A Model-Agnostic Derivative of Cutout for Image Data Augmentation

by

Nikoo Aghaei

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

Abstract

Data augmentation is a strong tool for enhancing the performance of deep learning models using different techniques to increase both the quantity and diversity of training data.

Cutout was previously proposed, in the context of image classification, as a simple regularization technique that increases Convolutional Neural Networks' robustness and performance by masking part of the input image. Cutout can also be applied along with other data augmentation or regularization techniques, further improving their effectiveness.

We extend the main idea of Cutout, where instead of randomly masking images as Cutout does, we use a model to identify which part of the image affects the model's decision the most. Similar to Cutout, our new approach can be used with other regularization techniques for better performance.

Our experiments using the CIFAR-10 and CIFAR-100 datasets with the same ResNet18 and WideResNet models and parameters used in the original Cutout paper show that we can get slightly higher accuracy but at a higher cost than Cutout. These results suggest that randomness is an effective driving factor in Cutout, making it accurate enough and time-efficient. Nonetheless, our approach can be helpful when accuracy is the main concern and the extra time cost is acceptable for the aimed application.

To my lovely grandmothers,

for highly valuing education even though they were deprived of it.

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Mario A. Nascimento, for all his guidance and dedicated involvement in this research. I have been extremely honoured to work under his supervision and learn from him. I would also like to acknowledge the valuable help of Prof. James R. Wright and his graduate student, Daniel Chui, for providing us with a GPU. I also acknowledge Jason Cannon's effort to initiate the idea of MANGO, explained in this research.

I would also like to thank my boyfriend, Bedir Tapkan, for being the best study buddy, always patient in listening to my ideas and discussing them with me. This journey would have been way harder without his company. I am further thankful to my friend Kiarash Aghakasiri for the helpful conversations about my code and all other friends and instructors who inspired me during all years of my education.

I must also express my profound gratitude to my parents for believing in me when I felt the weakest and always being there for me with their endless love and encouragement to make me confident and ambitious to move forward. I am also grateful for having a perfect sister, Nafiseh, who has always been my best friend and strong supporter. I would have never acquired any accomplishments throughout my life without my family.

Finally, my heartful thanks go to my friends, those who never let the distance between us lessen their care and support for me, and also those whom I found here, and they turned out to be my second family. Special thanks to Sana for always sparing some time for me, and many thanks to all my friends for making my life warm and colourful.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep Learning has proven to have an essential role in the improvement and progress of Computer Vision in recent years. Most of these advances have happened with the help of Convolutional Neural Networks (CNNs), capable of learning complex feature representational spaces [1].

For CNNs, the more complex spaces bring up more resource consumption such as memory, number of parameters, number of operations, etc. They commonly use a large number of learned parameters to obtain the needed representational power, which may lead to less generalization and more chance of overfitting [1]. Overfitting happens when a model fits the training data too well and fails to model the testing data. Regularization is one way to combat this issue.

Data augmentation is a vastly used approach in computer vision tasks among many regularization techniques because of being effective and easy to implement. Data augmentation improves the generalization in tasks such as image classification by increasing both the quantity and diversity of training data. Even simple image augmentations such as cropping or mirroring have been proven effective in enhancing model robustness and increasing accuracy [1].

Cutout [1] was proposed as a simple regularization technique that aims to improve model robustness by putting zero square masks in random sections of the input image. The authors of [1] claim that masking out contiguous regions of input will encourage

the model to consider the full context of the image instead of focusing only on a small set of specific visual features. Implementing Cutout to be applied to the data points while loading them, in parallel with the main training task, will have no extra computational cost. Also, Cutout can be combined with other regularization techniques such as dropout or other data augmentations to improve their results further. Several evaluations using popular image classification datasets such as CIFAR-10 and CIFAR-100 show Cutout's capability to regularize and increase model accuracy. More details on Cutout are provided later in Section 2.4.1.1.

