COMP5511 Assignment

Group J

Topic: Machine Learning Packages Comparison - PyTorch and TensorFlow

21000018G Li Yat Long (Leader), yat-long-vincent.li@connect.polyu.hk
20090346G Tang Siwei, siwei.tang@connect.polyu.hk
21057118G Wei Yu, yunick.wei@connect.polyu.hk
21052076G Huang Zuo, zuo-chris.huang@connect.polyu.hk
21107928G Lam Tsz Pok, tsz-pok.lam@connect.polyu.hk
21066157G Zhong Weiqi, 21066157g@connect.polyu.hk
21015103G Tsang Hang Ching, hangching.tsang@connect.polyu.h

1. Abstract

The scope of machine learning and data science has become even more popular in the last 20 years, which requires relative tools to finish the tasks. A variety of libraries and frameworks of neural networks and deep learning were created by various enterprises or developers. One of the most popular programming language Python is often used with the machine learning packages for creating artificial intelligence applications and machine learning model training, considering its splendid performance in the area of math calculating and module building,

Among various machine learning libraries, PyTroch and TensorFlow were selected in this report for comparison in execution time and accuracy, through running an image classification Python test program with different kernel size, epochs and batch size. In the end of this report, recommendations will be suggested for choosing PyTorch or TensorFlow on building artificial intelligence software for different project scope or environment.

2. Introduction

2.1 PyTorch

The torch based library was originally developed by Facebook's AI Research Lab (FAIR) in C with Lua as the interface, which had been rewritten in 60% C++ & CUDA and 32% Python due to the growth of Python in the machine learning area [1]. Currently, Pytorch is open source with support on Linux, macOS and Windows on Python, C++ or Java language[1]. In addition, GPU acceleration is supported if GPU supports Nvidia CUDA [2].

The advantages of PyTorch is that users can define the graph on-the-go which is easier to rectify the program and beginners can benefit with its less steep learning curve [3]. PyTorch enables the use of common debugging tools such that beginners can identify errors easily [4]. Another advantage of PyTorch is that it has a data parallelism feature which allows complex problems to be solved in a lower time cost [5]. Although its beginner friendly design, finding tutorials may be more difficult than TensorFlow and less reliable onproduction models as it has a small developer community [3].

2.2 TensorFlow

The TensorFlow deep learning library, originally developed by Google Brain Team, was released as free open source software under Apache License 2.0 in 2015 [1]. Similar to PyTorch, TensorFlow is written in 60% C++ with CUDA and 30% Python [1]. Currently, TensorFlow can be installed on Ubuntu which is based on Linux, Windows and macOS with Python, Java, C and Go versions available [6]. In addition, TensorFlow utilizes both Nvidia CUDA and Apple Metal for GPU acceleration if required hardware and software are fulfilled [7][8].

One of the biggest advantages of TensorFlow is its huge community behind, which offers a variety of resources including program code and tutorials for production or even beginners [3]. The reason it has developed a large user base is that it is an open source platform and that it is compatible with many common programming languages. Together with its ability to visualize machine learning models in browsers and offers better performance on production models and scalability, TensorFlow is loved by many users or even enterprises [3]. However, TensorFlow is time consuming for rapid prototypes and required to define the entire computation graph of the model in the beginning [3]. Another disadvantage is that it has a much steeper learning curve. The use of TensorFlow requires a deep understanding of the fundamental mathematical concepts including linear algebra and advanced calculus. In addition, TensorFlow adapts a unique structure such that it requires a more advanced debugging tool [9].

2.3 Comparison

Both PyTorch and TensorFlow are based on dataflow concepts in which the program is organised as computational blocks that data will transfer between blocks, benefiting the implementation of multi-core CPU parallel calculations and distributed cluster systems [1]. The full comparison table is listed below:

	PyTorch	TensorFlow
Open Source	Yes	Yes
OS Support	Linux macOS	Ubuntu macOS

	Windows	Windows
Program language support	Python C++ Java	Python C Java Go
CPU Training	Yes	Yes
GPU acceleration support	Yes (Nvidia CUDA)	Yes (Nvidia CUDA, Apple Metal)
Computation Graph	Dynamic	Static
Learning Curve	Lower	Higher
Tutorial	Less	Offered by a large community

Fig. 1. Feature comparison between PyTorch and TensorFlow

2.4 Popularity

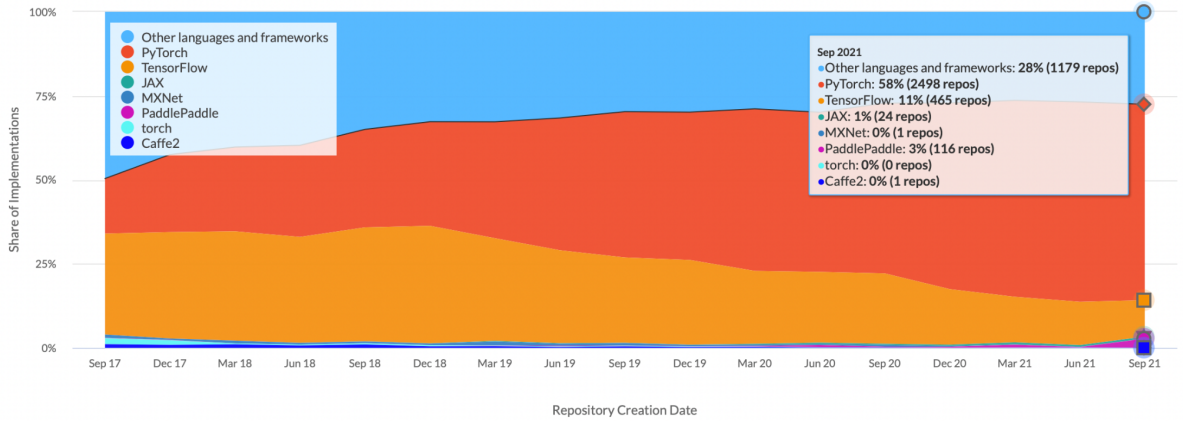


Fig. 2. Latest popularity of paper implementation grouped with various frameworks by Papers With Code [10]

