Comparison of Multiple Neural Network Architectures on Historical Landmark Recognition Tasks

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Siddhant Goel
Spring, 2020

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

COMPARISON OF MULTIPLE NEURAL NETWORK ARCHITECTURES ON HISTORICAL LANDMARK RECOGNITION TASKS

A PREPRINT

Siddhant Goel
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
sg2nq@virginia.edu

May 7, 2020

ABSTRACT

The University of Virginia is a historical landmark, and as such there are several architectural marvels from the 19th Century till the 21st Century. Several successful neural networks exist, and they have associated benefits and disadvantages to them. In this project, we attempt to create a software that can label several of the buildings on the grounds of UVA using the machine learning technique transfer learning. We benchmark the results of transfer learning on 5 different models, ResNet 50, Inception v3, MobileNet v2, Xception, and DenseNet 121, pre-trained on ImageNet to classify 18 buildings, using two libraries: PyTorch and TensorFlow.

Keywords UVA Landmarks · Architecture · Transfer Learning · PyTorch · TensorFlow · ResNet 50 · Inception v3 · MobileNet v2 · Xception · DenseNet 121

1 Motivation

Seeing tourists and new students on grounds anyone can tell that there is a clear gap in the knowledge of different buildings on grounds. While tours inform visitors of several different buildings, they are guided at the tour guide's pace and do not provide visitors the freedom to take their time in different places of interest. First year students, on the other hand, often get lost looking for their class rooms, and often tend to learn nothing about the buildings except their names. We wanted to fill this gap in the knowledge of students and visitors by giving them the ease of simply taking photos on their phone and having them predicted nearly instantaneously, thus allowing them to learn more about the buildings they are interested in. The problem is challenging as, unlike common Computer Vision projects, the buildings at UVA have a lot of common features, and very little variation in colors and appearances. Given the novel dataset, we wanted to test the abilities of transfer learning by picking 5 different and marvel neural networks, which were at some point in the past decade considered state of the art. A comparison of the results of two of the most popular libraries in Machine Learning, PyTorch and TensorFlow, was also of interest to us as researchers in the field. The technique of transfer learning was chosen to analyse and compare the abilities of the different neural networks.

2 Dataset

The novel dataset was collected with the help of students on grounds at UVA. Eric Stein created an application that allowed users to click and upload photos to our firebase database, which was then curated and used to train and validate the machine learning model. The dataset contained 20 different classes, namely (with number of images): Academical Village (408), Alderman Library (742), Alumni Hall (457), Aquatic Fitness Center (556), Bararo Hall (200), Brooks Hall (410), Clark Hall (844), Madison Hall (368), Minor Hall (760), New Cabell Hall (636), Newcomb Hall (716), Old

Cabell Hall (832), Olsson Hall (995), Rice Hall (1460), Rotunda (1569), Scott Stadium (1257), Thornton Hall (1219), University Chapel (857). A total of 14329 images with varying sizes were cut down to 300x300 as the largest image size required is 299x299 for Inception v3, and this allows us to use more data by reducing the load on google colab, as the file size decreased from 50GB to 392MB. Link to Dataset:

`https://firebasestorage.googleapis.com/v0/b/uva-landmark-images.appspot.com/o/dataset.zip?alt=media`

