Learnable Augmentation – Expanding on the "Smart Augmentation" Technique

Swati.Deshpande (19233301)
School of Computer Science
National University of Ireland, Galway

Supervisor
Dr. Michael.Schukat

In partial fulfillment of the requirements for the degree of
MSc in Computer Science (Artificial Intelligence)

August 28, 2020

DECLARATION

I, Swati Deshpande, do hereby declare that this thesis entitled LEARNABLE AUGMENTATION – EXPANDING ON THE "SMART AUGMENTATION" TECHNIQUE is a bonafide record of research work done by me for the award of MSc in Computer Science (Artificial Intelligence) from National University of Ireland, Galway. It has not been previously submitted, in part or whole, to any university or institution for any degree, diploma, or other qualification.

Signature: Swati.Deshpande, August 2020

Abstract

Data Augmentation is a process of increasing the trainable dataset by not collecting new data, but by using some augmentation techniques like Random Cropping, Mirroring, horizontal flipping etc. This technique is useful because it increases the existing dataset which helps in training the neural network and also lets the network generalize well on the data. Data Augmentation is said to increase the diversity of the dataset. However, data augmentation remains a time consuming task having to rely on intuition, trial and error and often luck. We would want an algorithm to augment data itself with little or no human input. One such algorithm is called "SMART AUGMENTATION" proposed in a paper published in early 2017 "Smart Augmentation Learning an Optimal Data Augmentation Strategy" Lemley et al. (2017). Smart Augmentation is a technique where one artificial neural network learns to classify while the other neural network simultaneously learns to augment data for the same task. Neural Network A learns the best augmentation strategy during training Neural Network B. This works in a way where Network A learns to generate data and the Network B learns to classify. Using the loss of Network B, Network A gets better and generates augmentations which are of use to the classifier (Network B). This technique overcomes certain problems such as over fitting and failing to generalize. The focus of this research is on expanding the smart augmentation technique mentioned in the paper by re-implementing it using a modern Deep Learning framework such as "PYTORCH", and verifying performance of the algorithm with different datasets. The performance is measured by the Loss and Accuracy of the model.

Keywords: Data Augmentation, Artificial Neural Network, Binary Classification, Smart Augmentation, Artificial Intelligence, Deep Learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the motivation behind this research and its purpose. It talks about pytorch and why implementing the algorithm on same is important and to evaluate its performance. Also, includes some of the problems occurring in Deep Neural Networks and the intention behind using algorithms like smart augmentation and briefly introduces the smart augmentation algorithm. In this research, a study is carried out on the Smart Augmentation technique and implementation of the same on modern Deep Learning framework (PYTORCH). Testing of this implementation on Datasets with and without Smart Augmentation to compare the performance of the technique is done.

1.1 Motivation

Pytorch is mainly used for computer vision applications as well as deep learning models. It is free and open source machine learning library released under BSD license. It was developed by Facebook's AI Research lab (FAIR) and primarily built on python which makes it so friendly for running many deep learning models. It has two important features, one is it makes use of tensor computing which plays a very important role in the development of any Neural Network model on pytorch. Another is it uses a optimized tensor library using GPU's which makes it faster than other frameworks such as Tensorflow. Why pytorch? To answer this question we look at some advantages as it is built on python and to build any Deep Learning/ Machine Learning model on python is very easy and the same ease is also seen in using pytorch. Also, it is very easy to learn and it provides a very high level productivity as it provides a simple but powerful API. It is very easy to code in pytorch and learn the basics quickly. It is also very good at debugging as it has a built in debugger and it can be set up by simply calling pdb.set_trace(). It has a feature called Data Parallelism which can distribute data into multiple

GPU's and is very useful in parallel processing of the data. It includes many useful libraries such as, GPyTorch which is an efficient implementation with GPU's and it is used in pytorch and many deep neural networks can be implemented. It has an optimization library called BoTorch and AllenNLP an open source library for NLP research. It has a Python API which is very useful in building neural networks by just calling the functions. The basic building block of the networks is the torch.nn library which consists of all layers that can be used to build the models to name a few are the convolution layers, containers and many more.

Trainable data in a deep learning problem is considered to be one of the most important aspects and good data is very important for learning the model as insufficient data can lead to poor results. Artificial Neural Networks always require huge amounts of training data to give accurate results. The amount of data depends on the complexity of the model as well. As the architecture of the model increases the data requirement becomes more. Especially in supervised learning problems, getting high-quality labeled data is very difficult. Data Augmentation is one such approach where small datasets could be expanded using augmentation techniques like Mirroring, Adding Gaussian Noise, Random Cropping, Blur + Flip, and Rotation. These techniques help in increasing the dataset and can solve the lack of data problem. However, this stands to be a trial and error approch which lacks accuracy.

Another important issue that has been found recently in Deep Neural Networks is that they can be fooled easily. There are many examples of a DNN being fooled as explained by Nguyen et al. (2015). It was said that changing an image of a class either by adding extra pixels, patches or reducing some pixels, DNN can interpret it to be something else entirely. Some images of patterns which researchers don't know, the Networks classified them to be of a certain class with almost 99.99% accuracy. Another example was a Neural Network that learned a stop sign which was modified by adding patches as speed limit of 40. Su et al. (2019) showed that a DNN can be fooled even with the simplest of tempering. Here, they show a single pixel change in a image can fool the Neural Network into giving incorrect predictions with high conficence.

In addition to above, another problem in DNN is "Overfitting". Overfitting occurs when the model stops learning the new features in the data but starts memorizing the data itself. This will result in poor generalization of data i.e. the model performs very well on train data but does not perform well on unseen data. It cannot give good predictions on the data it has not seen, this is the sign that the model has overfitted on the train data. It can be known that the model is overfitting when after some epochs the train loss reduces and the train accuracy keeps increasing where as the validation loss increases and validation accuracy keeps decreasing. There are many reasons for this to occur; one of the major causes is that the data is inadequate or if the data has same type of images and does not include diverse situations like images in different lighting, color etc.

Smart Augmentation was proposed to tackle the above problems to some extent. In this technique two Artificial Neural Networks are used which work in adversarial way. Both Neural networks try to improve their performance by reducing the loss of the other network. This will help in generating samples which can help model generalize on unseen data. The goal of Learnable Augmentation is that it tries to learn an optimal data augmentation strategy to improve the performance of the model.

1.2 Purpose

In this thesis, we discuss the performance of the Smart Augmentation technique with different diverse conditioned datasets and compare it with standard non-Smart Augmentation Approach and find out the advantages and disadvantages of using the Smart Augmentation approach. The results of this experiment are useful to know how efficient would be the technique across other models. Both, the training and evaluation of the model is carried out with all the datasets that are mentioned later.
Aim of the thesis is to study the efficiency of Smart Augmentation technique implemented on pytorch and investigate the further expansion of the algorithm to have better results.

1.3 Scope

The algorithm is implemented on pytorch which is an open source machine learning library, it uses tensors to store the arrays. Aim is to train and evaluate the implemented Smart Augmnetation algorithm on pytorch. Training and evaluation is conducted on GPU with 2 x 1080Ti (64 GB RAM) and Intel i7-7700K (4 core / 8 thread). Training and Evaluation is done on 3 different datasets. Experimental results are shown as the change in loss and the best validation accuracy seen in 1000 epochs. The learned combination of the random images of the class are also shown.

1.4 Thesis Structure

The next chapter serves the intention of introducing the reader with concepts of "Data Augmentation" and "Smart Augmentation". It gives an understanding on the Generative Adversarial Networks (GAN) and how smart augmentation can be used as a GAN. Chapter 3 and Chapter 4 give a brief outline on the works related around the research topic and introduce the data used for the experiments respectively. In addition,Chapter 5 will describe in detail, the model implemented in thesis and Chapter 6 discusses the experimental settings. Finally the results are evaluated in Chapter 7 along with the conclusion in Chapter 8.

Chapter 2

Background

This chapter gives in detail the background of research topic. It includes the important concepts which are required to understand the model. These concepts include the basic data augmentation approach which is the base of the approach. Then, the smart augmentation approach is explained in detail along with the figures to understand the algorithm. Finally, the Generative Adversarial systems are explained to show other existing algorithms that work like the smart augmentation approach. Conditional GAN's are also included as it behaves the way smart augmentation technique works and intention of these models can be considered similar.

2.1 Data Augmentation

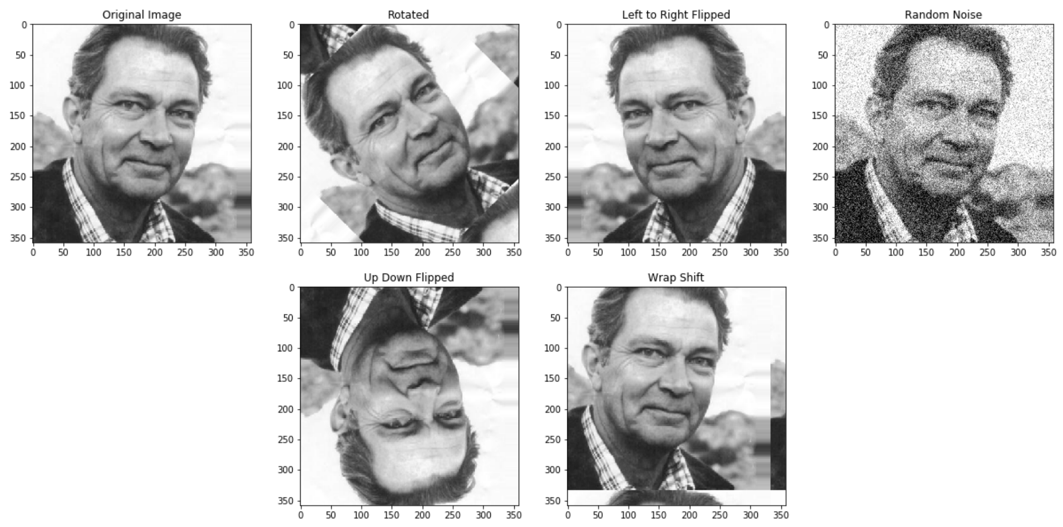


Figure 2.1: Basic Data Augmentation techniques

Data Augmentation is a regularization technique where the data is artificially modified by using many different variations. It is considered as a strategy to increase the dataset and diversity as well. Some of these augmentation techniques are Rotate, Flip, Adding random noise, shift etc. Figure 2.1 shows an example of some basic augmentation techniques. Data Augmentation is very helpful when it comes to training neural networks with small datasets. It will increase the dataset without adding new data but includes different augmentation conditions which will train the network not to overfit on the small dataset. It will try to ensure that the network will learn good features from the dataset. Augmentation not only servers the purpose of increasing the dataset, but also helps train neural network for changing data. For example, when an image is shifted in various different positions we will be training the network to not be effected by the shifting of the objects in the images and learn the correct features. Two types of augmentation methods are specified Lemley et al. (2017), one is unsupervised augmentation (regardless of the label of sample) and another is supervised augmentation (label of data is used). If Augmentation is used, it must be applied to all the classes of the dataset. However, with data augmentation new data is not added to the dataset, we can still achieve better results with this technique and model can be trained with less data. The aim of data augmentation is to prevent the model form overfitting. Data Augmentation can be applied in two ways, it can either be used as a pre-processing step before we begin the training of our model or it can be used during the training of the model in real-time. As a pre-processing step it is usually used to increase the dataset, when the trainable data is small. In other way data is not expanded but different augmentations are applied in each epoch while training, this is said to increase the generalizing capacity of the model.

2.2 Smart Augmentation

Smart Augmentation is the process when an artificial neural network learns the best augmentation strategy during training of the model itself. This falls under the real-time augmentation task, where the data is augmented during training. Here, there are two neural networks where network A aims to learn new data of a certain class. Network A receives an input image from the same class, the output of network A is another image which is very close to the input image it received. Network A compares this image with the input and the loss is used to inform Network A. However, Network A can't learn on its own which is why another neural network is added Network B. Task of Network B is to differentiate between the two classes. Network A learns to create a sample which is a combination of two or more data samples from the same class, this merged sample is used to train Network B to classify. The loss from network B is sent back to network A, this will help network A to learn such augmentation tricks which can help increase the classifying accuracy of network B. Hence, network B is called the “Classifier” and network A is called “Augmenter”. Network A generates such images which belong to the same class and which will have the features of two or more samples from the class which will help increase the performance of Classifier. The Loss (LA) from network A is used to improve the sample generating capacity and Loss (LB) from network B is used to improve the weights for the model. Total Loss of the model will be a sum of the losses of both network A and network B. Below Figure 2.2 shows the overall architecture of Smart Augmentation model.

