ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

TÍNH TOÁN SONG SONG

ĐỀ TÀI 3:

Đánh giá hiệu năng các thư viện Deep Learning trên các loại GPU khác nhau

---

Giáo viên hướng dẫn PGS.TS Thoại Nam
Lớp: L01, Nhóm: 9
Danh sách thành viên: Chung Minh Đệ - 1711020
Nguyễn Trương Đình Quân - 1712825
Trần Tiến Vũ - 1714023

Ho Chi Minh, 11/2019

Contents

1 Giới thiệu đề tài

- Đánh giá hiệu năng của các thư viện về Deep Learning như TensorFlow - Keras, Pytorch,... trên các loại GPU cards khác nhau.

2 Cơ Sở Lý thuyết

2.1 AI - Machine Learning - Deep Learning - Neural Network

- Artificial Intelligence(AI): hay còn gọi là trí thông minh nhân tạo, là một ngành thuộc lĩnh vực khoa học máy tính (Computer science), là trí tuệ do con người lập trình tạo nên với mục tiêu giúp máy tính có thể tự động hóa các hành vi thông minh như con người.

- Machine Learning(ML): là một lĩnh vực con của Trí tuệ nhân tạo(Artificial Intelligence) sử dụng các thuật toán cho phép máy tính có thể học từ dữ liệu để thực hiện các công việc thay vì được lập trình một cách rõ ràng.

- Deep learning(DL): đã và đang là một chủ đề AI được bàn luận sôi nổi. Là một phạm trù nhỏ của machine learning, deep learning tập trung giải quyết các vấn đề liên quan đến mạng Neuron nhân tạo nhằm nâng cấp các công nghệ như nhận diện giọng nói, tầm nhìn máy tính và xử lý ngôn ngữ tự nhiên. Deep learning đang trở thành một trong những lĩnh vực hot nhất trong khoa học máy tính. Chỉ trong vài năm, deep learning đã thúc đẩy tiến bộ trong đa dạng các lĩnh vực như nhận thức sự vật (object perception), dịch tự động (machine translation), nhận diện giọng nói,... những vấn đề từng rất khó khăn với các nhà nghiên cứu trí tuệ nhân tạo.

- Neural Network: Neural Network là một hệ thống các chương trình và cấu trúc dữ liệu mô phỏng cách vận hành của não người. Một mạng thần kinh như vậy thường bao gồm một lượng lớn các vi xử lý hoạt động song song, mỗi vi xử lý chứa đựng một vùng kiến thức riêng và có thể truy cập vào các dữ liệu trong bộ nhớ riêng của mình (đôi khi chúng không nhất thiết phải là phần cứng mà có thể là các phần mềm và giải thuật).

2.2 Các thư viện hỗ trợ cho Deep Learning

2.2.1 TensorFlow - Keras

- TensorFlow là một thư viện phần mềm mã nguồn mở dành cho máy học trong nhiều loại hình tác vụ nhận thức và hiểu ngôn ngữ. Nó hiện đang được sử dụng cho cả nghiên cứu lẫn sản xuất bởi 50 đội khác nhau trong nhiều sản phẩm thương mại của Google, như nhận dạng giọng nói, Gmail, Google Photos, và tìm kiếm, nhiều trong số đó đã từng sử dụng chương trình tiền nhiệm DistBelief của nó. TensorFlow nguyên thủy được phát triển bởi đội Google Brain cho mục đích nghiên cứu và sản xuất của Google và sau đó được phát hành theo giấy phép mã nguồn mở Apache 2.0 vào ngày 9/11/2015.

- Keras là một thư viện được phát triển vào năm 2015 bởi François Chollet, là một kỹ sư nghiên cứu deep learning tại google. Nó là một mã nguồn mở cho neural network được viết bởi ngôn ngữ python. Keras là một API bậc cao, được viết bằng Python và

có khả năng chạy trên nền của TensorFlow, CNTK hoặc Theano. Nó được phát triển với mục đích là cho phép thử nghiệm nhanh. Có thể đi từ ý tưởng đến kết quả với độ trễ ít nhất có thể là chìa khóa để thực hiện nghiên cứu tốt.

- Keras có một số ưu điểm như:

- Dễ sử dụng, xây dựng model nhanh.
- Có thể run trên cả CPU và GPU.
- Hỗ trợ xây dựng CNN, RNN và có thể kết hợp cả 2.

2.2.2 Pytorch

- PyTorch là một framework được xây dựng dựa trên python cung cấp nền tảng tính toán khoa học phục vụ lĩnh vực Deep learning. Pytorch tập trung vào 2 khả năng chính:

* Một sự thay thế cho bộ thư viện numpy để tận dụng sức mạnh tính toán của GPU.

* Một platform Deep learning phục vụ trong nghiên cứu, mang lại sự linh hoạt và tốc độ.

