Image Augmentation for Improvement of Performance of Deep Learning Image Classifiers

Minh T. Nguyen

ECE 4580 - Digital Image Processing

Bradley Department of Electrical and Computer Engineering, Virginia Tech

Blacksburg, VA, 24060

mnguyen0226@vt.edu

Abstract

This paper deals with the comparisons of image augmentation techniques in large training and validating datasets generalization using for building more accurate deep convolutional neural networks (CNN) for image classification tasks. I deal with several image augmentation styles taught in ECE 4580 - Digital Image Processing in the Spring 2021 including horizontal flip, vertical flip, randomized rotation, shear, translation, ideal high-pass filter, ideal low-pass filter, Gaussian high-pass filter, Gaussian low-pass filter, Butterworth high-pass filter, Butterworth lowpass filter, adaptive median filter (AMF), histogram equalization, invert, and Canny edge detection. As a training agent, I will use ResNet9 trained and validated with FastAI's Imagenette 160-pixel dataset to provide the model's training and validating accuracies [5]. Next, I will generate additional datasets corresponding to each augmentation method by augmenting each image in the original dataset, appending it to a new empty dataset, and copy the original images to the same new dataset. Through training and validating all datasets, the research will show that vertical flip, vertical shear, and horizontal shear enhance the accuracy of classification task, while Butterworth lowpass filter, ideal lowpass filter, and Canny edge detection decrease the classification accuracy.

Keywords: image augmentations, Gaussian filter, Butterworth filter, adaptive median filter, Canny edge detection, robust convolution neural networks.

I. INTRODUCTION

Deep convolutional neural networks have been shown to perform remarkably well in image classification tasks in recent years. However, these networks require extremely large training sets to train effectively and avoid overfitting. This problem of generalization can be critical to certain tasks, where it is either not possible or not feasible to obtain a suitably large dataset [1]. For example, William & Mary College's Geo Fellowship undergraduate research required students to college Virginia's road roughness index by driving around Virginia in five different road types including excellent road, good road, medium road, bad road, and very bad road for satellite images classification tasks [2]. It was easy for student to collect excellent, good, and medium road since driving in city and country sides were a breeze; however, the data collection process for bad and very bad road was challenging which required students to drive to mountain to be able to collect dirt road. This caused an issue of non-uniformed training dataset, and image augmentations are the solution to this problem. There is a total of ten sections in my report. This paper will start by introducing the Imagenette-160 dataset, Residual Network 9 (ResNet9) model, and the Cloudera GPU used for training in the second section. The third section will present about image-affine methods including horizontal flip, vertical flip, random rotation, horizontal shear, and vertical shear. The fourth section will discuss about image frequency filter including both high-pass and low-pass of ideal filter, Gaussian filter, and Butterworth filter. The fifth section will touch on the intensity augmentation methods including AMF, history equalization, and intensity invert. The sixth section is dedicated for Canny edge detection. Next will present the results and the comparisons performance of ResNet9 model after trained with each augmented dataset. The eighth section will be used as a quick tutorial on how to rerun my project from GitHub. The ninth section will clearly state my contribution in this project. The tenth section will wrap up the paper with a concise conclusion.

II. IMAGENETTE-160 DATASET & CLOUDERA

A. Imagenette-160 Dataset

Imagenette is a subset of ten easily classified class from ImageNet including tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball and parachute [5]. There are two version of Imagenette that my team was interested in: "320 px" dataset and "160 px" dataset. I choose 160 px for this project due to the initial size of my CPU as well as the reduction in image-augmenting time and model-training time. The dataset also comes with a CSV file with 1%, 5%, 25%, and 50 of the labels randomly changed to an incorrect label [5]. However, I was not interested in this adversarial attack domain for the classification task. Two of the main reasons that I picked Imagenette dataset were due to its lack of data samples and the large amount of background noise. Specifically, Imagenette only contains around 14K images which fit with my research on building larger dataset with image augmentation for more exploratory analysis; and some images contain humans in their background, e.g person

https://github.com/mnguyen0226/image-augmentation-dnn-performance

holding a fish or a dog (Figure 1). This can potentially cause images misclassification. However, image augmentation techniques are potential ways to solve this problem.

Figure 1. Sample of a person holding a fish – background noise.