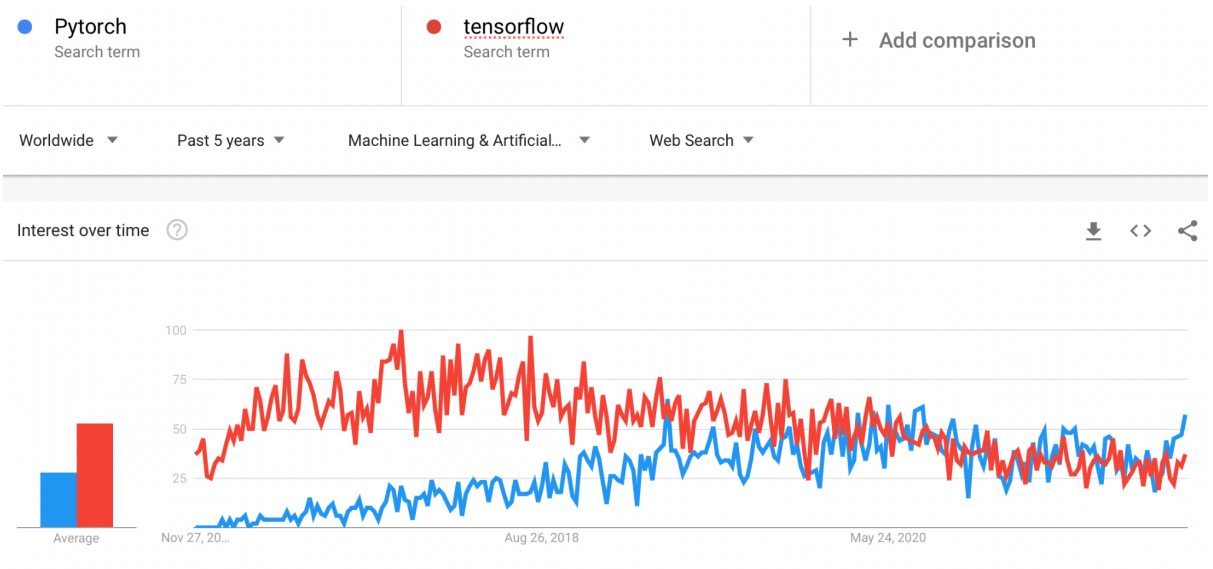


Fig. 3. Latest trend of web search among Pytorch and TensorFlow by Google [11]

According to Fig. 2, in the last 4 years both Pytorch and TensorFlow were the two most popular frameworks mentioned in research and papers, with 58% and 11% in September 2021 respectively. Meanwhile, Both Fig. 2 and Fig. 3 show the interest among Pytorch increased significantly in the last 4 years in which pytorch could be found more frequently than TensorFlow on papers and have nearly the same search quantities as TensorFlow.
In terms of market share, TensorFlow was still the most popular with 40.23% market share while Pytorch was in 4th place with 15.76% market share [12].

Hence, TensorFlow was chosen due to its leadership in the industry and PyTorch was chosen due to its significant gains in popularity in the last 4 years.

3. Testing Implementation

Two image classification Python programs using PyTorch or TensorFlow were downloaded from their official website respectively with modification on the variables of the training models to ensure both programs were performing the exact same training [13] [14]. The tests were executed on multiple Windows and macOS based machines, generating well-rounded model accuracy data and execution time for analysing the difference of performance between PyTorch and TensorFlow.

3.1 CIFAR-10 test

According to Krizhevsky [15], the dataset made by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, includes 6000 32x32 coloured photos in each class, with 10 classes in total. The classes consist of both vehicles and animals, including airplane, automobile, ship, truck, bird, cat, deer, dog, frog and horse. The ratio of training data to test data is 5:1.

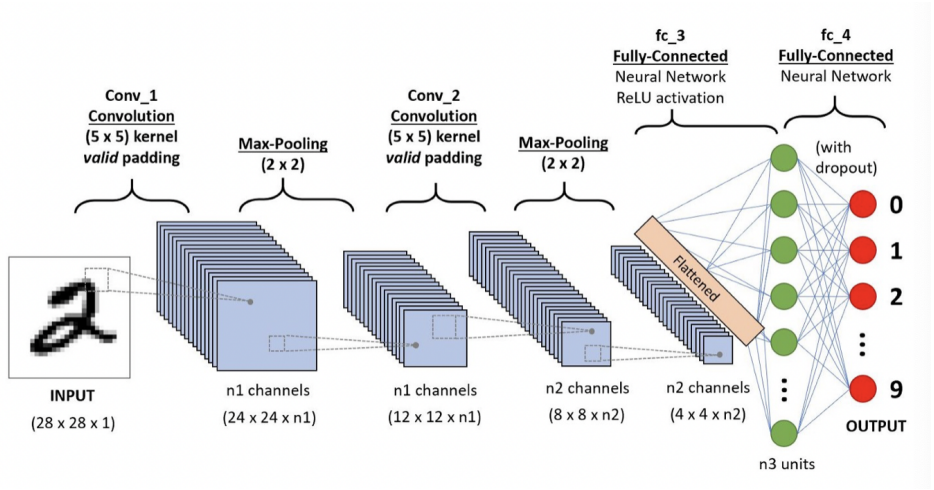


Fig. 4. How CNN classify an image by S. Saha [16]

The above data are imported to the python program for building an image classifier based on deep learning algorithm - Convolutional Neural Network (CNN). Fig. 4 shows the 4 main layers in CNN, including Convolutional, ReLU Layer, Pooling and Fully Connected Layer. JavaTpoint [17] explained that CNN divides a big image into smaller parts called filters for finding rough feature matches in the same position of two pictures and creating a map of similarity during convolution layer phase. Then the negative unit on the map created will be removed and replaced with zero by the Rectified Linear unit (ReLU) Layer. Finally, the image stack will be shrunk during pooling and a fully-connected phase will be applied, connecting all the previous layers by forming a single vector list for classification. Fig. 5 demonstrates that the image will be classified by identifying the high values pattern in the single vector list. In our test, Conv2D is chosen as the convolution layer and MaxPool2D is selected as the pooling layer, which replaces the specific area of pixel with the single maximum value of that area to shrink the image effectively. Similar to Fig. 4, two pooling layers are introduced between 3 convolution layers in our program to reduce size of image for best pattern search in CNN.

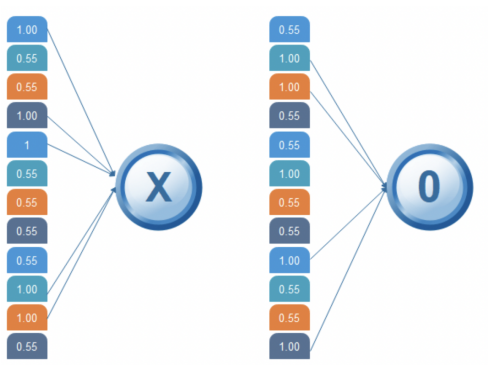


Fig. 5. Single vector list of fully-connected phase by JavaTpoint [17]

3.2 Epochs and Batch Size

Epochs	Batch size
10	4
10	8
10	16
10	32
20	4
20	8
20	16
20	32

Fig. 6. Test cases combination with different epochs and batch size values.

The python programs were tested in a variety of epochs and batch size combinations. According to Sharma [18], epoch defines the number of whole dataset processes forward and backward through the neural network and batch size defines the number of training samples in a single batch. For example, when epochs = 10 and batch size = 4, it means that the program will pass the data through the neural network 10 times and the data sample will split to 4 per batch which is 1250 images in this example.

3.3 Kernel Size and Trainable Parameters

Thevenot [19] explained that a 3x3 kernel (convolution matrix) which is a filter for passing the input image, will move over the whole image to capture in the image all squares of the same size. Rosebrock [20] points out that it is common to use a 5x5 or 7x7 sized kernel for learning large features in image size larger than 128x128 and 1x1 or 3x3 for image size smaller than 128x128. In our test, two common kernel sizes including 3x3 and 5x5 were applied to both PyTorch and TensorFlow test programs.