- Ưu điểm:

  * Mang lại khả năng Debug dễ dàng hơn theo hướng interactively, rất nhiều nhà nghiên cứu và kỹ sư đã dùng cả Pytorch và Tensorflow đều đánh giá cáo Pytorch hơn trong vấn đề Debug và Visualize.

  * Hỗ trợ tốt dynamic graphs.

  * Được phát triển bởi đội ngũ Facebook.

  * Kết hợp cả các API cấp cao và cấp thấp.

- Nhược điểm: Vẫn chưa được hoàn thiện trong việc deploy, áp dụng cho các hệ thống lớn,... được như framework ra đời trước nó như tensorflow.

2.3 CPUs, GPUs và sự khác nhau giữa chúng

2.3.1 CPUs

- CPU(Central Processing Unit) hay còn gọi là bộ xử lý trung tâm. Nó đóng vai trò như bộ não điều khiển hầu hết các thành phần còn lại ở trong máy tính mà tại đó mọi thông tin, thao tác, dữ liệu sẽ được tính toán kỹ lưỡng và đưa ra lệnh điều khiển mọi hoạt động của máy tính.

2.3.2 GPUs

- GPU (Graphics Processing Unit) là bộ xử lý chuyên dụng nhận nhiệm vụ tăng tốc, xử lý đồ họa cho bộ xử lý trung tâm CPU. Rất nhiều tính năng trên GPU vượt xa so với trình điều khiển đồ họa cơ bản như GPU của Intel.

- Gồm có 2 GPU mà nhóm sẽ sử dụng:
    * GPUs Tesla K80 - Google Colab.
    * GPUs Tesla P100 PCIE - Kaggle.

3 Nội dung và phương pháp nghiên cứu

3.1 Dataset

- Dataset được dùng để train model là Dog-Cat Dataset do nhóm sưu tầm và bổ sung, gồm 2 tập là Train Data (25.000 ảnh) và Validation Data(7.000 ảnh), mỗi tập được chia thành 2 lớp (Binary classification) và có 1 tập dữ liệu Test gồm 12500 tấm ảnh chó và mèo chưa được phân loại.

3.2 Model

- Xây dựng 1 model mới dựa trên kiến trúc mạng VGG16 để giải quyết bài toán phân loại ảnh(Image Classification) và gán nhãn cho ảnh.

3.3 Code hiện thực

- Source code hiện thực trên Python sử dụng các thư viện :

- Tensorflow - Keras: "**Bấm vào đây.**"
- Pytorch: "**Bấm vào đây.**"

4 Kết quả thực hiện

4.1 Google Colab

4.1.1 Kết quả trên CPUs

4.1.2 Kết quả trên GPUs - Tesla K80 - Google Colab

- Tensorflow - Keras:

-Dùng ImageDataGenerator để tạo để load hình ảnh vào model.

```
[14] import os
     from keras.preprocessing import image
     import matplotlib.pyplot as plt
     from keras.preprocessing.image import ImageDataGenerator

     train_datagen= ImageDataGenerator(rescale=1./255,rotation_range=180)
     validation_datagen= ImageDataGenerator(rescale=1./255)

     train_generator=train_datagen.flow_from_directory('../content/tem/train',target_size=(224,224),batch_size=64,class_mode='binary')
     validation_generator=validation_datagen.flow_from_directory('../content/tem/validation',target_size=(224,224),batch_size=32,class_mode='binary')
```

```
Found 19652 images belonging to 2 classes.
Found 5348 images belonging to 2 classes.
```

Hình 1

- Xây dựng model tựa vgg16.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               12845568
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 27,691,841
Trainable params: 27,691,841
Non-trainable params: 0
_________________________________________________________________
```

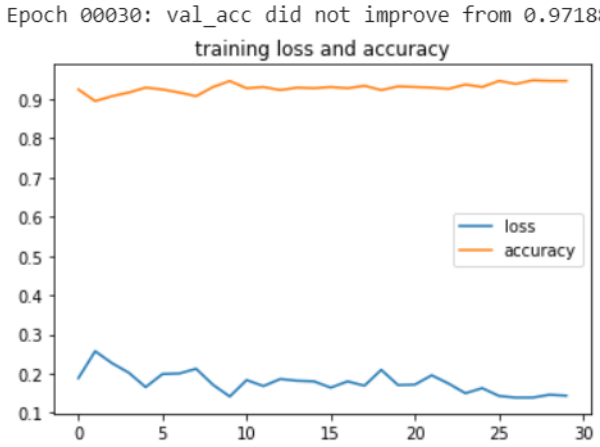
Hình 2

- Kết quả train model và validation.

