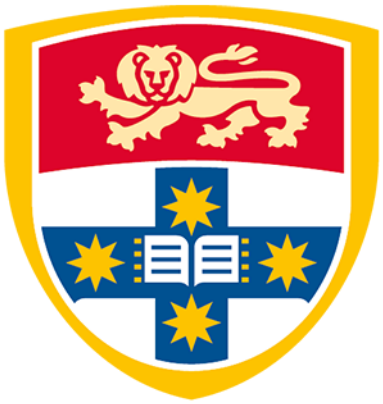
Deep Learning Assignment 1

Lecturer: Chang Xu

Tutor: Kwon Nung Choi

Multi-label Image Classification task: Kaggle Competition



THE UNIVERSITY OF SYDNEY

Anshu Kumar (490517666)
Sanna Nazir (490517677)
Samarth Sehgal (490528857)

May 2020

Contents

1 ABSTRACT

Neural Networks have become an increasingly efficient tool in the domain of computer vision. Some may say, they are even the core building blocks for computer vision technology nowadays. The field has experienced a shift from traditional statistical methods to modern deep learning architectures. This report is an exploration into various publicly available state-of-the-art neural networks for image detection and classification problems. The report explores and contrasts the frameworks of PyTorch and Tensorflow and builds on the technique of Transfer Learning. The report is concluded with self-reflection and potential future work for the students.

2 INTRODUCTION

The purpose of this report is to perform a multi-label image classification task on a relatively large image dataset. The dataset has been provided by the instructor for COMP5329 taught at the University of Sydney for Semester 1, 2020.

2.1 What's the aim of the study?

Similar to the aim of our first assignment, we have analyzed the aim of this study to be two-fold.

- In a big picture view, the assignment/task teaches us the tenets of being a deep learning engineer in a real-world scenario. Image classification is an increasing popular and widely used application of deep neural networks. Furthermore, it is also a widely researched field with major players/companies releasing their own custom built neural networks for greater use by the public. Some examples are: ResNet50, VGG19, AlexNet, ImageNet etc.
  The task enables us to deploy these efficient, pre-trained networks for our own custom applications and datasets. Thus, at a macroscopic level, the project is aimed at teaching the students essential core skills for building deep learning applications.

- At the course/student level, the project is aimed at inculcating the essential data handling and manipulation skills used in deep learning techniques. The key tasks for this project included handling the large amount of data in an efficient manner, balancing of the imbalanced classes, standardizing the irregular images for feeding the networks, employing existing deep infrastructures for custom applications, understanding the pros and cons of deep learning frameworks (used PyTorch and Tensorflow), evaluating and fine-tuning the metrics of the network and finally creating desirable results.

Thus, the aim of the study is to imbibe real-world data learning skills in the students. Along with encouraging the understanding of computer vision field and its diverse applications.

2.2 Why is the study important?

Deep learning is a rich family of methods, encompassing neural networks, hierarchical probabilistic models, and a variety of unsupervised and supervised feature learning algorithms [16]. The recent surge of interest in deep learning methods is due to the fact that they have been shown to outperform previous state-of-the-art techniques in several tasks, as well as the abundance of complex data from different sources (e.g., visual, audio, medical, social, and sensor)[16].

The significance of this study is based in the importance and popularity of deep learning architectures. Deep Neural Networks have revolutionized various fields of human knowledge and have

found applications in every industry. Ranging from healthcare to technology, energy to architecture, construction to e-commerce; deep neural networks are omnipresent. Not only existing industries, deep neural networks have spawned various new fields of human knowledge as well including the acceleration in the field of artificial intelligence. Consequently, accelerating the advent of industry 4.0. **Thus, this study is important because it encourages us to research the field of deep learning as well as build skills to develop deep neural networks of our own.**

2.3 Overall Introduction of used method

For the purposes of this project, we performed experimentation with both Tensorflow and PyTorch frameworks for deep learning algorithms. **However, after extensive testing, our final submission and results are based on the PyTorch framework.**

We have employed the technique of transfer learning using the ResNet50 network, the ResNext network as well as the VGG19 network (on tensorflow) to create the base of our custom neural network. On top of the existing base model, we have enabled layers to train the model on our custom dataset. Following the network creation, we have enabled the output layer with **20 neurons (for 20 classes), sigmoid activation function coupled with BinaryCrossEntropy Loss or BCEWithLogitsLoss** so that the network is able to handle multi-label classification.

