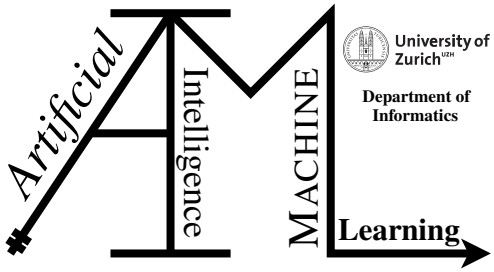
Data Augmentation for Improved Classification in Neural Networks

Master Thesis

Isabel Margolis
13-776-281

Submitted on
March 30 2021

Thesis Supervisor
Prof. Dr. Manuel Günther



Master Thesis

Author: Isabel Margolis, isabel.margolis@uzh.ch

Project period: October 1 2020 - March 31 2021

Artificial Intelligence and Machine Learning Group
Department of Informatics, University of Zurich

Acknowledgements

First and foremost, I would like to thank my thesis supervisor, Prof. Dr. Manuel Günther, for the opportunity to write my thesis within the Artificial Intelligence and Machine Learning Group at the University of Zurich and for his guidance and support through each stage of the process. I would like to thank my partner, Vinzenz, for his endless help and support throughout my studies. And to my father for inspiring my interest in machine learning and for his final review and proofreading. Lastly, I am thankful to everybody who has helped and supported me during my studies at the University of Zurich.

Abstract

Image classification models are often trained on additional transformations of the original images to increase the number and the variance of the training set. These transformations can include, for example, horizontal flips, rotations, and color transformations. Most of the recent data augmentation methods focus on improving data augmentation techniques during the training phase. There is very little research on test-time augmentation, which contains data augmentation during the testing phase. Often, only very simple transformations are used during testing, and the predictions are frequently aggregated through simple averaging or majority voting. I propose new aggregation methods to improve the predictive performance of image classification models during the testing phase. I show on four different datasets, namely CIFAR-10, MNIST, ImageNet, and a dataset containing images for melanoma classification, that other aggregation methods, besides the commonly applied averaging and majority voting, obtain better results.

Zusammenfassung

Um die Leistung von Bildklassifikationsmodellen zu verbessern, werden die Bilder beim Trainieren eines Modells oft durch verschiedene Transformationen vermehrt. So kann man die Anzahl und die Varianz der Bilder vergrössern, ohne zusätzliche Daten sammeln zu müssen. Solche Transformationen können zum Beispiel Rotationen, horizontale Verschiebungen und Farbveränderungen beinhalten. Oft führt dies zu besseren Ergebnissen und ist eine weit verbreitete Technik. Während die Augmentation der Bilder während der Trainingsphase bereits ausgiebig erforscht wurde, existieren nur wenige Arbeiten, die sich auf die Testphase beziehen. Oft werden nur sehr einfache Transformationen während dieser Phase angewendet und die gewonnenen Prognosen auf den Testbildern werden meistens durch Mittelwertbildung oder Mehrheitsentscheidung aggregiert. In dieser Arbeit zeige ich auf vier verschiedenen Datensätzen, CIFAR-10, MNIST, ImageNet und ein Datensatz mit Bildern zur Melanomklassifizierung, dass die Augmentation von Bildern während der Testphase verbesserte Resultate erbringen. Zusätzlich zeige ich, dass neue Methoden zur Aggregierung bessere Prognosen erzielen als die herkömmlichen Methoden.

Contents

Chapter 1

Introduction

Many image recognition applications have emerged with the development of deep learning, such as self-driving cars, autonomous robots, and facial recognition. There are many benefits to improving the capabilities in image recognition. For example, self-driving cars must be able to detect traffic lights and other objects correctly. In recent years, medical image recognition has become a very important application. Deep learning models are increasingly used to help recognize diseases in medical images, such as diagnosing brain diseases in brain MRI images (Cai et al., 2020). In this thesis, I aim to improve the accuracy of image classification models. Thus, the application of image classification on medical images is particularly interesting because even a tiny improvement in the accuracy can have life-and-death consequences.

Data augmentation is often used to improve the accuracy of image recognition tasks. Images are transformed to augment the current dataset without collecting new images. At the same time, they help increase the diversity of the data. Common examples of transformations include rotating, shifting, and cropping the original images. Machine learning algorithms are, in general, more effective the more data they have access to. This statement holds, even if the data is of lower quality (Perez and Wang, 2017). More data and increased diversity in the data reduce the chances of overfitting to the training set (Takahashi et al., 2020). Overfitting happens when the model performs very well on the training set but not on the validation and test set. In other words, the model does not generalize well to unseen data (Shorten and Khoshgoftaar, 2019). More specifically, it can be used as a regularization method. Compared to other regularization methods, such as dropout in neural networks, data augmentation does not require changing the network structure. Instead, it only operates on the input (Chen et al., 2020). Thus, data augmentation is especially beneficial when the amount of available data is small. This is often the case with medical data, such as cancer images, due to the rarity of diseases, patient privacy, the demand for expert labeling, and the cost to create and process the images (Shorten and Khoshgoftaar, 2019).

There exist several ways to use data augmentation for image classification tasks. In general, it can be divided into two stages: data augmentation during the training stage and data augmentation during the testing stage. The latter is also called test-time augmentation. Recent research focuses a lot on improving data augmentation techniques during training. These improvements often involve automatic choices of transformations and new transformation techniques. The goal is to find the ideal set of transformations such that the trained model performs best during the training stage. Along with data augmentation during the training stage, test-time augmentation can also be performed. However, there is very little research that focuses on this particular problem. Some papers mention that they have used test-time augmentation to boost the performance of their image classification model. But only very few papers propose a novel technique or test new methods. The goal of test-time augmentation is to improve the image classification score for the unseen test images. Instead of predicting the class of an image solely on the original image, it is augmented several times, creating multiple images. Then the predictions of each separate im-

age are aggregated to form the final class prediction. Common aggregation methods are simple averaging of the class probabilities and majority voting. The aim of this thesis is to investigate the effects of test-time augmentation and propose new aggregation methods.

1.1 Motivation

There is very little research on test-time augmentation, and besides simple averaging and majority voting, there are not many different propositions on how to aggregate the predictions during testing. Because other research has demonstrated that test-time augmentation has a positive impact on the accuracy of image classification models, and because new aggregation methods are rarely explored, this thesis aims to investigate and propose new methods for improving test-time augmentation. Additionally, although many well-performing classification models already exist that use advanced data augmentation during training, their overall accuracy on the test set may be further improved by improving test-time augmentation techniques.

1.2 Description of Work

I use four different datasets, namely CIFAR-10, MNIST, ImageNet, and a dataset containing images for melanoma classification, which from now on I will refer to as Melanoma, for evaluating different test-time augmentation methods. I choose the datasets such that any results in this thesis can likely be generalized to a large set of different datasets and unique classification problems. In the first stage, I train a neural network on each dataset using simple data augmentation methods. In the second stage, I implement and evaluate various data augmentation methods during the testing phase. Finally, in the third stage, I compare and discuss results found on the test sets of each dataset. The goal is to find a method that works well on all datasets and can likely be generalized and implemented in many other image classification problems.

1.3 Thesis Outline

This thesis is structured as follows:

- **Datasets:** In this chapter, I will give an overview of the four datasets that I use to evaluate various test-time augmentation methods. These four datasets include CIFAR-10, MNIST, ImageNet, and Melanoma.
- **Related Work**: I will give an overview of transformation techniques commonly used and provide some background regarding data augmentation in general. Additionally, I will introduce related work concerning data augmentation during the training stage and test-time augmentation.
- **Data Augmentation during the Training Stage:** Here, I will explain how I train the neural networks, which I will later use to compare test-time augmentation methods.
- **Test-Time Augmentation:** Test-time augmentation is the main focus of this thesis. In this chapter, I will explain in detail the various methods that I implement and evaluate.
- **Results:** In this chapter, I will share all the results from my experiments. Additionally, I will provide results from Welch's t-test to underline the statistical significance of my results.

- **Discussion:** I will discuss the results found in the previous chapter, discuss attempts to improve these results and bring forth potential limitations in my approach.
- **Conclusion and Future Work:** Finally, in this last chapter, I will sum up the results found in this thesis and provide some insight into areas where my research can be further improved or where it has remained inconclusive.

Chapter 2

Datasets

There exist several publicly available datasets containing labeled images. Each image can be represented as an array of the shape $D \times W \times H$, where D represents the color dimension, and W and H represent the width and height of the image in pixels. If I only indicate two dimensions, then I am referring to W and H. The first dimension, representing the color, should be clear from the context. If D is 3, then the color of the image is represented in the RGB color scheme. In general, any color can be created by combining red, green, and blue. The values for each color range between 0 and 255 and are scaled to be between 0 and 1. If D is 1, then the image is grey-scaled. The pixel values also range between 0 and 255 and are scaled to be between 0 and 1. 0 indicates black, and 1 indicates white.

I mostly use CIFAR-10[1], described in Section 2.1, for my investigations of data augmentation techniques and methods to improve image classification. I choose CIFAR-10 because it is a sufficiently small dataset that allows me to run tests without massive computation times. The ImageNet 2012 Classification Dataset, described in Section 2.3 has 1000 different categories, whereas CIFAR-10 only has ten different categories. The ImageNet set has about 25 times more images for training than CIFAR-10. Additionally, the resolution of the images belonging to ImageNet is much higher than the images of CIFAR-10. The images in CIFAR-10 are all equally sized with a resolution of 32×32 pixels. ImageNet, on the other hand, contains images of different sizes with an average resolution of about 500×400 pixels. However, to ensure that my discoveries can be generalized to other images and datasets, I also test my results on all the datasets described in this chapter.

I load the datasets using the `DataLoader`[2] class from PyTorch. CIFAR-10, MNIST, and many other datasets are included in `torchvision.datasets` and can easily be loaded. For the ImageNet 2012 Classification Dataset, `torchvision.datasets` only includes the protocol of how to split the dataset into training, validation, and test sets. The data, however, has to be downloaded from an external source.[3] Melanoma[4], described in Section 2.4, is considered a special case because it is a very specific application of image recognition where there are only two possible outcomes that are extremely unbalanced. Additionally, Melanoma is not included in `torchvision.datasets`, and it brings about further challenges that are not present in the other datasets. However, it is an interesting case of image classification, where data augmentation techniques may have many potentials, and I, therefore, decided to run my experiments on it. In general, I put importance on testing all methods in this thesis on various datasets. Each of the four datasets that I use is unique in some way and allows me to understand how well my methods generalize among various datasets and make meaningful conclusions.

[1] https://www.cs.toronto.edu/~kriz/cifar.html
[2] https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader
[3] http://www.image-net.org/challenges/LSVRC/2012/index
[4] https://doi.org/10.34970/2020-ds01

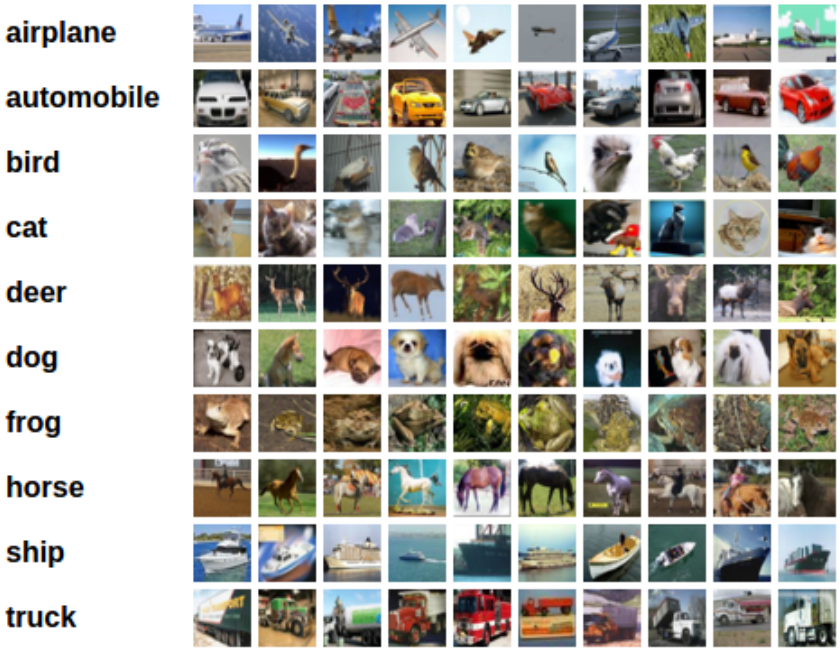


Figure 2.1: CIFAR-10 EXAMPLE. 10 random images for each class in the CIFAR-10 dataset. *Source: (Krizhevsky, 2012)*

2.1 CIFAR-10

CIFAR-10 contains 60'000 labeled 32×32 color images. There are ten possible labels: *plane*, *car*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. Figure 2.1 gives an example of 10 random images per label. The dataset is publicly available, and it is split into 50'000 training images and 10'000 test images. The labels of the test set are balanced; thus, there are precisely 1'000 images per category. Furthermore, the labels are mutually exclusive, meaning that no image can simultaneously belong to more than one category. In addition to CIFAR-10, there also exists the CIFAR-100 dataset. It is essentially the same as CIFAR-10, except that there are 100 different categories (Krizhevsky, 2012). I do not run my experiments on CIFAR-100 because of its similarity to CIFAR-10 and ImageNet. I do not suppose that additional experiments on CIFAR-100 would provide any novel insights for this thesis. For implementing my experiments and testing new ideas, I always use CIFAR-10 because of its small size and simplicity.

2.2 MNIST

MNIST[5] contains handwritten black and white digits from 0 to 9. The training set consists of 60'000 images, and the test set contains 10'000 images. The digits were centered and fit into a 28×28 field. It is a popular database to learn image classification techniques because there is very little preprocessing and formatting required (LeCun and Cortes, 2010). It is used extensively in optical character recognition and machine learning research. Additionally, because the task of

[5] http://yann.lecun.com/exdb/mnist/

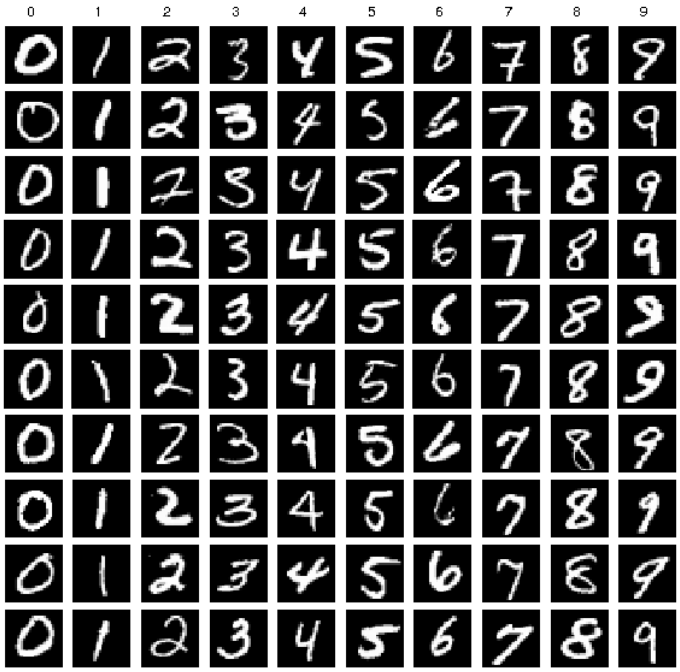


Figure 2.2: MNIST EXAMPLE. 10 random images for each class in the MNIST dataset. *Source: (Lim et al., 2016)*

classifying handwritten digits is somewhat more straightforward than most other image recognition tasks, MNIST is often used for fast-testing machine learning algorithms (Deng, 2012). In comparison to the other datasets mentioned in this chapter, MNIST contains grey-scale images. Hence, the first dimension, D, of the image is 1. Figure 2.2 shows a few examples from the MNIST dataset. The digits are all centered and scaled such that they have similar sizes. I include this dataset because it is quite different from the other datasets mentioned in this chapter. It is a very specific image recognition application that is easier than image recognition using CIFAR-10, ImageNet, or Melanoma. I am interested in whether data augmentation methods that prove helpful for more difficult classification tasks help classify digits as well.

2.3 ImageNet

ImageNet is an extensive collection of human-annotated photographs intending to further the research and the development of computer vision algorithms (Deng et al., 2009). It consists of 27 high-level categories and 21'841 subcategories. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC2012) contains images from ImageNet with 1000 different labels. ILSVRC2012 is one of the annual challenges to provide a benchmark for state-of-the-art algorithms. There are 1'281'167 images for training with 732 to 1300 images per label. The validation set consists of 50'000 images, 50 images per label, and the test set consists of 100'000 images with 100 images per label. However, the labels of the test set are not publicly available (Russakovsky et al., 2015). I use the given validation set as my test set and randomly split the given training set into a training and a validation set, with 1'181'167 training images and 100'000 validation images. Whenever I mention ImageNet in my results and discussions, I always mean the dataset of ILSVRC2012.