B. *ResNet 9 Model*

ResNet is a classic neural network used as a backbone for many computer visions tasks. This model was the winner of ImageNet challenge in 2015 [6]. The fundamental breakthrough with ResNet was it allowed researchers to train extremely deep neural networks with 150+ layers successfully. Prior to ResNet, training very deep neural networks was difficult due to the problem of vanishing gradients [4]. In short, the main based element of ResNet is the residual block. These layers put on top of each other, and every layer try to learn something underlying mapping of the desired function (Figure 2). Instead of having these block, researchers try and fit a residual mapping (Figure 3) [6].

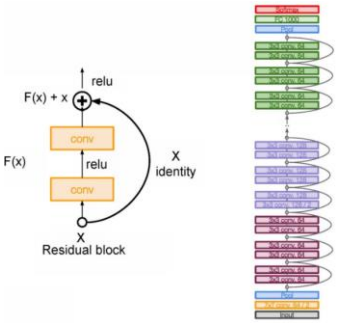

Figure 2. ResNet architecture and Layer Details.

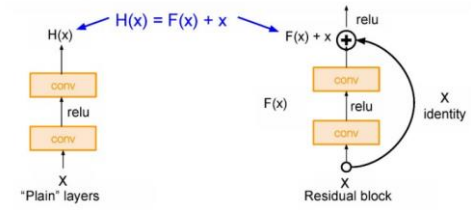

Figure 3. Plain Layers vs Residual Block.

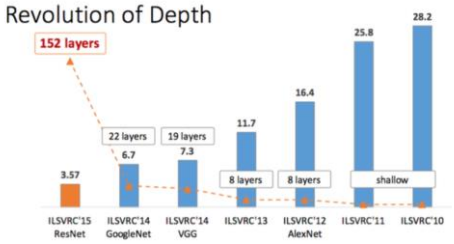

Figure 4. ImageNet Classification top-5 error (%).

Due to its robust performance (Figure 4), I chose ResNet as my based training model. Due to the initial computational power and limitation in time, I have decided to build ResNet 9 instead of doing transfer learnig ResNet 50 from Pytorch library. The model's summary can be found in (https://github.com/mnguyen0226/image-augmentation-dnn-performance/blob/main/classifier/model_summary.txt)

C. *Cloudera GPU / CPU*

Cloudera is an enterprise data cloud which provide the Cloudera's platform used for data analysis and machine learning to boost speed and yield insights from data via secure connection. Due to the need of a strong and reliable GPU / CPU, I rent Cloudera for couple of days for this project. Figure 5 shows my usage of Cloudera during image augmentations and model training sessions.

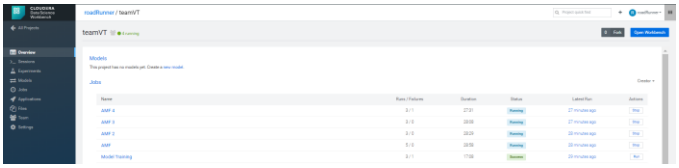

Figure 5. Cloudera GPU / CPU Training.

III. METHOD 1: IMAGE AFFINES

There are a total of six image affine augmentations including horizontal flip, vertical flip, rotation, horizontal shear, vertical shear, and translation.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix}$$

Figure 6. Transformation matrices of (a) horizontal flip, (b) vertical flip, (c) rotation, (d) horizontal shear, (e) vertical shear, and (f) translation respectively [7].

A. *Horizontal Flip*

The horizontal flip method can be found in **flip.py**. The method goes through each pixel of the input image and flip it horizontally by multiplying the pixel with the corresponding transformation matrix in Figure 6.

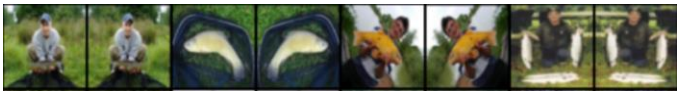

Figure 7. Original & horizontally flipped images.

B. *Vertical Flip*

The vertical flip method can be found in **flip.py**. The method goes through each pixel of the input image and flip it vertically by multiplying the pixel with the corresponding transformation matrix in Figure 6.

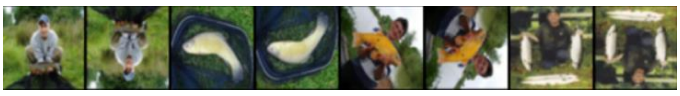

Figure 8. Original & vertically flipped images.

https://github.com/mnguyen0226/image-augmentation-dnn-performance

C. *Rotation*

The rotation method can be found in **rotation.py**. The method goes through each pixel of the input image and rotate it by multiplying the pixel with the corresponding transformation matrix in Figure 6. It is noted that I implemented this method to have "random" and "non-random" mode which allow researchers to choose to rotate image either randomly or a certain angle. For the sake of building a more diverse dataset, I randomize the rotation task.