```
Epoch 1/30
10/10 [==============================] - 15s 2s/step - loss: 0.7314 - acc: 0.4922 - val_loss: 0.6548 - val_acc: 0.6125

Epoch 00001: val_acc improved from -inf to 0.61250, saving model to vgg16_1.h5
Epoch 2/30
10/10 [==============================] - 8s 775ms/step - loss: 0.6672 - acc: 0.5844 - val_loss: 0.6829 - val_acc: 0.5312

Epoch 00002: val_acc did not improve from 0.61250
Epoch 3/30
10/10 [==============================] - 10s 954ms/step - loss: 0.6454 - acc: 0.6125 - val_loss: 0.5532 - val_acc: 0.7375

Epoch 00003: val_acc improved from 0.61250 to 0.73750, saving model to vgg16_1.h5
Epoch 4/30
10/10 [==============================] - 9s 931ms/step - loss: 0.6118 - acc: 0.6656 - val_loss: 0.5253 - val_acc: 0.7625

Epoch 00004: val_acc improved from 0.73750 to 0.76250, saving model to vgg16_1.h5
Epoch 5/30
10/10 [==============================] - 9s 933ms/step - loss: 0.5605 - acc: 0.7172 - val_loss: 0.4814 - val_acc: 0.7875

Epoch 00005: val_acc improved from 0.76250 to 0.78750, saving model to vgg16_1.h5
Epoch 6/30
10/10 [==============================] - 9s 932ms/step - loss: 0.5626 - acc: 0.7203 - val_loss: 0.4881 - val_acc: 0.7781

Epoch 00006: val_acc did not improve from 0.78750
Epoch 7/30
10/10 [==============================] - 9s 933ms/step - loss: 0.5250 - acc: 0.7453 - val_loss: 0.4200 - val_acc: 0.8219

Epoch 00007: val_acc improved from 0.78750 to 0.82188, saving model to vgg16_1.h5
Epoch 8/30
10/10 [==============================] - 9s 926ms/step - loss: 0.5526 - acc: 0.7188 - val_loss: 0.4301 - val_acc: 0.8187

Epoch 00008: val_acc did not improve from 0.82188
Epoch 9/30
10/10 [==============================] - 10s 951ms/step - loss: 0.4703 - acc: 0.7844 - val_loss: 0.5917 - val_acc: 0.6781
```

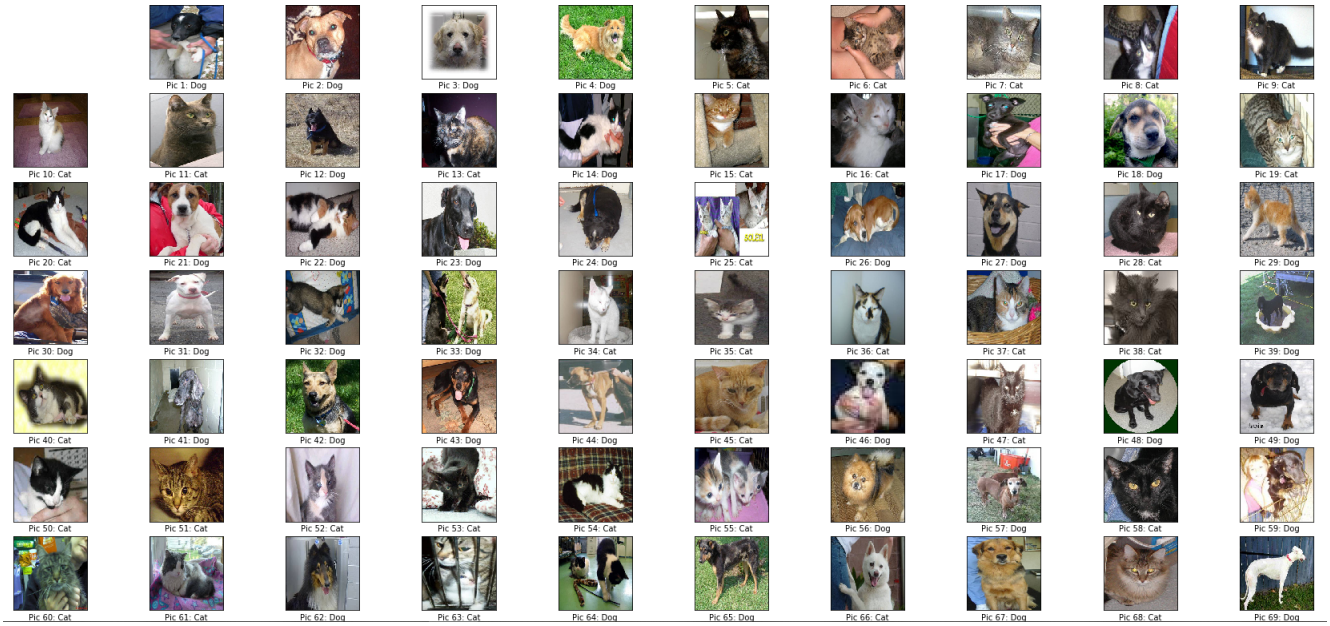
```
Epoch 00009: val_acc did not improve from 0.82188
Epoch 10/30
10/10 [==============================] - 10s 965ms/step - loss: 0.5136 - acc: 0.7594 - val_loss: 0.4412 - val_acc: 0.7844