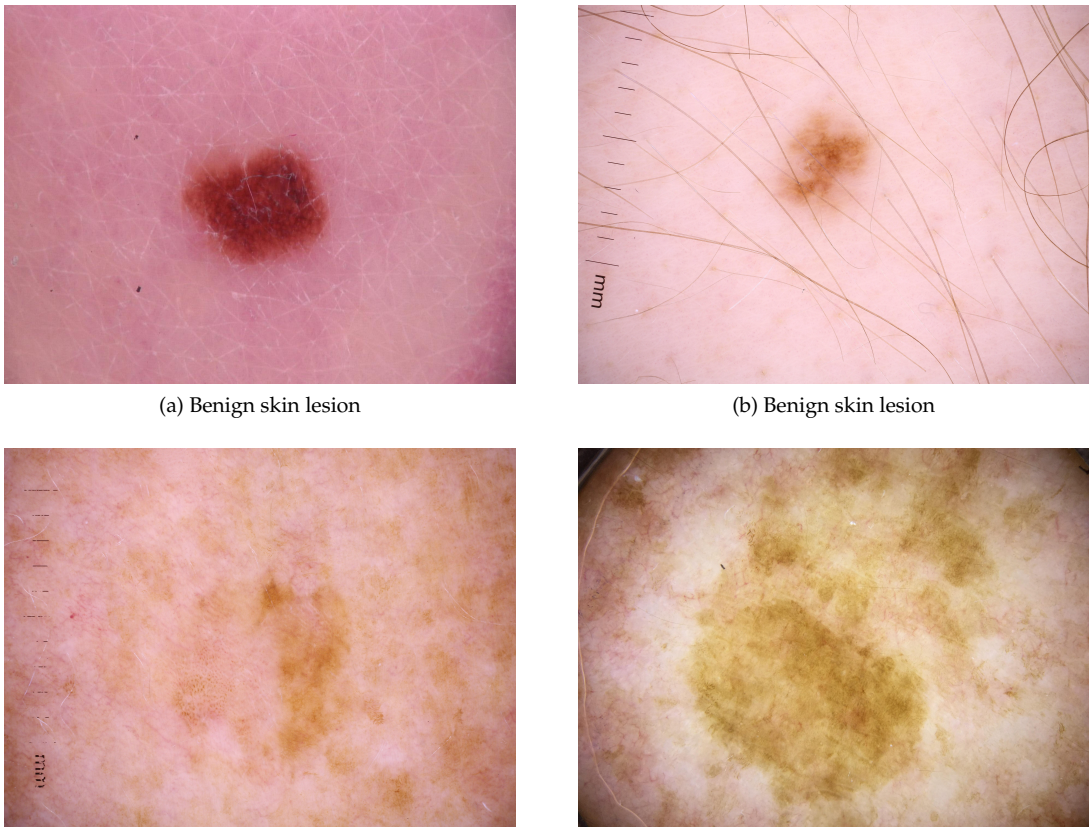
In contrast to CIFAR-10 and MNIST, the images in ImageNet come in different sizes and shapes. Because most neural networks require all images to be of the same size, I must reshape each image before passing it to the model. The difficulty with ImageNet is mainly its size and number of possible labels. Using such a large number of training images forces me to consider computational efficiency aspects. Although some methods might improve the image recognition accuracy, I sometimes opt for the worse but more computationally efficient method. ImageNet is also a useful dataset to ensure that specific methods that perform well on datasets with only a few labels also perform well on datasets with many different labels.

2.4 Melanoma

Melanoma belongs to the ISIC 2020 Challenge and has also been a Kaggle[6] competition that ended in August 2020. It contains 33'126 training images of skin lesions. The goal of the challenge is to identify whether the patient suffers from malignant melanoma, which is a serious form of skin cancer that develops from pigment-producing cells. The dataset was generated by the International Skin Imaging Collaboration (ISIC). Along with the images, an additional file with information on patient ID, sex, age, and general anatomic site, can be downloaded (Rotemberg et al., 2021). Medical images are especially interesting for this thesis because the amount of data is often small, and data augmentation has, therefore, a significant impact. Additionally, good image classification models for medical data are in big demand because they could potentially improve the diagnosis of diseases that are otherwise difficult to detect.

Figure 2.3 shows examples of images in Melanoma. Figure 2.3a and 2.3b are examples of benign skin lesions. Figure 2.3c and 2.3d are examples of malignant melanoma. Classifying malignant melanoma is a very challenging image classification task because compared to the other datasets, it is even challenging for a human to identify differences between benign and malignant skin lesions.

[6] https://www.kaggle.com/c/siim-isic-melanoma-classification



(a) Benign skin lesion

(b) Benign skin lesion

(c) Malignant melanoma

(d) Malignant melanoma

Figure 2.3: MELANOMA EXAMPLE. There are two possible labels for each image in Melanoma: benign skin lesion and malignant melanoma. (a) and (b) are examples of benign skin lesions. (c) and (d) are examples of malignant melanoma.

Chapter 3

Related Work

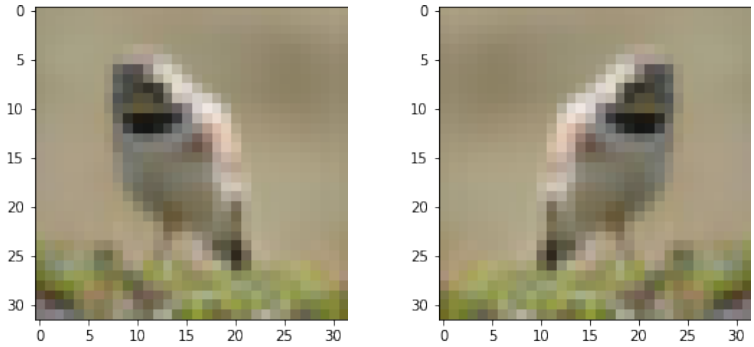
A lot of research has been done on data augmentation techniques and how they improve image classification. Some techniques, such as cropping and flipping, are very common. But there exist more advanced and less common techniques as well. This chapter will introduce the most common data augmentation techniques and a few promising advanced techniques. Additionally, I will give an overview of what has been done in terms of data augmentation during the training stage and test-time augmentation. Data augmentation during training is very common and is almost always used as an additional measure to improve image classification accuracy. There are a few standard approaches and related work for test-time augmentation; however, this is much less common than data augmentation during training. Not many papers are focused on new and improved methods for test-time augmentation.

3.1 Common Data Augmentation Techniques

In this section, I will introduce the standard transformations that are almost always used for data augmentation during the training stage of an image classification model. Although these techniques are very simple, many of these techniques have substantially improved image classification. They are commonly used to increase the number of training samples or to add additional variety to the set of training images. They are popular because they are easy to understand and implement, and the implementation code is often available in most deep learning frameworks (Mikołajczyk and Grochowski, 2018). Additionally, all these standard augmentation techniques can be applied sequentially to an image, thus creating a more extensive set of possible transformations.

3.1.1 Geometric Transformations

Geometric transformations are transformations that change the geometric shape of the image. These transformations include flipping, rotating, shifting, and cropping of an image. In general, these transformations help the model not to rely on the object's location, shape, and angle within the image. For example, in many image datasets, the relevant object is in the middle of the image. However, an unseen image might contain the object in a corner or near the image border. Thus, for the model to correctly classify those images, it is helpful to augment the training set such that it contains the object in varying positions and sizes.



(a) Original image

(b) Transformed image after horizontal flipping

Figure 3.1: HORIZONTAL FLIP. Example of horizontal flipping on a bird image from CIFAR-10. (a) shows the original image and (b) shows the image after applying horizontal flipping.

Flipping

Flipping can be done both vertically and/or horizontally. Horizontal flipping has proven useful on many datasets such as CIFAR-10 and ImageNet (Shorten and Khoshgoftaar, 2019). Figure 3.1 shows an original image from CIFAR-10 and the result of horizontally flipping the image. Depending on the image, it might be inappropriate to use flipping. For example, on MNIST, which contains images of numbers, flipping those images could potentially change their labels. Vertical flips are very rarely appropriate because most objects would never appear vertically flipped in real life. Huang et al. (2017) demonstrate on CIFAR-10 and CIFAR-100 that adding horizontal flips during training improves the error rate of many popular neural networks.

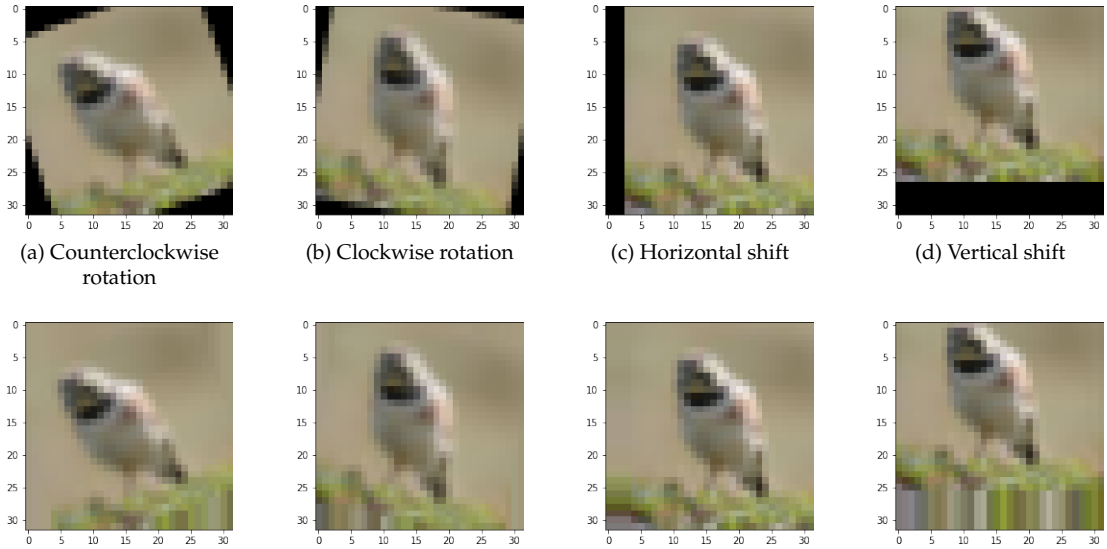
Rotation

Rotating augmentation is done by rotating the image to a specific degree. Depending on the rotation degree, this technique could also change the image such that it is no longer label-preserving (Shorten and Khoshgoftaar, 2019). Figure 3.2a and 3.2b show examples of rotating an image in clockwise and counterclockwise direction. 3.2a is rotated by 20 degrees in a counterclockwise direction, and 3.2b is rotated by 10 degrees in a clockwise direction.

Shift/Translation

Shifting is done by moving the image either up, down, left, or right while preserving its size. Shifting is very useful because the position of the object varies. This can help avoid positional bias in the data. For example, in facial recognition tasks, the sample images are often centered. Without shift augmentation, the model might only recognize a face in the center of the image (Shorten and Khoshgoftaar, 2019). Figure 3.2c and 3.2d show examples of shift transformations. 3.2c is shifted horizontally, and 3.2d is shifted vertically.

When rotating or shifting an image missing pixels will appear. An example of this can be seen in Figure 3.2. The top row has black areas where the image is undefined. There exist several methods on how to fill the unknown space. For example, the missing pixels can be filled with a constant value or an interpolation function. Figures 3.2e, 3.2f, 3.2g, and 3.2h show the above



(a) Counterclockwise rotation (b) Clockwise rotation (c) Horizontal shift (d) Vertical shift

(e) Counterclockwise rotation with interpolation (f) Clockwise rotation with interpolation (g) Horizontal shift with interpolation (h) Vertical shift with interpolation

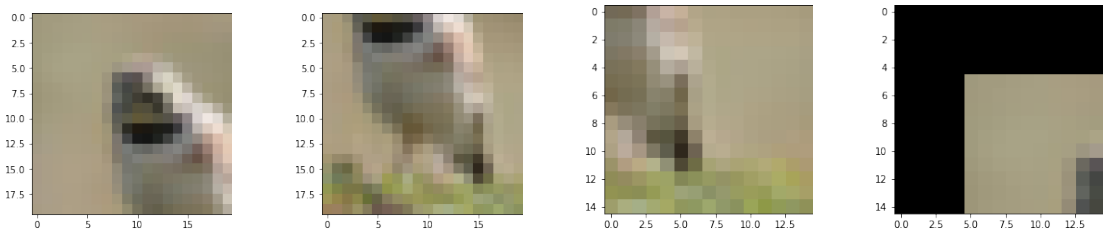
Figure 3.2: ROTATIONS AND SHIFTS. Examples of rotations and shifts with different directions. (a) and (b) show rotations in counterclockwise direction and clockwise direction, respectively. (c) shows a horizontal shift to the right. (d) shows a vertical shift up. (e), (f), (g), and (h) show the transformed images after interpolating the missing pixels from the images above.

images after using the function `extrapolate_mask`[1] from the bob.ip.base package, which interpolates the missing pixels using a nearest neighbors approach.

Cropping

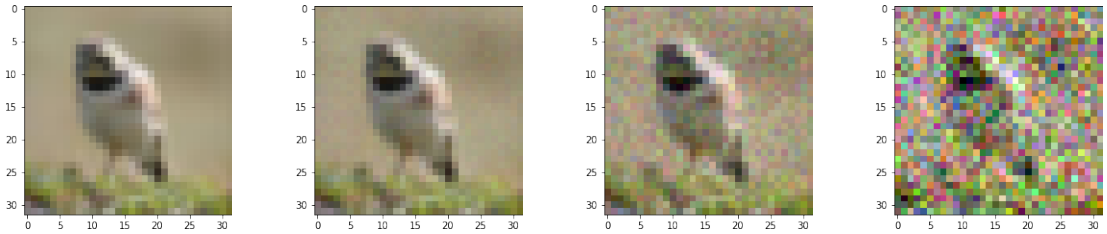
Random cropping is a data augmentation technique where a random crop of either fixed or random size is taken from the original image. Because objects that we want to identify in image recognition are not always fully visible in the image, it helps to add random crops to the training set. The model learns to identify subsets of an object correctly as well. A concern with random cropping is that there is a chance that the actual object to be classified might not appear in the chosen cropped image. Instead, there will only be background information present. Figure 3.3 shows examples of different crops of the original bird image. Depending on the crop size and the position, the bird might be difficult to classify. Figure 3.3d gives an example of a crop that overlaps the borders, which causes the cropped image to contain undefined pixels. The undefined pixels can be filled. However, the main object to classify is hardly in the cropped image anymore because of the large overlap. Additionally, there are other cropping techniques besides random cropping. For example, center cropping, which takes a crop from the center of the image.

[1] https://www.idiap.ch/software/bob/docs/bob/bob.ip.base/stable/py_api.html\#bob.ip.base.extrapolate_mask



(a) (b) (c) (d)

Figure 3.3: RANDOM CROPS. Examples of random crops from the same image. (a), (b), and (c) show examples of different crop positions and sizes. (d) shows an example where the cropped image goes beyond the borders, resulting in undefined pixels.



(a) $\sigma = 0$ (b) $\sigma = 0.01$ (c) $\sigma = 0.04$ (d) $\sigma = 0.14$

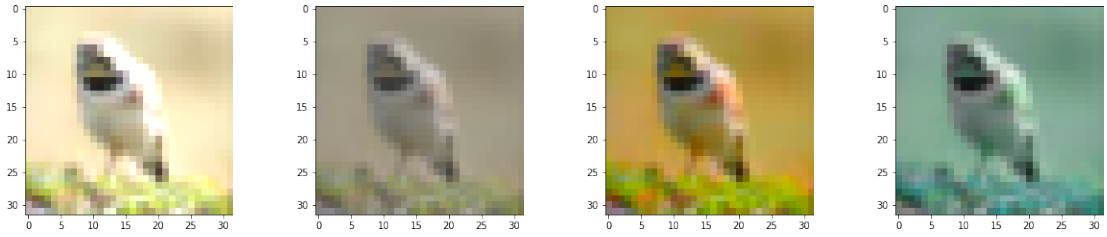
Figure 3.4: NOISE INJECTION. Examples of noise injections from a Gaussian distribution with varying standard deviations. (a) is the original image without Gaussian noise. (b), (c), and (d) contain Gaussian noise with $\sigma = 0.01$, $\sigma = 0.04$, and $\sigma = 0.14$, respectively.

3.1.2 Noise Injection

Noise injections are usually drawn from a Gaussian distribution (Shorten and Khoshgoftaar, 2019). Injecting the training images with noise can be beneficial because images are often corrupted by noise, thus, if a model is trained to consider this, it may perform better on unseen images. Figure 3.4 shows examples of injecting the original image with noise from a Gaussian distribution. The higher the standard deviation σ, the noisier the image. Images are often represented with scaled pixel values between 0 and 1. Therefore, the values for σ are chosen correspondingly. Figure 3.4a shows the original image with no noise injection. Figure 3.4b shows the augmented image with noise injection with $\sigma = 0.01$. Figure 3.4c corresponds to $\sigma = 0.04$ and Figure 3.4d to $\sigma = 0.14$.

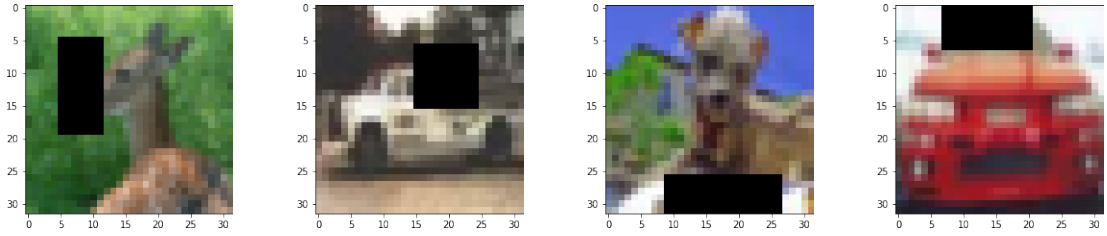
3.1.3 Color Space Transformations

Images are often represented in the RGB color model. Color space augmentations are achieved by changing the intensities of the channels. For example, this includes isolating a single color channel by setting the other two channels to zero (Shorten and Khoshgoftaar, 2019). Similar to geometric transformations, there are also many possibilities for changing the color space of an image. Figure 3.5 shows examples of changing the brightness, contrast, hue, and saturation of an image. These four color transformations can also be combined with each other. For images that are grey-scaled, not all color transformations are useful. The change in hue and saturation, for example, do not impact grey-scaled images. However, changes in brightness and contrast can be



(a) Increased brightness (b) Reduced contrast (c) Increased saturation (d) Change of hue

Figure 3.5: COLOR SPACE TRANSFORMATIONS. Transformed images after changing the color space. (a) contains increased brightness, (b) has reduced contrast, (c) has increased saturation, and (d) has a different hue.



(a) (b) (c) (d)

Figure 3.6: RANDOM ERASING. Examples of random erasing on images from CIFAR-10. Each patch is exactly 1/10 the size of the original images. The position is chosen randomly, and the shape of the rectangle is chosen randomly from a given range of aspect ratios.

applied.

3.1.4 Random Erasing

Random erasing is done by randomly placing a rectangle patch on the image and masking those pixels with either a constant value or random pixels (Shorten and Khoshgoftaar, 2019). This method is useful because the model is forced to concentrate on the entire image, not only a subset of the image. This technique has been shown to produce high accuracies on various datasets (Zhong et al., 2020). However, random erasing might not always be label-preserving depending on the images (Shorten and Khoshgoftaar, 2019). Figure 3.6 shows four examples of random erasing. Each image contains a black rectangle where pixels are removed from the original image. The size and aspect ratio can be adjusted. In Figure 3.6 each patch is 1/10 the size of the original image, and the shape of the rectangle is chosen randomly from a fixed range of possible aspect ratios.

