp.ylabel("Run Time in milliseconds")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig1.suptitle('Figure - Run time in HW1 features', fontsize=15)


#table
fig = mp.figure()
table_vals = []

np_array = np.array(run_times_1)
np_round_to_tenths = np.around(np_array, 1)
round_to_tenths = list(np_round_to_tenths)
table_vals.append(round_to_tenths)

np_array = np.array(run_times_2)
np_round_to_tenths = np.around(np_array, 1)
round_to_tenths = list(np_round_to_tenths)
table_vals.append(round_to_tenths)

np_array = np.array(run_times_5)
np_round_to_tenths = np.around(np_array, 1)
round_to_tenths = list(np_round_to_tenths)
table_vals.append(round_to_tenths)

np_array = np.array(run_times_10)
np_round_to_tenths = np.around(np_array, 1)
round_to_tenths = list(np_round_to_tenths)
table_vals.append(round_to_tenths)

row_labels = ['1-Hidden Layer', '2-Hidden Layer', '5-Hidden Layer', '10-Hidden Layer']
col_labels = ['2-Nodes', '5-Nodes', '10-Nodes', '50-Nodes', '100-Nodes']
the_table = mp.table(cellText=table_vals,
                     rowLabels=row_labels,
                     colLabels=col_labels,
                     loc='center')
the_table.auto_set_font_size(False)
the_table.set_fontsize(14)
the_table.scale(2, 2)