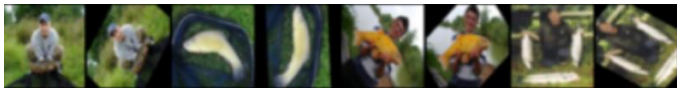

Figure 9. Original & randomly rotated images.

D. *Horizontal Shear*

The horizontal shear method can be found in **shear.py**. The method goes through each pixel of the input image and shear it horizontally by multiplying the pixel with the corresponding transformation matrix in Figure 6.

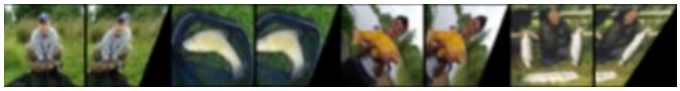

Figure 10. Original & horizontally sheared images.

E. *Vertical Shear*

The vertical shear method can be found in **shear.py**. The method goes through each pixel of the input image and shear it vertically by multiplying the pixel with the corresponding transformation matrix in Figure 6.

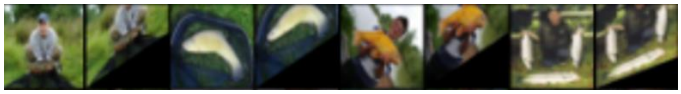

Figure 11. Original & vertically sheared images.

F. *Translation*

The translating method can be found in **translate.py**. The method goes through each pixel of the input image and translate it to random position by multiplying the pixel with the corresponding transformation matrix in Figure 6.

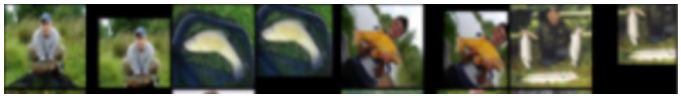

Figure 12. Original & translated images.

IV. METHOD 2: IMAGE FREQUENCY FILTERS

There are a total of six image frequency filters including ideal low-pass filter, Gaussian low-pass filter, Butterworth low-pass filter, ideal high-pass filter, Gaussian high-pass filter, Butterworth high-pass filter. It is noted that all six filters' transfer functions above are implemented in **lowpass.py** and **highpass.py**; however, the image processing of the filter will be managed through **frequency_filter.py**.

A. *Ideal Low-pass Filter*

Ideal low-pass filter method transfer function and its implementation can be found in **lowpass.py** and **frequency_filter.py** respectively. The cut-off frequency $D_0 = 8$ and transfer function of the ideal low-pass filter is [7]:

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

With various trials and errors, I found $D_0 = 8$ performed best.

Figure 13. Original & ideal low-pass filtered images.

B. *Gaussian Low-pass Filter*

Gaussian low-pass filter method transfer function and its implementation can be found in **lowpass.py** and **frequency_filter.py** respectively. The cut-off frequency $D_0 = 8$ and transfer function of the ideal low-pass filter is [7]:

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

With various trials and errors, I found $D_0 = 8$ performed best.

Figure 14. Original & Gaussian low-pass filtered images.

C. *Butterworth Low-pass Filter*

Butterworth low-pass filter method transfer function and its implementation can be found in **lowpass.py** and **frequency_filter.py** respectively. The cut-off frequency $D_0 = 8$, order $n = 2$ and transfer function of the ideal low-pass filter is [7]:

$$H(u,v) = \frac{1}{1 + \left[D(u,v)/D_0\right]^{2n}}$$

With various trials and errors, I found $D_0 = 8$ and order $n = 2$ performed best.

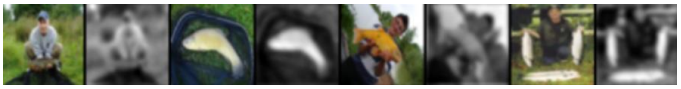

Figure 15. Original & Butterworth low-pass filtered images.

D. *Ideal High-pass Filter*

Ideal high-pass filter method transfer function and its implementation can be found in **highpass.py** and **frequency_filter.py** respectively. The cut-off frequency $D_0 = 8$ and transfer function of the ideal high-pass filter is [7]:

$$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \le D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases}$$

https://github.com/mnguyen0226/image-augmentation-dnn-performance

With various trials and errors, I found $D_0 = 8$ performed best.

Figure 16. Original & ideal high-pass filtered images.

