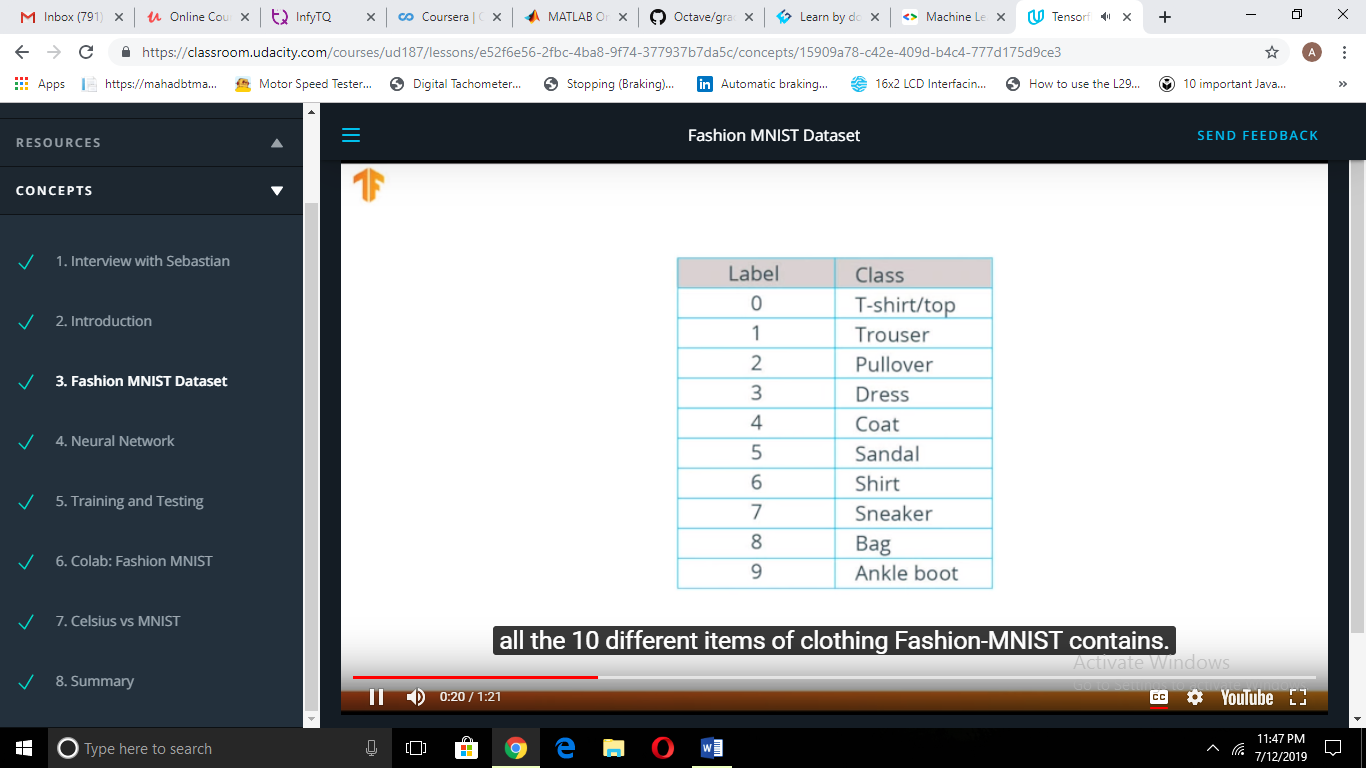
**Image Classification**

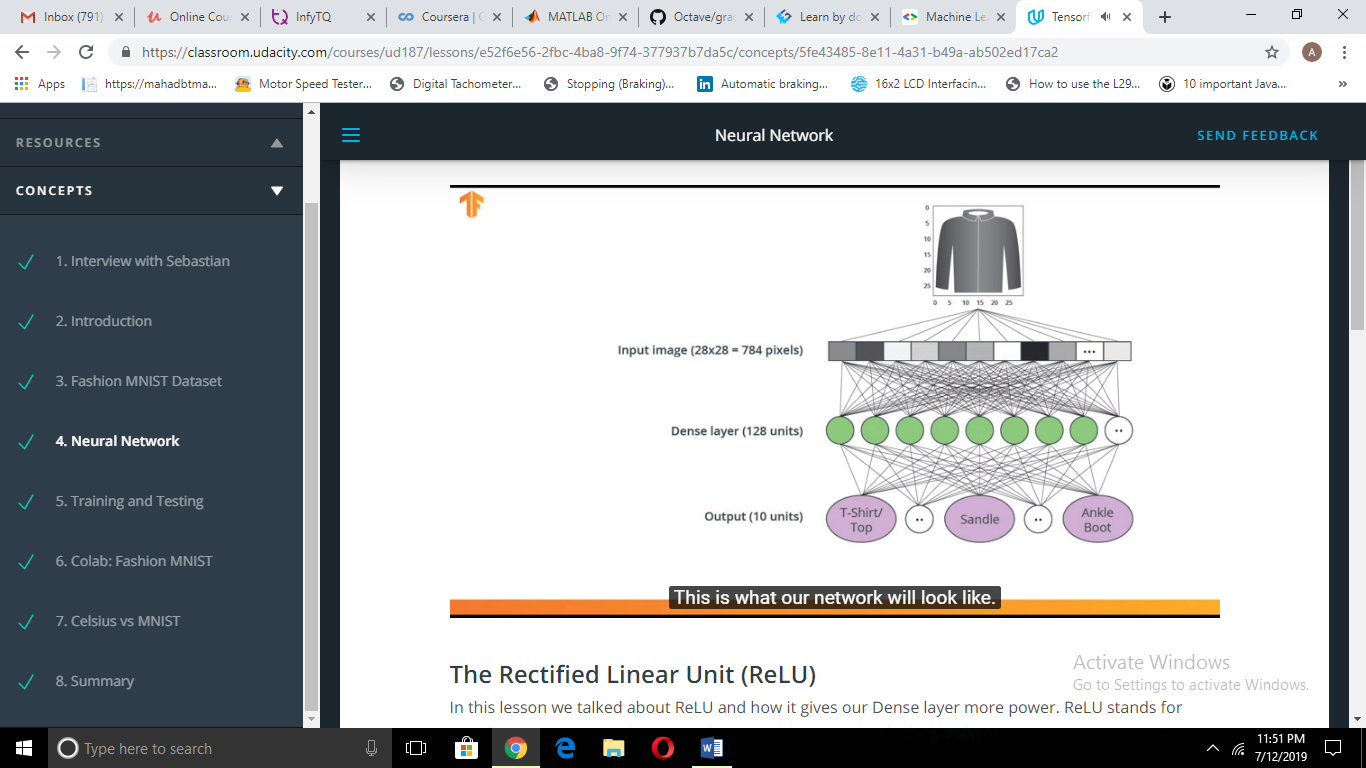
Dataset used: Fashion MNIST Dataset

I/P image size: 28\*28 =784 bytes (grey scale values)

Items of Clothing dataset contains;



Avl Images: 70,000=60000(training set)+ 10000(testing set)



Deep neural network is built for training. This model accepts an array of size 784 as input.

But, since the image is of size 28x28,we need to restructure the 2D data into a 1D array .The process of doing this is called flattening which is achieved by using a flattened layer.

tf.keras.layers.flatten(input\_shape=(28,28,1))

This input is then connected to a dense layer(of 128 units ,here.) Activation function- **ReLU, a mathematical function**  gives more power to this layer .