# Removing ticks and spines enables you to get the figure only with table
```

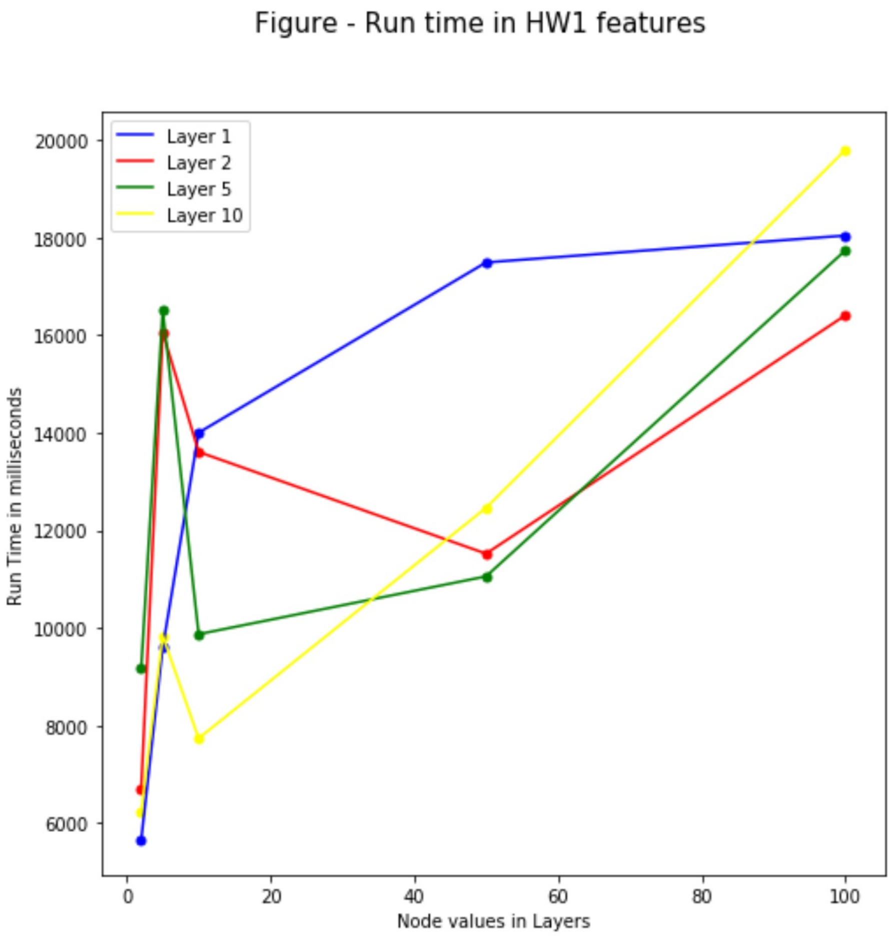


Figure - Run time in HW1 features

	2-Nodes	5-Nodes	10-Nodes	50-Nodes	100-Nodes
1-Hidden Layer	5654.8	9603.4	13999.7	17486.7	18036.6
2-Hidden Layer	6703.7	16056.5	13604.9	11521.0	16402.8
5-Hidden Layer	9167.2	16507.4	9871.6	11055.3	17731.1
10-Hidden Layer	6233.6	9835.8	7737.6	12460.6	19790.6

2a) No, the runtime doesnot have a certain trend with the number of layers and nodes in each layer. 1 hidden layer has a certain kind of effect while 2, 5, 10 hidden layers have similar kind of effect. For 1 Hidden Layer, as the no.of nodes per layer increases, the run-time also increases where runtime is proportional to no.of nodes in each layer. For 2 Hidden Layers, as the no.of nodes per layer increases, the run-time also increases till 5-nodes per layer. Then it decreases at 10, 50-nodes and then starts to increase again. For 5, 10 Hidden layers, as the no.of nodes per layer increases, the run-time also increases till 5-nodes per layer. Then it decreases at 10-nodes and then starts to increase again. But, of every layer individually, the maximum run-time is at 100-nodes per layer. The highest run-time for these 20 models is at 10-Hidden layers & 100-nodes per layer. The lowest run-time for these 20 models is at 1-Hidden layers & 2-nodes per layer.

Finally, the trend for each seperate layer is - It increases as nodes increase, then decrease and start to increase again at some point.

2b) The optimum result i.e., one with the least varying cross validation error is for 5-Hidden layers with 50-nodes per layer and for 10-hidden layers with 50-nodes. If we compare runtime for these 2 models, the runtime is least for 5-hidden layer with 50-nodes. Hence, I consider 5-Hidden layers with 50-nodes per layer to be the optimal model. If we have very less layers with less nodes per layer and if we have very high layers the less nodes per layer the error can be high. Here the error is highest for 10-Hidden layer with 2-nodes per layer than the 2-Hidden Layer with 2-nodes per layer. So, having layers with very less nodes is more susceptible to errors for this dataset. This might be because only one node bears the whole error percentage and when this error is backpropagated only that one node is responsible for the error in that particular layer and during backpropagating, the gradient descent may not be so optimal in finding the local minima with less no.of nodes.

Optimal Model - different learning rates

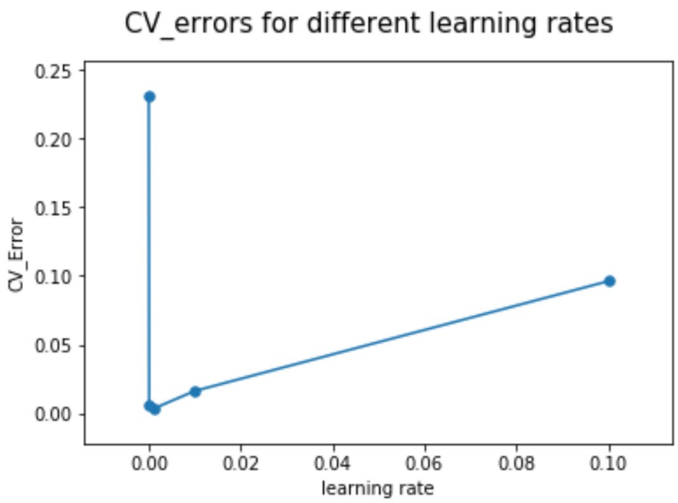
In [126]:

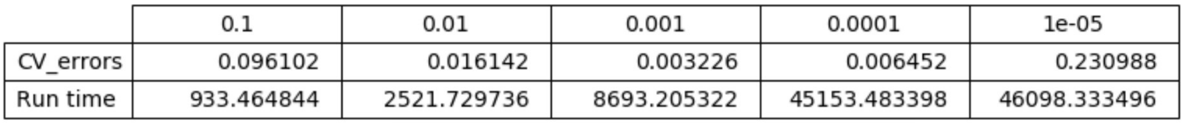
```
lr_cv_errors = []
lrate = [0.1,0.01,0.001,0.0001,0.00001]
lr_run_times = []

