The affect of color spaces on learning

Alon Netser[*]
School of Computer Science and Engineering
The Hebrew University of Jerusalem
alon.netser@mail.huji.ac.il

Yedid Hoshen
School of Computer Science and Engineering
The Hebrew University of Jerusalem
yedid.hoshen@mail.huji.ac.il

Abstract

Image classification is typically done in RGB space. Some papers found that different color spaces achieve better performance, in some specific tasks. In this work we tried to learn the best color space for CIFAR-10 classification task, by directly learning a per-color embedding. This will settle the question of the best color space once and for all. During the work we try to understand how color spaces affect learning, by examining the classification task on CIFAR-10 dataset.

1 Introduction

Image classification is one of the most fundamental applications in the field of computer vision. Most of the datasets used for image classification tend to consist of color images. These color images are represented in RGB format. To a computer, these images are just numbers and do not contain any inherent meaning. Most recent models developed for classification do not perform a color space transformation to the image and instead use the RGB image directly for classification.

Some papers claim that changing the color-space affect the performance of classification. We explore the importance of color spaces and examine whether changing them can affect the classification accuracy.

Our basic idea is trying to learn the best color-space for a specific classification task. We chose the simple CIFAR-10 [2] classification task as an example. The idea behind this choice is that the images in CIFAR-10 are low dimensional, and therefore color is a feature of high importance. For example, differentiating between a plane and a frog can done by noting that plane images contain mainly blue (the sky) and frog images contain mainly green.

Learning the best color-space is done as follows. The network gets the RGB images as input, and before doing any processing (such as convolutional layers) it can change the color-space. This will allow the network to choose the best color-space for the specific task given.

2 Previous work

Gowda et al. (2018) [1] claimed that color-spaces can significantly affect classification accuracy. They took the DenseNet model [4] as an example, and trained it multiple times, each with a different color space. Each time the raw RGB images were converted to some other color-space, and the rest of the training process continued as usual. Their results were that the overall accuracy remained quite similar, but certain classes achieved different accuracy in different color-spaces.
Furthermore, they showed that combining all color-spaces together achieved higher accuracy overall. The method they used to combine several color-spaces was stacking several copies of the network and giving each one a different color-space. In the end, the output of all seven networks was combined using an affine layer.

[*]Under the supervision of Yedid Hoshen

Preprint. Under review.

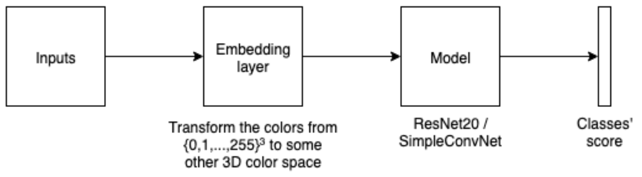

Inputs
Embedding layer
Model
Transform the colors from {0,1,...,255}³ to some other 3D color space
ResNet20 / SimpleConvNet
Classes' score

Figure 1: Model architecture

Our motivation was the following - assuming each color-space performs differently, how can one know which one is the best color-space? Furthermore, maybe a neural network can construct a color space that will be ideal for the given task, while being different from the other known color spaces. The objective was that instead of choosing the best color-space suited for a given task, *learn* the best color-space for the given task end-to-end with the training phase. This will settle the question of the best color space one and for all.

3 Method

In order to learn the best color space, we want to give the network the opportunity to change the color space during training before doing any further computation (e.g. convolutional/affine layers). The network receives the input images in RGB, and then it can change the color space to decrease the loss and increase performance. There are several aspects to examine, and we discuss them briefly as follows.

We choose the dimension of the output color space to be 3. One can decide to enable the network to convert the original RGB to another color space with a different dimension. We chose it to be 3 and thus maintain the dimension, in order to match with most of the known color spaces which are 3 dimensional. Working with color spaces with different dimensions might be interesting, and left as future work.

The main choice to be made is how to learn the color space. One simple option is to use a 3×3 matrix and a 3D bias vector to learn an affine transformation from the 3D input RGB images to a new 3D color space. Many of the existing color spaces are affine transformations of the RGB color space, so this idea seems reasonable. Another option is to use a neural network that gets a 3D vector as input, and output a 3D vector which is the color in the new color space.

