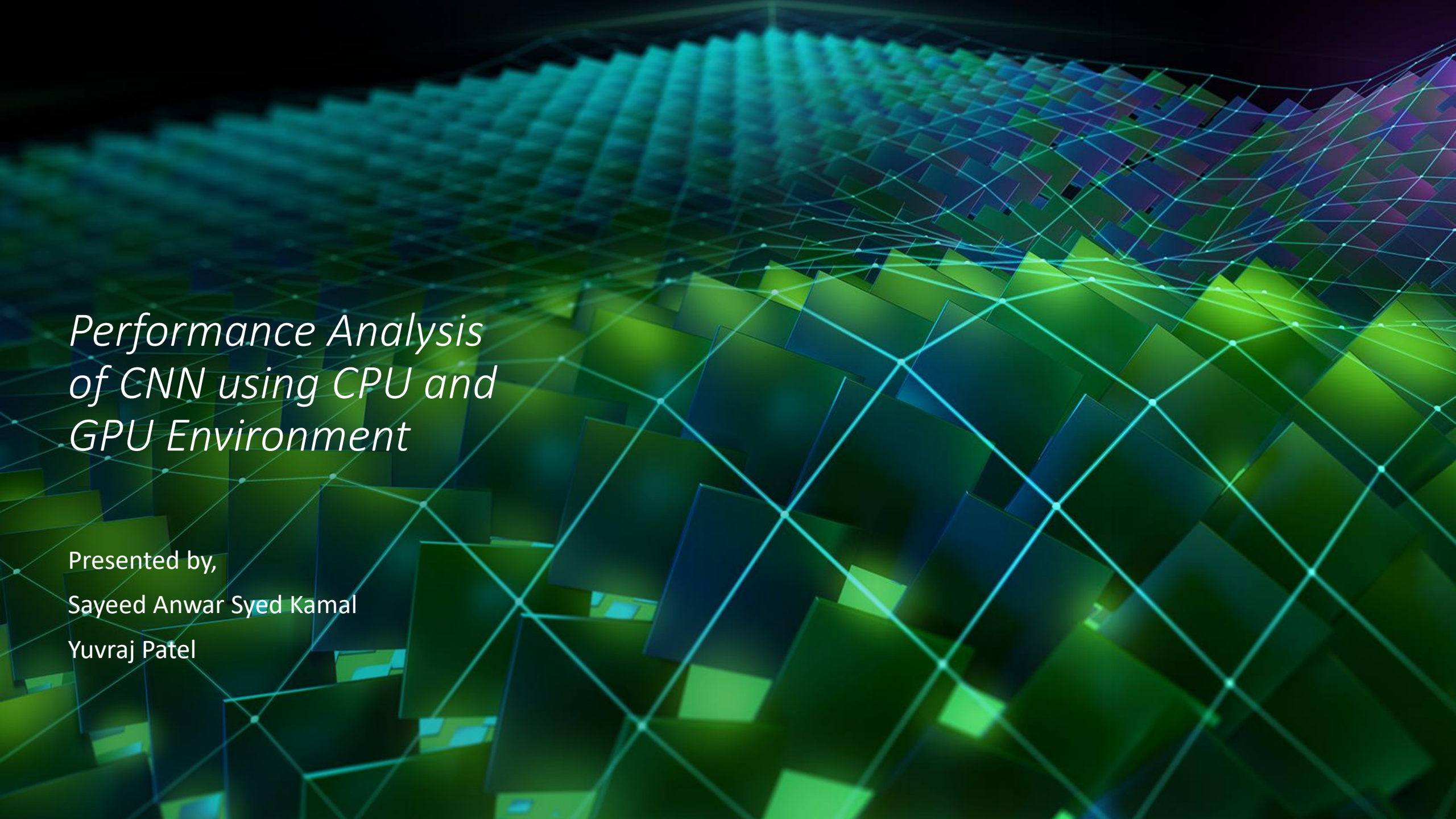


Performance Analysis of CNN using CPU and GPU Environment

Presented by,

Sayeed Anwar Syed Kamal

Yuvraj Patel

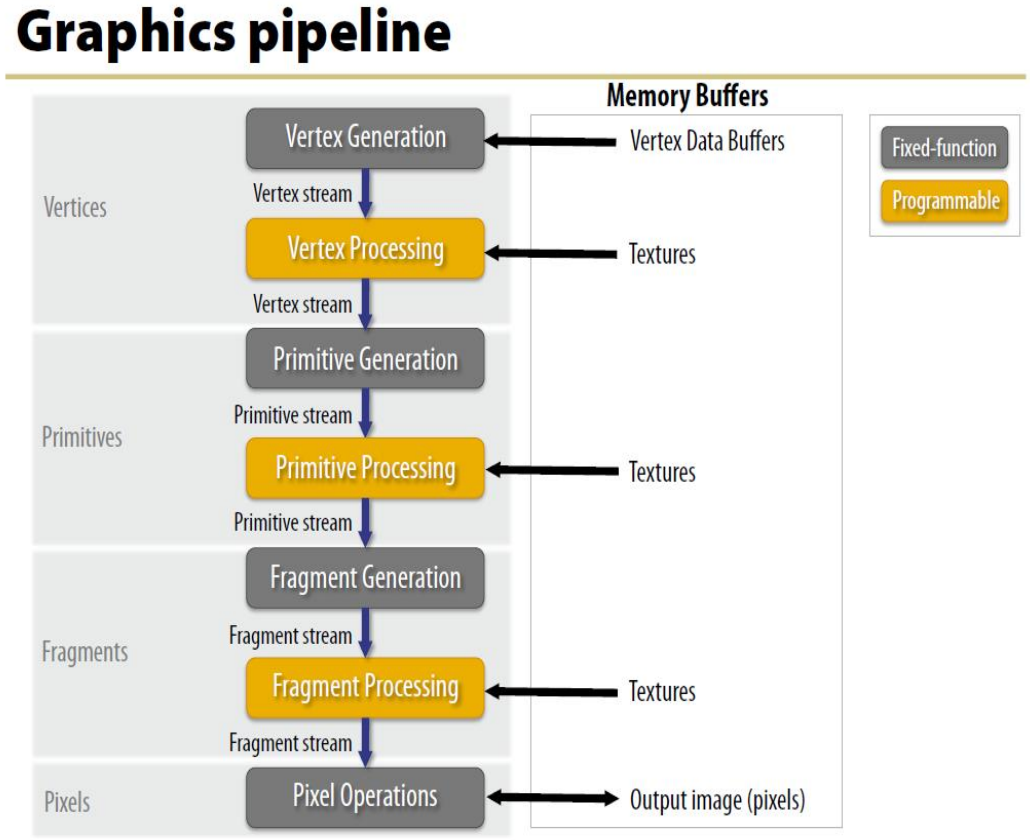
Contents

Introduction

- CNN is a class of deep learning neural networks. It is a huge breakthrough in the field of image recognition.
- They can be found at the core of everything from Facebook to self driving cars.
- They are fast and efficient.
- They are mainly used for image classification.
- Image classification is the process of taking an input (image) and outputting a class (cat or dog) or the probability that the input belongs to particular class.