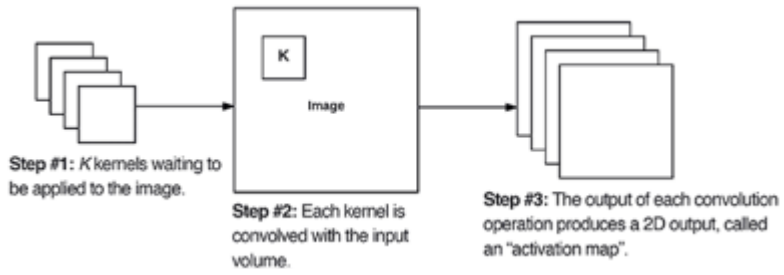


Fig. 7. How Filter works in Conv2D by Rosebrock [19]

In addition, in our testing program with kernel size 3x3, 32 filters were applied to the first Conv2D layers and 64 filters were applied to the second and third Conv2D layers with pooling applied between each Conv2D layer to reduce the spatial dimensions. For the 5x5 program, 6 filters were applied to the first Conv2D layers and 16 filters were applied to the second Conv2D layers with pooling applied after every Conv2D layer to reduce the spatial dimensions.

On the other hand, the trainable parameter is the count of the trainable filters of the specific layer according to Vasudev [21], which is calculated by ((shape of width of filter*shape of height filter*number of filters in the previous layer+(filter bias))*number of filters). Therefore, for the number of trainable parameter in the second Conv2D layer of the 3x3 program:

((3*3*32)+1)*64 = 18496

Total 18496 trainable parameters can be obtained. This also proves the trainable parameter of our 5x5 program is much smaller as much less filters were introduced compared to the 3x3 program.

```
       Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 30, 30]             896
         MaxPool2d-2          [-1, 32, 15, 15]               0
            Conv2d-3          [-1, 64, 13, 13]          18,496
         MaxPool2d-4            [-1, 64, 6, 6]               0
            Conv2d-5            [-1, 64, 4, 4]          36,928
            Linear-6                 [-1, 64]          65,600
            Linear-7                 [-1, 10]             650
================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

Fig. 8. Model of PyTorch program with 3x3 kernel size

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

Fig. 9. Model of TensorFlow program with 3x3 kernel size

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
         MaxPool2d-2            [-1, 6, 14, 14]               0
            Conv2d-3           [-1, 16, 10, 10]           2,416
         MaxPool2d-4             [-1, 16, 5, 5]               0
            Linear-5                  [-1, 120]          48,120
            Linear-6                   [-1, 84]          10,164
            Linear-7                   [-1, 10]             850
================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
```

