ML1819 Research Assignment 1

Team 1

Task 101: How consistent are the implementations of machine learning algorithms in different ML libraries?

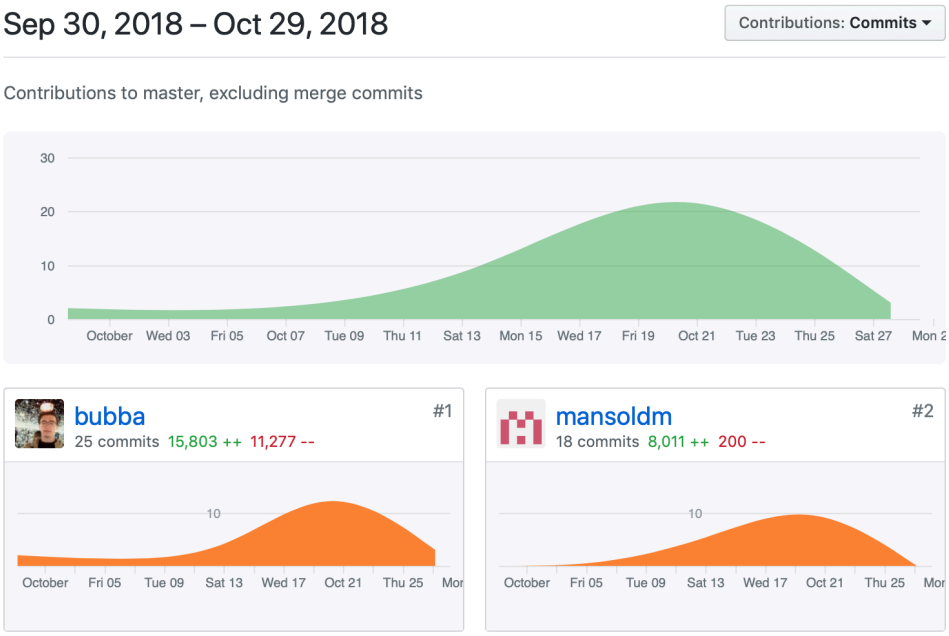
Luke Lau 15336810
Martino Mansoldo 15325057
Derrick Afrifa 15325022

Contributions

Luke Lau: TensorFlow model and training, paper writeup
Martino Mansoldo: PyTorch training, paper writeup
Derrick Afrifa: Paper writeup and research

Word count: 976

https://github.com/bubba/cs4404-project-1
https://github.com/bubba/cs4404-project-1/commits/master



CycleGAN Two Ways

A Comparison of Unpaired Image-to-Image Translation Performance with Cycle-Consistent Adversarial Networks implemented with different Machine Learning frameworks

Derrick Amponsa Afrifa
Trinity College, University of Dublin
Dublin, Republic of Ireland
afrifad@tcd.ie

Luke Lau
Trinity College, University of Dublin
Dublin, Republic of Ireland
laulu@tcd.ie

Martino Mansoldo
Trinity College, University of Dublin
Dublin, Republic of Ireland
mansoldm@tcd.ie

ABSTRACT

Although many Machine Learning libraries and frameworks implement the same set of algorithms, the manner in which they are implemented differs from library to library. One rarely finds objectively superior implementations. However, a number of key metrics can be used to evaluate the strengths and shortcomings of dissimilar approaches. This paper attempts to outline the differences in the results of the implementations of CycleGAN[10] in two popular frameworks for Machine Learning. Models are trained using the same data sets and key aspects of their performance are delineated. It is expected that an understanding of the nature of the differences will reveal new optimisation avenues.

KEYWORDS

machine learning, neural networks, generative adversarial networks, tensorflow, pytorch

1 INTRODUCTION

Generative Adversarial Networks (GANs) are a type of neural network, based around the architecture of a generator and discriminator competing against each other. The generator tries to generate images from noise that are similar to the training data, whilst the discriminator tries to catch out the generator and distinguish the "fake" generated images from the real images. CycleGAN[10] builds upon this by having two generators that translate images between two categories, $G : X \rightarrow Y$ and $F : Y \rightarrow X$. Additionally, it adds a cycle-consistency loss, where images are converted over and back to their original category and the difference is minimised, so that $F(G(X)) \approx X$ holds.

PyTorch and TensorFlow are two popular Machine Learning frameworks that either have either implementations or APIs for CycleGAN. We aim to find what qualitative (not performance) differences exist when implementing identical CycleGAN model architectures side-by-side.

