Introduction

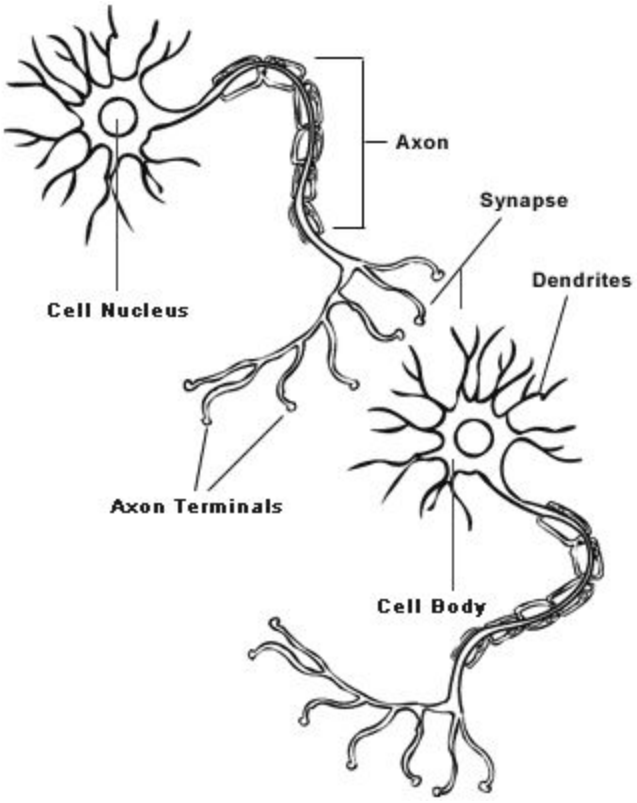
Convolution is a deep learning model that has made creating a custom deep learning model easier and better than normal neural networks. Convolution neural network architecture has made it the go-to model for any image processing deep learning model, it is widely used around the world to create solutions for real-life problems.
In this tutorial, we will be using Tensorflow and python to build our own convolution neural network model from scratch.

After you finish reading this tutorial, you should be able to know more about:

- What a neural network is
- How a neural network works.
- Convolution neural network
- Convolution neural network architecture.
- Implementing a convolution neural network from scratch.

What is a Neural Network

A neural network is said to be a computing system that consists of a number of simple but highly interconnected processing elements known as the neurons. These neurons make a series of transformations on user data in order to understand the data and make predictions from them.



The neural network is based on the structure of the human brain, it tries to copy lots of densely interconnected brain cells inside a computer so you can get it to learn and recognize patterns and also act based on those patterns. Neural networks learn on their own just like the human brain, they are not explicitly programmed to learn. The neural network takes in some input data and tries to learn the pattern on that data to make a good and targeted prediction, it can be used for both supervised and unsupervised problems.
Representation of a Neural Network

Basically, a neural network is composed of 3 types of layers:

- **Input layer:** This is the layer in which our data passes through, they receive input from the system in which the neural network model will try to learn about, the data can be structured and unstructured data depending on the problem we are trying to use the neural network to solve.
- **Hidden layer:** This layer is between the input and output layer, the complexity of our neural network model depends on the number of hidden layers we use. I start with one and then try to add more layers depending on the performance of the initial model. The hidden layer is responsible for learning the mapping between the inputting and output data.
- **Output layer:** The output layer contains the output produced by the neural network.

Most neural networks are fully connected, this means each hidden layer is fully mapped to the input layer and output layer on either side of the neural network. Each neuron is connected by a value called weights which are either positive or negative, the neuron with the highest weight will have more influence over another.
The formula for a neural network is

$$output = relu((W * Input) + b)$$

W and b are weights and biases, these are multidimensional matrices that are attributed to the layer. **Relu** is an activation function for flattening the output from one layer and passing it as an input to another layer.
We do a matrix multiplication between the input and weights, add the result of the matrix multiplication to the bias and pass it through the relu activation function, the activation function replaces all the negative values with zero, weights, on the other hand, contains the information learned by the neural network after being exposed to the training data.
The **weight matrices** are filled with small random values called random initialization, this is to prevent initializing all the values with zero, so as to avoid updating our values with repeated values, this will prevent our neural network from generalizing its learning processes.