Fig. 10. Model of PyTorch program with 5x5 kernel size

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         456
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
```

Fig. 11. Model of TensorFlow program with 5x5 kernel size

3.4 Testing execution

Fig. 12. The Cifar10 files for testing

All four python programs would check if the Cifar10 files are downloaded once the execution button is clicked. If the files were missing, the programs would download all the files to the test computer automatically and perform model training once the download is finished, which did not count as a valid test case due to the uncontrollable additional download time.

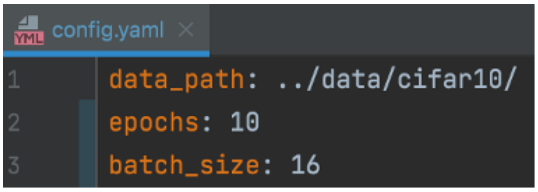


Fig. 13. Configuration file for the test programs

A separate configuration file was implemented to easily modify the variables of epochs and batch size, which are the main focus of our testing, reducing the chance of typographical-error on the existing programs.

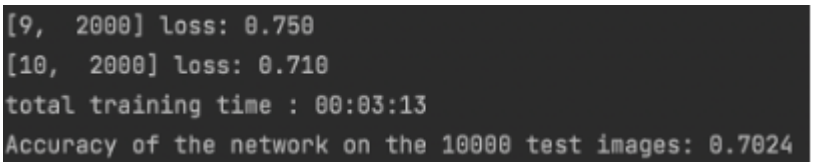


Fig. 14. Output during PyTorch program execution

Fig. 15. Output during TensorFlow program execution

All programs were tested based on Fig. 6, and each case was executed twice to ensure similar results were obtained which were based on the value of total execution time of the python program from start to finish and accuracy of the training model from the output of PyTorch and TensorFlow program shown in Fig. 14 and Fig. 15. In addition, as not all machines / platforms support GPU processing for PyTorch and TensorFlow, all python files only activate CPU for processing to ensure the fairness between the test results from different environments. The specification of the environments are listed below:

Testing environment 1:
Operating system: macOS Monterey (version 12.0.1)
CPU: M1 Pro (8 cores)
RAM: 16GB
Python version: 3.9.7
PyTorch version: 1.10.0
TensorFlow version: Tensorflow-macos 2.6.0

Testing environment 2:
Operating system version: Windows 10 (version 21H1)
CPU: Ryzen 2600
RAM: 16GB
Python version: 3.8
Pytorch version: 1.10.0
TensorFlow version: 2.6.1

Testing environment 3:
Operating system: macOS Big Sur (version 11.6.1)
CPU: 3.1GHz Dual-Core Intel Core i5 (I5-7267U)
RAM: 16GB
Python version: 3.9.7
PyTorch version: 1.10.0
TensorFlow version: 2.6.0

Testing environment 4:

Operating system: macOS Monterey (version 12.0.1)

CPU: 2.6 GHz Quad-core Intel Core i7 (I7-6700HQ)

RAM: 16GB

Python version: 3.9

PyTorch version: 1.10.0

TensorFlow version: 2.6.0

4. Test Results

Testing environment 1:

item	epochs	Batch size	pytorch test 1 time	pytorch test 1 accuracy	pytorch test 2 time	pytorch test 2 accuracy	TF test 1 time	TF test 1 accuracy	TF test 2 time	TF test 2 accuracy
1	10	4	00:04:11	0.7165	00:04:17	0.7154	00:08:49	0.6733	00:09:25	0.6941
2	10	8	00:03:23	0.7358	00:03:24	0.7153	00:05:57	0.6974	00:05:28	0.6966
3	10	16	00:03:04	0.7066	00:03:06	0.7132	00:04:13	0.6940	00:04:09	0.6908
4	10	32	00:02:49	0.6705	00:02:49	0.6585	00:02:44	0.6521	00:02:38	0.6391
5	20	4	00:08:22	0.6866	00:08:23	0.6955	00:19:29	0.6503	00:20:03	0.6746
6	20	8	00:06:47	0.703	00:06:45	0.7107	00:12:22	0.7014	00:11:55	0.6842
7	20	16	00:06:05	0.728	00:06:09	0.7256	00:08:52	0.7010	00:08:18	0.7032
8	20	32	00:05:41	0.7063	00:05:37	0.7292	00:05:38	0.6920	00:05:46	0.6926

Fig. 16. Results of testing environment 1 (Kernel size 3x3)

item	epochs	Batch size	pytorch test 1 time	pytorch test 1 accuracy	pytorch test 2 time	pytorch test 2 accuracy	TF test 1 time	TF test 1 accuracy	TF test 2 time	TF test 2 accuracy
1	10	4	00:02:28	0.623	00:02:28	0.6099	00:03:51	0.5538	00:03:49	0.5883
2	10	8	00:01:51	0.6515	00:01:50	0.6387	00:02:25	0.5790	00:02:25	0.6062
3	10	16	00:01:30	0.6237	00:01:31	0.6321	00:01:44	0.6133	00:01:44	0.6159
4	10	32	00:01:20	0.6115	00:01:20	0.6189	00:01:16	0.5387	00:01:15	0.5524
5	20	4	00:04:58	0.6172	00:04:53	0.6087	00:07:44	0.6159	00:07:54	0.5703
6	20	8	00:03:40	0.6376	00:03:40	0.6412	00:04:51	0.5905	00:04:53	0.5841
7	20	16	00:03:02	0.643	00:03:01	0.6633	00:03:32	0.6220	00:03:29	0.6265
8	20	32	00:02:40	0.6341	00:02:39	0.6519	00:02:28	0.6010	00:02:30	0.6057

Fig. 17. Results of testing environment 1 (Kernel size 5x5)

Testing environment 2:

item	epochs	Batch size	pytorch test 1 time	pytorch test 1 accuracy	pytorch test 2 time	pytorch test 2 accuracy	TF test 1 time	TF test 1 accuracy	TF test 2 time	TF test 2 accuracy
1	10	4	00:11:29	0.715	0:11:35	0.719	00:09:26	0.6632	0:09:41	0.6879
2	10	8	0:09:23	0.7351	0:09:09	0.7223	00:07:42	0.6918	0:07:12	0.6886
3	10	16	0:07:58	0.7034	0:08:00	0.7149	00:05:34	0.6826	0:05:26	0.6967
4	10	32	00:07:48	0.6583	0:07:47	0.648	00:04:42	0.6493	0:04:34	0.6526
5	20	4	00:24:13	0.6846	0:23:24	0.7034	00:21:11	0.6631	0:19:02	0.678
6	20	8	00:19:16	0.7178	0:17:45	0.707	00:14:44	0.6756	0:15:23	0.6911
7	20	16	00:17:27	0.7196	0:17:01	0.7212	00:11:45	0.6979	0:12:11	0.689
8	20	32	00:15:42	0.7255	0:15:21	0.7189	00:09:35	0.704	0:08:53	0.7069

Fig. 18. Results of testing environment 2 (Kernel size 3x3)

item	epochs	Batch size	pytorch test 1 time	pytorch test 1 accuracy	pytorch test 2 time	pytorch test 2 accuracy	TF test 1 time	TF test 1 accuracy	TF test 2 time	TF test 2 accuracy
1	10	4	0:07:26	0.6227	0:07:38	0.6188	0:06:53	0.6026	0:06:43	0.5811
2	10	8	0:04:45	0.64	0:05:02	0.642	0:04:50	0.5949	0:04:46	0.6064
3	10	16	0:03:58	0.6261	0:03:53	0.6206	0:03:15	0.6019	0:03:58	0.5899
4	10	32	0:03:15	0.5868	0:03:18	0.6051	0:02:55	0.5335	0:02:54	0.5449
5	20	4	0:14:15	0.6001	0:14:53	0.595	0:13:53	0.5847	0:13:54	0.5746
6	20	8	0:09:53	0.6329	0:10:23	0.6285	0:11:40	0.6122	0:10:59	0.5714
7	20	16	0:08:15	0.6463	0:08:05	0.6391	0:08:02	0.607	0:08:02	0.6016
8	20	32	0:06:27	0.6493	0:06:55	0.6558	0:05:57	0.6049	0:06:19	0.5949

Fig. 19. Results of testing environment 2 (Kernel size 5x5)

Testing environment 3:

tem	epochs	Batch size	pytorch test 1 time	pytorch test 1 accuracy	pytorch test 2 time	pytorch test 2 accuracy	TF test 1 time	TF test 1 accuracy	TF test 2 time	TF test 2 accuracy
1	10	4	00:11:41	0.7185	00:11:52	0.7209	00:09:30	0.6853	00:09:29	0.6927
2	10	8	00:08:36	0.7347	00:08:33	0.7303	00:06:15	0.6848	00:06:09	0.7027
3	10	16	00:07:16	0.7096	00:07:21	0.716	00:04:58	0.6888	00:04:57	0.6635
4	10	32	00:07:02	0.6483	00:06:54	0.6487	00:04:46	0.6433	00:04:43	0.6415
5	20	4	00:23:53	0.6932	00:23:39	0.7054	00:18:52	0.6575	00:19:12	0.6737