2 RELATED WORK

PyTorch began as the Python implementation of Torch, a Machine Learning framework for Torch, and provides a large amount of extensiblility for implementing deep architectures[2]. It builds up a computation graph dynamically as the program is run. In contrast, TensorFlow[1] builds up its computational graph before training and evaluating. This graph describes the flow of tensors throughout the network, hence the etymology. Benchmarks of Torch and TensorFlow[9] have shown that neither framework has a particular advantage in training time on GPUs.

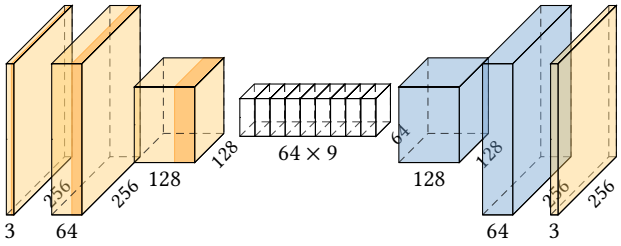


Figure 1: Generator architecture from left to right. The centre layer is the residual layer.

3 METHODOLOGY

We implemented the PyTorch's model as faithfully as possible in TensorFlow. They were then both trained on a dataset of images of Yosemite National Park, CA, US taken in both Summer and Winter, from the original paper as well.

The *generator* architecture, shown in figure 1, is made up of three convolution layers, an inner ResNet[4] and three deconvolution layers. The convolution and deconvolution layers have ReLU activations. The inner ResNet is made up of nine residual blocks, which sum the input to the block with the two convolutions of it, and help the network 'retain' information.

At a high level, the idea is that the downscaling convolution layers extract the features of the image, the ResNet applies the transformation $A \rightarrow B$ to this information, and the upscaling deconvolution layers then generate the image from the transformed information.

3.1 PyTorch

There are currently no implementations of GANs built into the PyTorch framework. The original implementation that accompanied the paper was in Torch, however the authors also provided an implementation for PyTorch. For this paper we used the PyTorch implementation with the default arguments and hyper-parameters.

The *discriminator* is a ConvNet that consists of 4 layers with Leaky ReLU activations and one layer without.

As per the definition of CycleGAN, two generators and two discriminators are used. Each network is optimised with Adam [5], a common extension of Stochastic Gradient Descent, using `torch.optim.Adam`.

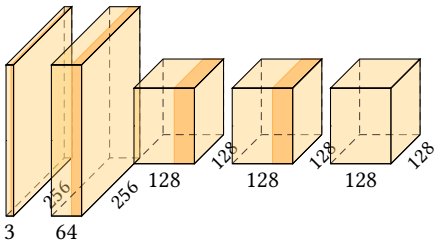


Figure 2: Discriminator architecture from left to right.

3.2 TensorFlow

TensorFlow provides a suite of tools for GANs in `tf.contrib.gan`. It has functions for creating, training and evaluating GANs, including both the original model as well as variants such as StarGAN[3], ACGAN[7] and of course, CycleGAN. For CycleGAN, TensorFlow provides `tf.contrib.gan.cyclegan_model`. The user of the API must supply the generator and discriminator, and TensorFlow will handle setting up the model to connect the two together.

The generator and discriminator models were created using the high-level `tf.contrib.layers` counterparts to the PyTorch implementation, including batch normalisation and reflective padding. The functions in `tf.contrib.gan` were left to their default parameters where possible, which matched those of the original paper.

The images needed to be preprocessed so that they had pixel values in the range $[-1, 1]$, and that they were all the same size. Additionally, there is reflective padding at the first, last and residual block convolution layers to prevent artefacts around image edges.

4 RESULTS AND DISCUSSION

Both networks had satisfactory results. Results for converting winter to summer are shown in figure 3. TensorFlow emphasised changing the color of the vegetation much more so than PyTorch, but both learnt to convert snow to lake water.

In figure 8, the individual losses of the generators and discriminator are very similar and do not have any major differences. Figure 5 shows the cycle-consistency loss (CCL) between the two different frameworks. At first glance PyTorch has a much larger loss, but when viewing the results qualitatively there was very little difference. The reason for the discrepancy in the CCLs is that TensorFlow[1] normalises the losses so that it is always between 1 and 0. This ensures that the CCL weight does not depend on the size of the images.

Otherwise, there is not much difference overall between the two implementations, which gives confidence for their reliability and stability.

5 LIMITATIONS

The model fails on images with snow in the immediate landscape, as it not always able to completely remove it. It also has difficulty converting images of landscapes outside of Yosemite and mountain ranges.

