Inhouse Project

Titled

Machine learning (Deep Learning)

Submitted to

Amity University Uttar Pradesh

In partial fulfilment of the requirements for the award of the degree Of

Bachelor of Technology

In

Computer Science and Engineering

By

Atharv Gupta

A2345921019

Under the guidance of

Dr. Krishna Kant Singh

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY AMITY UNIVERSITY, NOIDA ,UTTAR PRADESH, INDIA

DECLARATION

I, **Atharv Gupta**, student of B.tech Computer Science Engineering, hereby declare that the project titled " **Machine Learning** " which is submitted by me to the Department of Computer Science, Amity School of Engineering & Technology, Amity University, Noida, Uttar Pradesh, in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Computer Science & Engineering, has not previously formed the basis for the award of any degree, diploma or other similar title or recognition.

Place: Noida
Name: Atharv Gupta
Date: 22/06/2022
Enrolment No: A2345921019

CERTIFICATE

On the basis of the declaration submitted by Atharv Gupta, student of B.Tech Computer Science & Engineering, I hereby certify that the project titled "Machine Learning " which is submitted to the Department of Computer Science & Engineering, Amity School of Engineering & Technology, Amity University, Noida, Uttar Pradesh, in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Computer Science & Engineering is an original contribution with existing knowledge and faithful record of work carried out by them under my guidance & supervision. To the best of my knowledge this work has not been submitted in part or full for any degree or diploma to this university or elsewhere.

Place : Noida
Date : 22/06/2022

_____

Dr. Krishna Kant Singh
Department of Computer Science & Engineering
Amity School of Engineering & Technology,
Amity University,
Noida, Uttar Pradesh

ACKNOWLEDGEMENT

In the accomplishment of completion of my project on " LinkedIn Data Scrapping Tool ", I would like to convey my gratitude to Amity University, Noida for their invaluable guidance, which helped me learn new concepts and update my knowledge in Python and its libraries, which undoubtedly will also be useful in the future.

Also, my heartfelt gratitude to my faculty guide, **Dr. Krishna Kant Singh** Department of Computer Science & Engineering, Amity School of Engineering and Technology, Amity University, Noida, Uttar Pradesh. Without her valuable guidance & suggestions, I would not have been able to complete this project with such efficiency. She helped me in various phases of the completion of this project & I will always be thankful to you in this regard.

Atharv Gupta
A2345921019
B.Tech CSE-Evening
2021 - 2025

TABLE OF CONTENTS

ABSTRACT

Following the emergence of the concept of machine learning, the AI has attempted a new measurement. Every robot understands user's transformation with a significant amount of data. To get about the idea of ML, step by step basic method was used. Using common language handling methods (NLP) to organise the instructive foundation and learn about how a robotic chip reacts to the data provided to it. We thus studied how machine learning is effective in its own ways. A sub part of machine learning is Deep Learning. We will go deeper into this topic and learn how it is used, and where it can be helpful.

Introduction

The aim is to learn deep learning in machine learning. Deep Learning is an area of machine learning focused with artificial neural networks, which are inspired by the structure and function of the brain. Here, we used Python as our default programming language.

- **Artificial intelligence**

Branch of computer science that tries to imitate human intellect in machines. Deep learning and machine learning are two methods that may be used to accomplish this.

A group of similar techniques known as **machine learning** allow computers to be educated to carry out a certain activity without having to be explicitly programmed.