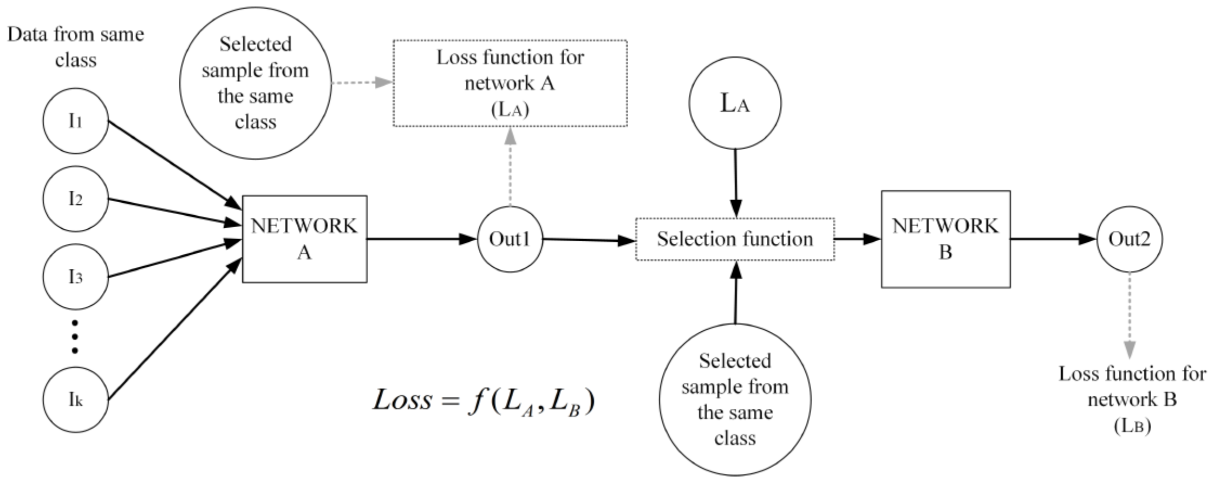

Data from same class
Selected sample from the same class
Loss function for network A (LA)
LA
I1
I2
I3
Ik
NETWORK A
Out1
Selection function
NETWORK B
Out2
Selected sample from the same class
Loss function for network B (LB)
$Loss = f(L_A, L_B)$

Figure 2.2: Overview of Smart Augmentation [Lemley et al. (2017)]

Two parameters alpha and beta are added to the losses of network A and network B, we can see how it helps increase the accuracy of network B.The images generated by Network A will be a learned combination of three random images from the same class. Smart Augmentation has shown to decrease overfitting, increase accuracy, and the generalizing capacity of the model.

2.3 Generative Adversarial Networks (GANs)

Generative Adversarial Nets was introduced by Goodfellow et al. (2014) where two neural networks were trained simultaneously. One was called the generative model (G) and other was discriminative model (D). Generator (G) learns to generate samples and the goal is to maximize the generating capacity where as Discriminator (D) learns to minimize making the mistake as it wants to discriminate from the true images and the generated images. As seen in Figure 2.3 an image is sent to the generator model and with the addition of random noise, a new sample is generated which is similar to other samples in the class and this is called a fake sample. Generator is trained and optimized to generate best augmentation samples. The Discriminator is trained to distinguish between real and fake samples. Training these two models simultaneously gives us an improved Generator and Discriminator. GAN's can also be interpreted as a Two Player Game, where the generator is a counterparty which is trying to produce fake money and the discriminator as a detective trying to determine which is real money and which is fake. To succeed this game, the generator should excel in generating samples which are very close to other samples.

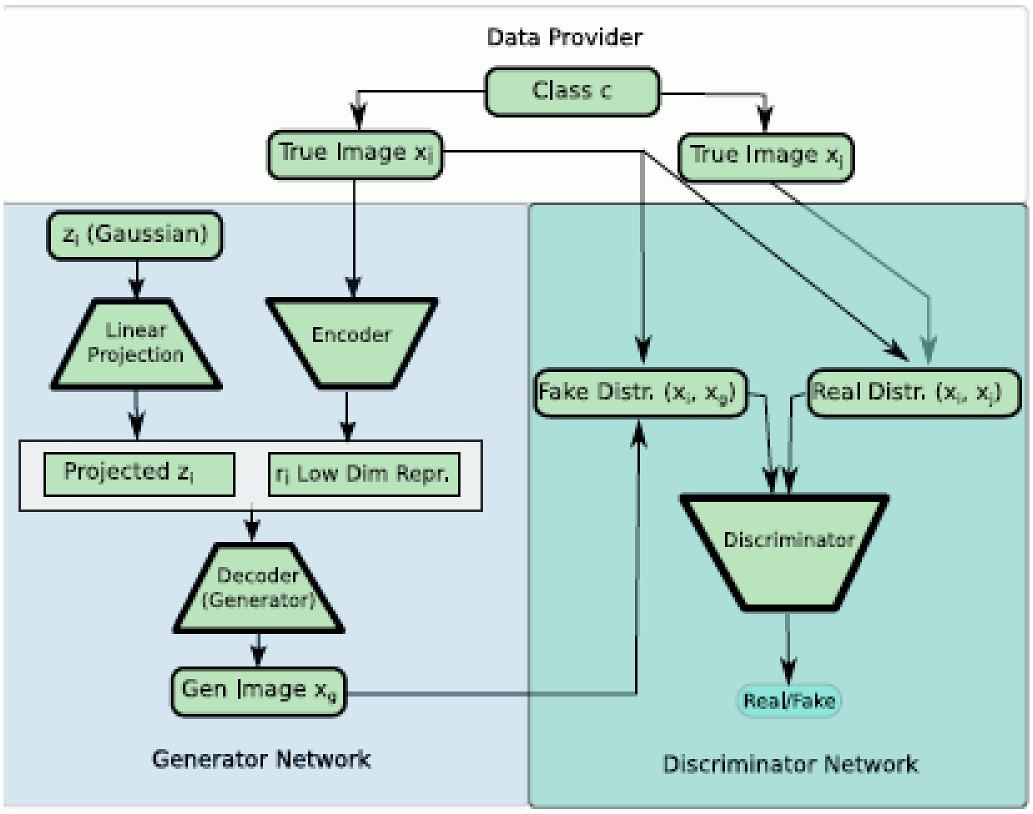


Figure 2.3: GAN Architecture [Antoniou et al. (2017)]

2.4 Conditional GANs

Conditional GAN is and expansion of GAN technique which conditionally generates the output. Here, the generator is trained to generate new samples with addition of some input condition. This condition can be anything such as class values, or the label of the class in binary classification. The discriminator is also conditioned as it is provided with the image as well as the label that either the image is fake or real. In classification task, the discriminator would expect a input with the correct class label. Smart Augmentation works slightly like a conditional GAN, this is covered in detail in Chapter 5.

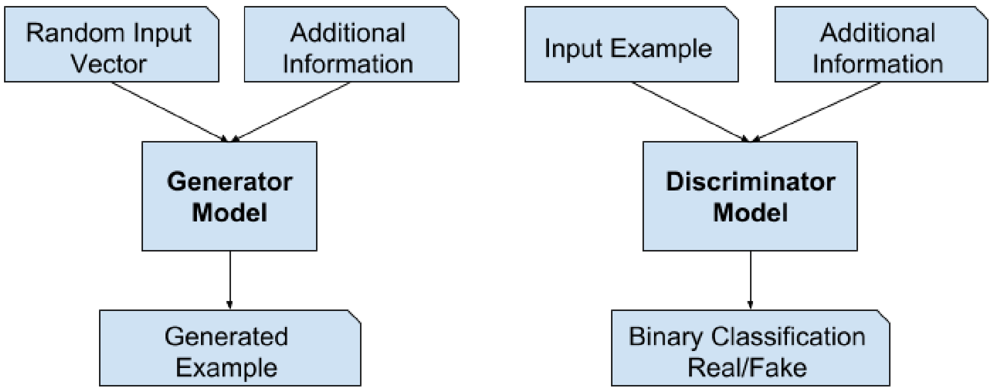


Figure 2.4: Conditional GAN Architecture

Figure 2.4 [1] shows the architecture of a Conditional GAN.

Chapter 2 gives a detailed background on the foundations of the thesis topic. To understand the above concepts and the relations between them is important. It summaries the connection of all topics to each other and how they are helpful in understanding the thesis better.

[1]https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

Chapter 3

Literature Review

This chapter gives a review of 9 good research papers that are around the thesis topic. The first article gives the introduction to the data augmentation techniques and the purpose of reviewing these articles. Next section explains an overview of the existing learnable augmentation techniques and briefly introduces the paper on smart augmentation. Finally, the limitations in existing study are addressed.

3.1 Introduction

The basic introduction on the concept of learnable data augmentation technique is provided in the study carried out in a paper published by Corcoran et al. (2020) it focused on how getting high-quality annotated data is difficult. This data is of importance because for best performance of a NN most important thing is the trainable data. There are many challenges in building the dataset for a highly complex NN model, as the need for good annotated data becomes more as the complexity of the model increases. This challenge is tackled by increasing the size of a small dataset using the data augmentation approach. Data augmentation is the process of creating new data samples based on making modifications to the existing dataset. For Example, considering an image of a human facial expression. Resizing one image give us different images of one original image and this helps the neural network to be trained better with all possibilities. Some of the common augmentation methods are Mirroring, Adding Gaussian Noise, Random Cropping, Blur + Flip, and Rotation. Data Augmentation in this study is performed on 2 types of images (i) Iris Segmentation of Low-Quality Smartphone Images where the images obtained have smaller size of Iris and is of poor quality; hence they have decided to use the high quality iris datasets which can mimic the low quality images obtained from smart phones. To achieve this they have used augmentation methods such as Shadowing and Motion

Blur. (ii) Iris Segmentation for Augmented Reality Headsets where the eye-gaze of the user is tracked from which new augmented samples are produced using Novel data augmentation method. Learnable augmentation is experimented where the neural network learns to perform the augmentation task and showed about 5%-7% increase in the performance among many classification tasks1. The purpose of this literature review is to know and understand different categories of learnable data augmentation techniques.

3.2 Overview of Learnable Augmentation Techniques

Lemley and Corcoran (2020) have summarised and discussed some of the most significant advanced data augmentation strategies. Learnable Data Augmentation was considered a shift from augmentation engineering to augmentation learning, where the former required a researcher or engineer to select the features before training and the latter has a neural network that learns to create augmented data samples. To test the augmentation methods on a task they have trained a network without augmentation and then tested it, then it is trained again with the augmented dataset and tested again. This study talks about many Augmentation strategies, (i) Using a neural network to learn the augmentation task and this is also called the "smart augmentation". Learning to augment data with a neural network where one neural network (augmenter) generates the data samples for network B (classifier). This new sample will be more generic for network B classification. Within SA, Another approach is called "neural augmentation" where two random images are used to train augmenter that will try to reduce the loss of the classifier. It is very similar to Smart Augmentation. (ii) Augmentation using reinforcement learning called the policy learning; it uses a policy to select the optimal strategy for augmenting the data. (iii) Evolutionary Image Augmentation which uses a construction of tree structure image transformations, in this automatic construction of images is used to create the augmented data. This study also includes some approaches on statistical generative techniques and GAN's, the former uses a statistical approach to generating new data samples from a given dataset. GAN based augmentation approaches have shown advantage over statistical generative techniques as they have used the "reward and penalty strategy" 2, GAN's architecture has a Generator and a Discriminator where the Generator creates fake samples and the discriminators task is to find the difference between real and fake images. These samples from generator are also referred to as hard augmentations. Some other GAN based approaches have been mentioned like Conditional GAN, Classic GAN which uses the technique of deep adversarial data augmentation and class-aware GAN where the discriminator outputs multiple probabilities. The Cyclegan and styleGAN could be used to train the networks to generate improved performance. Recent techniques like the “CutOut” and “MixUp” have considered being between smart and traditional augmentation technique, CutOut works by applying a zero sized mask to random locations of image and MixUp provides automatic augmentation by interpolation.