[1]See the source for more information

Figure 3: From left to right: Original winter, TensorFlow summer, PyTorch summer

Figure 4: From left to right: Original summer, TensorFlow winter, PyTorch winter

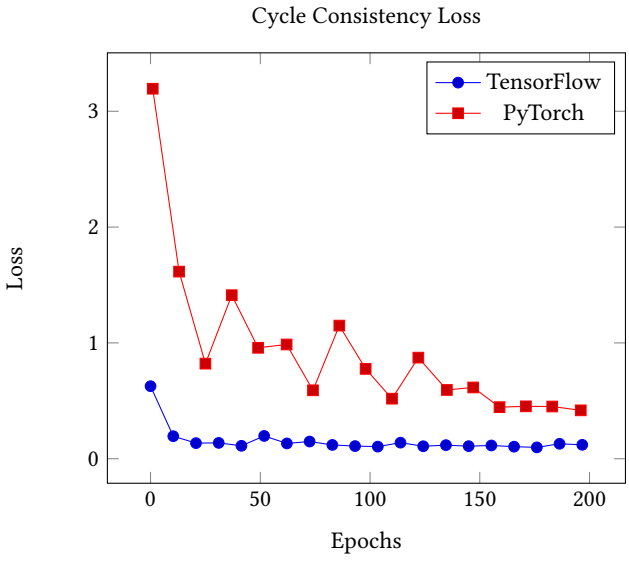


Figure 5: Cycle-Consistency Loss (Average of both generators)

(a) DSLR to phone, back to DSLR.

(b) Phone to DSLR, back to phone.

Figure 6: Images converted with CycleGAN to look like they were taken with either a phone or a DSLR. Bokeh gets added in the generated DSLR images, whilst it is stripped in the phone pictures.

We observed small, checkerboard artefacts in the output images. Odena et al.[6] describe the cause of these and methods to circumvent it.

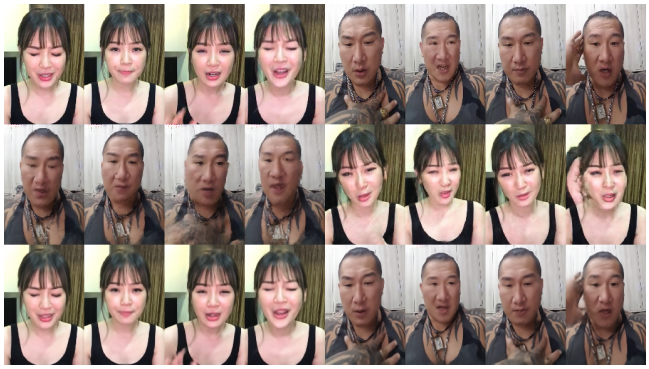
6 FURTHER WORK

6.1 Converting to DSLR

The TensorFlow model was also trained on a dataset of images of flowers, taken on phones and DSLRs. This dataset is also taken from the CycleGAN repository, and took around 20 hours to train. The results are shown in figure 6. The model learnt that DSLR photos typically have a bokeh effect, but unfortunately it also tries to swap out the colours. This may have been caused by one dataset having a particular set of colours more-so than the other, and we speculate this might be fixed by having a larger dataset to prevent overfitting. Interestingly enough though, the cycle-consistency loss also teaches the network how to reverse the colour swap almost perfectly.

6.2 Face-off facial recreation in Keras

This CycleGAN attempts to create a new face from an input face. The generated face mimics the facial expression of the input face. This architecture is implemented in Keras, a popular high level framework for machine learning. For the Generator a UNet[8] is used. For the discriminator the network is a convolutional network with Leaky ReLU as well as batch normalisation. An Adam[5] optimiser is employed in the training process. The link of the official authors' repository is: *https://github.com/tjwei/GANotebooks*



(a) From top to bottom: Input, Fake, Recreate of the input.

Figure 7: Face-off facial recreation

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*, Vol. 16. 265–283.

[2] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. 2015. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435* (2015).

[3] Yunjey Choi, Min-Je Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. 2017. StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation. *CoRR* abs/1711.09020 (2017). arXiv:1711.09020 http://arxiv.org/abs/1711.09020

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[5] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

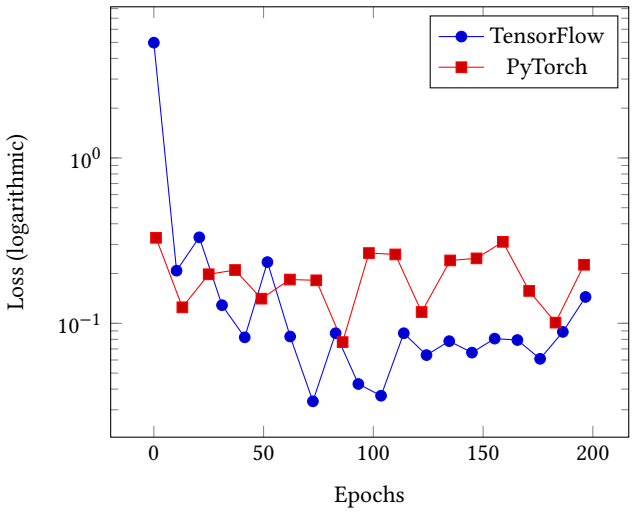
[6] Augustus Odena, Vincent Dumoulin, and Chris Olah. 2016. Deconvolution and Checkerboard Artifacts. *Distill* (2016). https://doi.org/10.23915/distill.00003

