Graphical Processing Units For Deep Learning

The key that gave rise to the era of Artificial Intelligence.

Saransh Agrawal
-170921060

Machine Learning

- Conventional Programming

- Machine Learning Approach

Steps in Machine Learning

1. Collection and Division of Dataset
2. Exploratory Data Analysis
3. Pre-Processing of Dataset
4. Model formation and Training
5. Model Evaluation

Training a Model

1. Forward Propagation
2. Calculate Cost
3. Backward Propagation
4. Update Parameters

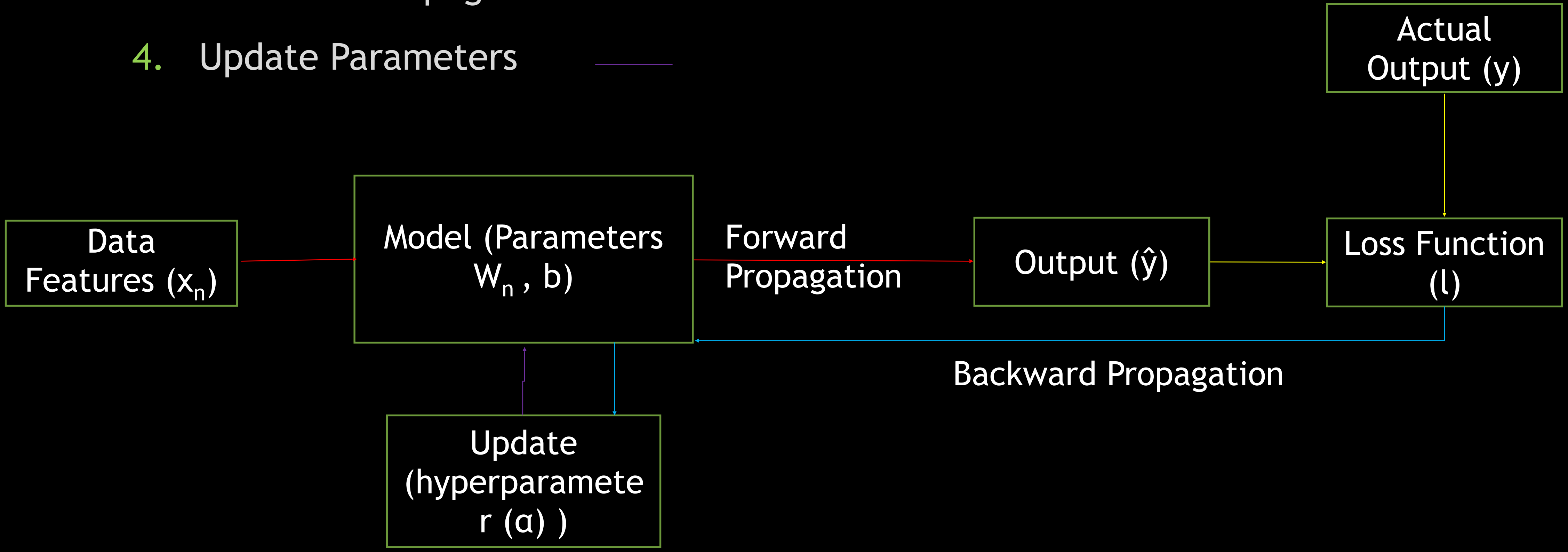


Fig 1: Block diagram of training a

