**Lab: Neural Networks (using OCI Data Science service)**

**Introduction**

In this lab we will build a Neural Network to recognize handwritten digits. We will cover the required theory behind Neural Networks first, after which you will go on and put the theory into practice using the OCI Data Science service.

Estimated lab time: 45 minutes (video +/- 15 minutes, exercise +/- 30 minutes)

This video will cover the theory behind Neural Networks.

**Objectives**

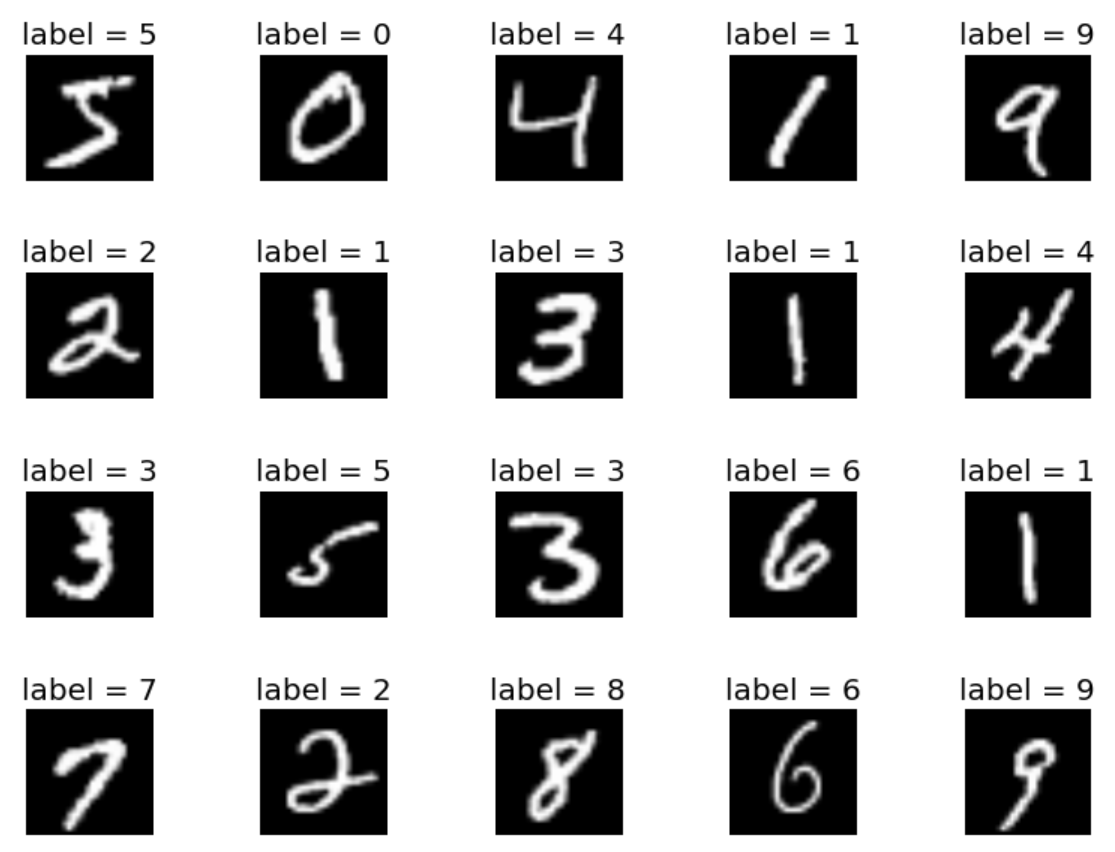
In this lab you will:

* Learn the theory behind Neural Networks.
* Learn how to prepare data for a NN.
* Learn the basics of constructing a Neural Network with Keras and Tensorflow.
* Learn how to visualize images with the matlib library.

**Task 1: Understanding the dataset and the architecture of the model**

1. Understanding the dataset

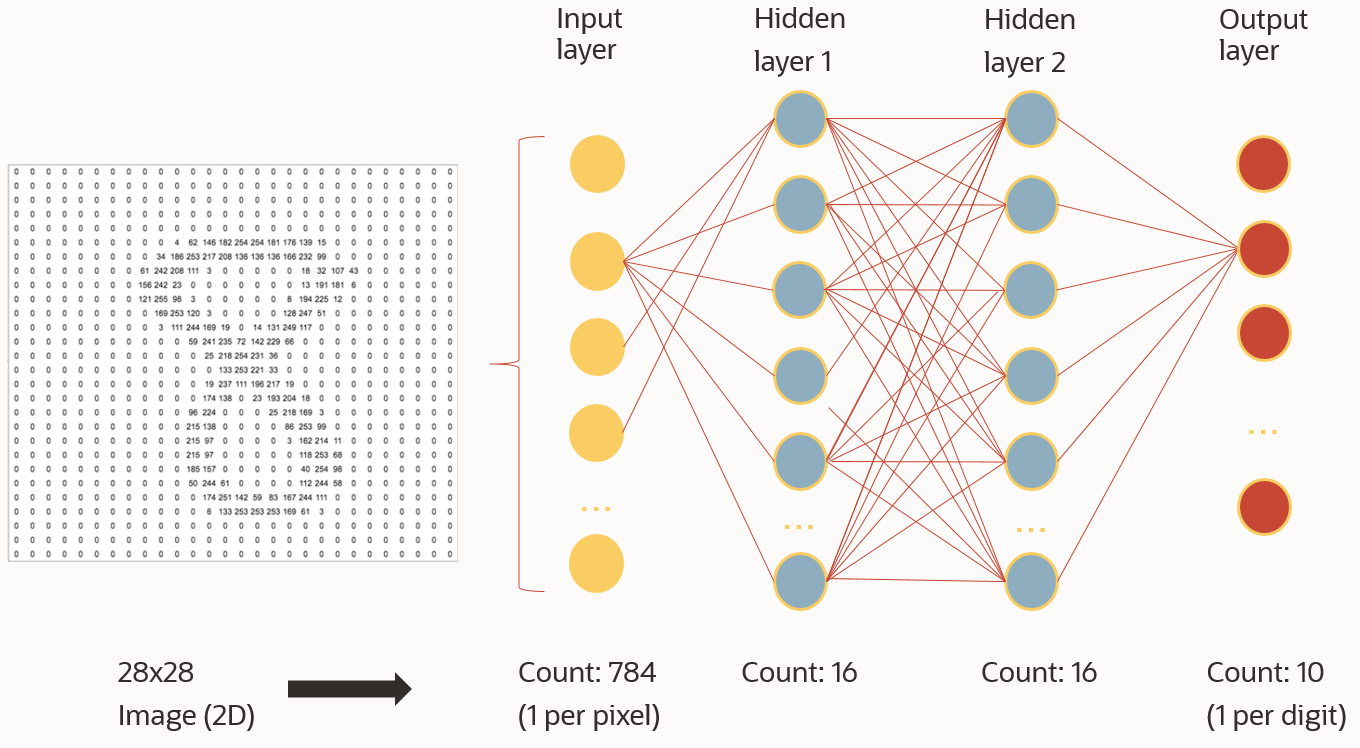
We will use the MNIST (Modified National Institute of Standards and Technology database) dataset for this, a real classic in Machine Learning. This dataset consists of thousands of images, each image representing a handwritten digit. All images have been labelled with the corresponding digit.



1. Understanding the architecture

Have a look at the architecture that we will build.

* Input layer: Our NN must be able to process one image in its input layer at a time. As you know, an image is 2D (in our case 28x28 pixels), but a basic NN input layer is flat (1D). Therefore, we will convert the 2D image into a long 1D array (28\*28=784 input neurons).
* We will have 2 hidden layers of 16 neurons each. The number of hidden layers and the number of neurons are somewhat arbitrary, and you may want to experiment with these.
* Output layer: This will have 10 neurons. Each neuron will represent the output for one of the digits (0 to 9).

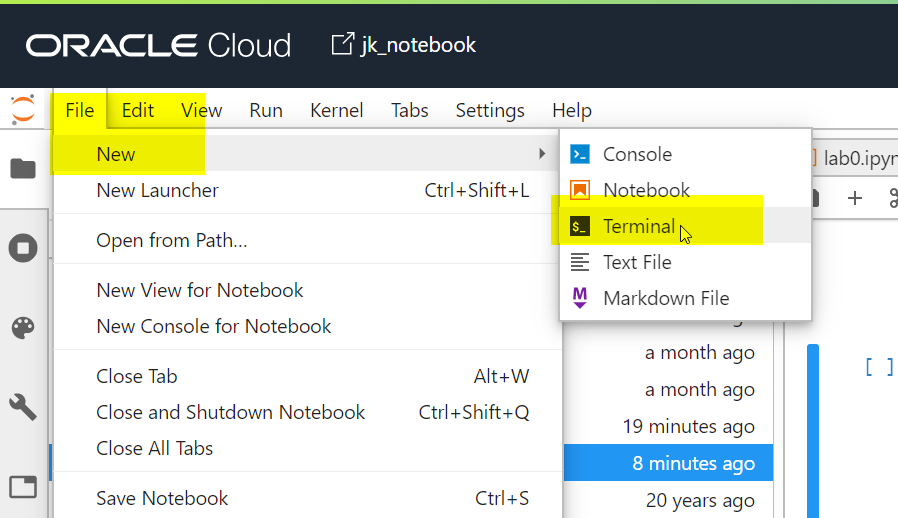


**Task 2: Install additional Python library idx2numpy**

In this lab we will require two Python library that by default are not installed in this Conda environment. We need keras to construct our model. And we need idx2numpy to convert the source images from IDX format to a native array format for our Neural Network. We need.

1. Open terminal

This is basically your OS access.



1. Install the new library

Switch to the right Conda environment, then use the PIP command tool to install the additional Python package.

Copyconda activate /home/datascience/conda/mlcpuv1

pip install keras

pip install idx2numpy

**Task 3: Downloading and unpacking the data**

1. Still in the terminal, download the MNIST data
2. Copywget https://objectstorage.eu-frankfurt-1.oraclecloud.com/p/Ho9fpzRD-oStl0uhuDUlZGPbx0ViU66I7oZ1vnUm2k\_IIjO5LTVh6jOooThfCxFY/n/odca/b/datascienceworkshop/o/t10k-images-idx3-ubyte.gz
3. wget https://objectstorage.eu-frankfurt-1.oraclecloud.com/p/3PRsmdGc7G3cRm6wQ2nMuWPQQjakgqulvGy7\_arPExK8QDa5zp9\_NwYJqSpI3Ymj/n/odca/b/datascienceworkshop/o/t10k-labels-idx1-ubyte.gz
4. wget https://objectstorage.eu-frankfurt-1.oraclecloud.com/p/kGVKLYuKWDoVeUHthfQ3nximY9ZThHKwFzG5B9bEVr11OXlL6u-mq0D0srcnTHWJ/n/odca/b/datascienceworkshop/o/train-images-idx3-ubyte.gz

wget https://objectstorage.eu-frankfurt-1.oraclecloud.com/p/RzExCygu\_bw-C57Dq6gy-UCL3r1ttYiAqxfy1uiejt35JDYwP7zLB\_AYQSB-J6Xa/n/odca/b/datascienceworkshop/o/train-labels-idx1-ubyte.gz

1. Unzip

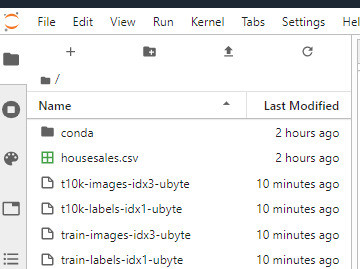
This will make all the files available in your root folder in the Data Science notebook.

Copygunzip train-images-idx3-ubyte.gz

gunzip train-labels-idx1-ubyte.gz

gunzip t10k-images-idx3-ubyte.gz

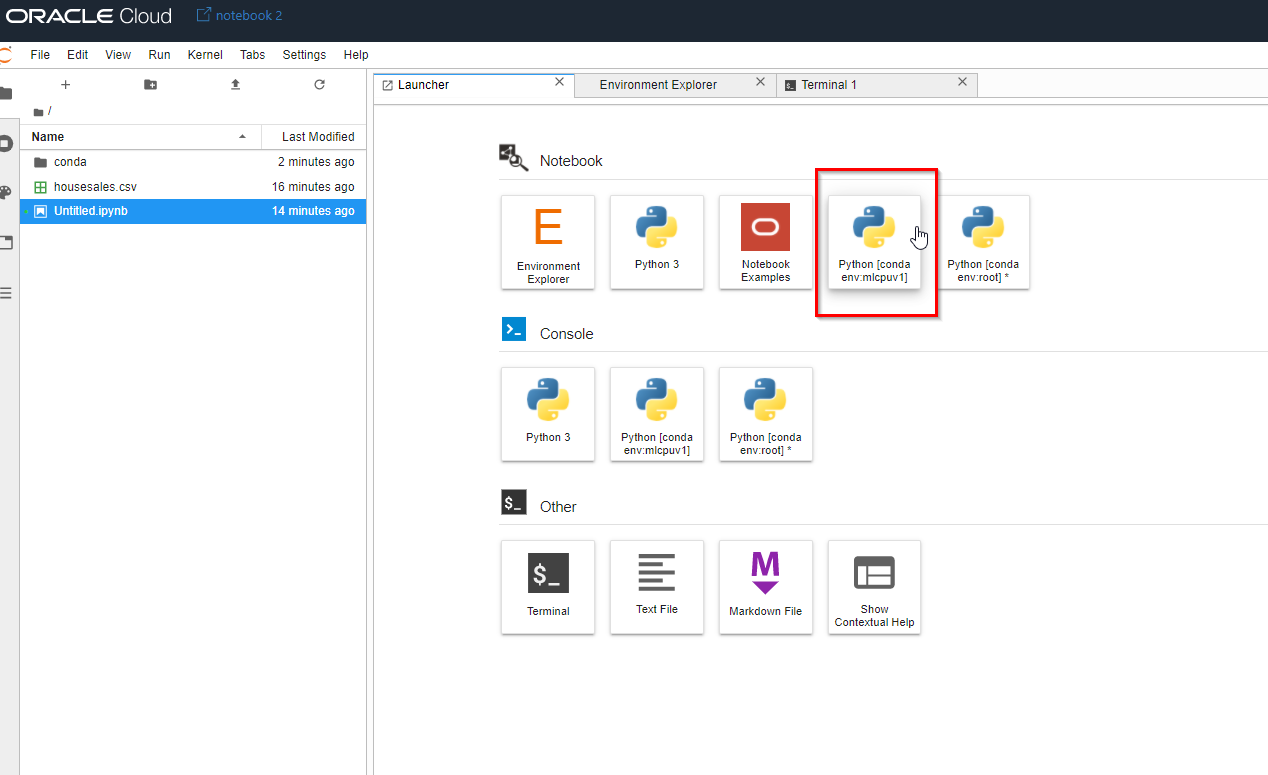
gunzip t10k-labels-idx1-ubyte.gz



**Task 4: Data Access and Exploration**

1. Start the Python notebook

It is important to select the Python environment with the Conda environment that we just installed. Look for the notebook that uses Conda **"mlcpuv1"** and open it.



1. Load the data into memory

Every time you see a piece of code as the following, please paste it into the notebook and click the run icon.

run script

Copy%matplotlib inline

import idx2numpy

import numpy as np

trainfile = 'train-images-idx3-ubyte'

trainfilelabels = 'train-labels-idx1-ubyte'

testfile = 't10k-images-idx3-ubyte'

testfilelabels = 't10k-labels-idx1-ubyte'

x\_train = idx2numpy.convert\_from\_file(trainfile)

y\_train = idx2numpy.convert\_from\_file(trainfilelabels)

x\_test = idx2numpy.convert\_from\_file(testfile)

y\_test = idx2numpy.convert\_from\_file(testfilelabels)

1. Inspect the datasets

In the previous lab we had to split the data into train and test ourselves. Notice that in this lab the split has already been done for us.

x\_train are the images. You could see every pixel as an input feature. y\_train are the labels of the image. This is a one-dimensional array with the digits as assigned by a person (0 to 9).

Equally, x\_train and y\_train are the images and corresponding labels for the test set.

1. Verify the data

How many images do we have for training, and what is the size of each image?

The following shows that the training data set has 60000 images. The other values indicate the dimensions of the image: 28x28 pixels.

Copyx\_train.shape

1. What is the shape of the labels for training?

The following shows that we have a list of 60000 entries. Each entry indicates the digit for the image (a value from 0 to 9).

Copyy\_train.shape

1. Let's do the same for the test images.

This will tell us that there are 10000 images for validation.

Copyx\_test.shape

1. And let's doublecheck the labels of the test images.

This will show that the images are labeled with the corresponding digit.

Copyy\_test.shape

1. How does one particular image actually look like?

Let's display one of the training images at random, in this case the one with index 5 (of 60000).

You can more or less see a shape of a digit show up. We'll show it as an actual image a bit later on.

Copyx\_train[5]

1. What is the label for this particular image?

Let's show the label by accessing the y\_train with the same index (5).

According to the labels, this is an image of the digit 2.

Copyy\_train[5]

1. Display the data as an image

Let's verify this by displaying the data as an image. We will use the matplotlib library to do so.

Copyimport matplotlib.pyplot as plt

plt.imshow(x\_train[5], cmap='Greys')

Indeed, we can see that this is a two.

**Task 5: Data Preparation**

The Neural Network that we want to build will have an input layer of 784 neurons. See also the architecture picture above. Each of the neurons will represent one pixel in the input image.

There are two issues that we have to address:

* The shape: We must convert the 2D shape of 28x28x1 pixels into a 1D array of 784 elements.
* The values: Our input neurons expect values between 0.0 and 1.0, however our actual input values are currently 0 to 255. We must scale these values as well.

1. Flatten the 28x28 array

Flatten the array of each image into a 784 array. Do this for train and test.

Copyx\_train = x\_train.reshape(x\_train.shape[0], 784)

x\_test = x\_test.reshape(x\_test.shape[0], 784)

1. Check the results

Let's check that this conversion was successful by checking the new shape of the training set.

Copyx\_train.shape

1. Scale the values of the pixels from 0-255 to 0.0-1.0.
2. Copyx\_train = x\_train.astype('float32')
3. x\_test = x\_test.astype('float32')
4. x\_train /= 255

x\_test /= 255

1. Check the results

Let's check the result by again displaying our example digit at index 5. You will see that there are no rows anymore in the array (it's 1D now), and that the values are between 0.0 and 1.0.

Copyx\_train[5]

**Task 6: Model build and training**

1. Doublecheck the shapes

Let's doublecheck the shapes of the input data before we start the training process.

Copyprint('x\_train shape:', x\_train.shape)

print('Number of images in x\_train', x\_train.shape[0])

print('Number of images in x\_test', x\_test.shape[0])

You should see x\_train shape: (60000, 784), Number of images in x\_train 60000, Number of images in x\_test 10000.

1. Construct the model

Our data is ready to go. Now it's time to build the neural network. Remember, we will build an input layer of 784 neurons, then two hidden layers of 16 neurons each, and finally an output layer of 10 neurons (one for each digit). If this is unclear, please review the architecture at the start of the lab. We are using the Tensorflow and Keras open source libraries for this.

Notice that there is no clear methodology that can tell you from the beggining what the right size and number of hidden layers would be required. To be able to determine this parameters you would need to "debug" the neural network. Change the number of hidden layers or the size of the neurons and monitor if the loss gets lower and the accuracy increases. Later validate this on the test set.

Notice how in the first model.add we have to specify both the input shape (784 neurons) and the first hidden layer (16 neurons).

Copyimport tensorflow as tf

import keras

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(16, input\_shape=(784, ), activation=tf.nn.relu))

model.add(Dense(16, activation=tf.nn.relu))

model.add(Dense(10, activation=tf.nn.softmax))

1. Train the model

At this point the initial architecture of our Neural Network is ready. It has random weights to start with. Next, we will train the model to optimize the weights.

Depending on the shape you are running the notebook on, this can take a few minutes.

Notice that the input for the model training is the training images (x\_train) and the training labels (y\_train). We have chosen 10 epochs. This means the neural network would run through the entire dataset 10 times.

* loss specifies the loss or also called the objective function. It calculates how far off the neural network's predictions are. The results are used to adjust the weights to minimize the loss.
* optimizer is a function used to minimize the loss. To do so we need to adjust the waits in the forward and the backpropagation. The optimizer is the function that would be used in that process.
* metrics is a function that is used to judge the performance of the model. You can specify one or more metrics. It is similar to the loss function but the result is not used when training the model. You could use as metric any of the loss functions available in Keras.
* Copymodel.compile(optimizer='adam',
* loss='sparse\_categorical\_crossentropy',
* metrics=['accuracy'])

model.fit(x=x\_train, y=y\_train, epochs=10)

**Task 7: Check model accuracy**

1. Check the accuracy of the last epoch.

You should see an accuracy of around 96%. This is the accuracy on the data in the training set. However, as you know by now, it's important to verify the accuracy of the model on unseen data.

NN accuracy

1. Visual verification of the model by predicting an example

First of all, let's check the performance intuitively through a visualization. Let's take an example image from the test set and check if the model is able to classify it correctly. We'll take a random index of 99.

You will see this is a 9.

Copyplt.imshow(x\_test[99].reshape(28, 28),cmap='Greys')

1. What is the official label for this digit?

You will see this is labelled as a 9 as well.

Copyy\_test[99]

1. Is our model able to correctly classify it as a 9?

The argmax function returns the output neuron that has the highest value. In this case this correctly predicts a 9.

Copypredict = model.predict(x\_test[99].reshape(1,784))

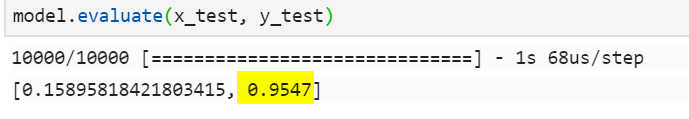
print(predict.argmax())

1. Numerical verification of the model

We can use model.evaluate to calculate the accuracy of prediction on the entire testset. This will run the prediction on the 10000 images in the testset and compare the predicted digits with the actual labels, and calculate an accuracy.

Copymodel.evaluate(x\_test, y\_test)

1. Conclusion of numerical verification

You should see an accuracy on unseen data of about 95%.

This is the actual accuracy of the model. In other words, the model is able to interpret an image of a digit and correctly classify it in 95% of the cases.