[7] A. Odena, C. Olah, and J. Shlens. 2016. Conditional Image Synthesis With Auxiliary Classifier GANs. *ArXiv e-prints* (Oct. 2016). arXiv:stat.ML/1610.09585

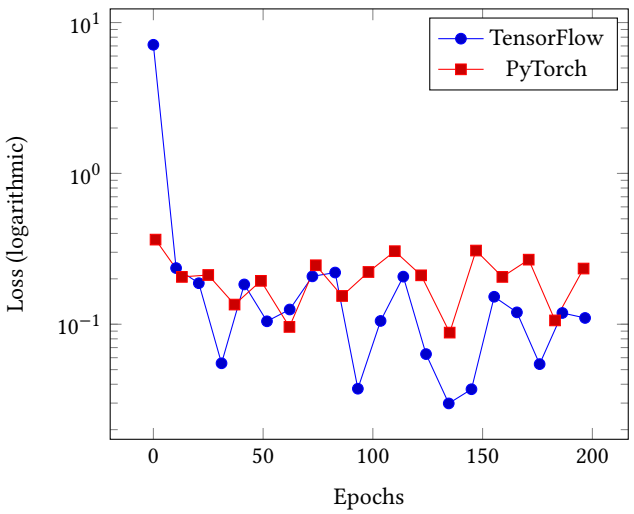
[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. *CoRR* abs/1505.04597 (2015). arXiv:1505.04597 http://arxiv.org/abs/1505.04597

[9] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking state-of-the-art deep learning software tools. In *Cloud Computing and Big Data (CCBD), 2016 7th International Conference on*. IEEE, 99–104.

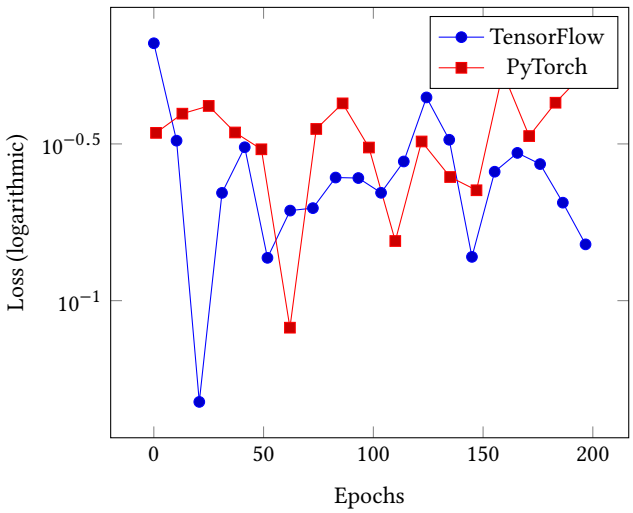
[10] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *CoRR* abs/1703.10593 (2017). arXiv:1703.10593 http://arxiv.org/abs/1703.10593



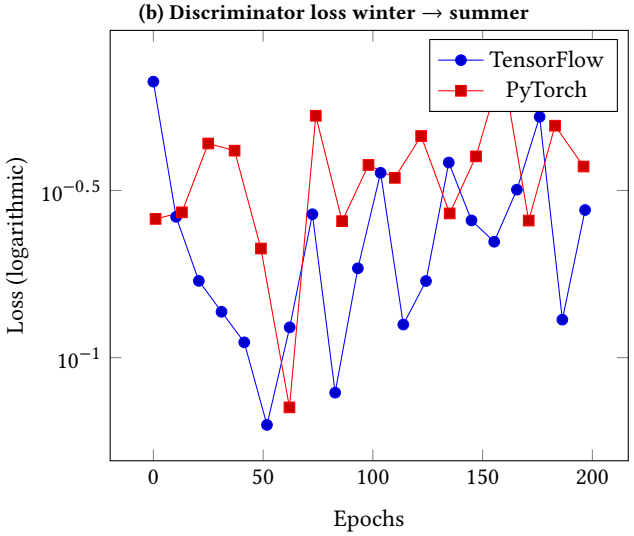
(a) Discriminator loss summer → winter



(b) Discriminator loss winter → summer



(c) Generator loss summer → winter



(d) Generator loss winter → summer

Figure 8: Individual losses of the generators and discriminators