Logistic Regression

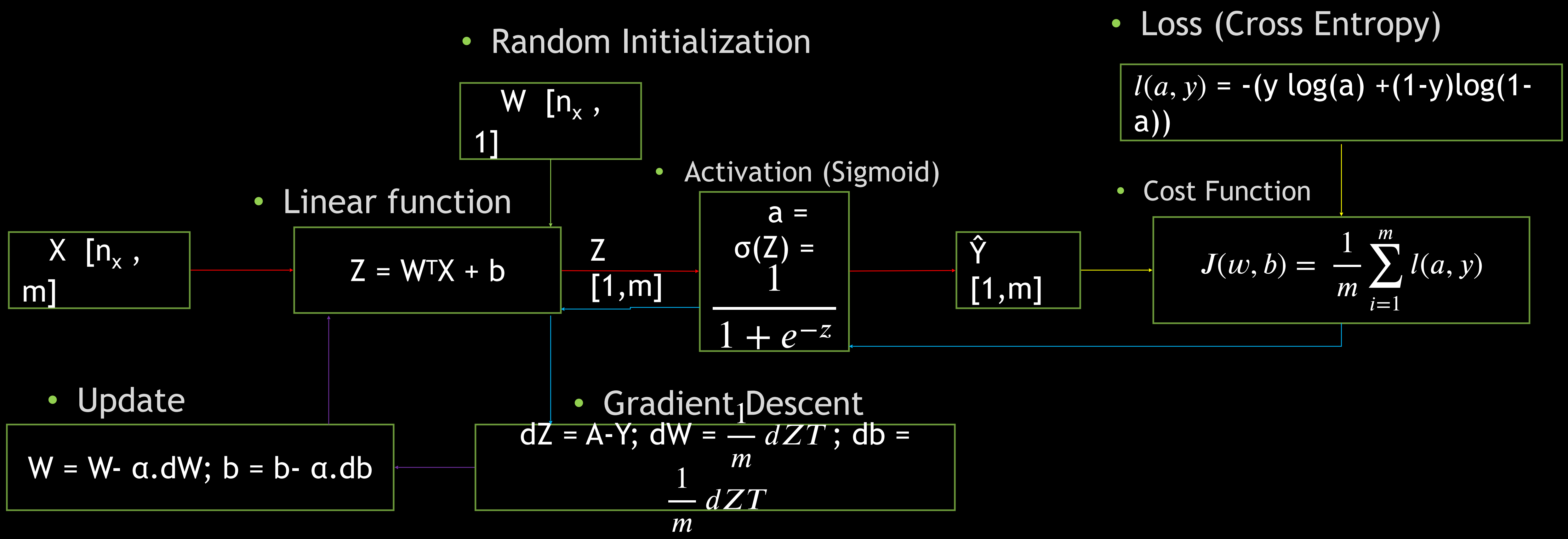


Fig 2: Block diagram including equations for training a ML model

Deep Neural Network

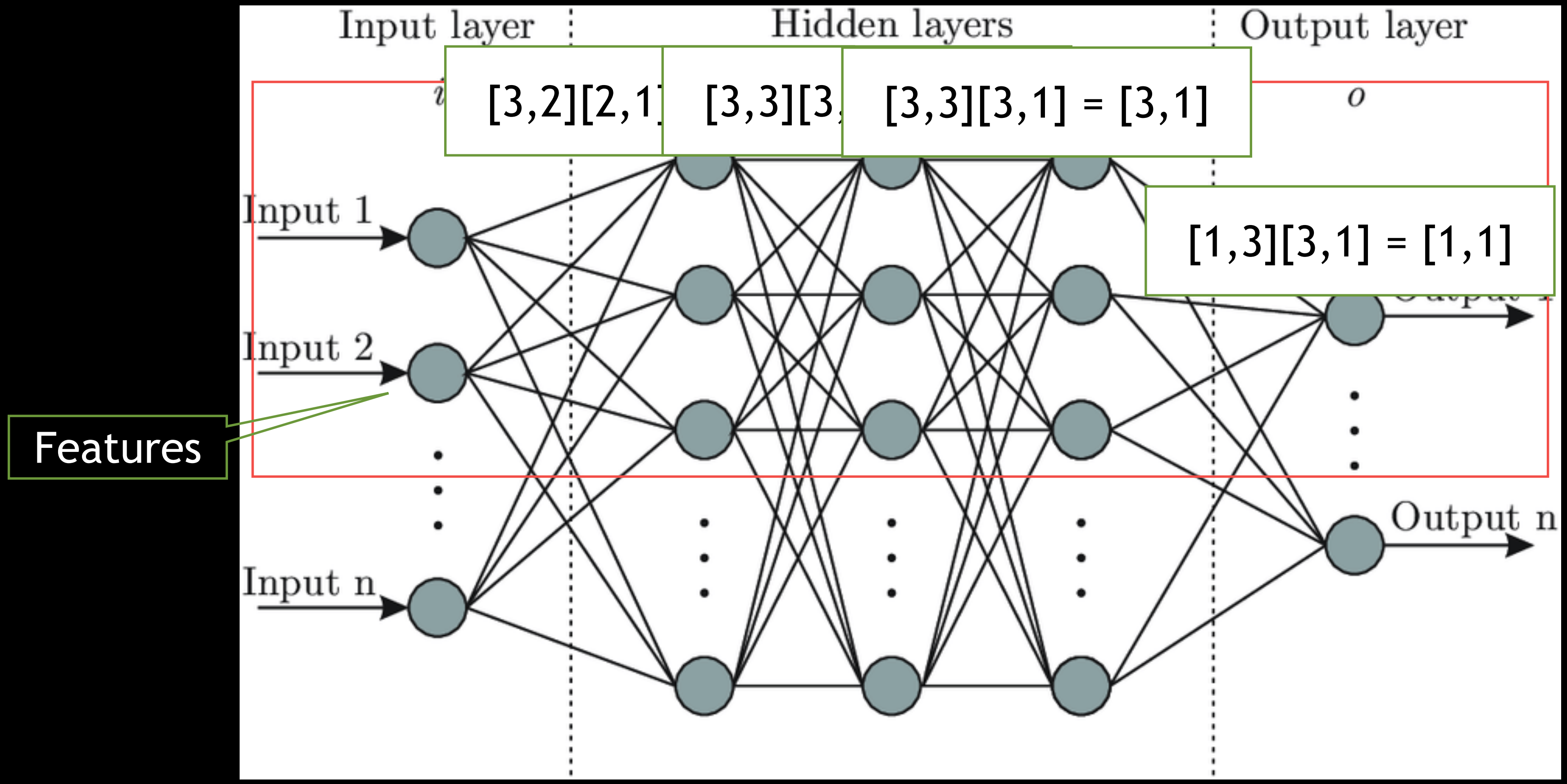


Fig 3: Architecture of Deep Neural Network

- W [number of neurons in current layer, number of neurons in previous layer]

- https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051

Convolution 2D

Input

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

[9, 9]

*

Filter

1	0	-1
1	0	-1
1	0	-1

[3, 3]

=

Output

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

[4, 4]