for lr in lrate:
    print("learning rate:", lr)
    start_time = time.time()*1000
    #Neural Network Model
    print('LAYER-5; NODES-50'.format(n))
    perceptron = MLPClassifier(hidden_layer_sizes=(50,50,50,50,50), activation='relu', solver='adam', learning_rate_init=lr, alpha=0, max_iter=10000, epsilon=0.001)
    print(perceptron)

    #Cross-validation scores
    cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
    cv_mean = np.mean(cv_score)
    print(cv_score)
    print("Cross validation Mean:", cv_mean)

    #Cross-validation errors
    cv_error = 1-cv_mean
    print('Cross validation error:', cv_error)
    end_time = time.time()*1000
    runtime = end_time - start_time
    print('Run Time:', runtime)
    lr_cv_errors.append(cv_error)
    lr_run_times.append(runtime)
```

```
learning rate: 0.1
LAYER-5; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(50, 50, 50, 50, 50), learning_rate='constant',
              learning_rate_init=0.1, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.96875    0.78125    0.875      1.         0.96774194 0.87096774
 1.         0.96774194 0.77419355 0.83333333]
Cross validation Mean: 0.9038978494623656
Cross validation error: 0.09610215053763438
Run Time: 933.46484375
learning rate: 0.01
LAYER-5; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(50, 50, 50, 50, 50), learning_rate='constant',
              learning_rate_init=0.01, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[0.9375     1.         1.         1.         0.96774194 1.
 1.         1.         1.         0.93333333]
Cross validation Mean: 0.9838575268817206
Cross validation error: 0.016142473118279432
Run Time: 2521.729736328125
learning rate: 0.001
LAYER-5; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(50, 50, 50, 50, 50), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         1.         1.        ]
Cross validation Mean: 0.9967741935483871
Cross validation error: 0.003225806451612856
Run Time: 8693.205322265625
learning rate: 0.0001
LAYER-5; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(50, 50, 50, 50, 50), learning_rate='constant',
              learning_rate_init=0.0001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         0.96774194 1.        ]
Cross validation Mean: 0.9935483870967742
Cross validation error: 0.006451612903225823
Run Time: 45153.4833984375
learning rate: 1e-05
LAYER-5; NODES-50
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(50, 50, 50, 50, 50), learning_rate='constant',
              learning_rate_init=1e-05, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
```

In [132]:

```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig2 = mp.figure()
mp.scatter(lrate,lr_cv_errors,s=30)
mp.plot(lrate,lr_cv_errors)

#specify the axes
mp.xlabel("learning rate")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig2.suptitle('CV_errors for different learning rates', fontsize=15)


#table
#table
fig = mp.figure()
table_vals = []

np_array = np.array(lr_cv_errors)
np_round_to_tenths = np.around(np_array, 6)
round_to_tenths = list(np_round_to_tenths)
table_vals.append(round_to_tenths)

np_array = np.array(lr_run_times)
np_round_to_tenths = np.around(np_array, 6)
round_to_tenths = list(np_round_to_tenths)
table_vals.append(round_to_tenths)

row_labels = ['CV_errors', 'Run time']
col_labels = lrate
the_table = mp.table(cellText=table_vals,
                      rowLabels=row_labels,
                      colLabels=col_labels,
                      loc='center')
the_table.auto_set_font_size(False)
the_table.set_fontsize(14)
the_table.scale(2, 2)