6	20	8	00:17:16	0.7132	00:17:15	0.7148	00:12:29	0.6940	00:12:34	0.7047
7	20	16	00:14:24	0.719	00:14:25	0.7277	00:09:38	0.6939	00:09:47	0.7140
8	20	32	00:13:57	0.7256	00:13:56	0.7208	00:09:17	0.6963	00:09:19	0.7070

Fig. 20. Results of testing environment 3 (Kernel size 3x3)

tem	epochs	Batch size	pytorch test 1 time	pytorch test 1 accuracy	pytorch test 2 time	pytorch test 2 accuracy	TF test 1 time	TF test 1 accuracy	TF test 2 time	TF test 2 accuracy
1	10	4	00:06:00	0.6302	00:06:12	0.6233	00:07:21	0.5937	00:06:59	0.5807
2	10	8	00:04:23	0.6369	00:04:37	0.6329	00:05:04	0.5802	00:05:00	0.5919
3	10	16	00:03:16	0.6327	00:03:17	0.619	00:03:51	0.5931	00:03:49	0.6016
4	10	32	00:02:43	0.5914	00:02:40	0.595	00:02:44	0.5566	00:02:44	0.5640
5	20	4	00:12:18	0.6302	00:12:30	0.612	00:12:43	0.5978	00:13:05	0.5819
6	20	8	00:08:42	0.6203	00:08:51	0.6361	00:09:43	0.5853	00:09:36	0.5891
7	20	16	00:06:39	0.6509	00:06:31	0.6616	00:07:26	0.6059	00:07:25	0.6099
8	20	32	00:05:31	0.6462	00:05:27	0.6499	00:05:24	0.5969	00:05:21	0.6134

Fig. 21. Results of testing environment 3 (Kernel size 5x5)

Testing environment 4:

tem	epochs	Batch size	pytorch test 1 time	pytorch test 1 accuracy	pytorch test 2 time	pytorch test 2 accuracy	TF test 1 time	TF test 1 accuracy	TF test 2 time	TF test 2 accuracy
1	10	4	0:16:00	0.718	0:15:02	0.7192	00:14:02	0.6898	0:14:23	0.680
2	10	8	00:12:19	0.7318	0:11:23	0.7294	00:09:25	0.6972	0:09:49	0.707
3	10	16	00:10:15	0.7152	0:09:46	0.7136	00:05:35	0.6831	0:05:56	0.680
4	10	32	00:10:27	0.6615	0:10:02	0.6546	00:05:09	0.6433	0:04:56	0.640
5	20	4	00:29:16	0.6811	0:27:20	0.68	00:24:51	0.6777	0:26:14	0.666
6	20	8	00:21:21	0.7142	0:23:15	0.7118	00:18:36	0.6866	0:16:43	0.687
7	20	16	00:20:00	0.7363	0:18:30	0.7339	00:12:03	0.7111	0:11:30	0.697
8	20	32	00:19:54	0.7236	0:18:41	0.7276	00:11:46	0.7062	0:10:42	0.699

Fig. 22. Results of testing environment 4 (Kernel size 3x3)

item	epochs	Batch size	pytorch test 1 time	pytorch test 1 accuracy	pytorch test 2 time	pytorch test 2 accuracy	TF test 1 time	TF test 1 accuracy	TF test 2 time	TF test 2 accuracy
1	10	4	0:08:29	0.6328	0:08:04	0.6359	0:08:22	0.5893	0:09:26	0.5752
2	10	8	0:05:38	0.6478	0:05:40	0.6472	0:05:56	0.6067	0:06:17	0.5952
3	10	16	0:04:33	0.6274	0:04:26	0.6175	0:05:10	0.6041	0:05:04	0.5792
4	10	32	0:03:55	0.6023	0:03:40	0.5775	0:03:29	0.5638	0:03:27	0.5744
5	20	4	0:15:02	0.6126	0:18:49	0.5981	0:21:25	0.583	0:20:27	0.5857
6	20	8	0:10:58	0.6222	0:11:20	0.619	0:13:19	0.5877	0:13:21	0.6091
7	20	16	0:08:32	0.6511	0:09:47	0.6416	0:09:12	0.6094	0:09:30	0.6125
8	20	32	0:07:44	0.6405	0:07:35	0.6401	0:06:47	0.6143	0:06:40	0.6094

Fig. 23. Results of testing environment 4 (Kernel size 5x5)

4.1 Model accuracy comparison

average accuracy 3x3			
epochs	batch size	pytorch	tensorflow
10	4	0.7177875	0.6834
10	8	0.7293375	0.6958125
10	16	0.7115625	0.684975
10	32	0.65605	0.64525
20	4	0.691225	0.667675
20	8	0.7115625	0.6906625
20	16	0.7264125	0.700975
20	32	0.7221875	0.7005625

Fig. 24. Average accuracy when kernel size was 3x3

average acccuracy 5x5			
epochs	batch size	pytorch	tensorflow
10	4	0.624575	0.5830875
10	8	0.642125	0.5950625
10	16	0.6248875	0.599875
10	32	0.5985625	0.5535375
20	4	0.6092375	0.5867375
20	8	0.629725	0.591175
20	16	0.6496125	0.61185
20	32	0.645975	0.6050625

Fig. 25. Average accuracy when kernel size was 5x5

The results from all 4 environments produced similar results on accuracy among PyTorch and TensorFlow tests, with the accuracy generated by PyTorch being a little higher than TensorFlow in nearly all cases. The above Fig. 24 and Fig. 25 demonstrated the average accuracy of all test cases with different epochs and batch sizes. TensorFlow seems to have no advantages in terms of accuracy in our test, with constantly 0.01 to 0.03 less accuracy model generated compared to PyTorch. The largest difference in accuracy was 0.0470265 when the kernel size was 5x5 with 10 epochs and batch size of 8 and the smallest difference in accuracy was 0.0108 when the kernel size was 3x3 with 10 epochs and batch size of 32. The average difference in accuracy generated was 0.0309453 and the standard deviation of difference in accuracy was 0.0103132.

4.2 Execution time

epochs	batch size	avg pytorch time	avg TF time	% TF slower
10	4	254	547	53.56489945
10	8	203.5	342.5	40.58394161
10	16	185	251	26.29482072
10	32	169	161	-4.968944099
20	4	502.5	1186	57.6306914
20	8	406	728.5	44.26904598
20	16	367	515	28.73786408
20	32	339	342	0.877192982

Fig. 26. Average execution time based on testing environment 1 (kernel size 3x3)

epochs	batch size	avg pytorch time	avg TF time	% TF slower
10	4	692	573.5	-20.66259808
10	8	556	447	-24.38478747
10	16	479	330	-45.15151515
10	32	467.5	278	-68.16546763
20	4	1428.5	1206.5	-18.40033154
20	8	1110.5	903.5	-22.91090205
20	16	1034	718	-44.01114206
20	32	931.5	554	-68.14079422

Fig. 27. Average execution time based on testing environment 2 (kernel size 3x3)

epochs	batch size	avg pytorch time	avg TF time	% TF slower
10	4	706.5	569.5	-24.05618964
10	8	514.5	372	-38.30645161
10	16	438.5	297.5	-47.39495798
10	32	418	284.5	-46.92442882
20	4	1426	1142	-24.86865149
20	8	1035.5	751.5	-37.7910845
20	16	864.5	582.5	-48.41201717
20	32	836.5	558	-49.91039427

Fig. 28. Average execution time based on testing environment 3 (kernel size 3x3)

epochs	batch size	avg pytorch time	avg TF time	% TF slower
10	4	931	852.5	-9.208211144
10	8	711	577	-23.22357019
10	16	600.5	345.5	-73.80607815
10	32	614.5	302.5	-103.1404959
20	4	1698	1532.5	-10.79934747
20	8	1338	1059.5	-26.28598395
20	16	1155	706.5	-63.48195329
20	32	1157.5	674	-71.73590504

Fig. 29. Average execution time based on testing environment 4 (kernel size 3x3)

In terms of execution time, testing environment 2, 3 and 4 generated a similar results pattern that TensorFlow performed much faster than PyTorch when kernel size was 3x3. According to Fig. 27, when the epoch was 10, the percentage indicating how TensorFlow was faster than PyTorch increased gradually from 21% faster when the batch size was 4 to 68% faster when the batch size was 32. The change in epoch would produce the same results pattern in percentage increase. Meanwhile, although testing environment 4 generated the same results as the results from environment 2 and 3, which TensorFlow outperform PyTorch and the

