Few-Shot Learning using Data Augmentation and Transfer Learning

Namit Mohale, Yashodhan Joshi, Anshum Pal
nm3191@nyu.edu, yj1400@nyu.edu, aap695@nyu.edu

1 Introduction

Adequate training of neural networks requires a lot of data. In the case of low data regime, which is the case with most of the problems today, the existing networks generalize poorly. To make progress on this foundational problem, we have decided to follow a low-shot learning technique by exploiting the method of aggressive data augmentation and transfer learning.

1.1 Related work

One-shot and low-shot learning. One class of approaches to one-shot learning uses generative models of appearance that tap into a global or a super-category level prior. Generative models based on strokes or parts have shown promise in restricted domains such as hand- written characters. They also work well in datasets without much intra-class variation or clutter.
Zero-shot learning. Zero-shot recognition uses textual or attribute-level descriptions of object classes to train classifiers. While this problem is different than ours, the motivation is the same: to reduce the amount of data required to learn classifiers. One line of work uses hand-designed attribute descriptions that are provided to the system for the novel categories.
Transfer learning. The ability to learn novel classes quickly is one of the main motivations for multitask and transfer learning. Thrun’s classic paper convincingly argues that “learning the n-th task should be easier than learning the first”, with ease referring to sample complexity. However, recent transfer learning research has mostly focused on the scenario where large amounts of training data are available for novel classes. For that situation, the efficacy of pre-trained ConvNets for extracting features is well known. There is also some analysis on what aspects of ImageNet training aid this transfer.[1]
Data Augmentation. The field of data augmentation is not new, and in fact, various data augmentation techniques have been applied to specific problems. The main techniques fall under the category of data warping, which is an approach which seeks to directly augment the input data to the model in data space. The idea can be traced back to augmentation performed on the MNIST set in [2].

A very generic and accepted current practice for augmenting image data is to perform geometric and color augmentations, such as reflecting the image, cropping and translating the image, and changing the color palette of the image. All of the transformation are affine transformation of the original image that take the form:

$$\mathbf{y} = \mathbf{W}\,\mathbf{x} + \mathbf{b}$$

The idea has been carried further in [3], where an error rate of 0.35% was achieved by generating new training samples using data augmentation techniques at each layer of a deep network. Specifically, digit data was augmented with elastic deformations, in addition to the typical affine transformation. Furthermore, data augmentation has found applicability in areas outside simply creating more data.

1.2 Problem, Goal, and Approach

Current recognition systems require days or even weeks of training on expensive hardware to develop good feature representations. The trained recognition systems may then be deployed as a service to be used by downstream applications. These downstream applications may need the ability to recognize new categories, but they may have neither the training data required, nor the infrastructure needed to retrain the models. Thus, there are two natural phases: in the first phase, we have the data and resources to train sophisticated feature extractors on large labeled datasets, and in the second phase, we want to add additional categories to our repertoire at minimal computational and data cost.

We are following the second phase. Our goal is to predict with high accuracy on a new class for which we have very few examples. Instead of collecting new data which would be both costly and time-consuming. Our approach is to use data augmentation and transfer learning to make more data for novel class, as well, as saving time by not training the whole model and using what the model already learned to extract basic features.

We are using aggressive data augmentation which uses various transformations like Coarse Dropout, Noise Embedding, Blur, Perspective Transformation, etc. to generate new examples, feeding to a pre-trained image-to-image GAN built on a U-Net architecture to generate even more samples and VGG16 model pre-trained on the **ImageNet1K** dataset for transfer learning.

2 Methods, Technical Depth and Innovation and Architecture

We are using two methods to make our Few-shot learning work. We apply Aggressive Data Augmentation which is followed by Transfer layer in which we Train last few layers of a pre-trained model on the Augmented data. Both Methods are used to boost the accuracy of the classification when there is low data. Combining these two approaches to boost more is our aim.

