Tasks:

- ### SGEMM GPU Kernel Performance Prediction

  The emergence of programmable graphics hardware has led to increasing interest in offloading numerically intensive computations to GPUs. This has made GPUs very prevalent in this era of high performance computing, but GPU programming remains challenging and it would therefore be handy to have a model which captures the main performance factors of GPU kernels that helps to estimate the run time, which could also potentially be used to identify its bottlenecks.

- ### Income Prediction

  Today's millennials are looking for simple and flexible payment solution that suits their financial situation without forcing them into one-sided credit agreements with unfair interest rates. Fintech companies' answer to that is the Pay-Later feature.
  However, there is a major risk involved here in guessing who will default on borrowings. And since not all the growing startups have access to the FICO score of an individual, it can be difficult to predict their repaying capability. So, small fintech companies try to come up with a novel way to evaluate the potential risk posed by lending money to consumers and then set the credit limits accordingly. One of the main parameters that could help us understand the creditworthiness of an individual is their salary. Therefore, in this dataset, we try to predict the income of the individual from basic demographic data and other parameters which can be easily obtained online or from any of the online merchants.

In this project, the objectives are to implement Neural Network and KNN to predict GPU performance and salary of an individual.

TASK 1 – GPU Dataset

Dataset Description:

1. SGEMM GPU kernel performance dataset

   This data set measures the running time of a matrix-matrix product A*B = C, where all matrices have size 2048 x 2048, using a parameterizable SGEMM GPU kernel with 241600 possible parameter combinations. It has **4 dependent** variables and **14 independent** variables - the first 10 are ordinal and can only take up to 4 different powers of two values, and the 4 last variables are binary.

Exploratory Data Analysis:

Refer the code for EDA of GPU dataset. Please note that EDA and Data Preparation part of GPU dataset is already explained in Assignment 1.

Data Preparation:

The data has been normalized by subtracting each value with its mean and divide by its standard deviation. The dataset is split into train and test set in the ratio of 70:30.

Already explained in detail in Assignment 1 and 2.

Experimentation:

Here, the target value must be considered as a binary value since all the below algorithms is run for variable classification. This is done by considering all the values of average runtime **above** the **median as 1** (this means computation takes more time) and **below the median as 0** (this means computation takes lesser time).

This dataset consists of all the possible combination of GPU kernel parameters. Since it represents the complete set of data for the given GPU configuration, it is safe to assume that anything above the median value as high time consuming.

Neural Network:

In this project, Keras has been used to build neural network models. It is TensorFlow's high level API for building and training models. It is extremely user-friendly and easy to extend. Models using different activation functions and optimizers have been created and compared. The model type used in all these models are **Sequential** which allows us to build layer by layer. Functional API is not used since there is no need to create complex network in this project. **Dense** is the layer type used. It connects all the nodes in the previous layer to the nodes in the current layer.

The metrics used for evaluation of the models are as follows:

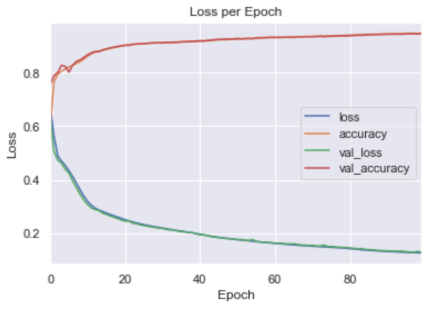
Confusion matrix, Classification chart, Training and Test accuracy

Epoch: 100, Callback with **EarlyStopping** has been applied which will terminate the training process if validation loss has stopped improving for 10 epochs.

Sigmoid Activation: *SGD Optimizer*

Layer (type)	Output Shape	Param #
Layer1 (Dense)	(None, 14)	210
Layer2 (Dense)	(None, 500)	7500
Prediction (Dense)	(None, 1)	501

Total params: 8,211



```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     34376
           1       0.96      0.94      0.95     34480

    accuracy                           0.95     68856
   macro avg       0.95      0.95      0.95     68856
weighted avg       0.95      0.95      0.95     68856

Accuracy: 0.9480219588706866
Confusion Matrix:
 [[32954  1422]
 [ 2157 32323]]
```

Wall time: 7 mins 22 secs

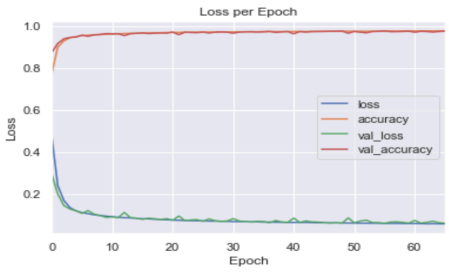
Adam Optimizer

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     34376
           1       0.98      0.96      0.97     34480

    accuracy                           0.97     68856
   macro avg       0.97      0.97      0.97     68856
weighted avg       0.97      0.97      0.97     68856