# Removing ticks and spines enables you to get the figure only with table
mp.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
mp.tick_params(axis='y', which='both', right=False, left=False, labelleft=False)
for pos in ['right','top','bottom','left']:
    mp.gca().spines[pos].set_visible(False)
mp.savefig('matplotlib-table.png', bbox_inches='tight', pad_inches=0.05)

#display the current graph
show()
```





	0.1	0.01	0.001	0.0001	1e-05
CV_errors	0.096102	0.016142	0.003226	0.006452	0.230988
Run time	933.464844	2521.729736	8693.205322	45153.483398	46098.333496

2c) I have observed here that for very high and low learning rates, the errors are higher. Here as the learning rate decreases, error decreases, but if the learning rate decreases too much, then the error starts to increase again. i.e., for very low learning rates, within the given no.of iterations, the model was not able to find the optimal local minima and similary for high learning rates, the model might have taken huge steps and the local minima might be missed and wasn't able to converge in the no.of iterations given.

It is obvious that as learning rate increases, run time increases as the model takes very small steps to converge to the minimum point. We can observe that it is not necessary for accuracy to increase as the runtime increases. Here we can see that runtime is highest for lowest learning rate and the accuracy is lowest for it. But if we consider moderate learning rate the runtime is also moderate but the accuracy is highest. Therefore, the relationship between run-time and accuracy follows a inverted parabola trend. We can also see that layers with high no.of nodes per layer have high run-time. But it is not the case that they have the highest accuracy. Accuracy increases till certain point with increase in run-time but after a certain point/threshold, even if the run-time is high, the accuracy may not be high.

2d) The neural network is not producing the same results every time. That is because it uses random weights each time to start, as the weights differ, the results like cross_validation scores and errors also differ. Yes, this might have a small impact on the expected fit because, the croos validation errors are varying hugely from 0.003 to 0.016 for some cases that I've seen.

GRADUATE STUDENT QUESTION

Early-stopping = 'FALSE' - 5 LAYERS

In [25]:

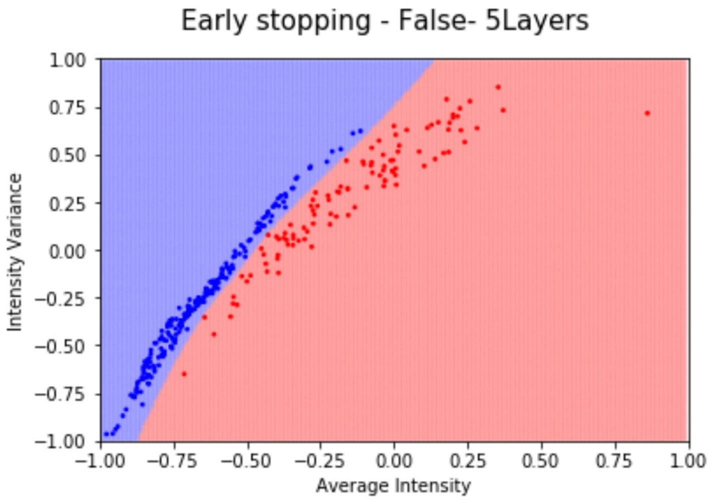
```
start_time = time.time()*1000
#Neural Network Model
print('LAYER-5; NODES-10'.format(n))
perceptron = MLPClassifier(hidden_layer_sizes=(100,100,100,100,100), activation='re
lu', early_stopping=False, solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
print(perceptron)

#Cross-validation scores
cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
cv_mean = np.mean(cv_score)
print(cv_score)
print("Cross validation Mean:", cv_mean)

#Cross-validation errors
cv_error = 1-cv_mean
print('Cross validation error:', cv_error)
end_time = time.time()*1000
runtime = end_time - start_time
print('Run Time:', runtime)
```