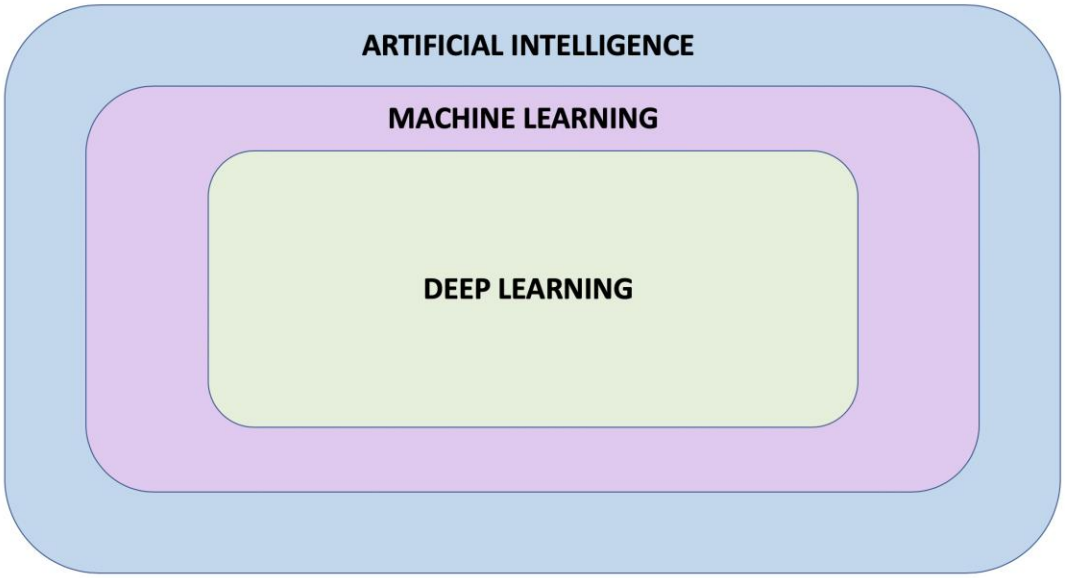
- **Neural Network**

A machine learning concept that takes its cues from the network of neurons (nerve cells) in the human brain. A key component of deep learning is neural networks, which will be addressed in this course.

- **Deep Learning**

A branch of machine learning that makes use of multi-layered neural networks. The terms "deep learning" and "machine learning" are frequently used interchangeably.

NOTE: Machine learning systems never have any previous knowledge to help them solve problems. They learn to solve these types of problems without any prior knowledge.



Deep learning and machine learning both have several branches, subfields, and specialised methods.

The division of ***Supervised Learning and Unsupervised Learning*** is a significant illustration of this variation.

- **Supervised Learning**

Its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately.

- **Unsupervised Learning**

It uses machine learning algorithms to analyze and cluster unlabeled datasets.

There are many types of neural network architectures. However, no matter what architecture you choose, the math it contains is not modified during different tasks. Instead, it is the internal variables (“weights” and “biases”) which are updated during every task.

For instance, the model begins by multiplying the input by a certain amount (the weight) and then adds another number to solve the Celsius to Fahrenheit conversion question. Finding the appropriate values for these variables is the first step in training the model; multiplication and addition are not switched out for another operation.

Literature Review

Methodology and Implementation

TensorFlow will get some example values in Celsius (0, 8, 15, 22, and 38) and their equivalent values in Fahrenheit (32, 46, 59, 72, 100). After that, we train a model to deduce the aforementioned formula through training.

First, we import *TensorFlow as tf for ease of use*.
Next, we import *NumPy as np.* (NumPy helps us to represent our data as highly performant lists.)

```
import tensorflow as tf
```

```
import numpy as np
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

The main goal of supervised machine learning is to devise an algorithm from a collection of inputs and outputs. We build two lists, ***celsius list and fahrenheit list***, which we use to train our model since the aim in this example is to develop a model that can calculate the temperature in Fahrenheit given the degrees in Celsius.

```
celsius_list    = np.array([-40, -10,  0,  8, 15, 22,  38],  dtype=float)
fahrenheit_list = np.array([-40,  14, 32, 46, 59, 72, 100],  dtype=float)

for i,c in enumerate(celsius_q):
  print("{} degrees Celsius = {} degrees Fahrenheit".format(c, fahrenheit_list[i]))
```

```
-40.0 degrees Celsius = -40.0 degrees Fahrenheit
-10.0 degrees Celsius = 14.0 degrees Fahrenheit
0.0 degrees Celsius = 32.0 degrees Fahrenheit
8.0 degrees Celsius = 46.0 degrees Fahrenheit
15.0 degrees Celsius = 59.0 degrees Fahrenheit
22.0 degrees Celsius = 72.0 degrees Fahrenheit
38.0 degrees Celsius = 100.0 degrees Fahrenheit
```

Here, we have our data in the form of a table showing some relation to the other half.

Thus, we have question and answer to the problem, but the relation is to be established.

Now, we input a layer of neural network where the task will be performed.