Accuracy: 0.9684849541071221
Confusion Matrix:
 [[33596   780]
 [ 1390 33090]]
```

Wall time: ~8 mins

ReLU Activation: *SGD Optimizer*

```
Layer (type)                 Output Shape              Param #
=================================================================
Layer1 (Dense)               (None, 14)                210
_________________________________________________________________
Layer2 (Dense)               (None, 28)                420
_________________________________________________________________
Layer3 (Dense)               (None, 28)                812
_________________________________________________________________
Layer4 (Dense)               (None, 40)                1160
_________________________________________________________________
Layer5 (Dense)               (None, 60)                2460
_________________________________________________________________
Prediction (Dense)           (None, 1)                 61
=================================================================
Total params: 5,123
```

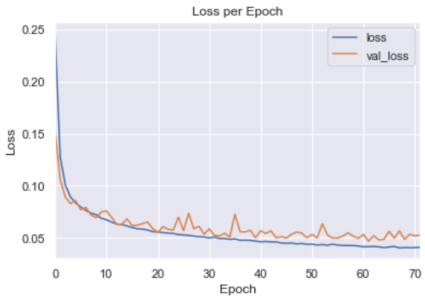



```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     34376
           1       0.99      0.96      0.97     34480

    accuracy                           0.97     68856
   macro avg       0.97      0.97      0.97     68856
weighted avg       0.97      0.97      0.97     68856

Accuracy: 0.9740037179040316
Confusion Matrix:
 [[33872   504]
 [ 1286 33194]]
```

Wall time: 4 mins 55 secs

Adam Optimizer

```
160664/160664 [==============================] - 3s 18us/sample - loss: 0.0429 - accuracy: 0.9821
Train accuracy:  0.9820744
68856/68856 [==============================] - 1s 18us/sample - loss: 0.0536 - accuracy: 0.9796
Test accuracy:  0.9795951

Accuracy: 0.9795950970140583
Confusion Matrix:
 [[33928   448]
 [  957 33523]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     34376
           1       0.99      0.97      0.98     34480

    accuracy                           0.98     68856
   macro avg       0.98      0.98      0.98     68856
weighted avg       0.98      0.98      0.98     68856
```



Wall time: 5 mins 34 secs

Leaky ReLU Activation: *Adam Optimizer*

Layer (type)	Output Shape	Param #
Layer1 (Dense)	(None, 14)	210
activation (Activation)	(None, 14)	0
Layer2 (Dense)	(None, 28)	420
activation_1 (Activation)	(None, 28)	0
Layer3 (Dense)	(None, 40)	1160
activation_2 (Activation)	(None, 40)	0
Layer4 (Dense)	(None, 60)	2460
activation_3 (Activation)	(None, 60)	0
Prediction (Dense)	(None, 1)	61

Total params: 4,311

```
Accuracy: 0.9777070988730103
Confusion Matrix:
 [[33850   526]
 [ 1009 33471]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     34376
           1       0.98      0.97      0.98     34480

    accuracy                           0.98     68856
   macro avg       0.98      0.98      0.98     68856
weighted avg       0.98      0.98      0.98     68856
```

Wall time: 2 mins 55 secs

Tanh Activation: *Adam Optimizer*

Layer (type)	Output Shape	Param #
Layer1 (Dense)	(None, 14)	210
Layer2 (Dense)	(None, 28)	420
Layer3 (Dense)	(None, 28)	812
Layer4 (Dense)	(None, 40)	1160
Layer5 (Dense)	(None, 60)	2460
Prediction (Dense)	(None, 1)	61

Total params: 5,123

```
Accuracy: 0.9706343673753921
Confusion Matrix:
 [[33920   456]
 [ 1566 32914]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     34376
           1       0.99      0.95      0.97     34480

    accuracy                           0.97     68856
   macro avg       0.97      0.97      0.97     68856
weighted avg       0.97      0.97      0.97     68856
```

Wall time: 7 mins 15 secs

ReLU has an upper hand over other activation functions since it does not have the **vanishing gradient** problem during back propagation. Also, it has lesser run time.

Leaky ReLU adds a slight slope in the negative range to prevent the **dying ReLU** issue. Hence both these activation functions are included

For each of the above cases, the number of the layers/neurons used were arrived after trying with different layer count / number of neurons and choosing the best combination out of it.

KNN:

KNN models were built for different k values. Uniform weights have been used in this case since distance and uniform both gave similar results. Error rate vs K Value is plotted to find the best k value.

Training Accuracy: After cross-fold validation with cv = 5