As described above, data augmentation has become a powerful tool to combat the overfitting problem. Among data augmentation techniques, Cutout has shown noticeable efficiency in terms of accuracy, considering its simplicity and ease of implementation. Randomly choosing a point inside the image and setting it as the center of a fixed-sized mask increases robustness and accuracy in Cutout. However, we conjecture there are more informed ways to choose the location of the mask, which may even perform better than Cutout.

In this thesis, we propose a data augmentation technique similar to Cutout, but instead of randomly choosing the mask location, we use the model itself to decide where to put the mask. In this method, we cover the part of the image that affects the model prediction the most. By following this approach, we will force the model to see other parts of the image and not focus only on some certain features, which in the end, should help the model to generalize well.

We hypothesize that by letting the model decide on the location of the mask, we will encourage the model to be more robust to the noise and changes in data, and we may be able to beat Cutout efficiency. Also, we think that the colour of the mask could be a factor in making the masking more effective. In order to evaluate different settings of our method and have a fair comparison between them and the original Cutout, we run experiments on CIFAR-10 and CIFAR-100 datasets using ResNet18 and WideResNet networks with the same parameters as those stated in the Cutout

paper [1]. Our evaluations show that using the model for masking the image yields better accuracy than Cutout, but the cost will increase. We will detail our approach in Chapter 3.

The main contributions of this thesis can be listed as follows:

- We improve the results of previously proposed Cutout by answering this question: Which part of the image, if not seen by the model, might affect the model's prediction result?
- We introduce a new approach for augmenting the data set with better accuracy than Cutout.
- We propose a framework that makes it possible to use the power of a model to improve the prediction accuracy of the same model.
- Finally, we evaluate the new approach and compare it with Cutout through several experiments.

This chapter proposed the problem we want to address and the approach we want to follow to solve it. The remainder of the thesis is organized as follows: Chapter 2 provides an overview of Machine Learning concepts and methods used in our research, followed by a literature review of previous related works to our method. In Chapter 3 we explain our approach in detail. Also, we will discuss over affecting parameters and different possibilities of choosing them. Chapter 4 provides details on each experiment with its parameters and results. Finally, in Chapter 5 we make a conclusion based on our experimental results from Chapter 4 and discuss potential future developments on the idea proposed in this research.

Chapter 2

Background

In this chapter, we explain some Machine Learning concepts used to implement our classification framework and review some research related to our proposed method. We have completed this chapter with the help of many useful resources, among which, [1–4] are the most used ones.

2.1 Image classification

Image classification, as one of the core problems in the computer vision domain, is the task of assigning a label from the fixed set of classes to an input image. A concise form of Image classification pipeline consists of:

- Input: A set of images, each labelled as one of the classes. This dataset is referred to as training data.
- Learning: Training a classifier or learning a model in order to learn members of classes based on the training data.
- Evaluation: Use of a classifier model to predict labels of the test data. The goal is to match as many predicted labels with the true labels of the test data as possible.

In a mathematical notation we can describe the above concepts as follow: The input dataset is shown as $X = \{x_1, x_2, .., x_N\}$ with its corresponding true labels as $Y =$

$\{y_1, y_2, .., y_N\}$ where N is the size of dataset. In this notation y_is for $1 \leq i \leq N$ are not necessarily unique, as many x_is can have a same label. We want to model the conditional distribution $P(y|x, X, Y)$ to be used to make inferences to find the optimal predicted label y for a test data point of x. One way is to have a parameterized function like $f^{\omega}(x) = P(y|x, X, Y)$ to approximate the discriminative distribution where ω is the set of function's parameters. One of the most successful approximation functions in discriminative distributions is deep neural networks. Section 2.2 will expand on the architecture of these models.

Challenges

From the perspective of a classification algorithm, each image is a 3-dimensional array of integer brightness values. For example, each image in the CIFAR-10 dataset is of 32 pixels width, 32 pixels height, and 3 colour channels: red, green, and blue (RGB). In total, each image of this dataset consists of $32 \times 32 \times 3$ or $3,072$ pixels. Each pixel is an integer number ranging between 0 (black) to 255 (white). The Image Classification task is to find a single image label based on these numbers. Image classification can involve many challenges from the perspective of a computer model. Some of these problems are listed below:

- Intra-class variation: When there are many different kinds of class types with nuanced differences in their appearance. For instance, the superclass tree in the CIFAR-100 dataset, which divides into five classes of maple, oak, palm, pine, and willow.

