POLITECNICO
MILANO 1863

Mitigating Class Dependency bias introduced with Data Augmentation

Christian Biffi 10787158
Alberto Cavallotti 10721275
Prof. Nicolò Felicioni

Academic year 2022/2023

Contents

1 Introduction

Our project began with a comprehensive study of neural networks, starting from the fundamental fully connected neural networks to the convolutional neural networks (CNNs) which we are utilizing for our work. After this preliminary study, we proceeded to analyze a recent paper [1] that explores the impact of data augmentation techniques and weight decay on neural networks. The paper reveals that, while these techniques lead to a model with an improved average quality, they also introduce a bias, leading to increased accuracy for certain classes while significantly reducing it for others. Building upon this observation, we are proposing a solution to address this problem by employing a specific architecture.

1.1 Basics of Neural Network

Neural networks represent a rapidly evolving field that has garnered significant attention and relevance across various disciplines in recent years. They offer the potential to address a multitude of problems in diverse domains, including but not limited to image classification, natural language processing, medical diagnostics, autonomous vehicles, and more.

The architecture of a neural network is designed to mimic the functioning of the human brain. It consists of interconnected nodes, referred to as artificial neurons, organized in layers. These layers typically include an input layer, one or more hidden layers, and an output layer. Each neuron receives inputs, processes them using a specific activation function, and produces an output. The connections between neurons, known as weights, are adjusted through a process called training, enabling the network to learn patterns and make predictions.
Training a neural network involves a critical step known as backpropagation. This algorithm iteratively adjusts the weights of the connections in the network based on the calculated error between the predicted output and the desired output, this calculation is based on a loss function and there are different types of it, each of one that is specific to an application. By repeatedly propagating this error backward through the network, the weights are fine-tuned to minimize the overall prediction error. This process allows the neural network to learn from labeled training data and generalize its knowledge to make accurate predictions on unseen data.

There are various types of neural networks, each designed to tackle specific tasks and data types. The fundamental architecture, which we initially used to comprehensively understand the functioning of neural networks, is the Fully Connected (FC) network. However, for our current project involving image classification, we are utilizing the Convolutional Neural Network (CNN) architecture.

1.1.1 Fully Connected network

A Fully Connected (FC) architecture is a type of neural network where all neurons in one layer are connected to every neuron in the subsequent layer. Each neuron in a layer receives inputs from all the neurons in the previous layer. The inputs are multiplied by corresponding weights and passed through an activation function, which introduces non-linearity into the network. The activation function helps determine the output value of each neuron based on the weighted sum of the inputs.
As previously said, there is an input layer, multiple hidden layer, and a final output layer that produces the final predictions or outputs based on the information processed through the network.

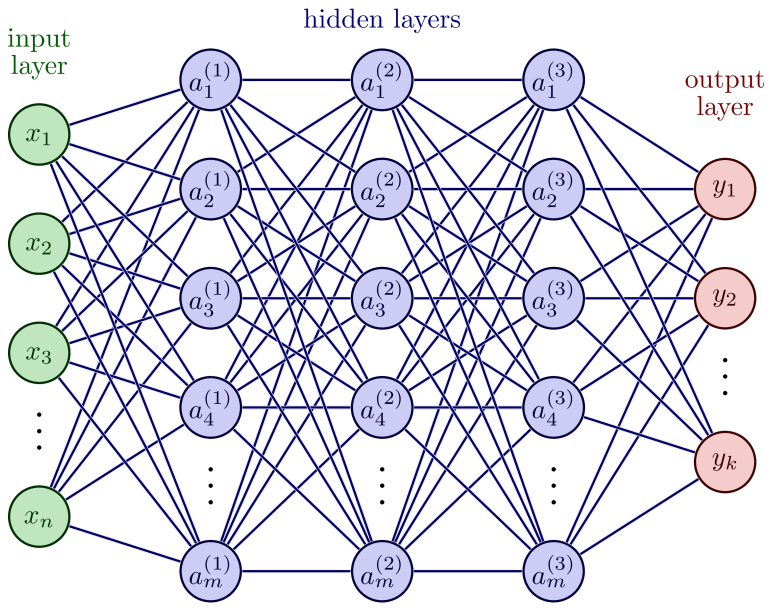

Figure 1: Example of a Fully Connected architecture

While fully connected architectures have been widely used in various applications, they can be computationally expensive and prone to overfitting when dealing with high-dimensional data. To address these limitations and efficiently handle tasks such as image processing, specialized architectures like Convolutional Neural Networks (CNNs) have been developed.