3.3 Smart Augmentation

Lemley et al. (2017) in their study have introduced a new technique called “Smart Augmentation” where a neural network learns the best augmentation strategy during training of the Deep Neural Network. Data augmentation is a technique to increase the dataset by making changes to the existing dataset. Two types of augmentation methods are specified, one is unsupervised augmentation (regardless of the label of sample) and another is supervised augmentation (label of data is used). This technique was initiated to deal with certain problems such as over fitting, failing to generalize on unknown data. Smart Augmentation comprised of network A that learns to create new datasets by merging two random samples from same class, output of network A is fed to another network ‘B’ and the loss from network B is back propagated to network A to update the weights. Here, network A is the Augmenter and network B is the classifier. The Loss (LA) from network A is used to improve network A’s sample generating capacity and Loss (LB) from network B is used to improve the weights for network B. Total Loss of the model will be a sum of the losses of both network A and network B which helps the augmenter generate best augmentations for network B. Further extending, multiple network A’s can be used, for example in a classification task one network per class can be used to generate more accurate samples. They have conducted experiments on NIVIDIA Titan X GPU’s, using 4 different datasets. Datasets included were Highly Constrained Faces Dataset (db1), augmented version of Highly Constrained Faces Dataset (db1a), FERET (db2), ADIENCE (db3) and MIT Places dataset (db4). All experiments were run for 1000 epochs with learning rate 0.01 using Stochastic Gradient Descent with Nesterov Momentum. Experimented on few combinations of the NN architecture, (i) SA with 1 network A showed a great reduction in over fitting also an improvement in accuracy from 83.52% to 88.46% on Feret dataset. (ii) SA VS traditional Augmentation, traditional augmentation improved the accuracy from 88.15% to 89.08% and with SA accuracy was 95.66% (iii) SA with 2 network A , the differences involved were they updated learning rate to 0.005, an increase in accuracy was seen from 92.94% to 93.35%.

Al Hajj et al. (2017) in their study proposed a method of how artificial video datasets can be generated using data augmentation methods. The purpose of this study was to generate artificial surgical videos in order to train the convolutional neural network (CNN) to perform the task of detecting the surgical tools on a tray. The model was trained to find out the different tools that are more likely being used by surgeons. They created a video of surgical tray and replicated the same gestures, these artificial videos were annotated. Standard data augmentation operations were carried out on the dataset which included rotation, random contrast enhancement etc. Here the data Augmentation module was done using "OpenCV and the CNN was pre-processed on Dell Precision Tower 5810: 8GB RAM and NVIDIA Geforce GTX 1080" 4. They used the One vs Others approach to build the dataset.

Related work is seen in an article by Li et al. (2020) a new auto-augmentation method called 'PointAugment' is introduced. This is similar to the smart augmentation approach and it only differs from SA is that PointAugment is applied to 3D point cloud processing. 3D datasets are much smaller compared to 2D Image datasets. Hence the need for increasing the datasets to train neural network is of much importance. Here the augmentations are performed considering each individual shapes For eg, applying rotation on a 3D image of sphere makes no difference. This technique applies such augmentation strategies depending upon the shape of the image. It also uses an adversarial training strategy to train the augmenter along with the classifier. It uses the losses from the classifier to let augmenter generate the samples which are optimal for the classifier. Experiment is conducted to support the claim that Point Augmentation is better than standard Data Augmentation techniques on datasets like PointNet, PointNet++, RSCNN, DGCNN which are trained with Point Augmentation and DGCNN gives the highest accuracy of 93.4% and without DA gives 92.2%. Architecture has a Augmenter and a Classifier, the augmenter follows two operations "shape-wise regression"5and "Point-wise regression"5, this will give an augmented data sample. The classifier extracts the features with which it performs the classification. The types of augmentations that are involved here are (i) Sample-aware where the augmentation strategy is based on the geometric shape of the sample. (ii) 3D augmentations, they include shape wise transformations and point wise transformations. (iii) Joint optimization involving jointly optimizing both the augmenter and the optimizer. Here Point Augment was successfully applied to Shape Classification and Shape Retrieval.

Lemley et al. (2018) have shown in this article how smart augmentation can be applied to significantly smaller devices without hampering the accuracy so that it would be more convenient to deploy them in smaller devices. Some results showing how smart augmentation works well on the VGG dataset with Test accuracy of 98.5%.

3.4 Augmentation Policy Learning

Cubuk et al. (2019) introduced a technique called “AutoAugment” in their article where it automatically searches the best augmentation strategy. It is augmentation with policy learning where a search space consists of policies and these are further divided into many sub-policies of which one is chosen at random. Here each policy gives options of choosing the different augmentation operations. And these operations are random image processing operations like rotation, flip etc. A search algorithm is used to pick out the best augmentation strategy and this is done with the help of Reinforcement Learning. They have conducted experiments on two cases (i) AutoAugment-direct where the augmentations is applied directly on the dataset to get the best policy (ii) Policies learned on one dataset are applied to other datasets which is called AutoAugment-transfer. The types of datasets used in the experiment are CIFAR-10, reduced CIFAR-10, CIFAR-100, SVHN, reduced SVHN and ImageNet. The search algorithm is implemented as a RNN. The Search Space had “ShearX/Y, TranslateX/Y, Rotate, Auto-Contrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout, and Sample Pairing” . Search algorithm consisted of a RNN and the training algorithm that finds the optimal policy. On the CIFAR-10 dataset, they attained 0.6% better error rate with the ShakeDrop model, Reduced CIFAR-10 dataset gave 0.2% better error rate on the Shake-Shake model. The performance of ImageNet model was similar to that policies found on the CIFAR-10. Authors were able to give best policies on ImageNet and SVHN dataset.

Test-time data augmentation is another new method focused in the study by Molchanov et al. (2020). Here, the predictions are averaged among many augmented data samples, and this has proven to show an increase in the predictive performance of the model. Purpose was to learn policies for test-time augmentations using greedy policy search.

3.5 Data Augmentation Using Generative Adversarial Networks

Antoniou et al. (2017) in their study have shown that Data Augmentation using the Generative Adversarial Networks provide good augmentations for the model to generalize on. Many regularization techniques like Dropout, normalization etc does not perform well when the data itself is short. They have recognized that a model of larger invariance can be learned too using a GAN. This helps in training the neural network better even with a smaller dataset. They have demonstrated how a model learnt from Data Augmentation Generative Adversarial Networks

(DEGAN) be applied to few-shot learning as well where it uses the idea of tangent distances. A most recent study carried out on GAN based one-shot model. They have experimented by training a DAGAN on small datasets by (i) Stochastic-gradient descent and (ii) one-shot meta-learning. They have made use of 3 datasets, Omniglot dataset, EMNIST dataset and VGG-Face dataset. Main base of GAN is that the learning from other problems to improve the learning on the problem of our interest. The DAGAN network consist of a Generator and a Discriminator, Generator takes an input image sends it to the encoder where it is converted into a low dimensional vector, a random vector (Gaussian Noise) is added to this and sent to the decoder which outputs the augmented image. The discriminator network is trained to find the difference between the real image and the fake image (augmented image). This will lead in generating new samples which are different from the real ones. The DAGAN was trained for 500 epochs, using a learning rate of 0.0001 and an Adam Optimizer. Results found on Omniglot 5 dataset with DAGAN gave a test accuracy of 82.131%, EMNIST dataset with DAGAN 100 samples per class gave 84.8%, VGG-Face dataset with DAGAN 25 samples per class gave 58.4%. Test on Matching Nets with DAGAN gave accuracy of 97.4%

3.6 Limitations of Existing Study

Within limitations of the study if on the MNIST dataset the digit 9 is rotated about 360 degree then the resulted image will be 6 and loses its original information, datasets need to be unaffected by the angle shift, in addition to the augmentation on AR requires affine transformation Corcoran et al. (2020) .The classic statistical models have not shown good results on more challenging datasets. Cyclegan shows performance lower than the traditional augmentation techniques Lemley and Corcoran (2020). Smart Augmentation could be only applied to Classification tasks and need to be extended to Regression problems as well. Use of network A for each class will result in scaling issues in multi class classification problems Lemley et al. (2017). This SA method does not apply to CNN3. Standard Data Augmentation techniques cannot be applied to 3D shapes Li et al. (2020). Data Augmentation (AutoAugment) approach working best for one dataset may not work well on another dataset, AutoAugment Transfer are not as useful as the AutoAugment applied on the datasets directly Cubuk et al. (2019). Data Augmentation cannot be transferred to one-shot learning and Meta learning. DAGAN does not perform well on the VGG-Face dataset

To summarize, the purpose of this chapter is to understand the study already existing around smart augmentation. This will provide a benchmark for researchers to build the thesis on. Understanding the techniques and approaches in the market we can easily extend the implementations and expand the model approaches.

Chapter 4

Data Analysis

Since data is very important in studying deep learning, in this chapter the datasets used for experiments are introduced. It contains information about some of the data pre-processing steps and gives an understanding about the type of datasets used in training a Convolutional Neural Network such as the Smart Augmentation approach. Detailed information about each dataset along with the sample visualizations is seen.

To judge the model, evaluation here is done on 3 datasets which would be sufficient to understand the performance of the algorithm. In order to know the efficiency of Smart Augmentation, a comparison of experiments is done on a model where Smart Augmentation is applied and on a model where Smart Augmentation is not applied. Inorder to make the experiments general with respect to all datasets, all the images were resized to 96X96 grayscale with pixel values regularized between 0 and 1. This experiment does not require a lot of data pre-processing steps as the augmentation of the images is done while training. However, one requirement is that the images need to be kept in separate folders by class so that while loading the data in pytorch, the labels of class are read along with the images. For example, in a binary classification we have 2 classes the images of class 0 need to be in a separate folder and the images of class 1 in a separate folder. Below shows the required path directory for images,
TrainData = “/dataset/Train”
ValidationData = “/dataset/Validate”
Where Train folder consists of two subfolders “Class0” and “Class1” and the same for Validate folder. These folders increases as the number of classes increase. All the datasets introduced below are suitable for Binary Classification and they contain only 2 classes.
Dog Vs Cat dataset : Class 0 - “Cats”, Class 1 - “Dogs”

Gender Classification dataset : Class 0 - “Male”, Class 1 - “Female”

Stanford Cars dataset : Class 0 - “Hatchback”, Class 1 - “Sedan”

4.1 Dog Vs Cat Dataset

This dataset is composed of Dogs and Cats images which were downloaded from kaggle datasets with link -d biaiscience/dogs-vs-cats. It consisted a total of 25,000 images of dogs and cats. This dataset was split into Training and Validation sets with 80% for training and 20% for validation. This is the largest of all datasets used for testing in this thesis. Figure 4.1 shows some examples of dataset

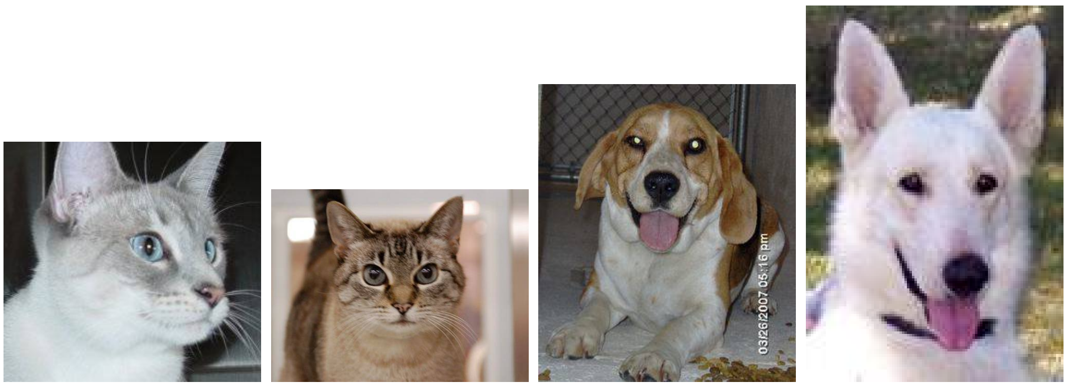


Figure 4.1: Samples for Dog Vs Cat dataset

4.2 Gender Classification Dataset

This dataset is composed of Male and Female images which were downloaded from internet where it was openly available. It consisted a total of 1,000 images of male and female. Each image is unique and no image has been repeated. This dataset was split into Training and Validation sets with 80% for training and 20% for validation. Total number of train samples was 800 of which 400 images belong to class 0 (male) and 400 images belong to class 1 (female). This is a smaller dataset used for testing in this thesis. Figure 4.2 shows some examples of dataset