```
layer1 = tf.keras.layers.Dense(units=1, input_shape=[1])
```

input_shape = [1] specifies that the input to this layer is a *single value.*
Units = 1 specifies the *number of neurons* in the layer.

Once layers are established, we put them together to create our model. The parameter for the Sequential model specification, which specifies the order of calculations from input to output, is a list of layers.

```
model = tf.keras.Sequential([layer1])
```

It is necessary to assemble the model before training. The model is supplied when it is assembled for training:

Loss function: A metric for gauging how much forecasts depart from the intended result. (The measured discrepancy is referred to as the "loss").

Optimizer function: A method of modifying internal values to lessen loss.

```
model.compile(loss='mean_squared_error',
              optimizer=tf.keras.optimizers.Adam(0.1))
```

The optimizer function computes modifications to the internal variables of the model during training.
The optimizer (Adam) and loss function (mean squared error) employed here are typical for straightforward models.

During training, the model receives values in Celsius, calculates them using the current internal variables (referred to as "weights"), and outputs values that are supposed to be the equivalent in Fahrenheit. The result won't be very close to the accurate figure because the weights are originally chosen at random. The optimizer function determines how the weights should be changed, while the loss function determines the difference between the actual output and the intended output.

The inputs serve as the first argument, while the desired outputs serve as the second. The ***verbose*** option regulates how much output the procedure generates, and the ***epochs*** argument indicates how many times this cycle should be executed..

```
history = model.fit(celsius_q, fahrenheit_a, epochs=500, verbose=False)
print("Finished training the model")
```

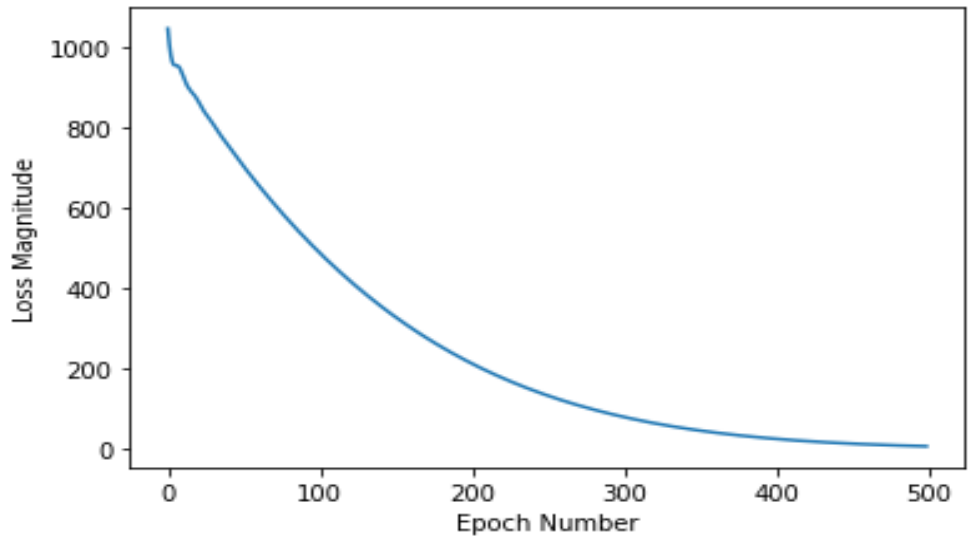
```
Finished training the model
```

A history object is returned by the **fit method**. This object allows us to visualise how our model's loss decreases after each training period.

NOTE: A large loss indicates that the model's predicted value of Fahrenheit degrees differs significantly from the equivalent value in the fahrenheit list.
We used Matplotlib to visualize this.

```
import matplotlib.pyplot as plt
plt.xlabel('Epoch Number')
plt.ylabel("Loss Magnitude")
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f1838924f50>]
```



Now that the link between ***Celsius_list and Fahrenheit_list*** has been learned, a model has been created.

So, for example, if the Celsius value is 100.