```
Scores:  [0.86820403 0.87240532 0.86795506 0.86717705 0.86897797]
Accuracy: 0.87 (+/- 0.00)
```

Test Accuracy:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88     34376
           1       0.89      0.87      0.88     34480

    accuracy                           0.88     68856
   macro avg       0.88      0.88      0.88     68856
weighted avg       0.88      0.88      0.88     68856

AUC Score:
 0.8827579417896368
```

```
Accuracy: 0.8827407923782967
Confusion Matrix:
 [[30736  3640]
 [ 4434 30046]]
```

TASK 2

Dataset Description:

2. Income dataset

   There are 14 independent variables and 1 dependent variable – salary which is classified into 2 categories: <=50K and >50K

Exploratory Data Analysis:

Refer the code for EDA of Income dataset. Please note that EDA part of Income dataset is already explained in Assignment 1.

Data Preparation:

Refer the code for EDA of Income dataset and description of GPU data preparation. Please note that EDA part of Income dataset is already explained in detail in Assignment 2.

Experimentation:

Here, the target variable consists of 2 values - <=50K which is considered as 0 and >50K as 1 for the analysis purpose.

Neural Network:

Model type and layer type is same as the previous dataset. The metrics used for evaluation of the models are as follows: **Confusion matrix, Classification chart, Training and Test accuracy**

Epoch: 100, Callback with **EarlyStopping** has been applied which will terminate the training process if validation loss stops improving for 10 epochs.

Two optimizers were used: **SGD** and **Adam**. Since Adam gave better results, its output is displayed below

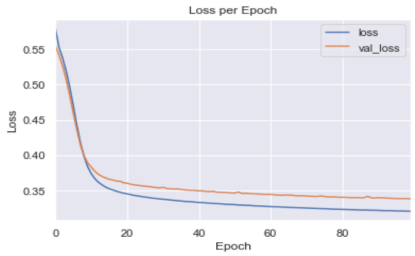
Sigmoid Activation

Number of Hidden Layers: 2

Total params: 8128

Wall time: 24.4 secs

Training Accuracy: 86.38%




```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      6150
           1       0.75      0.58      0.65      2102

    accuracy                           0.84      8252
   macro avg       0.81      0.75      0.77      8252
weighted avg       0.84      0.84      0.84      8252
```

```
Accuracy: 0.8425836160930683
Confusion Matrix:
 [[5743  407]
 [ 892 1210]]
```

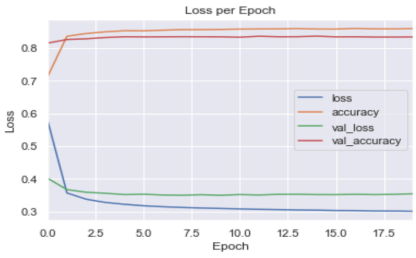
ReLU Activation:

Number of Hidden Layers: 2

Total params: 701

Wall time: 15.2 secs

Training Accuracy: 85.36%




```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      6150
           1       0.72      0.63      0.67      2102

    accuracy                           0.84      8252
   macro avg       0.80      0.77      0.78      8252
weighted avg       0.84      0.84      0.84      8252
```

```
Accuracy: 0.842462433349491
Confusion Matrix:
 [[5635  515]
 [ 785 1317]]
```

Leaky ReLU Activation:

Wall time: 13.7 secs

```
Confusion Matrix:
 [[5737  413]
 [ 895 1207]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      6150
           1       0.75      0.57      0.65      2102

    accuracy                           0.84      8252
   macro avg       0.81      0.75      0.77      8252
weighted avg       0.83      0.84      0.83      8252
```

Tanh Activation:

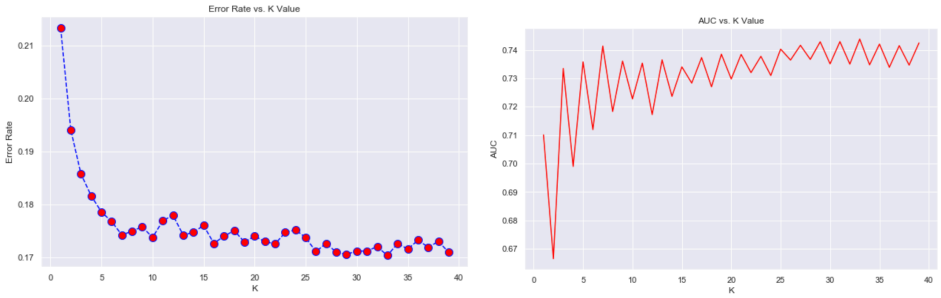
Wall time: 11.6 secs

```
Accuracy: 0.840159961221522
Confusion Matrix:
 [[5627  523]
 [ 796 1306]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.91      0.90      6150
           1       0.71      0.62      0.66      2102

    accuracy                           0.84      8252
   macro avg       0.80      0.77      0.78      8252
weighted avg       0.83      0.84      0.84      8252
```