1.1.2 Convolutional Neural network

Convolutional Neural Networks (CNNs) excel in processing structured grid-like data, such as images, due to their ability to automatically learn and extract relevant features. CNNs possess a layered architecture that allows for efficient and hierarchical feature extraction in an automatic way. The usual structure of a CNN consists of convolutional layers, pooling layers, and fully connected layers.

The convolutional layer is the fundamental building block of the CNNs. It applies a set of learnable filters (also called *kernels*) to the input data, performing convolutions to extract local features. Over multiple iterations, the kernel sweeps over the entire image. At each iteration, the kernel is applied to different regions of the input data and the dot product is calculated between the input pixels and the filter. The final output from the series of dots is known as a feature map or convolved feature. This operation captures spatial dependencies and preserves the spatial arrangement of features. Non-linear activation functions, such as ReLU, are applied element-wise to the feature maps to introduce non-linearity.

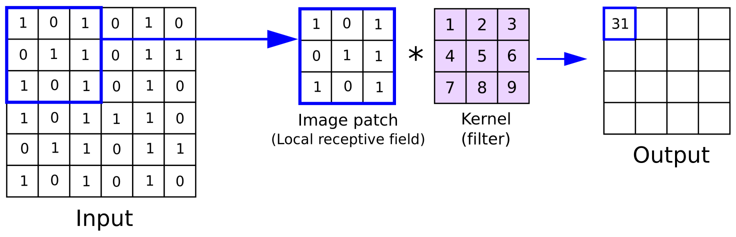

Figure 2: Visual representation of a convolutional layer

After a convolutional layer is often applied a *pooling* layer, which is a layer that compresses the data, reducing the spatial dimension. Common pooling techniques include max pooling and average pooling. Max pooling selects the maximum value within a local region, reducing the spatial resolution while retaining the most salient feature. Average pooling computes the average value within a region.

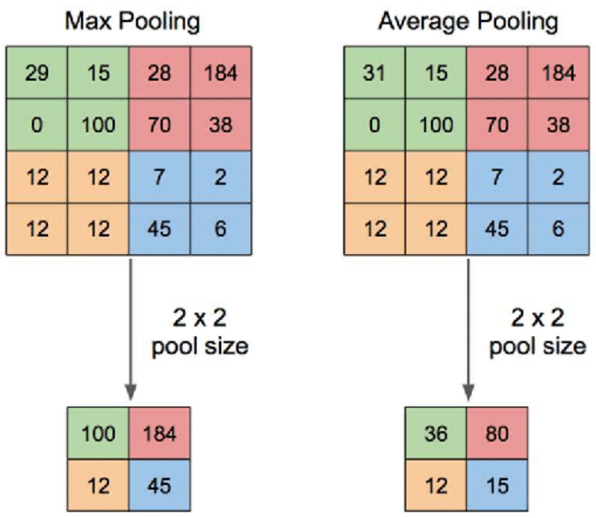

Figure 3: Visual representation of max and average pooling layer

These basic layers are typically repeated multiple times in a CNN architecture to extract increasingly complex features from the input images. This repetition allows the network to learn hierarchical representations, where lower layers capture simple local patterns, and higher layers capture more abstract and complex features.
The output of the last convolutional or pooling layer serves as the input to a fully connected layer in a CNN architecture. The fully connected layer operates similarly to a traditional neural network, where each neuron receives inputs from all neurons in the previous layer. This fully connected layer is responsible for the final image classification or any other task-specific prediction.