Epoch 00010: val_acc did not improve from 0.82188
Epoch 11/30
10/10 [==============================] - 9s 943ms/step - loss: 0.3973 - acc: 0.8297 - val_loss: 0.3834 - val_acc: 0.8375

Epoch 00011: val_acc improved from 0.82188 to 0.83750, saving model to vgg16_1.h5
Epoch 12/30
10/10 [==============================] - 9s 950ms/step - loss: 0.3935 - acc: 0.8234 - val_loss: 0.2888 - val_acc: 0.8781

Epoch 00012: val_acc improved from 0.83750 to 0.87813, saving model to vgg16_1.h5
Epoch 13/30
10/10 [==============================] - 9s 940ms/step - loss: 0.3664 - acc: 0.8562 - val_loss: 0.2057 - val_acc: 0.9156

Epoch 00013: val_acc improved from 0.87813 to 0.91563, saving model to vgg16_1.h5
Epoch 14/30
10/10 [==============================] - 9s 934ms/step - loss: 0.3129 - acc: 0.8625 - val_loss: 0.2673 - val_acc: 0.8906

Epoch 00014: val_acc did not improve from 0.91563
Epoch 15/30
10/10 [==============================] - 9s 931ms/step - loss: 0.3539 - acc: 0.8359 - val_loss: 0.6041 - val_acc: 0.7438

Epoch 00015: val_acc did not improve from 0.91563
Epoch 16/30
10/10 [==============================] - 9s 941ms/step - loss: 0.3277 - acc: 0.8453 - val_loss: 0.2031 - val_acc: 0.9094

Epoch 00016: val_acc did not improve from 0.91563
Epoch 17/30
10/10 [==============================] - 10s 950ms/step - loss: 0.2933 - acc: 0.8766 - val_loss: 0.2018 - val_acc: 0.9110

Epoch 00017: val_acc did not improve from 0.91563
Epoch 18/30
10/10 [==============================] - 9s 947ms/step - loss: 0.3147 - acc: 0.8547 - val_loss: 0.1790 - val_acc: 0.9531

Epoch 00018: val_acc improved from 0.91563 to 0.95312, saving model to vgg16_1.h5
Epoch 19/30
10/10 [==============================] - 9s 928ms/step - loss: 0.2740 - acc: 0.8891 - val_loss: 0.3182 - val_acc: 0.8688

Epoch 00019: val_acc did not improve from 0.95312
Epoch 20/30
10/10 [==============================] - 9s 933ms/step - loss: 0.2752 - acc: 0.8766 - val_loss: 0.2171 - val_acc: 0.9187

Epoch 00020: val_acc did not improve from 0.95312
Epoch 21/30
10/10 [==============================] - 9s 936ms/step - loss: 0.3242 - acc: 0.8578 - val_loss: 0.1858 - val_acc: 0.9281

Epoch 00021: val_acc did not improve from 0.95312
Epoch 22/30
10/10 [==============================] - 9s 947ms/step - loss: 0.2352 - acc: 0.8984 - val_loss: 0.1535 - val_acc: 0.9531

Epoch 00022: val_acc did not improve from 0.95312
Epoch 23/30
10/10 [==============================] - 10s 951ms/step - loss: 0.2714 - acc: 0.8750 - val_loss: 0.1332 - val_acc: 0.9437

Epoch 00023: val_acc did not improve from 0.95312
Epoch 24/30
10/10 [==============================] - 9s 944ms/step - loss: 0.2231 - acc: 0.9094 - val_loss: 0.1398 - val_acc: 0.9469

Epoch 00024: val_acc did not improve from 0.95312
Epoch 25/30
10/10 [==============================] - 9s 888ms/step - loss: 0.2604 - acc: 0.8959 - val_loss: 0.1507 - val_acc: 0.9469

Epoch 00025: val_acc did not improve from 0.95312
Epoch 26/30
10/10 [==============================] - 9s 930ms/step - loss: 0.2221 - acc: 0.9062 - val_loss: 0.1835 - val_acc: 0.9281

Epoch 00026: val_acc did not improve from 0.95312
Epoch 27/30
10/10 [==============================] - 9s 942ms/step - loss: 0.2535 - acc: 0.8938 - val_loss: 0.1511 - val_acc: 0.9375

Epoch 00027: val_acc did not improve from 0.95312
Epoch 28/30
10/10 [==============================] - 9s 933ms/step - loss: 0.2317 - acc: 0.9094 - val_loss: 0.1832 - val_acc: 0.9219