From the loss per epoch graph, it can be seen that accuracy gets saturated after a point, after which there is only a minor decrease in error. Also, it can be seen the val_loss starts to increase after some time, at which the process is stopped.

KNN:

KNN models were built for different k values. Uniform weights have been used in this case since both distance and uniform gave similar results.



Training Accuracy: After cross-fold validation with cv = 5

```
Scores:  [0.82913529 0.82030641 0.82753247 0.82337662 0.81688312]
Accuracy: 0.82 (+/- 0.01)
```

Test Accuracy:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      6150
           1       0.72      0.53      0.61      2102

    accuracy                           0.83      8252
   macro avg       0.79      0.73      0.75      8252
weighted avg       0.82      0.83      0.82      8252

AUC Score:
 0.7283612974093585
```

```
Accuracy: 0.8273145904023267
Confusion Matrix:
 [[5720  430]
 [ 995 1107]]
```

CONCLUSION:

GPU Dataset:

Following are the accuracy and test errors of the different models:

Models	Total Params	Hidden Layers	Wall time(sec)	Train Accuracy	Test Accuracy	Error
NN - Sigmoid, SGD	8211	2	444	94.84%	94.80%	0.05
NN - Sigmoid, Adam	8211	2	480	96.91%	96.80%	0.03
NN - ReLU, SGD	5123	5	295	97.62%	97.40%	0.03
NN - ReLU, Adam	5123	5	334	98.20%	97.95%	0.02
NN - Leaked ReLU, Adam	4311	4	175	97.91%	97.70%	0.02
NN - tanh, SGD	5123	5	435	97.60%	97.50%	0.03
NN - tanh, Adam	5123	5	191	97.20%	97.06%	0.03
KNN - K=1 *			11		85.60%	0.14

KNN - K=5 *			20	87.00%	88.20%	0.12
SVM - Gaussian *			516		97.62%	0.02
Decision Trees			28		99.04%	0.01
Boosting - AdaBoost *			7		79.32%	0.21

* The total time taken to run these models are higher since this wall time does not include the time taken in grid search.

Of the 5 different algorithms used, Decision Trees gave the best results for this dataset. SVM with Gaussian Kernel and Neural Network with ReLU Activation function and Adam optimizer also gave an accuracy of 98%.

Income Dataset:

Following are the accuracy and test errors of the different models:

Models	Total Params	Hidden Layers	Wall time(sec)	Train Accuracy	Test Accuracy	Error
NN - Sigmoid, SGD	8128	2	60	84.69%	84.33%	0.16
NN - Sigmoid, Adam	8128	2	24	86.38%	84.35%	0.16
NN - ReLU, SGD	801	2	34	85.36%	84.11%	0.16
NN - ReLU, Adam	801	2	15	85.36%	84.24%	0.16
NN - Leaked ReLU, Adam	731	3	13	85.53%	84.10%	0.16
NN - tanh, SGD	717	3	33	85.43%	84.30%	0.16
NN - tanh, Adam	717	3	11	85.38%	84.01%	0.16
KNN - K=1 *			11		78.65%	0.21
KNN - K=5 *			12	82.40%	82.70%	0.17
SVM - Gaussian *			8		84.92%	0.15
Decision Trees			1		84.66%	0.15
Boosting - AdaBoost *			10		86.23%	0.14

* The total time taken to run these models are higher since this wall time does not include the time taken in grid search.

Of the 5 main different algorithms used, Boosting gave the best results for this dataset since AdaBoost identifies misclassified data points, increasing their weights so that the next classifier will pay extra attention to get them right. Most of the other algorithms also had accuracy of ~85%. NN did not give the best possible result since we have just built a simple neural network model.

Also, though ~85% accuracy is decent enough, we don't know if this dataset covers all the states of USA. So, we may not be sure if this model would accurately predict the income of any individual from the United States. It is possible to get better results for the real-time data if the sample represent the whole population.

Apart from that, the above accuracy can itself be improved by implementing complex neural network models with shared layers. Adding dropout layer and usage of ensemble methods are other ways to improve the accuracy.