Zheng et al. (2020) find that translation and horizontal flips are the most useful data augmentation techniques on CIFAR-10. On the other hand, rotation and noise disturbances did not significantly impact the classification accuracy. They explain that the CIFAR-10 dataset contains very small images (32×32 pixels), which are also quite blurred. Therefore, adding additional noise or changing the image significantly has a large impact on the image structure and does not help improve the network.

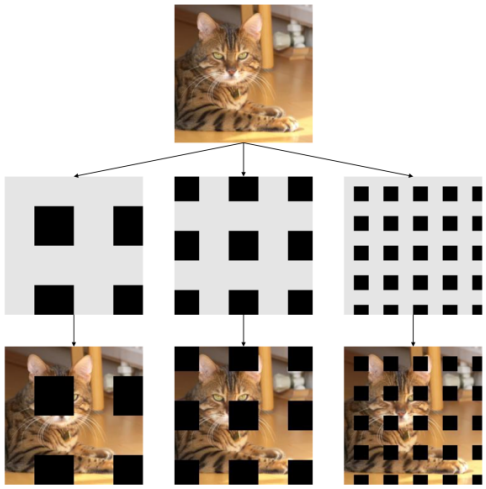


Figure 3.7: EXAMPLE OF GRIDMASK. First the grid is created based on a set of parameters that indicate the width, height, and space between the grids, and then the pixels in the original image are removed based on the created grid. *Source: (Chen et al., 2020)*

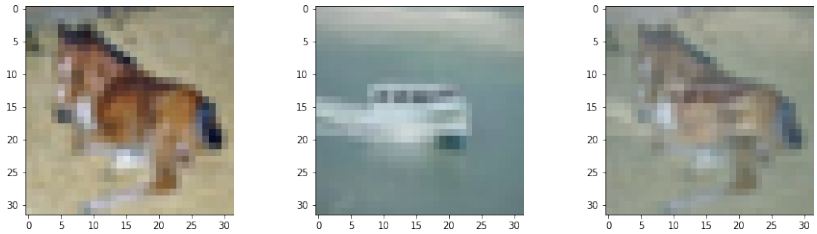
3.2 Advanced Data Augmentation Techniques

In addition to the standard data augmentation techniques, recent research has introduced several more advanced augmentation techniques, some of which are based on neural networks. I will give an overview of these techniques that have been tested and proven helpful in image classification tasks.

3.2.1 Information Dropping

Information dropping has proven to be highly effective because the model becomes robust against occlusion. The idea is to delete parts of the input data. This has proven to work best with small datasets (Chen et al., 2020). Random erasing, introduced in Section 3.1.4, belongs to the category of information dropping techniques. Because the location of the erased patches is chosen randomly, there always exists a risk of covering the main object. On the other hand, if a smaller patch size is chosen to reduce the risk of covering the object, the method might not be beneficial. Chen et al. (2020) propose a new method called GridMask, which uses structured dropping regions. Instead of randomly dropping an area within the image, GridMask lays a grid over the image and drops the corresponding areas accordingly. Figure 3.7 demonstrates this method. The first row shows the original image. The second row shows examples of different grids created based on a set of parameters that indicate the width, height, and space between the grids. The black areas indicate that those pixels are removed. The last row shows the transformed images based on the grid above.

Chen et al. (2020) demonstrate that GridMask improves the accuracy on ImageNet from 76.5% to 77.9%. Additionally, on CIFAR-10, they report an accuracy increase from 95.28% to 96.54% using GridMask.



(a) Original image (b) Randomly chosen image (c) Mixed image

Figure 3.8: EXAMPLE OF MIXING IMAGES. The original image in (a) has label *horse*. The randomly chosen image in (b) has label *boat*. The mixed image in (c) is created by simple pixel averaging and has label *horse*.

3.2.2 Mixing Images

Another interesting data augmentation technique, which has many variations, is mixing images. The simplest way to mix images is by taking two random images and producing a new image by averaging their pixel values (Shorten and Khoshgoftaar, 2019). Inoue (2018) introduces the technique *SamplePairing*. For an original image, *SamplePairing* randomly chooses another image and overlaps them to create a new augmented image. The overlapping takes place by averaging each pixel value from the original image and the randomly chosen image. The randomly chosen image does not necessarily have to be from the same class. The label of the transformed image is chosen to be the same as the original image label. For example, in Figure 3.8, the first image belongs to the category *horse*. Then another image is randomly chosen from the training set, which is from the category *boat*. Figure 3.8c shows the resulting mixed image from averaging the pixels in 3.8a and 3.8b. The label of the mixed image is the same as the label of the first image. Therefore the label is also *horse*. Inoue (2018) finds that choosing a random image from the entire training set gives better results than choosing an image from the same class or choosing an image from outside the training set. *SamplePairing* has been shown to improve the image classification accuracy on various datasets, including CIFAR-10.

Instead of taking the average of pixels in two images, Takahashi et al. (2020) propose a different method of mixing images. They propose a new data augmentation technique called RICAP (random image cropping and patching). Four images are mixed into one by taking a random patch from each image and concatenating them. Figure 3.9 illustrates this on an example. The red areas indicate the random crop that is taken from each of the four original training images. The center image shows the transformed image after patching the crops together. Furthermore, the crop sizes are selected so that the patched image remains the same size as the original images. Additionally, Takahashi et al. (2020) set the label of the mixed image to be probabilities proportional to the crop sizes. Hence, instead of using hard labels, where each image belongs to exactly one category with probability one, they used soft labels. Every image belongs to multiple categories, each with a probability. Takahashi et al. (2020) state that RICAP prevents CNNs from overfitting, which improves the accuracy in image classification tasks. Their results show an absolute improvement in the error rate of 1.04% on CIFAR-10 and an improvement of 1.63% on CIFAR-100.

Similar to RICAP, Summers and Dinneen (2019) propose other methods of mixing images. They examine many different methods, which consist of concatenating patches of both images and element-wise weighted averaging. In their experiments on CIFAR-10 and CIFAR-100, most methods improved the performance of the model. In particular, *VH-Mixup*, which takes a left top

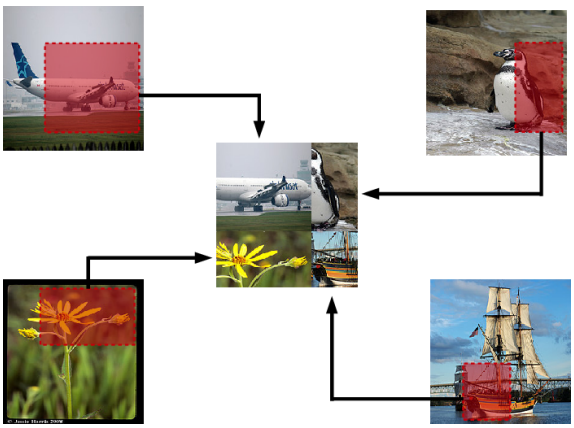


Figure 3.9: ILLUSTRATION OF RICAP. From each of the four images, a random crop is taken, indicated by the red area. The random crops are patched together to create a new transformed image. The label of the transformed image is a weighted mixture of the labels of the four images. *Source: (Takahashi et al., 2020)*

Figure 3.10: AUGMENTED IMAGE FROM A NEURAL NETWORK. This figure shows an example of an augmented image from a neural network that takes in two images of the same class. *Source: (Perez and Wang, 2017)*

rectangle from the first image, a right bottom rectangle from the second image, and the rest from element-wise weighted averaging, showed an absolute improvement of 1.6% on the error rate of CIFAR-10.

There exist more advanced methods for creating an augmented image from two original images. Perez and Wang (2017) came up with a method where one neural network is trained to create a transformed image from the input of two images from the same class, and a second neural network is trained to classify the images, where the input consists of the original images and the transformed images. Figure 3.10 shows an example of a transformed image that the first neural network produces from the input of two images. Although the experiments show that this method can work better than traditional data augmentations, the improvement is only minimal while the computation time is about three times longer (Perez and Wang, 2017).

3.2.3 Generative Adversarial Networks (GANs)

GANs have proven to be good at augmenting datasets (Perez and Wang, 2017). They are used to artificially create new samples similar to the original dataset (Shorten and Khoshgoftaar, 2019). Antoniou et al. (2018) have designed a Data Augmentation Generative Adversarial Network (DAGAN). The Generator takes an original image from class c, projects it to a lower-dimensional

space, and concatenates a random gaussian vector. Then the Decoder generates an augmented image from this vector. The Discriminator Network receives images from the same class c, representing the real distribution, and augmented images from the Generator, representing the fake distribution. It tries to identify whether the distribution is real or fake. Thus, the generated images must be similar to the original images from class c but different enough to be a different sample. In contrast to other data augmentation methods, DAGAN is a flexible model that automatically learns to augment data. An advantage of this is that it automatically discovers which transformations are valid for a given dataset. Some datasets, for example, MNIST, are more likely to be non-label preserving after a transformation. Horizontal flips are, for example, not a safe augmentation. Instead of manually deciding which transformations to include and which not to include, DAGAN does this automatically. Antoniou et al. (2018) have demonstrated on various datasets that DAGAN improves the image classification accuracy, even after standard data augmentations. In addition to DAGAN, there are many other variations of using GANs for creating additional augmented training images.

3.3 Data Augmentation during the Training Stage

All techniques mentioned in this chapter so far are mostly used during training. Recently, research has focused mainly on how to select the best data augmentation techniques automatically. Because this is an important development in data augmentation methods, I will introduce two techniques that try to tackle the automatic selection of transformations during training.

3.3.1 AutoAugment

Cubuk et al. (2019) introduce an algorithm that automatically searches for the best data augmentation policies such that training a neural network yields the best validation accuracy. The algorithm uses reinforcement learning. From a fixed set of policies that include specific image transformations and a probability that indicates how likely the transformation will be applied, the reinforcement learning algorithm finds the best policy. However, Cubuk et al. (2019) state that other search algorithms, such as a random walk, might generate better results.

3.3.2 Population Based Augmentation

In contrast to AutoAugment, Ho et al. (2019) introduce a new technique that is less computationally expensive. For image classification on CIFAR-10, AutoAugment requires 5'000 GPU hours, while Population Based Augmentation (PBA) requires only 5 GPU hours (Ho et al., 2019). The goal of PBA is to find the best schedule of data augmentation policies. Because PBA searches for a schedule instead of a fixed policy, the algorithm is more efficient. When searching for a fixed policy, the model must be trained until convergence for each possible policy, which is computationally expensive. However, when searching for a schedule, the model can reuse prior computations for schedules with the same prefixes.

The Population Based Algorithm was first introduced to tune the parameters and hyperparameters in a neural network such as the weights, the architecture, the loss function, and the optimization algorithm (Jaderberg et al., 2017). The algorithm starts by initializing a set of models with their weights and hyperparameters. The models are trained for a number of steps, and then the parameters of the worst-performing models are replaced by the parameters of the best-performing models.

3.4 Test-Time Augmentation

Test-time augmentation is a technique where multiple different data augmentations are performed on the test image, and each transformed image is propagated through the trained neural network. The final prediction is obtained through an aggregation function. The most common approaches are averaging and majority voting. In contrast to data augmentation on the training set, where advanced techniques are being used to find the best data augmentations, only simple techniques such as rotation and translation are commonly used for the test set (Molchanov et al., 2020).

Zheng et al. (2020) have tested various data augmentation techniques such as shifts, horizontal flips, and noise disturbance. In the testing phase, Zheng et al. (2020) used the same distribution of data augmentation techniques as in the training phase and calculated the final prediction using majority voting. Zheng et al. (2020) report that the average classification accuracy of CIFAR-10 has increased from 85.7% to 93.4% using data augmentation on both the training and the test images. However, data augmentation on the training images was more effective than on the test images.

Krizhevsky et al. (2012) used test-time augmentation by extracting five patches from the original test images. They extracted one patch from each corner of the image and the fifth patch from the center. Additionally, they flipped each patch horizontally, resulting in ten augmented images. Then they averaged the predictions made by the last layer of the network to obtain the final prediction. Hence, from the output probabilities for each category, they took the average over all the crops, resulting in a new vector of probabilities. Then the final prediction is the category with maximum probability. Howard (2013) out-performed this method by not just considering crops. Instead, he considered the combination of five translations, two flips, three scales, and three views, which in total resulted in 90 augmented images per test image. The views are large crops, in the top left corner, the center, and the bottom right corner. Due to running time concerns, he implemented a greedy algorithm that finds the ten best transformations instead of all 90. Similar to Krizhevsky et al. (2012), he averaged the network outputs for each augmented image to receive the final prediction. Howard (2013) obtained an improvement in the image classification error rate of 2% in ILSVRC2013 compared to Krizhevsky et al. (2012).

In ILSVRC2014, the winning team states that they have also used test-time augmentation. They fed the original test images and their horizontally flipped augmentations into the trained neural network. Then they averaged the outputs of both the original and the flipped test images (Simonyan and Zisserman, 2015).

Molchanov et al. (2020) have come up with a more advanced technique for test-time augmentation. They introduce a greedy policy search (GPS) that learns a test-time augmentation policy based on the predictive performance on the validation set. From a set of possible data augmentations (sub-policies), GPS iterates through all sub-policies and adds them to the current policy. If the validation performance increases, GPS keeps the sub-policy. Molchanov et al. (2020) state that their method improves the accuracy of image classification on various datasets, such as CIFAR-10 and ImageNet, compared to using the same data augmentation techniques as during the training stage and using the standard approach with crops and flips.

So far, these test-time augmentation approaches have focused mainly on the data augmentation techniques and not on the aggregation function. For aggregating the predictions of each transformed version of a test image, they often used simple averaging of the output probabilities from the softmax layer. However, Shanmugam et al. (2020) show in their experiments that simple averaging might not always be optimal. Instead, they used weighted averaging. They searched for the optimal weights by minimizing the cross-entropy loss between the true labels and the output of the aggregation function using gradient descent. They report that this method improves on simple averaging, greedy policy search, and no test-time augmentation.

Chapter 4

Data Augmentation during the Training Stage

The main goal of this thesis is to investigate data augmentation methods on the test set. However, I also use data augmentation during training because it improves the performance of the trained models. For each epoch during training, I transform each training image according to a set of transformations with given hyperparameters. The hyperparameters indicate, for example, the possible range of rotation angles to apply. Multiple transformations can be applied sequentially, and each transformation is given a probability that indicates the likelihood of that transformation to be applied. I choose the hyperparameters empirically by experimenting with different combinations and deciding based on the nature of the dataset. For example, with MNIST, I do not use any color transformations that change the hue or saturation because the images are grey-scaled. Additionally, I do not add horizontal flips for the MNIST dataset because it is not always label-preserving. Table 4.1 provides a full list of all transformations with the chosen hyperparameters that I use for training. The variable `p` in Table 4.1 indicates the probability of the corresponding transformation to be applied. Therefore, a specific image is randomly transformed in each epoch, creating a new version of the original image used for training and then discarded again. This has the effect that the network is introduced to a larger variation of the images without artificially enlarging the entire training set. Thus, as long as the image processing is fast enough, the training time of the neural network is similar to training the neural network on only the original training images.

4.1 Model Specifications

In this section, I will introduce the model specifications that I use for training. I use the same model architecture and loss function for all datasets. The hyperparameters, such as the learning rate, may differ slightly. I manually adjusted the learning rate if I found that the model was not learning well.

4.1.1 Model Architecture

I use the ResNet-18 architecture in my experiments, which is a deep residual neural network with 18 layers (He et al., 2016). Specifically, in this thesis I use Python as my main programming language, and use the implementation of ResNet-18 from the `torchvision.models`[1] package

[1] https://pytorch.org/docs/stable/torchvision/models.html

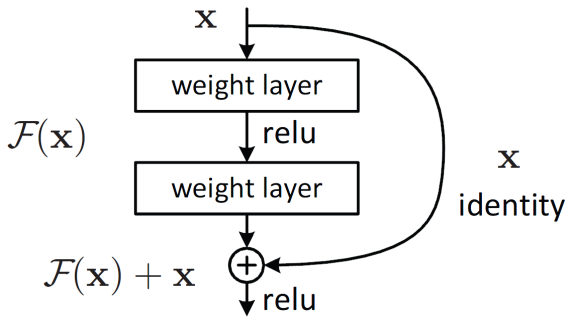


Figure 4.1: RESIDUAL UNIT. This figure shows an example of a residual unit. x is the input of the residual unit and $F(x) + x$ is the output. $F(x)$ consists of two layers. The skip connection is the identity function. *Source: (He et al., 2016)*

from PyTorch. ResNet-18 can be used both as a pretrained or a non-pretrained model. The pretrained model was trained on images from ImageNet. I tested both versions, and the pretrained model converged considerably faster and usually resulted in higher image classification accuracies. However, because it was pretrained on the training images of ImageNet, my validation results on ImageNet may not be as accurate as the validation results on the other datasets because the ImageNet training and validation sets overlap. However, this should not affect the test scores after test-time augmentation.

Residual neural networks solve the problem of vanishing gradients. During back-propagation, the gradient in the first layers is calculated using the chain rule, thus multiplying the derivatives of all the subsequent layers. Repeated multiplication of the derivatives eventually makes the gradient very small, which results in the vanishing gradient problem. Residual neural networks solve this problem by adding skip connections to the architecture. The architecture can be separated into multiple residual units. An example of a residual unit can be seen in Figure 4.1. The input x of the residual unit is propagated through several layers and additionally also through a skip connection, which in the figure is an identity mapping. Then, both the output of the layers and the output of the skip connection are summed up. An activation function, in this case, ReLU, takes the sum, and its output is fed to the next residual unit. In Figure 4.1, the skip connection is the identity function. However, suppose the dimensions of x are not the same as $\mathcal{F}(x)$. In that case, the skip connection includes a convolutional layer and batch normalization for downsampling or zero paddings for upsampling, such that the dimension matches $\mathcal{F}(x)$. The full architecture of ResNet-18 can be seen in Figure 4.2. The solid arcs are identity mappings; thus, the dimensionalities of $\mathcal{F}(x)$ and x are the same. The dotted arcs, which are annotated as skip connections, include an extra step to increase or decrease the dimensionality of x to match the dimensionality of $\mathcal{F}(x)$. In this example, the input images are of size 100×100 pixels. However, the pretrained network from PyTorch requires the input sizes to be at least 224×224. Therefore, after performing data augmentation, I scale each image to 224×224 before feeding it to the model.

The final layer of the ResNet-18 model outputs logits for each of the possible classes. Those logits can be transformed into probabilities when applying the softmax function. However, in the testing stage of my thesis, I use the original logits.