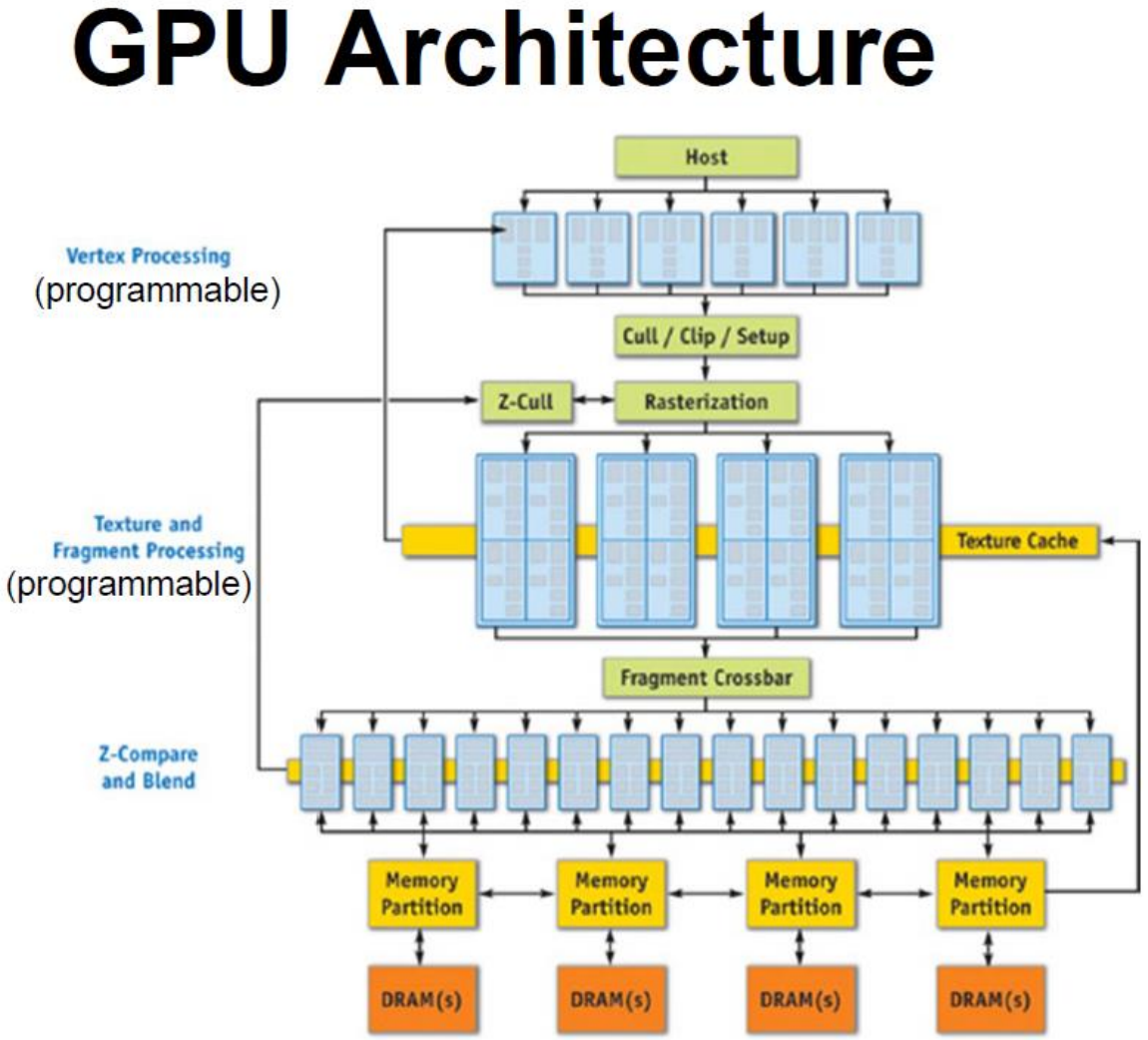
GPU Working

- Vertex Processing - Vertex Transformation. Each vertex is transformed independently.
- Primitive (Triangles) Processing - Vertexes are organized into Triangles.
- Rasterization - Process of converting a given image described by its vector graphic format into raster image. Each primitive is rasterized separately.
- Fragment processing - Fragments are shaded to compute a color at each pixel. Each fragment is processed independently.
- Pixel operation - Fragments are blended into the frame buffer at their pixel location.