Handling of the data is done by means of batching to overcome the problem of memory crashes owing to the large size of the dataset. For our final submission using PyTorch, the inbuilt **DataLoader** class has been employed. For Tensorflow, the same was handled using the **fit_generator** function that enables batching images for fitting on the algorithm.

2.4 Dataset overview

The data for performing this task has been provided by the unit coordinator for COMP5329, taught at the University of Sydney during semester 1, 2020. The dataset is divided into three parts:

1. The images folder: The folder consists of 40,000 images each labelled as 0.jpg, 1.jpg and so on. This dataset is then manually divided into two parts of train_data (consisting of first 30,000 images) and test_data (consisting of next 10,000 images)

2. The train.csv file: Contains a csv with three columns (ImageID, Labels, Captions). The name of the image is stored in the first column with its associated labels and captions in the second and third columns respectively. **For the purposes of this project, we have only use labels and not the captions for building our model.**

3. The test.csv file: Contains a csv with two columns (ImageID, Captions). The name of the image is stored in the first column with its associated captions in the second column. **For the purposes of testing and generating predictions, we have only generated the labels for the test images. (Thus, dropping the captions column.)**

2.4.1 Preliminary Data Exploration

Some visual exploration samples from the given dataset.

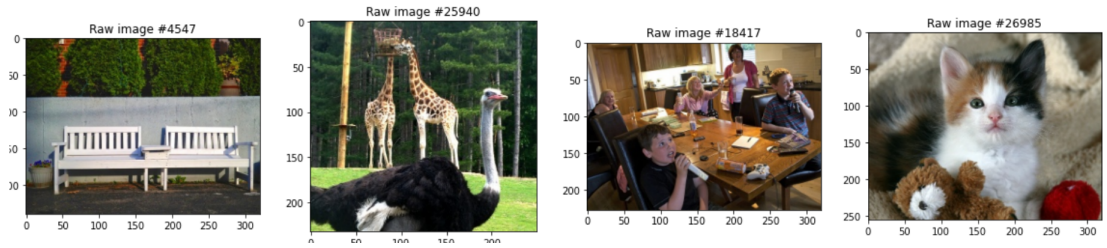


Figure 1: Raw Image Samples

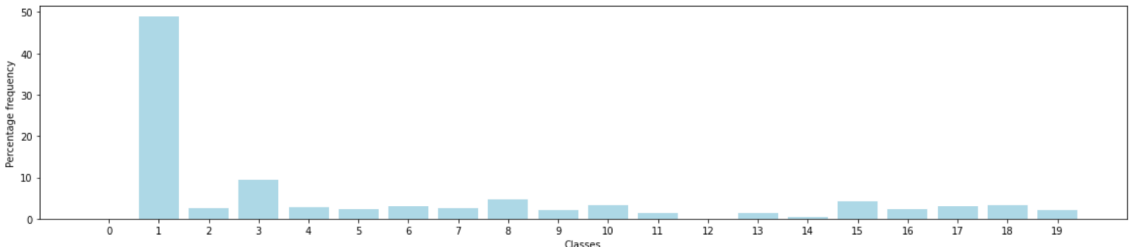


Figure 2: Visualizing Classes distribution in dataset

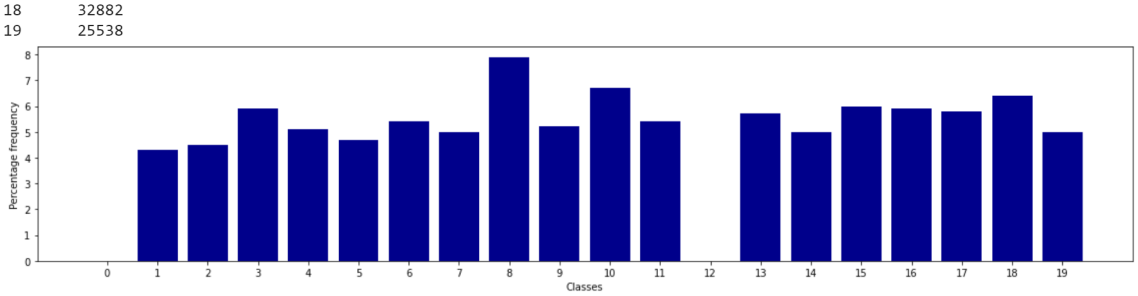


Figure 3: Visualizing Classes distribution after balancing (only for VGG19 in Tensorflow)

```
Labels_train: (30000, 20)
 [[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0.]]

label_dict_train['1478.jpg'] [1, 6]
[0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

label_dict_train['2109.jpg'] [1]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

label_dict_train['1292.jpg'] [1, 2, 3]
[0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Figure 4: One-Hot Encoding the Labels

3 RELATED WORKS

We have divided the research of our previous works across different verticles as below:

3.1 Managing Class Imbalance