relationship between batch size and the performance difference between TensorFlow and PyTorch was in inverse relation, the increase in percentage was much significant from 9% faster to 103% faster during epoch equals to 10 and 10% faster to 72% faster during epoch equals to 20.

However, the test result generated by testing environment 1 showed a completely opposite result, that TensorFlow was much slower than PyTorch except the case of batch size equals to 32 which generated similar performance compared to PyTorch. When the batch size was 4 and epoch was 10, TensorFlow performed 54% slower than PyTorch. Once the batch size increased, the performance difference between TensorFlow and PyTorch decreased and they basically had the same performance when batch size was 32. The same situation also occurred when the epoch was 20. The cpu architecture of testing environment 1 was based on ARM compared to X64 on other 3 environments, that the normal TensorFlow version stated by the official website could not be installed and tensorflow-macos were installed instead according to Apple Developer website, which may illustrate why the performance were different from others [8]. Compared to results from environment 3, in the test with 10 epochs and batch size equals to 4, PyTorch ran 64% faster and Tensorflow ran 4% faster, which may be illustrated as Tensorflow was not optimised as good as PyTorch on the new architecture.

epochs	batch size	avg pytorch time	avg TF time	% TF slower
10	4	148	230	35.65217391
10	8	110.5	145	23.79310345
10	16	90.5	104	12.98076923
10	32	80	75.5	-5.960264901
20	4	295.5	469	36.99360341
20	8	220	292	24.65753425
20	16	181.5	210.5	13.77672209
20	32	159.5	149	-7.046979866

Fig. 30. Average execution time based on testing environment 1 (kernel size 5x5)

epochs	batch size	avg pytorch time	avg TF time	% TF slower
10	4	452	418	-8.133971292
10	8	293.5	283	-3.71024735
10	16	235.5	216.5	-8.775981524
10	32	196.5	174.5	-12.60744986
20	4	874	833.5	-4.859028194
20	8	608	679.5	10.52244297
20	16	490	482	-1.659751037
20	32	401	368	-8.967391304

Fig. 31. Average execution time based on testing environment 2 (kernel size 5x5)

epochs	batch size	avg pytorch time	avg TF time	% TF slower
10	4	366	430	14.88372093
10	8	270	302	10.59602649
10	16	196.5	230	14.56521739
10	32	161.5	164	1.524390244
20	4	744	774	3.875968992
20	8	526.5	579.5	9.145815358
20	16	395	445.5	11.335578
20	32	329	322.5	-2.015503876

Fig. 32. Average execution time based on testing environment 3 (kernel size 5x5)

epochs	batch size	avg pytorch time	avg TF time	% TF slower
10	4	496.5	534	7.02247191
10	8	339	366.5	7.503410641
10	16	269.5	307	12.21498371
10	32	227.5	208	-9.375
20	4	1015.5	1256	19.14808917
20	8	669	800	16.375
20	16	549.5	561	2.049910873
20	32	459.5	403.5	-13.87856258

Fig. 33. Average execution time based on testing environment 4 (kernel size 5x5)

When the kernel size was 5x5, the results of environment 1 showed 35% slower when epochs equals to 10 and batch sizes was 4 and the time difference between TensorFlow and PyTorch would decreased until when epochs equals to 10 and batch sizes was 32 which TensorFlow performed 5% faster. The same scenario also occurred when the epoch was 20.

However, the results provided by environment 2, 3 and 4 showed similar execution time between TensorFlow and PyTorch. The results of environment 2 demonstrated that in most

cases TensorFlow performed 1% to 8% faster except in the test case with 20 epochs and batch size of 8, while TensorFlow performed 10% slower. However, the results of environment 3 and 4 demonstrated the opposite results, which the results from environment 3 showed that in most cases TensorFlow performed 2% to 15% slower except in the test case with 20 epochs and batch size of 32, while TensorFlow performed 2% faster.

Compared to the results obtained from 5x5 to 3x3, testing environment 1 performed consistently that when the batch size decreased, the execution time difference between PyTorch and TensorFlow increased which TensorFlow performed slower than than PyTorch. Meanwhile, when the batch size was 32, the performance of TensorFlow was a little faster in most cases. On the other hand, the 5x5 results obtained from environment 2/3/4 showed similar execution time between PyTorch and TensorFlow, unlike the 3x3 results, which may be explained by the smaller number of parameters in the 5x5 program.

5. Fairness

In general, the tests done in this report are fair in terms of the scope of the two python programs. Total 4 environments were chosen with 2 operating systems, 4 different CPU chips and 2 different CPU architectures were tested. This wide variety covers some of the common environments people will use when implementing PyTorch and TensorFlow. Meanwhile, each test case was tested twice on each environment, generating a total 8 results of each case. If two results of the same test case from the same environment differed significantly, the test would be reimplemented to ensure the results were similar. This should ensure the performance in each environment was accurate. In addition, almost identical PyTorch and TensorFlow versions were selected, ensuring the results were comparable among different environments.

However, although the results are fair, some factors or scenarios cannot be reflected from the results. For instance, the results can not represent the execution time when GPU is used instead of CPU to train the model. Both PyTorch and TensorFlow support CUDA which is a parallel computing platform and programming model offered by Nvidia through its own selected GPUs to support computensive accelerations [2][7][22]. The GPU acceleration which is supposed to speed up the training execution time is common for how people used to implement machine learning algorithms. On the other hand, CPU/GPU and RAM usage was not measured in our test, which may be an important consideration for some users who only

have limited resources like ram on their machine. Finally, other types of neural networks were not tested and the results shown on this report may not apply to the program with other filters/ neural networks.

6. Pratical Problems