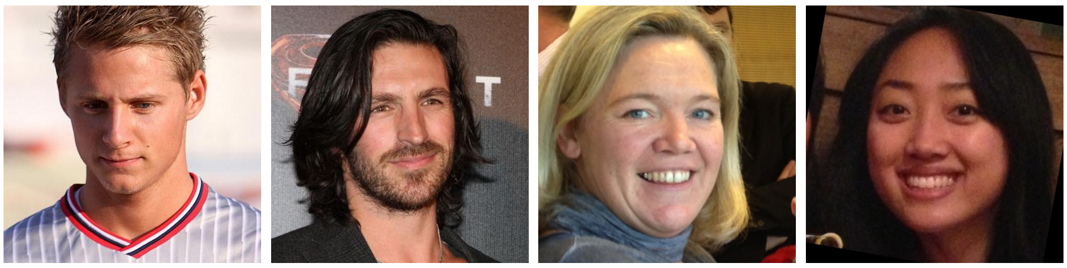


Figure 4.2: Samples for Gender Classification dataset

4.3 Stanford Cars Dataset

This dataset is composed of Hatchback and Sedan Car images which were downloaded from kaggle datasets as well. For this experiment, the dataset consisted a total of 1,500 images of Hatchback and Sedan Cars. This dataset was split into Training and Validation sets with 80% for training and 20% for validation. Figure 4.3 shows some examples of dataset

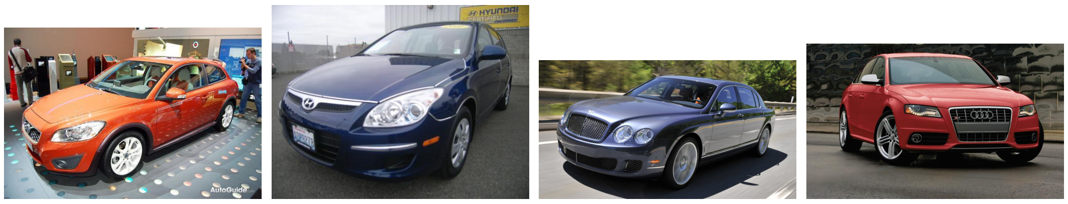


Figure 4.3: Samples for Stanford Cars dataset

In this chapter we understood about the data and the type of images being used in this experiment.

Chapter 5

Methodology

In this chapter, we introduce the model in detail and explain the working of the algorithm along with a deeper understating of the model architecture. This section puts light on all the logic behind the working of the model. First, we look over some points on pytorch and its importance as in Chapter 2 pytorch is covered in detail. Secondly, we go ahead and look at Convolutional Neural Networks (CNN's) and understand their importance. Next, we understand about the input data and some pre-processing steps. In detail, the structure of Neural Network A and Neural Network B is seen in this section. The combined loss function, which is the backbone of the entire model as it functions as an adversarial loss is covered in detail. Finally, the training loop of the model is explained where all the function take place. We also see the validation part of the model. The primary goal of the thesis was to see and understand how smart augmentation technique introduced in Lemley et al. (2017) would work on one of the fastest deep learning frameworks (pytorch).

5.1 Pytorch

Pytorch API has many packages such as torch, torch.nn where the former is a package of multi-dimensional tensors used in optimization and the latter is a building block layers for neural networks as it consists of Convolution Layers, Pooling Layers, Linear Layers etc. In pytorch, it is very easy to create a neural network and it can be done by just using the torch.nn module.

5.2 Convolutional Neural Networks

Convolutional Neural Networks also called CNN's or ConvNet's. They are a regularized version of MultiLayer Perceptron's. It is an algorithm which has proven to work very well for tasks such as image classification, face detection and many more computer vision tasks. It has the ability to assign weights to many objects of the image and extract information and can distinguish between one or more objects. It can capture the temporal dependencies of image by using filters. The goal of these networks is to reduce the images to a lower from than their actual size without losing valuable information from the data. Below figure [1] shows a typical structure of a CNN,

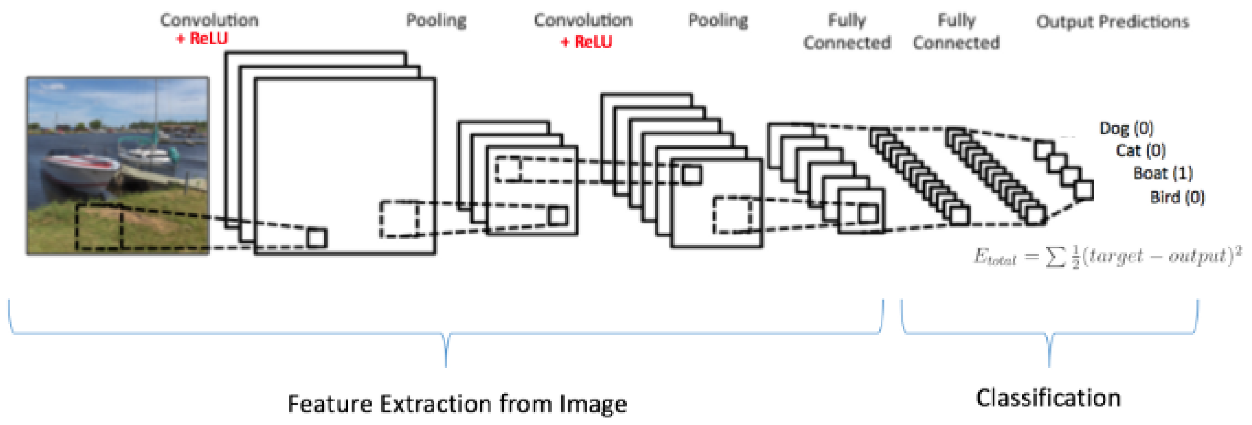


Figure 5.1: Convolutional Neural Network

The Convolution layers extract features from an input image by sliding the kernel on the image and develop another image which is called the feature map. The filter that slides through the image is called a kernel which is a matrix. Some important points in a CNN are, Stride length is a rate at which the kernel slides over the image, it moves by pixel increments. Padding is when we add 0's at the end of an image to make sure the kernel slides over all the edges of the image. Most common activation function used for CNN's is ReLU, which stands for Rectified Linear Unit. It converts all negative values to 0. It can be used in pytorch by simply calling torch.nn.ReLU(). Pooling Layer helps in reducing the size of the input image. Typically a Convolution Layer is always followed by a pooling layer, as it helps in increasing the speed of the neural network and it makes learning more robust. Pooling layers creates a reduced map. There are two types of pooling, Max pooling and average pooling. In max pooling, the maximum values from a feature map are selected to produce a reduced map. In average pooling, the average values from a feature map are selected to produce a reduced map. A fully connected layer is the last layer of the Convolutional Neural Network. The feature maps and reduced maps that were generated by the previous layers are flattened and converted to a vector. These vectors are fed to a fully connected layer that extracts the high level relationships between the features. The output of this layer is always a single dimensional vector.

[1]https://towardsdatascience.com/understanding-convolutional-neural-networks-221930904a8e

Using the formula shown in Figure 5.2 [1], we can calculate the output dimension after each convolution operation. By this we can create the fully connected layers with respect to different image dimensions.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}:	number of input features
n_{out}:	number of output features
k:	convolution kernel size
p:	convolution padding size
s:	convolution stride size

Figure 5.2: Convolution Operation

Creating a neural network in pytorch is pretty easy. The major layers described above in the CNN class can be implemented using,

- torch.nn.Conv2d(input_channels, output_channels, kernel_size, stride, padding)
- torch.nn.relu(x)
- torch.nn.MaxPool2d(kernel_size, stride, padding)
- torch.nn.Linear(in_features, out_features)

In pytorch, every operation in a neural network is defined in a single class. It also includes a forward method which computes the gradients of a forward propagation that includes the steps as outlined above. Here, when creating each layer manually we need to correctly give the input channels and output channels and the correct sizes. Let us know more about the model developed in this thesis using CNN's.

[1]https://towardsdatascience.com/pytorch-basics-how-to-train-your-neural-net-intro-to-cnn-26a14c2ea29

5.3 Model Architecture

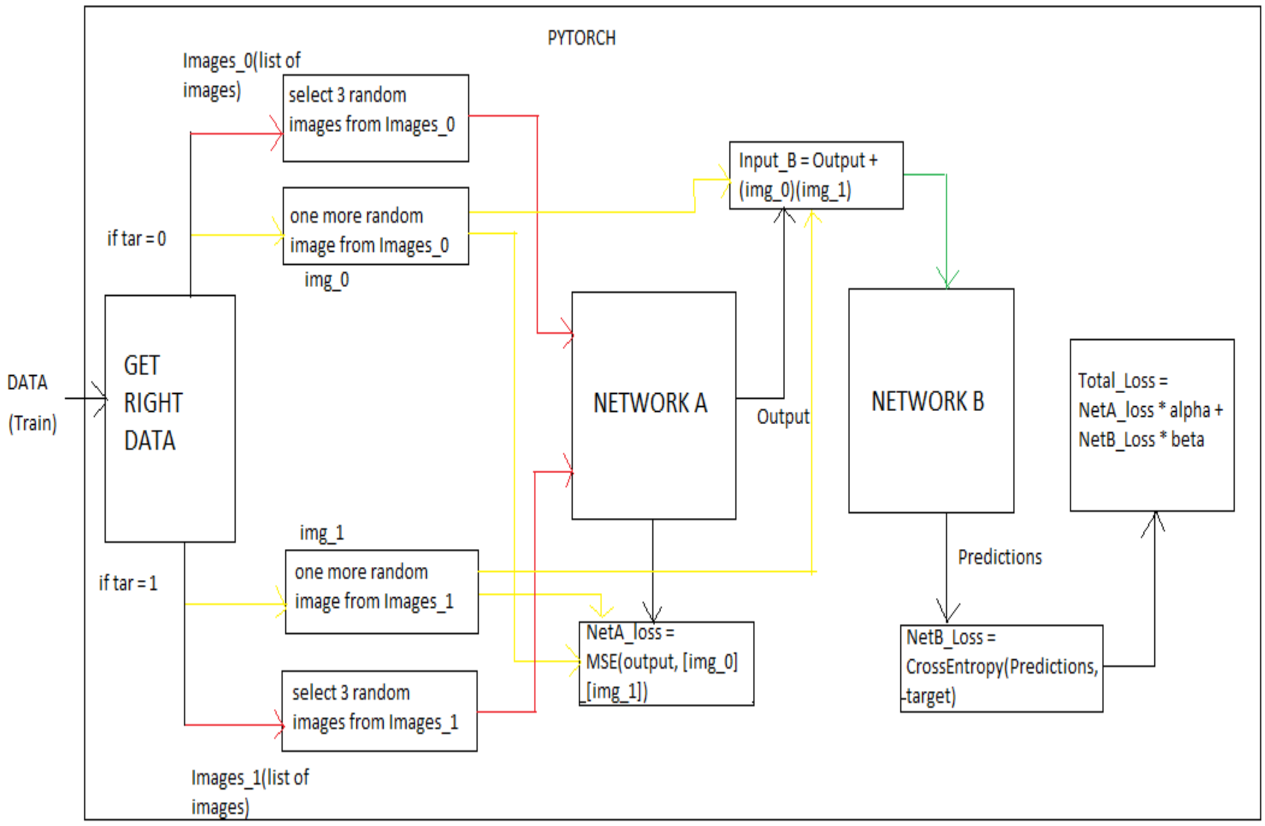

PYTORCH
Images_0(list of images)
select 3 random images from Images_0
one more random image from Images_0
img_0
if tar = 0
Input_B = Output + (img_0)(img_1)
DATA (Train)
GET RIGHT DATA
NETWORK A
Output
NETWORK B
Total_Loss = NetA_loss * alpha + NetB_Loss * beta
if tar = 1
img_1
one more random image from Images_1
Predictions
NetA_loss = MSE(output, [img_0] [img_1])
NetB_Loss = CrossEntropy(Predictions, target)
select 3 random images from Images_1
Images_1(list of images)

Figure 5.3: Model (Training part)