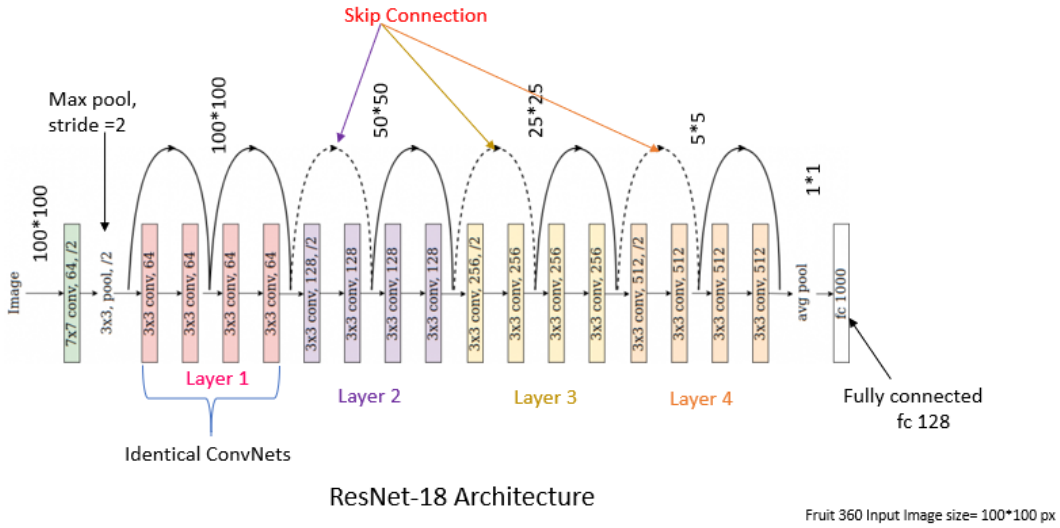


Figure 4.2: RESNET-18 ARCHITECTURE. This figure shows the architecture of ResNet-18. It describes each layer and where the skip connections are located. The solid arcs represent identity mappings, and the dotted arcs are skip connections with an extra step for adjusting the dimensionality. *Source: (Singhal, 2020)*

4.1.2 Loss Function

During training I use the categorical cross-entropy loss[2] provided by PyTorch. For Melanoma I use weighted cross-entropy because the labels are imbalanced with 584 malignant and 32'542 benign cases. To calculate the weights I use the same function as scikit-learn:[3]

$$w_c = \frac{N}{C * n_c}$$

where N is the total number of samples, C the total number of classes, and n_c the total number of samples in class c. Hence,

$$w_{\text{benign}} = \frac{33'126}{2 * 32'542} = 0.5090$$
$$w_{\text{malignant}} = \frac{33'126}{2 * 584} = 28.3613$$

4.1.3 Hyperparameters

I use the stochastic gradient descent optimizer from PyTorch with a momentum factor of 0.9. The learning rate is slightly different depending on the dataset because, in some cases, the model did not seem to learn well. Additionally, the learning rate decays by 0.1 at every seventh epoch. The batch sizes differ as well, depending on the dataset. The exact learning rates and batch sizes for each model can be found in Appendix A in Table A.1.

[2] https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_sample_weight.html

4.2 Data Augmentation Techniques

	CIFAR-10 & ImageNet	MNIST	Melanoma
Horizontal Flip	p = 0.5	p = 0	p = 0.5
Random Crop	proportion = 0.8 p = 0.5	proportion = 0.8 p = 0.5	proportion = 0.8 p = 0.5
Color Jitter	brightness = (0.3, 1.8) contrast = (0.3, 1.8) saturation = (0, 2) hue = (-0.1, 0.1) p = 0.2	brightness = (0.1, 2) contrast = (0.1, 2) p = 0.2	brightness = (0.6, 1.4) contrast = (0.6, 1.4) saturation = (0, 2) hue = (-0.1, 0.1) p = 0.5
Gaussian Noise	min. std = 0 max. std = 0.1 p = 0.2	min. std = 0 max. std = 0.1 p = 0.2	min. std = 0 max. std = 0.1 p = 0.2
Shift	min. width = 0 max. width = 0.2 min. height = 0 max. height = 0.2 p = 0.2	min. width = 0 max. width = 0.2 min. height = 0 max. height = 0.2 p = 0.2	min. width = 0 max. width = 0.1 min. height = 0 max. height = 0.1 p = 0.5
Rotation	min. angle = 0 max. angle = 40 p = 0.2	min. angle = 0 max. angle = 40 p = 0.2	min. angle = 0 max. angle = 180 p = 0.5
Random Erasing	scale = (0.1, 0.1) p = 0.2	scale = (0.1, 0.1) p = 0.2	scale = (0.1, 0.1) p = 0.2

Table 4.1: TRANSFORMATIONS FOR TRAINING. Chosen data augmentation techniques with given hyperparameters for training. `p` stands for the probability that the respective transformation is chosen.

Table 4.1 shows all the transformations with the corresponding hyperparameters that I use during training. I only consider the most common data augmentation techniques because they are easy to combine, and more advanced techniques go beyond the scope of this thesis.

Horizontal Flip I use the function `RandomHorizontalFlip` from torchvision[4] and adjust `p`, which indicates the probability of horizontal flipping being applied. Hence, for the experiments with the MNIST dataset I use `p=0`.

Random Crop I use the `RandomCrop` function from torchvision with a fixed crop proportion, with which I calculate the height and width of the cropped image. For each dataset, I use a crop proportion of 0.8 and a probability of 0.5 for applying the crop transformation. Specifically, if the original image is 230×400 pixels, and my crop proportion is 0.8, then the cropped image is 184×320. A crop proportion instead of a fixed crop size is necessary because, in ImageNet, the images have different sizes. Thus, it is not feasible to set a fixed crop size. Additionally, the crop placement is randomly chosen. A small value or `None` for `padding` in `RandomCrop` ensures that the crop placement does not overlap the borders by too much.

[4] https://pytorch.org/docs/stable/torchvision/transforms.html

Color Jitter I use the class `ColorJitter` from torchvision, where I set values for the hyperparameters `brightness`, `contrast`, `saturation`, and `hue`. For CIFAR-10, ImageNet, and Melanoma, I change the brightness, contrast, saturation, and hue. However, I choose a smaller range for the brightness and contrast for Melanoma because the skin lesions become difficult to detect with more extreme changes. For MNIST, I only change the brightness and contrast because the saturation and hue do not result in any changes for grey-scaled images. The ranges for brightness and contrast are larger for MNIST compared to the other datasets because the digits are still well visible after more extreme transformations. `p` indicates the probability of a color transformation to be applied.

Gaussian Noise I add Gaussian noise with a randomly chosen standard deviation (σ) from a fixed range. I choose the range such that σ would never be larger than 0.1 because with too much noise injected, it will be difficult to classify the image correctly. After choosing σ, I take a sample from a Gaussian Normal distribution with mean 0 and variance σ of the same size as the image. Then I add the sample to the pixel values of the image, which results in the augmented image with noise injection. Additionally, to ensure that the pixel values remain between 0 and 1, I replace values above 1 with one and values below 0 with zero.

Shift I use the function `shift`[5] from the bob.ip.base[6] package, where I set `offset` depending on the shift direction and length. Specifically, the direction of the shift, thus whether to shift horizontally or vertically, is always given. The length of the shift is chosen randomly from a given range.

Rotation I choose a minimum and maximum rotation angle that is applied with probability `p`. I use the `rotate`[7] function from the bob.ip.base package. `rotation_angle` indicates the rotation angle and can be either positive or negative. If it is positive, the image is rotated in a counterclockwise direction by the given degree. If it is negative, the image is rotated clockwise by the given degree. In my experiments, I randomly select an angle from a given range. With the given rotation angle, I randomly choose whether to rotate the image clockwise or counterclockwise, both with equal probability. For example, if my rotation angle is 20, and it randomly chooses to rotate in a clockwise direction, I set `rotation_angle` to -20.

For both shifting and rotation I use the function `extrapolate_mask()` from the bob.ip.base package which interpolates the missing pixels using information from the closest neighbors.

Random Erasing I use the function `RandomErasing` from torchvision. The hyperparameter `scale` gives the range of proportion of the erased area against the input image. It is assigned a tuple that indicates the minimum and maximum possible proportions of the patch. For all experiments, I choose `scale=(0.1,0.1)`. Thus, the erased area is exactly $1/10$ of the original image. This ensures that the erased area is not too large, which could cause difficulties in categorizing the image correctly. The position of the erased area is chosen randomly, and I use a constant fill method, such that all the patches are black.

[5] https://www.idiap.ch/software/bob/docs/bob/bob.ip.base/stable/py_api.html#bob.ip.base.shift

[6] https://www.idiap.ch/software/bob/docs/bob/bob.ip.base/stable/py_api.html

[7] https://www.idiap.ch/software/bob/docs/bob/bob.ip.base/stable/py_api.html#bob.ip.base.rotate

Dataset	Training Size	Validation Size	Test Size
CIFAR-10	40'000	10'000	10'000
MNIST	50'000	10'000	10'000
ImageNet	1'181'167	100'000	50'000
Melanoma	23'958	4'374	4'360

Table 4.2: DATASET SPLITS. Number of training, validation, and test images per dataset.

Dataset	DA	Training Score	Validation Score	Epochs	Training Instances
CIFAR-10	No	100%	95.25%	16	40'000
	Yes	93.28%	96.02%	34	1'360'000
MNIST	No	100%	99.58%	9	50'000
	Yes	99.33%	99.63%	10	500'000
ImageNet	No	93.63%	70.41%	8	1'181'167
	Yes	71.51%	73.20%	26	30'710'342
Melanoma	No	0.9744	0.8321	19	23'958
	Yes	0.9022	0.8689	102	2'443'716

Table 4.3: TRAINING SCORES. Training scores on CIFAR-10, MNIST, ImageNet, and Melanoma with and without data augmentation (DA). For CIFAR-10, MNIST, and ImageNet, the accuracies are given. For Melanoma, the AUC score is given.

4.3 Training Scores

I train a ResNet-18 model on all datasets mentioned in Chapter 2 using data augmentation. I also train each model without data augmentation for comparison. I split each dataset according to Table 4.2. CIFAR-10 and MNIST already have pre-defined train and test splits. I use the same split and additionally sample 10'000 images from the train set to use for validation. ImageNet has pre-defined training, validation, and test splits. However, because the labels for the test images are not publicly available, I use the 50'000 validation images as my test set and randomly split the training set into a training and a validation set. Melanoma also contains test images without publicly available labels. Therefore, I split the training set into a training, a validation, and a test set. In this case, a random split would not be ideal because the labels are very unbalanced. There exist multiple images from the same patients, and there are almost identical images of the same skin lesions. I use a split found by Chris Deotte, who competed in the corresponding Kaggle competition, and whose split has been a very valuable and widely used contribution.[8] Specifically, he assigned each image to one of 15 splits, such that all splits are equally distributed based on the labels and the proportion of the same images per patient. I use the first 11 splits for training, splits 12 and 13 for validation, and splits 14 and 15 for testing.

Table 4.3 shows the training and validation scores on the datasets with and without data augmentation. All models are evaluated on the original images of the validation set. All these results are with a pretrained ResNet-18 model with an early stopping threshold of 10 epochs. Hence, whenever the validation score does not improve for ten rounds, the training is terminated, and the parameters of the best scoring model are saved. The values under the column "Training Instances" indicate how many different images the model sees during training. For the models without data augmentation, this is precisely the size of the training set. For the models with data

[8] https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/164092

augmentation, it is the number of images in the training set multiplied by the number of epochs; because for each image and each epoch, a new transformation is applied given the sequence of transformations and probabilities from Table 4.1.

The models without data augmentation during training score worse on all datasets. Interestingly, the training score is always higher without data augmentation, but the validation score is smaller. This demonstrates that without data augmentation, the model is more likely to overfit to the training images. And hence, adding data augmentation during training can be beneficial in preventing overfitting.

Additional Comments on Melanoma In the corresponding Kaggle competition of Melanoma, participants had to submit probabilities that the test image is malignant melanoma. The submissions were then evaluated on the area under the ROC curve (AUC), which is a performance measurement independent of the class imbalance. It relies on the concept of some adjustable threshold. In particular, the AUC score receives logits or probabilities and returns a value that ranges from 0 to 1, with 0 being the worst score and 1 the best score that a model can achieve. I follow this approach and use the AUC score instead of the accuracy score for Melanoma. Therefore, I do not convert probabilities to labels but feed the probabilities directly to the score function. Since I implement all my methods using CIFAR-10, I have to adapt some methods for the Melanoma dataset. Therefore, this dataset serves as an interesting application and an experiment on whether the implemented methods can generalize well to different prediction problems.

A few data augmentation techniques have proven useful in the Kaggle competition, which I do not apply in this thesis because they are particular to the Melanoma dataset and beyond the scope of this thesis. Because some images include hair and some do not, it proved helpful to randomly add hair strands to the skin lesions during training.[9] This is an interesting case of data augmentations where the augmentation is very specific to the dataset and the classification problem since it would not make sense to add hair strands to the other datasets. Additionally, techniques of removing hair strands from the image have also proven helpful.[10]

Melanoma contains not only images of benign and malignant skin lesions but also information on the respective patients. Both the sex and the age of the patients are available. Additionally, information on the location of the skin lesion is also obtainable. Ideally, the predictions on the images and the given information on the patient and the location of the skin lesion should be aggregated to form a final prediction. However, for simplicity, I only consider the images. Therefore, my results, both during training and later during testing, are considerably lower compared to other results found on Kaggle or in papers mentioning the melanoma dataset.

The goal of the training stage is to train models on each of the four datasets, CIFAR-10, MNIST, ImageNet, and Melanoma, and to ensure that they perform relatively well such that the results in the next chapter are meaningful. The goal is not to train the best possible model. To do that, I would have to spend considerable time fine-tuning the hyperparameters of the model, using an automated approach to find the best set of data augmentation to use during training and experiment with different model architectures. I believe that although the validation scores given in Table 4.3 may not be the best possible scores, the scores are quite reasonable compared to what other papers have reported. Thus, in the following chapter, which is the main part of this thesis, I will extensively discuss how to use data augmentation techniques during the testing stage.

[9] https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/159176
[10] https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/165582

Chapter 5

Test-Time Augmentation

Test-time augmentation is the main focus of this thesis. As introduced in Chapter 3, test-time augmentation is a technique for improving the image classification accuracy during the testing stage. However, there is not a lot of research focusing on test-time augmentation techniques and methods. Figure 5.1 shows the workflow of the general procedure. The images in the test set are augmented, resulting in several modified images. Each image is fed to the network, which outputs logits or probabilities for each of the possible classes. Those logits are aggregated to form the final prediction. The most common approach is to augment each image in the test set using standard data augmentation techniques, such as horizontal flips, crops, and shifts. Then the resulting probabilities or logits are aggregated using simple averaging to form the final prediction. Shanmugam et al. (2020) discuss that this simple method might not always be optimal. In fact, in their experiments, they showed that using weighted averaging instead of simple averaging, with the weights corresponding to the importance of each transformation, results in better image classification accuracy. In this thesis, I build upon these insights and test various methods to aggregate the outputs of the augmented test images. In Section 5.1 I will describe all the methods that I implement and evaluate in my experiments. Some of these methods are very similar to already introduced methods in Chapter 3. However, they serve as a comparison to new approaches. Additionally, my set of transformations differ from the transformations in other papers. Most of the papers mentioned in Chapter 3 that use test-time augmentation used horizontal flips, crops, and shifts. I additionally use color transformations, random noise injection, rotations, and random erasing.

In contrast to data augmentation during the training stage, I use fixed hyperparameters for the transformations instead of ranges of possible values. In some cases, however, I allow some randomness. First, the crop is of fixed size, but the location of the crop is chosen randomly. Second, I add two shifts, a vertical and a horizontal shift. Whether the vertical shift is shifted up or down, and the horizontal shift is shifted left or right, is chosen at random. Similarly, the rotation angle is fixed, but the direction of the rotation, clockwise or counterclockwise, is chosen randomly. Lastly, the size of the random erasing patch is fixed, but the location is chosen randomly. I could have removed this randomness by adding additional transformations. For example, instead of randomly choosing the direction of the 10-degree angle, I could have just added both the 10-degree rotation clockwise and the 10-degree rotation counterclockwise. But because a smaller set of transformations simplifies the analysis and building of my methods, I decided to run most of my experiments without additional transformations. Listings A.1, A.2, and A.3 in Appendix A show all the transformations with the hyperparameters that I use for test-time augmentation on each dataset.

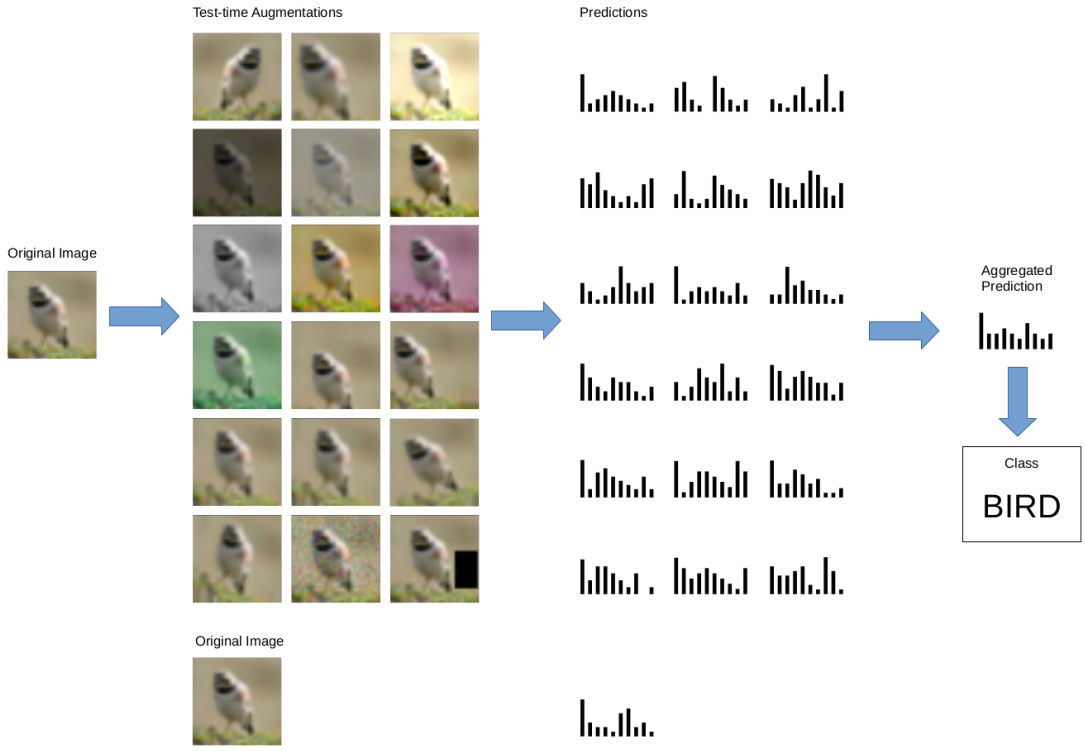


Figure 5.1: WORKFLOW OF TEST-TIME AUGMENTATION. The original image is augmented into several different modified versions. Each modified image is fed to the network, which produces an output of logits or probabilities for each category. Those outputs are aggregated to form the final prediction.

