**Deep Learning**:

Deep Learning is a **group of techniques** based on **Neural Networks** that can **learn complex patterns** **directly from the data.**

**Why this called deep learning? What is deep about it?** So, the deepness comes from the number of things that we stack on top of each other. Deep Learning is a layer of neurons, stacked one after each other. And learning comes from machine learning. Deep Learning is a part of machine learning, its one technique of machine learning and that’s where learning comes from.

**Neural Network**:

Neural Network based on **neurons**. How they work is that they **take some inputs and do some calculations on the neurons and pass output to next layer and next layer, next layer**. At the end they have an **output and abundance of neurons** come up and create a neural network. Once we give enough training examples, it is able to learn the patterns. And how it happens is, it gives examples and comes with an output and compare the output with the actual output, the real-world value versus the prediction and this error is calculated and this error is used or the difference between the real data and what we predicted is used to correct the neural network and at the end we have a network that can actually predict accurately.

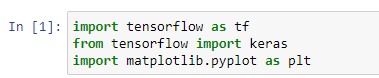
**Keras**:

It is basically a wrapper around the TensorFlow library. TensorFlow is an advanced **API** or library where we would need to do more things hands-on or decide things manually. Whereas Keras is kind of like a higher level one where we don’t need to go into the details of things, we can just complete or like build a neural network and just like one or two lines of code. Whereas in TensorFlow we might need to do it maybe 10 lines of code. So, it’s kind of like a **simplified version of TensorFlow** and we will build simple neural network.

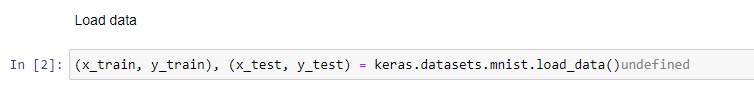
**MNIST** **Dataset**:

This dataset contains a collection of 70,000 and 28 \* 28 images of handwritten digits from 0 to 9. The dataset is already divided into training (60,000) and testing (10,000) sets in grayscale.

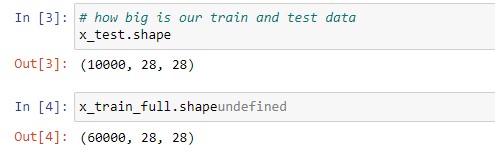
Now, **import all the relevant libraries** such as TensorFlow from Keras and matplotlib.



Load data from keras:



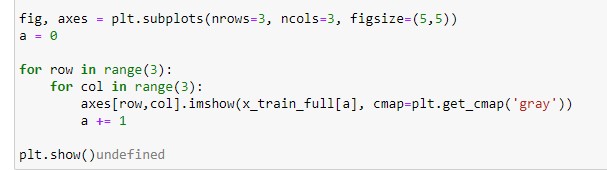
Now, testing the training and testing data’s shape.



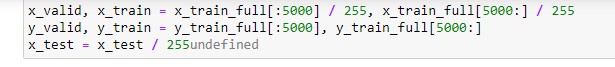
Testing one instance.

After testing the once instance of training data now we have to create an image to look-up.

Here, we are showing the nine images which is easy task for humans by we have to build a neural network that machine detect to get the actual number.

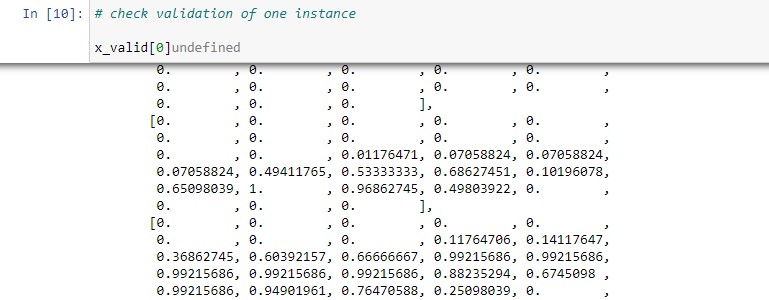


Normalization:

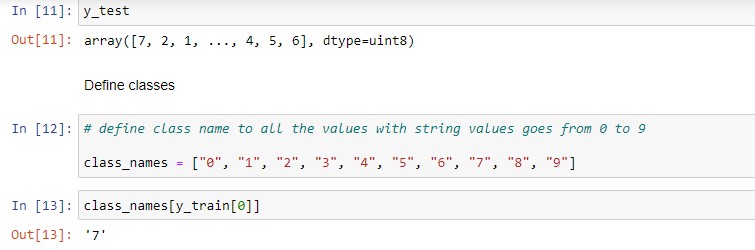


Check validation of one instance:

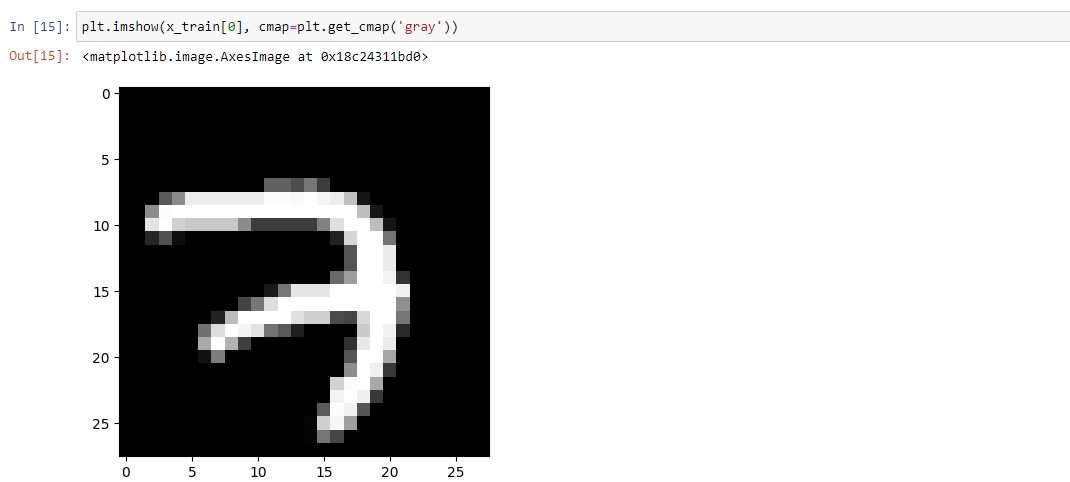
Here, we have to expect all the values lies between 0 to 1.



Define classes:



Displaying image using matplotlib library:

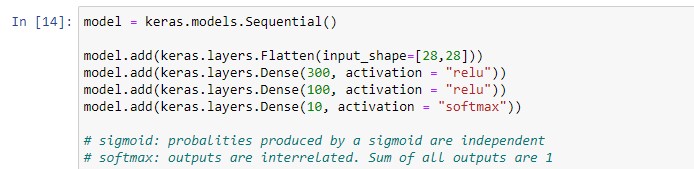


Build Neural Network:

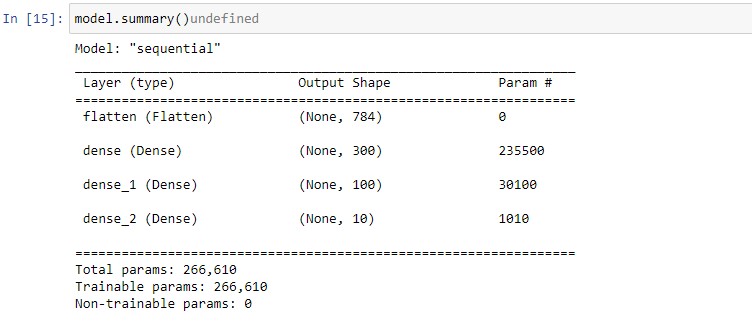
**First contact with Keras**:

The core data structures of Keras are **layers** and **models**. The simplest type of model is the Sequential model, a linear stack of layers. For more complex architectures, we should use the Keras functional API, which allows to build arbitrary graphs of layers.

We are creating a model and adding layers. So, basically model add a Keras layer model, add a Karas layer, add a Keras layer. In first line, we have model with sequential model from Keras. The second line here is a layer that we are adding an input layer. And the flattened layer is because we have a 28 to 28 matrix. We want to make it one big string of 784 pixels. Input shape is 28, 28 and flattening. It means that adding them one after another, making it one long line and that is the number of inputs. The next line or third line we are creating actually the next two lines are creating hidden layers. Hidden layers or cold dense layers, they are connected with everything that is before them or after them in a dense way. Dense means every neuron is connected to every other neuron. For the first hidden layer, I have a 300 neurons layer and I’m going to keep the activation very low because the building blocks of neural networks values is one of the best ones that we can. For the second hidden layer, we have 100 neurons and we are setting the activation to be with “**relu**”. And the last layer is an output layer. We have 10 output neurons. Why is that? It’s because there are 10 classes and we want to classify our images, the input images into 10 different classes. So that it starts from 0 to all the way to 9. And that makes 10 different classes. And we want our model to make a prediction for each of these classes. So, every time they give it, an image is going to tell us how likely it is that it says 0 and how likely it is that it says 1. And so on and so forth and also how likely it is that it’s 9 and at the end we will have the possibilities of which number this image is. We are using Softmax. It looks like, it goes from 0 to 1. It is basically similar to sigmoid. We are using Softmax not sigmoid because in this MNIST dataset, the problem specifically the classes are not related to each other, so they cannot one image cannot be half, one and half the other. They are completely independent. And sigmoid creates possibility that creates probabilities that are independent to each other. Whereas for Softmax all the possibilities of summation of all the possibilities and lead to 1. So, they sum to 1.



Once we build our model, look for the summary. So, this little function summary() that tells us what our model looks like. The flattened layer, the input layer has 784 neurons. Dense has 300 neurons. The first hidden layer, the dense too. First hidden has 300, second has 100 and output layer has 10 neurons.

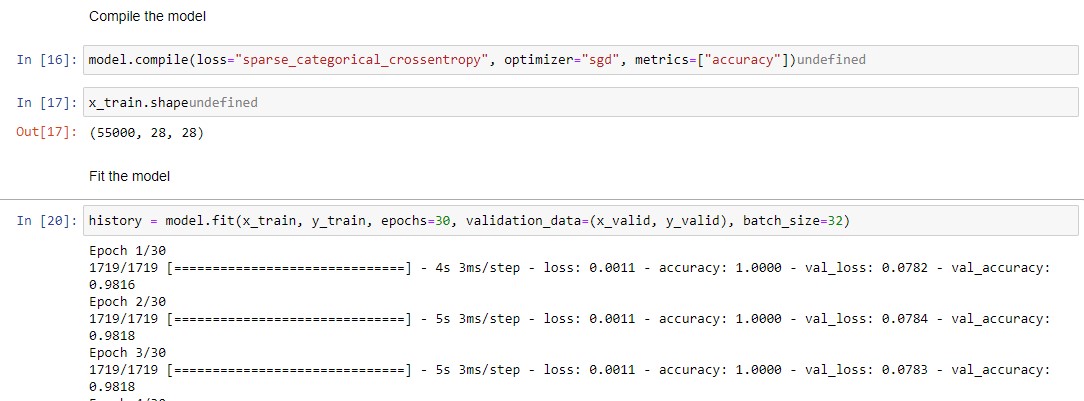


Every connection between two neurons, we have the weights and for every neuron itself we had the bias. So, in total we have lot of parameters i.e., 235500 and our model is updating the weights and biases to give us the best security possible.

**Compile the model**:

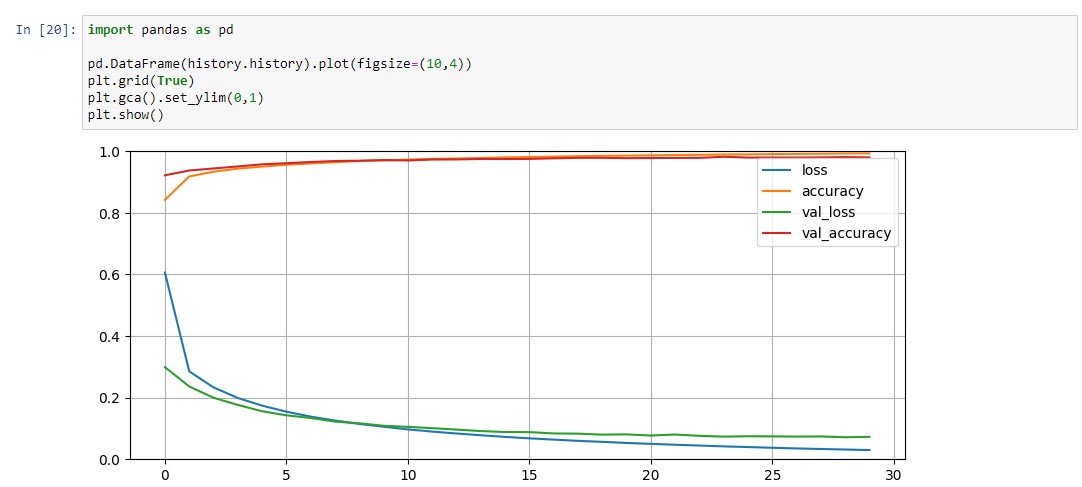
Here, we are doing some hyper parameters and hyper parameters are things that we have to set ourself beforehand. For example, for loss, we are using sparse categorical cross entropy. And that means that is this thing sparse categorical cross entropy is how we are going to measure how good our model is doing.

According to the Keras documentation of SparseCategoricalCrossentropy class, this crossentropy loss function is used when there are two or more label classes. And that’s why we are using this crossentropy. As an optimizer, we are using a **Stochastic Gradient Descent** (SGD). And the last one is metrics. This is basically what we want our model to report on while it’s training. So, we also cannot see the loss reported. And now, fit our model that’s why we have the training x and y values for the training. With the fit model we have epochs, validation, batch size. And these are the things that we can set to our fit model. Epoch means how many times we want this model to be run on the whole dataset. And batch size means how may data points are given to the model at each time.



Loss is getting lower and lower after each epoch whereas accuracy becomes higher and higher.

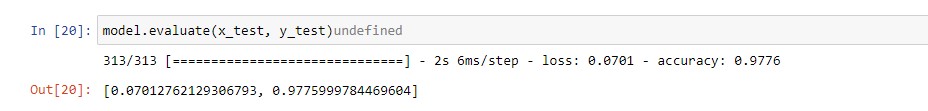
**How to evaluate Neural Network Performance?**



Here, what we see is as time goes by, how loss changing the training loss and accuracy and loss validation and accuracy changing. So, the blue and green line here the difference are losses. They both gone down. Of course, training loss is getting lower, faster. And the validation loss, after a while, it starts getting over a little bit slower.

Whereas for accuracy, we have more or less the same thing training accuracy goes quicker towards the end gets higher whereas the validation accuracy kind of gets a little bit better but it’s not a groundbreaking. The plot is what we expected. If there was overfitting, what would we see is accuracy of the training would get higher and higher whereas the accuracy of validation would more or less. So, it’s good we do not have overfitting on this one.

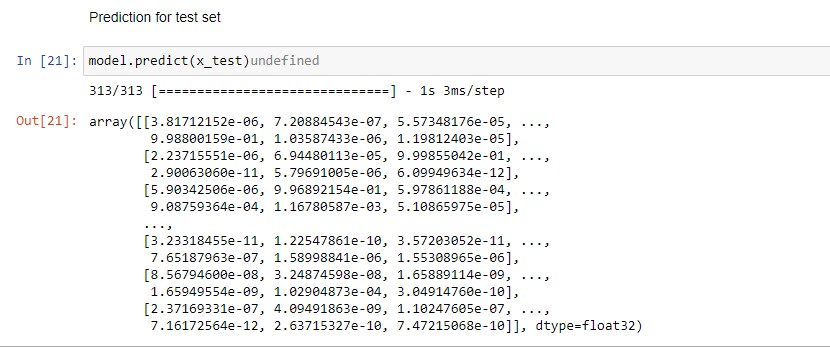
Now, we have a function evaluate that will tell us how our model is doing on the test set.



We see that our model loss is 0.07 and accuracy is 0.98.

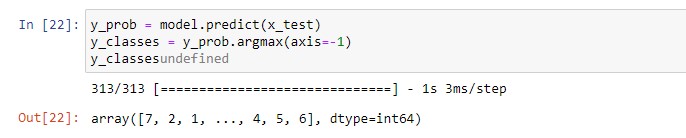
**Prediction for test set**:

model.predict for test set, test that inputs, it’s going to look like this.



What it shows us is that this is an array that has length of 10 and it is showing us what possibility or probability it gives for each of the classes. So, 3.81712152e-06, the probability of the first example and the test set to be zero. To be one, to be two etc., etc., etc.

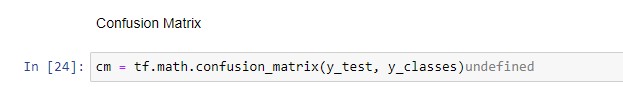
But of course, this is not really useful to see what we want to see is just tell us what number it is you think that it is. So, then we can use **argmax function** and then it will tell us what number it thinks it is.



It shows that 7 was the highest probability. 2 was the highest probability etc.

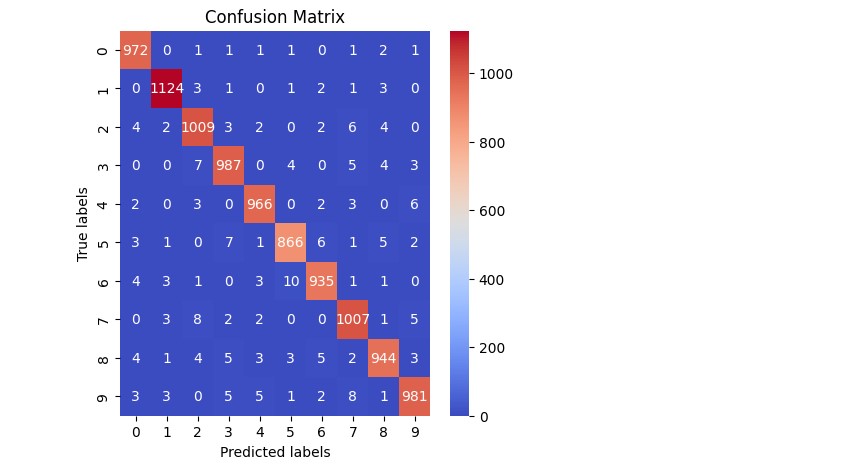
**Confusion Matrix**:

To see which number was confused with which number the most. We are using the confusion matrix with TensorFlow confusion matrix function.



Now we are using seaborn, we can use matplotlib but it has nicer looking graphics. Heatmap is used to show relationships between two variables plotted on each axis.





Most of the ones were classified as ones. 3’s and 5’s is confusing each other (**see true labels 5 and predicted labels 3**) which makes sense that handwritten threes and fives might look each-other.