Usually this problem deals with recall and precision trade-off. In situation where we want to predict minority class, we either need to use metrics beyond the scope of precision and recall, like cost-sensitive learning. In cost-sensitive learning mis-classification of minority classes are more heavily penalized than that of majority classes[15].

Other simple way of dealing with class imbalance is sampling. Depending on the size of dataset, we can either do oversampling instances of the minority class or under-sampling instances of majority class[15].

There are now some advanced versions of sampling available like SMOTE, which creates new instances for minority class by forming convex combination of neighboring instances[15].

3.2 ResNet

ResNet was the winner of ILSVRC 2015. It became arguably the most grounbreaking work in the field of computer vision after defeation AlexNet.

It was developed by Kaiming He to overcome vanishing gradient problem [11]. ResNet is developed with many different layers in it's different versions: 34, 50, 101, 152 and 1202. ResNet50 has 49 convolution layers and 1 fully-connected layer at the end. The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more than one layers[6].

3.3 Transfer Learning

Transfer learning is a type of machine learning in which a model can be reused as starting point for other model. It is most popular in deep learning, where trained weights of the models are saved and further used in various NLP and Computer Vision tasks [8].

Three major tranfer learning scenarios/usage are:[5]

- It can be used as a fixed feature extractor [5].
- It can be used to fine-tune layers of Neural Network [5].
- Used as pretrained models, as we know some of the neural networks may even take several weeks to train across multiple GPUs [5].

3.4 PyTorch

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs [3]. PyTorch is an optimized tensor library for deep learning which uses GPUs and CPUs. It enables fast, flexible experimentation and efficient production through a user-friendly front-end, distributed training, and ecosystem of tools and libraries.[1]

4 TECHNIQUES

4.1 Principle of Method Used

To facilitate multi-label classification of a very large dataset of images, we are using Transfer Learning combined with DataLoaders in Pytorch. Upon training and testing our data on different pretrained neural networks, we finally picked up ResNet50 to train and evaluate our data on.

4.1.1 Transfer Learning

Transfer Learning is a Deep Learning technique wherein neural network models are already trained on a problem similar to the problem that is being solved[8]. A good example of transfer learning can be trying to recognize trucks by using a pre-trained dataset previously used to recognize cars.