```
print(model.predict([100.0]))
```

```
[[211.33772]]
```

The correct answer is 100×1.8+32=212 , so the model is performing well.

To review

1. A model with a dense layer was made.
2. We used 3500 samples to train it (7 pairs, over 500 epochs).
3. Until it could calculate the right Fahrenheit value for each Celsius number, our model had to fine-tune the variables in the Dense layer.

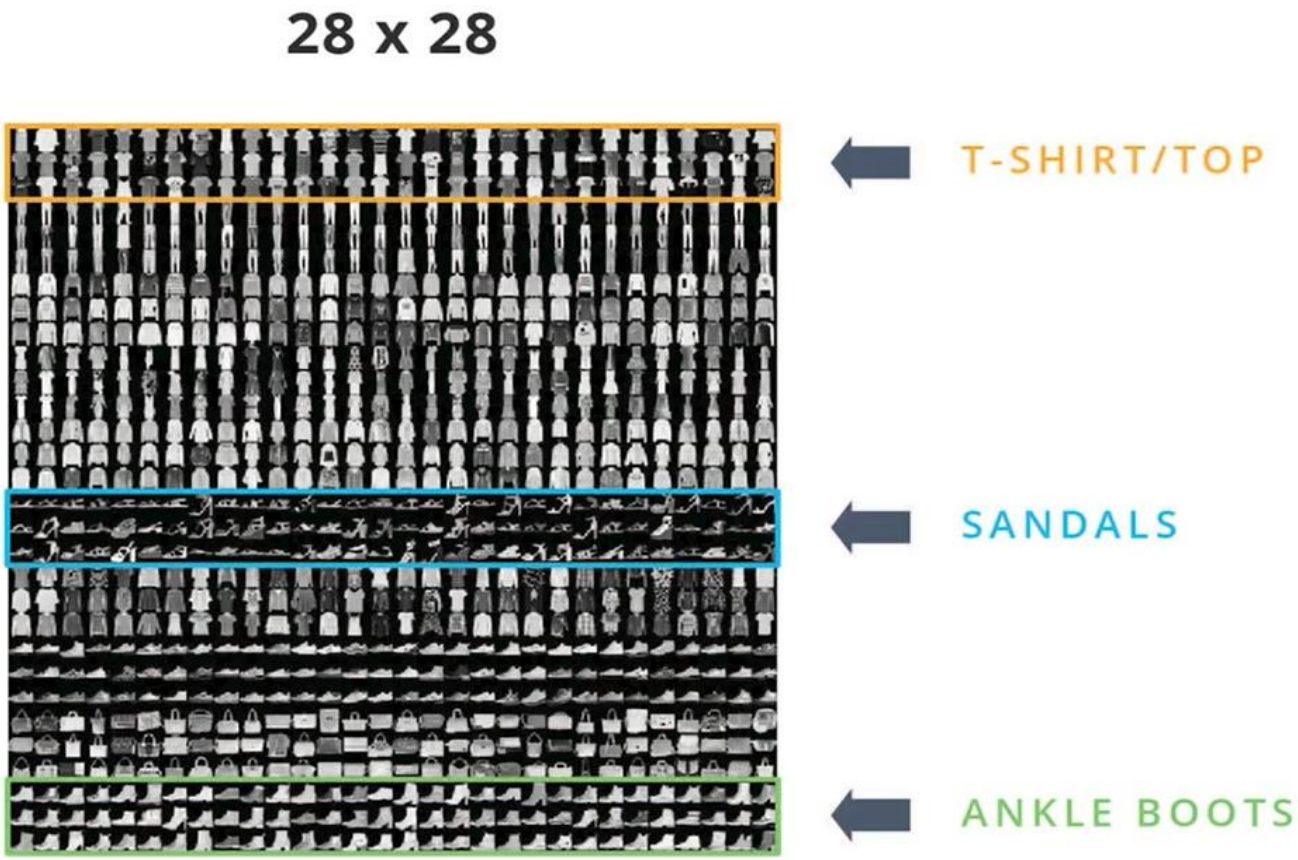
Thus, completing the first example successfully.

In the next section we move to a non-linear complex example with a dense layer.

Fashion MNIST

The fashion MNIST dataset is presented here; it is claimed that it comprises a 28 by 28 pixel grayscale picture of several types of clothes.

It includes pictures of tops, t-shirts, and even shoes.



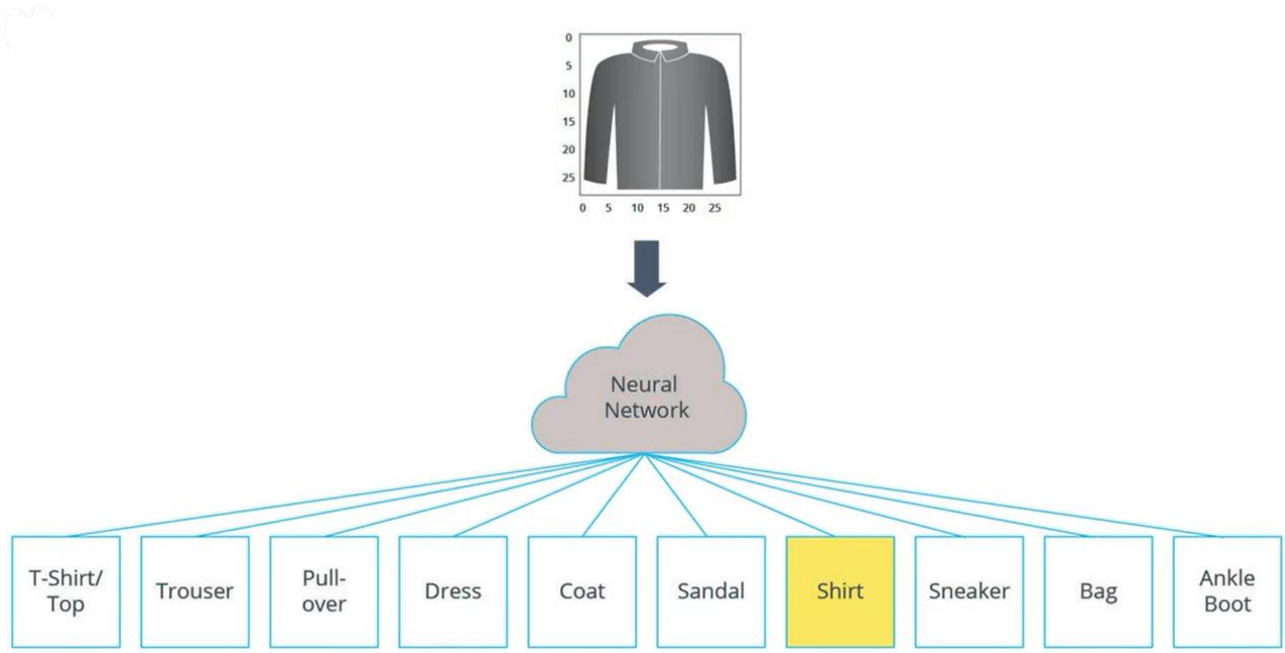