Data selection is the one of the most important practical problems needed to be considered, especially in supervised learning like the image classification program in our test, which highly affected the model accuracy. It is difficult to determine the quantity or type of data needed for each label to obtain features effectively. For instance, images of each label are required to implement an image classifier, which the developer not only needs to consider the quantity of samples, but also the resolution and how the images show the object. Therefore, developers are always encouraged to start with a smaller amount of data to review if their plan, the data selection and the parameters of the program is feasible to generate the desired results [23].

After the success of the small amount of data, more data should be implemented for training to further improve the accuracy of the model for practical use, which these large datasets may be a headache to process by the computer with slow training speed. It is suggested to review the format of the data files to reduce the memory required and increase the speed of loading [23]. In addition, cloud computing platforms like Amazon Web Services are often encouraged to be implemented, in which the CPU/GPU cores and amount of RAM can be scaled up or down anytime to fulfill the latest needs of the development.

7. Recommendations

Based on our own test results, TensorFlow outperformed PyTorch in execution time when running on X86(64) machines which most people use nowadays, and better accuracy could be obtained from PyTorch. However, these cannot be the only consideration when choosing between TensorFlow or PyTorch. TensorFlow is more mature and production ready compared to PyTorch for building large scale deep-learning models like text-based applications, image recognition and voice search, which its implementations are commonly used by international enterprise like Facebook’s image recognition system Siri voice recognition and every Google app [24].

In contrast, PyTorch has higher flexibility in the training and testing process of the learning. The structure of the training and testing process are shown in the program and developers can modify the code to check the output or try to use different structures to get better performance, which developers can only modify some parameters in training and testing process of TensorFlow through fit() and evaluate() functions but the structure cannot be modified. This flexibility allows developers to apply their creative thinking which may build a better training model. Meanwhile, the language of PyTorch is simpler which is more suitable for beginners or making rapid prototypes [24].

Meanwhile, for the user currently using Macs with Apple Silicon including M1, M1 Pro and M1 Max chips, PyTorch may be a better option if the objective can be fulfilled by both TensorFlow and PyTorch. PyTorch not only performed better on both execution time and accuracy in our tests but also simpler to be installed. As mentioned, TensorFlow currently has limited support on Apple Silicon with no installation tutorial mentioned on the official TensorFlow website.

macOS

CPU

Version	Python version	Compiler	Build tools
tensorflow-2.7.0	3.7-3.9	Clang from xcode 10.11	Bazel 3.7.2
tensorflow-2.6.0	3.6-3.9	Clang from xcode 10.11	Bazel 3.7.2
tensorflow-2.5.0	3.6-3.9	Clang from xcode 10.11	Bazel 3.7.2
tensorflow-2.4.0	3.6-3.8	Clang from xcode 10.3	Bazel 3.1.0
tensorflow-2.3.0	3.5-3.8	Clang from xcode 10.1	Bazel 3.1.0
tensorflow-2.2.0	3.5-3.8	Clang from xcode 10.1	Bazel 2.0.0
tensorflow-2.1.0	2.7, 3.5-3.7	Clang from xcode 10.1	Bazel 0.27.1
tensorflow-2.0.0	2.7, 3.5-3.7	Clang from xcode 10.1	Bazel 0.27.1
tensorflow-2.0.0	2.7, 3.3-3.7	Clang from xcode 10.1	Bazel 0.26.1
tensorflow-1.15.0	2.7, 3.3-3.7	Clang from xcode 10.1	Bazel 0.26.1
tensorflow-1.14.0	2.7, 3.3-3.7	Clang from xcode	Bazel 0.24.1
tensorflow-1.13.1	2.7, 3.3-3.7	Clang from xcode	Bazel 0.19.2
tensorflow-1.12.0	2.7, 3.3-3.6	Clang from xcode	Bazel 0.15.0
tensorflow-1.11.0	2.7, 3.3-3.6	Clang from xcode	Bazel 0.15.0
tensorflow-1.10.0	2.7, 3.3-3.6	Clang from xcode	Bazel 0.15.0

Fig. 34. TensorFlow version for macOS listed on TensorFlow official website [25]

Fig. 34 listed the version supported for macOS according to TensorFlow, however, none of them worked on our M1 Pro Macbook Pro (testing environment 1). All the packages mentioned can only be installed on intel-based Macs like our testing environment 3 and 4. According to Apple [8], users with apple silicon chips and macOS 12.0 (Monterey) or later can install TensorFlow with tensorflow-macos for CPU processing and tensorflow-metal for GPU acceleration through Metal. However, the installation is still not easy as errors were faced by our members when inputting the command mentioned on the Apple website to the

machine, which can be solved by adding '--no-dependencies' after 'pip install tensorflow-macos==2.6.0'. This frustrating installation process with its poor performance obtained can significantly affect productivity especially on high demand projects.

8. Conclusion

In this report, we have reviewed the similarities and difference between two popular machine learning libraries - PyTorch and TensorFlow, and implemented a simple test on 4 machines to test their performance on execution time and accuracy by running the python program with same features, kernel size, epoch and batch size. The results showed that tensorflow has faster execution time on X86(64) based machines and PyTorch has a little better accuracy. These factors may be considered by users if both PyTorch and TensorFlow can fulfill the needs of the projects. Otherwise, TensorFlow is a more mature platform for production with lots of tutorials and documents provided by its community and commonly used by enterprises. In contrast, PyTorch is more friendly for beginners and more suitable for building quick prototypes. However, these are not the only considerations and requirements in order to make a good machine learning program, selection of training data also plays a significant role to the accuracy and effectiveness of the model and may be more important than the selection of a machine learning framework itself.

References