Convolutional Neural Network

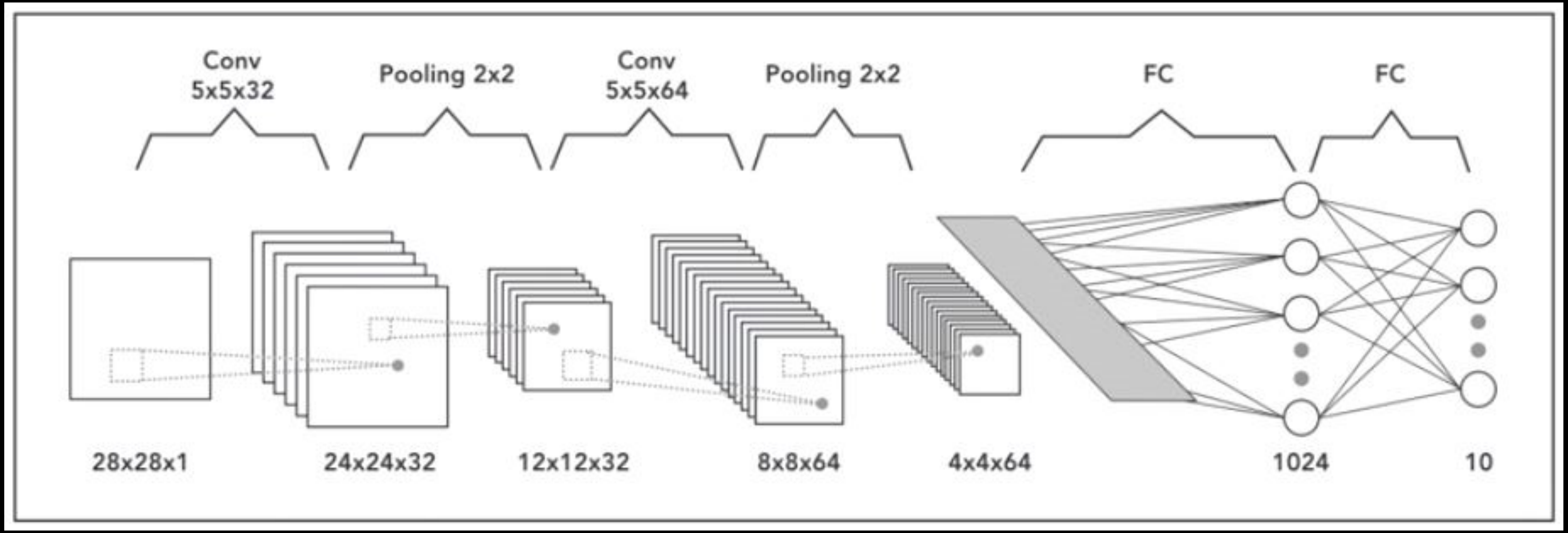


Fig 4: Architecture of Convolutional network

- https://engmrk.com/convolutional-neural-network-3/

CPU versus GPU

CPU

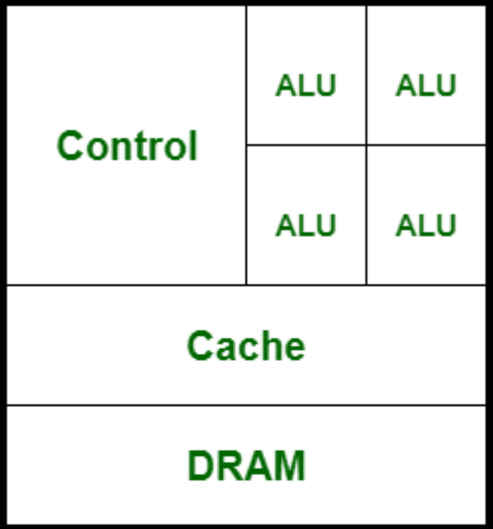


Fig 5: Architecture of CPU

- Strong individual cores (more versatile ALU)
- Less number of cores

GPU

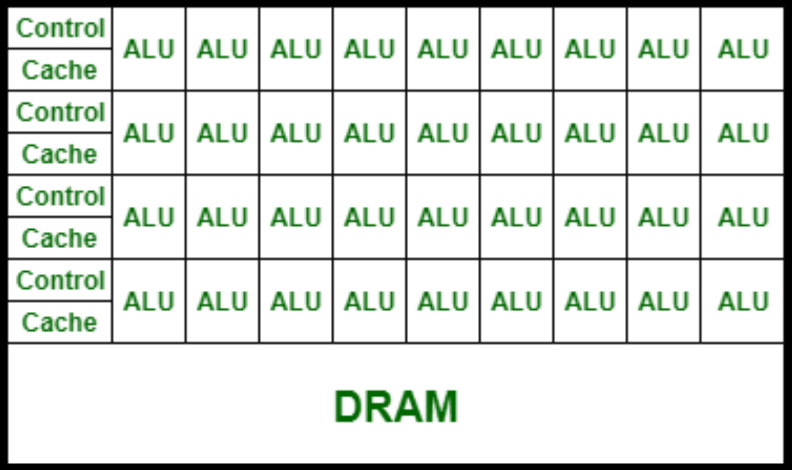


Fig 6: Architecture of GPU

- Weak individual cores
- More number of cores

https://www.geeksforgeeks.org/difference-between-cpu-and-gpu/

Why does having more cores make a difference?

- Given an input of size [9,9], if we want to divide it by number 255, say one ALU is capable of completing one division.
- Here, a total of 81 computations need to be done

- For CPU:

  with 4 available ALU, it will take

  = 81/4

  = 21 total runs.

- For GPU:

  with 36 available ALU, it will take

  = 81/36

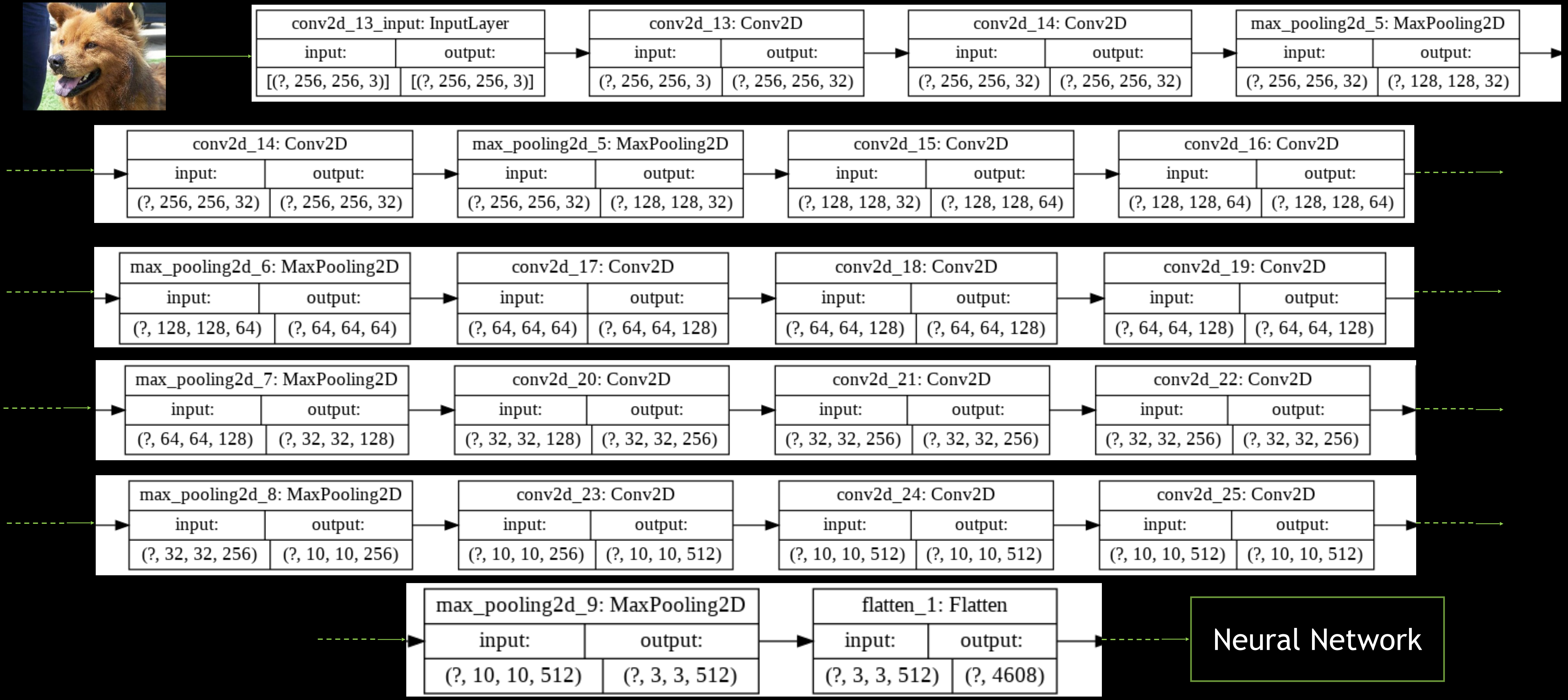
  = 3 total runs.

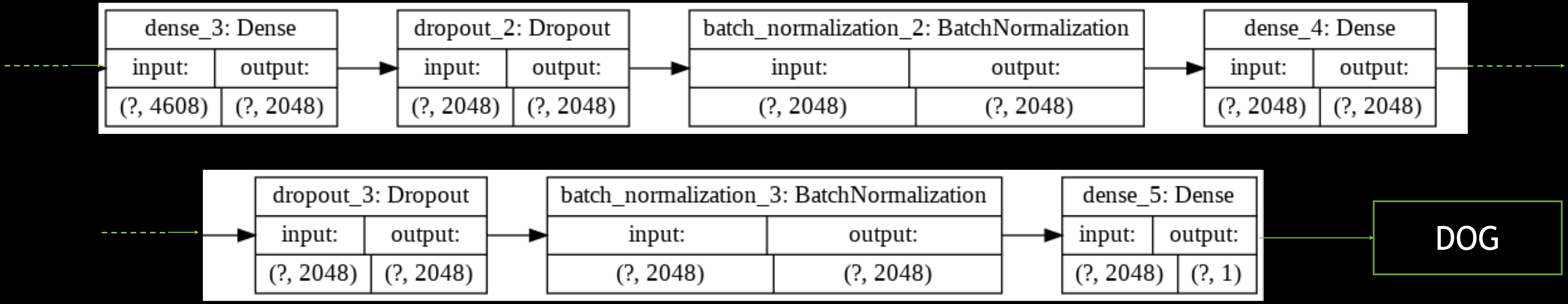
* Just for visualization, not actual figures.

CPU vs GPU vs High end GPU

Intel core i5-8250U	Nvidia GeForce GT-1030	Nvidia T4 Tensor core
• Cores: 4 • Threads: 8	• CUDA cores: 384	• CUDA cores: 2560 • Turing Tensor cores: 320
• Base Frequency: 1.60 GHz • MAX Turbo Freq: 3.4 GHz	• Base clock frequency: 1228 MHz • Boost: 1468 MHz	• Base clock frequency: 585 MHz • Boost: 1590 MHz
• RAM: 8GB (MAX 32GB) • MAX Bandwidth: 37.5 Gb/s	• RAM: 2GB DDR5 • MAX Bandwidth: 48GB/s	• RAM: 16GB GDDR6 • MAX Bandwidth: 300Gb/s
	• CUDA compute compatibility: 6.1	• CUDA compute compatibility: 7.5
https://ark.intel.com/content/www/us/en/ark/products/124967/intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html	https://www.nvidia.com/en-us/geforce/graphics-cards/gt-1030/specifications/	https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-product-brief.pdf

Visualization using a CAT vs DOG Model





Summary

- 15 Conv2D layers
- 6 Max pooling 2D layers
- 3 Dense layers
- Biggest matrix size = [m, 256, 256, 32]

```
Total params: 21,463,713
Trainable params: 21,455,521
Non-trainable params: 8,192
```

Performance on CPU

Training Conditions:

- Pre-Processing step: Normalization, divide each pixel by 255.
- Batch size: 16
- 20,000 training images, 5000 test images