The ReLu network, which we will explore in the next sections, allows the computer to categorise clothing into one of ten different categories based on the input.

To accurately reflect the input supplied, the dataset's 70,000 total photos are used.

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

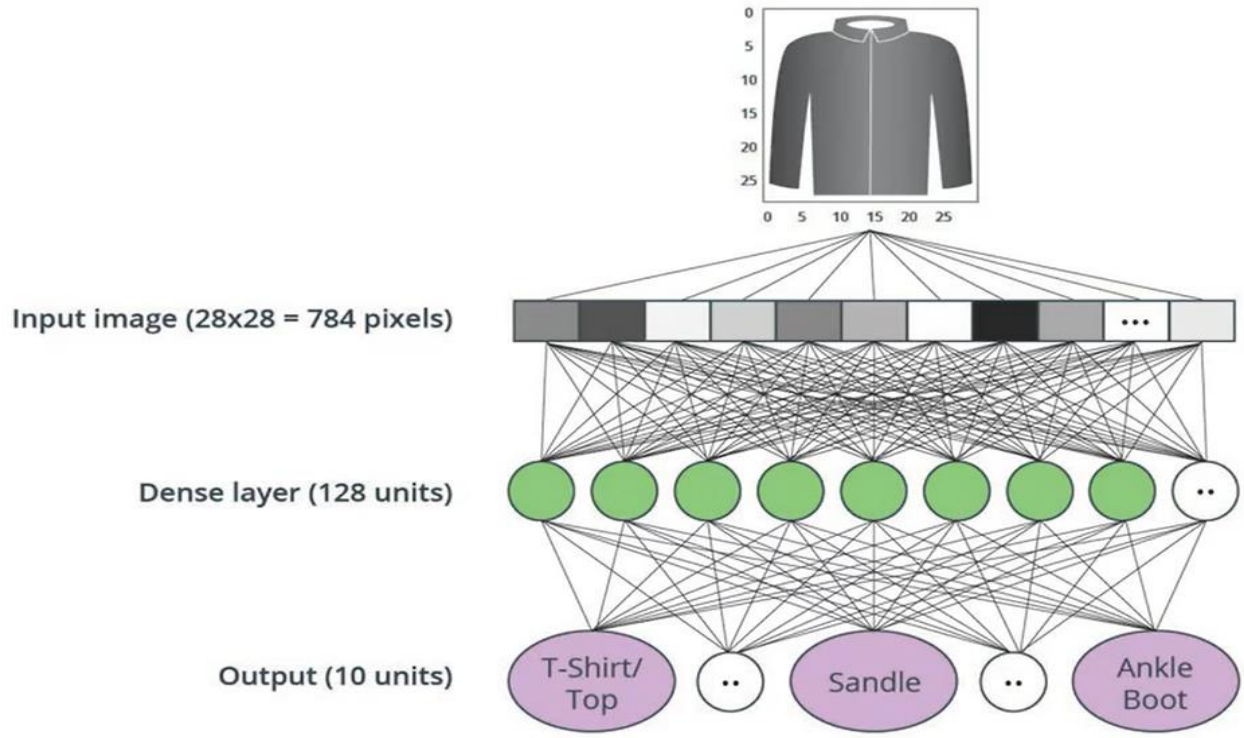
Each image is ***28x28 pixels = 784 bytes (in size)***

In order to determine which of the 10 various types of apparel a given set of 784 bytes of input represents, we need a neural network.



The next step is to build a neural network that operates similarly to the model shown above.

28x28 is equal to 784 units (2D Image)



Due to each image's 28 pixel height and 28 pixel width, the input to our model is an array of length 784. Since a vector is what our neural network uses for a picture.

Flattening is the process of transforming a 2D picture into a vector.

Code for this process is:-