Epoch 00028: val_acc did not improve from 0.95312
Epoch 29/30
10/10 [==============================] - 9s 938ms/step - loss: 0.2295 - acc: 0.8984 - val_loss: 0.1564 - val_acc: 0.9469

Epoch 00029: val_acc did not improve from 0.95312
Epoch 30/30
10/10 [==============================] - 9s 950ms/step - loss: 0.2217 - acc: 0.9094 - val_loss: 0.1839 - val_acc: 0.9125

Epoch 00030: val_acc did not improve from 0.95312
```

Hình 3



Hình 4

- Kết quả dự đoán trên tập test với model vgg16 vừa được train.



Hình 5

- Pytorch:

- Load dữ liệu vào model.

```
Loaded 19652 images under train
Loaded 5348 images under validation
Classes:
['cat', 'dog']
```

Hình 6

- Xây dựng model tựa vgg16 .

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:08<00:00, 67.1MB/s]
1000
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

Hình 7

- Kết quả train model và validation.

```
[Epoch 1/3] Iteration 200 -> Train Loss: 0.3347, Accuracy: 0.875, Training complete in 0m 58s
[Epoch 1/3] Iteration 400 -> Train Loss: 0.3265, Accuracy: 0.938, Training complete in 26257958m 39s
[Epoch 1/3] Iteration 600 -> Train Loss: 0.3234, Accuracy: 0.844, Training complete in 1m 54s
[Epoch 2/3] Iteration 800 -> Train Loss: 0.3211, Accuracy: 0.969, Training complete in 26257959m 36s
[Epoch 2/3] Iteration 1000 -> Train Loss: 0.3151, Accuracy: 0.938, Training complete in 2m 51s
[Epoch 2/3] Iteration 1200 -> Train Loss: 0.3261, Accuracy: 0.875, Training complete in 26257960m 33s
[Epoch 3/3] Iteration 1400 -> Train Loss: 0.3292, Accuracy: 0.875, Training complete in 3m 48s
[Epoch 3/3] Iteration 1600 -> Train Loss: 0.3226, Accuracy: 0.812, Training complete in 26257961m 29s
[Epoch 3/3] Iteration 1800 -> Train Loss: 0.3237, Accuracy: 0.750, Training complete in 4m 45s
```

Hình 8

- Kết quả dự đoán trên tập test.

Hình 9

4.2 Kaggle

4.2.1 Kết quả trên CPUs

4.2.2 Kết quả trên GPUs - Tesla P100 - Kaggle

- Tensorflow - Keras:
- Pytorch:

- Load data vào môi trường train.

```
Loaded 19652 images under train
Loaded 5348 images under validation
Classes:
['cat', 'dog']
```

Hình 10

- Xây dựng model tựa vgg16 .

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

Hình 11

- Kết quả train model và validation .