GPU Architecture

- GPUs utilize parallelism and pipelining more effectively than a general-purpose CPU.
- Shader processors are generally SIMD, single instruction on every vertex or fragment.
- Modern GPUs have a unified Shader cores. Dynamic task scheduling helps to reduce the load on all cores. Example: Frames with more edges require more vertex shaders and Frames with more primitives require more fragment shaders.
- Unified Shaders are more efficient as they limit the number of idle shader cores, Instruction set is shared among all the shader cores, program determines the type of shader.



CUDA

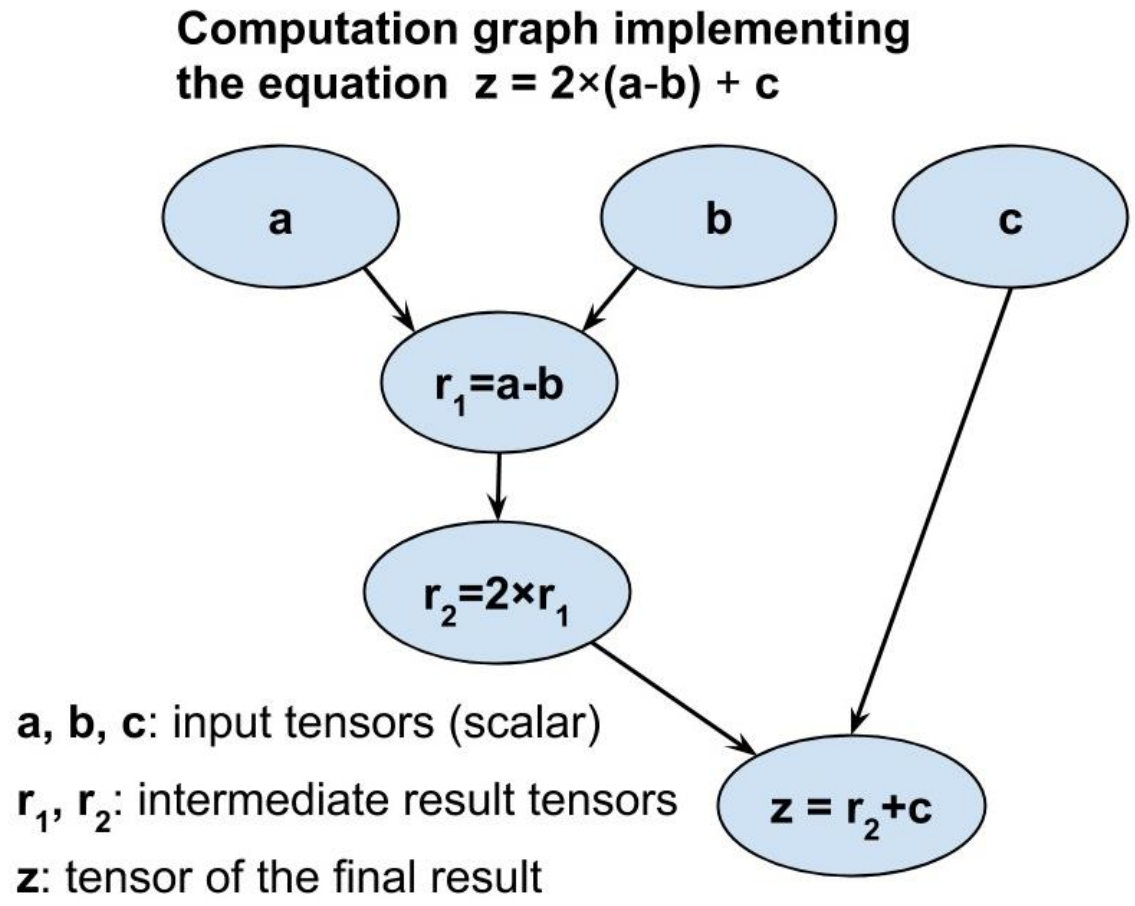
- Compute Unified Device Architecture (CUDA), It is a GPU programming language released by Nvidia. It is written in standard C language with some extensions related to GPU computation.
- The input data is transferred to the GPU as a texture or a vertex values. The computation is then performed by the vertex or fragment shader. The vertex or fragment shaders performs a routine for every vertex or fragment.
- GPU programs can gather or read data from the DRAM but cannot write to specific locations in the DRAM. CUDA features a parallel data cache or on-chip shared memories with very fast general read and write access, that threads are used to share data with each other. Thus, applications can take advantage of it by minimizing over fetch and round-trips to DRAM.
- A thread is basic unit of execution and a thread block is a batch of threads ,that cooperate efficiently by sharing data through some fast-shared memory and synchronize their execution to coordinate memory accesses.