- Background clutter: When an object blends into its background, making it harder to be recognized.

- Viewpoint variation: When an object is posing in many different ways with respect to the camera.

- Illumination conditions: When there are huge effects of illumination on the pixel values.
- Occlusion: When the objects of interest can be partially or completely occluded.

A good classification model has adequate sensitivity to the inter-class (between different classes) variations while being invariant to the intra-class (within the classes) variations. Examples of these challenges are shown in Figure 2.1.

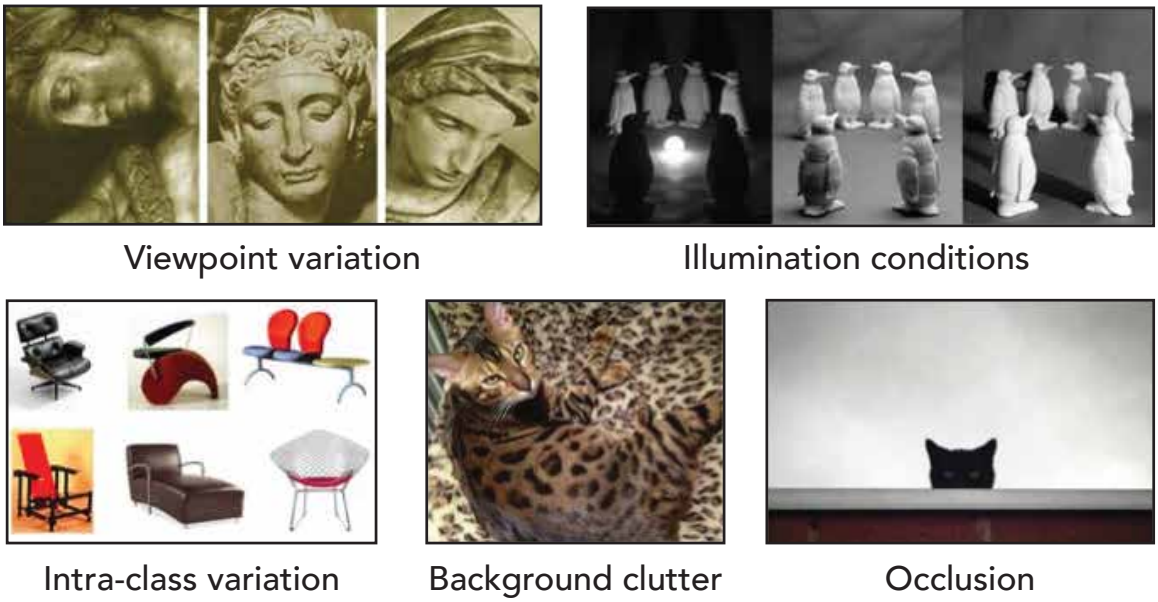


Figure 2.1: Image classification challenges [3].

2.2 Neural Network Architecture

The idea of Neural Network architecture comes from the brain's biological neural system. Stimulus provokes neurons from their dendrites connections, and this signal passes through the axon to other neurons' dendrites. Similar to the neurological system, Neural Networks consist of layers of neurons connected like an acyclic graph known as a feedforward network. Each neuron of a layer gets inputs from neurons of the previous layer, combines them as an affine transformation based on the weights assigned to the connections, passes the result through an activation function, and

finally sends the activation function's output to the next layer. The mathematical representation of a single layer neural network for an input vector of $x \in \mathbb{R}^N$ is:

$$y = f(Wx + b) \tag{2.1}$$

which maps the input to the scalar output $y \in \mathbb{R}^1$. In Equation (2.1) W is a weight matrix, b is a bias vector and function $f()$ is the nonlinear activation function which is explained later in this section. By applying functions composition on the function $f()$ from Equation (2.1) we can formalize a multilayer neural net with two hidden layer as $y = f_3(W_3 f_2(W_2 f_1(W_1 x + b_1) + b_2) + b_3)$ in which f_is are the activation functions, W_is are weight matrices, and b_is are the biases for each layer $i \in \{1, 2, 3\}$. Figure 2.2 shows an example of a one-layer network and Figure 2.3 shows a multilayer neural network with 2 hidden layers of size 4. Layers in these two networks are called fully-connected as all neurons in one layer are pairwise connected to the neurons from the previous layer, but they share no connections within the same layer.