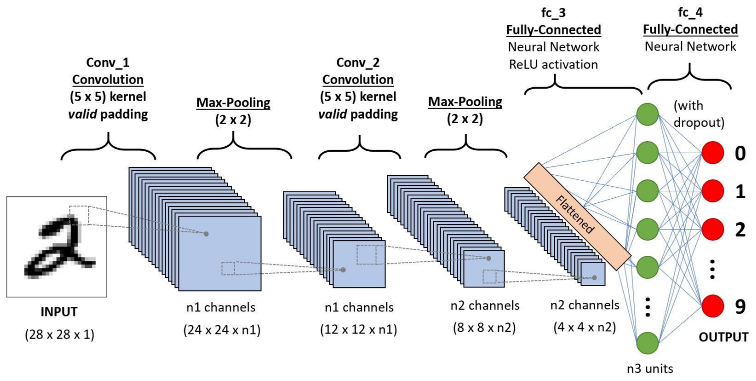

Figure 4: Complete architecture of a Convolutional Neural Network

1.2 Data augmentation bias

Data augmentation is a technique widely used when training a neural network to expand the dataset and its diversity. It consists of artificially expanding the training dataset through a variety of transformations, such as image rotation, flipping, scaling, or adding noise. By generating additional samples that are similar to the original data, data augmentation effectively enriches the training set, allowing models to generalize better and exhibit improved performance. As shown in the article by Balestriero, Bottou, and LeCun [1], it is proven that while the average test accuracy improves when using data augmentation, there is a strong per-class favoritism, with also some classes that are heavily penalized.

In this project we are trying to propose a solution that mitigate this effect and improve model fairness among all classes.

2 Implementation

Our implementation aims to address the issue of bias introduced by data augmentation through the creation of a "combined" neural network, composed of two separate sub-networks. The first sub-network is exclusively trained on the original images from the dataset, while the second sub-network is trained using augmented data.

The primary objective is to develop a network capable of distinguishing which of the two sub-networks' predictions should carry more weight for each specific image. This approach would like to ensure that predictions from the sub-network trained solely on original images receive greater importance for classes that demonstrate superior performance with such data. Conversely, predictions from the sub-network trained with augmented data are given more weight for classes that exhibit improved performance with augmented data.

In order to create our "Combined net", we drew inspiration from the implementation of a Siamese neural net [2]. We utilized all the layers of a normal ResNet18, excluding the last linear layer, and duplicated them in two separate sequential layers. Subsequently, we concatenated the outputs of the two sequential layers into a single linear layer, which generates predictions for our data.
This specialized network also takes two different images as input. To facilitate training of our network, we developed a specific dataloader. This dataloader selects an image from the dataset and produces two distinct images as outputs. One image remains in its original state, while the other undergoes a data augmentation technique by applying a selected lower bound crop.

3 Test

In our study, we utilized ResNet [3] as our base network architecture and the TinyImageNet200 dataset [4] for our experiments.
We conducted a series of experiments involving 13 different levels of random crop, which served as a form of data augmentation. Each network was trained from scratch for a total of 20 epochs.

The complete code is available at the Github repository [5].

3.1 ResNet 18

The initial step involved training the ResNet18 for each crop percentage. As illustrated in the following graph, even with limited training and a reduced dataset, noticeable variations in performance can be observed across different classes. Some classes exhibit significant degradation in performance, while others demonstrate substantial gains in accuracy.