TensorFlow and Tensors

- TensorFlow is a framework that defines and runs computations involving tensors. TensorFlow represents tensors as n-dimensional array of base datatypes. Base datatypes could be a float32, int32, or a string.
- A tensor is a mathematical object analogous to vector but in a more general form. It is represented by an array of components that are functions to the coordinates of a space.
- The main object you manipulate and pass around is the tf.Tensor. TensorFlow programs work by first building a graph of tf.Tensor objects, detailing how each tensor is computed based on the other available tensors and then by running parts of this graph to achieve the desired results.
- A Tensor has the following properties:

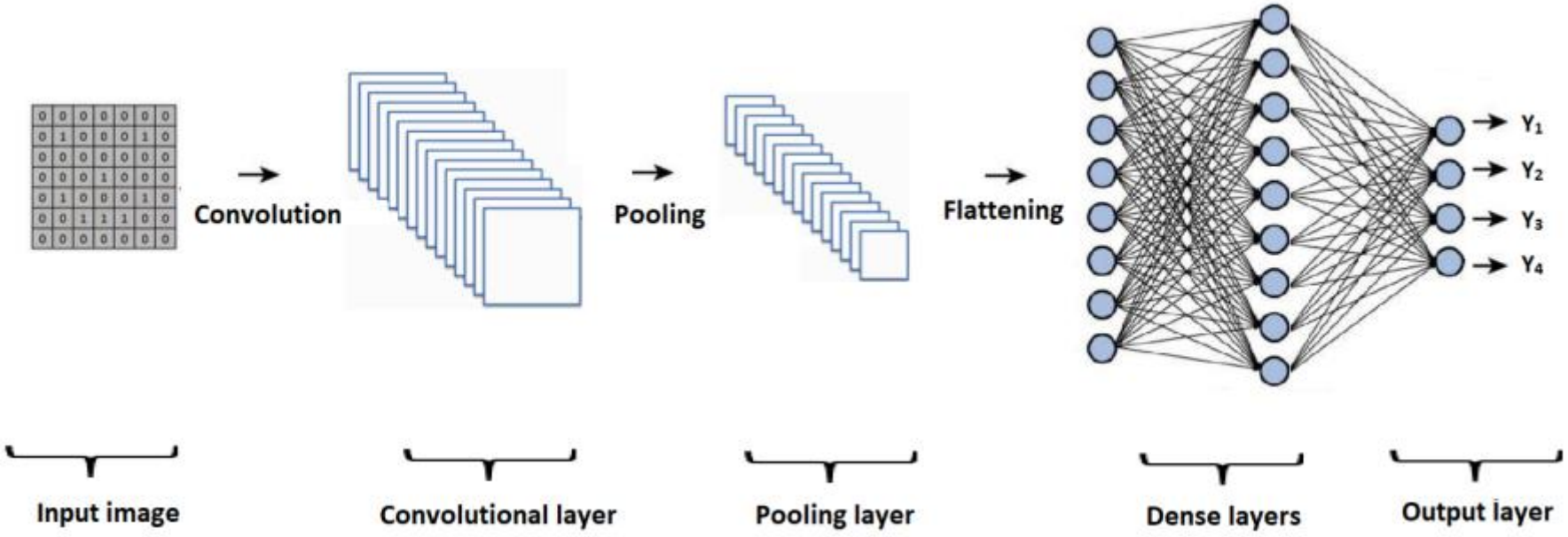
  a) Data Type

  b) Shape



Convolutional Neural Network (CNN)

- A CNN can successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The network can be trained to understand the sophistication of the image better.
- An RGB image which has been separated by its three-color planes — Red, Green, and Blue. It gets computationally intensive as we increase the image dimensionality. For example: 4K images.

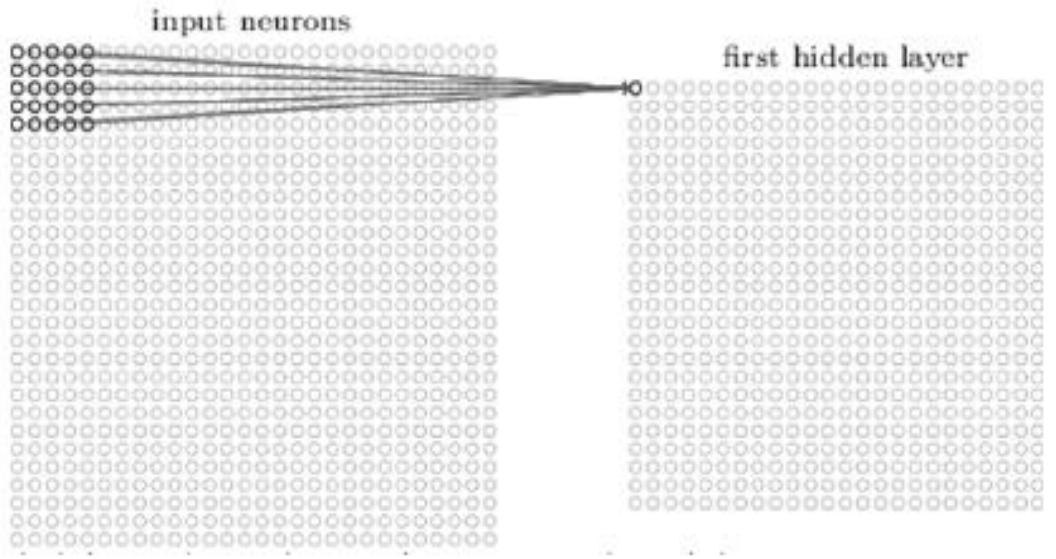


Why CCNs?