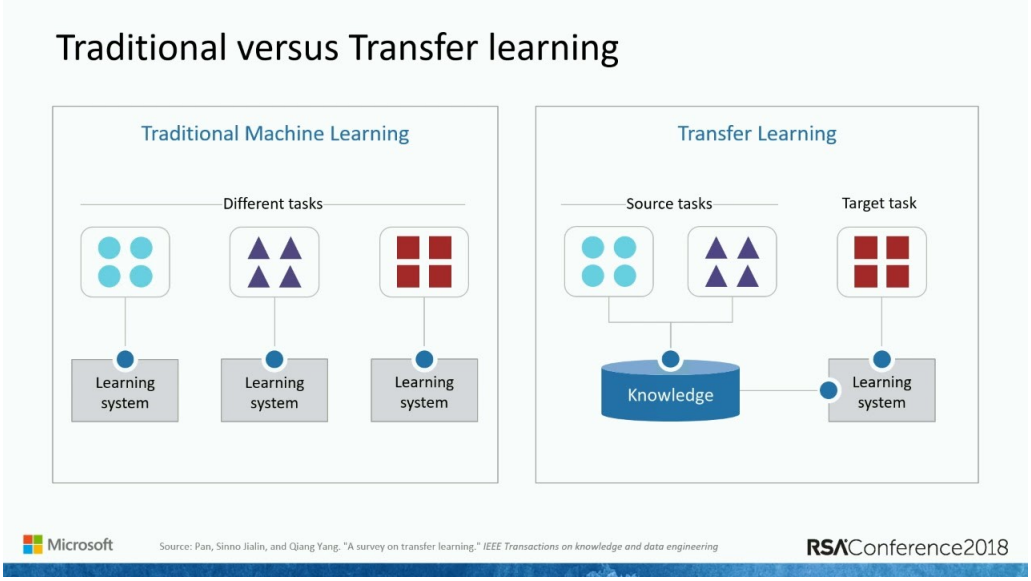


Figure 5: Transfer Learning [12]

As the image [12] describes, in traditional Machine Learning or Deep Learning, we perform different tasks in order to facilitate the learning system. On the other hand, a range of tasks are imported as knowledge in Transfer Learning and only the targeted task is performed in the learning system in addition to the knowledge.

4.1.2 Transfer Learning on PyTorch

PyTorch, like Tensorflow, supports the use of Transfer Learning to train Deep Learning models. A trained model can be saved or pretrained model can be loaded on PyTorch by using a dictionary of saved states (model.load_state_dict(best_model_wts)) which essentially includes the weights and the biases in the model. Another way of saving or loading can be by saving/loading the whole .pb file (model file) which contains the whole model. This model can be easily reused on a new system with limited or no knowledge about the model. The pretrained model can also be used to fine tune the parameters to achieve best results. Following image [9] shows a snippet of the code that loads a pretrained model and resets finally fully connected layer.

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

Figure 6: Transfer Learning for ResNet18 on PyTorch [9]

4.1.3 DataLoaders

One big advantage of Pytorch is its support for DataLoaders that help taking care of loading and running tests on very large, memory consuming, datasets in a quick and efficient way. Dataloaders are an efficient data generation scheme that generate the data on multiple cores in real time and feed it right away to the deep learning model [7]. The image [2] below further illustrates the working of data loader.

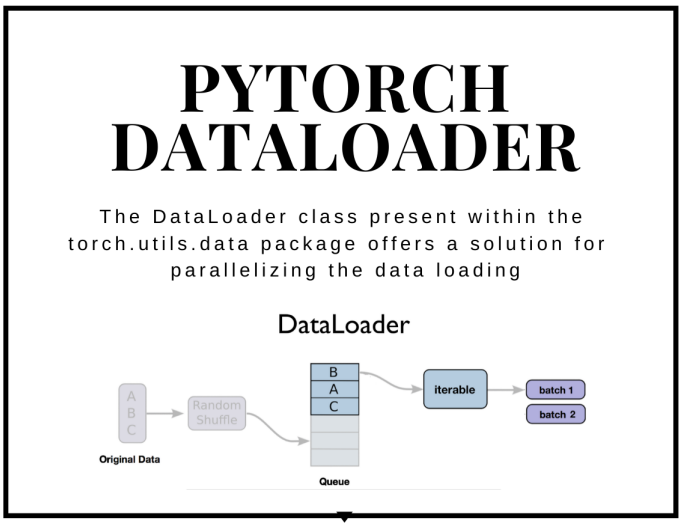


Figure 7: DataLoaders in PyTorch [2]

4.1.4 ResNet50

ResNet50 is a residual learning framework that is used to ease the training of networks that are substantially deep that the ones used previously [4]. This framework is trained and evaluated on the ImageNet dataset with a depth of upto 152 layers, achieving an error of 3.57% on the ImageNet

test data. Owing to its advantages, we decided to use the ResNet50 framework in our model. We added two more layers on top of the ResNet50 pretrained model, namely, Average Pooling layer and linear layer to map our outputs to the 20 labels as required by the question. A snippet of the code used in the source file is attached below:

```
class ResNet50(nn.Module):
    def __init__(self,num_outputs):
        super(ResNet50,self).__init__()
        self.resnet = resnet_cls
        layer4 = self.resnet.layer4
        self.resnet.layer4 = nn.Sequential(
                                    nn.Dropout(0.5),
                                    layer4
                                    )
        self.resnet.avgpool = AvgPool()
        self.resnet.fc = nn.Linear(2048, num_outputs)
        for param in self.resnet.parameters():
            param.requires_grad = False

        for param in self.resnet.layer4.parameters():
            param.requires_grad = True

        for param in self.resnet.fc.parameters():
            param.requires_grad = True
```

Figure 8: ResNet50 [4]

4.2 Justification of Reason-ability

Since we are working on a large dataset of 30000 training images and 10000 testing images all labelled into one or more classes, an efficient training and evaluation of this dataset was very time consuming and the RAM or GPU could crash due to memory overload. Also, running such a large dataset again and again for parameter tuning would have proved out to be redundant. For this reason, Transfer Learning is used to reduce the time of training our own model and using its weights to further test the dataset. A ResNet50 pretrained model is used which has a great depth and very less error. Pytorch was preferred as a library for training and evaluating the dataset owing to the constraints of the assignment. Additionally, PyTorch offered various benefits in batching of dataset in order to reduce memory overload and run the model without causing crashes.

4.3 Advantages and Novelty

Transfer Learning has various benefits over traditional learning techniques:

- **Improved Baseline Performance**: Since we are using an already trained model and augmenting it with the knowledge of an isolated learner, there is a high possibility of improved performance of the baseline due to the knowledge transfer.
- **Model Development Time**: Since the model is already trained on a similar dataset, it will be much easier to learn the target task as compared to learning from scratch, thus helping in reducing the overall running time of the training and evaluation functions.
- **Improved Final Performance**: Making the best use of transfer learning and adding new layers to the existing pretrained model, better results can be easily achieved, thus resulting in higher performance.

The below graph [10] depicts the advantages with better baseline performance (higher start), efficiency gains (higher slope), and better final performance (higher asymptote):

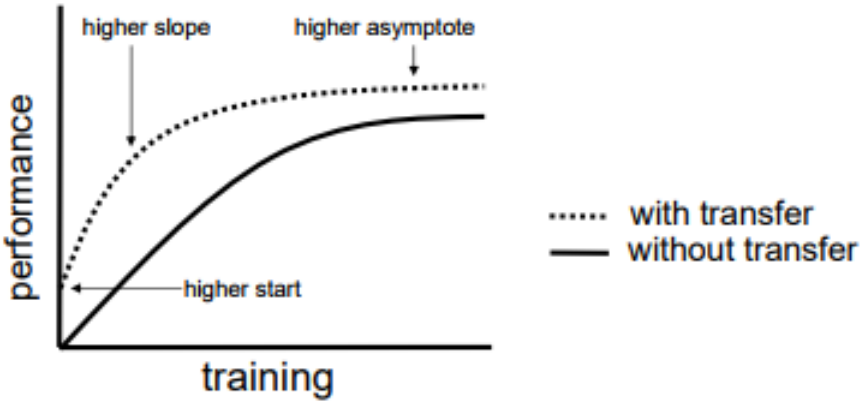


Figure 9: Transfer Learning: Use Vs Non-Use [10]

PyTorch was used as a preferred choice of library owing to its benefits such as support of DataLoaders, that allow easy batch-ing of dataset, making it easier and faster to train and evaluate without and memory overload. Also, compared to TensorFlow, PyTorch is much easier to debug, hence making it easy for the programmer to detect and solve issues in the code. While TensorFlow uses static graphs for computations, PyTorch uses dynamic computation graphs.

ResNet50 was used due to its many advantages such as the depth of the network (which is much more than other available pretrained models like VGG16) and its low error outputs. An ensemble of the ResNet50 achieves just 3.57% error on the ImageNet test set. This result won the first place on the ILSVRC 2015 classification task [10].

Novelty in our approach was achieved by adding more layers to the pre-trained ResNet50 dataset. The two layers that were added were the Average Pooling layer and the linear layer to map our outputs to the 20 labels as required by the question. Average Pooling was performed in order to reduce variance and reduce the computation complexity of the ResNet model, and extract the low level features from the neighborhood.

4.4 Modules Used in the development

VGG19 Resnet50 ResNext Tensorflow PyTorch Rescaling Images Encoding labels: One Hot Encoder Loss Function Final Output Layer Neurons Final Activation Function PyTorch DataLoader PyTorch Transformers Adam Optimizer The Fit_Generator Random Oversampler (Class balancing for Tensorflow)

5 EXPERIMENTATION AND RESULTS

The experimentation for the given image classification task has been performed using transfer learning on two frameworks: **Tensorflow and PyTorch**. However, for the purposes of this project report, we will be discussing the algorithm and approach submitted as our final submission i.e. **applying transfer learning on ResNet50 using PyTorch framework.**

5.1 Experiment setting

5.1.1 Outline

The dataset for the classification task has been provided by the lecturer. Some base code from lab and official PyTorch documentation has been used. [9][13]

The experiment builds a custom neural network using existing neural network frameworks and trains some parameters of the network using our dataset. The output layer possesses 20 neurons with each returning a probability of the data instance falling in a certain class. Finally, predictions are generated using a threshold percentile value (here, 90%) that allows the function to pick two label classes for each image.

5.1.2 Performance Metrics

The key performance metrics used to measure the performance of our neural network are:

1. **Precision:**
   Precision talks about how precise/accurate your model is. Out of those predicted positive, how many of them are actual positive[14]. Mathematically:

$$Precision = \frac{True\ Positive}{True\ Positive\ +\ False\ Positive}$$

2. **Recall:**
   Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive) [14]. Mathematically:

$$Recall = \frac{True\ Positive}{True\ Positive\ +\ False\ Negative}$$

3. **F1-Score:**
   F1 Score is needed when you want to seek a balance between Precision and Recall[14]. Mathematically:

$$F1Score = 2 * \frac{Precision\ *\ Recall}{Precision\ +\ Recall}$$

5.1.3 Experiment steps

The steps performed within our project are as follows:

1. First, the data in image folder was divided to form the test and train sets (10,000 and 30,000 respectively).

2. Using PyTorch Custom DataLoader Classes, the train images were divided into training and validation sets (24,000 and 6,000) respectively. And imported into individual dataloaders.

3. The neural network model was built using ResNet50 model present in the torch.models library. Based on the existing model, custom class for ResNet50 was built that modified the network for the purpose of our application.

4. Output dimensions for last layer were changed to 20 (number of classes), Loss Function was changed to 'BCEWithLogitsLoss.'

5. Using our pre-trained weights (from previous experiment iterations), the weights were loaded into the base model and a neural network object was created.

6. Using this object, we can further run more training iterations (and keep saving the progress) or simply generate required metrics and predictions for the test data.

5.2 Experiment Observations

For preparation of the dataset to create validation and training data:

```
Train Features: torch.Size([128, 3, 224, 224])
Train Labels: torch.Size([128, 20])

Validation Features: torch.Size([128, 3, 224, 224])
Validation Labels: torch.Size([128, 20])
```

Figure 10: DataLoader tensors created for features and labels

```
TRAINING
training examples:  24000
batch size:  128
batches available:  188

VALIDATION
validation examples:  6000
batch size:  128
batches available:  47
```

Figure 11: Sampling of training and validation data

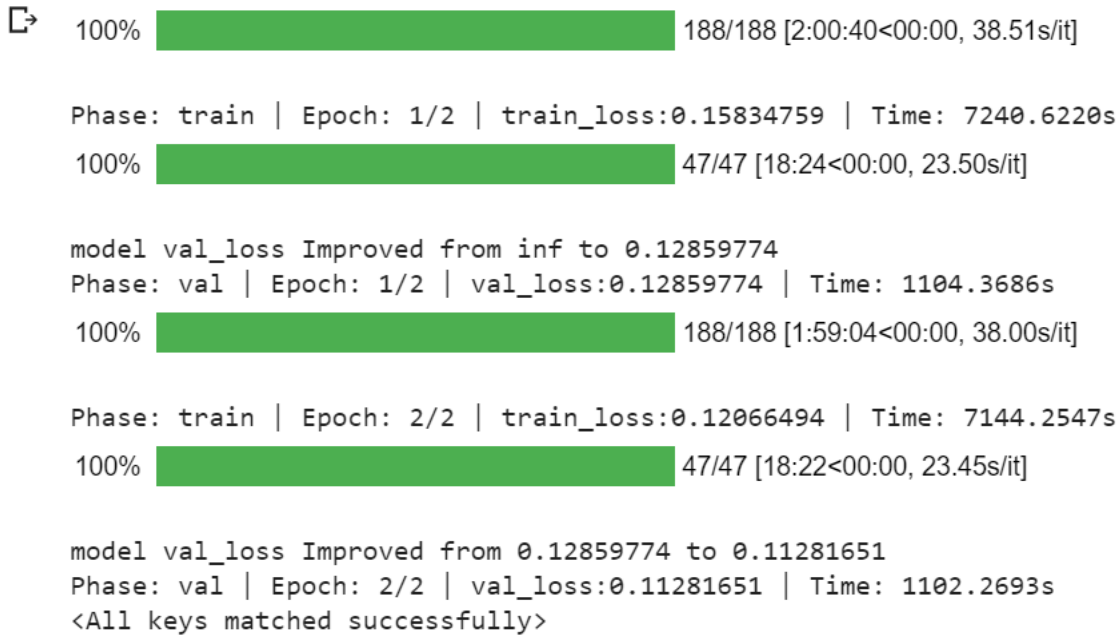


Figure 12: Training time taken by the algorithm (to create the saved model file)

PARAMETER	THRESHOLD PERCENTILE			
	80	85	90	95
PRECISION	0.33	0.41	0.55	0.86
RECALL	0.9	0.86	0.8	0.68
F1-SCORE	0.46	0.53	0.62	0.73
TIME TAKEN FOR PREDICTION	35 minutes	35 minutes	35 minutes	35 minutes

Figure 13: Experiment Observations on final Custom ResNet50

5.3 Analysis of results observed

Based on the above observations, we can see that our algorithm has still a lot of scope for improvement. Since we have trained our model on only two epochs (2.5hours/epoch) owing to time and resource constraints, there are still many things left for performance improvement.

On analyzing the available results, we can see that the threshold percentile variation impacts the performance metrics greatly. This is because we have defined the percentile function in a manner that picks two classes for each image when the percentile is set to 90. When decreased, the labels picked keep increasing to 3,4 and so on. Furthermore, we can also see that varying the labels assigned to validation images results in increasing one of the two parameters: precision or recall while simultaneously decreasing the other parameter. This can be seen across all observations. Thus, a metric for measuring appropriate performance wold be the mean F1-score that allows a balance of the two variables.

6 DISCUSSION

6.1 Final choice of algorithm and parameters

Based on the above observations, resource constraints (time and computational) and justification of choices, the final network chosen is: *Custom ResNet50 using PyTorch Framework*

6.2 Limitations of our Experiment

As we dived into performing this multi-label image classification task, we were faced with many hurdles. The first of which was grasping the concept of creating a multi-label classifier and consequent implementation of the same. After thorough research and study, we were able to successfully understand how to change the required parameters of a single-label classifier net to a multi-label one.
Furthermore, we mistakenly took the route of using Tensorflow to start with. Since some of us had previous experience with tensorflow, we first began testing the experiment on the same. However, it proved to be unfruitful and the models were often crashing (since we were unable to find a fix for batching the images to the model). Once we solved the issue of memory crashing, we realized that TF was still taking a very very long time in training epochs and consequently, we gave up on the approach.
When we switched to PyTorch, we came across the DataLoader concepts that could overcome the issue we were facing in TF and thus, we began exploring the same. However, due to our own personal constraints (assignments, exams etc. for other subjects as well), we were unable to devote much dedicated time to this task.