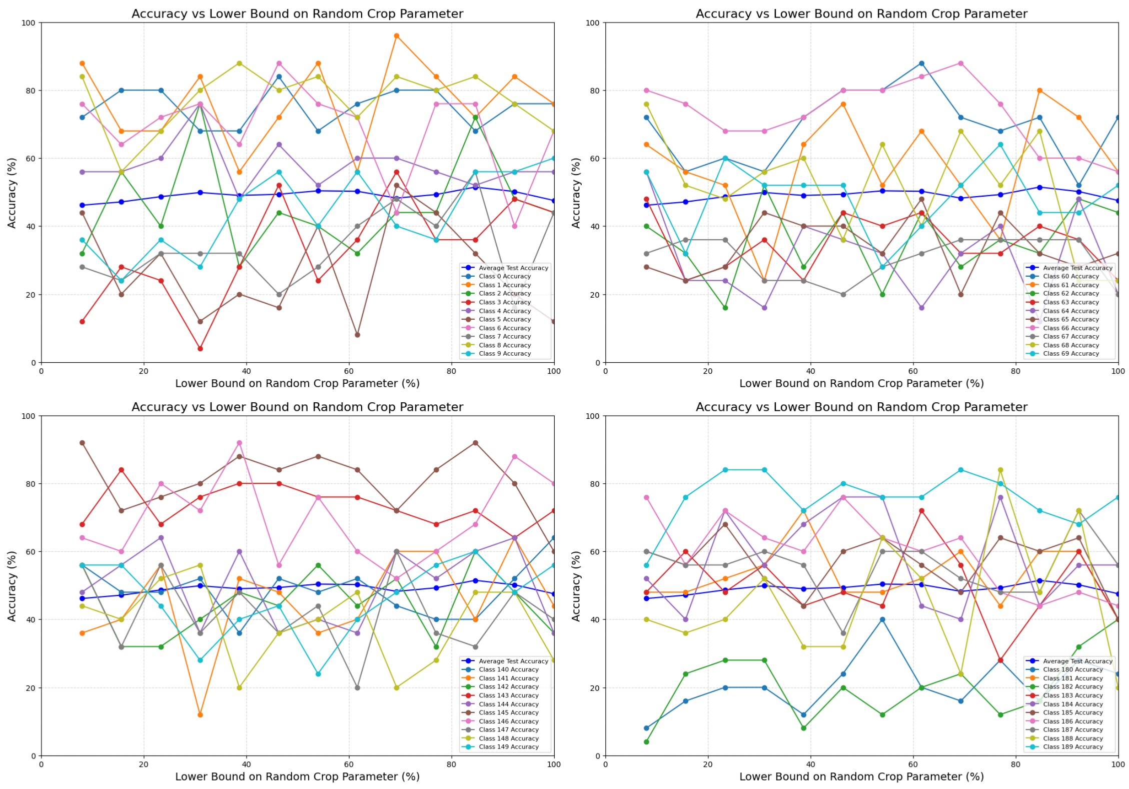

Figure 5: Specifically we can see that the class 3, 180 and 182 have a drop of about 20% or more in accuracy from the net trained solely on original images and the one trained with lower bound of random crop set at 0.08. Instead with the class 66, 145 and 186 we have a gain of about 20% or more in accuracy.

3.2 Combined net without Pre-Training

As a second step, we conducted tests on our initial implementation of the combined network, training it again from scratch for each different level of lower bound crop.
However, this approach did not yield significant results. Both the general accuracy and the accuracy for most of the individual classes were lower compared to training a single ResNet18 with the same number of epochs. We attribute this behavior to the limited number of epochs used to train

the network. It is likely that with a higher number of epochs, the training from scratch approach would surpass the performance of a single ResNet.

Despite the overall results not meeting our expectations, we did observe some improvements in addressing the bias introduced by data augmentation. This can be observed in the following images

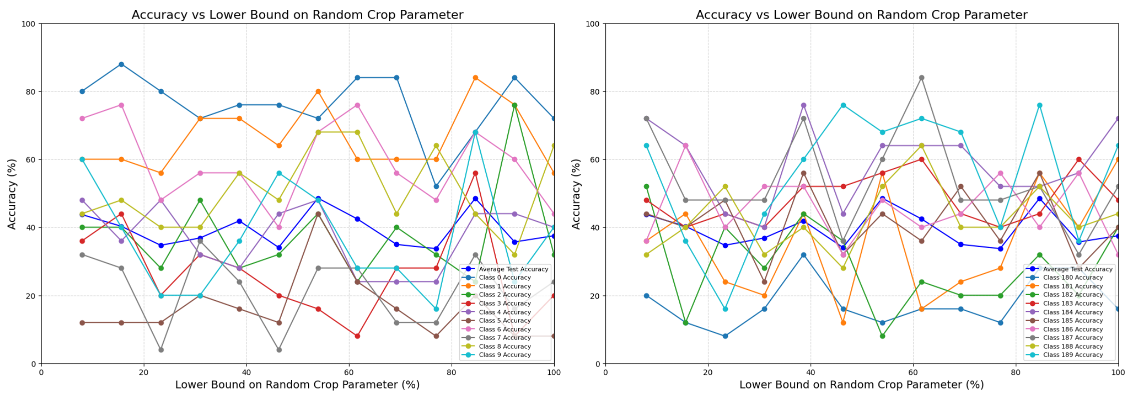

Figure 6: Classes 3, 180, and 182, which experienced a loss of approximately 20% in accuracy from the original images to the 0.08 lower bound random crop when using the single ResNet18, no longer exhibit this bias. Indeed, we can observe an improvement in their performance.

3.3 Combined net with Pre-Train

The third experiment we have done was still using the same approach of the previous combined net, but with a pre-train of the two separate sub-nets. So starting from the pre-trained ResNet18 with 20 epochs each, we froze all the layer before the linear layer and removed this last layer, we then only finetuned the last concatenated linear layer of the model for additional 10 epochs.