How do you decide what type of transfer learning you should perform on a new dataset? This is a function of several factors, but the two most important ones are the size of the new dataset (small or big), and its similarity to the original dataset (e.g. ImageNet-like in terms of the content of images and the classes, or very different, such as microscope images). Keeping in mind that ConvNet features are more generic in early layers and more original-dataset-specific in later layers, here are some common rules of thumb for navigating the 4 major scenarios:

1. *New dataset is small and similar to original dataset.* Since the data is small, it is not a good idea to fine-tune the ConvNet due to overfitting concerns. Since the data is similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes. We have some the class in our custom dataset that belongs to this type.

2. *New dataset is large and similar to the original dataset.* Since we have more data, we can have more confidence that we won't overfit if we were to try to fine-tune through the full network.

3. *New dataset is small but very different from the original dataset.* Since the data is small, it is likely best to only train a linear classifier. Since the dataset is very different, it might not be best to train the classifier form the top of the network, which contains more dataset-specific features. Instead, it might work better to train the SVM classifier from activations somewhere earlier in the network. We have some the class in our custom dataset that belongs to this type.

4. *New dataset is large and very different from the original dataset.* Since the dataset is very large, we may expect that we can afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. In this case, we would have enough data and confidence to fine-tune through the entire network.

Practical advice. There are a few additional things to keep in mind when performing Transfer Learning:

1. *Constraints from pretrained models*. Note that if you wish to use a pretrained network, you may be slightly constrained in terms of the architecture you can use for your new dataset. For example, you can't arbitrarily take out Conv layers from the pretrained network. However, some changes are straight-forward: Due to parameter sharing, you can easily run a pretrained network on images of different spatial size. This is clearly evident in the case of Conv/Pool layers because their forward function is independent of the input volume spatial size (as long as the strides "fit"). In case of FC layers, this still holds true because FC layers can be converted to a Convolutional Layer: For example, in an AlexNet, the final pooling volume before the first FC layer is of size [6x6x512]. Therefore, the FC layer looking at this volume is equivalent to having a Convolutional Layer that has receptive field size 6x6 and is applied with padding of 0.

2. *Learning rates*. It's common to use a smaller learning rate for ConvNet weights that are being fine-tuned, in comparison to the (randomly initialized) weights for the new linear classifier that computes the class scores of your new dataset. This is because we expect that the ConvNet weights are relatively good, so we don't wish to distort them too quickly and too much (especially while the new Linear Classifier above them is being trained from random initialization).

We would be considering both of these advices while building our model.

The trickiest part about traditional GANs is that they learn completely in an unsupervised manner. To add some factor of supervision, we use the U-Net Autoencoder type architecture for the Generator which maps the class features to noise so that image generation is class-driven and not random. The Discriminator follows a Patch-GAN type of architecture which discriminates every 30x30 region of the input image. The entire network is trained on ImageNet1k for 1000 epochs and the pre-trained network is then used to transfer the learning for the Novel classes.

2.1 Architecture and Design

Our model consists of VGG16 which is Pre-trained on ImageNet1K dataset. We would change the classifier and the layer that represent the bottleneck features.