Training a Neural Network

When training a neural network we need to discuss the **gradient descent** and the **cost function**. We need to define the cost function and use the gradient descent optimization to minimize it. If we change the weight of any connection in a neural network model, we will see a drastic change in the output from the other neurons, that change will affect all other neurons and activation function in the other layers.
Cost function: This is the measure of how good a neural network will perform with respect to a given training data and the expected target. Cost function depends on values such as weight and biases. It determines how good a neural network model is. The Cost function is in form

C(W, B, Sr, Er)

Where W is our neural network's weights, B is our neural network's biases, Sr is the input of a single training sample, and Er is the desired output of that training sample. The weight and biases are values we often change to improve the performance of our model. The magnitude of the error of a specific neuron (relative to the errors of all the other neurons) is directly proportional to the impact of that neuron's output on our cost function.
Gradient descent: The gradient of a function is the vector whose elements are its partial derivatives with respect to each parameter. Each element of the gradient descent tells us how the cost function would change if we applied a small change to that particular parameter. We aim for the gradient descent to reach the deepest slope. Steps for the gradient descent are as follows:

- Compute the gradient of the current location.
- Modify the parameters by a value proportional to its gradient element and in the opposite direction of that element.
- Recompute the gradient using the new parameters
- Repeat the steps.

Forward propagation: This is the process of moving forward through the neural network, movement if from the input through the hidden layer, through the activation functions,

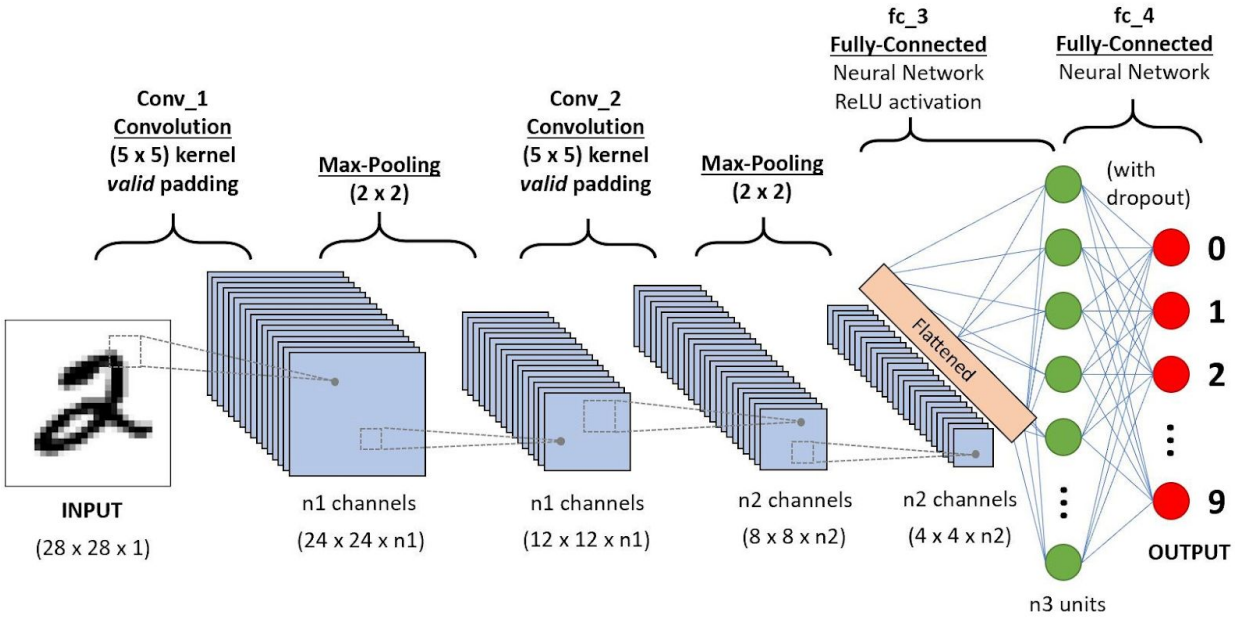
weights and biases to the output layer. The objective of the forward propagation is to calculate the activations at each neuron for every hidden layer to the output layer.

Backward propagation: This is the process of moving the errors backwards through the neural network model and passing through the same weights and biases we use during the forward propagation step. We calculate the error value to each neuron starting from the layer closest to the output all the way back to the input layer of the model.