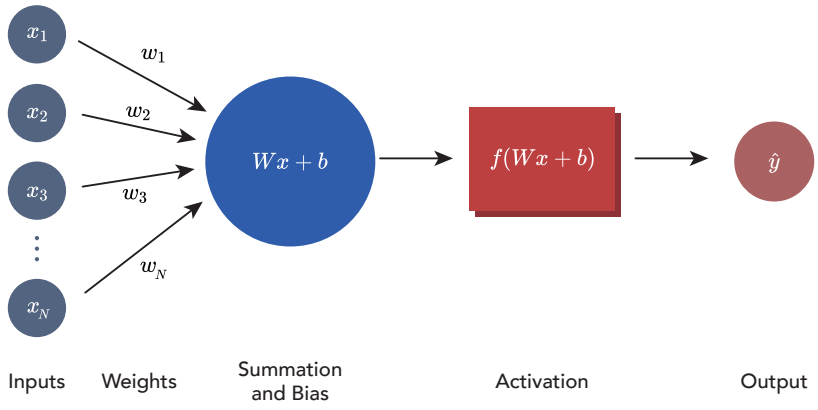


Figure 2.2: (a) A sample of fully connected 1-layer Neural Network with an input of size N.

Activation Function

The non-linearity of activation functions is the key point enabling neural networks to have the approximation power of modelling complex transformations. Activation functions apply a fixed certain of operations on an input number. Many types of

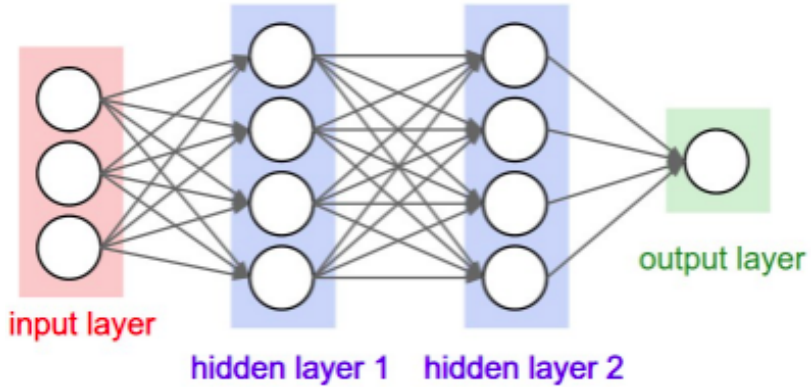


Figure 2.3: A sample of fully connected 3-layer Neural Network. The connections are between neurons of different layers, but not within a layer [3].

these functions are used in practice, such as sigmoid, tanh, softmax, ReLU, and leaky ReLU. We explain those related to our research:

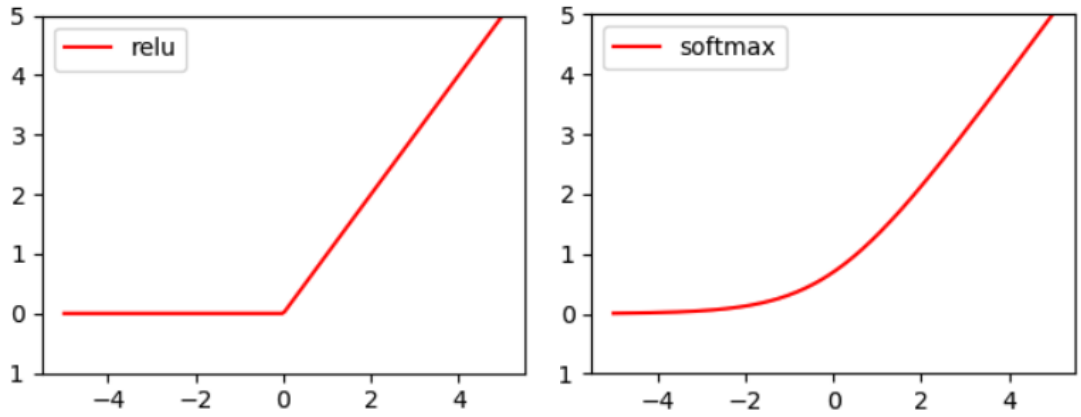
- **ReLU:** The Rectified Linear Unit is a piecewise linear function defined as $f(x) = max(0, x)$. Unlike some activation functions such as tanh and sigmoid, ReLU is easy to implement without using any costly operation.

- **Softmax:** Softmax activation function is mostly used as an activation function in the output layer. Unlike other scalar activation functions, softmax gets a vector of N values (output of a layer) as input and output a vector of normalized probability distributions. Softmax is formalized as:

$$softmax_i(x) = \frac{exp(x_i)}{\sum_n^N exp(x_n)} \tag{2.2}$$

Figure 2.4 plots these activation functions.

In order to model the conditional distribution $P(y|x, X, Y)$ mentioned in the previous section, the output of the classification neural network must be a valid probability function. A valid probability function outputs probabilities in the range of $[0, 1]$ with their sum equal to 1. A common choice to achieve this is to use softmax activation function in the last layer of the neural network.



(a) ReLU (b) Softmax

Figure 2.4: Activation functions.

Loss Function

As explained above, given a dataset, our goal is to build a model with a set of weights leading the model to predict labels consistent with actual labels in the training data. Loss functions determine how well a model performs the classification task in each step and help to improve the results using the loss from the previous training step. One of the commonly used loss functions for a model with probability output values in the range of $[0, 1]$ is cross-entropy loss [5]. Suppose for a given input there are N possible classes. $\hat{y}$ is a vector of the model's predicted probabilities for all classes that sum up to 1, and y is the actual probability vector for all classes. Cross-entropy loss of the data point for each class n where $0 \leqslant n < N$ is calculated as:

$$-y_n log(\hat{y_n}) \tag{2.3}$$

The final cross-entropy loss for the data point will be the sum of the losses for all classes from the above equation:

$$H(y, \hat{y}) = \sum_{n=0}^{N-1} y_n log(\hat{y_n}) \tag{2.4}$$

In multi-class classification, a special case of cross-entropy loss named categorical cross-entropy loss is used where y is a one-hot vector, meaning it is 1 only at the true label index and 0 everywhere else. As a result, the categorical cross-entropy loss for

the data point is:

$$H(y, \hat{y}) = -log(\hat{y}_c) \tag{2.5}$$

Where $\hat{y}_c$ is the predicted probability for the actual label c. If multiple samples in a batch are processed simultaneously, then the total loss of neural network will be the average cross-entropy loss of all samples in the batch. Batch size is the number of training samples utilized in one training iteration and is a hyperparameter usually set based on the memory constraints or some values in powers of 2. Many vectorized operations work faster when having inputs of size powers of 2.

Regularization

Increasing the number of layers and their size in a neural network (i.e. increasing the network capacity) results in growing the representable functions space and model ability to express more diverse functions. Figure 2.5 shows an example of three neural networks with different capacities in a binary classification problem. All these models have one hidden layer of different sizes.