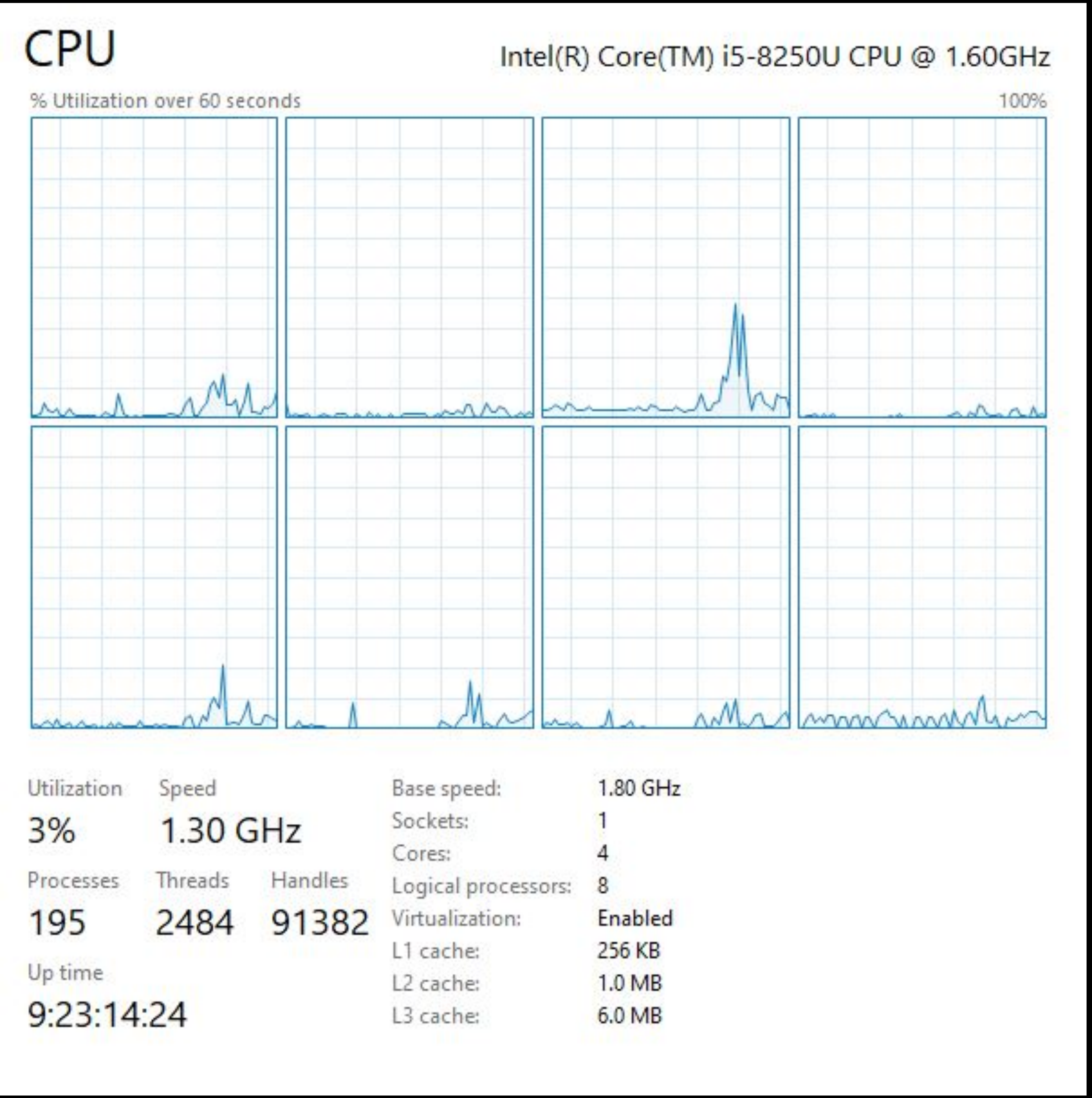
```
In [*]: with tf.device('/CPU:0'):
        #callbacks = myCallback()
            checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("D:\python_practise_deeplearning\Cat_V_dog\cat_V_dog_dropout.h5", save_bes
            early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience = 10, restore_best_weights = True)

            history = model.fit(train_generator,
                                epochs=100,
                                verbose=1,
                                callbacks = [checkpoint_cb, early_stopping_cb],
                                validation_data=test_generator, workers = -1)

        Epoch 1/100
          42/1250 [>.............................] - ETA: 1:58:20 - loss: 0.8928 - acc: 0.5298
```