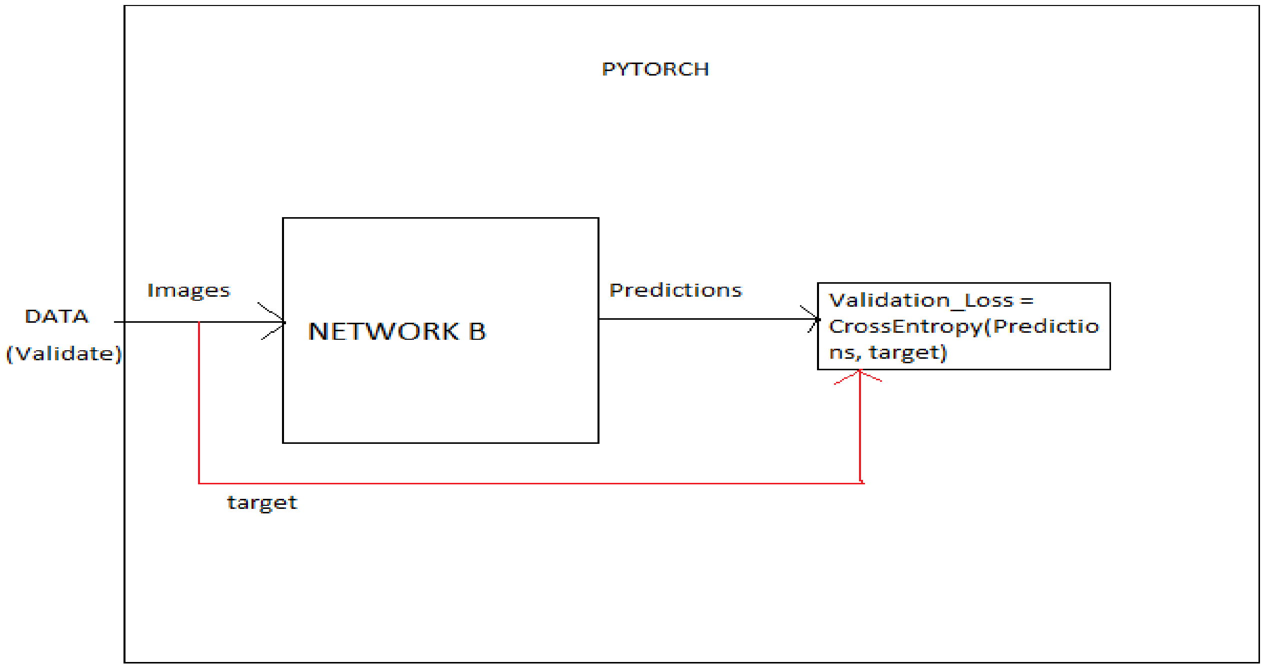


Figure 5.4: Model (Validation part)

Figure 5.3 shows an overview of the training part of model that is built on pytorch as part of the thesis. The model can be clearly divided into two parts i.e. Training and Validation. Here, Training is done along with the Smart Augmentation technique and Validation of the model is done only on the second Neural Network (Classifier) to compare the results as to how effective is training the model with smart augmentation technique. There are 4 important parts to the model,

- GET RIGHT DATA: This method can be considered as a data pre-processing step, which is mainly concerned with getting the correct data to be augmented during training of the model.
- NETWORK A: This is a Artificial Neural Network which is built using the pytorch API package, it acts as a NN that generates data (Augmenter).
- NETWORK B: This is another Artificial Neural Network which is built using the pytorch API package, it acts as a NN that classifies (Classifier).
- Total Loss: This is a combined loss function that helps in influencing Network A by using the loss of Network B.

These points are explained clearly further in this section. Briefly, the flow of the model is as follows. Input data is read and sent to a method called Get_Right_Data which will return required input data with correct shapes. This data is sent to Network A which will output a single image. This output is sent to Network B to get predictions.

Figure 5.4 shows an overview of the validation part of the model. It includes Network B which is the classifier as seen above, here the trained model is evaluated with the validation set. In deep learning, model validation is carried out after each epoch during training so that it gives an unbiased information as to how good the training process is carried out. This is understood more clearly while explaining the working of the algorithm. In order to understand each component of the model correctly, we now focus on knowing the details of every part of the algorithm along with its workflow.

5.3.1 Input Data to Network A

It is necessary that the input data is as required; the data that is loaded onto pytorch needs to go through certain steps before it can be sent to the network for training. Let us know what these requirements are, from the data that is read we need to select 3 random images from the same class. These images are stored in a list and as all the images are resized to 96X96 as seen in Chapter 4, Below figure shows the input shape of each image,

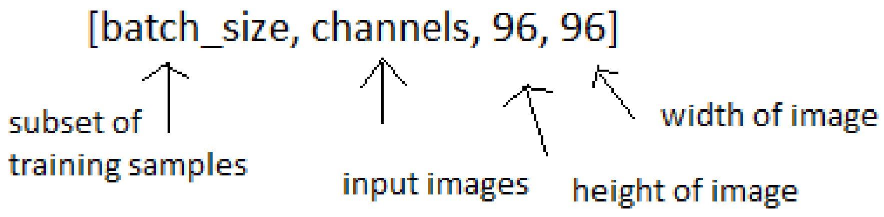


Figure 5.5: Input Shape of Image

The shape of each image will be [batch_size, netAInpchannels, 96,96] where netAInpchannels here in this thesis experiment is 3 i.e. 3 random images are used, this value can differ to other experiments depending on the requirement. These 3 images are merged into a single image and this merged sample is sent to Network A which will produce an image which is similar to other images of that class. However, when we load the data labels of the class are also read, these labels get jumbled when we select the random images from the entire data. It can select either all from the same class or can select one from the required class and the other two from different class. In this way, even the labels of the images get mismatched and the Network won't know which image belongs to which class and this prevents the model from learning anything and will create one of the scenarios of a dataset with noise. To prevent this, we make use of the method Get_Right_Data which is explained in detail below.

5.3.2 Get Right Data Method

As mentioned earlier, this method can be considered as a data pre-processing step. It is a method that is written in python and is called during the training process. It makes use of basic python techniques such as lists, random.choice and some pytorch technique such as torch.stack to merge the data. The main goal of this method is to send correct data to the network A and avoid the noise creating in the dataset. In a training loop, we go over all the samples present in the train dataset and this data is read along with its class labels. Firstly, these images are separated by their classes so that the labels of each image are intact and unchanged. Two separate lists are created according to the class labels of these images i.e. images of class 0 are appended to separate list and the same for images of class 1. This way we can select the random images of class 0 first and once all the images of class 0 are iterated over, the step is repeated for class 1 images. It also selects a single random image from the same class, which is used to compare how close the produced image is from other images of the class. This method returns 3 arguments, the merged single image of 3 random images, the label of these images and another random image from the same class. These 3 inputs are made use in the next phase where the merged sample is sent to Network A.

5.3.3 Network A - Augmenter

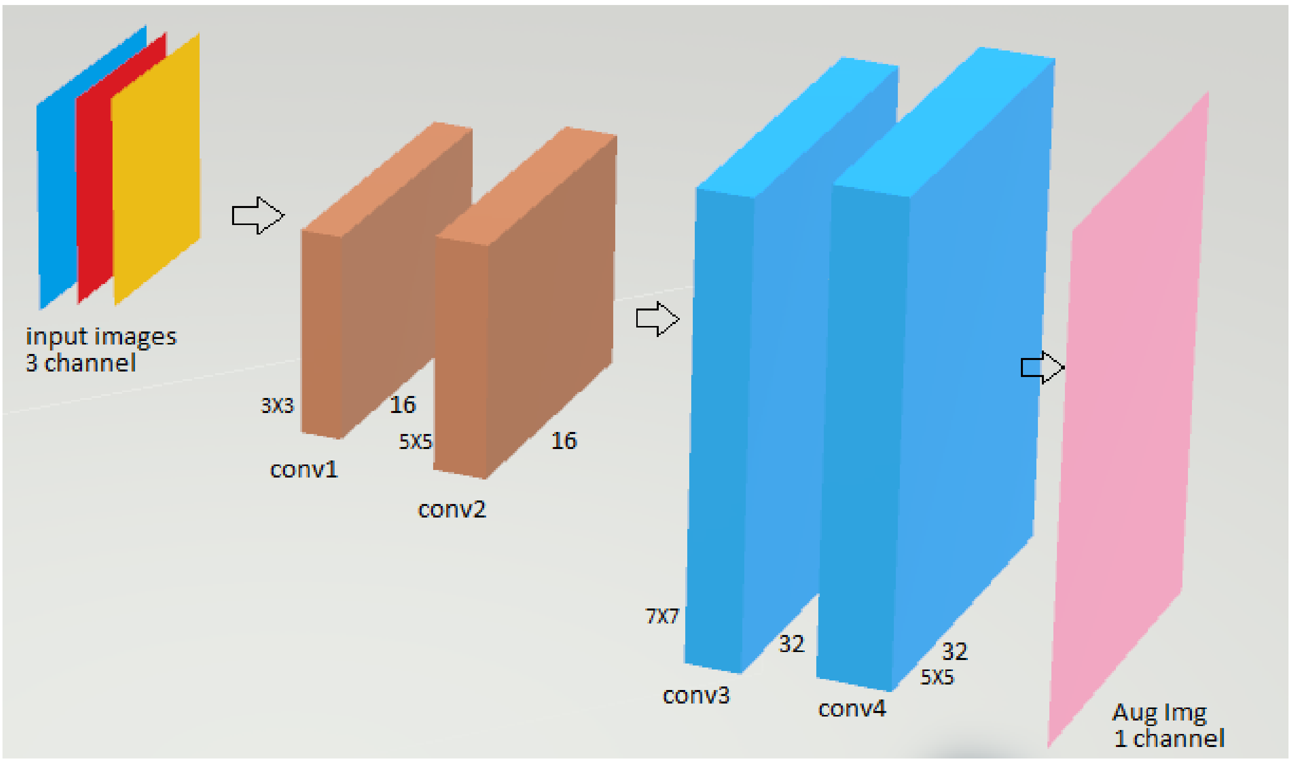


Figure 5.6: Network A

Figure 5.6 shows in detail the set up of the Artificial Neural Network A which is referred to sometimes as Augmenter in this thesis. It is a simple Convolutional Neural Network with stack layers; it is created in pytorch using the torch.nn Module. The importance of the network is, it takes in a multi-channel image and outputs an image with single channel. Network A consists of four Convolution Layers, let us break down this architecture in detail. The first layer of the network is the Input Layer, which takes in the merged image which was processed in the previous step. It expects the input to be of shape [batch_size, netAInpchannels, 96, 96]. From the figure the input images in 3 colors show the random images selected from the same class. Each image is a gray-scale image i.e. it is of shape [batch_size, 1, 96, 96]. After merging these 3 single-channel images, we get the shape of merged image to be [batch_size, 3, 96, 96] which is as required by the input layer. The next layer of the network is a 2d convolution layer with 16 filters and kernel size of 3X3, stride set to 1 and padding set to 2. In addition, one more 2d convolution layer with 16 filters and kernel size of 5X5, stride set to 1 and padding set to

2. Next is a 2d convolution layer with 32 filters and kernel size 7X7 and the output layer of this NN is a 2d convolution layer with slightly larger filter of 32 filters and kernel size of 5X5. This final Convolution layer produces an image of single channel, i.e. the shape of this image is [batch_size, 1, 96, 96]. In pytorch, the building of neural networks is in a class and it consists of a forward method and we can use any tensor operations within this method.