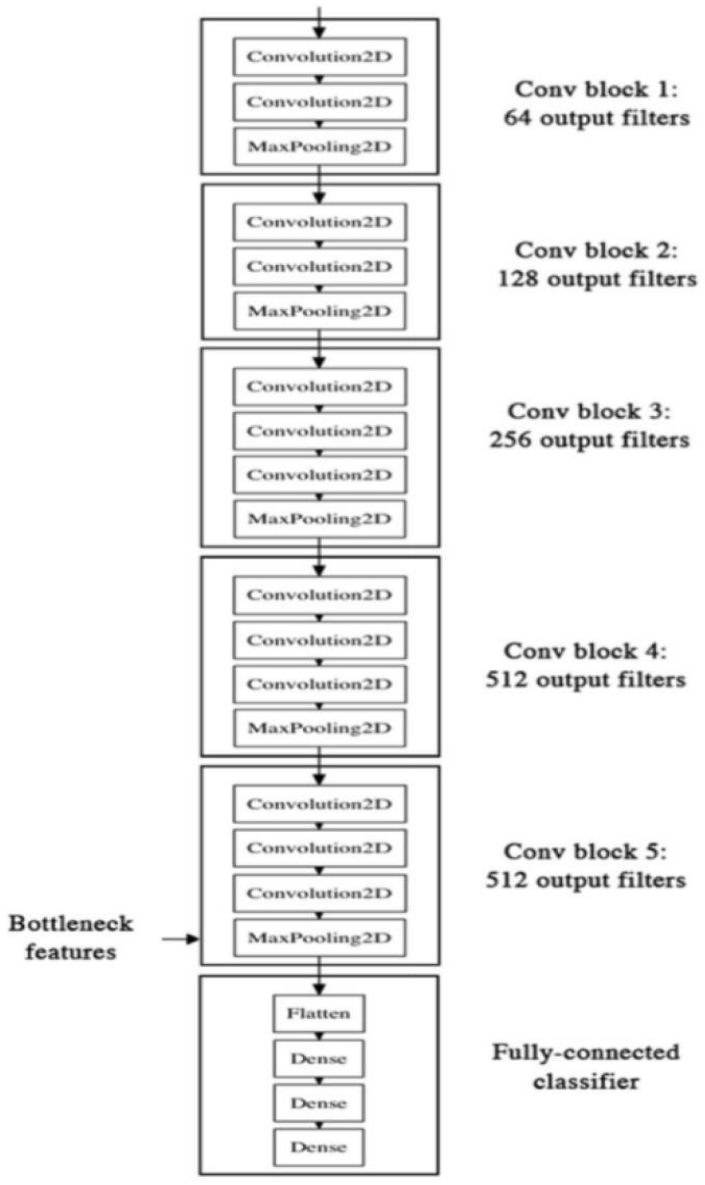


Figure 1: VGG16 Architecture

The Generator is built on the following architecture :

- The architecture of generator is a modified U-Net.
- Each block in the encoder is (Conv -> Batchnorm -> Leaky ReLU)
- Each block in the decoder is (Transposed Conv -> Batchnorm -> Dropout(applied to the first 3 blocks) -> ReLU)
- There are skip connections between the encoder and decoder (as in U-Net)