Surprisingly, this experiment resulted in a substantial improvement in the linearity of the results across all levels of lower bound of random crop applied. Additionally, we observed that this model effectively treated the classes more fairly when data augmentation was applied to the sub-net.

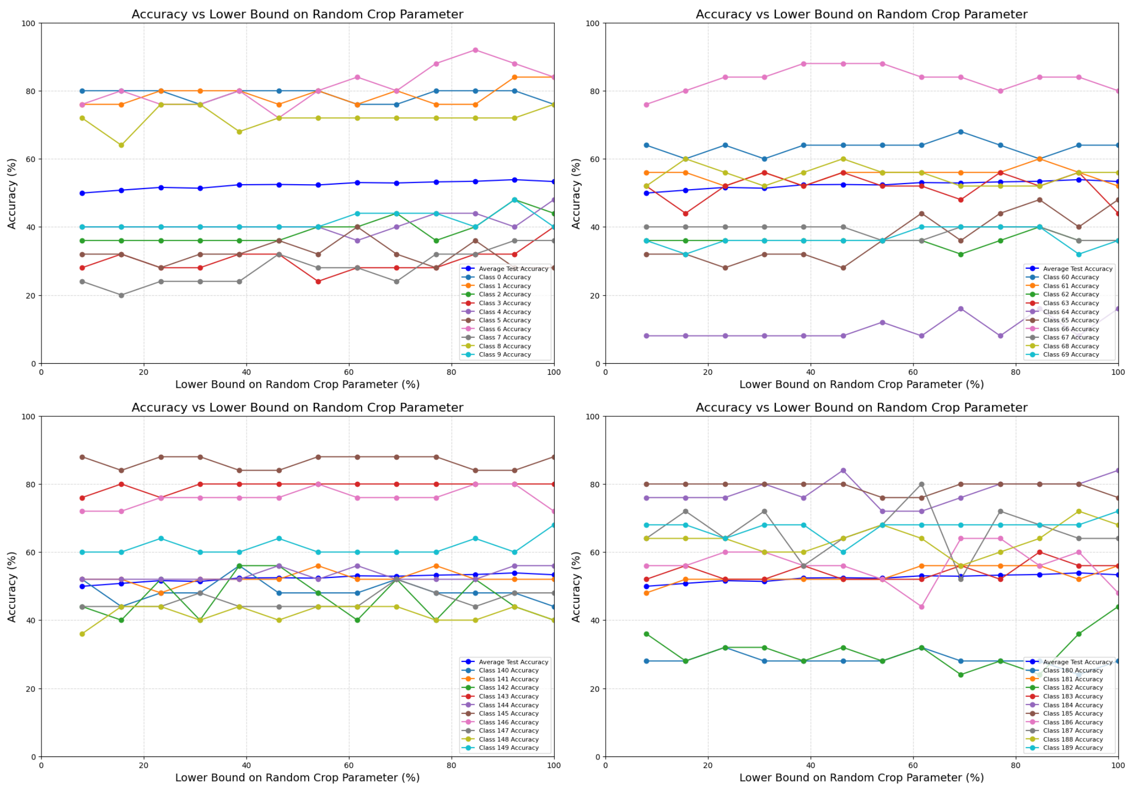

Figure 7: In this images we can see the linearization of the various classes compared to the same plot viewed with the single ResNet18.

3.4 ResNet 34

To obtain further confirmation of our results, we trained a ResNet34 model with a similar number of parameters as our combined net. The training process involved using the same 13 different crops and training for 20 epochs, just like the other experiments. By doing so, we aimed to rule out the possibility that our results were solely due to the larger number of parameters in the network. Our findings indicate that there is indeed an advantage in implementing a combined net, beyond the parameter count.

3.5 Comparing the results

In order to validate our results, we conducted a final comparison using various indexes to evaluate the performance of the different networks. Of particular interest are the CVaR 5%, which represents the mean accuracy of the worst 5% of classes in our dataset, and the Top 5%, which computes the mean accuracy of the top 5% of classes in our dataset.