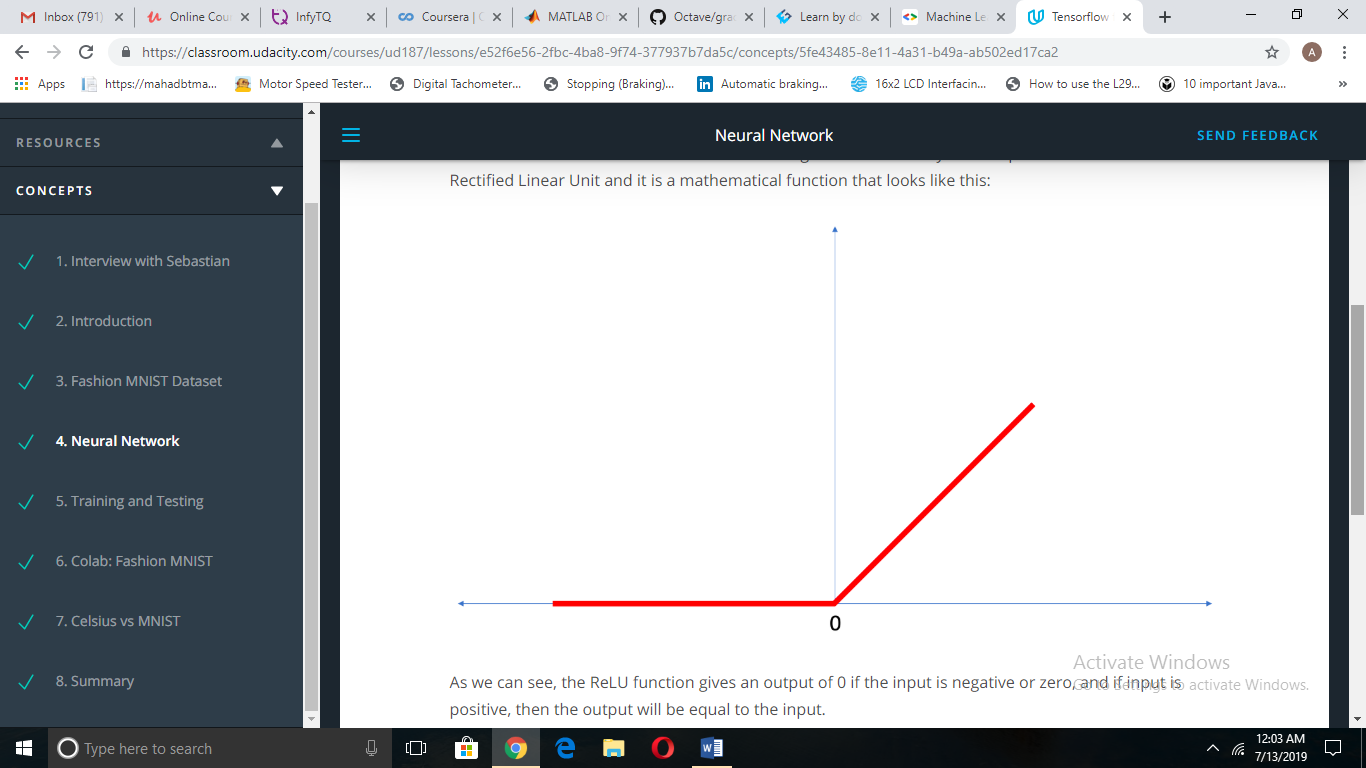
Tf.keras.layers.Dense(128,activation=tf.nn.relu)

ReLU –Rectified Linear Unit gives o/p zero for negative i/p and positive and equal to i/p for positive i/p which gives a network ability to solve non-linear problems.

Other terms:

Softmax: A function that provides probabilities for each possible output class

Classification: A machine learning model used for distinguishing among two or more output categories



Finally , the last layer contains 10 units because we have 10 classes /types of clothing items.

tf.keras.layers.Dense(10, activation=tf.nn.softmax)

Steps:

1. Install & import dependencies.
2. Import dataset, define column names (if reqd)
3. Explore dataset & preprocess it before being used for training(normalization, shuffling)
4. Build the model: Structuring/setting up layers, compiling the model
5. Train and evaluate the metrics parameters of the model
6. Make predictions and explore.