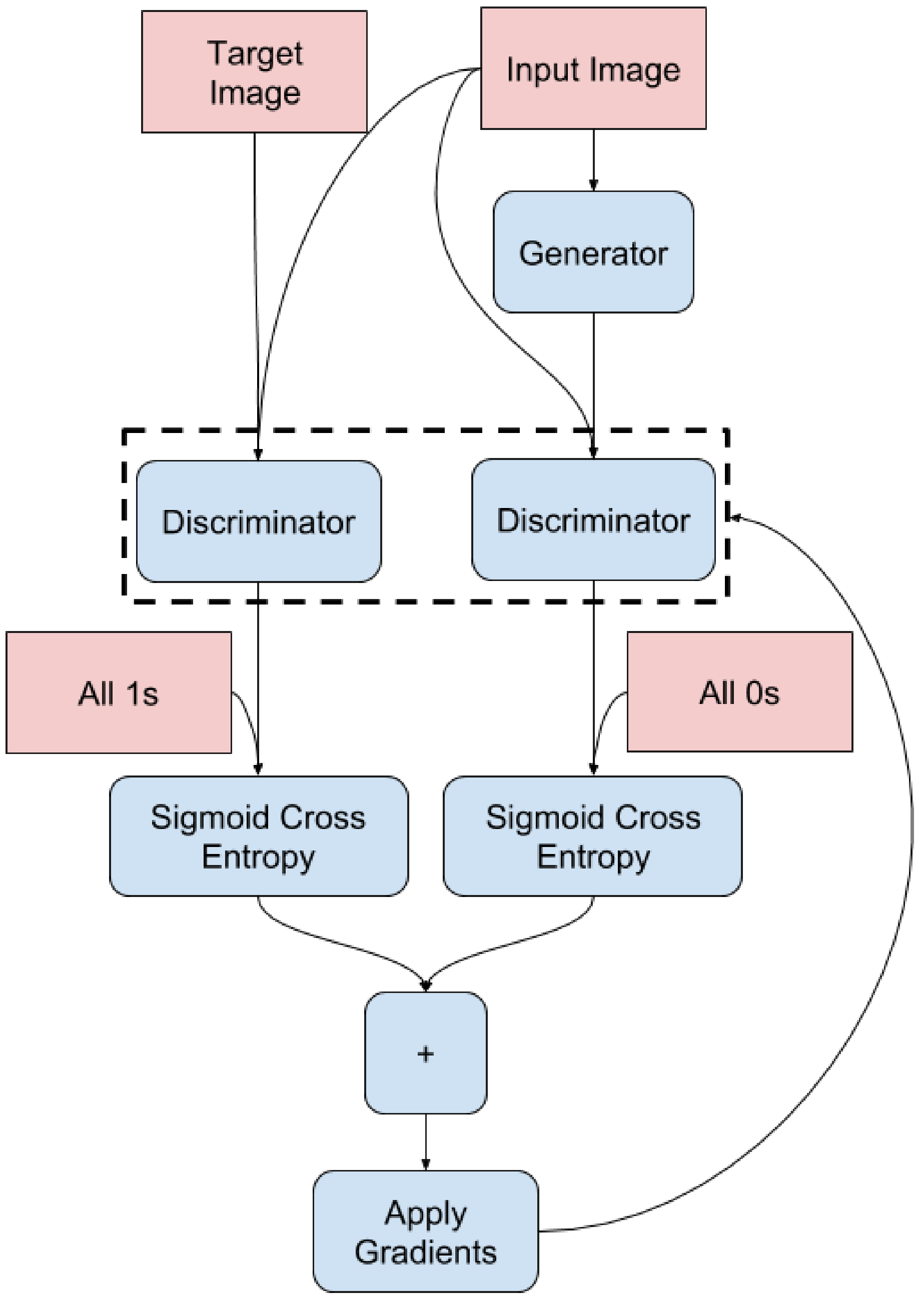
The Discriminator is built on the following architecture :