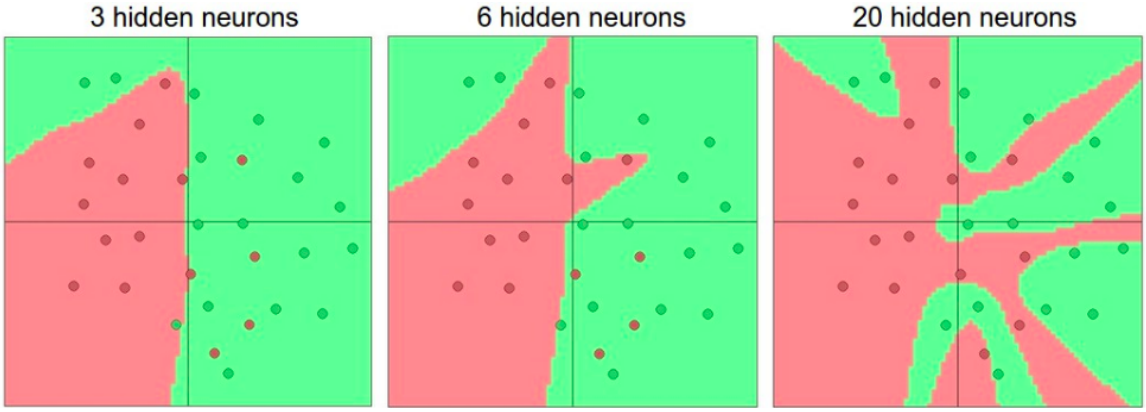


Figure 2.5: Larger Neural Networks can represent more complicated functions with a more chance of overfitting. The data are shown as circles colored by their class, and the decision regions by a trained neural network are shown underneath [3].

As is clear from Figure 2.5, higher capacity gives the network the ability to model more complicated functions. Even though this might seem an absolute advantage, it might result in a higher chance of overfitting. Overfitting happens when a model fits

too closely to the training data in a way that fits the noise as well or fails to predict future unseen data reliably. In the example shown in Figure 2.5, even though the model with 20 hidden units models all data points perfectly, it divides the data space into many separate prediction regions, which can lead to overfitting. Regularization is one solution to control network capacity, prevent learning a more complex or flexible model and help to avoid overfitting. Figure 2.6 shows the changes in the network with one layer of size 20 from Figure 2.5 after applying $L2$ regularization with three different values of strength (λ). In the following, we mention some popular regularization techniques related to our research.

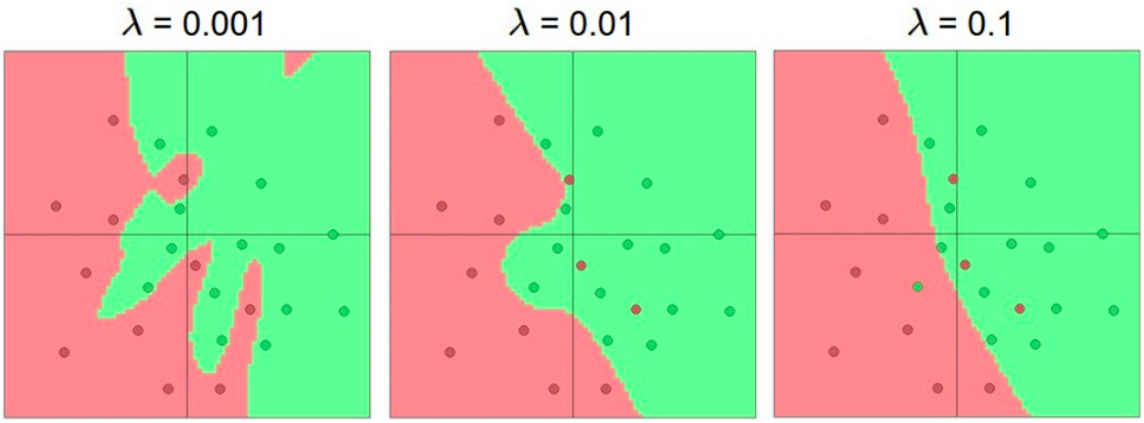


Figure 2.6: The effects of regularization strength: Each neural network above has 20 hidden neurons, but changing the regularization strength (λ) makes its final decision regions smoother with a higher regularization [3].