5.1 Methods

In this section, I will describe in detail the methods that I implement for test-time augmentation. All the methods that require fitting are fitted on the validation set after training. More specifically, I choose a fixed set of data transformations for test-time augmentation and augment each image in the validation set, respectively. After training, I save the raw predictions of the last layer of the network, the logits, of each transformed validation image. This results in a multi-dimensional array of logits, $V \in \mathbb{R}^{T \times N \times C}$, with T equal to the number of transformations, N equal to the number of validation images, and C equal to the number of classes in the dataset. With V, I fit various methods, which I then test on the test set. The advantage of this approach is that I do not have to re-train the model for each method that I test. I can use the same array of logits, V, to implement and experiment with different test-time augmentation methods.

First, I will provide a few definitions and notations that I will use throughout this chapter. For any matrix A, the function g returns a vector consisting of the indices per row with the largest

value:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1C} \\ a_{21} & a_{22} & \dots & a_{2C} \\ \vdots & & & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NC} \end{bmatrix} \tag{5.1}$$

$$g(A) := \begin{bmatrix} \arg\max_c a_{1c} \\ \arg\max_c a_{2c} \\ \vdots \\ \arg\max_c a_{Nc} \end{bmatrix} \tag{5.2}$$

The function g is used to calculate the predictions from a matrix of logits or probabilities. For each dataset, I index all the transformations used for test-time augmentation. For example, horizontal flipping may have index 1. Then, for any transformation $t \in \{1, ..., T\}$, V_t is a matrix of logits corresponding to transformation t:

$$V_t = \begin{bmatrix} l_{11}^{(t)} & l_{12}^{(t)} & \dots & l_{1C}^{(t)} \\ l_{21}^{(t)} & l_{22}^{(t)} & \dots & l_{2C}^{(t)} \\ \vdots & & \vdots & \\ l_{N1}^{(t)} & l_{N2}^{(t)} & \dots & l_{NC}^{(t)} \end{bmatrix} \in \mathbb{R}^{N \times C} \tag{5.3}$$

where $l_{nc}^{(t)}$ is the logit for the n-th validation image, class c, and transformation t. Additionally, the function δ is defined as:

$$\delta(\lambda, \theta) := \begin{cases} \theta & , \theta > \lambda \\ 0 & , \theta \leq \lambda \end{cases} \tag{5.4}$$

Finally, $\boldsymbol{y}$, which I will use later, indicates the vector of the true validation classes, and $\hat{\boldsymbol{y}}$ indicates the class predictions.

5.1.1 Greedy Approach

The greedy approach, which I will simply refer to as Greedy, tries to find the subset of data transformations from the given list of possible transformations that will result in the highest validation score. It is similar to the approach by Molchanov et al. (2020), except that my set of possible data augmentations differs. The procedure of this method is given as:

$$\hat{T}_1 := \{\}, \quad \hat{V}_1 := \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \in \mathbb{R}^{N \times C}$$

$$\hat{t}_s := \underset{t \in \{1,\dots,T\} \setminus \hat{T}_s}{\arg\max} \Big[\text{score}(\boldsymbol{y}, g(V_t + \hat{V}_s)) \Big]$$

$$\hat{T}_{s+1} = \hat{T}_s \cup \{\hat{t}_s\}, \quad \hat{V}_{s+1} = \hat{V}_s + V_{\hat{t}_s}, \quad \text{for } s \in \{1, ..., T\}$$

where score is either the accuracy score or the AUC score. The goal of Greedy is to find the optimal set of transformations, $\hat{T}_s$, such that the score of the sum of logits in that set is maximized.

$\hat{T}$ = [];
bestScore = 0;
count = 0;
while *count smaller than* `early_stopping` ***and*** $\hat{T}$ *does not contain all transformations* **do**
    runningScore = 0;
    for *every possible transformation* t *not in* $\hat{T}$ **do**
        take sum of logits of transformation t and all transformations in $\hat{T}$;
        calculate predictions;
        calculate score;
        if *score is greater than runningScore* **then**
            set runningScore to calculated score;
            set bestTransformation to t;
        end
    end
    if *runningScore is greater than bestScore* **then**
        set bestScore to runningScore;
        set count to 0;
    else
        increase count by 1;
    end
    add bestTransformation to $\hat{T}$;
end
remove last count entries of $\hat{T}$;
return $\hat{T}$;

Algorithm 1: ALGORITHM OF GREEDY. This algorithm explains the procedure of Greedy described in Section 5.1.1.

In my experiments, I implement an early stopping condition, where I terminate the algorithm as soon as the score does not improve for a given number of iterations. This number is given in `early_stopping`. In this case, I choose a small value for `early_stopping` because experiments show that once the algorithm deteriorates, it is unlikely that it improves again. Hence, I choose `early_stopping=3`. The final set of best data augmentations, which I will refer to as $\hat{T}$, is used on the test images for comparison with the other methods.

The full algorithm is given in Algorithm 1. The function receives the validation logits V and labels $\boldsymbol{y}$ and returns a set of transformations $\hat{T}$.

5.1.2 Weighted Approach

Instead of finding the best subset of transformations, the idea behind the weighted approach is to weigh each transformation based on its importance. This was also done by Shanmugam et al. (2020). They used gradient descent to optimize the weight vector. I use various common optimization algorithms. Some of these algorithms rely on different fixed hyperparameters. I fine-tune those hyperparameters on CIFAR-10, MNIST, and Melanoma by splitting the validation set randomly using 75% for fitting and 25% for evaluating the weights with different hyperparameters. Specifically, for each hyperparameter, I define a set of values to test. For each combination of hyperparameters, I run the algorithm on the larger subset of the validation images and calculate the final best weights. Then I evaluate the weights on the smaller subset of the validation

images and save the results. Finally, after testing all combinations, I receive the best scoring set of hyperparameters. Then I re-fit the weights using the complete validation set and test it on the test set. In this case, the holdout sample of the validation set is only used to ensure that hyperparameters are chosen that generalize well to an unseen set of images. However, all the images are used for re-fitting the weights. I do not run this grid search on ImageNet because of time concerns. Instead, I choose a combination of hyperparameters similar to the best combination found on CIFAR-10 and adjust them to save computation time. The exact hyperparameters that I test for each algorithm and the results of the grid search are given in Appendix B.

I test four different optimization algorithms to obtain the weights. They differ in the general procedure but also in the constraints of the weights. For some algorithms, the weights are restricted to be non-negative, whereas, in other algorithms, I allow negative weights. I introduce each algorithm in detail in the following sections. However, the idea of the weighted approach is always the same. I want to find the optimal weight vector $\tilde{\boldsymbol{\theta}}$ such that when taking the weighted sum of logits based on $\tilde{\boldsymbol{\theta}}$, the validation score is maximized. This can be written as:

$$\bar{V}(\boldsymbol{\theta}) := \sum_{t=1}^{T} \theta_t V_t \tag{5.5}$$

$$\tilde{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \left[\text{score}(\boldsymbol{y}, g(\bar{V}(\boldsymbol{\theta})))\right] \tag{5.6}$$

where T is the number of transformations, and V_t is the matrix of logits for transformation t given in Eq. (5.3).

Random Walk

In Random Walk, the weight vector is first initialized, and then in every iteration, some part of the vector changes. If the new vector improves the score, it is accepted; otherwise, it is rejected. There are many variants of Random Walk, and often adjustments must be made depending on the problem.

I adjust the hyperparameters for the initialization method, whether to normalize the weights or not and the number of early stopping rounds. I test two different initialization methods. The first initialization method is `equal`; thus, all weights are initialized to one. The second initialization method is `random`, where the initial weights are randomly sampled from a uniform distribution over $[0, 1)$. Additionally, the `normalization` hyperparameter can be set to `True` or `False`. If it is set to `True`, I normalize the initial weight vector. Because in every step of the algorithm, I always change two values of the vector; the first I increase by a randomly generated number, and the second I decrease by the same randomly generated number, the vector remains normalized throughout the process. If `normalization` is set to `False`, then the initial weight vector will not additionally be normalized at initialization. Normalization is not necessary; however, it allows me to infer the importance of each transformation based on the final weight. Additionally, the randomly generated number is chosen from a range such that the weight vector can never be negative. Again, this is not a necessity; however, since the weights represent the importance of each transformation, negative weights are difficult to interpret. Table 5.1 shows the initial weight vector, $\boldsymbol{\theta}$, given the different possibilities for the initialization method and normalization. T is the number of transformations, and u_t is a randomly generated number from a uniform distribution between 0 and 1 for each $t \in \{1, ..., T\}$. I run a grid search on CIFAR-10, MNIST, and Melanoma to identify the best combination of initialization, in addition to the best early stopping value. The results of the grid searches can be found in Appendix B in Tables B.1, B.2, and B.3. The final chosen hyperparameters for each dataset can be found in Table 5.2.

	Equal Initialization	Random Initialization
With Normalization	$\boldsymbol{\theta} = \begin{bmatrix} \frac{1}{T} & \dots & \frac{1}{T} \end{bmatrix}'$	$\boldsymbol{\theta} = \begin{bmatrix} \frac{u_1}{\sum_{t=1}^{T} u_t} & \cdots & \frac{u_T}{\sum_{t=1}^{T} u_t} \end{bmatrix}'$
Without Normalization	$\boldsymbol{\theta} = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}'$	$\boldsymbol{\theta} = \begin{bmatrix} u_1 & u_2 & \dots & u_T \end{bmatrix}'$

Table 5.1: WEIGHT INITIALIZATION. Weight initialization possibilities depending on the hyperparameters for initialization and normalization.

Dataset	early_stopping	normalization	initialization
CIFAR-10	500	True	Equal
MNIST	10	True	Random
ImageNet	10	True	Equal
Melanoma	10	False	Equal

Table 5.2: RANDOM WALK HYPERPARAMETERS. Chosen Random Walk hyperparameters for each dataset based on the best grid search results. For ImageNet, I choose similar hyperparameters as CIFAR-10 but reduce the number of early stopping iterations to shorten the computation time.

The score of the new weight vector is calculated by summing up the logits with respect to the weights, given in Eq. (5.5), and making predictions using the weighted sum of logits. If the score improves, the new weight vector is kept, and the algorithm continues changing values on the new weight vector. If the score does not improve, the new weight vector is discarded, and the algorithm continues with the previous weight vector. This process continues until the early stopping condition is met. Hence, as soon as the algorithm does not improve for a certain number of rounds based on the value in `early_stopping`, the learning process is terminated, and the weight vector with the best score is returned and saved for testing.

Algorithm 2 shows the procedure in pseudo-code. Because the resulting weight vector differs for different random seeds, I run this procedure 20 times and return the average validation and test scores along with the standard deviation over the 20 rounds.

Evolutionary Algorithms

Evolutionary algorithms follow the idea of survival of the fittest, which in turn causes a rise in the fitness of the population (Eiben and Smith, 2015). This idea can be used to optimize a vector. There are many variants of this algorithm, but the general scheme is given in Algorithm 3.

I implement a very simple variant of evolutionary algorithms, EA Own, which only uses mutation and selection. In summary, given a weight vector $\boldsymbol{\theta}$, it creates n new weight vectors $\{\hat{\boldsymbol{\theta}}^{(i)}\}_{i \in \{1,\dots,n\}}$ by adding random Gaussian noise to $\boldsymbol{\theta}$. Additionally, weights smaller than one are set to zero to prevent overfitting. A new weight vector $\hat{\boldsymbol{\theta}}^{(i)}$ is, therefore, defined as:

$$\hat{\boldsymbol{\theta}}^{(i)} := \begin{bmatrix} \delta(1, \theta_1 + \epsilon_1^{(i)}) \\ \vdots \\ \delta(1, \theta_T + \epsilon_T^{(i)}) \end{bmatrix}$$

were δ is defined in Eq. (5.4) and $\epsilon_t^{(i)} \sim \mathcal{N}(0, 1)$.

Then, for each new weight vector $\hat{\boldsymbol{\theta}}^{(i)}$, for $i \in \{1, ..., n\}$, the validation score is calculated and a fixed number of best weights are kept. That number is given in `n_elites`. The average of

initialize $\tilde{\theta}$;
count = 0;
while *count smaller than* `early_stopping` **do**
  choose two random indices to change;
  generate a random value;
  add the random value to the weight in the first chosen index;
  subtract the random value from the weight in the second chosen index;
  calculate the weighted sum of logits;
  calculate predictions;
  calculate score;
  if *score improved* **then**
    keep $\tilde{\theta}$ and set count to 0;
  else
    revert back to previous $\tilde{\theta}$;
    increase count by 1;
  end
end
return $\tilde{\theta}$;

Algorithm 2: ALGORITHM OF RANDOM WALK. This algorithm explains the procedure of Random Walk described in Section 5.1.2.

those `n_elites` best weights is calculated, which results in a new weight vector, $\tilde{\theta}$, with which the process is repeated. Thus, instead of recombining pairs of parents to create offsprings, the offsprings are created by taking the mean of all parents, which I then evaluate. This process is repeated multiple times until it reaches the early stopping threshold. Algorithm 4 gives the algorithm in pseudo-code.

Similar to Random Walk, I test both an equal and a random initialization. In contrast to Random Walk, however, I do not perform normalization. I run a grid search for each dataset, except ImageNet, with different values for `n`, `n_elites`, and `early_stopping`. The results of the grid search on CIFAR-10, MNIST, and Melanoma can be found in Appendix B in Tables B.4, B.5, and B.6. The chosen hyperparameters based on the grid search results can be found in Table 5.3. Additionally, because this algorithm also returns different weights based on different random seeds,

initialize population with random candidate solutions;
evaluate each candidate;
while *termination condition not yet satisfied* **do**
  select parents;
  recombine pairs of parents;
  mutate the resulting offspring;
  evaluate new candidates;
  select individuals for the next generation;
end
return best candidate;

Algorithm 3: EVOLUTIONARY ALGORITHM. This algorithm explains the general scheme of an evolutionary algorithm.

initialize $\tilde{\theta}$;
count = 0;
while *count smaller than* `early_stopping` **do**
  create `n` new vectors by adding Gaussian noise;
  set negative and small weights to zero;
  calculate weighted sum of logits for each new vector;
  calculate predictions;
  calculate scores;
  choose best `n_elites` weight vectors;
  $\tilde{\theta}$ = average of best `n_elites` weight vectors;
  if *score with* $\tilde{\theta}$ *improved* **then**
    set count to 0;
  else
    increase count by 1;
  end
end
return $\tilde{\theta}$;

Algorithm 4: ALGORITHM OF EA OWN. This algorithm explains the procedure of EA Own described in Section 5.1.2.

Dataset	n	n_elites	initialization	early_stopping
CIFAR-10	1000	10	Random	10
MNIST	100	20	Random	50
ImageNet	100	20	Random	10
Melanoma	100	50	Ones	10

Table 5.3: EA OWN HYPERPARAMETERS. Chosen hyperparameters for EA Own for each dataset based on the best grid search results. For ImageNet I choose similar hyperparameters to CIFAR-10 but adjust `n` and `n_elites` to reduce the computation time.

I run the algorithm 20 times and save the average validation and test scores and the standard deviation.

Since this algorithm is a very simplified version of evolutionary algorithms, I also experiment with another variant. I experiment with ESTool[1], which contains implementations of various evolutionary algorithms including SimpleGA (Ha, 2017).

SimpleGA follows the general scheme given in Algorithm 3, where the offsprings are created by crossover, thus randomly selecting the values of either parent 1 or parent 2 both with 50% probability. SimpleGA can be adjusted with different hyperparameters. From the current population, the best samples, called the elites, are chosen. The proportion of elites must be given. From the elite sample, two random parents are chosen, and a child is created by crossover. This is done as many times as the given population size, hence creating the next generation. After this recombination process, Gaussian noise is added to each sample of the new generation. This is the mutation process. The standard deviation of the Gaussian noise decreases over time, but the initial standard deviation must be given. This procedure is repeated multiple times until a given stopping criteria is reached. I adjust the hyperparameters for `population_size`, `standard_deviation`, and `elite_proportion` in my experiments. I run grid search on CIFAR-10, MNIST, and Melanoma

[1] https://github.com/hardmaru/estool

Dataset	population_size	standard_deviation	elite_proportion
CIFAR-10	250	0.7	0.2
MNIST	250	0.5	0.1
ImageNet	20	0.7	0.2
Melanoma	20	0.7	0.1

Table 5.4: SIMPLEGA HYPERPARAMETERS. Chosen hyperparameters for SimpleGA for each dataset based on the best grid search results. For ImageNet, I choose similar hyperparameters to CIFAR-10 but adjust `population_size` to reduce the computation time.

to identify the best combination of hyperparameters such that the score on the validation images is maximized. The grid search results with the tested hyperparameters can be found in Appendix B in Tables B.7, B.8, and B.9. The chosen hyperparameters can be found in Table 5.4. Additionally, I use `early_stopping=10` for all experiments with SimpleGA.

Similar to Random Walk and EA Own, I also repeat the algorithm 20 times and save the average validation scores, test scores, and standard deviation.