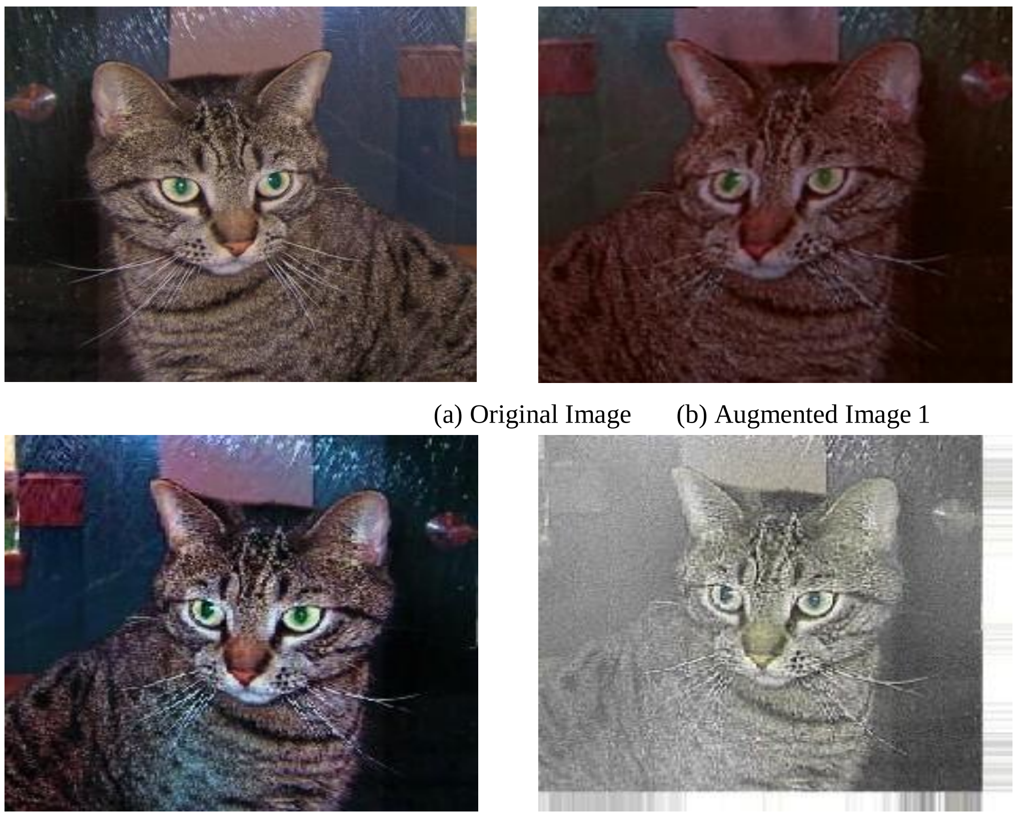
- The Discriminator is a PatchGAN.
- Each block in the discriminator is (Conv -> BatchNorm -> Leaky ReLU)
- The shape of the output after the last layer is (batch_size, 30, 30, 1)
- Each 30x30 patch of the output classifies a 70x70 portion of the input image (such an architecture is called a PatchGAN).
- Discriminator receives 2 inputs.
    - Input image and the target image, which it should classify as real.
    - Input image and the generated image (output of generator), which it should classify as fake.
    - We concatenate these 2 inputs together in the code (`tf.concat([inp, tar], axis=-1)`)

The input and the target images both belong to the same class. The aim is to minimize the distance between the generated image and both the input and the target image. This is made possible by inducing noise vector of 1x1x512 and concatenating it with the down-sampled 1x1x512 input image and up-sampling them with a U-Net to form a 256x256x3 image.

The architecture used is shown below. The credits for this image are:

https://www.tensorflow.org/alpha/tutorials/generative/pix2pix


Target Image
Input Image
Generator
Discriminator
Discriminator
All 1s
All 0s
Sigmoid Cross Entropy
Sigmoid Cross Entropy
+
Apply Gradients



(a) Original Image (b) Augmented Image 1

(c) Augmented Image 2 (d) Augmented Image 3

Figure 2: Original and Augmented Image of Cat Class

2.2 Data Set and Evaluation

We are using a custom data set that would consist 5 classes namely (”Human”, ”Dog”, ”Cat”, ”Horse”, ”Robot”) . We have chosen these classes as to see how transfer learning perform on the class that the model knows how to extract feature for like (”Dog”, ”Cat”, ”Human”) and how would it perform on classes they are not even seen by the Pre-trained model like (”Robot”, ”Horse”).

To break down the final performance metrics, we report the average accuracy separately on the test samples from the novel classes that the model has seen in some scene in the past like aforementioned (”Dog”, ”Cat”, ”Human”*)* and on all Novel classes.

2.3 Results

We have used 1,5,10 and 20 Original Examples of different classes separately. Then Created 10, 100, and 1000 Augmented data images for each class with mediumly-aggressive augmentation approach and then trained only the classifier layer of VGG16 pre-trained on ImageNet1k dataset.

Hyperparameters used:

Epoch: 12.
Loss: categorical_crossentropy.
Optimizer: rmsprop.
1010 Images in total from different classes were to make the test data, and 162 samples are used for validation.