- A 400X400 image in 3 channels (RGB) - 480000 input neurons for MLPs
- Hidden layer of 1000 neurons- 480 million parameters!!!
- Take advantage of the spatial relationship of the pixels.
- Each hidden unit is connected to a section of neurons in the input layer
- Using CNNs, with a kernel size of 40 X 40, we have

1600 X 5 (F) X 3 (C) = 24,000 weights !!

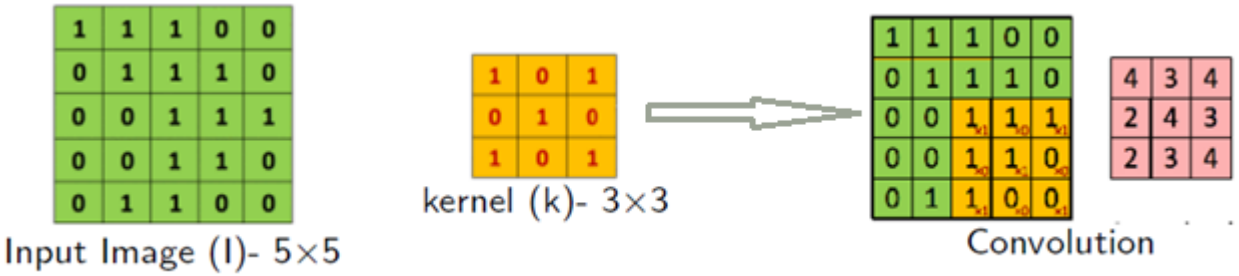
- 3 main layers in CNNs - Convolution, Pooling and Fully connected layer



Convolution layer

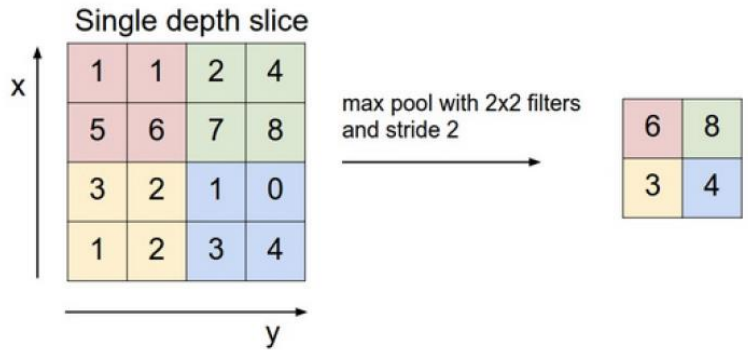
- Reduces the dimensionality of an image without losing the features that are critical for getting a good prediction.

- The kernel slides across the input and sum of element wise multiplication of input with the kernel results in feature map

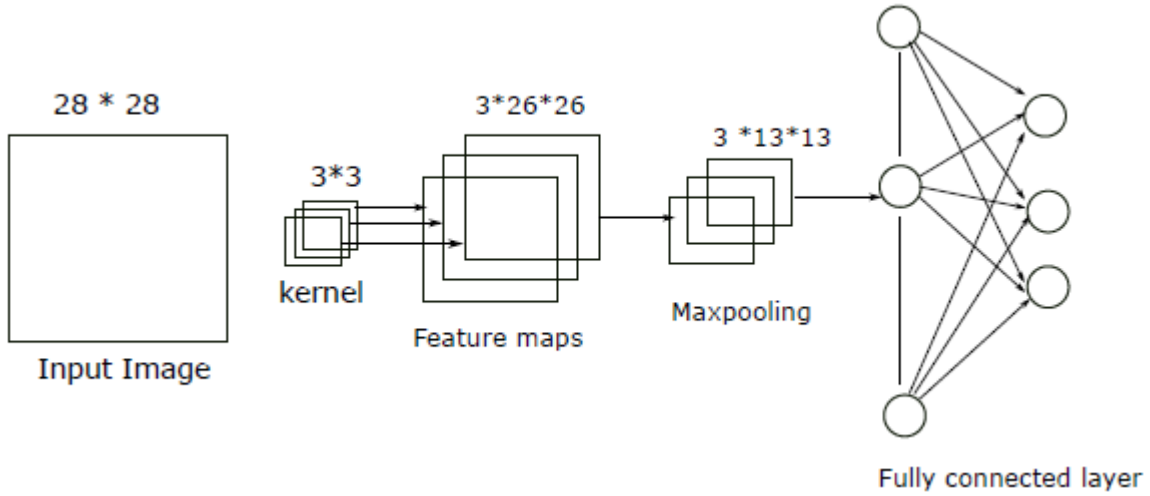


Pooling Layer

- It is responsible for reducing the spatial size of the Convolved Feature.
- It extracts the dominant features which are rotational and positional invariant, which doesn't affect the training.
- Max Pooling also performs as a Noise Suppressant.