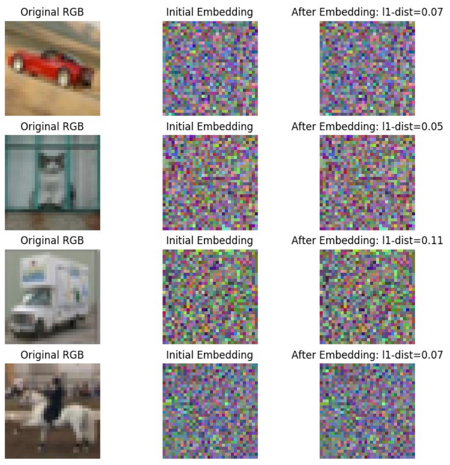
We choose another option, which we think is the most general way of expressing a color space. We use an embedding layer of size $N^3 \times 3$, so each (r, g, b) vector in $\{0, 1, \ldots, 255\}^3$ is quantized into the $\{0, 1, \ldots, N\}^3$ and then transformed to the 3D vector that is assigned to the corresponding vector $(i, j, k) \in \{0, 1, \ldots, N\}^3$. This way enables us to express any color space, not only a one that can be expressed using a multiplication by a matrix or a neural-network.

Figure 1 shows the general idea of the architecture we use in order to learn the new color space.

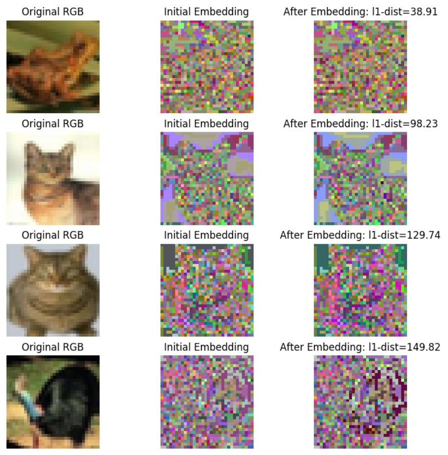
4 Experiments

The specific neural networks we work with are not of great importance, as learning the color space is the topic we investigate. We choose to work with *ResNet20* and *SimpleConvNet*. The ResNet20 is the model taken from [3]. This is the variant the authors created to examine the results on the CIFAR-10 dataset. The SimpleConvNet is a simple convolutional neural-network which is basically 2 conv-pool-relu layers followed by 2 affine-relu, and finally an affine layer to predict the classes' scores.

The results of the two models were quite similar. The ResNet reached slightly higher test accuracy, but the embedding learned and the conclusions of the different experiments were quite the same. Therefore, the main focus was on the SimpleConvNet model. The other intuition why we paid extra focus on the SimpleConvNet model is because the it has less expressiveness, and therefore the color space might affect it more - It's more likely that the model will change the color space in order to increase the performance.


Original RGB
Initial Embedding
After Embedding: l1-dist=0.07
Original RGB
Initial Embedding
After Embedding: l1-dist=0.05
Original RGB
Initial Embedding
After Embedding: l1-dist=0.11
Original RGB
Initial Embedding
After Embedding: l1-dist=0.07

(a) Embedding of size 256^3 initialized as random $\mathcal{N}(0,1)$


Original RGB
Initial Embedding
After Embedding: l1-dist=38.91
Original RGB
Initial Embedding
After Embedding: l1-dist=98.23
Original RGB
Initial Embedding
After Embedding: l1-dist=129.74
Original RGB
Initial Embedding
After Embedding: l1-dist=149.82

(b) Embedding of size 32^3 initialized as random $\mathcal{N}(0,1)$

Figure 2: Examining the embeddings: original RGB image, initial embedding and the learned embedding

4.1 Learning an embedding layer

We train each model with and without an embedding layer in the beginning of the network. Each experiment was attempted with many combinations of hyper-parameters, such as learning-rate (between 0.0001 and 0.1), batch-size (between 8 and 128) and momentum (0.8 - 0.95). In order to increase regularization we also tried weight-decay, between 0 and 0.1. The reported results are the best among the different training processes.

- The baseline ResNet20 over-fitted and reached about 100% train-accuracy and 78% test-accuracy.
  The version with the embedding also reached about 100% train-accuracy, but only 19% test-accuracy.

- The baseline SimpleConvNet also over-fitted and reached about 100% train-accuracy, and 65% test-accuracy.
  The version with the embedding also reached about 100% train-accuracy, and 16% test-accuracy.