The layer structure for CNNs changes as follows:

model = tf.keras.Sequential([

tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,

input\_shape=(28, 28, 1)),

tf.keras.layers.MaxPooling2D((2, 2), strides=2),

tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),

tf.keras.layers.MaxPooling2D((2, 2), strides=2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation=tf.nn.relu),

tf.keras.layers.Dense(10, activation=tf.nn.softmax)

])

This network layers are:

* **"convolutions"** tf.keras.layers.Conv2D and MaxPooling2D— Network start with two pairs of Conv/MaxPool. The first layer is a Conv2D filters (3,3) being applied to the input image, retaining the original image size by using padding, and creating 32 output (convoluted) images (so this layer creates 32 convoluted images of the same size as input). After that, the 32 outputs are reduced in size using a MaxPooling2D (2,2) with a stride of 2. The next Conv2D also has a (3,3) kernel, takes the 32 images as input and creates 64 outputs which are again reduced in size by a MaxPooling2D layer. So far in the course, we have described what a Convolution does, but we haven't yet covered how you chain multiples of these together. We will get back to this in lesson 4 when we use color images. At this point, it's enough if you understand the kind of operation a convolutional filter performs
* **output** tf.keras.layers.Dense — A 128-neuron, followed by 10-node *softmax* layer. Each node represents a class of clothing. As in the previous layer, the final layer takes input from the 128 nodes in the layer before it, and outputs a value in the range [0, 1], representing the probability that the image belongs to that class. The sum of all 10 node values is 1.

**Compile the model**

Before the model is ready for training, it needs a few more settings. These are added during the model's *compile* step:

* *Loss function* — An algorithm for measuring how far the model's outputs are from the desired output. The goal of training is this measures loss.
* *Optimizer* —An algorithm for adjusting the inner parameters of the model in order to minimize loss.
* *Metrics* —Used to monitor the training and testing steps. The following example uses *accuracy*, the fraction of the images that are correctly classified.

Splitting of datasets:

Datasets are typically split into different subsets to be used at various stages of training and evaluation of the neural network.

Training Set: The data used for training the neural network.

Test set: The data used for testing the final performance of our neural network.

The test dataset was used to try the network on data it has never seen before. This enables us to see how the model generalizes beyond what it has seen during training, and that it has not simply memorized the training examples.

In the same way, it is common to use what is called a Validation dataset. This dataset is not used for training. Instead, it it used to test the model during training. This is done after some set number of training steps, and gives us an indication of how the training is progressing. For example, if the loss is being reduced during training, but accuracy deteriorates on the validation set, that is an indication that the model is memorizing the test set.

The validation set is used again when training is complete to measure the final accuracy of the model.

Detailed Steps with commands:

Steps:

1. Install & import dependencies.

!pip install -U tensorflow\_datasets

from \_\_future\_\_ import absolute\_import, division, print\_function, unicode\_literals

# Import TensorFlow and TensorFlow Datasets

import tensorflow as tf

import tensorflow\_datasets as tfds

tfds.disable\_progress\_bar()

# Helper libraries

import math

import numpy as np

import matplotlib.pyplot as plt

print(tf.\_\_version\_\_)

tf.enable\_eager\_execution()

import logging

logger = tf.get\_logger()

logger.setLevel(logging.ERROR)

1. Import dataset, define column names (if reqd)

dataset, metadata = tfds.load('fashion\_mnist', as\_supervised=True, with\_info=True)

train\_dataset, test\_dataset = dataset['train'], dataset['test']