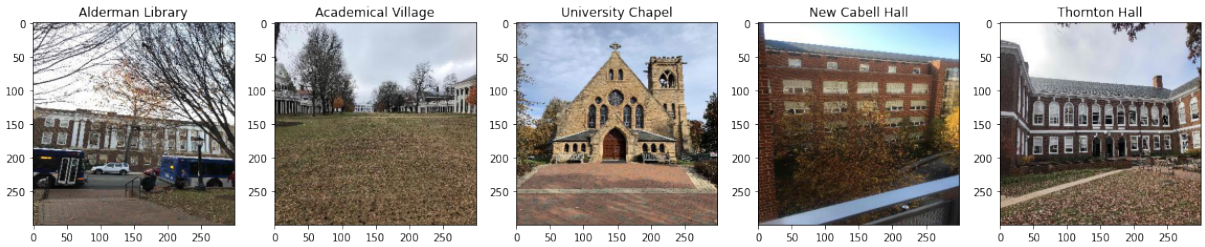

Figure 1: A set of sample images in the dataset

3 Methods & Experiment

iPython, using libraries PyTorch and TensorFlow, on Google Colab was used to perform this experiment. GPU provided by Google Colab was used to increase the speed of training. In PyTorch, random.shuffle and SubsetRandomSampler were used to import the data from the dataset in a shuffled manner. Data was transformed using torchvision.transforms in a standard manner, cropping to the center to match the input for the pre-trained neural networks (299x299 for Inception v3, 224x224 otherwise) and normalized as per the expectations of the models. Learning rate of 0.01 with a momentum of 0.9 was used in the stochastic gradient descent optimizer. A custom function was written to train and test models, reporting the training and validation accuracy and loss for each epoch, as well as the total time taken to complete training. 7 epochs were used to train each of the models.

In TensorFlow, ImageDataGenerator.flow_from_directory was used to import the data from the dataset. Data was transformed using preprocess_input functions provided by each of the models in keras.applications. Base models were imported from the model of choice, one Global Average Pooling 2D and one fully connected Dense layer with softmax activation were added to re-purpose the model for our use. First, the base model was left locked, and only the final layers were trained with learning rate of 0.1 and a momentum of 0.9 in the stochastic gradient descent optimizer. Second, the entire model was unlocked was made trainable and retrained with the stochastic gradient optimizer with a 0.01 learning rate and 0.9 momentum. 10 epochs were used to train each of the models in both steps.

The hyper parameters were chosen because they are considered the standard, and are thus used as the default in both languages. In order to prevent a bias in the research, they were not tuned for either library, and were used consistently to yield easily reproducible results in research. This prevents a bias between not only the libraries, but also the pretrained models that were used, as tuning hyper parameters might be easier in one model than in another.

Hyper Parameters and Settings

Setting	PyTorch	TensorFlow Step 1	TensorFlow Step 2
Learning Rate	0.01	0.1	0.01
Momentum	0.9%	0.9%	0.9%
Number of epochs	7	10	10

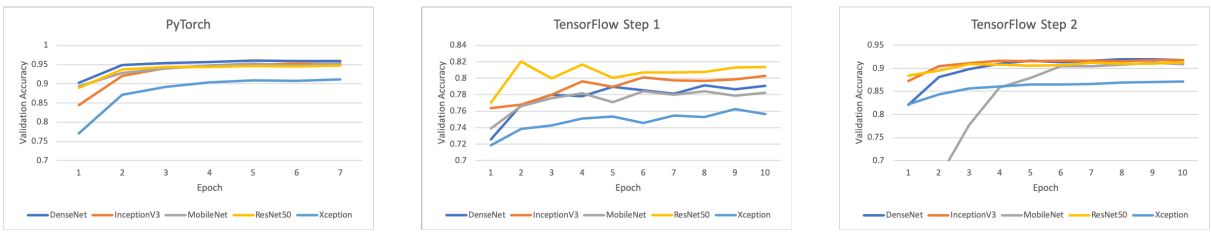

Figure 2: Diminishing Returns

The figures above show that the returns of continuous training diminish after the 5th step in Pytorch and 8th step in both parts of Tensorflow. Thus, 7 and 10+10 epochs were chosen to retrain the networks for the purposes of this project. The models were chosen because they are unique and important for the different inventions they brought about. ResNet was the first model to introduce skip connections(He et al.). ResNet50 reformulates the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. Inception brought about the changes to the inception techniques first used in LeNet to reach state of the art results (Szegedy et al.). This presented a way of scaling up networks in an efficient manner, so as to minimize computational costs, while improving results. MobileNets are some of the most streamlined and efficient models that achieve satisfactory results, and tradeoff accuracy against latency (Sandler et al). MobileNetV2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer. DenseNet networks used several skip connections, such as those in ResNet, in parallel (Huang et al). DenseNet 121 showed that shorter connections between layers close to the input and those close to the output lead to improved results. Xception presented a novel way of using inception modules (Chollet). Xception uses inception modules as the intermediate step in between regular convolution and depth-wise convolution. This was then used to create depthwise separable convolutions are used to replace Inception modules.
Finally, a flask server was created to host the machine learning model on Google Cloud Platform's Kubernetes.

4 Results

All tests below were conducted using 7 epochs in PyTorch and 10+10 in TensorFlow. Top-1 accuracy was used as the metric.