Score Based

The score-based approach, which I will refer to as Score Based, is a straightforward approach to find the best weights without using an optimization algorithm. The weight for each transformation is equal to the validation score of using only that transformation. For example, the weight for the transformation of random erasing is equal to the score I receive on the validation images after doing random erasing on each image. Additionally, I choose a threshold for the score for which the weight becomes zero. Hence, if the score for a specific transformation is below the given threshold, the weight for that transformation is set to zero. This can be seen as a regularization technique because I am not considering transformations that are not helpful. I tune the threshold on each dataset individually and choose the threshold that maximizes the validation score using the resulting weights. The following equation shows this method in mathematical terms:

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_T \end{bmatrix} := \begin{bmatrix} \text{score}(\boldsymbol{y}, g(V_1)) \\ \text{score}(\boldsymbol{y}, g(V_2)) \\ \vdots \\ \text{score}(\boldsymbol{y}, g(V_T)) \end{bmatrix}$$

$$\tilde{\boldsymbol{\theta}} = \begin{bmatrix} \delta(\lambda, \theta_1) \\ \delta(\lambda, \theta_2) \\ \vdots \\ \delta(\lambda, \theta_T) \end{bmatrix}, \quad \lambda \in \mathbb{R}^+$$

where the function δ is given in Eq. (5.4), and λ is the threshold. Given the weight vector $\tilde{\boldsymbol{\theta}}$, the weighted sum of logits is calculated as in Eq. (5.5).

5.1.3 Majority Voting

Majority voting is probably the most straightforward technique for test-time augmentation. Zheng et al. (2020) used majority voting in their experiments. I implement majority voting mainly to compare its results with other methods mentioned in this chapter. The general idea of majority

voting for test-time augmentation is to make separate predictions for each transformation. Instead of aggregating each transformation by summing up the logits, I aggregate the final class predictions of each transformation. For example, with 20 different transformations, I get a vector of 20 class predictions for an image. Then I choose the most frequent prediction for the final prediction. This is given as:

$$\hat{\boldsymbol{y}} = \begin{bmatrix} \text{mode}(\arg\max_c l_{1c}^{(1)}, \arg\max_c l_{1c}^{(2)}, ..., \arg\max_c l_{1c}^{(T)}) \\ \text{mode}(\arg\max_c l_{2c}^{(1)}, \arg\max_c l_{2c}^{(2)}, ..., \arg\max_c l_{2c}^{(T)}) \\ \vdots \\ \text{mode}(\arg\max_c l_{Nc}^{(1)}, \arg\max_c l_{Nc}^{(2)}, ..., \arg\max_c l_{Nc}^{(T)}) \end{bmatrix}$$

where mode returns the most frequent integer and $l_{nc}^{(t)}$ are the logits in V_t defined in Eq. (5.3).

5.1.4 Averaging

Averaging is the method that most research papers mention when doing test-time augmentation. The last layer of the model is often a softmax layer that converts the logits to probabilities. Those probabilities are averaged over all transformations. Then, the label with maximum probability is chosen. I test this method both with the logits and the probabilities after applying the softmax function to the logits. The following formulas explain averaging on the original logits V. The procedure using probabilities instead of logits is analogous.

$$\text{avg}(V) = \frac{\sum_{t=1}^{T} V_t}{T} = \begin{bmatrix} \frac{l_{11}^{(1)} + l_{11}^{(2)} + ... + l_{11}^{(T)}}{T} & \cdots & \frac{l_{1C}^{(1)} + l_{1C}^{(2)} + ... + l_{1C}^{(T)}}{T} \\ \vdots & & \vdots \\ \frac{l_{N1}^{(1)} + l_{N1}^{(2)} + ... + l_{N1}^{(T)}}{T} & \cdots & \frac{l_{NC}^{(1)} + l_{NC}^{(2)} + ... + l_{NC}^{(T)}}{T} \end{bmatrix} \in \mathbb{R}^{N \times C}$$

Then, the prediction $\hat{\boldsymbol{y}}$ is calculated using the function g defined in Eq. (5.2):

$$\hat{\boldsymbol{y}} = g\big(\text{avg}(V)\big)$$

I do not have to fit Majority Voting and Averaging on the validation set because these methods do not require optimizing any parameters. However, for Greedy and the four methods belonging to the weighted approach, I have to add an additional step to fit the subset and weights to the validation set. In the following chapter, I will provide all my results using the different test-time augmentation methods introduced in this chapter.

Chapter 6

Results

In this chapter, I will provide my results from the experiments using different test-time augmentation methods mentioned in Chapter 5. I adapt and tune all the methods using only the validation images. I then compare them with each other using the results on the test images. Table 6.1 gives an overview of all the approaches and the resulting validation and test scores on the different datasets. The scores on CIFAR-10, MNIST, and ImageNet are accuracy scores, and the scores on Melanoma are AUC scores. I repeat the experiments for Random Walk, EA Own, and SimpleGA 20 times per dataset with different randomly generated seeds. The noted scores in Table 6.1 correspond to the average scores over the 20 repetitions. The last column contains the standard deviation over the 20 seeds on the test set. Additionally, all the experiments for test-time augmentation are done after training. During the training stage, I use a pre-trained ResNet-18 architecture and include data augmentation.

The red boxes in Table 6.1 indicate the best methods on the test set for each dataset. Before comparing the performances of the methods, it is important to note that Majority Voting and both Averaging methods do not require an additional fitting on the validation set and are therefore simpler and less computationally expensive than Greedy and the weighted approaches. Additionally, those are also the methods that are most often used for test-time augmentation. I will refer to Majority Voting, Averaging (probabilities), and Averaging (logits) as the simple methods.

For CIFAR-10, the best method is my variant of the evolutionary algorithm, EA Own. It scores on average 0.42% higher than without using any test-time augmentation. In fact, all methods score better than without test-time augmentation. Additionally, Score Based and SimpleGA perform quite well on CIFAR-10. Although EA Own performs better on average, the standard deviation of SimpleGA is much smaller and performs only slightly worse than EA Own. Out of the simple methods, Averaging (probabilities) scores best with only a 0.09% lower accuracy than EA Own.

On MNIST, all methods perform better than no test-time augmentation as well. The best-scoring methods on MNIST are SimpleGA and EA Own, with a 0.14% and 0.13% improvement compared to not using test-time augmentation. Similar to CIFAR-10, the standard deviation of SimpleGA is the smallest. Out of the simple methods, Majority Voting scores best with a 0.06% lower accuracy than SimpleGA.

The results on ImageNet are similar to CIFAR-10 and MNIST. SimpleGA scores best with an absolute increase in accuracy of 1.66% compared to no data augmentation during the testing stage. EA Own scores only 0.01% worse than SimpleGA. Out of the simple methods, Averaging (probabilities) scores best with a 0.62% lower accuracy than SimpleGA.

On Melanoma, EA Own scores best, with an increase of 0.0258 in the AUC score compared to not using test-time augmentation. Score Based and Greedy score relatively high as well. SimpleGA scores best on the validation set but quite low on the test set. Hence, there seems to be an overfitting issue regarding SimpleGA. Out of the simple methods, Averaging (probabilities)

Dataset	Method	Validation Score	Test Score	Standard Deviation
CIFAR-10	No Data Augmentation	96.02%	95.58%	-
	Greedy	96.52%	95.87%	-
	Random Walk	96.46%	95.87%	0.0335
	EA Own	96.66%	96.00%	0.0287
	SimpleGA	96.69%	95.95%	0.0005
	Score Based	96.32%	95.96%	-
	Majority Voting	96.28%	95.89%	-
	Averaging (probabilities)	96.24%	95.91%	-
	Averaging (logits)	96.28%	95.83%	-
MNIST	No Data Augmentation	99.63%	99.56%	-
	Greedy	99.71%	99.64%	-
	Random Walk	99.69%	99.64%	0.0174
	EA Own	99.75%	99.69%	0.0173
	SimpleGA	99.76%	99.70%	0.0002
	Score Based	99.69%	99.63%	-
	Majority Voting	99.68%	99.64%	-
	Averaging (probabilities)	99.68%	99.62%	-
	Averaging (logits)	99.68%	99.62%	-
ImageNet	No Data Augmentation	73.20%	68.36%	-
	Greedy	74.57%	69.68%	-
	Random Walk	74.29%	69.59%	0.2182
	EA Own	74.74%	70.01%	0.0131
	SimpleGA	74.80%	70.02%	0.0329
	Score Based	74.27%	69.52%	-
	Majority Voting	73.87%	69.11%	-
	Averaging (probabilities)	74.10%	69.40%	-
	Averaging (logits)	73.87%	69.15%	-
Melanoma	No Data Augmentation	0.8689	0.8627	-
	Greedy	0.8897	0.8863	-
	Random Walk	0.8784	0.8809	0.0000
	EA Own	0.8899	0.8885	0.0005
	SimpleGA	0.9010	0.8837	0.0020
	Score Based	0.8886	0.8883	-
	Majority Voting	-	-	-
	Averaging (probabilities)	0.8784	0.8820	-
	Averaging (logits)	0.8784	0.8809	-

Table 6.1: VALIDATION AND TEST SCORES. Validation and test scores of all the experiments for test-time augmentation. The scores for CIFAR-10, MNIST, and ImageNet are given as accuracies. The scoring metric for Melanoma is AUC. The experiments for Random Walk, EA Own, and SimpleGA were repeated 20 times, and the average validation and test scores are given, along with the standard deviation. The red boxes indicate the best scoring methods for each dataset.

scores best with a 0.0065 lower AUC score than EA Own. In contrast to CIFAR-10 and MNIST, SimpleGA has the highest standard deviation over the 20 seeds. The results of Majority Voting are missing because it is not easily applicable for Melanoma. I evaluate Melanoma using the AUC score, which relies on the concept of some adjustable threshold and requires real values. Majority Voting, on the other hand, returns binary class predictions.

Because the scores of Random Walk, EA Own, and SimpleGA differ depending on the random seed, I run a t-test to compare their results with each other and with the methods that I only run once. To do this, I use Welch's t-test. Welch's t-test is similar to the Student's t-test, except that it does not assume equal variance. Thus, both groups do not have to have an equal sample size and variance. For Random Walk, EA Own, and SimpleGA, I have 20 different samples because the final test score varies depending on the randomly generated seed. However, for all the other methods included in my experiments, the test scores remain the same for different seeds because the methods are non-random. Thus, if I compare methods across both groups, for example, Random Walk with Greedy, then the Student's t-test may be unreliable because the assumption of equal variance is not given. Therefore, I opt for the Welch's t-test, and use the function `ttest_ind`[1] from SciPy.

Given two methods, M_1 and M_2, with M_1 being a method that depends on a random seed, and M_2 being another method, the Null and alternative hypothesis are given as:

- H_0: M_2 belongs to the distribution of M_1.
- H_1: M_2 is significantly different from M_1.

If M_2 belongs to the distribution of M_1, then there is no significant difference between the methods M_1 and M_2. This is important because Random Walk, EA Own, and SimpleGA require a longer computation time to optimize the weights. Thus, if their test scores are not significantly better than faster methods, there is no benefit from using weight-optimizing algorithms. I use a significance level of 0.01; thus, given the p-value calculated by Welch's t-test, I reject the Null hypothesis for $p \leq 0.01$. The results of Welch's t-test are given in Table 6.2. In most cases, the methods are significantly different from each other. I am mostly interested in EA Own and SimpleGA, because those seem to be the best-performing algorithms. On MNIST and ImageNet, there is no significant difference between EA Own and SimpleGA. However, they are both significantly better than all other methods. On Melanoma, EA Own scores best, followed by Score Based. According to Welch's t-test, there is no significant difference between the two. Because Score Based only requires optimizing a threshold, it is computationally less expensive than EA Own. On CIFAR-10, Score Based is also the next best scoring model after EA Own. However, in this case, EA Own seems to be significantly better than Score Based.

In the following chapter, I will continue the discussion of my results, introduce further attempts to improve on the test scores, and highlight the limitations of my approach.

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html

Dataset	Method	Random Walk	EA Own	SimpleGA
CIFAR-10	No Data Augmentation	**2.48e-19**	**1.25e-23**	**5.51e-56**
	Greedy	1.00e+00	**4.02e-14**	**2.39e-43**
	Random Walk	-	**3.12e-15**	**2.74e-09**
	EA Own	**3.12e-15**	-	**3.60e-07**
	SimpleGA	**2.74e-09**	**3.60e-07**	-
	Score Based	**3.91e-10**	**7.66e-06**	**3.37e-26**
	Majority Voting	1.75e-02	**8.15e-13**	**5.65e-41**
	Averaging (probabilities)	**5.04e-05**	**2.79e-11**	**1.25e-37**
	Averaging (logits)	**5.04e-05**	**2.94e-16**	**1.08e-46**
MNIST	No Data Augmentation	**3.07e-14**	**3.51e-18**	**1.58e-55**
	Greedy	1.00e+00	**1.13e-10**	**1.55e-48**
	Random Walk	-	**8.26e-11**	**5.29e-12**
	EA Own	**8.26e-11**	-	2.09e-02
	SimpleGA	**5.29e-12**	2.09e-02	-
	Score Based	2.15e-02	**4.80e-12**	**8.31e-50**
	Majority Voting	1.00e+00	**1.13e-10**	**1.55e-48**
	Averaging (probabilities)	**7.77e-05**	**3.09e-13**	**6.57e-51**
	Averaging (logits)	1.00e+00	**1.13e-10**	**1.55e-48**
ImageNet	No Data Augmentation	**7.33e-16**	**2.25e-41**	**7.94e-34**
	Greedy	8.81e-02	**4.24e-28**	**8.89e-21**
	Random Walk	-	**7.98e-08**	**4.81e-08**
	EA Own	**7.98e-08**	-	2.30e-01
	SimpleGA	**4.81e-08**	2.30e-01	-
	Score Based	1.78e-01	**2.34e-31**	**6.11e-24**
	Majority Voting	**1.03e-08**	**2.26e-36**	**7.19e-29**
	Averaging (probabilities)	**1.22e-03**	**3.65e-33**	**1.04e-25**
	Averaging (logits)	**4.03e-08**	**5.35e-36**	**1.69e-28**
Melanoma	No Data Augmentation	**0.00e+00**	**5.19e-34**	**6.59e-21**
	Greedy	**0.00e+00**	**6.81e-14**	**1.83e-05**
	Random Walk	-	**6.09e-24**	**7.23e-06**
	EA Own	**6.09e-24**	-	**1.25e-09**
	SimpleGA	**7.23e-06**	**1.25e-09**	-
	Score Based	**0.00e+00**	9.74e-02	**5.05e-09**
	Majority Voting	-	-	-
	Averaging (probabilities)	**0.00e+00**	**1.17e-22**	**1.50e-03**
	Averaging (logits)	1.00e+00	**6.09e-24**	**7.23e-06**

Table 6.2: WELCH'S T-TEST. This table provides the p-values of Welch's t-test. The bold values represent p-values below the significance level of 0.01. In those cases, I reject the Null hypothesis that both methods belong to the same distribution.

Chapter 7

Discussion

In this chapter, I will discuss the results and usability of the various test-time augmentation methods introduced in Chapter 5. I will compare the different methods for improving test-time augmentation and discuss my attempts to improve the test scores. Lastly, I will discuss and examine limitations in my approach.

The results in Chapter 6 show that the weighted approach, using a weight optimizing algorithm, scores best on all datasets. In particular, on each dataset, EA Own or SimpleGA score best. Both algorithms are variants of evolutionary algorithms. In EA Own, the weights are non-negative. All negative weights are set to zero. I reason that negative weights are difficult to interpret since I assume that the weights indicate their importance towards the classification of the images. For SimpleGA, I do not enforce that restriction. However, the negative weights in SimpleGA are a concern because I suspect that they encourage overfitting on the validation set. The validation score of SimpleGA is highest across all datasets. The test score, on the other hand, is not always the highest. For Melanoma, in particular, the test score is comparatively low. The optimal weights found by each algorithm can be found in Appendix C. For Random Walk, which does not score as well as the other two optimization algorithms, the weights are non-negative, and for all datasets, except Melanoma, they are normalized. For Melanoma, the best combination found by grid search using Random Walk is equal initialization without normalization. Hence, the weights are initialized to ones. Table C.4 shows that the average optimal weights remain all ones for Random Walk. In fact, for none of the 20 rounds with different seeds, the optimal weight differs from all ones. Although the grid search finds that this initialization results in the best score, it is unclear whether equal initialization without normalization is a good technique for Random Walk.

Regarding computational efficiency, EA Own and SimpleGA are, in most experiments, the most expensive algorithms. The running time for both algorithms depend on the size of the dataset, the number of transformations, and the chosen hyperparameters. The hyperparameters, such as `early_stopping`, can be adjusted to speed up computation time. However, often there is a trade-off between computation time and performance. Additionally, even with hyperparameters that allow a faster computation, it will still require more time than other methods, such as Majority Voting and Averaging. On the other hand, the test scores are significantly higher using EA Own or SimpleGA compared to faster methods. EA Own and SimpleGA can be used in many different applications and in combination with other related work. First, other datasets can be used besides CIFAR-10, MNIST, ImageNet, and Melanoma. Second, other model architectures can be used besides a ResNet-18 model. In particular, all methods introduced in Chapter 5 are applied after training the neural network. Therefore, all methods can be applied during testing, no matter what model architecture or hyperparameters are chosen during training, as long as the output of the network contains logits or probabilities for each possible class. In particular, more advanced data augmentation techniques mentioned in Chapter 3, such as AutoAugment,

can be combined with the test-time augmentation methods introduced in this thesis. Important to note, however, is that depending on the method for training, for example, the chosen data augmentation technique, the test-time augmentation methods may not have the same effect. I will come back to this point in Section 7.2. Additionally, other transformations can be used during testing, besides the transformations that I consider. However, not all techniques mentioned in Chapter 3 can be well combined with all my methods for test-time augmentation. For example, mixing images may not work well with Greedy and the weighted approaches because the exact or very similar transformations have to be applied to both the validation and the test set, such that the optimal subset or weight can be successfully inferred to the test set. On the other hand, Majority Voting and Averaging can be easily applied to other transformations, such as mixing images. They only rely on a set of transformations for each test image. It is not required that each transformation must be similar among the images.

In my experiments, I use very similar transformations both during training and testing. However, it is possible that if specific transformations are used for test-time augmentation, for example, random erasing, which was not used during training, that this may have a negative effect on the test score. I will further elaborate on this point in Section 8.2. In contrast to the other methods, Majority Voting cannot be easily applied in some cases, as seen with Melanoma, because it returns class predictions instead of logits or probabilities.

In the following section, I will introduce further attempts to improve the test score.