```
tf.keras.layers.Flatten (input_shape=(28, 28, 1))
```

A 28 by 28 pixel picture is transformed into a 784 pixel 1D array using this layer.

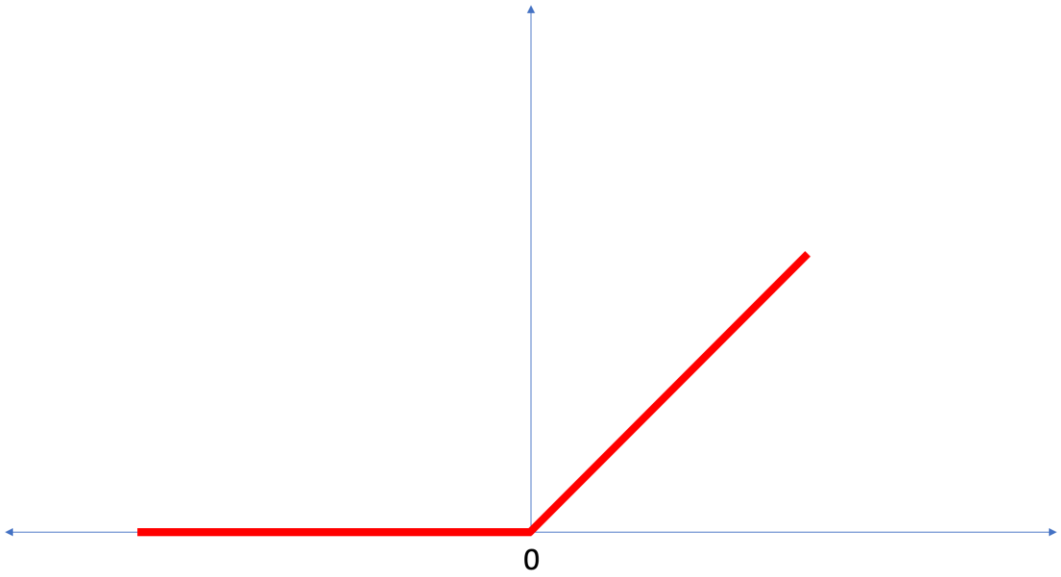
The input will be entirely linked to the network's first dense layer, which will have 128 units.

Code for this process is:-

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

The Rectified Linear Unit (ReLU)

ReLU stands for Rectified Linear Unit and it is a mathematical function.



If the input is positive or zero, the ReLU function will produce an output of 0, and if the input is negative or zero, the output will be equal to the input.

The network can address nonlinear issues thanks to ReLU..

New Terms learnt:

- **Flattening:** The process of turning a two-dimensional picture into a one-dimensional vector.
- **ReLU:** A model's activation function that helps it tackle nonlinear issues.
- **Softmax:** A function that offers probability for every potential output class.
- **Classification:** A machine learning model used to differentiate between two or more output categories.

A selection of datasets available for usage with TensorFlow are provided by **TensorFlow Datasets**.
Typically, datasets are divided into multiple subsets for usage at various phases of neural network training and assessment.

- ***Training Set:*** The data used for training the neural network.
- ***Test set:*** The data used for testing the performance of the neural network.

When training is finished, the validation set is utilised once more to assess the model's ultimate accuracy.

We now import our MNIST dataset for fashion in two sets.