5.3.4 Network B - Classifier

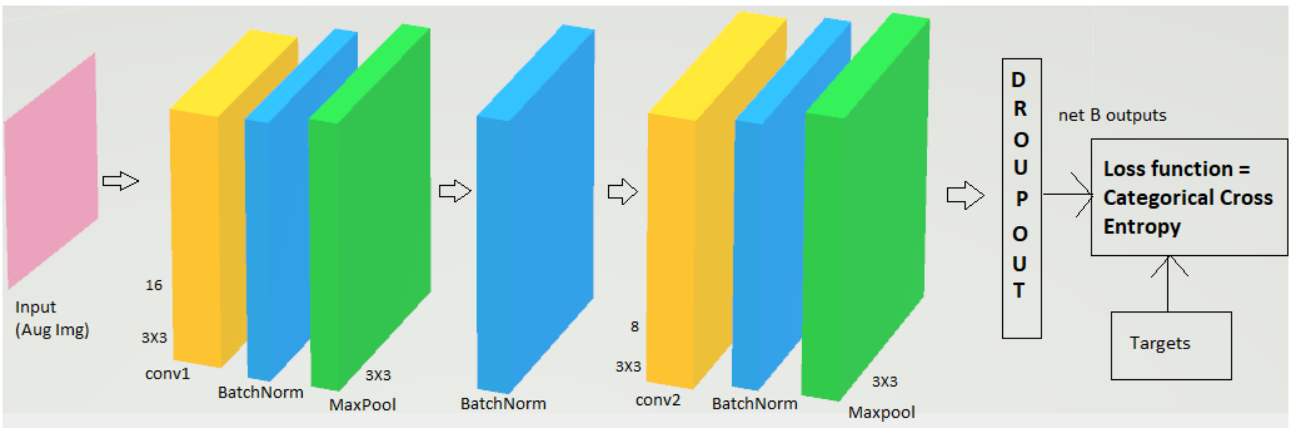


Figure 5.7: Network B

Figure 5.7 shows in detail the structure of Convolutional Neural Network which is the classifier. It consists of 2 Convolution layers along with layers of max pooling and batch normalization. Max Pooling layers as we already saw are the layers that convert all the negative values from a feature map to 0. Batch Normalization layers are used in very deep neural networks to standardize the input of each layer in a batch. This helps in stabilizing the learning process. After each Convolution layer and activation function ReLU is used. In pytorch, max pool layer and batch normalization layers are a part of the pytorch API. It can be set up by simply using torch.nn.BatchNorm2d(), torch.nn.MaxPool2d() respectively. First layer is a 2d conv layer with 16 filters and kernel size of 3X3, stride length of 1 and padding of 2. Followed by this is the relu activation function and next is the batch normalization layer and max pooling layer. The same set of layers are stacked on top with only the filter size of the adjacent conv2d layer changed to 8 filters and kernel size of 3X3. Finally, there is a fully connected layer and in pytorch this layer is set up by using torch.Linear(). This layer has 1024 units and next is a dropout layer with probability set to 0.5. Dropout layers are used in neural networks to avoid overfitting of the data. Dropout is randomly selects the outgoing hidden neurons and drops them off by setting them to 0. After, this there is one more fully connected layer with output units as 2. This is because we are using this network for Binary Classification, and we get the predictions as probabilities with shape [batch_size, 2]. Finally, the last layer is a Softmax Layer which is used to get the predictions of the class values.

Network A and Network B are connected in the forward pass, as the output image given by Network A is used to train the Network B. The cost function of Network A tries to minimize the loss of Augmenter by back propagation. The loss of Network B helps Augmenter to produce better Augmentations, however Netwrok A can't learn on its own because of which we use the loss of Network B to influence Network A. This is called the Combined Loss Function which is covered in detail below.

5.4 Loss Function

In neural networks, the task of learning is a simple optimization problem. Here the algorithm tries to find the optimal weights in order to get best predictions. Typically, gradient descent is used in all algorithms as the optimizer which tends to update weights after every iteration. The gradient is nothing but the error between the true values and the predicted values. Our aim is to minimize this particular error, here the objective function is called the loss function and the error is called the loss. Each neural network tries to minimize this loss value, so that the model learns and converges at a point where it has learnt all the features from the dataset used for training. Optimizer used here in this experiment is the stochastic gradient descent often called as SGD. It is one of the type of gradient descent. Here, stochastic means random probability i.e. in each iteration only one sample is selected randomly instead of entire dataset. In this it uses only batch_size of 1 for every iteration; samples are at random and shuffled. Since, SGD chooses only one sample from the dataset at random, the path taken to reach the minima is noiser. However, that does not matter as long as it reached the minimum.

5.4.1 Loss Function for Network A

In this section, we define the loss function for NN A which is the Mean Squared Error (MSE). It is one of the commonly used loss function in many deep learning tasks, it is the sum of squared distances between the target image and the predicted image. Figure 5.8 [1] is the formula for the mean squared error loss function.

[1]https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

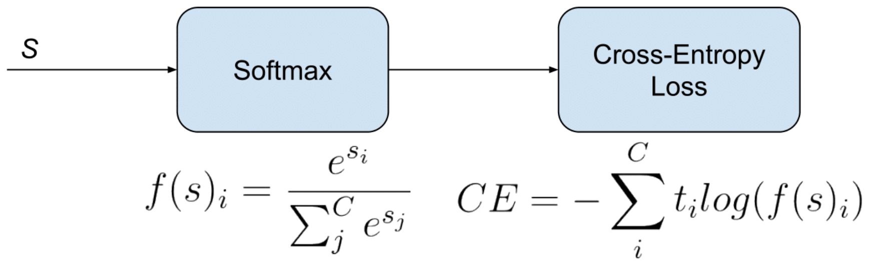
$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

Figure 5.8: Mean Squared Error

Here, the loss is the measure of how well the Augmenter has generated the combined images. The loss calculated is between the output image of Augmenter and another target image that was selected randomly from the same class. This loss (Loss A) is used to train Augmenter by the means of back propagation. In pytorch, loss can be back propagated by calling loss.backward() function, which is built-in in pytorch.

5.4.2 Loss Function for Network B

Since, NN B is a binary classifier and its predictions are probabilities ranging from 0 to 1, the loss function used here is Categorical Cross-Entropy Loss. The loss is defined as follows [1],



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

Figure 5.9: Categorical Cross-Entropy Loss

The above loss function is also referred to as Softmax Loss and that is because it consists of Softmax activation function and the Cross-Entropy Loss. The loss for Classifier (Loss B) is the measure of how well the network has learned to classify the images of two classes. This loss is calculated between the true values and the predictions given by Network B. Since, network B is being trained on the images generated by Network A, we back propagate the loss of Network B to Network A and this is done by the combined loss function as shown below.

5.4.3 Combined Loss Function - Total Loss

To influence Network A using Network B, we add the loss of Network B and Network A which is called Total Loss in this thesis and we back propagate the total loss. Two parameters alpha and beta are added to the losses. The loss function is defined as below,

[1] https://gombru.github.io/2018/05/23/cross_entropy_loss/

Total loss = Loss A * alpha + Loss B * beta

Figure 5.10: Combined Loss Function [Lemley et al. (2017)]

The two NN are connected again in the back propagation step and the loss from NN B is sent to NN A which will let NN A produce better images for NN B and decrease the loss of NN B and help increase its accuracy.

5.5 Working and Code Flow

5.5.1 Loading Data

In this section, we will learn the working of the model put together all the components described above and also give a code work through. In pytorch, the first step is to import all the necessary libraries and define the required image height and width. Following this will be loading the data; here there is quite different way to load data i.e. the images that are loaded must be converted to tensors in order to process. Hence, the pytorch transforms are used here; the transforms give us many options as to resize the images according to our requirement, here the images are resized to 96X96 as mentioned earlier. Then they are converted to grayscale using the transforms.Grayscale() option. Finally images are converted to Tensors using transforms.ToTensor() function. The transforms also give the flexibility to normalize the image values here itself; however we have done this step later in the process. By giving the train and validation directory same way as mentioned in Chapter 4, we use datasets.ImageFolder to apply the transforms defined earlier to all the images in the train and validation directory. Finally using torch.utils.data.DataLoader, it loads all the data from the image directory. The images we get in this load are the images that will have been converted to tensors and along with the labels of the classes. The images are loaded according to bacth_size which makes it easy to further go over all images in batch_size iterations. All the hyper parameters are set here as well, including the values for epochs, Network A input channels, alpha, beta and the learning rates.

Next, the two neural networks are defined as mentioned in detail in above sections Network A and Network B where the architecture is shown. These networks are defined in two separate classes and are called when using NetA = NetworkA(),NetB = NetworkB(). Network A and Network B are the class names and NetA and NetB are used to call these neural networks during training process. Loss functions and optimizers for both the networks are defined next, along with defining some empty lists to keep track of the accuracies and losses.

5.5.2 Training Loop

This is the main training loop for the entire model where all the components covered earlier are called. It works in a way that for every epoch NetA.train and NetB.train is called simultaneously which puts both the networks in training mode. In every epoch, all the images in the train dataset are iterated over by batch_size. The data that is iterated over here are the images and the labels of each image indicating the class it belongs to. This data is sent to the Get_Right_Data method for the data pre-processing, where it fetches the correct images and performs the merging tasks as seen earlier and returns 3 arguments. In pytorch, there are important changes when training a neural network they are using the built-in methods such as optimizer_netA.zero_grad() which is called before sending the input to Network A. This will clear the old gradients from last step, if it is not called then the loss that is back propagated will just keep accumulating and the model does not work. After this, the input is sent to network A, the output from Net A is referred to as Augmented Image and this image is merged with another random image that was returned from Get_Right_Data method. This will be new input to Network B. Loss A is calculated between the augmented image and the random image of the same class, this will see the degree of how close is the image output from Net A is to other distributions of the class. Next, before sending the new input to Network B we call optimizer_netB.zero_grad() and get the predictions from Network B. Loss B is calculated between these predictions and the target values. Loss B is back propagated to Network B to minimize the networks loss. The total loss is calculated following this using the formula as shown above. Finally, the total loss is back propagated to Network A, and Network A tries to minimize this its loss which in turn minimizes the loss of Network B. Hence the total loss of the model is reduced. The back propagation is done in pytorch by calling loss.backward() function as already mentioned. The last step is to call optimizer_netA.step() and optimizer_netB.step() which updates the weights after each step. Loss of the model reported is the total loss of the model, and the accuracy is also calculated by finding out the number of correct predictions.

Next phase is the validation part of the model. Here, we call NetB.eval() which puts the model into evaluation mode. This is also executed for each epoch and is in the same training loop. Each image in the validation set is iterated over and sent through the Network B for validating the training process. Validation loss is calculates same as before with the outputs given by Network B and the targets. Validation accuracy and Validation Loss both are calculates and this will give us unbiased information on the training of the model.

This chapter gives a detailed explanation on the entire methods used in building the model. Along with its working and all the components involved. Gives a background around Convolutional Neural Networks and pytorch. Model architecture and each part are explained with all details.

Chapter 6

Experimental Settings

In this chapter we discuss, all the experiments that were run as part of evaluation of the model. It covers the steps included in training, evaluating and testing this model. Finally, a list is shown to summarize all the experiments that were done in this thesis.

A total of 3 datasets are tested here, as seen earlier what the datasets include. Running of the model is done in two steps, in the first step model is trained and validated at once in the loop. This trained model is saved using the basic torch.save command in pytorch. Finally the saved model is loaded and it is tested with the test set. Data is divided into 3 splits, the training set, validation set and test set with the split percent of 80% for train, 20% for test. All the experiments are run for 1000 epochs each with a learning rate of 0.001. There are two combinations in which the testing is carried out. First the model is trained and tested for all datasets with the Smart Augmentation approach and the results are computed. Then, to compare how the Smart Augmentation approach has performed, we train and test the model for all datasets without the Smart Augmentation approach i.e. the first set of experiments include both the Neural Networks A and B where as the second set of experiments without the Smart Augmentation technique include only Network B (the Classifier). Additional parameters that were used are as already covered; the optimizer used is the Stochastic Gradient Descent with the Nesterov momentum of 0.9 and the values of alpha 0.3, beta to 0.7.

6.0.1 List of Experiments

- DogVsCat dataset WITH Smart Augmentation, train samples: 20,000, test samples: 5,000.
- DogVsCat dataset WITHOUT Smart Augmentation, train samples: 20,000, test samples:

5,000.

- GenderClassification dataset WITH Smart Augmentation, train samples: 800, test samples: 200.
- GenderClassification dataset WITHOUT Smart Augmentation, train samples: 800, test samples: 200.
- Stanford Cars dataset WITH Smart Augmentation, train samples: 1200, test samples: 300.
- Stanford Cars dataset WITHOUT Smart Augmentation, train samples: 1200, test samples: 300.

After running the above experiments the results are computed that for each epoch there is train accuracy, train loss, test accuracy, test loss. In this thesis, only the best accuracy of all the epochs is shown. This will help us know the best performance of the model with respect to all the different combinations. Below, shows the table summarizing the above information.