The results are as follows:

In these tables, each row accounts for the number of original examples used for each class and the columns account for the number of augmented images created using all the original samples for each class. The accuracy results are obtained after combining the augmented images with the original samples and then training on the network.

-	10	100	1000
1	77.4752%	83.2921%	80.0743%
5	89.6040%	87.1287%	88.9109%
10	90.6931%	91.9802%	92.3762%
20	-	91.9802%	91.9802%

Table 1: Accuracy on Test data which include both Novel and Base classes.

-	10	100	1000
1	78.2178%	84.9010%	75.4950%
5	87.8713%	81.6832%	83.6832%
10	87.6238%	86.6337%	87.1287%
20	-	87.6238%	87.8713%

Table 2: Accuracy on Test data which only include Novel classes.

Unfreezing the last 3 layers: Attempting to realize the best training method, we also tried unfreezing the last three layers of the network and then training the data on it. The results we achieved with 1 original example is as follows:

-	10	100	1000
1	16.8317%	19.5545%	19.0594%

Table 3: Accuracy on Test data by unfreezing last 3 layers of VGG16.

The result are bad and they can be explained as the data is small. Since data is small, it is not a good idea to fine-tune the ConvNet due to overfitting concerns. Since the data is somewhat similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes.

2.4 Result Analysis

We can clearly see in the table-1 and table-2 above and in the graphs below that the trend in accuracies for the novel classes alone lie close to base+Novel classes which clearly reflects that the VGG-16 model is learning the features of novel classes after training the last layer only.

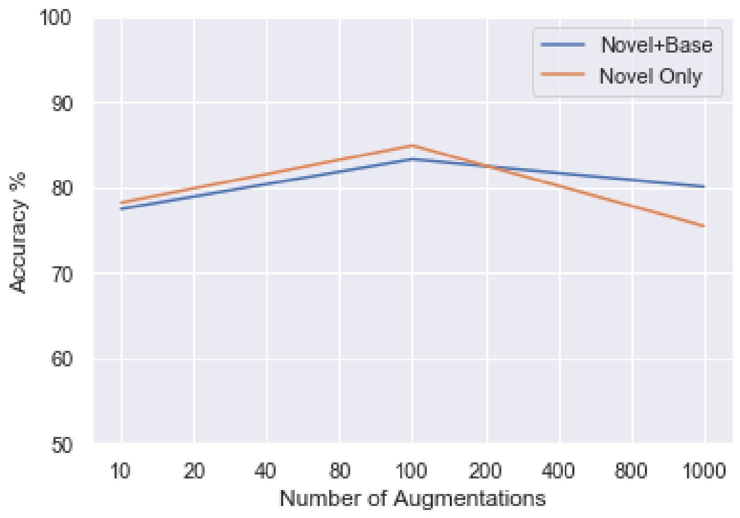


Figure 3: Accuracy vs Number of Augmented Image from 1 Image

Here we see that the accuracy for 1000 augmentation falls. We propose that the reason behind this is overfitting of the data as only one original image was used to create so many augmentations that the model ended up learning more features of that one original image instead of generalizing the entire class.