Examining the embedding learned showed that the original RGB images was turned into something that looks like noise, and still the network was able to learn the training-set properly and reach 100% accuracy. The models were able to memorize the training-set, even when the images looked like a completely noise image. However, the performance on the test-set was extremely poor (15%-20%, which is slightly better than random guessing).

The fact that the models over-fitted so badly suggests that an embedding of size 256^3 is too large and enables the model to memorize the training data.

In order to deal with this issue, we quantized the colors from $\{0, 1, \ldots, 255\}^3$ to $\{0, 1, \ldots, 32\}^3$ and learned an embedding of size 32^3 which seems more reasonable. The results improved and the test accuracy reached about 40% (in the SimpleConvNet model). The embedding also changed much more, and each image actually changed its colors (although the change is not so significant to human eyes).

In Figure 2 we show the initial and learned embeddings, and the ℓ_1 distance between the source image and the image after the embedding is written above the image. As expected, the smaller embedding changed much more than the bigger one. The numerical difference between the original embedding and the learned one is summarised in Table 1.

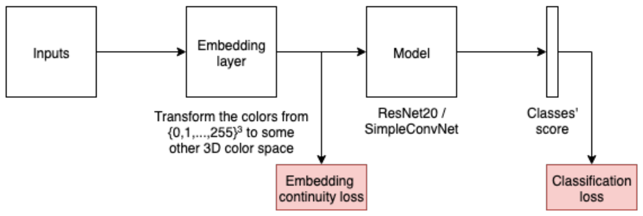

Inputs
Embedding layer
Model
Transform the colors from {0,1,...,255}³ to some other 3D color space
ResNet20 / SimpleConvNet
Classes' score
Embedding continuity loss
Classification loss

Figure 3: Model's architecture with the continuity loss, in addition to the classification loss

4.2 Enforce continuity

We suspect that the reason why the previous experiments did not work was mainly because the initial color space was completely random and non-continuous. This is because the embedding layer was initialized with $\mathcal{N}(0,1)$, so each 3D vector that is associated with some (r,g,b) vector is chosen from a normal distribution independently. Therefore, similar colors are initialized with different random vectors.

Continuity seems like a crucial property for a color space, since similar colors should be transformed to similar colors in the embedding space. Otherwise, the locality will not be preserved. Locality is one of the main reasons convolutional neural networks work well - the convolution layers use the fact that pixels that are close to each other are related. When very similar colors are transformed into completely different ones, recognizing the objects in the images seems impossible.

Our experiment's intuition is to punish the model if the embedding it learned is not continuous. The way we did it was to build a loss on the embedding layer to force it to be continuous. In each iteration we sample N random colors, and punish according to their euclidean distance from their neighbors (i.e. ±1 or 0 change in each coordinate). We saw it as a regularization to prevent the model from memorizing the training-set in its embedding layer, so it will not overfit so badly.

We tried many different values for N, from 1 to 1024. This loss did not work well, and its value did not decrease much. We also didn't see much impact on the results, with and without the continuity-loss.

Another idea to improve the impact of the continuity-loss was to sample N random colors from the mini-batch images' colors, and not just N random colors. This seems more reasonable, since these are the images that will be processed in this iteration and the embedding vectors of their colors are about to change. This version of the loss caused it to decrease better, but still very slowly. The impact on the classification results remained not significant.

Figure 3 shows architecture with the continuity loss, in addition to the original classification loss. The numerical difference between the original embedding and the learned one is summarised in Table 1.

4.3 Initialize the embedding with RGB instead of random

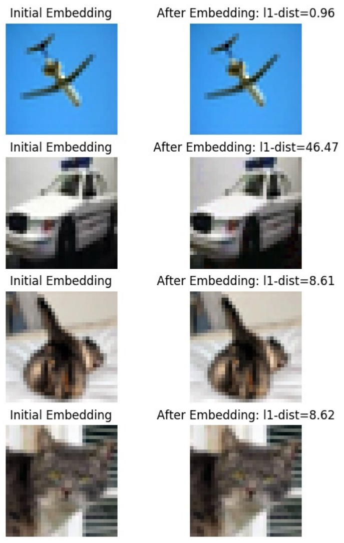
Due to the previous results, we thought the learning might do better by initializing the embedding differently. As a start, we tried to initialize the embedding layer as the identity mapping (i.e. RGB), instead of random $\mathcal{N}(0,1)$. The training now reached similar results as the baseline, so the generalization was finally achieved.