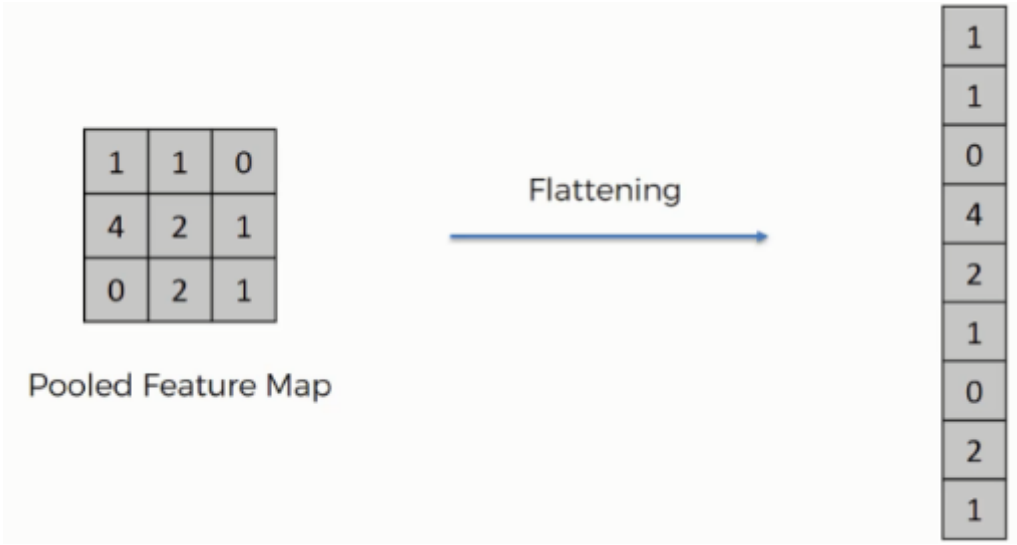


Maxpooling



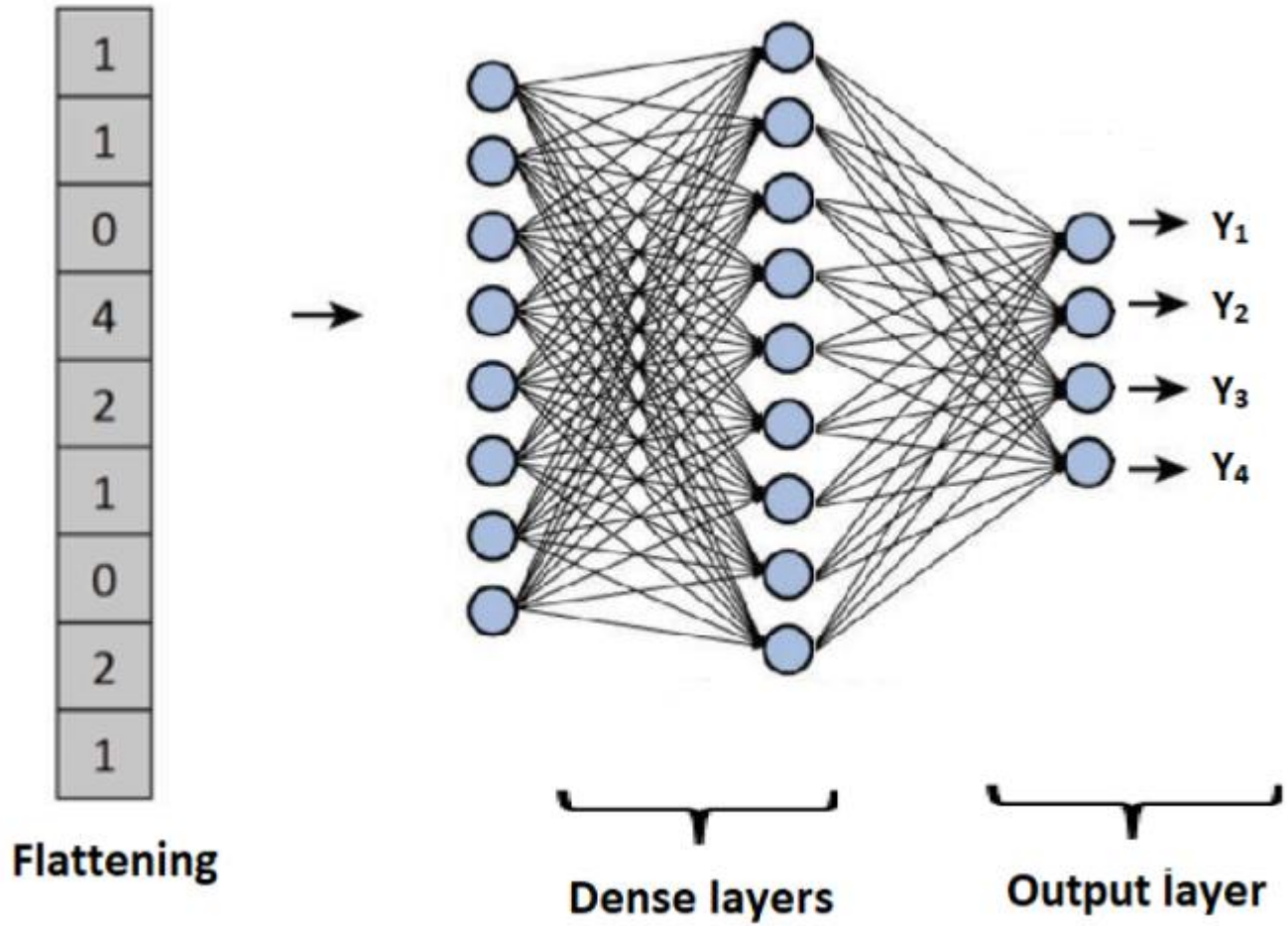
Flatten layer

- The image is converted to a column vector.
- Flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training.