7.1 Improvements

Although I try to cover various datasets and methods with my experiments, many improvements can be made in several different areas. I will discuss potential improvements for future work in Chapter 8. However, I experiment with two additional ideas to improve my test-time augmentation approaches. To further improve data augmentation during the testing phase, I experiment with a larger set of transformations and image clusters. During the testing, described in the previous chapters, I focus on a small fixed set of transformations given in Appendix A in Listings A.1, A.2, and A.3. Depending on the dataset, this includes 13 to 19 single transformations. These transformations include flips, crops, color transformations, noise injection, shifts, rotations, and random erasing. However, I do not combine these transformations with each other. For example, a transformed test image is not both horizontally flipped and injected with Gaussian noise. Because the combination of transformations, and in general a larger set of transformations, may improve the results of the methods introduced in Section 5.1, I add an experiment on CIFAR-10. The details of this experiment is given in Section 7.1.1. My second attempt for improving test-time augmentation involves clustering of the images. Instead of fitting the methods mentioned in Section 5.1 on all validation images and inferring the results to all test images, I try to find suitable clusters among the validation and test images, such that results during validation are only used on test images within the same cluster. Details of this attempt can be found in Section 7.1.2.

7.1.1 Combination of Transformations

I run all the methods described in Section 5.1 using an enlarged set of transformations. I include the single transformations given in Appendix A Listing A.1, and all combinations of combining two transformations. However, I do not combine similar transformations. Specifically, I do not combine color transformations if both transformations are applying the exact opposite. For example, if the first transformation increases the brightness of the image and the second transformation decreases the brightness. And I do not combine multiple rotations. Finally, this new set of transformations, which I will refer to as the set of double combinations, includes 162 different trans-

Algorithm	Single (19 transformations)	Double (162 transformations)
Greedy	95.87%	95.93%
Random Walk	95.87%	95.81%
EA Own	96.00%	96.01%
SimpleGA	95.95%	96.04%
Score Based	95.96%	96.11%
Majority Voting	95.89%	95.81%
Averaging (probabilities)	95.91%	96.06%
Averaging (logits)	95.83%	95.80%

Table 7.1: COMBINATION OF TRANSFORMATIONS. Comparison of the test accuracy when using single transformations and the set of double combinations for test-time augmentation on CIFAR-10.

formations. Then I run the same experiments mentioned in Section 5.1 using the set of double combinations. The results on CIFAR-10, compared with the results of using only the single transformations, can be found in Table 7.1. Greedy, EA Own, SimpleGA, Score Based, and Averaging (probabilities) seem to benefit from a more extensive set of transformations. Random Walk, Majority Voting, and Averaging (logits), on the other hand, deteriorate. Out of all methods evaluated on CIFAR-10, Score Based using 162 transformations results in the best test accuracy. Therefore, increasing the number and variation of test-time augmentations may indeed be beneficial. Some methods, for example, Greedy, will score at least as good by including more transformations because the choice of transformations increases. If the additional transformations are of no help, the algorithm will choose the same set as with using only single transformations. Therefore, it is not surprising that Greedy scores better using the set of double combinations. Although enlarging the set of transformations seems to improve the accuracy in most cases, it also extends the total computation time.

7.1.2 Clusters

For both the greedy and the weighted approaches introduced in Sections 5.1.1 and 5.1.2, I fit the algorithm on the entire set of validation images and use the optimal subset or weight on the test images. However, it is possible that depending on the nature of the images, different validation images would result in different subsets of transformations or different optimal weights. In this additional experiment, I cluster the images based on their deep features, i.e., the output of the second to last layer of the trained ResNet-18 model, which is an average pooling layer. I only test this approach on CIFAR-10 because it is sufficiently small to conduct new experiments. Additionally, I only evaluate this attempt using Greedy. Specifically, for each image in the test set, I find the k most similar images in the validation set based on the deep features, run the greedy algorithm to receive a subset of transformations, and use the found subset of transformations on the test image to make a prediction. I choose a very simple similarity function C defined as:

$$C(A, B) = \sum_{\substack{i \in \{1,\dots,n\}, \\ j \in \{1,\dots,m\}}} (A_{ij} - B_{ij})^2$$

where A and B are two matrices representing the deep features of two images, and n and m represent the dimension of those matrices. Additionally, I run the experiment several times with different values for k. All the results are given in Table 7.2. Only the experiments with $k = 1000$

k	Test Score
5	95.44%
10	95.33%
50	95.37%
100	95.71%
500	95.80%
1000	95.95%
2000	95.83%
3000	95.96%
No clustering	95.87%

Table 7.2: CLUSTERS. Results of using clustering before running Greedy on CIFAR-10. k represents the cluster size.

and $k = 3000$ improve the greedy approach without clustering. The improvements are minimal compared to the increased computation time of calculating the k most similar validation images for each test image.

7.2 Limitations

In this section, I will discuss potential limitations in my study. There are multiple times where I make generalizations, where they may not be valid.

- For Random Walk, EA Own, and SimpleGA, I run grid search with different hyperparameters on CIFAR-10, MNIST, and Melanoma. I do not run grid search on ImageNet because of time concerns. Instead, I use similar hyperparameters found for CIFAR-10 but adjust them to reduce computation time. Therefore, results found on ImageNet for Random Walk, EA Own, and SimpleGA may not be optimal. Additionally, the choice of values to test for each of the hyperparameters is chosen arbitrarily and may not reflect the optimal values.
- Related to the point above, I do not repeat the grid search for the set of double combinations introduced in Section 7.1.1. Therefore, the results found in Table 7.1 may not be optimal for Random Walk, EA Own, and SimpleGA.
- All results in Table 6.1 are evaluated on a single validation and test set. It is unclear how the scores would differ on new validation and test sets. A more reliable approach would be to use cross-validation or resampling of the test set and save the average scores and standard deviations. However, this requires additional computation time.
- Because some transformations, for example, random cropping, contain random elements, the logits differ depending on the random seed. Consequently, the final test scores for each method differ as well, depending on the chosen seed.
- For simplicity, I use the same ResNet-18 architecture for all datasets and only adapt the learning rate if the model seems to learn very poorly. Because each dataset is quite different, generalizing one model architecture to all datasets may not be the best approach. In fact, there are many other reasons to believe that the training scores found in Table 4.3 are not optimal. In Chapter 3 I discuss related work corresponding to data augmentation. A lot of work has been done to improve data augmentation during training, such as the automatic selection of transformations. I only use simple transformation functions and do not select

them automatically. Therefore, it is possible that my final test scores would differ if the models were trained with a different architecture, different hyperparameters, and different data augmentations. I cannot guarantee that my proposed test-time augmentation methods that score best would also score best using an entirely different model or training procedure.

- Although I test four different datasets, I cannot guarantee that my results hold for other applications as well. However, results that are consistent among CIFAR-10, MNIST, ImageNet, and Melanoma, may be more likely to generalize to other datasets.
- Finally, in my experiments, I primarily work with logits. Only in the case of averaging probabilities do I convert the logits to probabilities. However, in many cases, the last layer of the network is a softmax layer that outputs probabilities. All methods introduced in Chapter 5 can be applied to matrices of probabilities as well, but the final results might differ.

Chapter 8

Conclusion and Future Work

I have demonstrated on four different datasets that test-time augmentation for image classification tasks positively affects image classification accuracy. I showed that there are different methods for test-time augmentation. The most common methods are averaging the probabilities (or logits) of each transformation and majority voting. Additionally, I implemented a greedy algorithm and four different weighted approaches. The greedy algorithm searches for the best subset of transformations to aggregate instead of aggregating all transformations. The goal of the weighted approach is to find optimal weights for each transformation. The final predictions are then aggregated according to those weights. Details of this method are provided in Section 5.1.2. In my experiments, I have demonstrated that the standard approaches of averaging and majority voting often perform worse than the greedy or weighted methods. In the following sections, I will further discuss my conclusions of this thesis, and finally, I will offer some suggestions for future work.

8.1 Conclusion

All test-time augmentation methods provide significantly better test scores than using no test-time augmentation. Therefore, transforming the test images and aggregating the individual results to form a final prediction improves the accuracy of the image recognition model. EA Own, which belongs to the weighted approach, where the weights are not necessarily normalized, but they are non-negative, scored best on CIFAR-10 and Melanoma. SimpleGA, which also belongs to the weighted approach, and does not have any restrictions on the weights, scored best on MNIST and ImageNet.

In some cases, for example, in medical diagnosis, even the slightest improvement in accuracy can be of great importance. Therefore, including test-time augmentation techniques is beneficial. On the other hand, adding test-time augmentations extends the computation time. Instead of feeding just the original image to the model, each image must be transformed multiple times and fed to the model. Then, the individual outputs must be aggregated to a final prediction. When using the greedy approach, or the weighted approach, the optimal subset of transformations or the optimal weights must first be calculated. This step adds extra computation time during training. Test-time augmentation is, therefore, instrumental in applications where a slight increase in accuracy is much more critical than fast performance. For self-driving cars, for example, this might not be the case because they must be able to recognize new images quickly. Table 7.1 shows that using a more extensive set of transformations can improve the accuracy of some methods. However, the larger the set of transformations, the longer the computation time during training and testing. Similarly, the clustering attempt mentioned in Section 7.1.2 shows that for specific cluster sizes k, there may be a slight improvement in the accuracy. However, the step of additionally

clustering the images increases the overall computation time. Out of the simple methods, which require transforming the test images multiple times and aggregating individual predictions but do not require an additional fitting step, averaging the probabilities seems to score best. This is the method that is most commonly applied.

In conclusion, my results from several different experiments and on four different datasets show that test-time augmentation, in general, improves the overall test score. Still, depending on the dataset and the aggregating method, the improvement may be only minimal. Thus, depending on the specific dataset and problem, the increased computation time may not be worth the incremental increase in the test score. However, in many cases, I believe that the advantage in the classification score using test-time augmentation and using more advanced techniques than averaging or majority voting outweighs the disadvantages. This is especially the case with image recognition tasks for medical diagnoses, such as the diagnosis of brain tumors from brain MRIs.

8.2 Future Work

This thesis provides the groundwork for improving test-time augmentation techniques. However, there are still many open questions.

- In my experiments, I find that using a weighted approach for test-time augmentation performs better than other methods. However, it remains unclear which weight optimization algorithm to use and with what restrictions. The goal would be to find an optimization algorithm that scores best on all datasets.
- The computation time of the weighted approaches is a concern. Many other optimization algorithms besides the three algorithms that I tested exist, and a more efficient algorithm might result in the same score improvement and less computation time than my proposed algorithms.
- Instead of using the same model architecture and similar hyperparameters and transformation functions for each dataset, it would be useful to make individual adjustments.
- As mentioned already in Chapter 7, I use very similar transformations both for training and testing. It would be interesting to investigate whether this is necessary or if transformations can differ without affecting the test score.
- In Section 7.1 I implement and evaluate two experiments on CIFAR-10, a combination of transformations and clustering, and find that in some cases, there is indeed an improvement in the test score. However, both methods require more work. First, it is unclear what the ideal number of transformations for test-time augmentation should be and whether results found on CIFAR-10 generalize to other datasets as well. Second, when finding clusters among the images, I use a very simple similarity function. Other functions and other image features might improve this method.
- Because I focused more on the testing stage of the model rather than the training stage, my training scores are not as good compared to current benchmarks. For future research, it would be interesting to test whether some of my proposed test-time augmentation methods can further improve the current best models.
- It would be interesting to investigate whether machine learning algorithms, such as Random Forest, could be helpful for test-time augmentation. For example, a model could be trained on features extracted from the logits of each transformation, which predicts the final class.

- Finally, it would be helpful if the set of best transformations for each dataset is known before applying test-time augmentation. This would likely improve the test scores and speed up the process. Therefore, it would be interesting to run test-time augmentation on multiple different datasets and investigate patterns. Ideally, one could limit the set of transformations based on the nature of the dataset.

Appendix A

Hyperparameters

Table A.1 shows the learning rates and batch sizes that I use for each data augmentation method during training. Listings A.1, A.2, and A.3 give the exact list and ordering of the transformations used for test-time augmentation.

Dataset	DA	Learning Rate	Batch Size
CIFAR-10	No	0.001	16
	Yes	0.001	16
MNIST	No	0.001	16
	Yes	0.001	16
ImageNet	No	0.001	64
	Yes	0.001	64
Melanoma	No	0.0001	16
	Yes	0.0001	16

Table A.1: HYPERPARAMETERS FOR TRAINING. Chosen learning rates and batch sizes for each dataset with and without data augmentation (DA) during training.

```
Original
Horizontal Flip
Crop with proportion 0.8
Color Jitter with brightness 0.5
Color Jitter with brightness 1.5
Color Jitter with contrast 0.5
Color Jitter with contrast 1.5
Color Jitter with saturation 0
Color Jitter with saturation 2
Color Jitter with hue -0.2
Color Jitter with hue 0.2
Random Noise with standard deviation 0.05
Shift with width proportion 0.2
Shift with height proportion 0.2
Rotation with angle 5
Rotation with angle 10
Rotation with angle 15
Rotation with angle 20
Random Erasing with scale (0.1, 0.1)
```

Listing A.1: Data augmentation transformations with hyperparameters for test-time augmentation on CIFAR-10 and ImageNet.

```
Original
Crop with proportion 0.8
Color Jitter with brightness 0.5
Color Jitter with brightness 1.5
Color Jitter with contrast 0.5
Color Jitter with contrast 1.5
Random Noise with standard deviation 0.05
Shift with width proportion 0.2
Shift with height proportion 0.2
Rotation with angle 5
Rotation with angle 10
Rotation with angle 15
Rotation with angle 20
Random Erasing with scale (0.1, 0.1)
```

Listing A.2: Data augmentation transformations with hyperparameters for test-time augmentation on MNIST.

```
Original
Horizontal Flip
Crop with proportion 0.8
Color Jitter with brightness 0.6
Color Jitter with brightness 1.4
Color Jitter with contrast 0.6
Color Jitter with contrast 1.4
Color Jitter with saturation 0
Color Jitter with saturation 2
Color Jitter with hue -0.2
Color Jitter with hue 0.2
Random Noise with standard deviation 0.05
Shift with width proportion 0.1
Shift with height proportion 0.1
Rotation with angle 45
Rotation with angle 90
Rotation with angle 135
Rotation with angle 180
Random Erasing with scale (0.1, 0.1)
```

Listing A.3: Data augmentation transformations with hyperparameters for test-time augmentation on Melanoma.

Appendix B

Grid Search Results

I run a grid search for specific hyperparameters of Random Walk, EA Own, and SimpleGA for the datasets CIFAR-10, MNIST, and Melanoma. I do not run the grid search on ImageNet because of time concerns. Additionally, I run each combination of hyperparameters 20 times with different random seeds and provide the average validation scores. Tables B.1, B.2, and B.3 give the results of grid search for Random Walk. Tables B.4, B.5, and B.6 show the results of grid search for EA Own. And finally, Tables B.7, B.8, and B.9 give the results of grid search on SimpleGA. The red boxes in the tables highlight the chosen combination of hyperparameters for the respective dataset. If several combinations score equally well, I choose the hyperparameters that result in the fastest computation time.

early_stopping	normalization	initialization	Validation Accuracy (%)
10	True	Random	97.09
10	True	Equal	97.22
10	False	Random	97.15
10	False	Equal	97.20
100	True	Random	97.20
100	True	Equal	97.23
100	False	Random	97.18
100	False	Equal	97.20
200	True	Random	97.23
200	True	Equal	97.25
200	False	Random	97.16
200	False	Equal	97.20
500	True	Random	97.19
500	True	Equal	97.29
500	False	Random	97.21
500	False	Equal	97.20

Table B.1: CIFAR-10 RANDOM WALK. Grid search results for Random Walk on CIFAR-10 with different values for early stopping, normalization, and initialization.

early_stopping	normalization	initialization	Validation Accuracy (%)
10	True	Random	99.79
10	True	Equal	99.76
10	False	Random	99.78
10	False	Equal	99.76
100	True	Random	99.78
100	True	Equal	99.78
100	False	Random	99.78
100	False	Equal	99.76
200	True	Random	99.78
200	True	Equal	99.78
200	False	Random	99.77
200	False	Equal	99.76
500	True	Random	99.77
500	True	Equal	99.78
500	False	Random	99.77
500	False	Equal	99.76

Table B.2: MNIST RANDOM WALK. Grid search results for Random Walk on MNIST with different values for early stopping, normalization, and initialization.

early_stopping	normalization	initialization	Validation AUC Score
10	True	Random	0.8939
10	True	Equal	0.8931
10	False	Random	0.8979
10	False	Equal	0.8995
100	True	Random	0.8899
100	True	Equal	0.8905
100	False	Random	0.8975
100	False	Equal	0.8995
200	True	Random	0.8906
200	True	Equal	0.8910
200	False	Random	0.8968
200	False	Equal	0.8995
500	True	Random	0.8905
500	True	Equal	0.8900
500	False	Random	0.8963
500	False	Equal	0.8995

Table B.3: MELANOMA RANDOM WALK. Grid search results for Random Walk on Melanoma with different values for early stopping, normalization, and initialization.

n_elites	n	initialization	early_stopping	Validation Accuracy (%)
5	100	random	10	97.29
5	100	random	50	97.30
5	100	ones	10	97.33
5	100	ones	50	97.28
5	500	random	10	97.36
5	500	random	50	97.34
5	500	ones	10	97.35
5	500	ones	50	97.32
5	1000	random	10	97.33
5	1000	random	50	97.35
5	1000	ones	10	97.35
5	1000	ones	50	97.37
10	100	random	10	97.32
10	100	random	50	97.31
10	100	ones	10	97.29
10	100	ones	50	97.32
10	500	random	10	97.34
10	500	random	50	97.33
10	500	ones	10	97.36
10	500	ones	50	97.34
10	1000	random	10	97.42
10	1000	random	50	97.34
10	1000	ones	10	97.35
10	1000	ones	50	97.37
20	100	random	10	97.30
20	100	random	50	97.32
20	100	ones	10	97.30
20	100	ones	50	97.32
20	500	random	10	97.34
20	500	random	50	97.31
20	500	ones	10	97.35
20	500	ones	50	97.33
20	1000	random	10	97.36
20	1000	random	50	97.32
20	1000	ones	10	97.38
20	1000	ones	50	97.34
50	100	random	10	97.31
50	100	random	50	97.31
50	100	ones	10	97.30
50	100	ones	50	97.28
50	500	random	10	97.33
50	500	random	50	97.31
50	500	ones	10	97.33
50	500	ones	50	97.33
50	1000	random	10	97.34
50	1000	random	50	97.32
50	1000	ones	10	97.35
50	1000	ones	50	97.32