E. Gaussian High-pass Filter

Gaussian high-pass filter method transfer function and its implementation can be found in **highpass.py** and **frequency_filter.py** respectively. The cut-off frequency $D_0 = 8$ and transfer function of the ideal high-pass filter is [7]:

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

With various trials and errors, I found $D_0 = 8$ performed best.

Figure 17. Original & Gaussian high-pass filtered images.

F. Butterworth High-pass Filter

Butterworth high-pass filter method transfer function and its implementation can be found in **highpass.py** and **frequency_filter.py** respectively. The cut-off frequency $D_0 = 8$, order $n = 2$ and transfer function of the ideal high-pass filter is [7]:

$$H(u,v) = \frac{1}{1 + \left[D_0 / D(u,v)\right]^{2n}}$$

With various trials and errors, I found $D_0 = 8$ and order $n = 2$ performed best.

Figure 18. Original & Butterworth high-pass filtered images.

V. METHOD 3: IMAGE INTENSITY TRANSFORMATIONS

There are a total of three image intensity transformations including AMF, histogram equalization, and intensity-invert.

A. Adaptive Median Filter

AMF implementation can be found at **amf.py**. The median filter performs well if the spatial density of the salt-and-pepper noise is low [7]. Also, the AMF seeks to preserve details while simultaneously smoothing non-impulse noise. Unlike other filter, the AMF increase the size of rectangular neighborhood during filtering [7]. The AMF algorithms uses two processing levels, denoted level A and level B, at each point (x,y) [7]:

Level A : If $z_{min} < z_{med} < z_{max}$, go to Level B
Else, increase the size of S_{xy}
If $S_{xy} \leq S_{max}$, repeat level A
Else, output z_{med}.

Level B : If $z_{min} < z_{xy} < z_{max}$, output z_{xy}
Else output z_{med}.

with elements denoted as:

z_{min} = minimum intensity value in S_{xy}
z_{max} = maximum intensity value in S_{xy}
z_{med} = median of intensity values in S_{xy}
z_{xy} = intensity at coordinates (x, y)
S_{max} = maximum allowed size of S_{xy}

It is noted that the AMF does not seems to change anything from the original images since there are minimum noise in the original images.

Figure 19. Original & AMF images.

B. Histogram Equalization

The histogram equalization method can be found in **hist_eq.py**. The method automatically determines a transformation that seeks to produce an image with a uniform histogram of intensity values. The result will be in erroneous color.

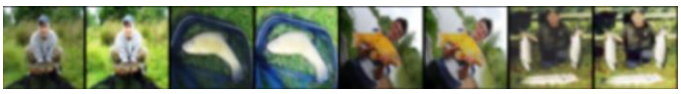

Figure 20. Original & histogram equalized images.

C. Intensity Invert

The intensity invert method can be found in **invert.py**. The method goes through each pixel of the input image and invert its intensity by letting 255.0 deducted from its current intensity.

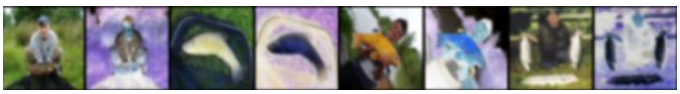

Figure 21. Original & intensity inverted images.

VI. METHOD 4: CANNY EDGE DETECTION

Although the algorithm is more complex, the performance of the Canny edge detection is superior in general to other edge detection methods [7]. The implementation of this method can be found at **canny_edge.py**.

The algorithms consist of four steps [7]:

1. Smooth the input image with the Gaussian filter.
2. Compute the gradient magnitude and angle images.
3. Apply nonmaximal suppression to the gradient magnitude image.

https://github.com/mnguyen0226/image-augmentation-dnn-performance

4. Use double thresholding and connectivity analysis to detect and link edges.

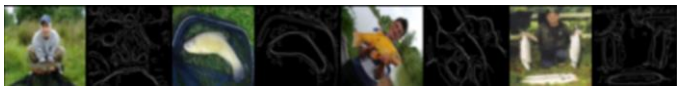

Figure 22. Original & Canny edge detected images.

VII. Experimental Results & Discussions

A. Results Table

The image augmentation methods were implemented to the original Imagenette-160 dataset to generate a new dataset. It is noted that these new datasets also include the original images. The table below showed the ResNet 9's performances, augmenting time, and training time among all newly generated datasets.