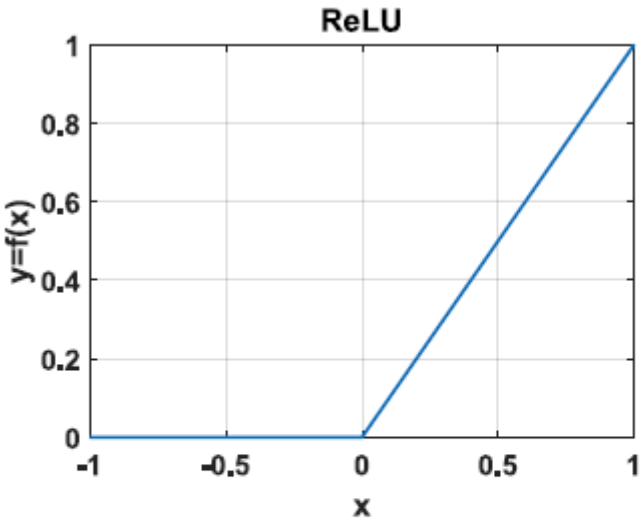
Dense layer

- A linear operation in which every input is connected to every output by a weight.



ReLu activation function

- The Rectified Linear Unit (ReLU) activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero.
- It is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.
- It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

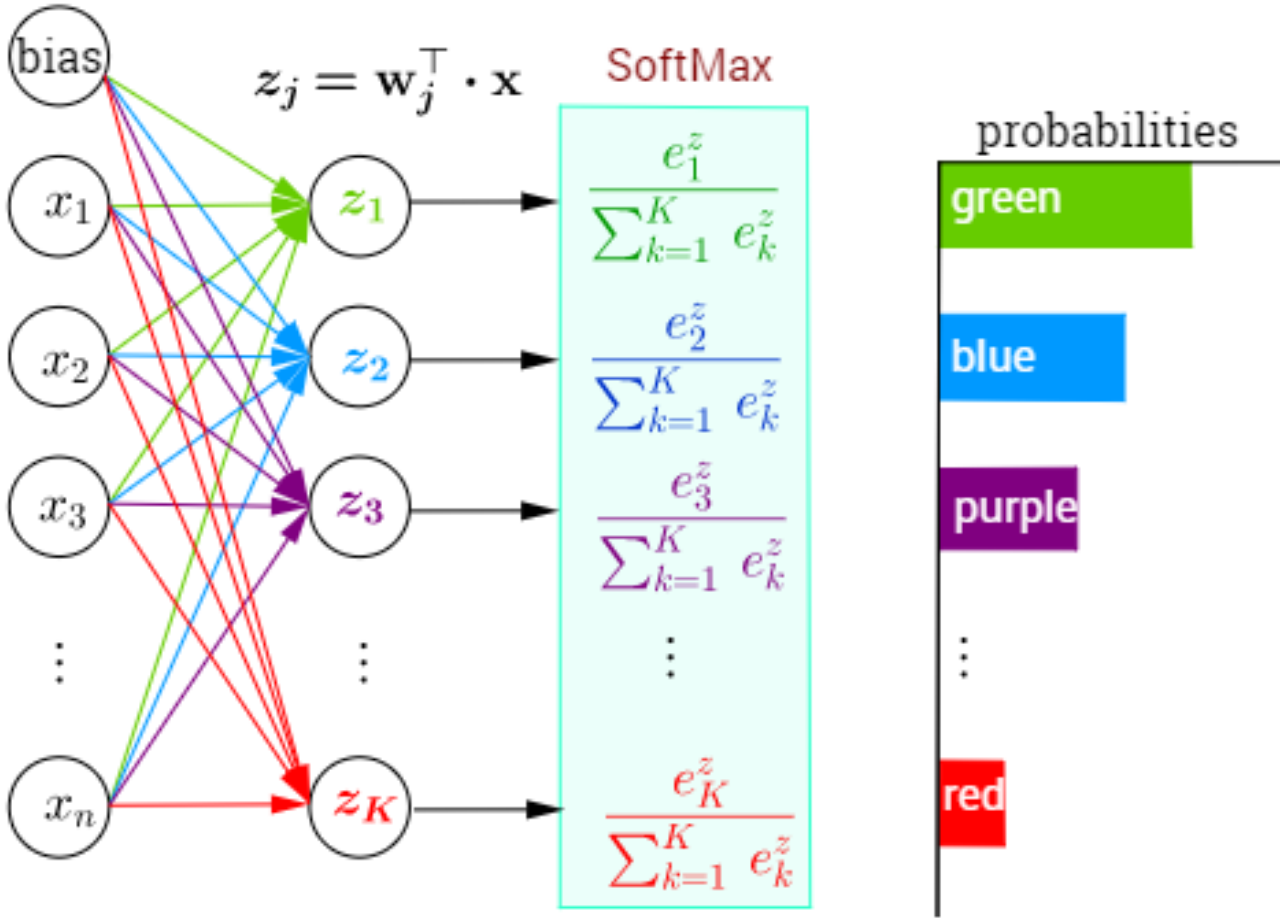


$$f(x) = max(0, x)$$

Softmax fucntion

- SoftMax function: The SoftMax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1.

- We can accommodate many classes since the output values are between the range [0,1].



Implementation of CNN using MNIST Dataset

In [ ]:
```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

In [ ]:
```
import matplotlib.pyplot as plt
image_index = 7777
print(y_train[image_index]) # The label is 8
plt.imshow(x_train[image_index], cmap='Greys')
```

In [ ]:
```
x_train.shape
```

In [ ]:
```
# Reshaping the array to 4-dims so that it can work with the Keras API
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
# Making sure that the values are float so that we can get decimal points after division
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# Normalizing the RGB codes by dividing it to the max RGB value.
x_train /= 255
x_test /= 255
print('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])
```

Implementation of CNN using MNIST Dataset

In [ ]:
```
# Importing the required Keras modules containing model and layers
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))
```

In [ ]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x=x_train,y=y_train, epochs=10)
```