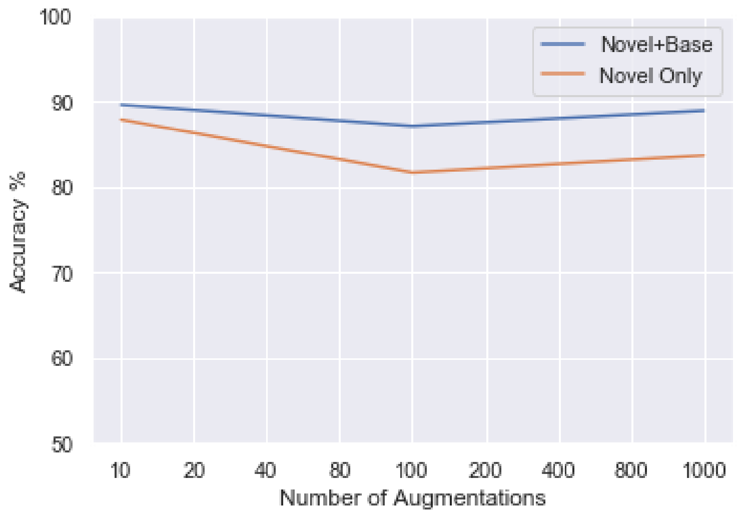


Figure 4: Accuracy vs Number of Augmented Image from 5 Image

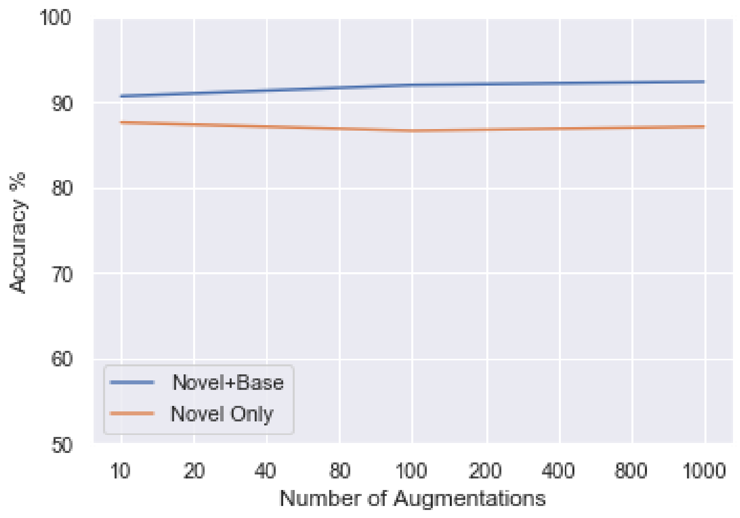


Figure 5: Accuracy vs Number of Augmented Image from 10 Image

We believe that the minor shifts in accuracies sometimes are because of randomness of the network and training process as the shift is 1

3 Conclusion/Future Work

Data augmentation has been shown to produce promising ways to increase the accuracy of classification tasks. While traditional augmentation is very effective alone, other techniques enabled by CycleGAN and other similar networks are promising. We experimented with our own way of combining training images allowing a neural net to learn augmentations that best improve the ability to correctly classify images. If given more time, we would like to explore more complex architecture and more varied datasets. Finally, although GANs and neural augmentations do not perform much better than traditional augmentations and consume almost 3x the compute time or more, we can always combine data augmentation techniques. Perhaps a combination of traditional augmentation followed by neural augmentation further improves classification strength.

Given the plethora of data, we would expect that such data augmentation techniques might be used to benefit not only classification tasks lacking sufficient data, but also help improve the current state of the art algorithms for classification. Furthermore, the work can be applicable in more generic ways, as ”style” transfer can be used to augment data in situations were the available data set is unbalanced. For example, it would be interesting to see if reinforcement learning techniques could benefit from similar data augmentation approaches. We would also like to explore the applicability of this technique to videos. Specifically, it is a well known challenge to collect video data in different conditions (night, rain, fog) which can be used to train selfdriving vehicles. However, these are the exact situations under which safety is the most critical. Can our style transfer method be applied to daytime videos so we can generate night time driving conditions? Can this improve safety? If such methods are successful, then we can greatly reduce the difficulty of collecting sufficient data and replace them with augmentation techniques, which by comparison are much more simpler.
Code URL[1]

[1]**Github Repo:** https://github.com/ksw25/Deep-Learning/tree/master/Deep %20learning • %20project

References

[1] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[2] H. S. Baird. Document image analysis. chapter Document Image Defect Models, pages 315–325. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[3] L. M. Gambardella D. C. Ciresan, U. Meier and J. Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. CoRR, abs/1003.0358, 2010.