Fig 7: CPU Training snapshot

Base CPU utilization



CPU utilization during training

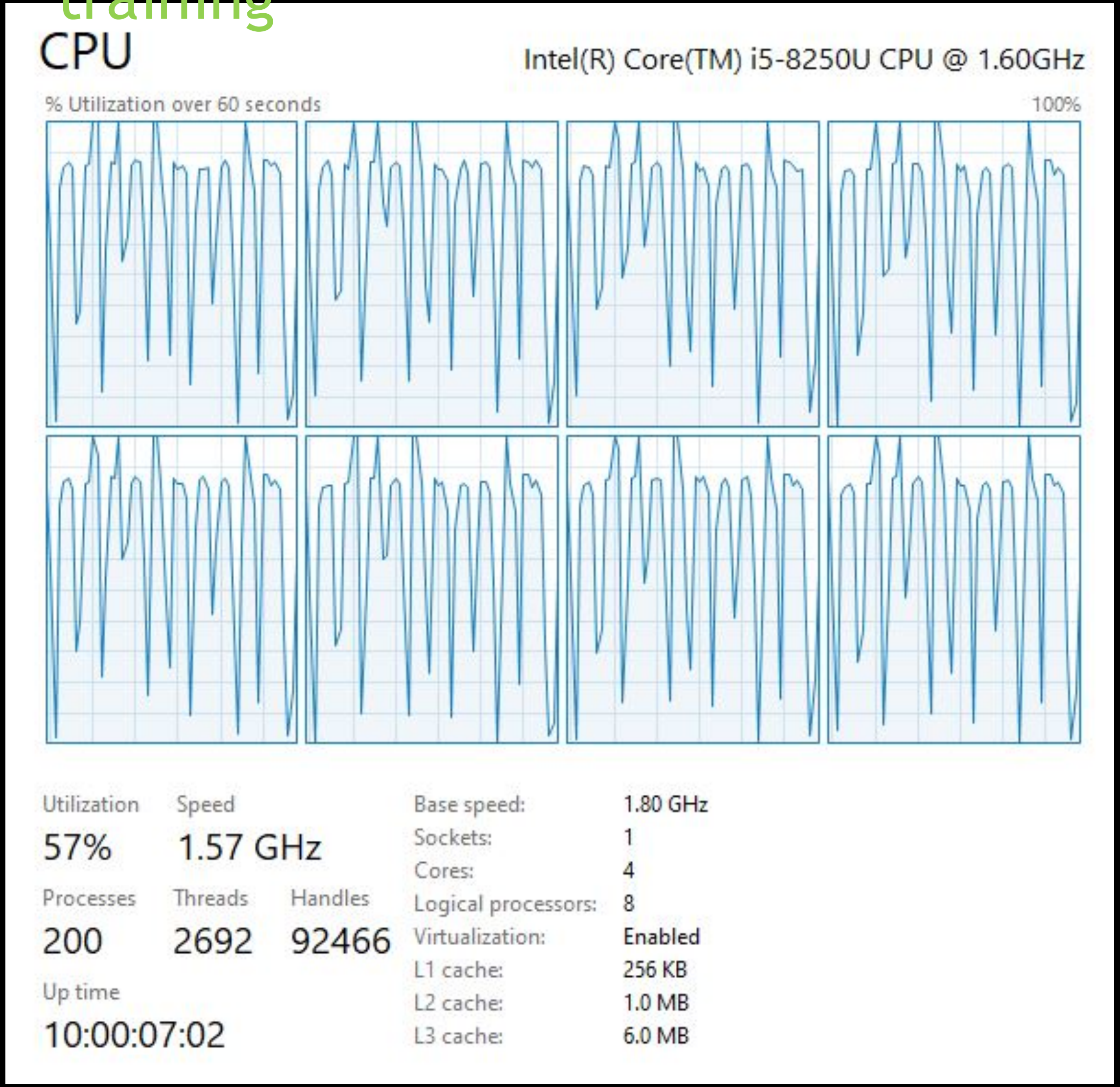


Fig 8: CPU utilization graph (% utilization over 60sec)

Performance on NVIDIA GeForce MX150

Training Conditions:

- Pre-Processing step: Normalization, divide each pixel by 255.
- Batch size: 16
- 20,000 training images, 5000 test images

```
1250/1250 [==============================] - 678s 542ms/step - loss: 0.7842 - acc: 0.5810 - val_loss: 0.6393 - val_acc: 0.6388
Epoch 2/10
1250/1250 [==============================] - 665s 532ms/step - loss: 0.6121 - acc: 0.6958 - val_loss: 0.6204 - val_acc: 0.7032
Epoch 3/10
 171/1250 [===>..........................] - ETA: 8:54 - loss: 0.5085 - acc: 0.7719
```

11 min/epoch

Fig 9: GPU MX150 Training snapshot

<u>NOTE</u>: MX150 and GT-1030 are both based on same chipset (GP-108), the latter is desktop version, while MX-150 is modified for laptops.