3.5.1 ResNet18

```
+----------+----------+---------+---------+----------+---------+
|   Crop % |   Mean % |     Var |     Std |  CVaR 5% |  Top 5% |
|----------+----------+---------+---------+----------+---------|
| 0.08     |    46.5  | 399.508 | 19.9877 |      8.4 |    88.8 |
| 0.156667 |    48.54 | 404.049 | 20.101  |     13.6 |    86.4 |
| 0.233333 |    49.5  | 404.01  | 20.1    |     10.4 |    84.8 |
| 0.31     |    50.2  | 404.864 | 20.1212 |     12.8 |    88   |
| 0.386667 |    49.82 | 443.304 | 21.0548 |      8.8 |    87.6 |
| 0.463333 |    50.38 | 386.749 | 19.6659 |     14.8 |    84.4 |
| 0.54     |    50.78 | 397.7   | 19.9424 |     14.4 |    90   |
| 0.616667 |    50.22 | 419.489 | 20.4814 |     10.4 |    85.6 |
| 0.693333 |    48.86 | 396.523 | 19.9129 |     10   |    89.2 |
| 0.77     |    50.02 | 356.502 | 18.8813 |     13.6 |    87.6 |
| 0.846667 |    51.5  | 388.975 | 19.7224 |     13.2 |    87.6 |
| 0.923333 |    50    | 353.93  | 18.813  |     16.8 |    86.8 |
| 1        |    48.8  | 382.07  | 19.5466 |     13.6 |    86.4 |
+----------+----------+---------+---------+----------+---------+
```

3.5.2 ResNet34

```
+----------+----------+---------+---------+----------+---------+
|   Crop % |   Mean % |     Var |     Std |  CVaR 5% |  Top 5% |
|----------+----------+---------+---------+----------+---------|
| 0.08     |    49.9  | 373.538 | 19.3271 |     12.8 |    86.8 |
| 0.156667 |    50.28 | 342.755 | 18.5137 |     14.4 |    82.8 |
| 0.233333 |    51.96 | 355.858 | 18.8642 |     14.8 |    83.6 |
| 0.31     |    53.5  | 386.322 | 19.6551 |     14.4 |    88.4 |
| 0.386667 |    50.32 | 422.008 | 20.5428 |     13.2 |    87.2 |
| 0.463333 |    51.38 | 376.136 | 19.3942 |     14   |    86.4 |
| 0.54     |    52.58 | 343.702 | 18.5392 |     14.4 |    83.6 |
| 0.616667 |    52.96 | 360.24  | 18.98   |     14.8 |    87.2 |
| 0.693333 |    52.48 | 358.683 | 18.9389 |     17.6 |    86.4 |
| 0.77     |    53.42 | 376.426 | 19.4017 |     16   |    89.6 |
| 0.846667 |    51.82 | 347.706 | 18.6469 |     14.8 |    87.6 |
| 0.923333 |    51.56 | 335.243 | 18.3096 |     16.4 |    87.2 |
| 1        |    51.38 | 319.372 | 17.871  |     15.2 |    82   |
+----------+----------+---------+---------+----------+---------+
```

3.5.3 Combined net without Pre-Train

```
+----------+----------+---------+---------+-----------+----------+
|   Crop % |   Mean % |     Var |     Std |   CVaR 5% |   Top 5% |
|----------+----------+---------+---------+-----------+----------|
| 0.08     |     44.4 | 344.925 | 18.5721 |       9.6 |     81.2 |
| 0.156667 |    39.42 | 333.411 | 18.2595 |       8.4 |     75.2 |
| 0.233333 |    34.42 | 329.149 | 18.1425 |       2.8 |     76.4 |
| 0.31     |    37.82 | 408.41  | 20.2091 |       3.6 |     82.4 |
| 0.386667 |    42.22 | 337.64  | 18.375  |       9.2 |     79.6 |
| 0.463333 |    34.46 | 359.024 | 18.9479 |       4.4 |     75.6 |
| 0.54     |    49.26 | 409.661 | 20.2401 |      12.4 |     87.2 |
| 0.616667 |    43.14 | 445.568 | 21.1085 |       6.8 |     87.2 |
| 0.693333 |    35.56 | 431.564 | 20.7741 |       4   |     78.4 |
| 0.77     |    34.54 | 354.601 | 18.8309 |       1.6 |     73.2 |
| 0.846667 |    48.8  | 411.176 | 20.2775 |      10.8 |     86   |
| 0.923333 |    36.02 | 459.175 | 21.4284 |       2   |     76.4 |
| 1        |    38.08 | 336.396 | 18.3411 |       5.6 |     76.4 |
+----------+----------+---------+---------+-----------+----------+
```