```
[Epoch 1/3] Iteration 200 => Train Loss: 0.3488, Accuracy: 0.812, Training complete in 0m 59s
[Epoch 1/3] Iteration 400 => Train Loss: 0.3481, Accuracy: 0.906, Training complete in 437644m 45s
[Epoch 1/3] Iteration 600 => Train Loss: 0.3389, Accuracy: 0.750, Training complete in 0m 55s
[Epoch 2/3] Iteration 200 => Train Loss: 0.3734, Accuracy: 0.875, Training complete in 437644m 47s
[Epoch 2/3] Iteration 400 => Train Loss: 0.3510, Accuracy: 0.875, Training complete in 0m 52s
[Epoch 2/3] Iteration 600 => Train Loss: 0.3421, Accuracy: 0.969, Training complete in 437644m 44s
[Epoch 3/3] Iteration 200 => Train Loss: 0.3842, Accuracy: 0.906, Training complete in 0m 54s
[Epoch 3/3] Iteration 400 => Train Loss: 0.3480, Accuracy: 0.844, Training complete in 437644m 40s
[Epoch 3/3] Iteration 600 => Train Loss: 0.3496, Accuracy: 0.844, Training complete in 0m 50s
```

Hình 12

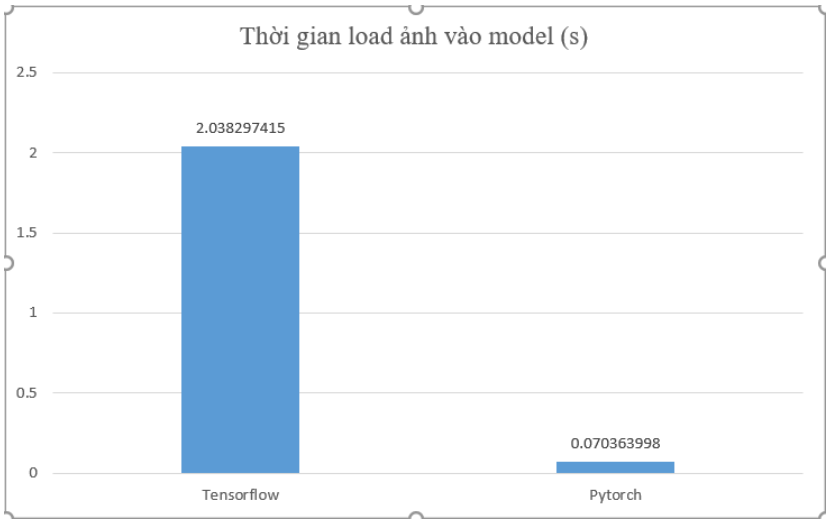
Hình 14

5 Đánh giá kết quả, nhận xét

5.1 Trên Google Colab

- Thời gian load ảnh vào model:

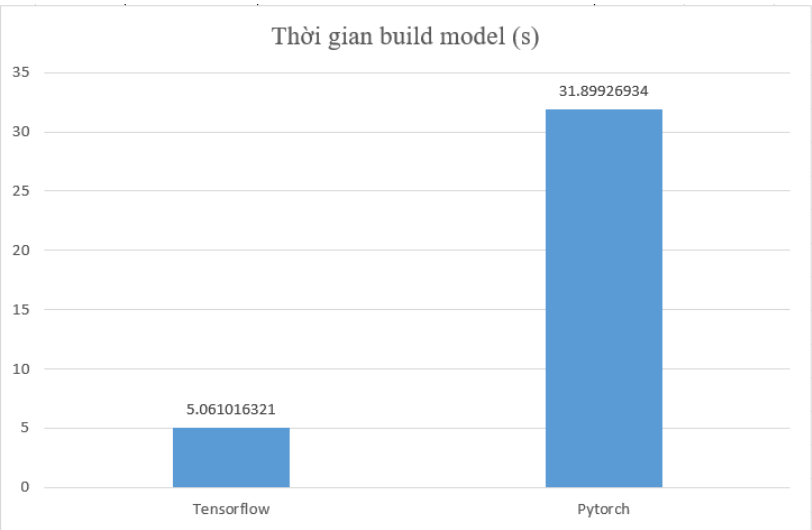
Thời gian load ảnh vào model (s)	
Tensorflow	2.038297415
Pytorch	0.070363998



Hình 15

- Thời gian build model:

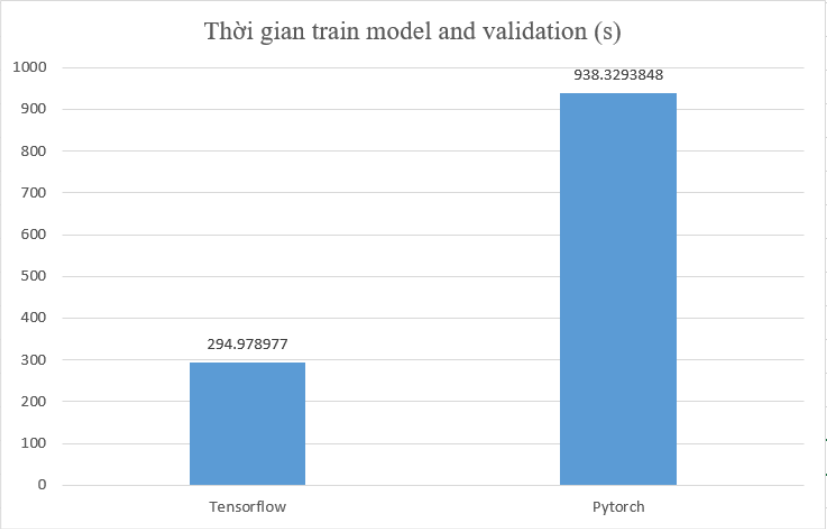
Thời gian build model (s)	
Tensorflow	5.061016321
Pytorch	31.89926934



Hình 16

- Thời gian train model and validation:

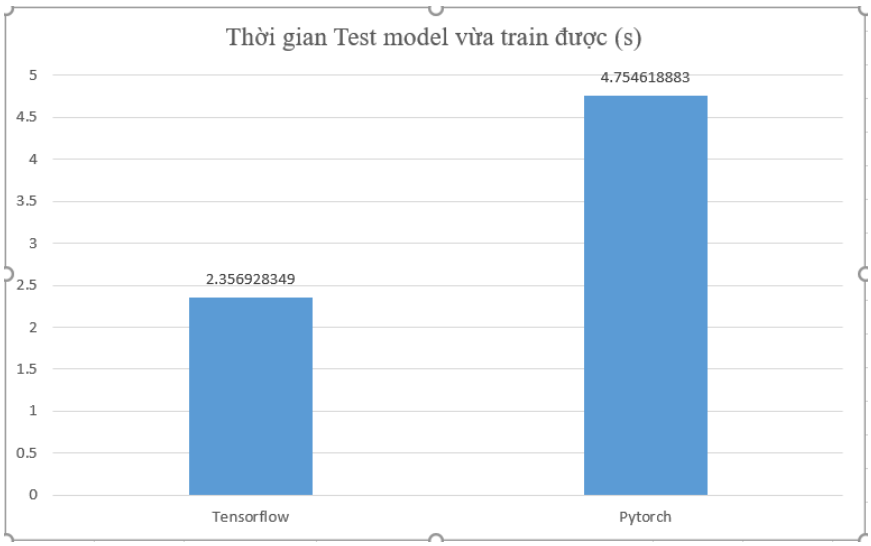
Thời gian train model and validation (s)	