Exp	Dataset	TotalSamples	epochs	learning rate	alpha	beta	SmartAug
1	DogVsCat	25,000	1000	0.001	0.3	0.7	YES
2	DogVsCat	25,000	1000	0.001	0.3	0.7	NO
3	DogVsCat	25,000	10	0.001	0.3	0.7	YES
4	DogVsCat	25,000	10	0.001	0.3	0.7	NO
5	GenderClassification	1,000	1000	0.001	0.3	0.7	YES
6	GenderClassification	1,000	1000	0.001	0.3	0.7	NO
7	GenderClassification	1,000	50	0.001	0.3	0.7	YES
8	GenderClassification	1,000	50	0.001	0.3	0.7	NO
9	Stanford Cars	1,700	1000	0.001	0.3	0.7	YES
10	Stanford Cars	1,700	1000	0.001	0.3	0.7	NO

Table 6.1: List of Experiments

All the experiments are run on python version 3.7.7 and pytorch version 1.6.0 on a GPU, which is helpful to run the DNN models. Details as mentioned earlier with 2 x 1080Ti (64 GB RAM) and Intel i7-7700K (4 core / 8 thread).

This chapter summarizes the type of experiments and the parameters used along with the environmental settings. Next chapter mentions the results of the experiments along with the analysis.

Chapter 7

Results

Here, all the results are discussed along with learning curves and performance curves. Finally, all the results are broken down and analyzed individually and the performances are compared and interpreted. Also, some of the augmented images that are produced during learning are shown as well.

Exp	Dataset	SmartAug	epochs	Test Accuracy
1	DogVsCat	YES	1000	80.89%
2	DogVsCat	NO	1000	79.05%
3	DogVsCat	YES	10	74.78%
4	DogVsCat	NO	10	77.45%
5	GenderClassification	YES	1000	72.23%
6	GenderClassification	NO	1000	68.56%
7	GenderClassification	YES	50	66.88%
8	GenderClassification	NO	50	69.01%
9	Stanford Cars	YES	1000	59.00%
10	Stanford Cars	NO	1000	60.19%

Table 7.1: Results

7.1 Learning Curves

Learning curves in deep learning show the graphical representation of relation between the lerning of the model against the time taken (epochs). Here, two learning curves are shown one is the loss curve showing the training and validation losses against

7.1.1 Experiments 3 and 4

For the DogVsCat Dataset, the learning curves are shown for 10 epochs

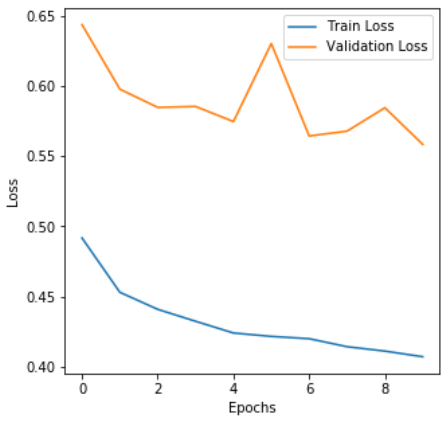


Figure 7.1: Loss curve with SmartAugmentation

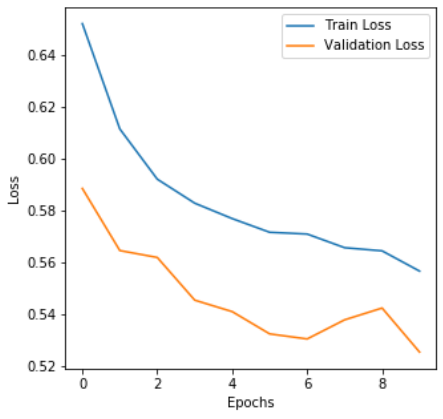


Figure 7.2: Loss curve without SmartAugmentation

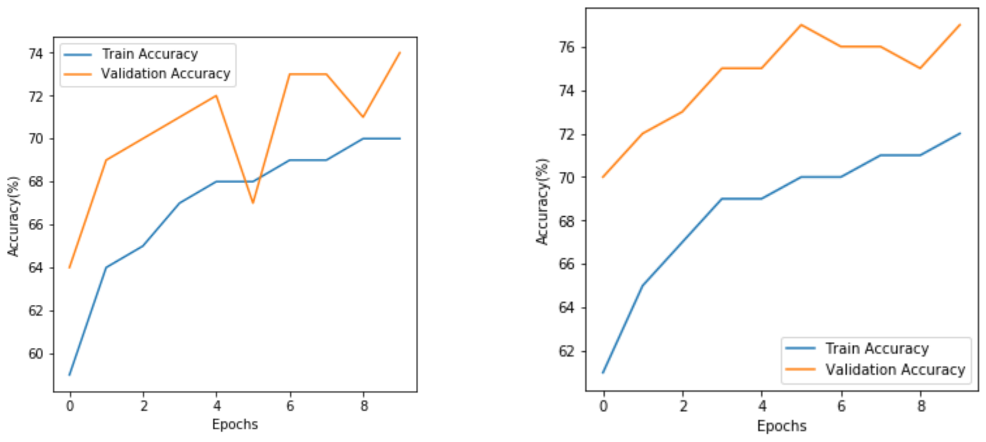


Figure 7.3: Accuracy curve with SmartAugmentation(Left)
without SmartAugmentation(Right)

7.1.2 Experiments 5 and 6

For the Gender Classification Dataset, the learning curves are shown for 1000 epochs

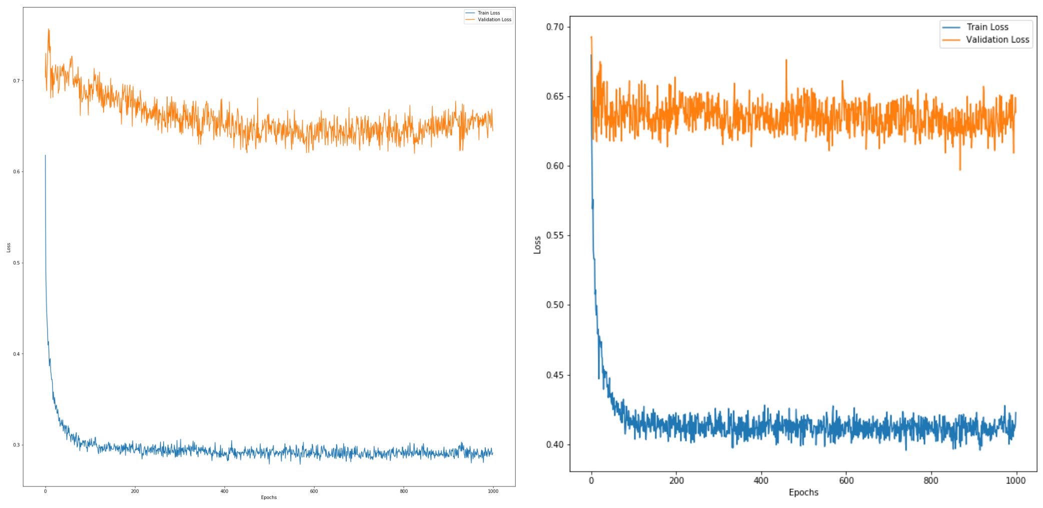


Figure 7.4: Loss curve with SmartAugmentation(Left)
without SmartAugmentation(Right)

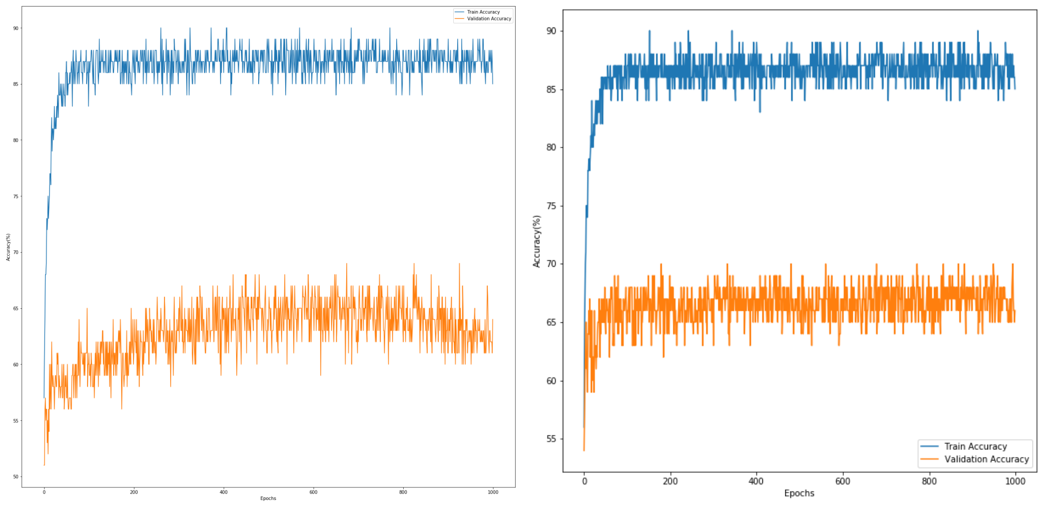


Figure 7.5: Accuracy curve with SmartAugmentation(Left)
without SmartAugmentation(Right)

7.1.3 Experiments 9 and 10

For the Stanford Cars Dataset, the learning curves are shown for 1000 epochs

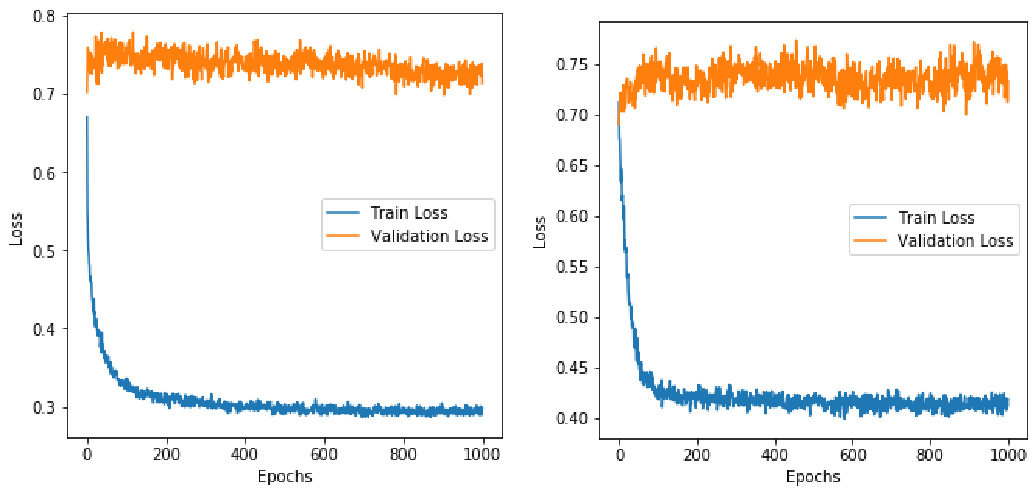


Figure 7.6: Loss curve with SmartAugmentation(Left)
without SmartAugmentation(Right)

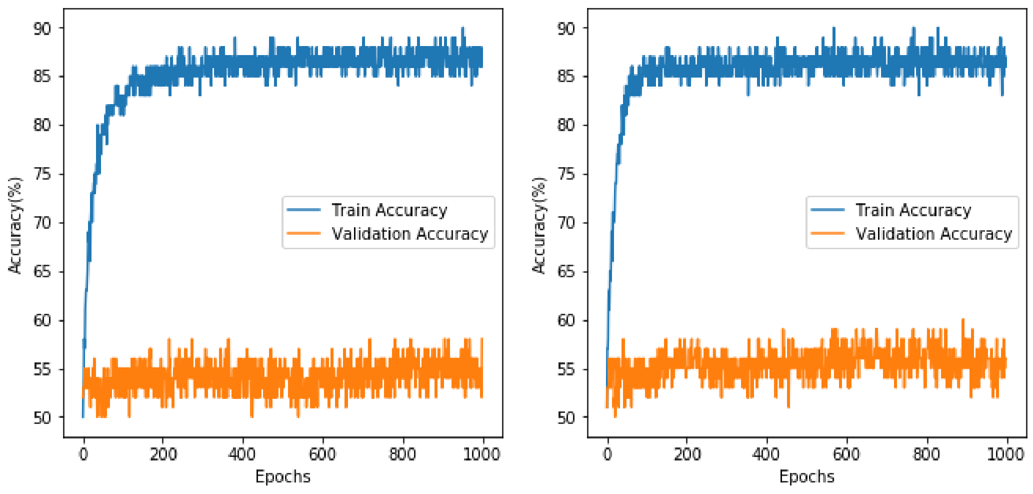


Figure 7.7: Accuracy curve with SmartAugmentation(Left)
without SmartAugmentation(Right)