```
dataset, metadata = tfds.load('fashion_mnist', as_supervised=True, with_info=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

Dataset loading yields metadata, a training dataset, and a test dataset.

- The model is trained using train_dataset.
- The model is tested against test_dataset

```
class_names = metadata.features['label'].names
print("Class names: {}".format(class_names))
```

These data must be standardised to the range [0,1] in order for the model to function correctly. Then, using the test and train datasets as examples, we develop a normalisation function and apply it to each image.

```
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)

# The first time you use the dataset, the images will be loaded from disk
# Caching will keep them in memory, making training faster
train_dataset =  train_dataset.cache()
test_dataset  =  test_dataset.cache()
```

The layer serves as the neural network's fundamental building piece. From the data that is supplied into a layer, a representation is extracted. The representation that emerges from a network of interconnected layers should hopefully be useful for the current issue.
Chaining together basic layers makes up a significant portion of deep learning. several levels, such as **tf.keras.layers** Dense, with internal parameters that may be changed (or "learned") during training.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

The model requires a few additional tweaks before it is prepared for training. These are included for building the model:

- ***Loss function*** - A formula for calculating the deviation between the model's outputs and the desired output. This quantifies loss, which is the purpose of training.
- ***Optimizer*** - A method for modifying a model's internal parameters in order to reduce loss.
- ***Metrics*** - employed to keep track of the training and testing processes. The accuracy, or percentage of photos that are properly categorised, is used in the example that follows.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

The iteration behaviour for the train dataset is first defined as follows:

- Repeat indefinitely while defining a dataset.
  repeat() (the epochs parameter described below limits how long we perform training).

- To prevent our model from inferring anything from the samples' order, the dataset.shuffle(60000) function randomlyizes the order.

- Also, dataset. batch(32) informs the model. when updating the model variables, utilise batches of 32 photos and labels.

Training is performed by calling the model.fit method:

1. Using train dataset, feed the model the training data.

2. The model gains experience linking labels to visuals.

2. The epochs=5 parameter limits training to 5 full iterations of the training dataset, so a total of 5 * 60000 = 300000 examples.

```
BATCH_SIZE = 32
train_dataset = train_dataset.cache().repeat().shuffle(num_train_examples).batch(BATCH_SIZE)
test_dataset = test_dataset.cache().batch(BATCH_SIZE)
```

```
model.fit(train_dataset, epochs=5, steps_per_epoch=math.ceil(num_train_examples/BATCH_SIZE))
```

The accuracy and loss metrics are provided as the model moves. On the training set of data, this model achieves an accuracy of roughly 0.88, or 88 percent.

The model's performance on the test dataset is then contrasted.

```
test_loss, test_accuracy = model.evaluate(test_dataset, steps=math.ceil(num_test_examples/32))
print('Accuracy on test dataset:', test_accuracy)
```

After the model has been trained, we can use it to use it to predict certain photos.

```
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)
```

```
predictions.shape
```

Each label for an image in the testing set has been predicted by the model in this instance. Let's look at the initial forecast:

```
predictions[0]
```

The model is very certain that this picture depicts a shirt, or class_names[6]

To view all ten class predictions, we may graph this.

```
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')
```

Let's look at the 0th image, predictions, and prediction array.

```
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)
```

Let's plot a few photographs together with their forecasts. Prediction labels that are accurate are blue, while those that are inaccurate are red. The number indicates the projected label's percentage (out of 100). Keep in mind that even when something is certain, it might still be erroneous.

```
# 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)
```