ResNet 9's Performances Comparison Table			
Datasets	***Latest Accuracy (%)***	***Data Augmenting Time on CPU (minutes)***	***ResNet 9 Training Time on GPU (minutes)***
Imagenette-160 Original	72.09	0	17
Imagenette-160 Horizontal Flip	62.24	45	17
Imagenette-160 Vertical Flip	73.54	45	17
Imagenette-160 Random Rotation	65.80	47	17
Imagenette-160 Horizontal Shear	72.16	47	17
Imagenette-160 Vertical Shear	73.41	45	17
Imagenette-160 Translation	63.22	48	17
Imagenette-160 Ideal Highpass	66.64	185	17
Imagenette-160 Ideal Lowpass	59.96	185	17
Imagenette-160 Gaussian Highpass	66.88	185	17
Imagenette-160 Gaussian Lowpass	63.71	185	17
Imagenette-160 Butterworth Highpass	62.55	185	17
Imagenette-160 Butterworth Lowpass	59.94	185	17
Imagenette-160 AMF	69.44	595	17
Imagenette-160 Hist Equal	68.30	46	17
Imagenette-160 Intensity Invert	67.62	43	17
Imagenette-160 Canny Edge Detect	61.99	130	17

Table 1. ResNet 9's Performances Comparison Table.

It is noted that the choice of 30 epochs was arbitrary. The accuracy at epoch 30 does not mean that it is the best accuracy. Please check the GitHub repository for best accuracy information of each trained model.

B. Discussion on top 3 best-performed ResNet models

According to Table 1, the top three best performance augmentation methods that provide the best ResNet 9 classifiers are (1) Imagenette-160 Vertical Flip, (2) Imagenette-160 Vertical Shear, and (3) Imagenette-160 Horizontal Shear.

Below are the performance graphs of the top three models. Their accuracies are all higher than the original's one. This makes sense since the dataset of all top three performers were double the size of the original data set.

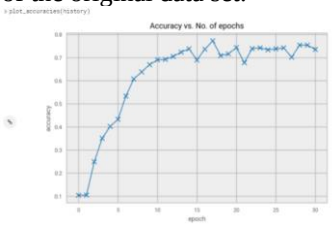

Image 23. Accuracy vs Epoch of Vertical Flip ResNet 9.

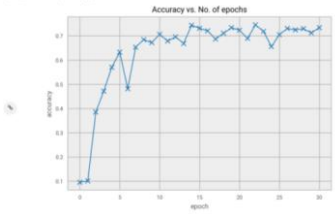

Image 24. Accuracy vs Epoch of Vertical Shear ResNet 9.

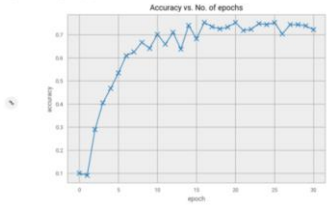

Image 25. Accuracy vs Epoch of Horizontal Shear ResNet 9.

These results provide me the observation that by flipping or shearing, the model has learned edges that it has not seen before, thus provide slightly higher accuracies.

C. Discussion on top 3 worst-performed ResNet models

According to Table 1, the top three worst performance augmentation methods that provide the best ResNet 9 classifiers are (1) Imagenette-160 Lowpass Butterworth, (2) Imagenette-160 Ideal Lowpass Filter, and (3) Canny Edge Detection.

Below are the performance graphs of the worst three models. Their accuracies are all lower than the original's one. This makes sense since the datasets trained for the model were filled with either noises or images that lost information.

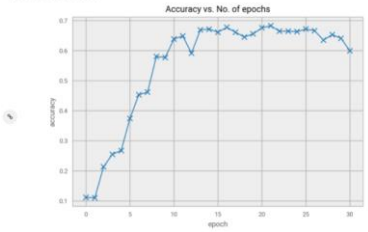

Image 26. Accuracy vs Epoch of Butterworth Lowpass ResNet 9.

https://github.com/mnguyen0226/image-augmentation-dnn-performance

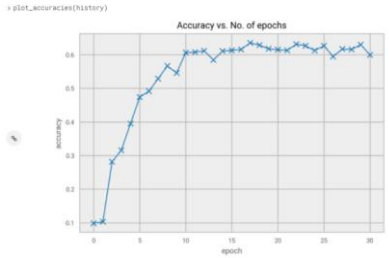

Image 27. Accuracy vs Epoch of Ideal Lowpass ResNet 9.