7.2 Augmented Images

Below figures show some examples of the augmented images produced,

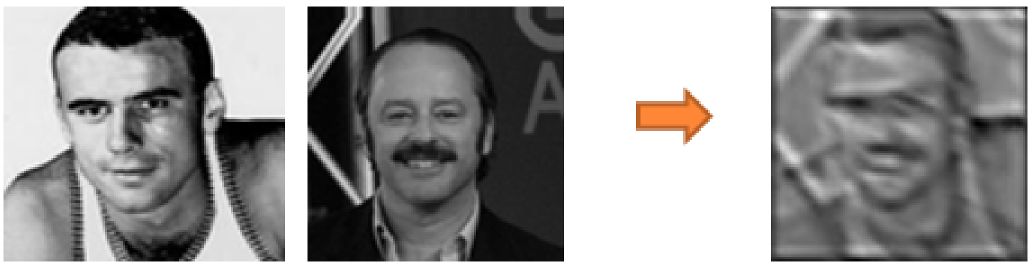


Figure 7.8: Augmentations_1

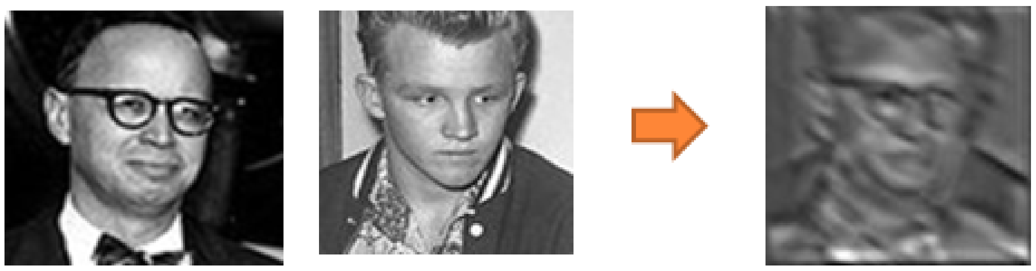


Figure 7.9: Augmentations_2

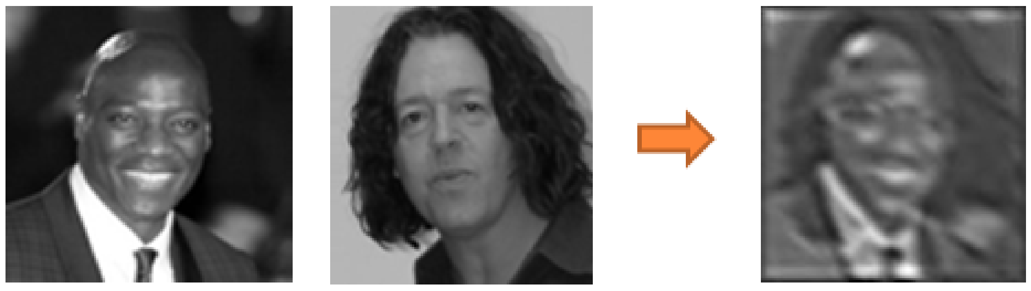


Figure 7.10: Augmentations_3

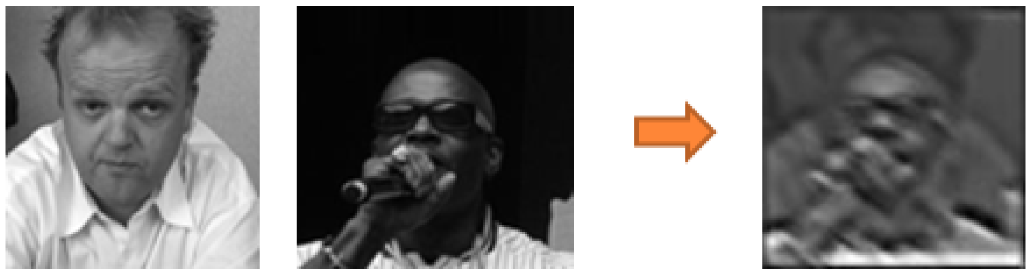


Figure 7.11: Augmentations_4

7.3 Analysis

In this section, we analyze in detail the results reported above. From experiments 1 to 4 we use the same dataset (DogVsCat) only with different combinations i.e. changing the number of epochs along with using the non smart augmentation model. The best result was given for combination of experiment 1 where the model was trained with 1000 epochs and it gave an accuracy of 80.89%. And without Smart augmentation gave an accuracy of 79.05%, for parameters of exp 1 and 2 Smart Augmentation proved to be effective with an increase in accuracy by +1.84%. From experiment 3 and 4, it was seen that the number of training epochs was reduced to 10 epochs and here without smart augmentation model gave an accuracy of 77.45% and with smart augmentation it gave an accuracy of 74.78%. With this parameter setting without smart augmentation model seemed to perform better. "It can be said that smart augmentation require large training epochs".

Next we look at experiments 5 to 6, on the Gender Classification dataset exp 5 and 6 were trained for 1000 epochs each. With smart augmentation gave an accuracy of 72.23% and without smart augmentation it gave an accuracy of 72.23%. This dataset is smaller when compared to the DogVsCat dataset. In this combination, smart augmentation performed better by 5.22%. Here, smart augmentation approach works better when trained with 1000 epochs. Another set of experiments performed on the same dataset but trained for lesser epochs i.e. trained for 50 epochs each. With smart augmentation approach gave an accuracy of 66.88% and without smart augmentation, the model gave about 66.88%. Here, the non smart augmentation model works better when trained with fewer epochs.

Finally, with the Stanford cars dataset when trained with 1000 epochs following were the results. With smart augmentation approach, the model gave a accuracy of 59.00% and without smart augmentation technique the model gave an accuracy of 60.19%. Here, on this model the non smart augmentation technique worked better. The reasons could be that the model performs different on many different datasets and the datasets could include noise as well.

Figure 7.8, Figure 7.9, Figure 7.10 and Figure 7.11 show the outputs from Network A the augmenter which are the learned combination of the random images from a class. The two random images on the left give a learned combination of the image as on the right.

This chapter showed in detail the results which included the test accuracies in each experiment. Then the graphical illustrations of all the experiments which had the loss curve and the accuracy curve for both smart augmentation approach and without smart augmentation approach. Some of the augmentations produced during the training of the model are seen as well. Finally, the interpretation of the results is done.

Chapter 8

Conclusions and Further Work

This is the final chapter of the thesis, which brings a close to the agenda established at the beginning in Chapter 1. It summaries the model developed above in pytorch and draws conclusions from the results presented in the previous chapter. In the final section of this chapter we also discuss some of the limitations as well as contributions from this research. Finally, the further work of this research is mentioned on how this research can be expanded and implemented more things on top of it.

8.1 Research Summary

The start of the thesis, the intension behind implementing the algorithm on pytorch was given. As pytorch is one of the fastest deep learning frameworks. It also shed light on some of the issues that could be tackled by smart augmentation approach. The scope of the thesis was mentioned along with how the training and evaluation of the model would be carried out. In Chapter 2 all the elements around smart augmentation was explained which included the basic data augmentation technique and the smart augmentation approach was given in detail, along with some of the topics related such as Generative Adversarial Networks and Conditional GAN's. In Chapter 3 9 good research articles were reviewed which included smart augmentation as the base. It introduces different data augmentation techniques already existing and how current challenges were tackled by data augmentation techniques, and also gives an overview of all the learnable augmentation techniques. It also includes the review of the paper using which the model in this research is implemented. Some new ideas seen in the research papers such as the AutoAugment using policy learning, finally the study on Generative Adversarial techniques was seen. Chapter 4 gives in detail the description of the datasets that were used during the experimentation of the model (Evaluation). Chapter 5 is the backbone of the thesis structure that gives the entire model explanation in detail, which includes a brief intro to pytorch and in detail the explanation of Convolutional Neural Networks, finally the model architecture is broken down and each part is detailed clearly. It also includes the entire working of the model which brings a good understanding of the model. Chapter 6 gives in detail all the experiments that were run as part of the thesis and explains all the hyper parameters and the different combinations of the parameters. Finally, in Chapter 7 all the results of evaluations were seen along with the augmented images and the learning curves followed by the interpretation of the results.

8.2 Research Weaknesses

- The results provided were not perfect i.e. the system performed better Lemley et al. (2017), however the system showed poor results in pytorch.
- In pytorch, the smart augmentation model takes a lot of time to train the same number of epochs when compared to the non-smart augmentation model.
- The augmented images could be clearer in order to get better results from the classifier.
- The model seems to converge very soon at about 100 epochs and the accuracy could be increased if the model converges after 500 epochs.
- This model does not handle well the datasets with noise as seen the case with Stanford datasets. This could be avoided to make the model perform better in datasets with noise.

8.3 Research Contributions

- Chapter 1 and Chapter 2 give a great knowledge in understanding the entire concept of smart augmentation. It covers good points in understating this approach and could be implemented easily by the information. It also gives good understating on the pytorch framework and how it can motivate researchers to build more algorithms in same.
- Chapter 3 help us understand the potential of smart augmentation approach and how it can be used in other domains as well where there is requirement of a system to perform well on the data and reduce the overfitting problem.

- Any researcher can reproduce the results provided and the datasets tested can be easily understood in Chapter 4. In order to begin with any model implementation and testing it is very important to know the data and what kind of data is being tested.
- Chapter 5 gives the in and out of the entire implementation as well as the model on the whole. As said, any researcher could easily re-implement the model in this thesis and produce the results. It gives a detailed study on the approach, the technique involved and the pytorch framework details.
- Chapter 7 shows how smart augmentation is effective on few datasets under certain conditions of the hyper parameters.

8.4 Further Work

This model currently uses a single Network A (the Augmenter) which takes in images from both classes of the dataset. Current model in the thesis can be expanded to Multiple Neural networks A i.e. we can have a neural network for each class. For example the class 0 data can be augmented by NN A1 and the data of class 1 can be augmented by NN A2. This way each neural network learns the correct features from each class. The networks produce best augmentations of both the classes. This can significantly improve the performance of the model. Not all the results can improve, however we can expect some results to improve with this technique. Right now, the technique is only used to do classification tasks however we can extend it for Regression tasks as well and can be investigated how effective the approach is for regression tasks.

Appendix A

Code

The code written as part of the project is uploaded to github link
https://github.com/swatideshpande2228/Smart-Augmentation_MS_Thesis

There are two code files uploaded here,

- A python file with extension (.py), 'SmartAugmentation.py' contains the code for the Smart Augmentation technique with two neural networks.
- Another python file with extension(.py), 'Without_SmartAugmentation.py' contains the code without the approach to compare the techniques.

References

Hassan Al Hajj, Mathieu Lamard, Béatrice Cochener, and Gwenolé Quellec. Smart data augmentation for surgical tool detection on the surgical tray. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4407–4410. IEEE, 2017. 12

Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017. 6, 8, 14

Peter Corcoran, Claudia Costacke, Viktor Varkarakis, and Joseph Lemley. Deep learning for consumer devices and services 3—getting more from your datasets with data augmentation. *IEEE Consumer Electronics Magazine*, 9(3):48–54, 2020. 10, 15

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 113–123, 2019. 14, 15

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 7

Joseph Lemley and Peter Corcoran. Deep learning for consumer devices and services 4—a review of learnable data augmentation strategies for improved training of deep neural networks. *IEEE Consumer Electronics Magazine*, 9(3):55–63, 2020. 11, 15

Joseph Lemley, Shabab Bazrafkan, and Peter Corcoran. Smart augmentation learning an optimal data augmentation strategy. *Ieee Access*, 5:5858–5869, 2017. 3, 6, 7, 12, 15, 20, 30, 42

Joseph Lemley, Shabab Bazrafkan, and Peter Corcoran. Learning data augmentation for consumer devices and services. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–3. IEEE, 2018. 13

Ruihui Li, Xianzhi Li, Pheng-Ann Heng, and Chi-Wing Fu. Pointaugment: an auto-augmentation framework for point cloud classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6378–6387, 2020. 13, 15

Dmitry Molchanov, Alexander Lyzhov, Yuliya Molchanova, Arsenii Ashukha, and Dmitry Vetrov. Greedy policy search: A simple baseline for learnable test-time augmentation. *arXiv preprint arXiv:2002.09103*, 2020. 14

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015. 2

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019. 2