On PyTorch we attempted the ResNet50 and ResNext models for training. However, we were only successful in running the ResNet model completely. No time was left to solve the errors we were facing in ResNext. Consequently, we gave up on the model (because of our deadline) and hope to carry forward our attempts after this submission.

6.3 Personal Reflection

We believe there is a large room for improvement in our model solution. The algorithm takes a long training time per epoch. Future work would entail the analysis of why this happens and how it can be reduced. Furthermore, we can also increase the training epochs of our model to better the performance.

Another option for future work is the inclusion of captions in the training data. While we researched the topic of encoding text features (using NLTK encoders, TFIDF, Google Universal Sentence Encoder etc.), due to constraints on time and resources we were unable to build and implement the same. Next step for our group would include feature-izing the labels as well; for input parameters to the network.

Based on the level of execution we were able to implement, we believe we have developed keen interest and founding skills for developing neural networks on PyTorch. We have also recognized grassroots differences between the PyTorch Framework and the Tensorflow framework. Furthermore, during the course of the project, we have also gained important skills of handling our data in different manners (The DataLoader Eureka!) and building a deep neural application.

7 CONCLUSION

To conclude the report, for the immediate purposes of our task, our neural network performs reasonably well. And we hope to carry on further work on the same to build better performance.

As discussed in detail under limitations and personal reflection, further work is required in enhancing the performance of the neural network and future work will be carried on.

8 BIBLIOGRAPHY

References

[1] End-to-end machine learning framework. https://pytorch.org/features/, Last accessed on 2020-05-25.

[2] Python dataloader. https://cdn.journaldev.com/wp-content/uploads/2020/02/PyTorch-Data-Loader.png, Last accessed on 2020-05-25.

[3] Pytorch documentation. https://pytorch.org/docs/stable/index.html, Last accessed on 2020-05-25.

[4] Resnet-50. https://www.kaggle.com/keras/resnet50, Last accessed on 2020-05-25.

[5] Transfer learning. https://cs231n.github.io/transfer-learning/, Last accessed on 2020-05-25.

[6] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[7] Afshine Amidi and Shervine Amidi. A detailed example of how to generate your data in parallel with pytorch. https://stanford.edu/ shervine/blog/pytorch-how-to-generate-data-parallel, Last accessed on 2020-05-25.

[8] Jason Brownlee. How to improve performance with transfer learning for deep learning neural networks, 2019. https://machinelearningmastery.com/how-to-improve-performance-with-transfer-learning-for-deep-learning-neural-networks/, Last accessed on 2020-05-25.