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

1. Explore dataset & preprocess it before being used for training(normalization, shuffling)

num\_train\_examples = metadata.splits['train'].num\_examples

num\_test\_examples = metadata.splits['test'].num\_examples

print("Number of training examples: {}".format(num\_train\_examples))

print("Number of test examples: {}".format(num\_test\_examples))

def normalize(images, labels):

images = tf.cast(images, tf.float32)

images /= 255

return images, labels

# The map function applies the normalize function to each element in the train

# and test datasets

train\_dataset = train\_dataset.map(normalize)

test\_dataset = test\_dataset.map(normalize)

# Take a single image, and remove the color dimension by reshaping

for image, label in test\_dataset.take(1):

break

image = image.numpy().reshape((28,28))

# Plot the image - voila a piece of fashion clothing

plt.figure()

plt.imshow(image, cmap=plt.cm.binary)

plt.colorbar()

plt.grid(False)

plt.show()

#plot25 images

# Take a single image, and remove the color dimension by reshaping

for image, label in test\_dataset.take(1):

break

image = image.numpy().reshape((28,28))

# Plot the image - voila a piece of fashion clothing

plt.figure()

plt.imshow(image, cmap=plt.cm.binary)

plt.colorbar()

plt.grid(False)

plt.show()

1. Build the model: Structuring/setting up layers, compiling the model

model = tf.keras.Sequential([

tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,

input\_shape=(28, 28, 1)),

tf.keras.layers.MaxPooling2D((2, 2), strides=2),

tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),

tf.keras.layers.MaxPooling2D((2, 2), strides=2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation=tf.nn.relu),

tf.keras.layers.Dense(10, activation=tf.nn.softmax)

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

1. Train and evaluate the metrics parameters of the model

BATCH\_SIZE = 32

train\_dataset = train\_dataset.repeat().shuffle(num\_train\_examples).batch(BATCH\_SIZE)

test\_dataset = test\_dataset.batch(BATCH\_SIZE)

model.fit(train\_dataset, epochs=10, steps\_per\_epoch=math.ceil(num\_train\_examples/BATCH\_SIZE))

test\_loss, test\_accuracy = model.evaluate(test\_dataset, steps=math.ceil(num\_test\_examples/32))

print('Accuracy on test dataset:', test\_accuracy)

1. Make predictions and explore.

for test\_images, test\_labels in test\_dataset.take(1):

test\_images = test\_images.numpy()

test\_labels = test\_labels.numpy()

predictions = model.predict(test\_images)

np.argmax(predictions[0])#returns class no. with max probability

test\_labels[0]#returns actual class

def plot\_image(i, predictions\_array, true\_labels, images):

predictions\_array, true\_label, img = predictions\_array[i], true\_labels[i], images[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img[...,0], cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class\_names[predicted\_label],

100\*np.max(predictions\_array),

class\_names[true\_label]),

color=color)

def plot\_value\_array(i, predictions\_array, true\_label):

predictions\_array, true\_label = predictions\_array[i], true\_label[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

thisplot = plt.bar(range(10), predictions\_array, color="#777777")

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

thisplot[predicted\_label].set\_color('red')

thisplot[true\_label].set\_color('blue')

i = 0

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

plot\_image(i, predictions, test\_labels, test\_images)

plt.subplot(1,2,2)

plot\_value\_array(i, predictions, test\_labels)

i = 12

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

plot\_image(i, predictions, test\_labels, test\_images)

plt.subplot(1,2,2)

plot\_value\_array(i, predictions, test\_labels)

# Plot the first X test images, their predicted label, and the true label

# Color correct predictions in blue, incorrect predictions in red

num\_rows = 5

num\_cols = 3

num\_images = num\_rows\*num\_cols

plt.figure(figsize=(2\*2\*num\_cols, 2\*num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+1)

plot\_image(i, predictions, test\_labels, test\_images)

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)

plot\_value\_array(i, predictions, test\_labels)

#on single

predictions\_single = model.predict(img)

print(predictions\_single)

plot\_value\_array(0, predictions\_single, test\_labels)\_ = plt.xticks(range(10), class\_names, rotation=45)

np.argmax(predictions\_single[0])