We have seen what neural networks are and how they work. There is much to discuss on neural networks but that will not be discussed here as this is not in the scope of the tutorial. We are going to discuss convolution neural networks, another type of neural network. They are usually more effective than the neural networks and are mostly used in image processing applications.

Convolution Neural Network

A convolution neural network is very similar to an ordinary neural network or it can be described as an advanced neural network that is made up of neurons with learnable weights and biases.
Convolution neural networks already assume that the inputs are images, and it then encodes some important features into the architecture making the function very efficient to implement thereby reducing the properties in the network.



How Convolution Neural Network Works?

In this section, we will be looking at how the convolution neural network works.

Convolution Neural Network Architecture

The convolution neural network is a deep learning algorithm that takes in images as input, assigns weights and biases to various features in the images so that they can be differentiable from one another. The convolution neural network consists of

- Convolution Layers
- Pooling Layers
- Fully-Connected Layers

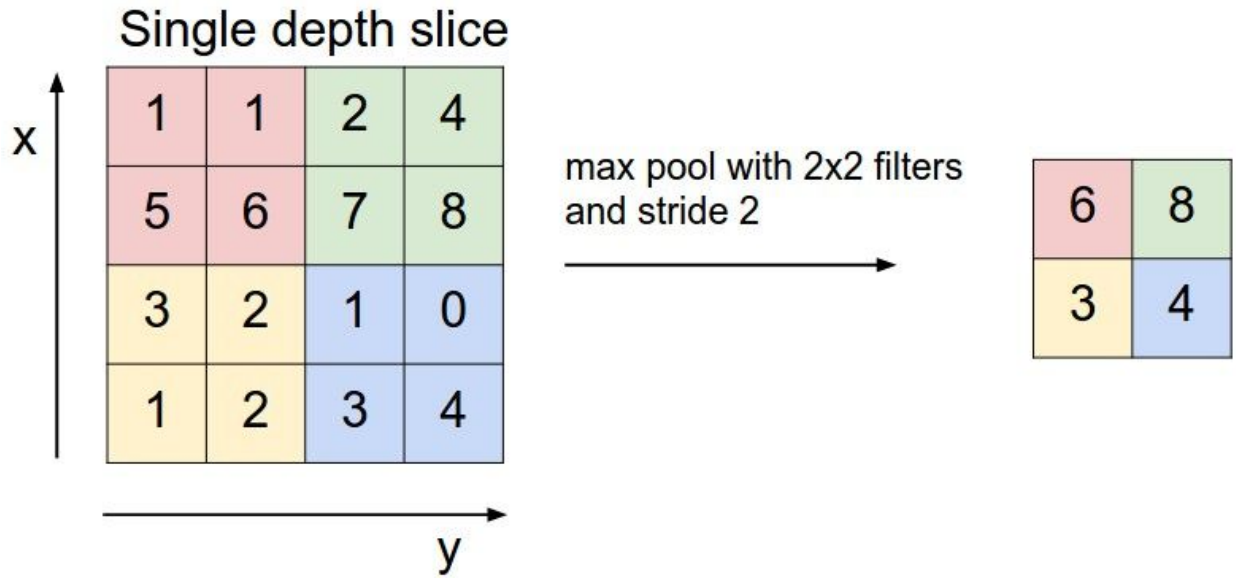
Convolution Layers: This is the major building block used in a convolution neural network that does most of the heavy task. The convolution layer parameter consists of a set of learnable filters. The filters are usually small but are moved to cover the full depth of the input pixel. E.g a filter of a convolution network might have a size of 3x3x3 (3 pixels width and height and the last 3 to represent the colour channel). The way this works is that we slide the filter across the width and height of the input and then compute the dot products between the entries of the filter and input at any position. When we move the filter over the width and height of the input, we are in turn producing a 2-dimensional feature map that responds to that filter at any spatial position convolution layer also depends on some features for it to work well. They are:

- **Depth:** This is the number of filters we would want to use base on how we see fit. Each of the filters has a responsibility of learning distinct useful features from the input.
- **Stride:** We slide the filter using the stride if we set the stride to 1, we are set to move the filter one pixel at a time, if the stride is set to 2, then the filter will move 2 pixels at a time and so on. The stride determines how we want to move the filters over the input. Smaller strides work better in practice.
- **Filters:** The filters have input weights and produce and output value. The size of the input is a fixed square called a patch
- **Feature Maps:** This is the output of a filter applied to the previous layer, A filter is applied to a layer by drawing it across the whole previous layer, drawn a pixel at a time with each position resulting in activation of the neuron and output stored in the feature maps.
- **Zero Padding**: This is the addition of zeros to the borders of the input to provide convenience when moving the filters across the edges of the inputs. Zero padding allows us to control the spatial size of the input volumes. Padding keeps the spatial sizes constant and this improves the performance of the convolution, if the convolution network was not zero-padded, the size of the volume will be reduced by a small amount after each convolution and we will have washed out information.

Polling Layers: The work of the pooling layer is to reduce the feature map of the previous layer. The pooling layer is almost always found in the convolution network architecture, it reduces the spatial size of the representation to reduce the number of parameters and computation in the network. This helps with reducing overfitting in the model. It involves selecting a pooling operation, the size of the pooling layer is always smaller the size of the feature maps. If the pooling layer is a 2x2 pixel it means that the size of the feature map will be reduced by 2. The result is somewhat halved. If the 2x2 pooling layer is applied to the feature map of 8x8 (64 pixels), it will produce an output feature map of 4x4 (16 pixels). The value for the pooling layer is assigned rather than learned from the network. There are two common types of pooling layers.

- **Average pooling:** Calculate the average value for each filter in the feature map. This was often used in the [ast but the max-pooling has proven to work better.

- **Max pooling:** Calculate the maximum value from each patch of the feature maps.



The pooling layer helps in making the representation become approximately invariant to small translations of the input.

Fully Connected Layers: This is the final stage of the convolution network, fully connected layers are used to connect our features for making predictions on the convolution network. The fully connected layer contains a non-linear activation function or a softmax activation in order to produce class/label predictions.

Types of Convolution Neural Network.

- **LeNet**. The first successful applications of Convolutional Networks were developed by Yann LeCun in the 1990s. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.
- **AlexNet**. The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error).
- **GoogLeNet**. The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google. Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).
- **VGGNet**. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance.
- **ResNet**. Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015. It features special *skip connections* and heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network.

Hands-on Implementation

Now we are going to see how CNN is implemented with Python and TensorFlow. Python and TensorFlow are the two primary languages for building a deep learning network. There are other libraries that can be used also like PyTorch, fast.ai etc.

We are going to build a deep convolution neural network to predict the clothing wear from the fashion MNIST data set.

The Data

The fashion MNIST data contains a total of 70,000 grayscale images and contains over 10 classes. Each image can be categorised into one of the 10 classes, each image is a 28x28 pixel low-resolution image, 60,000 images were used for training the convolution neural network while 10,o00 images were kept for testing the model.
The classes are as follows

1. T-shirt/top
2. Trouser
3. Pullover
4. Dress
5. Coat
6. Sandal

7. Shirt
8. Sneaker
9. Bag
10. Ankle boot

Install Tensorflow

Firstly we will need to install TensorFlow. The latest version for TensorFlow as of writing this tutorial is tensorflow2. It can be installed on your notebook as

```
!pip install tensorflow==2
```

Import Packages

```
from __future__ import absolute_import, division, print_function, unicode_literals

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

From the above we import the necessary packages needed to build the model, we import the TensorFlow, Keras, datasets, layers and models all from the TensorFlow library. The Tensorflow dataset is the library we are going to download our data from. Layers is for building the models.

Next, we download the data and preprocess the data.

```
# Download the data from the keras dataset library
fashion_mnist = keras.datasets.fashion_mnist

# Split the data into train and test and load them into a numpy array
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Here, we download the data from the keras.datasets api, split the data into train and test data along with their labels, assign the class names in a list.

```
print('The total number of training images is {}'.format(train_images.shape))
print('The total number of test images is {}'.format(test_images.shape))

The total number of training images is (60000, 28, 28)
The total number of test images is (10000, 28, 28)
```

We can confirm from the train and test shape that we have 60,000 28x28 pixel images for training and 10,000 28x28 pixel images for testing.
Next, we scale the images so the pixels can be represented between 0 and 1

```
# Scale the images by dividing each images value by 255.0
train_images = train_images / 255.0

test_images = test_images / 255.0
```