Table B.4: CIFAR-10 EA OWN. Grid search results for EA Own on CIFAR-10 with different values for number of elites to keep, number of vectors to generate `n`, initialization method, and early stopping.

n_elites	n	initialization	early_stopping	Validation Accuracy (%)
5	100	random	10	99.798
5	100	random	50	99.798
5	100	ones	10	99.802
5	100	ones	50	99.802
5	500	random	10	99.798
5	500	random	50	99.798
5	500	ones	10	99.798
5	500	ones	50	99.798
5	1000	random	10	99.794
5	1000	random	50	99.800
5	1000	ones	10	99.800
5	1000	ones	50	99.798
10	100	random	10	99.800
10	100	random	50	99.800
10	100	ones	10	99.800
10	100	ones	50	99.794
10	500	random	10	99.798
10	500	random	50	99.800
10	500	ones	10	99.800
10	500	ones	50	99.800
10	1000	random	10	99.800
10	1000	random	50	99.798
10	1000	ones	10	99.800
10	1000	ones	50	99.800
20	100	random	10	99.802
20	100	random	50	99.804
20	100	ones	10	99.800
20	100	ones	50	99.798
20	500	random	10	99.800
20	500	random	50	99.800
20	500	ones	10	99.800
20	500	ones	50	99.800
20	1000	random	10	99.798
20	1000	random	50	99.800
20	1000	ones	10	99.800
20	1000	ones	50	99.800
50	100	random	10	99.800
50	100	random	50	99.798
50	100	ones	10	99.800
50	100	ones	50	99.798
50	500	random	10	99.802
50	500	random	50	99.800
50	500	ones	10	99.800
50	500	ones	50	99.800
50	1000	random	10	99.800
50	1000	random	50	99.798
50	1000	ones	10	99.800
50	1000	ones	50	99.800

Table B.5: MNIST EA OWN. Grid search results for EA Own on MNIST with different values for number of elites to keep, number of vectors to generate `n`, initialization method, and early stopping.

n_elites	n	initialization	early_stopping	Validation AUC Score
5	100	random	10	0.8903
5	100	random	50	0.8893
5	100	ones	10	0.8903
5	100	ones	50	0.8895
5	500	random	10	0.8896
5	500	random	50	0.8890
5	500	ones	10	0.8896
5	500	ones	50	0.8891
5	1000	random	10	0.8894
5	1000	random	50	0.8890
5	1000	ones	10	0.8893
5	1000	ones	50	0.8891
10	100	random	10	0.8907
10	100	random	50	0.8900
10	100	ones	10	0.8905
10	100	ones	50	0.8905
10	500	random	10	0.8897
10	500	random	50	0.8891
10	500	ones	10	0.8897
10	500	ones	50	0.8891
10	1000	random	10	0.8895
10	1000	random	50	0.8891
10	1000	ones	10	0.8893
10	1000	ones	50	0.8891
20	100	random	10	0.8909
20	100	random	50	0.8904
20	100	ones	10	0.8905
20	100	ones	50	0.8902
20	500	random	10	0.8902
20	500	random	50	0.8892
20	500	ones	10	0.8902
20	500	ones	50	0.8892
20	1000	random	10	0.8897
20	1000	random	50	0.8891
20	1000	ones	10	0.8898
20	1000	ones	50	0.8891
50	100	random	10	0.8913
50	100	random	50	0.8904
50	100	ones	10	0.8913
50	100	ones	50	0.8905
50	500	random	10	0.8906
50	500	random	50	0.8902
50	500	ones	10	0.8904
50	500	ones	50	0.8902
50	1000	random	10	0.8903
50	1000	random	50	0.8893
50	1000	ones	10	0.8903
50	1000	ones	50	0.8893

Table B.6: MELANOMA EA OWN. Grid search results for EA Own on Melanoma with different values for number of elites to keep, number of vectors to generate n, initialization method, and early stopping.

population_size	standard_deviation	elite_proportion	Validation Accuracy (%)
20	0.2	0.1	97.22
20	0.2	0.2	97.22
20	0.2	0.3	97.22
20	0.5	0.1	97.18
20	0.5	0.2	97.26
20	0.5	0.3	97.25
20	0.7	0.1	97.19
20	0.7	0.2	97.24
20	0.7	0.3	97.25
100	0.2	0.1	97.24
100	0.2	0.2	97.25
100	0.2	0.3	97.22
100	0.5	0.1	97.25
100	0.5	0.2	97.26
100	0.5	0.3	97.26
100	0.7	0.1	97.21
100	0.7	0.2	97.24
100	0.7	0.3	97.26
250	0.2	0.1	97.27
250	0.2	0.2	97.27
250	0.2	0.3	97.30
250	0.5	0.1	97.26
250	0.5	0.2	97.29
250	0.5	0.3	97.32
250	0.7	0.1	97.30
250	0.7	0.2	97.33
250	0.7	0.3	97.28

Table B.7: CIFAR-10 SIMPLEGA. Grid search results for SimpleGA on CIFAR-10 with different values for population size, standard deviation, and elite proportion.

population_size	standard_deviation	elite_proportion	Validation Accuracy (%)
20	0.2	0.1	99.82
20	0.2	0.2	99.83
20	0.2	0.3	99.83
20	0.5	0.1	99.81
20	0.5	0.2	99.83
20	0.5	0.3	99.83
20	0.7	0.1	99.82
20	0.7	0.2	99.81
20	0.7	0.3	99.82
100	0.2	0.1	99.81
100	0.2	0.2	99.83
100	0.2	0.3	99.82
100	0.5	0.1	99.82
100	0.5	0.2	99.83
100	0.5	0.3	99.83
100	0.7	0.1	99.82
100	0.7	0.2	99.84
100	0.7	0.3	99.83
250	0.2	0.1	99.83
250	0.2	0.2	99.83
250	0.2	0.3	99.83
250	0.5	0.1	99.83
250	0.5	0.2	99.84
250	0.5	0.3	99.83
250	0.7	0.1	99.83
250	0.7	0.2	99.83
250	0.7	0.3	99.84

Table B.8: MNIST SIMPLEGA. Grid search results for SimpleGA on MNIST with different values for population size, standard deviation, and elite proportion.

population_size	standard_deviation	elite_proportion	Validation AUC Score
20	0.2	0.1	0.8716
20	0.2	0.2	0.8745
20	0.2	0.3	0.8748
20	0.5	0.1	0.8738
20	0.5	0.2	0.8734
20	0.5	0.3	0.8740
20	0.7	0.1	0.8762
20	0.7	0.2	0.8760
20	0.7	0.3	0.8726
100	0.2	0.1	0.8720
100	0.2	0.2	0.8729
100	0.2	0.3	0.8733
100	0.5	0.1	0.8738
100	0.5	0.2	0.8728
100	0.5	0.3	0.8738
100	0.7	0.1	0.8721
100	0.7	0.2	0.8740
100	0.7	0.3	0.8739
250	0.2	0.1	0.8735
250	0.2	0.2	0.8730
250	0.2	0.3	0.8740
250	0.5	0.1	0.8746
250	0.5	0.2	0.8744
250	0.5	0.3	0.8737
250	0.7	0.1	0.8715
250	0.7	0.2	0.8737
250	0.7	0.3	0.8732

Table B.9: MELANOMA SIMPLEGA. Grid search results for SimpleGA on Melanoma with different values for population size, standard deviation, and elite proportion.

Appendix C

Optimal Weights

Tables C.1, C.2, C.3, and C.4 give the optimal weights found by Random Walk, EA Own, SimpleGA, and Score Based. Because I run Random Walk, EA Own, and SimpleGA 20 times with different seeds, the weights provided here are averaged over the 20 repetitions. The highlighted transformations are the three first transformations found by Greedy. They are indicative of the most important transformations per dataset according to the greedy algorithm. Additionally, the highlighted weights indicate the three highest weights per method. They should also indicate the most important transformations according to each respective weighted approach method.

Transformation	Random Walk	EA Own	SimpleGA	Score Based
Original	0.0543	0.1586	-1.7896	**0.9602**
Horizontal Flip	**0.0855**	0.1160	**9.8699**	**0.9596**
Random Crop	**0.0817**	**7.5453**	**6.3778**	0.9450
Color Jitter (brightness 0.5)	0.0551	0.1184	-0.9076	0.9556
Color Jitter (brightness 1.5)	0.0375	0.0189	-0.3975	0.0000
Color Jitter (contrast 0.5)	**0.0678**	2.2698	2.7202	0.9546
Color Jitter (contrast 1.5)	0.0327	0.0114	-3.2860	0.9484
Color Jitter (saturation 0)	0.0286	**4.2538**	2.5470	0.0000
Color Jitter (saturation 2)	0.0472	0.2363	1.3980	0.9514
Color Jitter (hue -0.2)	0.0331	0.0546	0.5353	0.0000
Color Jitter (hue 0.2)	0.0476	0.0698	1.6260	0.0000
Random Noise	0.0448	0.0377	-0.5249	0.0000
Vertical Shift	0.0647	0.0328	0.6722	0.0000
Horizontal Shift	0.0582	0.0286	1.2511	0.9464
Rotation (angle 5)	0.0411	**3.4339**	**5.3120**	**0.9573**
Rotation (angle 10)	0.0551	1.7683	3.3541	0.9522
Rotation (angle 15)	0.0515	0.2857	-0.2074	0.9458
Rotation (angle 20)	0.0586	0.2856	2.0116	0.0000
Random Erasing	0.0550	3.2794	3.0093	0.0000

Table C.1: CIFAR-10 WEIGHTS. Optimal weights found by the four algorithms for the weighted approach on CIFAR-10. The highlighted transformations and weights are the three most important transformations based on the different methods.

Transformation	Random Walk	EA Own	SimpleGA	Score Based
Original	0.0479	0.2601	-1.6942	**0.9963**
Random Crop	**0.0924**	6.8521	1.8947	0.9954
Color Jitter (brightness 0.5)	0.0750	0.2964	0.1279	**0.9963**
Color Jitter (brigtness 1.5)	0.0678	**11.0821**	**3.3072**	0.9949
Color Jitter (contrast 0.5)	0.0726	**10.0107**	**3.4854**	0.9962
Color Jitter (contrast 1.5)	0.0642	0.2321	1.2917	0.9940
Random Noise	**0.0795**	**6.8960**	**3.3887**	0.9961
Vertical Shift	0.0640	0.3462	-0.5367	0.0000
Horizontal Shift	0.0755	0.2761	-1.2251	0.9950
Rotation (angle 5)	0.0715	0.3704	-0.2260	**0.9965**
Rotation (angle 10)	0.0711	0.6854	2.3089	0.9950
Rotation (angle 15)	0.0621	0.5371	0.8701	0.9949
Rotation (angle 20)	**0.0866**	3.1081	-0.0113	0.0000
Random Erasing	0.0698	6.2348	2.0552	0.0000

Table C.2: MNIST WEIGHTS. Optimal weights found by the four algorithms for the weighted approach on MNIST. The highlighted transformations and weights are the three most important transformations based on the different methods.

Transformation	Random Walk	EA Own	SimpleGA	Score Based
Original	0.0778	**3.6648**	**3.9622**	**0.7320**
Horizontal Flip	**0.1217**	**10.3063**	**9.9357**	**0.7320**
Random Crop	**0.1032**	**5.4414**	**5.5554**	0.0000
Color Jitter (brightness 0.5)	0.0330	0.0987	-2.3761	0.0000
Color Jitter (brightness 1.5)	0.0569	0.6289	1.2038	0.0000
Color Jitter (contrast 0.5)	0.0442	0.2594	0.9341	0.0000
Color Jitter (contrast 1.5)	0.0604	0.5332	1.3556	0.0000
Color Jitter (saturation 0)	0.0347	0.4357	1.0808	0.0000
Color Jitter (saturation 2)	0.0362	0.5301	1.0075	0.0000
Color Jitter (hue -0.2)	0.0215	0.2639	0.5395	0.0000
Color Jitter (hue 0.2)	0.0220	0.2193	0.1658	0.0000
Random Noise	0.0372	0.4322	1.3329	0.0000
Vertical Shift	0.0811	3.3330	3.4466	0.0000
Horizontal Shift	**0.0919**	2.9224	2.2960	0.0000
Rotation (angle 5)	0.0197	0.0269	-3.4991	0.0000
Rotation (angle 10)	0.0283	0.1974	0.5681	0.0000
Rotation (angle 15)	0.0423	0.0352	-0.3446	0.0000
Rotation (angle 20)	0.0365	0.0354	0.2404	0.0000
Random Erasing	0.0523	0.1781	0.3149	0.0000

Table C.3: IMAGENET WEIGHTS. Optimal weights found by the four algorithms for the weighted approach on ImageNet. The highlighted transformations and weights are the three most important transformations based on the different methods.

Transformation	Random Walk	EA Own	SimpleGA	Score Based
Original	1.000	0.2411	0.5063	0.0000
Horizontal Flip	1.000	0.3488	0.7238	0.0000
Random Crop	1.000	0.4397	2.2090	0.0000
Color Jitter (brightness 0.6)	1.000	0.0735	-2.4806	0.0000
Color Jitter (brightness 1.4)	1.000	0.2564	0.1565	0.0000
Color Jitter (contrast 0.6)	1.000	0.0560	-3.3363	0.0000
Color Jitter (contrast 1.4)	1.000	0.3903	1.4561	0.0000
Color Jitter (saturation 0)	1.000	0.2917	0.1629	0.0000
Color Jitter (saturation 2)	1.000	0.2011	-1.5542	0.0000
Color Jitter (hue -0.2)	1.000	1.5959	2.7304	0.0000
Color Jitter (hue 0.2)	1.000	0.1491	-2.2312	0.0000
Random Noise	1.000	0.1993	-0.1010	0.0000
Vertical Shift	1.000	**7.2606**	**8.2523**	**0.8808**
Horizontal Shift	1.000	0.1102	-5.9190	0.0000
Rotation (angle 45)	1.000	0.3759	0.7683	0.0000
Rotation (angle 90)	1.000	0.2805	-1.5776	0.0000
Rotation (angle 135)	1.000	**7.7241**	**4.3001**	0.0000
Rotation (angle 180)	1.000	1.3310	3.3788	**0.8717**
Random Erasing	1.000	**3.4235**	**4.9039**	0.0000

Table C.4: MELANOMA WEIGHTS. Optimal weights found by the four algorithms for the weighted approach on Melanoma. The highlighted transformations and weights are the three most important transformations based on the different methods.

List of Figures

List of Tables

List of Listings

Bibliography

Antoniou, A., Storkey, A., and Edwards, H. (2018). Data Augmentation Generative Adversarial Networks. *arXiv preprint arXiv:1711.04340* [stat.ML].

Cai, L., Gao, J., and Zhao, D. (2020). A review of the application of deep learning in medical image classification and segmentation. *Annals of Translational Medicine*, 8(11).

Chen, P., Liu, S., Zhao, H., and Jia, J. (2020). GridMask Data Augmentation. *arXiv preprint arXiv:2001.04086* [cs.CV].

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2019). Autoaugment: Learning augmentation strategies from data. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June(Section 3):113–123.

Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Deng, L. (2012). The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, 29(6):141–142.

Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition.

Ha, D. (2017). Evolving Stable Strategies. http://blog.otoro.net/2017/11/12/evolving-stable-strategies/. [Online; accessed 23-February-2021].

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Ho, D., Liang, E., Stoica, I., Abbeel, P., and Chen, X. (2019). Population based augmentation: Efficient learning of augmentation policy schedules. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:4843–4856.

Howard, A. G. (2013). Some Improvements on Deep Convolutional Neural Network Based Image Classification. *arXiv preprint arXiv:1312.5402* [cs.CV].

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:2261–2269.

Inoue, H. (2018). Data Augmentation by Pairing Samples for Images Classification. *arXiv preprint arXiv:1801.02929* [cs.LG].

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. (2017). Population Based Training of Neural Networks. *arXiv preprint arXiv:1711.09846* [cs.LG].

Krizhevsky, A. (2012). Learning Multiple Layers of Features from Tiny Images. Technical Report TR-2009, University of Toronto, Toronto.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/.

Lim, S.-H., Young, S. R., and Patton, R. M. (2016). An analysis of image storage systems for scalable training of deep neural networks. *The seventh workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware (in conjunction with ASPLOS'16)*.

Mikołajczyk, A. and Grochowski, M. (2018). Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122.

Molchanov, D., Lyzhov, A., Molchanova, Y., Ashukha, A., and Vetrov, D. (2020). Greedy Policy Search: A Simple Baseline for Learnable Test-Time Augmentation. In *Proceedings of the 36thConference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020*, volume 124.

Perez, L. and Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *ArXiv preprint arXiv:1712.04621*.

Rotemberg, V., Kurtansky, N., Betz-Stablein, B., Caffery, L., Chousakos, E., Codella, N., Combalia, M., Dusza, S., Guitera, P., Gutman, D., Halpern, A., Helba, B., Kittler, H., Kose, K., Langer, S., Lioprys, K., Malvehy, J., Musthaq, S., Nanda, J., and Soyer, P. (2021). A patient-centric dataset of images and metadata for identifying melanomas using clinical context. *Scientific Data*, 8:34.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Shanmugam, D., Blalock, D., Balakrishnan, G., and Guttag, J. (2020). When and Why Test-Time Augmentation Works. *arXiv preprint arXiv:2011.11156* [cs.CV].

Shorten, C. and Khoshgoftaar, T. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6:1–48.

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Singhal, G. (2020). Transfer Learning with ResNet in PyTorch. https://www.pluralsight.com/guides/introduction-to-resnet. [Online; accessed 02-February-2021].

Summers, C. and Dinneen, M. J. (2019). Improved Mixed-Example Data Augmentation. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1262–1270.

Takahashi, R., Matsubara, T., and Uehara, K. (2020). Data Augmentation Using Random Image Cropping and Patching for Deep CNNs. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):2917–2931.

Zheng, Q., Yang, M., Tian, X., Jiang, N., and Wang, D. (2020). A full stage data augmentation method in deep convolutional neural network for natural image classification. *Discrete Dynamics in Nature and Society*, 2020.

Zhong, Z., Zheng, L., Kang, G., Li, S., and Yang, Y. (2020). Random Erasing Data Augmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):13001–13008.