However, examining the embedding that was learned showed that it did not change much. The version with the smaller embedding changed more, but still the changes did not seem interesting. Figure 2 shows the embeddings, and these look quite similar before and after the learning process.

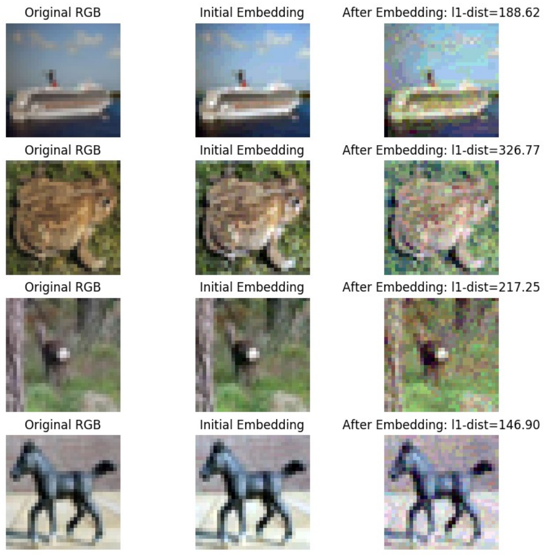
In Figure 4 we show the learned embedding, and the ℓ_1 distance between the source image and the image after the embedding is written above the image. The numerical difference between the original embedding and the learned one is summarised in Table 1.

4.4 Initializing the embedding with a random 3D polynomial

This raised an interesting question - what kind of color spaces do CNN accept? i.e. what kind of initialization functions for an embedding will work? Our guess was that initializing the embedding

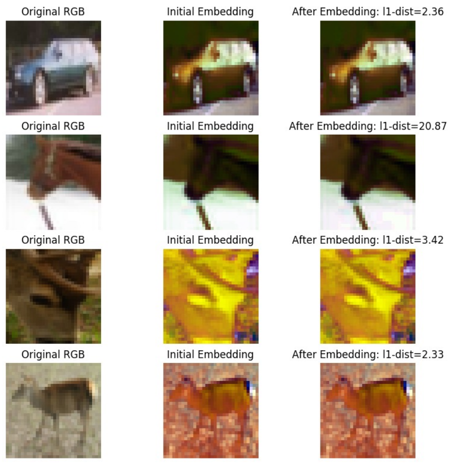

Initial Embedding
After Embedding: l1-dist=0.96
Initial Embedding
After Embedding: l1-dist=46.47
Initial Embedding
After Embedding: l1-dist=8.61
Initial Embedding
After Embedding: l1-dist=8.62

(a) Embedding of size 256^3 initialized as RGB

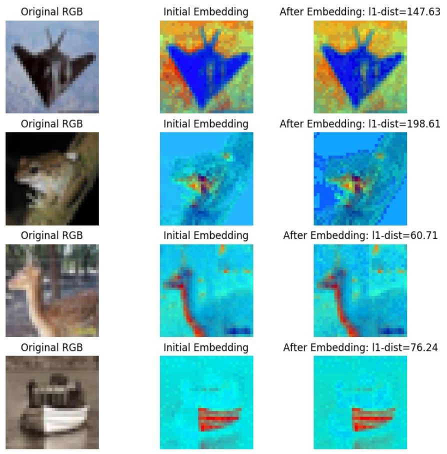

Original RGB
Initial Embedding
After Embedding: l1-dist=188.62
Original RGB
Initial Embedding
After Embedding: l1-dist=326.77
Original RGB
Initial Embedding
After Embedding: l1-dist=217.25
Original RGB
Initial Embedding
After Embedding: l1-dist=146.90

(b) Embedding of size 32^3 initialized as RGB

Figure 4: Examining the embeddings: original RGB image and after the learned embedding. In the right there is also the quantized image to $\{0, 1, \ldots, 31\}^3$


Original RGB
Initial Embedding
After Embedding: l1-dist=2.36
Original RGB
Initial Embedding
After Embedding: l1-dist=20.87
Original RGB
Initial Embedding
After Embedding: l1-dist=3.42
Original RGB
Initial Embedding
After Embedding: l1-dist=2.33

(a) Embedding of size 256^3 initialized as RGB


Original RGB
Initial Embedding
After Embedding: l1-dist=147.63
Original RGB
Initial Embedding
After Embedding: l1-dist=198.61
Original RGB
Initial Embedding
After Embedding: l1-dist=60.71
Original RGB
Initial Embedding
After Embedding: l1-dist=76.24

(b) Embedding of size 32^3 initialized as RGB

Figure 5: Examining the embeddings: original RGB image, after a random 3D polynomial, and finally after the learned embedding.

with any continuous function that is approximately one-to-one will reach the same results as RGB (see the discussion section for further intuition).