```
LAYER-5; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(100, 100, 100, 100, 100),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[1.         1.         1.         1.         0.96774194 1.
 1.         1.         1.         1.        ]
Cross validation Mean: 0.9967741935483871
Cross validation error: 0.003225806451612856
Run Time: 12887.58642578125
```

In [26]:

```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = MLPClassifier(hidden_layer_sizes=(100,100,100,100,100), activation='relu',
early_stopping=False, solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

```
In [27]: ## Visualize Results
         #plot the points
         fig = mp.figure()
         mp.scatter(X,Y,s=3,c=colors)

         #plot the regions
         mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

         #setup the axes
         mp.xlim(-1,1)
         mp.xlabel("Average Intensity")
         mp.ylim(-1,1)
         mp.ylabel("Intensity Variance")

         #Label the figure
         fig.suptitle('Early stopping - False- 5Layers', fontsize=15)
         show()
```



EARLY STOPPING TRUE (5-LAYERS)

In [30]:

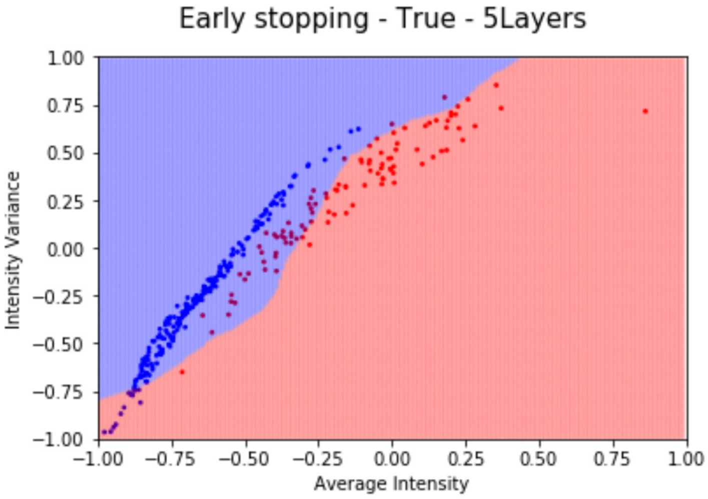
```
start_time = time.time()*1000
#Neural Network Model
print('LAYER-5; NODES-10'.format(n))
perceptron = MLPClassifier(hidden_layer_sizes=(100,100,100,100,100), activation='re
lu', early_stopping=True, solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
print(perceptron)

#Cross-validation scores
cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
cv_mean = np.mean(cv_score)
print(cv_score)
print("Cross validation Mean:", cv_mean)

#Cross-validation errors
cv_error = 1-cv_mean
print('Cross validation error:', cv_error)
end_time = time.time()*1000
runtime = end_time - start_time
print('Run Time:', runtime)
```

```
LAYER-5; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=True, epsilon=0.001,
              hidden_layer_sizes=(100, 100, 100, 100, 100),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[0.9375     0.65625    0.65625    0.67741935 0.67741935 0.74193548
 0.67741935 0.67741935 0.67741935 0.8       ]
Cross validation Mean: 0.7179032258064516
Cross validation error: 0.2820967741935484
Run Time: 937.270263671875
```

In [28]:

```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = MLPClassifier(hidden_layer_sizes=(100,100,100,100,100), activation='relu',
early_stopping=True, solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

```
In [29]: ## Visualize Results
         #plot the points
         fig = mp.figure()
         mp.scatter(X,Y,s=3,c=colors)

         #plot the regions
         mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

         #setup the axes
         mp.xlim(-1,1)
         mp.xlabel("Average Intensity")
         mp.ylim(-1,1)
         mp.ylabel("Intensity Variance")

         #Label the figure
         fig.suptitle('Early stopping - True - 5Layers', fontsize=15)
         show()
```



EARLY STOPPING FALSE (10-LAYERS)

In [31]:

```
start_time = time.time()*1000
#Neural Network Model
print('LAYER-5; NODES-10'.format(n))
perceptron = MLPClassifier(hidden_layer_sizes=(100,100,100,100,100,100,100,100,100,
100), activation='relu', early_stopping=False, solver='adam', alpha=0, max_iter=100
00, epsilon=0.001)
print(perceptron)

#Cross-validation scores
cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
cv_mean = np.mean(cv_score)
print(cv_score)
print("Cross validation Mean:", cv_mean)

#Cross-validation errors
cv_error = 1-cv_mean
print('Cross validation error:', cv_error)
end_time = time.time()*1000
runtime = end_time - start_time
print('Run Time:', runtime)
```

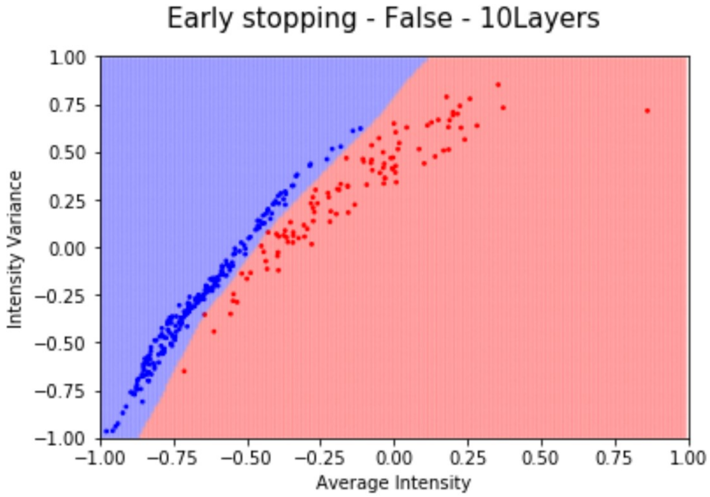
```
LAYER-5; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=0.001,
              hidden_layer_sizes=(100, 100, 100, 100, 100, 100, 100, 100, 100,
                                  100),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[1.         1.         0.96875    1.         0.96774194 1.
 1.         0.96774194 1.         0.96666667]
Cross validation Mean: 0.9870900537634408
Cross validation error: 0.012909946236559167
Run Time: 17292.814208984375
```

In [32]:

```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = MLPClassifier(hidden_layer_sizes=(100,100,100,100,100,100,100,100,100,100),
activation='relu', early_stopping=False, solver='adam', alpha=0, max_iter=10000, ep
silon=0.001)
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

In [33]:

```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('Early stopping - False - 10Layers', fontsize=15)
show()
```



EARLY STOPPING TRUE (10-LAYERS)

In [34]:

```
start_time = time.time()*1000
#Neural Network Model
print('LAYER-5; NODES-10'.format(n))
perceptron = MLPClassifier(hidden_layer_sizes=(100,100,100,100,100,100,100,100,100,
100), activation='relu', early_stopping=True, solver='adam', alpha=0, max_iter=1000
0, epsilon=0.001)
print(perceptron)

#Cross-validation scores
cv_score = cross_val_score(perceptron, simpleTrain, trainDigits, cv=10)
cv_mean = np.mean(cv_score)
print(cv_score)
print("Cross validation Mean:", cv_mean)

#Cross-validation errors
cv_error = 1-cv_mean
print('Cross validation error:', cv_error)
end_time = time.time()*1000
runtime = end_time - start_time
print('Run Time:', runtime)
```