[1] M. N. Gevorkyan, A. V. Demidova, T. S. Demidova, and A. A. Sobolev, “Review and comparative analysis of machine learninglibraries for machine learning,” DCM&ACS, vol. 27, no. 4, pp. 305-315, 2019. [Online]. doi: 10.22363/2658-4670-2019-27-4-305-315 [Accessed Oct 25, 2021]

[2] PyTorch, *CUDA SEMANTICS,* 2019. [Online]. Available: https://pytorch.org/docs/stable/notes/cuda.html [Accessed Oct 23, 2021]

[3] Y. Jain, *Tensorflow or PyTorch : The force is strong with which one?,* 2018. [Online]. Available: https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4 [Accessed Oct 23, 2021]

[4] EDUCBA, *What is PyTorch*, 2020. [Online]. Available: https://www.educba.com/what-is-pytorch/ [Accessed Oct 23, 2021]

[5] G. Boesch, *Pytorch vs Tensorflow: A Head-to-Head Comparison*, 2021. [Online]. Available: https://viso.ai/deep-learning/pytorch-vs-tensorflow/ [Accessed Oct 25, 2021]

[6] TensorFlow, *Install TensorFlow 2,* 2021. [Online]. Available: https://www.tensorflow.org/install [Accessed Oct 24, 2021]

[7] Nvidia, *GPU-ACCELERATED TENSORFLOW,* 2021. [Online]. Available: https://www.nvidia.com/en-sg/data-center/gpu-accelerated-applications/tensorflow/] [Accessed Nov 1, 2021]

[8] Apple, *Getting Started with tensorflow-metal PluggableDevice,* 2021. [Online]. Available: https://developer.apple.com/metal/tensorflow-plugin/ [Accessed Nov 7, 2021]

[9] JavaTpoint, *Advantage and Disadvantage of TensorFlow*, n.d. . [Online]. Available: https://www.javatpoint.com/advantage-and-disadvantage-of-tensorflow [Accessed Oct 26, 2021]

[10] Paperswithcode, *Trends*, 2021. [Online]. Available: https://paperswithcode.com/trends [Accessed Oct 23, 2021]

[11] Google, Trends, 2021. [Online]. Available: https://trends.google.com/trends/explore?cat=1299&date=today%205-y&q=Pytorch,tensorflow [Accessed Oct 23, 2021]

[12] SLINTEL, *TENSORFLOW VS PYTORCH,* 2021. [Online]. Available: https://www.slintel.com/tech/data-science-machine-learning/tensorflow-vs-pytorch [Accessed Oct 23, 2021]

[13] PyTorch, *TRAINING A CLASSIFIER,* 2021. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#what-about-data [Accessed Oct 30, 2021]

[14] TensorFlow, *Convolutional Neural Network (CNN),* 2021. [Online]. Available:https://tensorflow.google.cn/tutorials/images/cnn [Accessed Nov 2, 2021]

[15] A. Krizhevsky, *The CIFAR-10 dataset*, n.d. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html[Accessed Oct 30, 2021]

[16] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53[Accessed Nov 5, 2021]

[17] JavaTpont, *Working of Convolutional Neural Network*, n.d. [Online]. Available: https://www.javatpoint.com/working-of-convolutional-neural-network-tensorflow [Accessed Nov 5, 2021]

[18] S. Sharma, Epoch vs Batch Size vs Iterations, 2017. [Online]. Available: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9 [Accessed, Nov 11, 2021]

[19] A. Thevenot, *Conv2d: Finally Understand What Happens in the Forward Pass,* 2020. [Online]. Available: https://towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148[Accessed Nov 11, 2021]

[20] A. Rosebrock, *Keras Conv2D and Convolutional Layers*, 2018. [Online]. Available: https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/ [Accessed Nov 12, 2021]

[21] R. Vasudev, *Understanding and Calculating the number of Parameters in Convolution Neural Networks (CNNs),*2019. [Online]. Available: https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d [Accessed Nov 11, 2021]

[22] M. Heller, *What is CUDA? Parallel programming for GPUs*, 2018. [Online]. Available: https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html [Accessed Nov 8, 2021]

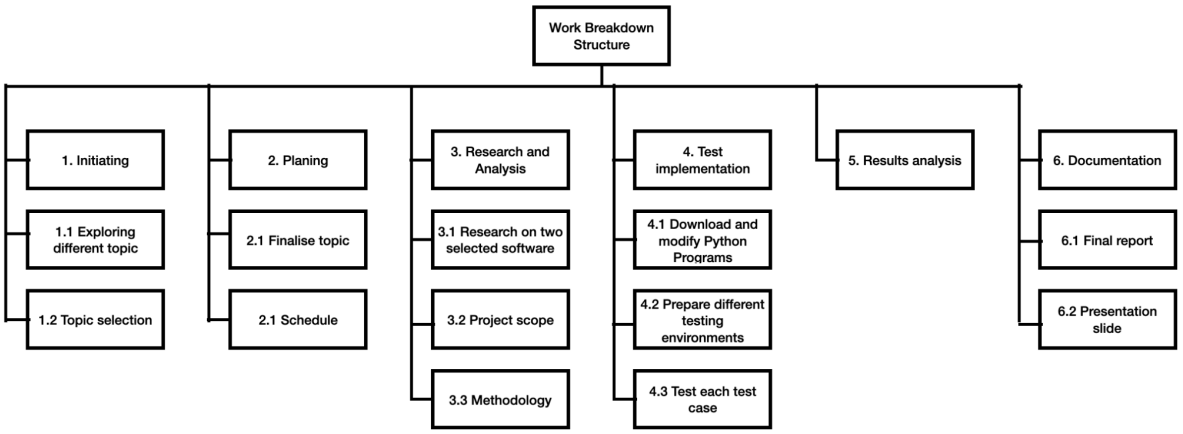
[23] J. Brownlee, *7 Ways to Handle Large Data Files for Machine Learning*, 2017. [Online]. Available: https://machinelearningmastery.com/large-data-files-machine-learning/ [Accessed Nov 7, 2021]

[24] S. Welch, *Pytorch vs. TensorFlow: What You Need to Know, 2020*. [Online]. Available: https://www.udacity.com/blog/2020/05/pytorch-vs-tensorflow-what-you-need-to-know.html [Accessed Nov 2, 2021]

[25] TensorFlow, *Build from source*, 2021. [Online]. Available: https://www.tensorflow.org/install/source [Accessed Nov 24, 2021]

Appendix - Project Plan

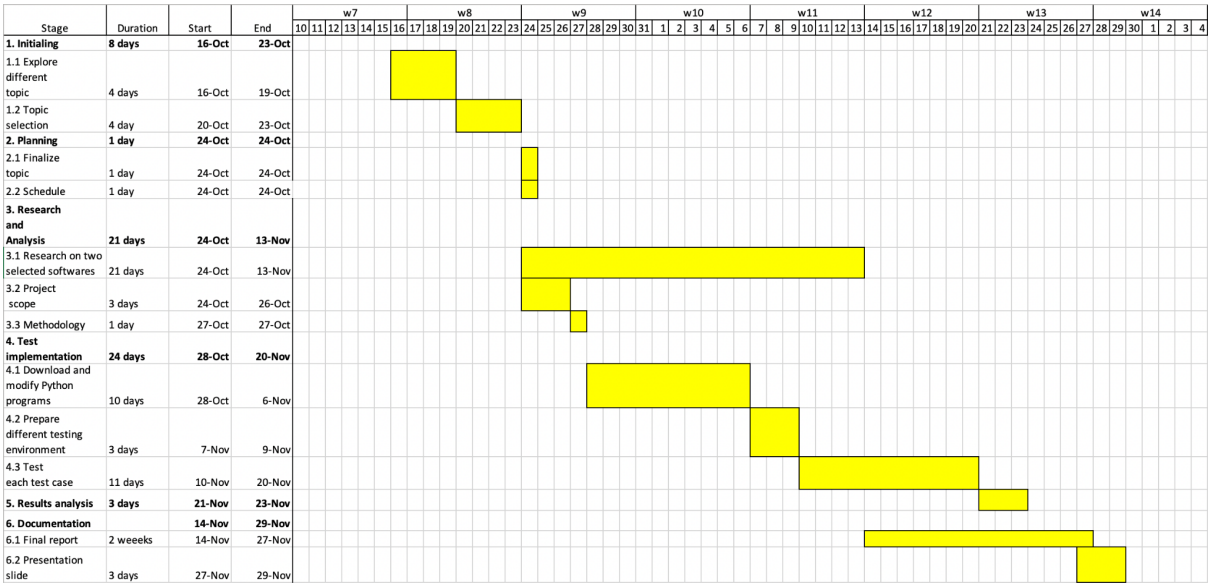
Work Breakdown Structure (WBS)



WBS Dictionary

1. Initialing
   1.1 Exploring different topic
   1.2 Topic selection - select which category of software need to compare
2. Planning
   2.1 Finalize topic - select which two softwares need to compare
   2.2 Schedule - planning the whole project schedule
3. Research and Analysis
   3.1 Research on two selected softwares - features, pro & cons, installation method
   3.2 Project scope
   3.3 Methodology
4. Test implementation
   4.1 Download and modify Python programs
   4.2 Prepare different testing environments
   4.3 Test each test case
5. Results analysis
6. Documentation
   6.1 Final report
   6.2 Presentation slide

Gantt Chart



Stage	Duration	Start	End
1. Initialing	**8 days**	**16-Oct**	**23-Oct**
1.1 Explore different topic	4 days	16-Oct	19-Oct
1.2 Topic selection	4 day	20-Oct	23-Oct
2. Planning	**1 day**	**24-Oct**	**24-Oct**
2.1 Finalize topic	1 day	24-Oct	24-Oct
2.2 Schedule	1 day	24-Oct	24-Oct
3. Research and Analysis	**21 days**	**24-Oct**	**13-Nov**
3.1 Research on two selected softwares	21 days	24-Oct	13-Nov
3.2 Project scope	3 days	24-Oct	26-Oct
3.3 Methodology	1 day	27-Oct	27-Oct
4. Test implementation	**24 days**	**28-Oct**	**20-Nov**
4.1 Download and modify Python programs	10 days	28-Oct	6-Nov
4.2 Prepare different testing environment	3 days	7-Nov	9-Nov
4.3 Test each test case	11 days	10-Nov	20-Nov
5. Results analysis	**3 days**	**21-Nov**	**23-Nov**
6. Documentation		**14-Nov**	**29-Nov**
6.1 Final report	2 weeeks	14-Nov	27-Nov
6.2 Presentation slide	3 days	27-Nov	29-Nov

Contribution table

Section	Student	Percentage
Introduction	Tang Siwei	15%
	Wei Yu	40%
	Lam Tsz Pok	15%
	Li Yat Long	30%
Python code	Li Yat Long	60%
	Tsang Hang Ching	30%
	Huang Zuo	10%
Methodology (Report)	Li Yat Long	100%
Python Testing	Li Yat Long	50%
	Huang Zuo	25%
	Tsang Hang Ching	25%

Results analysis	Li Yat Long	100%
Fairness	Li Yat Long	100%
Pratical problems	Huang Zuo	50%
	Li Yat long	50%
Recommendation	Tang Siwei	30%
	Li Yat Long	40%
	Tsang Hang Ching	30%
Conclusion	Li Yat Long	100%
Presentation slides	Li Yat Long	50%
	Tang Siwei	20%
	Huang Zuo	10%
	Zhong weiqi	10%
	Wei Yu	10%