Resnet 50

Metric	PyTorch	TensorFlow
Best Validation Loss	0.2297	0.5484
Best Validation Accuracy	94.71%	91.22%
Total Training Time	32m 3s	59m 42s

Inception v3

Metric	PyTorch	TensorFlow
Best Validation Loss	0.2242	0.4437
Best Validation Accuracy	95.20%	91.78%
Total Training Time	14m 14s	47m 20s

MobileNet v2

Metric	PyTorch	TensorFlow
Best Validation Loss	0.2137	0.5139
Best Validation Accuracy	95.34%	91.27%
Total Training Time	8m 49s	16m 44s

Xception

Metric	PyTorch	TensorFlow
Best Validation Loss	0.4013	0.5223
Best Validation Accuracy	91.14%	87.11%
Total Training Time	13m 35s	32m 24s

DenseNet 121

Metric	PyTorch	TensorFlow
Best Validation Loss	0.1763	0.6116
Best Validation Accuracy	96.04%	91.99%
Total Training Time	16m 24s	30m 51s

5 Discussion

Densenet 121 has the best overall accuracy, with a valiation accuracy of 96.04% for PyTorch. This observation aligns with TensorFlow, where DenseNet 121 is also the best model, at 91.99%. The remaining 4 models in both cases are closely behind DenseNet 121. Given a trade-off between time and performance, MobileNet v2 might seem like the best model going forward, as it would make predictions faster and give relatively good results, due to a less complex model. It would also be easier to retrain than using more complex models which take more time with diminishing returns.
It is evident from this research that the models train very similarly across the libraries and to achieve satisfactory results. Models perform consistently across the two libraries. Transfer learning is a valuable tool for image classification problems and can achieve optimal results with minimal work. The hyper parameters, which might yield better results upon adjustment, work well with the default values of 0.001 for learning rate and 0.9 for momentum. Stochastic gradient descent as an optimizer and categorical cross entropy as the loss function yield desirable results in transfer learning in this problem.
Our original concern was that it might be difficult to retrain models trained on ImageNet, which contains a wide variety of categories that are extremely different, such as a car and a dog, to differentiating between buildings, which are relatively similar barring some features such as color, shape and size of bricks and the way they are laid out. However, our tests have successfully demonstrated that this is not an issue, and that models trained on ImageNet can be retrained to different types of computer vision problems and achieve high accuracy.
Docker files and Kubernetes serve as a useful method to deploy machine learning models to production, by simply creating Flask servers in Python that use the trained neural networks to predict the results and output them to the user. A web service, hosted on Google Cloud Kubernetes, is used in this project to display results to the users of the app.

6 Conclusions

Data collection through crowd-sourcing is an effective way of obtaining clean data, given the co-operation of the crowd. The data achieved was important in moving the project forward and demonstrated the ability of crowd-sourced data. The experiments demonstrate that transfer learning is a viable machine learning for the computer vision problem of landmark recognition. Both simple and complex models are capable of achieving good results using transfer learning, but there is a trade-off between time and resources, and accuracy. The results for top-1 accuracy demonstrate that the machine learning model created in this project can serve as a viable product for landmark classification at the University of Virginia. The application created to make this project available to users is available on both the Google Play Store and the Apple App Store, with the name UVA Landmark Recognition, published by Eric Stein.

References

[1] Chollet, François. “Xception: Deep Learning with Depthwise Separable Convolutions.” ArXiv:1610.02357 [Cs], Apr. 2017. arXiv.org, http://arxiv.org/abs/1610.02357.

[2] He, Kaiming, et al. “Deep Residual Learning for Image Recognition.” ArXiv:1512.03385 [Cs], Dec. 2015. arXiv.org, http://arxiv.org/abs/1512.03385.

[3] Huang, Gao, et al. “Densely Connected Convolutional Networks.” ArXiv:1608.06993 [Cs], Jan. 2018. arXiv.org, http://arxiv.org/abs/1608.06993.

[4] Sandler, Mark, et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” ArXiv:1801.04381 [Cs], Mar. 2019. arXiv.org, http://arxiv.org/abs/1801.04381.

[5] Szegedy, Christian, et al. “Rethinking the Inception Architecture for Computer Vision.” ArXiv:1512.00567 [Cs], Dec. 2015. arXiv.org, http://arxiv.org/abs/1512.00567.