[9] Sasank Chilamkurthy. Transfer learning for computer vision tutorial. https://pytorch.org/tutorials/beginner/transfer$_l earning_t utorial.html training - the - model, Last accessed on 2020 - 05 - 25$.

[10] Tamoghna Ghosh Dipanjan Sarkar, Raghav Bali. Advantages of transfer learning. https://www.oreilly.com/library/view/hands-on-transfer-learning/9781788831307/ed5e9ab9-0003-4234-8ab2-9ea1b03c76be.xhtml, Last accessed on 2020-05-25.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[12] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[13] PyTorch Documentation. Developing custom pytorch dataloaders. https://pytorch.org/tutorials/recipes/recipes/custom$_d ataset_t ransforms_l oader.html?highlight = dataloader, Last accessed on 2020 - 05 - 25$.

[14] Koo Ping Shung. Accuracy, precision, recall or f1? https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9, Last accessed on 2020-05-25.

[15] Devin Soni. Dealing with imbalanced classes in machine learning, 2018. https://towardsdatascience.com/dealing-with-imbalanced-classes-in-machine-learning-d43d6fa19d2, Last accessed on 2020-05-25.

[16] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

9 APPENDIX

9.1 Hardware and Software specifications

The code was implemented using Google Colabratory Open Source Computing Platform. Thus, resources provided by Google were used. This comprised of their standard CPU configuration with a RAM space of 12 GB provided.

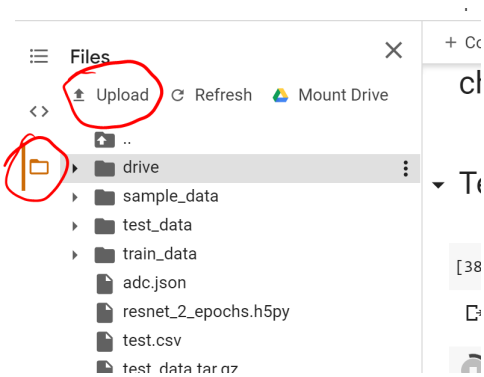
However, we used a laptop for running the same online, which has below features:

- Processor: Intel i7
- OS: Windows 10
- RAM: 8 GB
- GPU: 4 GB
- Model: HP Pavilion 360

9.2 Instructions for running the code

Follow the below instructions to run the ipynb file attached in the submission:

1. Open https://colab.research.google.com/ and open the submitted .ipynb file.
2. Upload "train.csv", "test.csv" and upload the pretrained weights file (resnet_2_epochs.h5py) attached in the submission, on google colab's local directory.



3. Once the files are uploaded, run all cells in the notebook. The training and testing images dataset is stored as a compressed file in google drive. Google colab will extract it itself, you need to give authentication to use your google account. For best results, store the files in your google drive only and enter its directory.

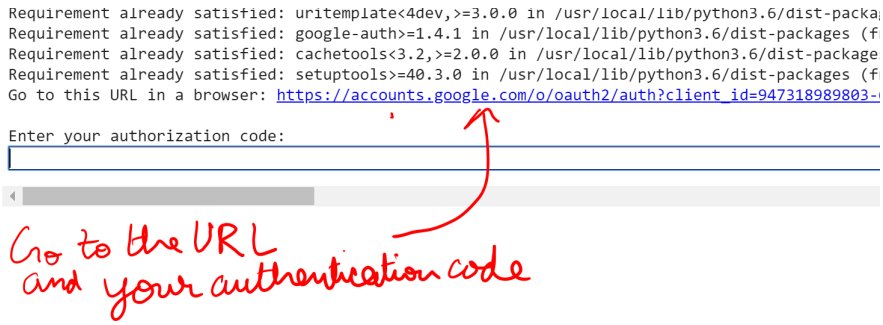

```
# Test
#https://drive.google.com/open?id=1wzOJPY7FBZoEHyWMAd89cL9cAYwH7nP4
# Train
#https://drive.google.com/open?id=15zC2Qx243_TnCZA8-Aw5kN-7MwOzNghc
# Labels
# https://drive.google.com/open?id=1cM_BusmHKqEla-c7LKh6L7kxCi-HGjxM

download = drive.CreateFile({'id': '1wzOJPY7FBZoEHyWMAd89cL9cAYwH7nP4'})
download.GetContentFile('train_data.tar.gz')

download = drive.CreateFile({'id': '1Mj-0XmAe6SrstwG7ixOJVoiX6yg-udOX'})
download.GetContentFile('test_data.tar.gz')

download = drive.CreateFile({'id': '1cM_BusmHKqEla-c7LKh6L7kxCi-HGjxM'})
download.GetContentFile('train.csv')

!tar -xvf train_data.tar.gz
!tar -xvf test_data.tar.gz
```



4. Once everything is imported, all the cells will run as expected and output will be displayed.

5. The final test label file (.csv) will be downloaded in google colab's local directory, right click on that and click download to save in your local device.