Tensorflow	294.978977
Pytorch	938.3293848



Hình 17

- Thời gian Test model vừa train:

Thời gian Test model vừa train được (s)	
Tensorflow	2.356928349
Pytorch	4.754618883

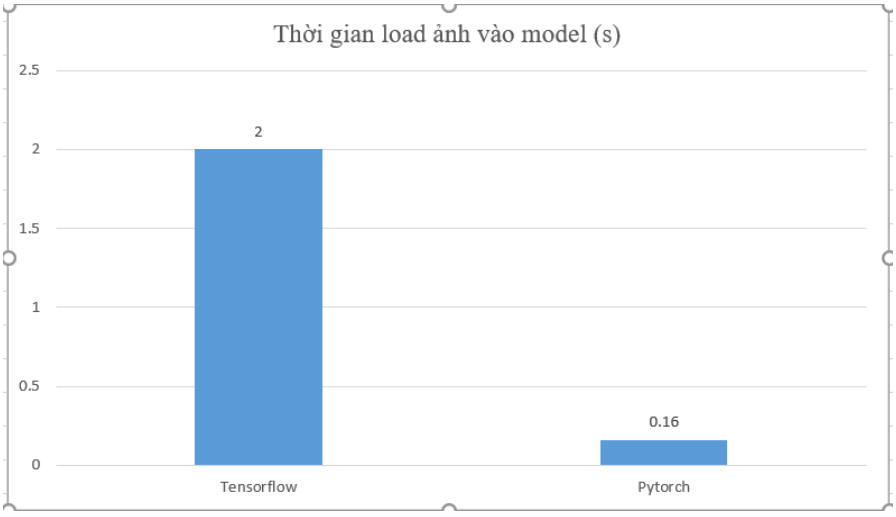


Hình 18

5.2 Trên Kaggle

- Thời gian load ảnh vào model:

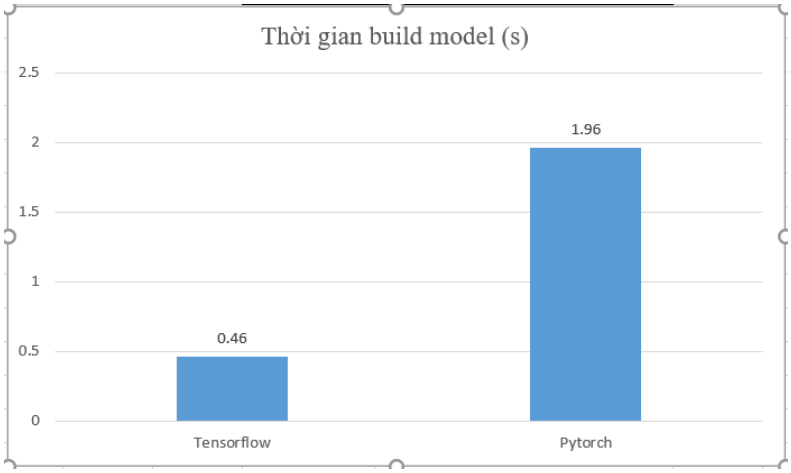
Thời gian load ảnh vào model (s)	
Tensorflow	2
Pytorch	0.16



Hình 19

- Thời gian build model:

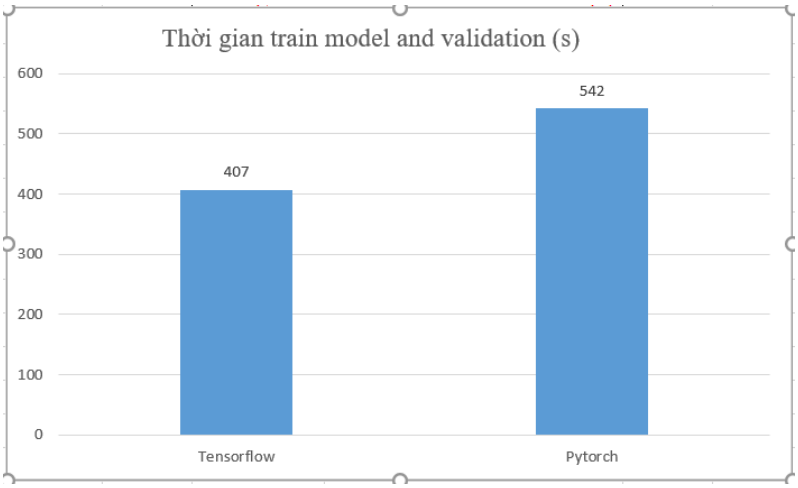
Thời gian build model (s)	
Tensorflow	0.46
Pytorch	1.96



Hình 20

- Thời gian train model and validation:

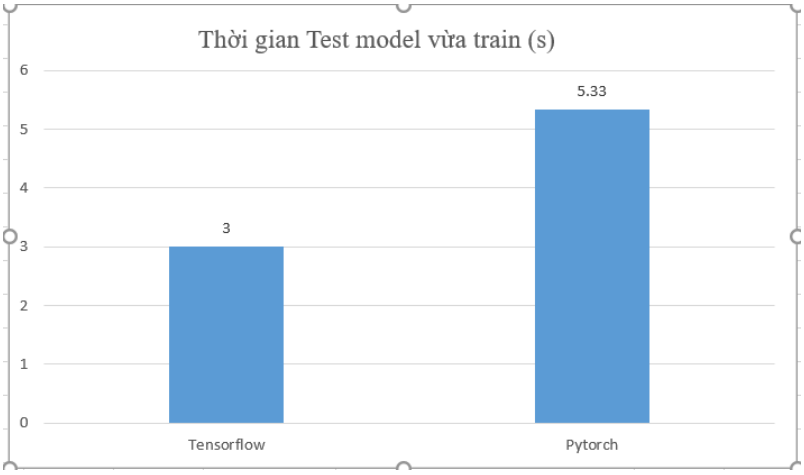
Thời gian train model and validation (s)	
Tensorflow	407
Pytorch	542



Hình 21

- Thời gian Test model vừa train:

Thời gian Test model vừa train (s)	
Tensorflow	3
Pytorch	5.33



Hình 22

6 Kết luận

- Công cụ Deep Neural Learning là một công cụ thường xuyên được sử dụng trong nghiên cứu và trong công nghiệp với một số nghiên cứu đến việc tạo ra các thư viện mới, đánh giá hiệu quả thư viện, thực hiện các kỹ thuật để tối ưu hóa và tạo ra các thiết bị chuyên dụng.

- Công việc hiện tại cho thấy được cách tiếp cận liên quan đến hoạt động của thư viện có sẵn mã nguồn mở, so sánh hiệu suất của chúng trong các GPU và CPU khác nhau.

- Ta thấy được rằng trong bài toán này thư viện Tensorflow có hiệu suất tốt hơn, nhưng Pytorch sử dụng ít tài nguyên GPU hơn so với Tensorflow.