A simple (though interesting) option is to initialize the embedding layer with a random multivariate polynomial. We initialized the embedding with random 3D polynomial of degree 3. Each one of the coefficients was drawn from a uniform distribution between -1 and 1. The training continued as previously, and the model can change the embedding during training to decrease the loss and increase performance.

The interesting results were that the models with the RGB initialization and the models with the random polynomial initialization **reached the same classification results**. This supports our intuition that the specific color space does not matter, as long as it is continuous and one-to-one.

(a) Original image (b) After shuffling the colors

Figure 6: Shuffle the colors in the dataset

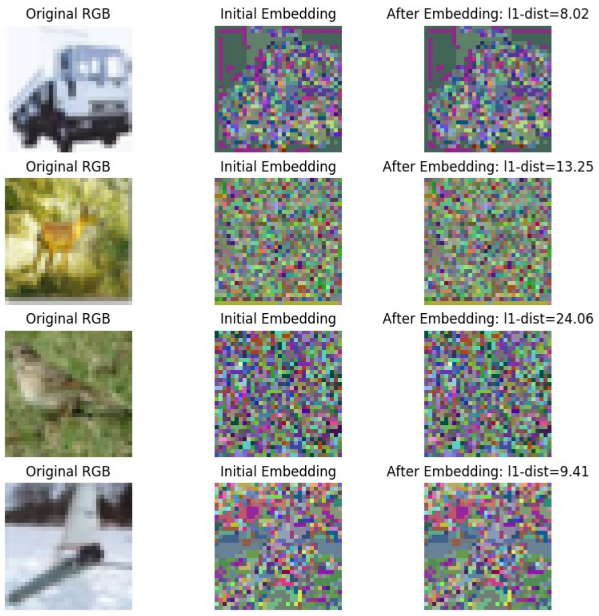

Original RGB
Initial Embedding
After Embedding: l1-dist=8.02
Original RGB
Initial Embedding
After Embedding: l1-dist=13.25
Original RGB
Initial Embedding
After Embedding: l1-dist=24.06
Original RGB
Initial Embedding
After Embedding: l1-dist=9.41

Figure 7: Embedding learned while training on the shuffled-colors images

In Figure 5 we show the initial and learned embeddings, and the ℓ_1 distance between the source image and the image after the embedding is written above the image. The numerical difference between the original embedding and the learned one is summarised in Table 1.

4.5 A setting where learning a color space works better

Our previous results showed that learning the color space embedding did not improve the results, and with some initializations the performance was about the same as the baseline network (without the embedding). This raised the question of whether there exists a setting where learning a color space and enabling the model to change it can help the learning process.

Intuitively, we shuffled the RGB colors of the images in our dataset. Formally, first we quantized the RGB colors from range $\{0, 1, \ldots, 255\}^3$ to the range $\{0, 1, \ldots, 32\}^3$. Second, we chose some permutation $\sigma : \{0, 1, \ldots, 32\}^3 \longrightarrow \{0, 1, \ldots, 32\}^3$ and saved it for later use. Then, for each color in each image, we quantized the color to the range $\{0, 1, \ldots, 32\}^3$, and then applied the permutation σ we sampled earlier. Figure 6 shows one sample image from the dataset before and after the color shuffling.

We then trained the model with and without an embedding layer of size 32^3, initialized as random $\mathcal{N}(0, 1)$. The images that the model received as input during training were the images after shuffling the colors, as described before.