3.5.4 Combined net with Pre-Train

```
+----------+----------+---------+---------+-----------+----------+
|   Crop % |   Mean % |     Var |     Std |   CVaR 5% |   Top 5% |
|----------+----------+---------+---------+-----------+----------|
| 0.08     |    49.4  | 345.045 | 18.5754 |      15.2 |     85.6 |
| 0.156667 |    50.24 | 360.465 | 18.9859 |      16.8 |     86.8 |
| 0.233333 |    50.76 | 341.932 | 18.4914 |      17.2 |     86.8 |
| 0.31     |    50.76 | 358.656 | 18.9382 |      16   |     87.2 |
| 0.386667 |    51.52 | 339.547 | 18.4268 |      16.8 |     86.4 |
| 0.463333 |    51.22 | 355.489 | 18.8544 |      16.8 |     86.4 |
| 0.54     |    51.56 | 344.248 | 18.5539 |      17.2 |     85.6 |
| 0.616667 |    52.58 | 348.205 | 18.6602 |      17.6 |     88.8 |
| 0.693333 |    52.46 | 345.918 | 18.5989 |      16.4 |     88.4 |
| 0.77     |    52.8  | 334.472 | 18.2886 |      17.6 |     86.8 |
| 0.846667 |    52.98 | 348.542 | 18.6693 |      17.6 |     88   |
| 0.923333 |    53.2  | 350.231 | 18.7145 |      16.4 |     88.4 |
| 1        |    53.24 | 319.098 | 17.8633 |      19.2 |     87.2 |
+----------+----------+---------+---------+-----------+----------+
```

Based on the tables presented, we observe a clear improvement in performance for the CVaR 5% metric when using our implementation of the Combined net compared to ResNet18 and ResNet34, at the cost of a slight sacrifice in the performance of the Top 5% classes. Furthermore, it is evident that the variance is lower in the combined pre-trained net compared to the other networks. This indicates that our implementation treats the classes more fairly, as all of them are closer to the mean accuracy, which is also higher compared to the other implementations.

4 Conclusion

Our solution in general, especially when pretrained, has reduced the gap in class accuracy across different crops. It slightly disadvantaged the classes that performed better with the basic ResNet models, but, more importantly, it increased the average accuracy of classes that generally performed poorly. This introduced a general fairness in training the dataset by reducing the disadvantage that some classes had compared to others. Due to technological and time constraints, we had to train all networks with a maximum of 20 epochs per crop. However, networks with tens of millions of parameters would yield better and more precise results with a higher number of epochs. Therefore, it would be possible to further increase the accuracy of disadvantaged classes and make all the graphs more linear by using a larger number of epochs.

Building upon these findings, there are several other experiments that can be conducted to further validate and consolidate the obtained results. Firstly, it is crucial to repeat the experiments already performed in order to ensure consistent and linearized results, eliminating any outliers or inconsistencies in our plots.
An intriguing experiment would involve testing this implementation on the full ImageNet dataset, using the ResNet50 architecture. This would enable us to verify whether the biases reported in the original paper [1] are reduced, if not completely eliminated, by our approach.
Furthermore, exploring the effectiveness of this approach with different forms of data augmentation would be valuable. Conducting experiments with various augmentation techniques can help assess the robustness and generalization capabilities of the combined net.
Overall, by conducting these additional experiments, we can strengthen the validity of our findings and gain further insights into the performance and versatility of the proposed approach.

References

[1] Randall Balestriero, Leon Bottou, and Yann LeCun. *The Effects of Regularization and Data Augmentation are Class Dependent*. 2022. arXiv: `2204.03632 [cs.LG]`.

[2] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. "Siamese neural networks for one-shot image recognition". In: *ICML deep learning workshop*. Vol. 2. 1. Lille. 2015.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[4] Ya Le and Xuan Yang. "Tiny imagenet visual recognition challenge". In: *CS 231N* 7.7 (2015), p. 3.

[5] Christian Biffi and Alberto Cavallotti. *Mitigating Data Augmentation bias*. URL: `https://github.com/creix/DataAugmentation_Bias`.