In [ ]:
```
model.evaluate(x_test, y_test)
```

In [ ]:
```
image_index = 4444
img_rows = 28
img_cols = 28
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
pred = model.predict(x_test[image_index].reshape(1, img_rows, img_cols, 1))
print(pred.argmax())
```

Results and Analysis

- The top result corresponds to the GPU training and the bottom result corresponds to the CPU training.
- The weights are updated in each epoch; 10 epochs is optimum for this task as it gave an accuracy of about 98-99%.
- When the training time for each epoch is compared it might not look significant but as we increase the number of epochs or look at the overall training time for 10 epochs the ***GPU training time is nearly 1/3 of the total CPU training time***.
- GPUs are more efficient than CPUs in implementing CNN as GPUs can handle large data sets.

```
In [6]: model.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])
        model.fit(x=x_train,y=y_train, epochs=10)

Epoch 1/10
60000/60000 [==============================] - 9s 156us/step - loss: 0.2124 - accuracy: 0.9371
Epoch 2/10
60000/60000 [==============================] - 6s 106us/step - loss: 0.0858 - accuracy: 0.9733
Epoch 3/10
60000/60000 [==============================] - 6s 107us/step - loss: 0.0587 - accuracy: 0.9815
Epoch 4/10
60000/60000 [==============================] - 6s 104us/step - loss: 0.0462 - accuracy: 0.9845
Epoch 5/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.0361 - accuracy: 0.9885
Epoch 6/10
60000/60000 [==============================] - 6s 102us/step - loss: 0.0302 - accuracy: 0.9896
Epoch 7/10
60000/60000 [==============================] - 6s 105us/step - loss: 0.0233 - accuracy: 0.9919
Epoch 8/10
60000/60000 [==============================] - 6s 105us/step - loss: 0.0229 - accuracy: 0.9922
Epoch 9/10
60000/60000 [==============================] - 6s 105us/step - loss: 0.0202 - accuracy: 0.9935
Epoch 10/10
60000/60000 [==============================] - 7s 111us/step - loss: 0.0172 - accuracy: 0.9943

Out[6]: <keras.callbacks.callbacks.History at 0x26ee9064448>
```

```
In [6]: model.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])
        model.fit(x=x_train,y=y_train, epochs=10)

Epoch 1/10
60000/60000 [==============================] - 16s 261us/step - loss: 0.2094 - accuracy: 0.9369
Epoch 2/10
60000/60000 [==============================] - 16s 267us/step - loss: 0.0864 - accuracy: 0.9733
Epoch 3/10
60000/60000 [==============================] - 17s 279us/step - loss: 0.0602 - accuracy: 0.9811
Epoch 4/10
60000/60000 [==============================] - 17s 278us/step - loss: 0.0460 - accuracy: 0.9851
Epoch 5/10
60000/60000 [==============================] - 17s 277us/step - loss: 0.0360 - accuracy: 0.9883
Epoch 6/10
60000/60000 [==============================] - 17s 291us/step - loss: 0.0314 - accuracy: 0.9894
Epoch 7/10
60000/60000 [==============================] - 17s 284us/step - loss: 0.0264 - accuracy: 0.9913
Epoch 8/10
60000/60000 [==============================] - 18s 295us/step - loss: 0.0223 - accuracy: 0.9926
Epoch 9/10
60000/60000 [==============================] - 17s 283us/step - loss: 0.0207 - accuracy: 0.9930
Epoch 10/10
60000/60000 [==============================] - 17s 280us/step - loss: 0.0190 - accuracy: 0.9935

Out[6]: <keras.callbacks.callbacks.History at 0x28a86bc6448>
```

Results and Analysis

- The GPU architecture plays a key role in efficient implementation of the neural network.
- Some of the dominant features of the GPU vs CPU are listed below:

CPU	GPU
More flexible to support various kind instructions	GPUs are designed to perform in parallel the same kind of computation
CPUs have few complex computational cores	GPUs have more computational units and having a higher bandwidth to retrieve from memory
It is not programmed to do highly parallelized computation	It is highly parallel, highly multithreaded multiprocessor
Interconnection between CPU cores is complex	Interconnection between cores is easier than CPU

Example

- Even if GPU and CPU cores are been equalized, the implementation of neural network on GPU will be much more efficient than CPU.
- GPUs are designed to perform in parallel the same kind of computation.
- Neural Networks are structured in a very uniform manner such that at each layer of the network thousands of identical artificial neurons perform the same computation.
- The GPU follows Single Instruction Multiple Data(SIMD) principle.
- Therefore the structure of a neural network fits quite well with the kinds of computation that a GPU can efficiently perform.

Inference

- Convolutional Neural Networks is not only a breakthrough in the fields of Image Processing and Video Processing, but it has also made a huge impact in the development of AI and Computer Vision.

- This is all made possible since GPU shader cores are unified and programmable (CUDA).

- GPU architecture has flexibility along with modern software libraries like TensorFlow. It allows data scientist, engineers, mathematicians, and other researchers to implement such computationally intensive task with reduced time and space complexity.

- It had opened the door for new possibilities and technology.

References

- https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
- https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- https://ieeexplore.ieee.org/document/4700015
- https://www.cs.cmu.edu/afs/cs/academic/class/15462-f11/www/lec_slides/lec19.pdf
- http://meseec.ce.rit.edu/551-projects/spring2015/3-1.pdf
- https://www.tensorflow.org/guide/tensor
- https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/
- https://datascience.stackexchange.com/questions/19220/choosing-between-cpu-and-gpu-for-training-a-neural-network

Thank you!

Questions are welcomed!!