Initialization type	Embedding size	ℓ_0	ℓ_1	ℓ_2	ℓ_∞
$\mathcal{N}(0,1)$	256^3	$13,196,629$	452.28	0.5	0.17
$\mathcal{N}(0,1)$ with continuity-loss	256^3	$16,777,216$	68510.02	68.81	3.26
RGB	256^3	$13,589,404$	4176.12	2.4	0.15
Random 3D polynomial	256^3	$13,471,112$	2503.80	1.58	0.1
$\mathcal{N}(0,1)$	32^3	$80,180$	1343.75	15.15	1.88
$\mathcal{N}(0,1)$ with continuity-loss	32^3	$80,197$	1613.99	24	2.99
RGB	32^3	$80,190$	721.11	5.44	0.25
Random 3D polynomial	32^3	$80,007$	249.68	2.91	0.34

Table 1: Comparing the differences between the learned embedding and the initial one.
The ℓ_0 distance shows that most of the colors did change their values. The ℓ_1 and ℓ_2 distances show that the total change is not significant. The ℓ_∞ distance shows that the change was not significant, in almost every setting. Clearly, the smaller embedding changed much more than the bigger one. The continuity-loss affect was significant, and the embedding changed much more (especially in the bigger embedding).

The results showed that the model with the embedding layer achieved higher accuracy than the baseline model. The baseline reached about 29% test accuracy, and the model with the embedding layer reached about 39%.

In Figure 7 we show the initial and learned embeddings, and the ℓ_1 distance between the source image and the image after the embedding is written above the image.

This experiment supports the fact that there might be settings where the learning to change the color space might increase the performance, despite the previous experiments that showed a learning task where it did not help much.

5 Discussion

In this work we tried to learn the best color space the task of classifying the CIFAR-10 dataset, and the color space was trained end-to-end with the classification task.

The initialization of the embedding layer was very significant. A "good" initialization reached about the same results as the baseline, and a "bad" initialization reached extremely poor results.

We tried to understand what is a good initialization of the embedding. In other words, we tried to understand what color space do convolutional neural networks can get during training and reach approximately the same accuracy. Precisely, what kind of transformations can one do to the color space and the neural network will still learn.

Our conjecture is that any color space that is continuous and approximately one-to-one will suffice. The continuity is necessary because similar colors should be transformed into similar colors in the embedding space. The one-to-one is necessary because if many RGB colors are transformed into one color in the embedding space, then obviously some information is lost. In the extreme case where all RGB colors are transformed into one color, the learning will surely fail, so this motivates our one-to-one criterion for good color spaces.

The continuity seemed important enough so we will enforce the model to learn continuous embeddings, using a designated loss function. We thought of it as a good regularization, as well as a crucial property of color spaces. The experiments showed that it worked slightly better, but the change was not significant.

Our experiments showed that initializing the embedding with random $\mathcal{N}(0,1)$ worked pretty bad, and the model memorized the training data while reaching extremely poor test accuracy. Regularizing with the continuity loss and decreasing the embedding dimension improved the results, but they were still quite bad. Initializing the embedding with the identity mapping (i.e. RGB) caused the learning to work pretty well, about the same as the baseline. The surprising result was that initializing it with a completely random multivariate polynomial worked about the same. This strengthened our conjecture that continuous and one-to-one initialization will work well.

Our experiments made us skeptical regarding the conclusions of Gowda et al. (2018) [1], stating that the color space is significant. We saw that the color spaces in CIFAR-10 classification task was not significant for the successful training, as long as it's reasonable (as discussed previously).

However, we do not rule out the possibility that the color space will matter in some tasks. We wanted to witness an example where the color space does matter, and learning to change the color space helps. In order to do so, we successfully constructed a setting where a model capable of changing its color space worked better. This setting was somewhat artificial, but it still showed that such scenarios exist.

6 Conclusion

Learning the best color space is not a trivial task. In our work we tried to do it in many different ways, but it did not show promising results, and it seemed that the model preferred to stay with its current color space keep it quite unchanged. As a future work, it will be interesting to find a realistic setting where learning the best color space improves the performance. It will be interesting to see what kind of color spaces the model learns, and whether it will be related to some the known color spaces.

References

[1] Gowda, S. N., & Yuan, C. (2018, December). ColorNet: Investigating the importance of color spaces for image classification. In Asian Conference on Computer Vision (pp. 581-596). Springer, Cham.

[2] Krizhevsky, A., Nair, V., & Hinton, G. (2014).
The CIFAR-10 dataset. online: http://www.cs.toronto.edu/kriz/cifar.html, 55.

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[4] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).