GPU utilization during training

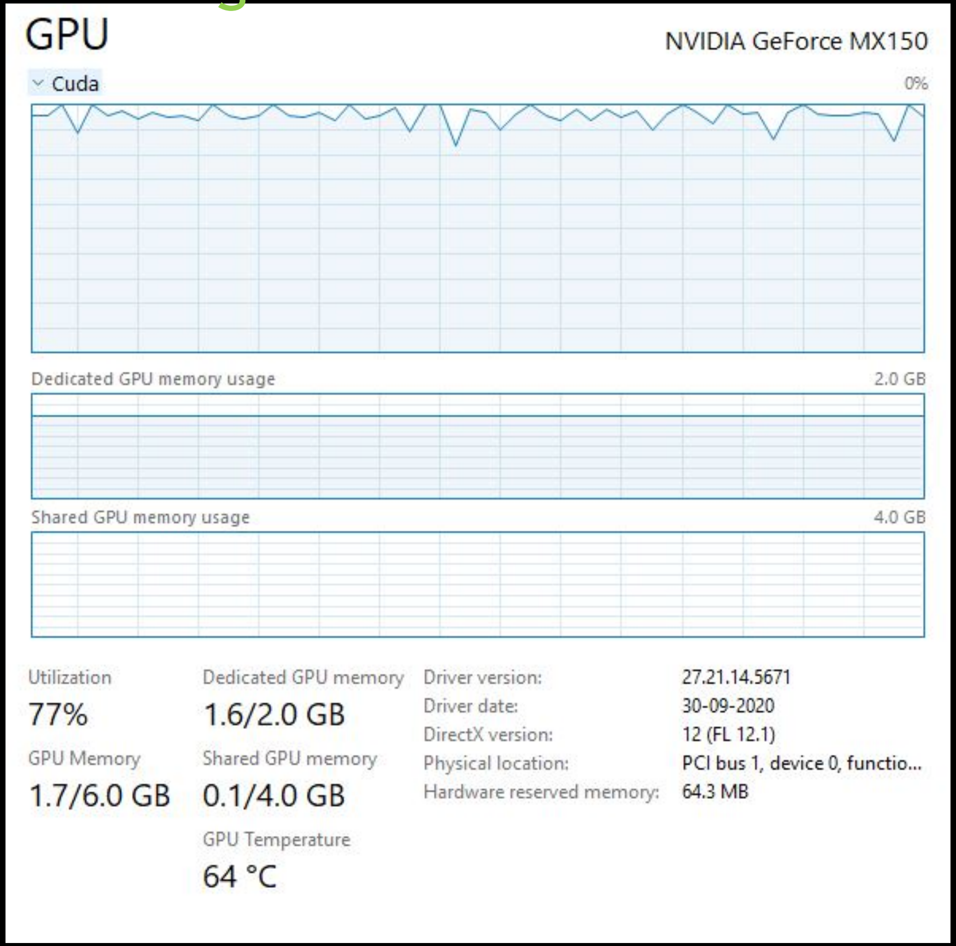


Fig 10: GPU MX150 utilization graph (% utilization over 60sec)

Increasing Batch size to 32

```
  (1) Resource exhausted:  OOM when allocating tensor with shape[32,128,64,64] and type float on /job:localhost/replica:0/tas
k:0/device:GPU:0 by allocator GPU_0_bfc
         [[node sequential/conv2d_5/Relu (defined at <ipython-input-13-062747f25914>:10) ]]
Hint: If you want to see a list of allocated tensors when OOM happens, add report_tensor_allocations_upon_oom to RunOptions f
or current allocation info.

0 successful operations.
0 derived errors ignored. [Op:__inference_train_function_2507]

Function call stack:
train_function -> train_function
```

Fig 11: OOM error

<u>OOM error</u>: Out of memory, since tensor of shape [32,128,64,64] could not fit in RAM.

Performance on NVIDIA T4

Training Conditions:

- Pre-Processing step: Normalization, divide each pixel by 255.
- Batch size: 128
- 20,000 training images, 5000 test images

```
Epoch 2/20
157/157 [==============================] - ETA: 0s - loss: 0.6215 - acc: 0.6567INFO:tensorflow:Assets written to: /content/driv
157/157 [==============================] - 379s 2s/step - loss: 0.6215 - acc: 0.6567 - val_loss: 0.6812 - val_acc: 0.5118
Epoch 3/20
157/157 [==============================] - ETA: 0s - loss: 0.5720 - acc: 0.7117INFO:tensorflow:Assets written to: /content/driv
157/157 [==============================] - 389s 2s/step - loss: 0.5720 - acc: 0.7117 - val_loss: 0.6617 - val_acc: 0.5484
```