- Một số bài toán sử dụng thư viện Pytorch có hiệu suất tốt hơn. Đầu tiên là do Pytorch sử dụng ít tài nguyên GPU, giảm thiểu khả năng tắc nghẽn giữa GPU và CPU. Thứ hai là Pytorch sử dụng chức năng tự động hóa, ít rời khỏi công việc, khó khăn hơn cho các lập trình viên. Tensorflow thì đơn giản và trực quan hơn nhiều. Tuy nhiên, nó có thể ảnh hưởng đến thuật toán tổng quát, và do đó, trong các thuật toán Neural Networks không được tối ưu hóa.

- Trong tương lai, chúng ta có thể nghiên cứu nhiều hơn vào tác động của từng thư viện trong GPU như là sử dụng nhiều cụm GPU hoặc dùng đánh giá các hiệu suất của các thư viện khi sử dụng mạng CNN và RNN khác. Có thể so sánh nhiều hơn với các thư viện khác như Caffe, Caffe2, Microsoft CNTK, Theano và DeepLearning4j, ...

7 Tài Liệu Tham Khảo

References

[1] Dog Cat dataset refferrent. "**https://www.kaggle.com/c/dogs-vs-cats/data**", last access: 20/10/2019.

[2] "**https://www.analyticsvidhya.com/blog/2019/01/build-image-classification-model-10-minutes/**", last access: 30/10/2019.

[3] Step by step VGG16 implementation in Keras. "**https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c**", last access: 20/10/2019.

[4] https://towardsdatascience.com/how-to-train-an-image-classifier-in-pytorch, "**https://towardsdatascience.com/how-to-train-an-image-classifier-in-pytorch-and-use-it-to-perform-basic-inference-on-single-images-99465a1e9bf5**", last access: 30/10/2019.

8 Phụ lục

- Link Demo của nhóm: "**Bấm vào đây.**"