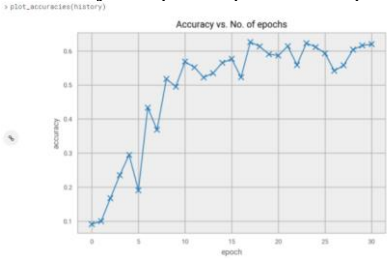

Image 28. Accuracy vs Epoch of Canny Edge ResNet 9.

These results provide me the observation that for the two low-pass filters, they have corrupted the clean original dataset with corrupted noise since lowpass filters were used for blurring images. Besides, the Canny edge detection has deleted a lot of information in the images (and provided with just edges), thus the trained model was also corrupted. Moreover, for all three methods above, after augmented, the images have been gray scaled which also contribute to the misclassification.

D. Discussion training time and augmenting time

From Table 1, it is easy to notice that the training times for the models are reasonable – 17 minutes. This is due to the used of the GPU for training models. Besides, the augmenting times for the new dataset are very large, especially AMF. Although I used a lot of CPU power to augment, the fact that the code for these augmenting methods are written from scratch by me (instead of using OpenCV or Sklearn built-in function) had cause the computing time to increase exponentially. However, it is necessary for me as students to implement these image processing methods, thus, the time trade-off is worthy.

VIII. How To Preproduce Results

A. Augment images and train models

Due to the method was written from scratch and the Imagenette-160 dataset is quite large, it is recommended that you have CUDA v10.0, Pytorch v1.2.0, and Cloudera.

For image-augmentation task, follow these steps:

1. Make sure that your machine is at least 2 vCPU / 4Gib Memory.
2. Set up the right image-input path and image-save path in **process_image.py**.
3. Choose the augmentation methods that you prefer in **process_image.py**.
4. Run **python preprocess_image.py.**

For model-training task, follow these steps:

1. Make sure that your machine is at least 2 vCPU / 4Gib Memory with 1 GPU.
2. Choose the augmented dataset that you want in **model_train.py**.
3. Choose the path that you want to save your model in **model_train.py**.
4. Run **python model_train.py**.

The pretrained models and augmented datasets are also provided in the GitHub page of this paper.

B. Test augmentation methods

To validate the image augmentation methods only (with 1 image), run **python test_image_aug.py**.

IX. My Contribution

Although the project is a teamwork, due to conflict in time (due to midterms and early finals in some classes) and implementing ways, my teammate and I split up doing way of testing the network's performance with the same dataset and some similar augmentation methods. I did this project by myself and write mostly of my own code.

In terms of code, I wrote everything in this GitHub by myself but the "intensity" section where I had to use my teammate's code.

In terms of preprocessing data, creating model, training models, testing models, and setting up the GitHub repository I did all by myself.

In term of writing the final report, I did all by myself.

X. Conclusion

In conclusion, vertical flip, vertical shear, and horizontal shear are the top three augmentation methods, while Butterworth lowpass filter, ideal lowpass filter, and Canny edge detection are the bottom three augmentation methods for populating small dataset in image classification task with deep neural networks (ResNet 9). It is noted that this research only focused on ResNet 9, thus there are unlimited more models to discover to future researchers.

References

[1] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," Journal of Big Data, vol. 6, no. 1, 2019.

[2] D. Runfola, "2020 - 2021 geoFellowship," *William & Mary*. [Online]. Available: https://www.wm.edu/as/data-science/researchlabs/geolab/cci/index.php. [Accessed: 07-May-2021].

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv.org, 10-Dec-2015. [Online]. Available: https://arxiv.org/abs/1512.03385. [Accessed: 06-May-2021].

[4] K. Patel, "Architecture comparison of AlexNet, VGGNet, ResNet, Inception, DenseNet," *Medium*, 08-Mar-2020. [Online]. Available:

https://github.com/mnguyen0226/image-augmentation-dnn-performance

https://towardsdatascience.com/architecture-comparison-of-alexnet-vggnet-resnet-inception-densenet-beb8b116866d. [Accessed: 08-May-2021].

[5] Fastai, "fastai/imagenette," GitHub. [Online]. Available: https://github.com/fastai/imagenette. [Accessed: 08-Apr-2021].

[6] P. Dwivedi, "Understanding and Coding a ResNet in Keras," *Medium*, 27-Mar-2019. [Online]. Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33#:~:text=ResNet%20is%20a%20powerful%20backbone,mitigate%20the%20vanishing%20gradient%20problem. [Accessed: 08-May-2021].

[7] R. C. Gonzalez and R. E. Woods, *Digital image processing*. New York, NY: Pearson, 2018.