Finally, we use the trained model to make a prediction about a single image.

```
# Grab an image from the test dataset
img = test_images[0]

print(img.shape)
```

The tf.keras models are designed to make forecasts on a lot, or collection, of instances at once. We must thus include it in a list even if we are just utilising one image:

```
# Add the image to a batch where it's the only member.
img = np.array([img])

print(img.shape)
```

Now we predict the image:

```
predictions_single = model.predict(img)

print(predictions_single)
```

```
plot_value_array(0, predictions_single, test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
```

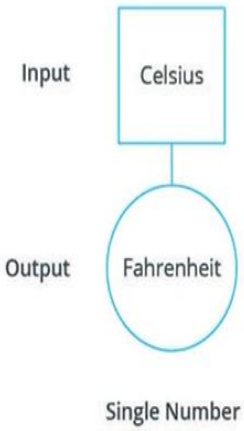
model.predict provides a list of lists, one for each image in the lot of data. Take the predictions for our (only) image in the batch:

np.argmax(predictions_single[0])

And, as before, the model predicts a label of 6 (shirt).

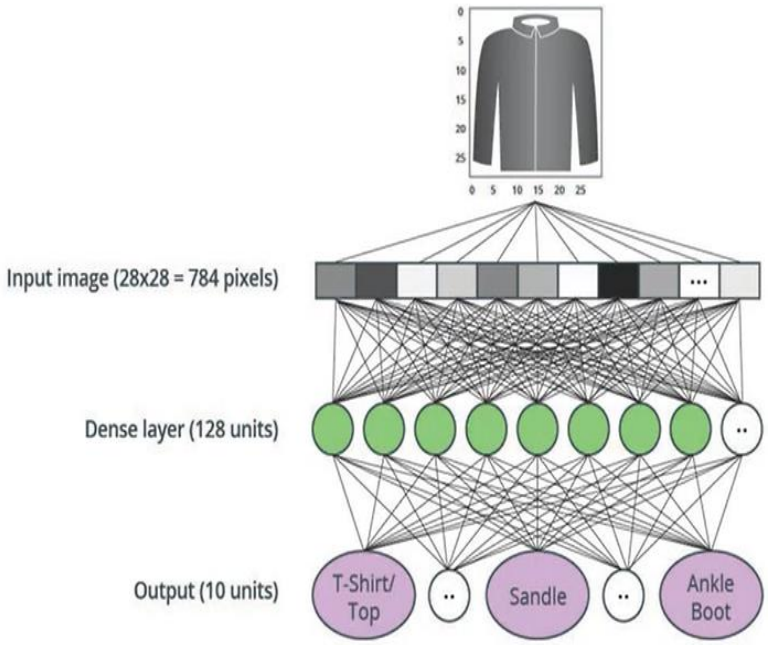
Comparing both the above models:

Celsius to Fahrenheit



Regression
Model output is a number
Loss function: mean_squared_error

Fashion MNIST



Classification
Model output is a set of numbers that represent probabilities (sum is 1)
Loss function: sparse_categorical_crossentropy

	Classification	Regression
Output	List of numbers that represent probabilities for each class	Single Number
Example	Fashion MNIST	Celsius to Fahrenheit
Loss	Sparse categorical crossentropy	Mean squared error
Last Layer Activation Function	Softmax	None

- **Regression**

A model that outputs a single value.

For instance, an estimate of a land's value.

- **Classification**

A model that generates a probability distribution over a number of different categories

For instance, 10 probabilities—one for each style of clothing—were produced by Fashion MNIST. Remember that we generated this probability distribution using Softmax as the activation function in our final Dense layer.

Summary

For the purpose of classifying photos of apparel, we trained a neural network. The Fashion MNIST dataset, which has 70,000 greyscale pictures of apparel items, was utilised to do this. 10,000 of them were utilised to test the network's performance after we had used 60,000 of them to train it. We had to flatten the 28 28 pictures into 1d vectors with 784 components in order to feed them into our neural network. Our network included two layers: an output layer with 10 units, which matched the 10 output labels, and a fully linked layer with 128 units (neurons). For each class, these 10 outputs show the probability. The probability distribution was generated using the softmax activation function.