Fig 12: GPU T4 Training snapshot

6.3 min/epoch

Stochastic VS Batch gradient descent

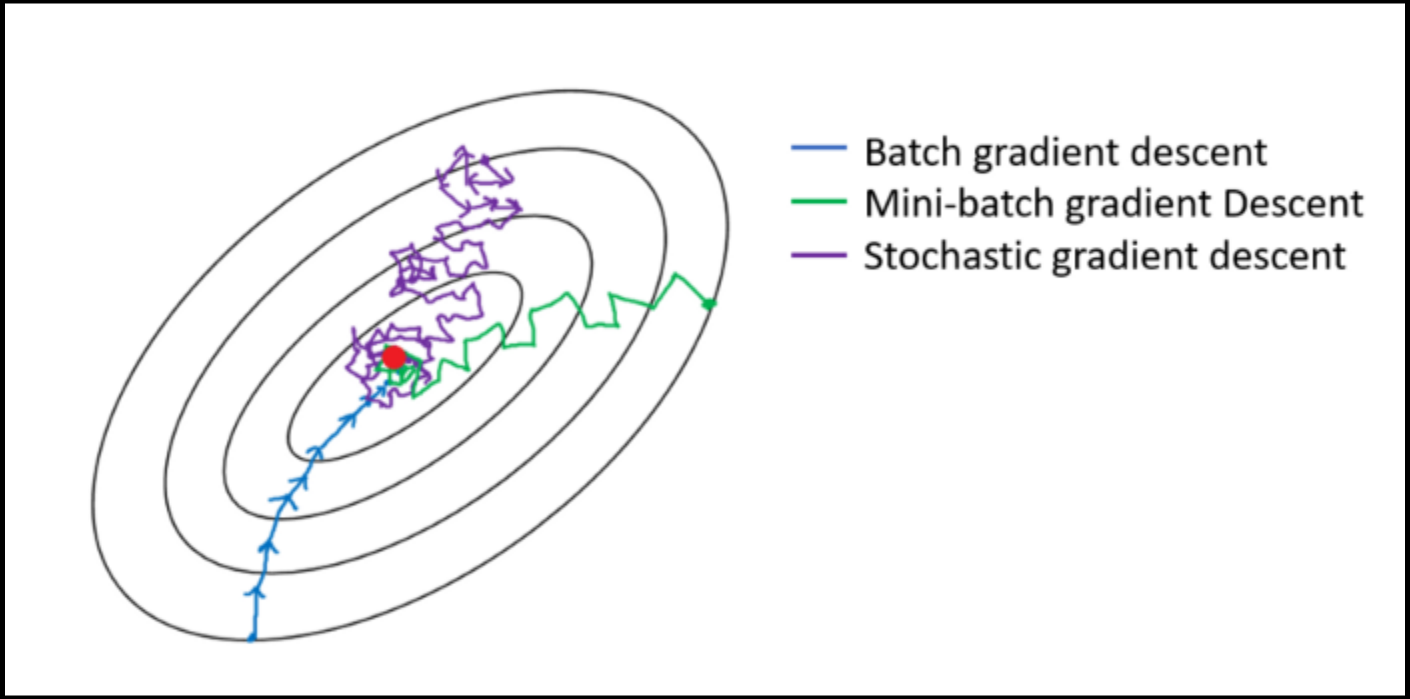


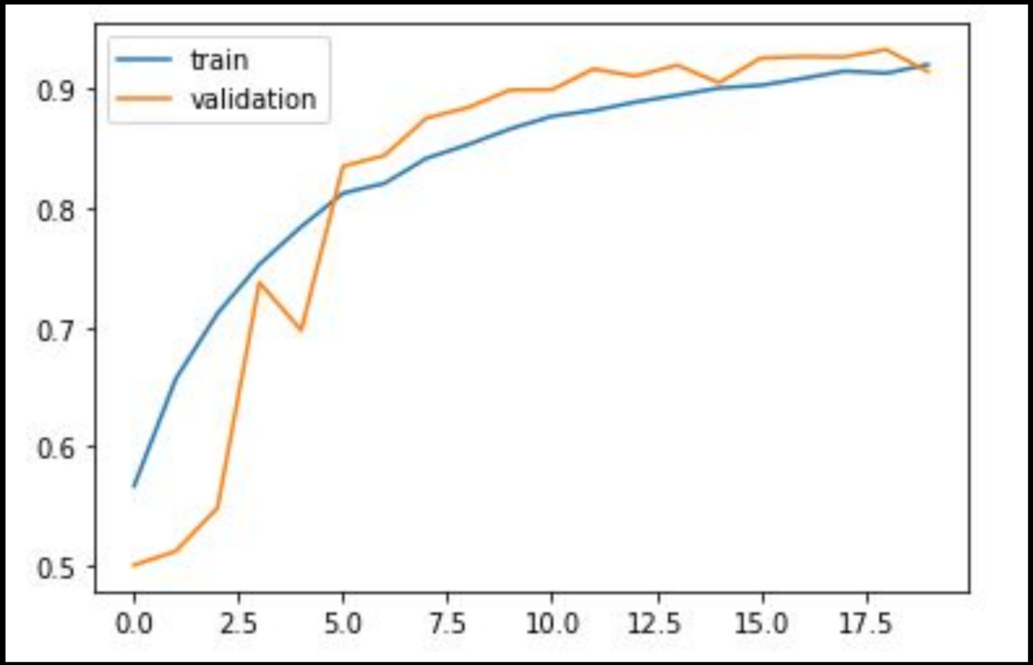
Fig 13: Contour plot for decreasing cost by gradient descent

- A large batch size is better.
- High speed, large capacity RAM accessible to GPU.

https://towardsdatascience.com/an-introduction-to-gradient-descent-c9cca5739307

Accuracy

Accuracy

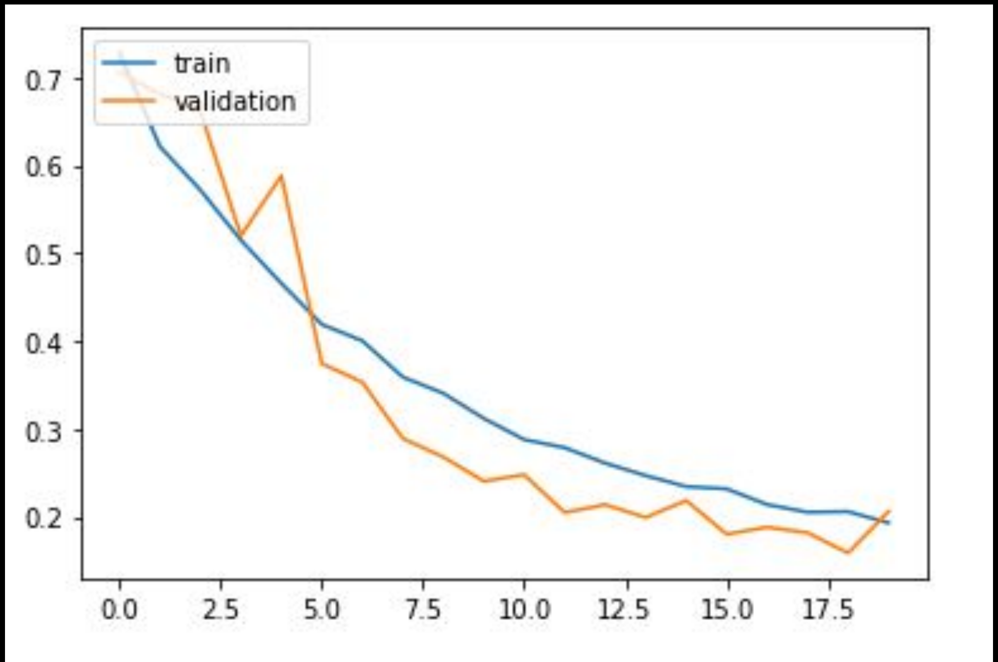


Epoc
h

Fig 14(a): Accuracy vs epoch

Loss

Loss



Epoc
h

Fig 14(b): Loss vs epoch

Bibliography

1. Deep Learning: Ian Goodfellow, Yoshua Bengio and Aaron Courville
2. Performance on CPU vs GPU for deep learning workloads: Amr Kayid and Yasmeen Khaled: https://www.researchgate.net/publication/224626495_Using_GPUs_for_machine_learning_algorithms
3. Performance and Scalability of GPU-based Convolutional Neural Networks: Daniel Strigl, Klaus Kofler and Stefan Podlipnig, http://www.dps.uibk.ac.at/~klaus/Klaus_Kofler_-_Institute_for_Computer_Science_files/GPUCNN.pdf
4. Tensorflow: https://www.tensorflow.org
5. Original Code: https://github.com/saranshagarwal202/Cats_versus_Dogs