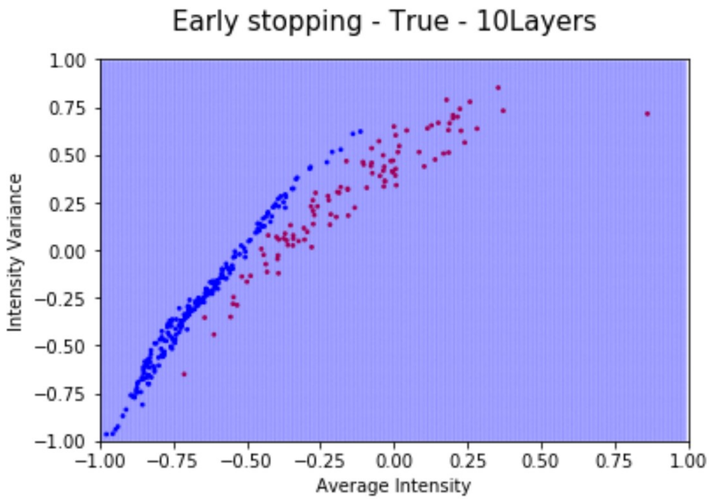
```
LAYER-5; NODES-10
MLPClassifier(activation='relu', alpha=0, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=True, epsilon=0.001,
              hidden_layer_sizes=(100, 100, 100, 100, 100, 100, 100, 100, 100,
                                  100),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=10000, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)
[0.65625    0.65625    0.65625    0.67741935 0.67741935 0.67741935
 0.67741935 0.67741935 0.67741935 0.66666667]
Cross validation Mean: 0.6699932795698925
Cross validation error: 0.3300067204301075
Run Time: 1671.4755859375
```

In [35]:

```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

# Declare Model
model = MLPClassifier(hidden_layer_sizes=(100,100,100,100,100,100,100,100,100,100), activation='relu', early_stopping=True, solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

In [36]:

```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('Early stopping - True - 10Layers', fontsize=15)
show()
```



2e) I have tried for different layers with high nodes 50 & 100 even though all of them are not shown here. Here are my conclusions, Early stopping makes the model to underfit the data. If early_stopping="True", the model will automatically stop training once the validation score stops improving. i.e., if the validation scores stops improving it might assume that the data is scattered/arranged in the similar way like before and assumes no new data present and training stops. So, this will make the model to underfit the data as training is not performed completely on the data.

In [ ]:

```
2D REGION FR OPTIMAL NEURAL NETWORK
```

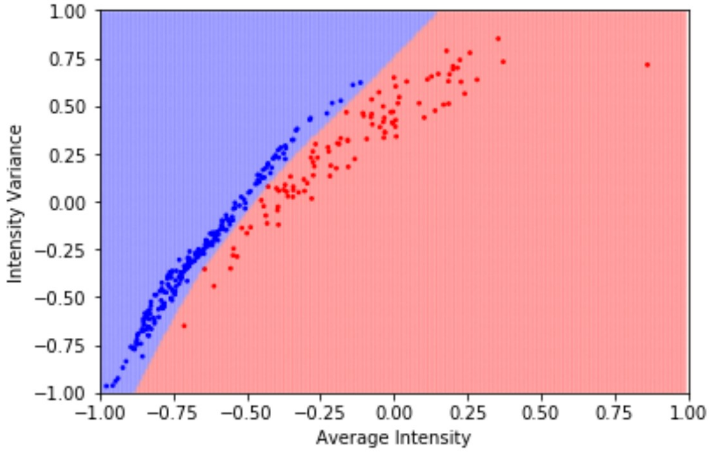
In [155]:

```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

# Declare Model
model = MLPClassifier(hidden_layer_sizes=(50,50,50,50,50), activation='relu', early_stopping=False, solver='adam', alpha=0, max_iter=10000, epsilon=0.001)
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

In [156]:

```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('Fig 3.1 - Optimal Neural Network', fontsize=15)
show()
```

Fig 3.1 - Optimal Neural Network



The optimal neural network that I have drawn is for 5 hidden layers with 50 nodes each. This is the most stable model where the cross validation errors are low and less varying. This is the model that also has lowest runtime after the highest accuracy. Also, from the figure we can see that the data seperated more accurately without overfittig the model.

In [ ]: