University of Salford, MSc Data Science

Module: Machine Learning & Data Mining

Session: Workshop Week 6

Topic: Introduction to Neural Networks with Keras

Tools: Google Colaboratory

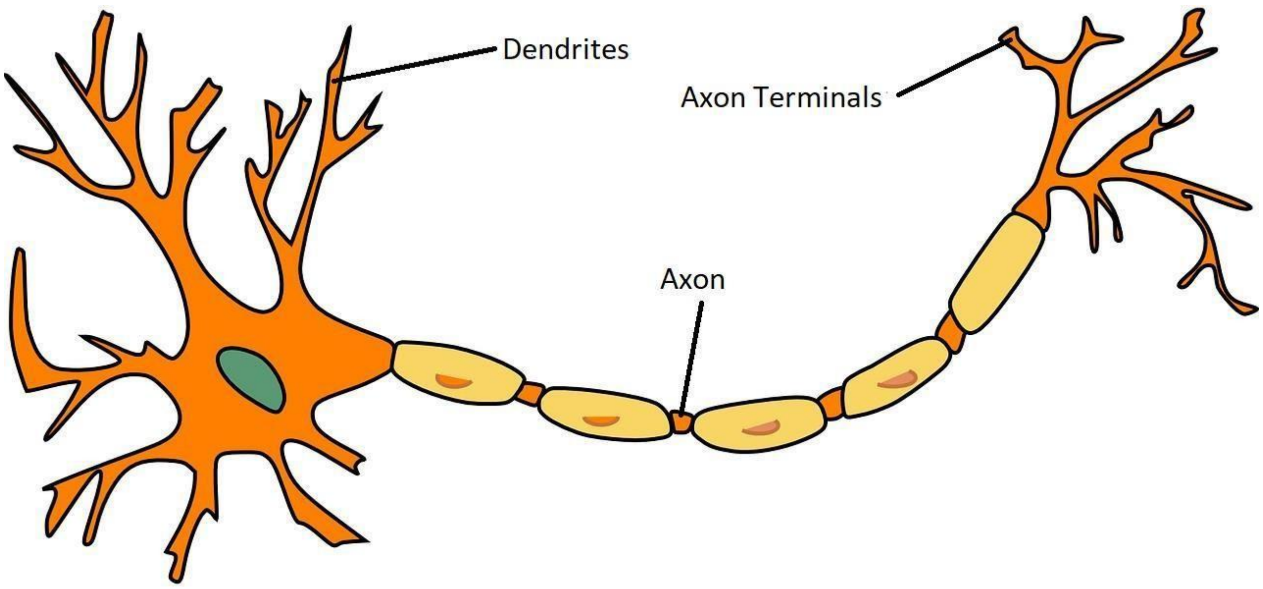
Objectives:

After completing this workshop, you will be able to:

- Use Google Colab and make use of a GPU for running your code
- Use Keras to build and train a neural network
- Develop neural network with appropriate parameters
- Analyse neural network performance using common metrics including accuracy, precision, recall and F1 score.

Introduction:

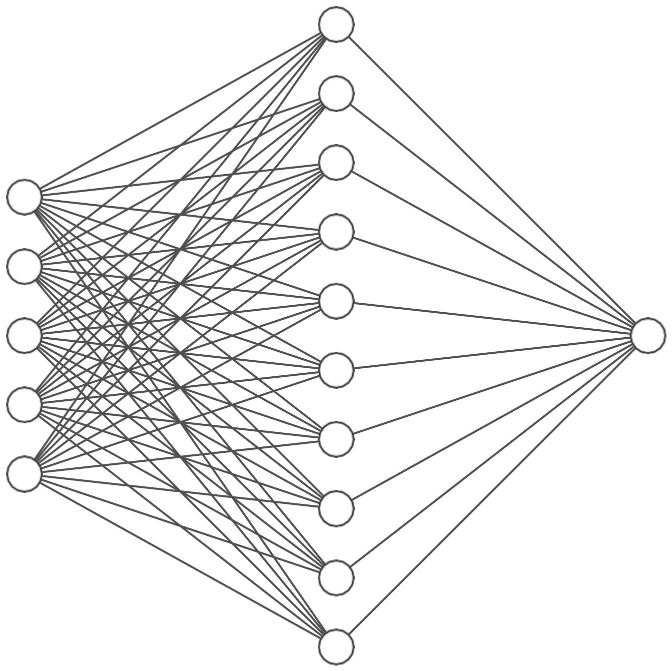
Neural networks are a class of machine learning models which have proved successful in tackling a wide range of classification and regression problems and lie at the heart of many recent advances in machine learning and artificial intelligence. Neural networks are inspired by the structure of neurons in the brain and nervous system as illustrated below. The brain is formed by billions of highly interconnected neurons, which interact in complex networks.



In a feedforward artificial neural network, neurons are connected sequentially, in a series of layers, with neurons connected to those in the layers next to them. The simplest form of neural network consists of three layers; an input layer, a hidden layer, and an output layer, as shown on the next page. The first layer is the input layer and is used to input the features into the network for a given example in the dataset. Each input neuron corresponds to one feature (or variable) contained in the data.

The neurons in the hidden layer are connected to those in the input layer and each connection has a ‘weight’ associated with it. The value of the input is multiplied by the weight associated with the connection and all these values are added together and the result is put through an activation function. This value is the output which is passed through to the next layer in the network. Through successive layers of the network, it can learn increasingly complex combinations of the input variables which can then be used in the final layer to make a prediction.

We train the neural network using our training dataset, and as we train the network the weights of the neurons are updated to improve the performance of the network.



A simple Neural Network with one hidden layer. Diagram created using NN-SVG

Introducing Google Colab:

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute Python/R code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources, including GPUs.

Introducing Tensorflow & Keras:

TensorFlow is a free and open-source software library for machine learning and artificial intelligence and is particularly well suited for building and training deep neural networks. It was developed by the Google Brain team for internal use at Google but was released in 2015. TensorFlow offers multiple levels of abstraction. The high-level Keras API offers a very easy and intuitive way to build and train models, and this is what we will be using in this workshop.

Introducing the Dataset & Scenario:

The file for today's workshop is saved on Blackboard as **Cardiotocographic.csv**. It has the following variables.

BPM	Beat per minutes
APC	Accelerations per second
FMPS	Fetal movement per second
UCPS	Uterine contractions per second
DLPS	Light declaration per second
SDPS	Severe declaration per second
PDPS	Prolonged declaration per second
ASTV	% of abnormal short term Variability
MSTV	Mean of short term Variability
ALTV	% of abnormal long term Variability
MLTV	Mean of long term Variability

Width	Width of FHR Histogram
Min	Min Width of FHR Histogram
Max	Max Width of FHR Histogram

NSP	Fetal State Class code
	N=normal (1);
	S=Suspect (2);
	P=Pathologic (3)

Cardiotocography is a recording of the fetal heart rate obtained by ultrasound and is used in pregnancy to assess fetal well-being. We therefore want to use these variables to predict the target variable (NSP), which is the Fetal State Class code - or in other words, the diagnosis.

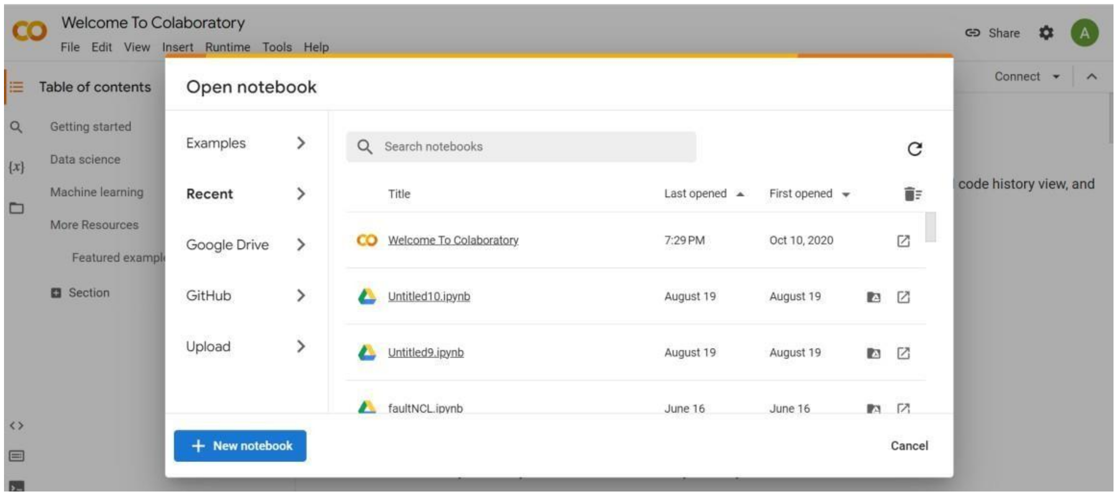
There are three values this can take:

- N = Normal (1)
- S = Suspect (2)
- P = Pathologic (3)

A machine learning model which can predict diagnosis based on the cardiotocographic data could aid clinicians in making an accurate diagnosis. The aim of today's workshop is to investigate whether we can build an artificial neural network to support clinicians in screening pregnancies to identify those which potentially have issues so that they can receive further medical attention.

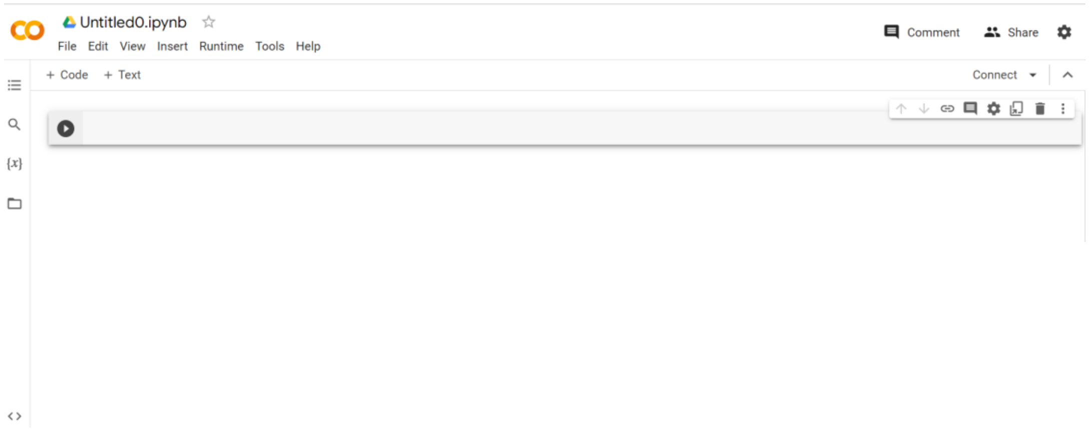
Part One: Creating the Notebook, Exploring and Pre-processing the Data

1) To use Google Colab, you will need a Google Account. If you do not have one, you will need to set one up first. Login to your Google Account, and once you are logged in, follow this link.



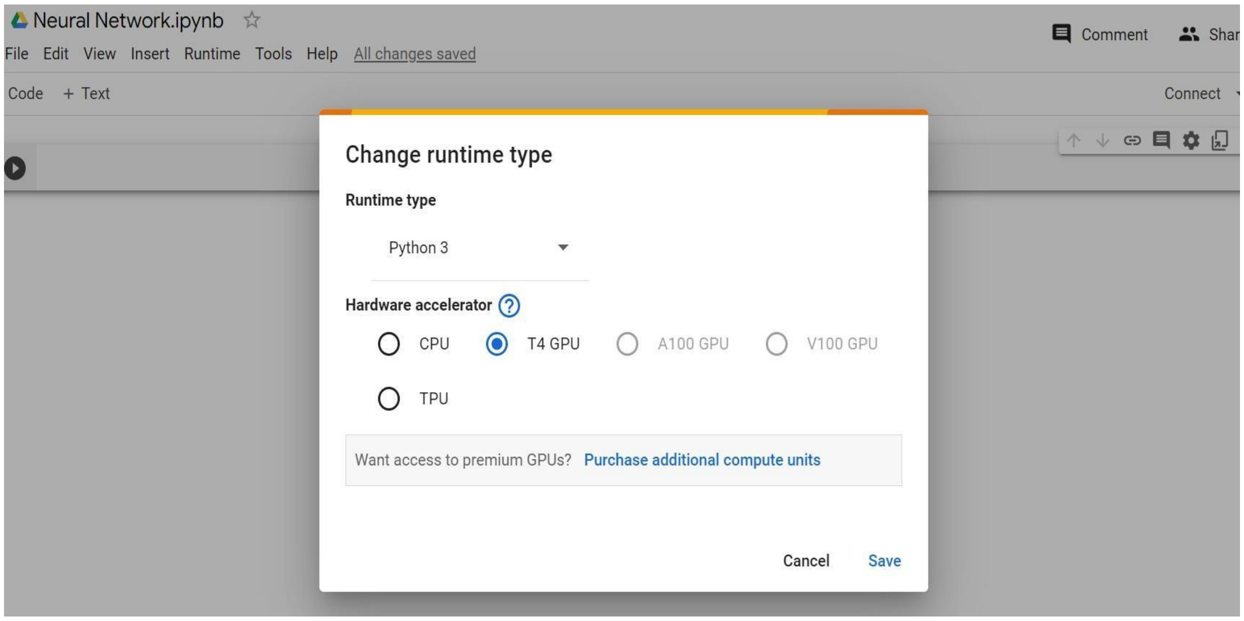
Title	Last opened	First opened
Welcome To Colaboratory	7:29 PM	Oct 10, 2020
Untitled10.ipynb	August 19	August 19
Untitled9.ipynb	August 19	August 19
faultNCL.ipynb	June 16	June 16

Click on the New notebook. It will open a new notebook like the image below:



This will open a new notebook, which you can rename by clicking on the file name to the top of the screen (It will have defaulted to 'Untitled0').

2) Click Runtime and select Change Runtime Type and then set the Hardware Accelerator to T4 GPU and click Save, as shown below.



Change runtime type

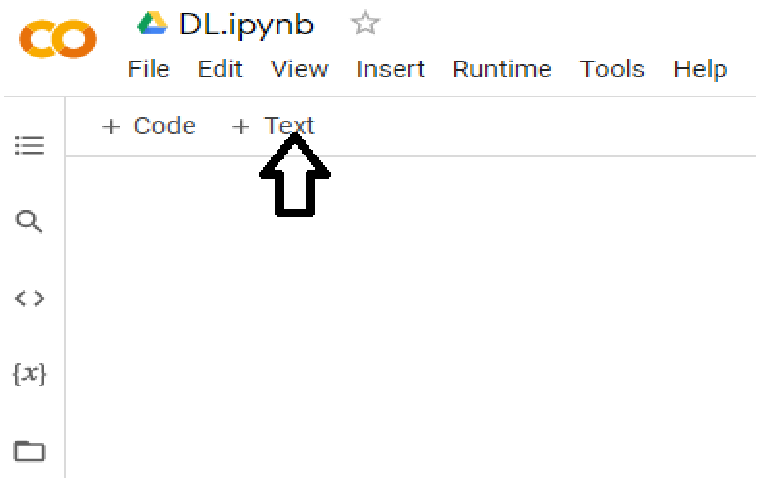
Runtime type: Python 3

Hardware accelerator: CPU [✓ T4 GPU] A100 GPU V100 GPU TPU

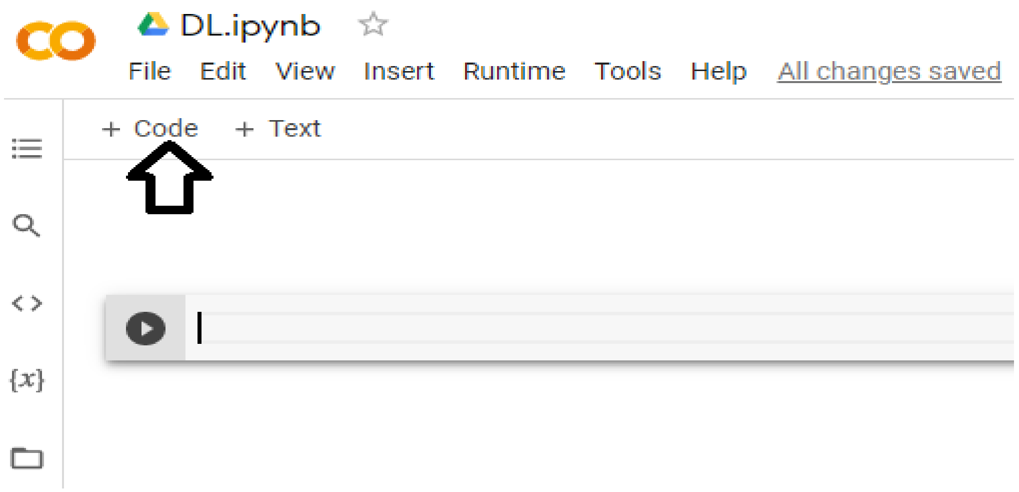
Want access to premium GPUs? Purchase additional compute units

Cancel Save

3) You can add two types of cells into your notebook using the +Code and +Text buttons at the top left. You can use the text cells as you would in Jupyter notebooks to add comments, headings etc.

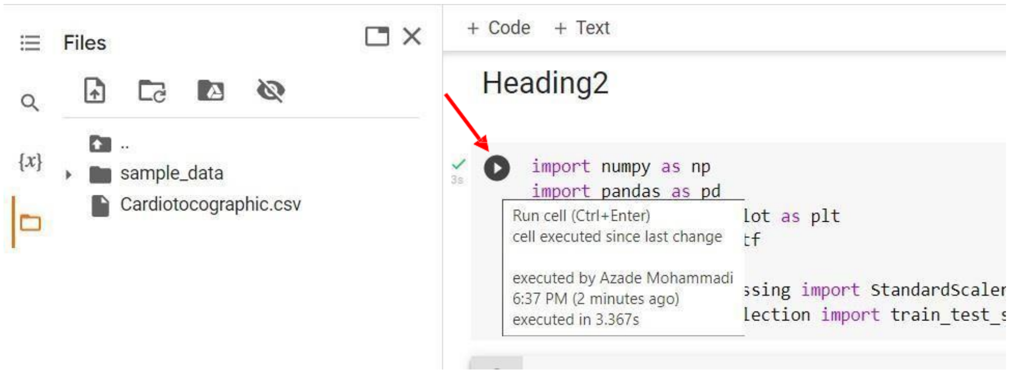


Add a code cell and run the imports we need.



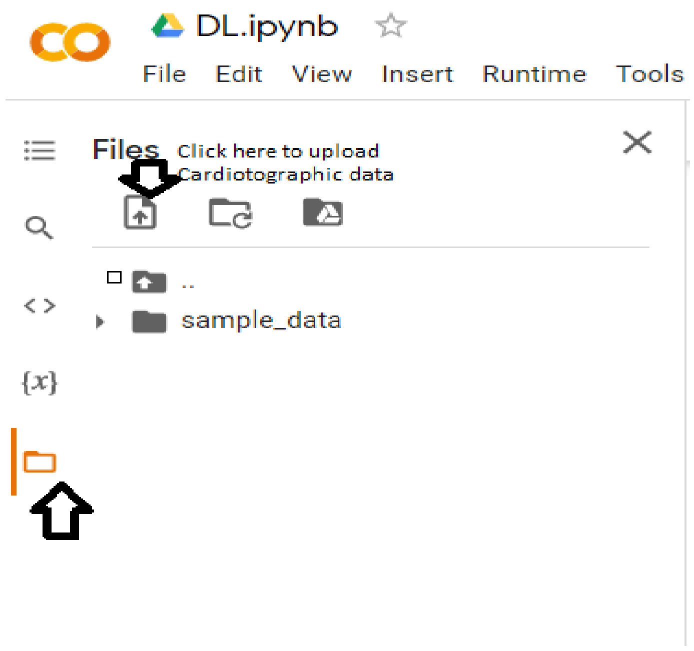
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

To execute the code, press Ctrl+Enter or click on the button at the left side of the cell, as shown below:



```
import numpy as np
import pandas as pd
lot as plt
tf
ssing import StandardScaler
lection import train_test_s
```

4) You can upload the data to the Colab by clicking on the File icon to open the "File Explorer" pane and then clicking on the File Upload icon at the top of this pane as shown below.

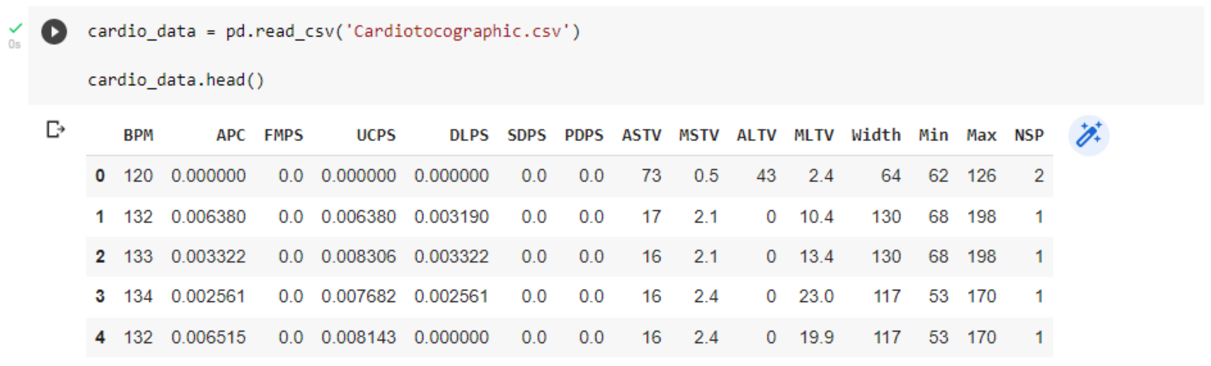


NB - Files are uploaded to the current runtime so are deleted when a session ends.

The file you need to upload for today's workshop is saved on Blackboard as **Cardiotocographic.csv**.

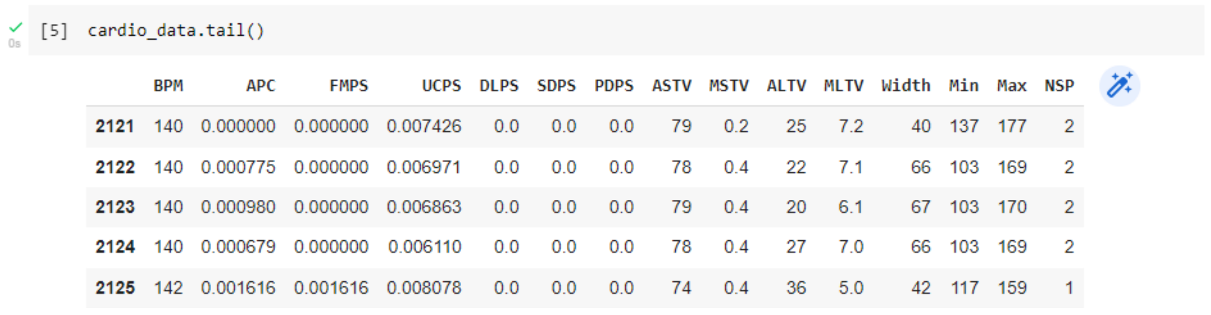
5) Having uploaded the dataset, we can now use the read_csv function to read it into a pandas dataframe. We can then inspect the first and last rows.

```
cardio_data = pd.read_csv('Cardiotocographic.csv')
```



```
cardio_data = pd.read_csv('Cardiotocographic.csv')

cardio_data.head()
```

	BPM	APC	FMPS	UCPS	DLPS	SDPS	PDPS	ASTV	MSTV	ALTV	MLTV	Width	Min	Max	NSP
0	120	0.000000	0.0	0.000000	0.000000	0.0	0.0	73	0.5	43	2.4	64	62	126	2
1	132	0.006380	0.0	0.006380	0.003190	0.0	0.0	17	2.1	0	10.4	130	68	198	1
2	133	0.003322	0.0	0.008306	0.003322	0.0	0.0	16	2.1	0	13.4	130	68	198	1
3	134	0.002561	0.0	0.007682	0.002561	0.0	0.0	16	2.4	0	23.0	117	53	170	1
4	132	0.006515	0.0	0.008143	0.000000	0.0	0.0	16	2.4	0	19.9	117	53	170	1

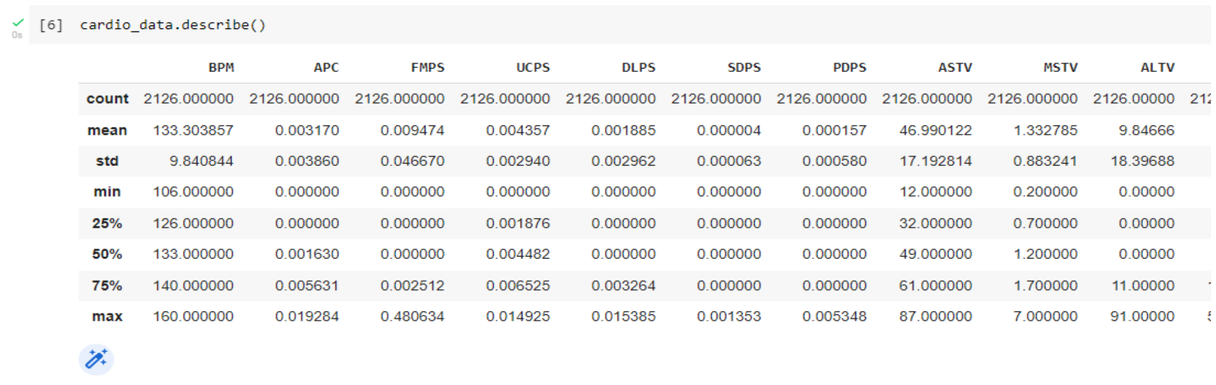


```
[5] cardio_data.tail()
```

	BPM	APC	FMPS	UCPS	DLPS	SDPS	PDPS	ASTV	MSTV	ALTV	MLTV	Width	Min	Max	NSP
2121	140	0.000000	0.000000	0.007426	0.0	0.0	0.0	79	0.2	25	7.2	40	137	177	2
2122	140	0.000775	0.000000	0.006971	0.0	0.0	0.0	78	0.4	22	7.1	66	103	169	2
2123	140	0.000980	0.000000	0.006863	0.0	0.0	0.0	79	0.4	20	6.1	67	103	170	2
2124	140	0.000679	0.000000	0.006110	0.0	0.0	0.0	78	0.4	27	7.0	66	103	169	2
2125	142	0.001616	0.001616	0.008078	0.0	0.0	0.0	74	0.4	36	5.0	42	117	159	1

6) We can also explore the data:

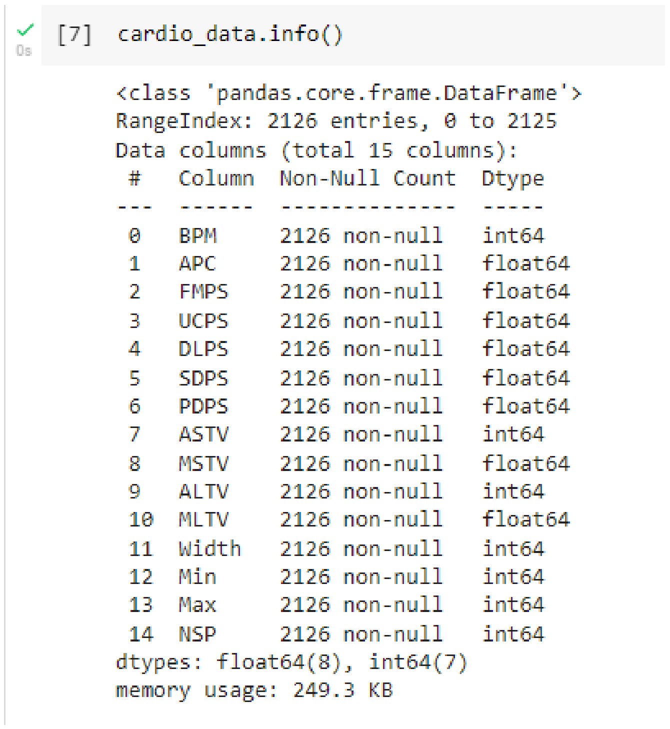
```
cardio_data.describe()
```



```
[6] cardio_data.describe()
```

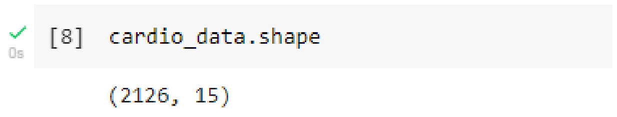
	BPM	APC	FMPS	UCPS	DLPS	SDPS	PDPS	ASTV	MSTV	ALTV
count	2126.000000	2126.000000	2126.000000	2126.000000	2126.000000	2126.000000	2126.000000	2126.000000	2126.000000	2126.00000
mean	133.303857	0.003170	0.009474	0.004357	0.001885	0.000004	0.000157	46.990122	1.332785	9.84666
std	9.840844	0.003860	0.046670	0.002940	0.002962	0.000063	0.000580	17.192814	0.883241	18.39688
min	106.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	12.000000	0.200000	0.00000
25%	126.000000	0.000000	0.000000	0.001876	0.000000	0.000000	0.000000	32.000000	0.700000	0.00000
50%	133.000000	0.001630	0.000000	0.004482	0.000000	0.000000	0.000000	49.000000	1.200000	0.00000
75%	140.000000	0.005631	0.002512	0.006525	0.003264	0.000000	0.000000	61.000000	1.700000	11.00000
max	160.000000	0.019284	0.480634	0.014925	0.015385	0.001353	0.005348	87.000000	7.000000	91.00000

```
cardio_data.info()
```



```
[7] cardio_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2126 entries, 0 to 2125
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   BPM     2126 non-null   int64
 1   APC     2126 non-null   float64
 2   FMPS    2126 non-null   float64
 3   UCPS    2126 non-null   float64
 4   DLPS    2126 non-null   float64
 5   SDPS    2126 non-null   float64
 6   PDPS    2126 non-null   float64
 7   ASTV    2126 non-null   int64
 8   MSTV    2126 non-null   float64
 9   ALTV    2126 non-null   int64
 10  MLTV    2126 non-null   float64
 11  Width   2126 non-null   int64
 12  Min     2126 non-null   int64
 13  Max     2126 non-null   int64
 14  NSP     2126 non-null   int64
dtypes: float64(8), int64(7)
memory usage: 249.3 KB
```

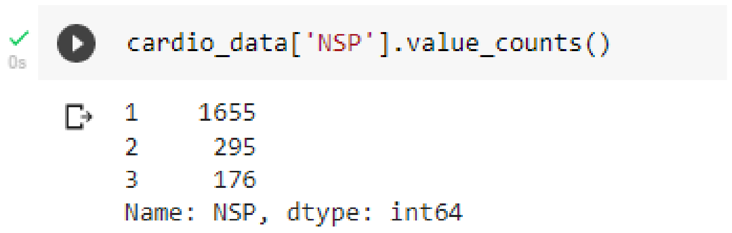
```
cardio_data.shape
```



```
[8] cardio_data.shape

(2126, 15)
```

```
cardio_data['NSP'].value_counts()
```

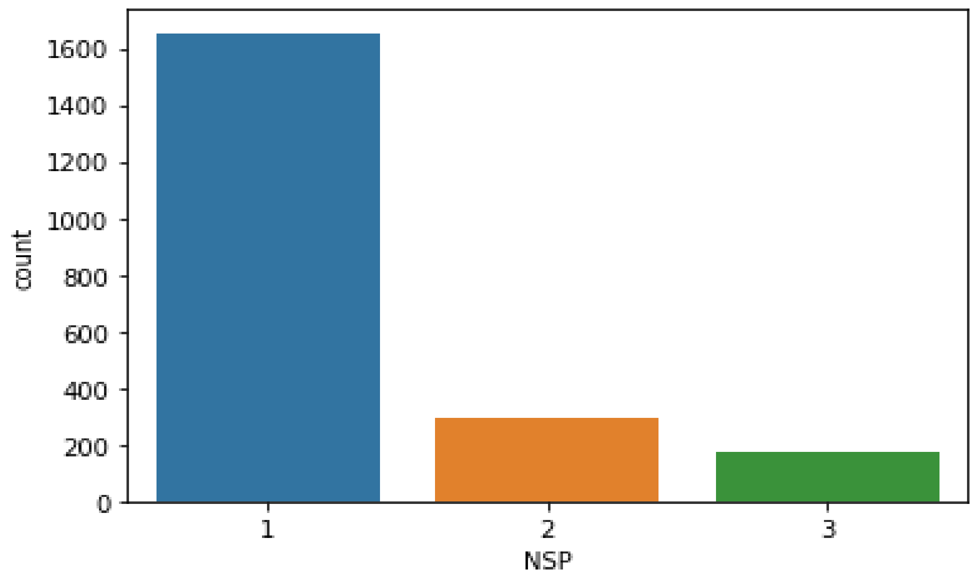
The above code shows the number of samples in each class



```
cardio_data['NSP'].value_counts()

1    1655
2     295
3     176
Name: NSP, dtype: int64
```

We can see we have imbalanced classes with 77.8% of the observations belong to the Normal class. We can also use the seaborn countplot to visualise this.

sns.countplot(cardio_data, x="NSP")



7) Next, we are going to use the train_test_split from Scikit Learn to divide the data into a training dataset and a test dataset.

Normalising the data is important when training a neural network. Not doing so may mean it takes longer to converge during training or fails to converge at all. In this workshop we will use the StandardScaler estimator from Scikit Learn. This scales the data so that the mean of each variable is 0, and the standard deviation is 1.

We also deduct 1 from the values of the class labels in the NSP column. This is because Keras assumes our class labels start at 0, whereas on page 5 you can see that for this dataset the class labels are 1, 2 and 3. Once we have deducted 1, class 0 is Normal, class 1 is Suspect, and class 2 is Pathologic.

```
X = cardio_data.drop('NSP',axis=1)
y = cardio_data['NSP'] -1

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, stratify=y, random_state=0)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)
```

Part Two: Building and Training our Neural Network

We're now ready to build our neural network. We do this using a Keras Sequential model which allows us to build up our neural network layer by layer.

8) We do this by instantiating the model and then using the ‘add’ method to add layers to the model.

When adding a layer, we need to specify what type of layer we are adding. In a simple feedforward network, we use a ‘Dense’ layer, which means each neuron in the layer is connected to each neuron in the previous layer.

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(9,activation='relu',input_shape=(14,)))
model.add(tf.keras.layers.Dense(3,activation='softmax'))
```

For each layer, we specify the number of neurons it is going to contain, and the activation function we want to use for that layer. Notice, for the first layer we define the input shape. We set this to 14 as there are 14 input variables in the data.

In the final layer, the number of neurons should be equal to the number of classes we have for our target variable. In this case we have three classes (Normal, Suspect and Pathologic), so we have three neurons in the final layer. For a multiclass classification problem, we use a softmax activation function in the final layer. This activation function ensures that the total value across the output neurons sums to 1, meaning we can interpret the output for each neuron as the probability that the observation belongs to that class.

9) The next step is to compile the model. At this stage, we have to select an optimizer and a loss function:

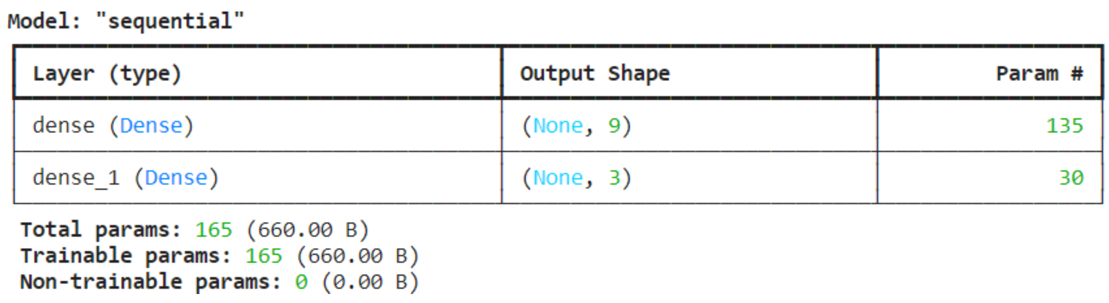
- The **optimizer** is a function which determines how fast the network ‘learns’ – i.e., how quickly it updates the weights in the network. If it’s too sensitive, it will update the weights too much based on the current training examples, meaning the values jump around and it doesn't converge. If it's not sensitive enough, it will take much longer to train. A common optimizer is Adam.

- The **loss** is a function which is used to calculate how big the error is in the neural network's predictions. When it is being trained, the weights in the neural network are updated to try and minimize the loss. Categorical cross entropy is typically the loss function we use for classification.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics= ['accuracy'])
```

10) We can use summary() to view a summary of the neural network we have built.

```
model.summary()
```



```
Model: "sequential"
┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 9)              │           135 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 3)              │            30 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 165 (660.00 B)
 Trainable params: 165 (660.00 B)
 Non-trainable params: 0 (0.00 B)
```

11) Similar to the way we train a model in Scikit Learn, we use the fit method to train the model. At this stage we can set the following parameters.

- **batch_size:** This is the number of training examples which are fed through the network in one go before updating the weights.

- **epochs:** In one epoch, **all** the training data is passed through the network once.

- **verbose:** This determines how much information we are given on the training and setting this to 2 outputs more detailed information as we go through the training.

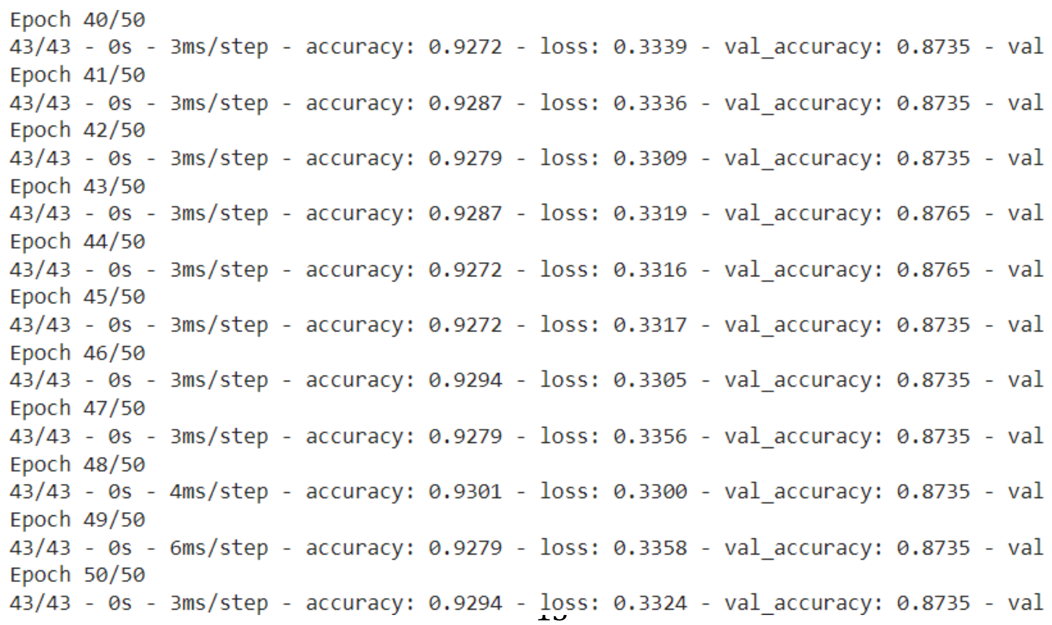
- **validation_split:** This determines how much data is left out of the training data and used to calculate the validation metrics during each epoch.

On page 12, we noted that we have imbalanced classes. There are many more observations of class 0, and so this will have a larger impact on the overall value of the loss function than observations belonging to the other two classes. In practice, this is likely to mean the resulting trained model may be more accurate at correctly identifying observations in class 0 than in the other two classes. We can address this when training by **weighting** the classes in inverse proportion to the % of observations in that class. We use the **class_weights** parameter to do this.

```
class_weights = { 0:1, 1:5.6, 2:9.4}

history = model.fit(X_train, np.asarray(y_train), \
          batch_size = 32, epochs= 50, \
          verbose=2, class_weight=class_weights, \
          validation_split=0.2)
```

Once training has begun, you should see output similar to the below.

NB – Exact results will vary from training run to training run, so the values you see will not exactly match the below.



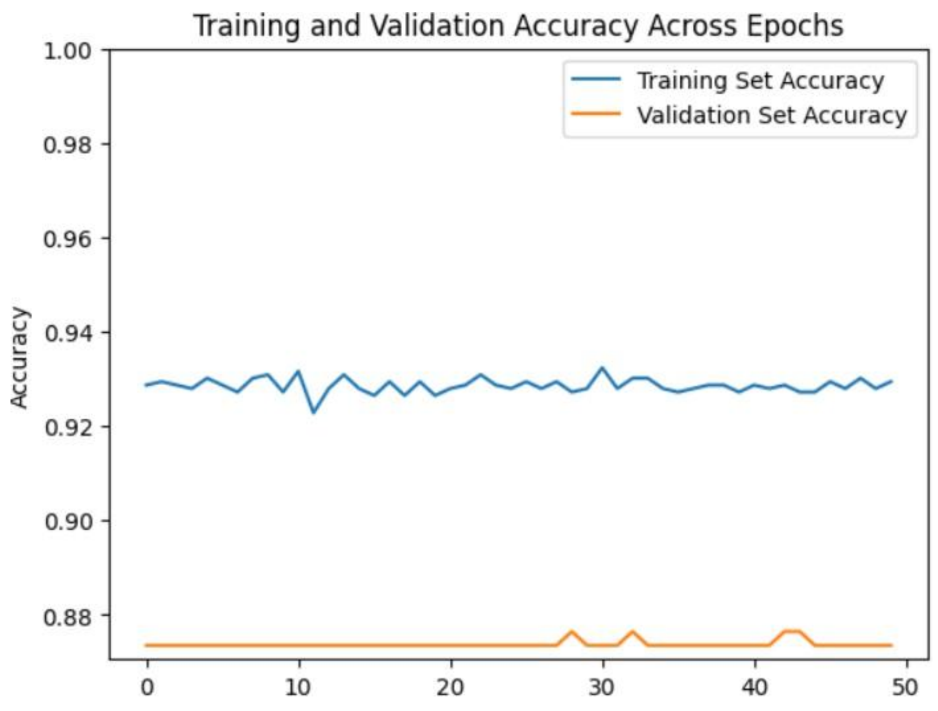
```
Epoch 40/50
43/43 - 0s - 3ms/step - accuracy: 0.9272 - loss: 0.3339 - val_accuracy: 0.8735 - val
Epoch 41/50
43/43 - 0s - 3ms/step - accuracy: 0.9287 - loss: 0.3336 - val_accuracy: 0.8735 - val
Epoch 42/50
43/43 - 0s - 3ms/step - accuracy: 0.9279 - loss: 0.3309 - val_accuracy: 0.8735 - val
Epoch 43/50
43/43 - 0s - 3ms/step - accuracy: 0.9287 - loss: 0.3319 - val_accuracy: 0.8765 - val
Epoch 44/50
43/43 - 0s - 3ms/step - accuracy: 0.9272 - loss: 0.3316 - val_accuracy: 0.8765 - val
Epoch 45/50
43/43 - 0s - 3ms/step - accuracy: 0.9272 - loss: 0.3317 - val_accuracy: 0.8735 - val
Epoch 46/50
43/43 - 0s - 3ms/step - accuracy: 0.9294 - loss: 0.3305 - val_accuracy: 0.8735 - val
Epoch 47/50
43/43 - 0s - 3ms/step - accuracy: 0.9279 - loss: 0.3356 - val_accuracy: 0.8735 - val
Epoch 48/50
43/43 - 0s - 4ms/step - accuracy: 0.9301 - loss: 0.3300 - val_accuracy: 0.8735 - val
Epoch 49/50
43/43 - 0s - 6ms/step - accuracy: 0.9279 - loss: 0.3358 - val_accuracy: 0.8735 - val
Epoch 50/50
43/43 - 0s - 3ms/step - accuracy: 0.9294 - loss: 0.3324 - val_accuracy: 0.8735 - val
```

Part Three: Evaluating Our Neural Network

Now we have trained our neural network we want to review the accuracy and validation metrics and also see how this network performs when making predictions for the test data which we held back.

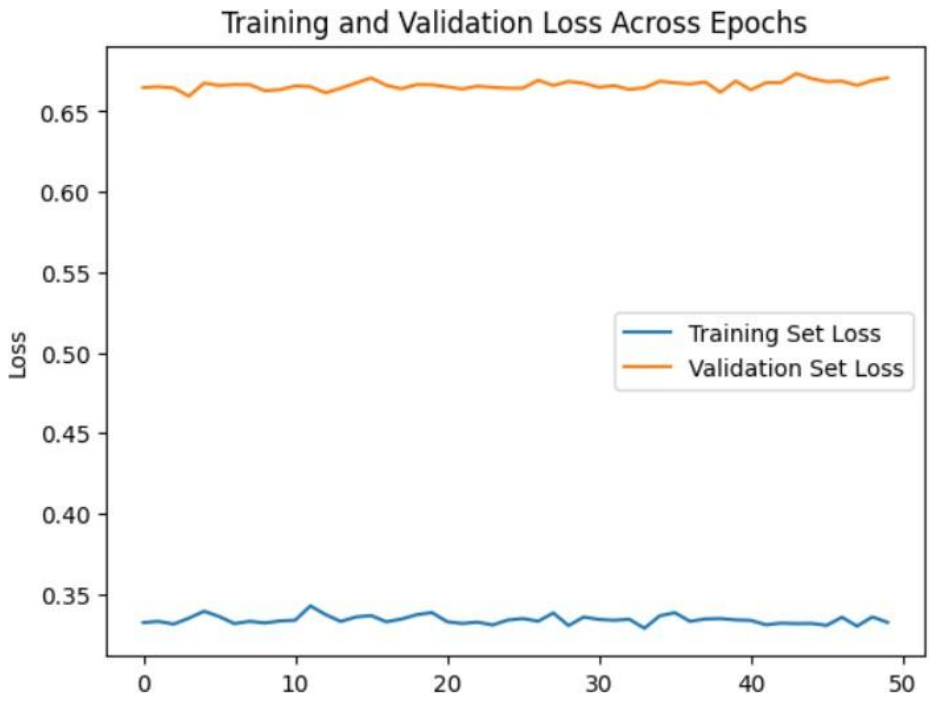
12) Once the training has been completed, we can plot the training and validation accuracy using the below code.

```
accuracy = history.history['accuracy']
validation_accuracy  =  history.history['val_accuracy']

plt.plot(accuracy, label='Training Set Accuracy')
plt.plot(validation_accuracy, label='Validation Set Accuracy')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy Across Epochs')
plt.legend()
```



13) We can also plot the training and validation loss using the code below. The characteristic shape of this curve shows a decrease in loss. If our model is starting to overfit, the validation loss may start to increase again.

```
loss  = history.history['loss']
validation_loss = history.history['val_loss']

plt.plot(loss, label='Training Set Loss')
plt.plot(validation_loss, label='Validation Set Loss')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Across Epochs')
plt.legend()
```



14) From Scikit Learn, we are going to import the confusion_matrix and classification_report functions which we can use to evaluate our model performance on the test data.

```
from sklearn.metrics import confusion_matrix, classification_report
```

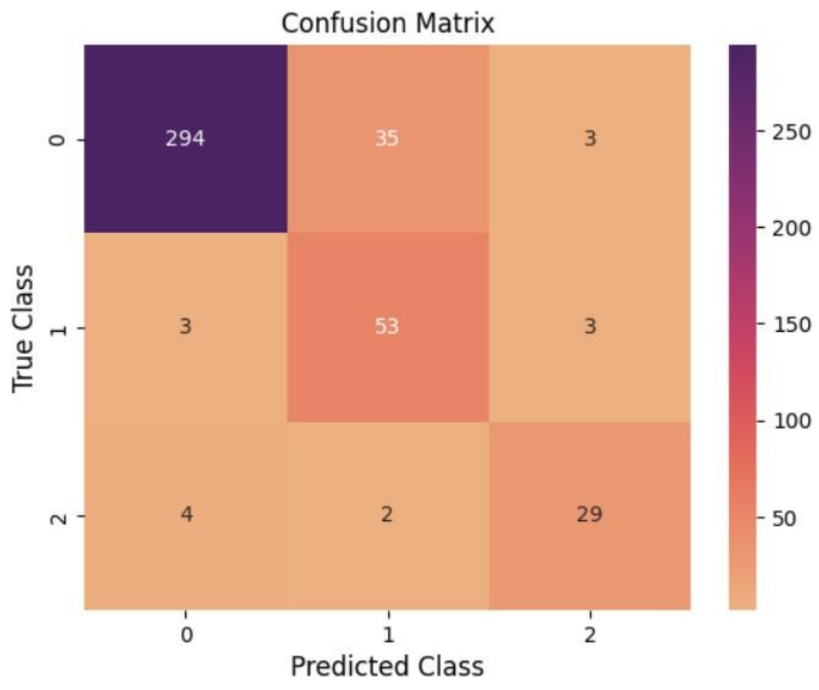
15) We can use the predict function on our trained model to generate predictions for the test data. This will output three values for each observation, one for each of the output neurons. As these outputs represent the predicted probabilities for each of the three classes, the final predicted class corresponds to the one with the maximum probability value. We can use the argmax function for this.

```
y_pred = model.predict(X_test)
y_pred = y_pred.argmax(axis=1)
```

16) Use the confusion_matrix function to generate a confusion matrix using our generated predictions (y_pred) and the true class labels for the test data (y_test). We use a seaborn heatmap to visualise these results.

```
confusion_matrix = confusion_matrix(y_test,y_pred)


ax = sns.heatmap(confusion_matrix, cmap='flare',annot=True, fmt='d')


plt.xlabel("Predicted Class",fontsize=12)
plt.ylabel("True Class",fontsize=12)
plt.title("Confusion Matrix",fontsize=12)


plt.show()
```

NB – Exact results will vary from training run to training run, so the values you see will not exactly match the values below.



17) Finally, we can use the classification_report function to view some key evaluation metrics for this model.

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.89      0.93       332
           1       0.59      0.90      0.71        59
           2       0.83      0.83      0.83        35

    accuracy                           0.88       426
   macro avg       0.80      0.87      0.82       426
weighted avg       0.91      0.88      0.89       426
```

Imagine that this model was going to be used to screen pregnancies to identify those which need further medical attention. In this hypothetical scenario, it is important that as many Suspect and Pathologic cases are identified as possible. Given this, what evaluation metric(s) would you recommend using to compare the performance of different models?

Part Four: Further Investigation

22) Try running this code again with the maximum number of epochs increased to 200. Take a look at the value of the loss and validation loss when these are plotted against the epochs? If the validation loss has started increasing after a number of epochs, what might this suggest?

23) Try changing the number of neurons in the hidden layer and see what impact this has on the performance of the model. Investigate how the model performance and training time changes when you vary the number of neurons in this layer.

24) As a Data Scientist, you should be aware of the ethical impact of your work. What do you think are some of the ethical considerations of using a machine learning model like this in a real-world medical setting, particularly if it is used to screen patients and guide decisions about follow-up treatment and care?