Next, we reshape the image

```
train_images_reshaped = train_images.reshape(-1, 28, 28,1)
test_images_reshaped = test_images.reshape(-1,28,28,1)
```

Model Architecture

Creating a convolution neural network is similar to that for a neural network, It's a simple and straightforward process that can be achieved in less than 10 lines of code depending on the number of layers you need.

```
model = tf.keras.models.Sequential([
 tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(28, 28, 1)),
 tf.keras.layers.MaxPooling2D(2, 2),
 tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
 tf.keras.layers.MaxPooling2D(2,2),
 tf.keras.layers.Flatten(),
 tf.keras.layers.Dense(128, activation='relu'),
 tf.keras.layers.Dense(10, activation='softmax')
])
```

```
model.summary()
```

We design our convolution neural network, The model contains convolution layer, a fully connected layer and an output layer, the first convolution layer takes in 64 nodes, a 3x3 convolution filter to serve as the filter, a relu activation function to flatten the output. This layer is accompanied by a 2x2 Max pooling layer. The second convolution layer contains the same parameter as the first convolution layer. The fully connected layer connects the convolution layer to the dense layer and uses the relu activation function. Since we have 10 outputs, we have a dense output model with a softmax activation function.

Compile and Fit the Model

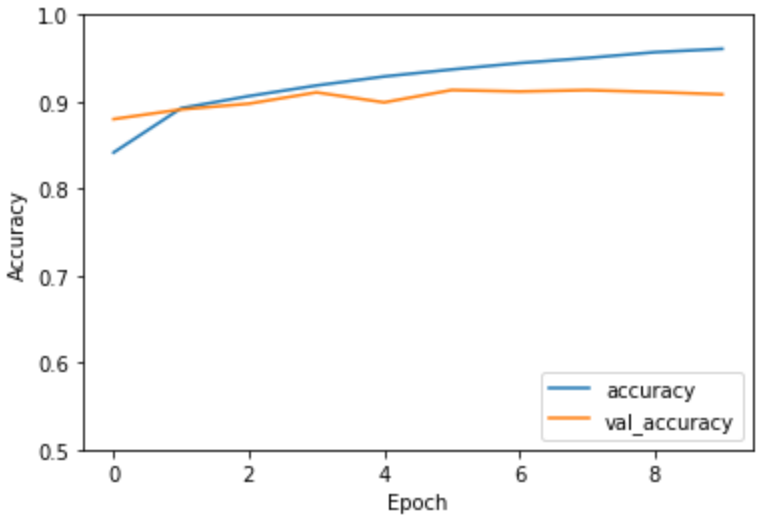
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history=model.fit(training_images_reshaped, training_labels, epochs=10,
validation_data=(test_images_reshaped, test_labels))
```

We compile the model using the ADAM optimizer and a sparse categorical cross-entropy and accuracy serve as our preferred metric. After that, we train the model on the train and test dataset.

Plot the train and validation accuracy

```
# Plot the model training accuracy and validation accuracy
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```

The code above is used to plot the training history of our model on both the validation and test data.



From the graph, we see a steady increase in both the train and validation accuracy, this is an indication of a good model. The training accuracy was 96% compared to 91% of the test data.
You can find the link to the Colab notebook here.

Further Reading
Convolution neural networks are a vast topic and I possibly cannot touch on every topic out there. But I will be dropping some resources here that might be useful to you on your advanced studies. There are also other sites on the internet you can follow to learn more about convolution networks

1. CS231n Convolutional Neural Networks for Visual Recognition
2. Crash Course in Convolutional Neural Networks for Machine Learning
3. Convolutional Neural Networks cheatsheet
4. A step-by-step neural network tutorial for beginners
5. Neural Networks
6. Understanding Neural Networks

Summary

In this tutorial, you learnt about

- Neural networks.
- How a neural network works.
- Convolution neural network and how it works.
- Convolution neural network architecture.
- Types of Convolution neural networks.
- Implementing a convolution neural network with Python and TensorFlow.

Thanks for reading through this post, I hope your curiosity was satisfied. Also, if you have any questions do not hesitate to ask in the comment section and I will attend to them with immediate effect.