- **L2 regularization** is the sum of squared magnitudes of all weights added as a penalty to the loss function:

$$Loss(Data|Model) + \lambda \sum_{i=0}^{D-1} w_i^2 \tag{2.6}$$

  In above equation, D is the input data dimension and λ is the regularization strength. L2 regularization penalizes large weights and prefers deffuse weight vectors which means it has the advantage of encouraging the model to use all the inputs a little instead of using some of them a lot.

- **Dropout** [6], as a simple and extremely effective regularization technique, zeros out the activation value of a neuron (hidden or visible) with some probability p during the training process. As a result, it will change the layer's size and connections to other layers in each layer update, making the dropout have the effect of approximately averaging over many smaller sub-networks. Dropout discourages the situations where layers of a network co-adapt to correct previous layers' mistakes, forcing the model to learn more robust features. Figure 2.7 shows the method's idea.

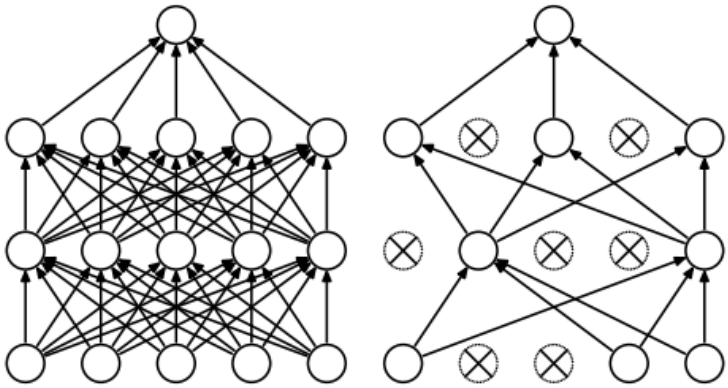


Figure 2.7: Dropout Neural Net Model. (a) A standard neural net with 2 hidden layers. (b) An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. [6]

- **Batch normalization** [7] can be considered as a popular regularization technique used to initialize neural networks properly. At the beginning of training, batch normalization makes all network's activations take a unit gaussian distribution, i.e. a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$. Batch normalization layer is mostly inserted after fully connected layers and before non-linearities. As normalization is a differentiable operation, batch normalization can be seen as a preprocessing at each layer but integrated into the network itself.

- **Transfer learning** is another technique helping neural networks faster conver-

gence. Transfer learning [8, 9] is about using the weights of a model trained on a big dataset, such as ImageNet [10], to initialize weights in a model for a new task. Instead of copying the weights in the whole network, usually, only the weights in convolutional layers are transferred. Deep neural models learn a hierarchical representation of the image data points [11] and also low-level geometric attributes such as corners or lines are pretty common among image datasets. As a result, transfer learning becomes a successful method as feature extractors trained on larger datasets can be transferred to train classifiers on smaller datasets, solving the problem of lacking enough images and speeding up training a new model on a different but relevant task. There is ongoing research on understanding the relationship between transferred domains [12].

- **Pretraining** [13] conceptually close to transfer learning is to initialize the weights using models trained on a big dataset. The difference between transfer learning and pretraining is that in transfer learning, the model architecture is transferred as well as the weights. However, in pretraining, there is the flexibility of having a different architecture design.

Optimization

While the loss function determines the quality of a particular set of weights for the network, the optimizer aims to update the weights in a way that the overall network loss is minimized.

Gradient Descent is a simple optimization method that updates each parameter using its gradient. One common version of Gradient Descent is Stochastic Gradient Descent (SGD). SGD calculates the gradient for one sample data instead of whole data, making it efficient for large datasets that cannot be held in RAM entirely. Also, SGD makes it possible to jump out of local minimums. Learning rate hyperparameter determines at which pace the weights get updated. Monitoring loss function during training can help analyze the neural network training process. The number of epochs

necessary for the model convergence is determined based on the data and the network design. Number of epochs is the total number of iterations needed to process each image from the training dataset.

Backpropagation

One of the main steps in neural network training is computing the loss function gradient with respect to the network parameters. Backpropagation is a way of calculating these gradients by moving backwards through the layers recursively and applying the chain rule from calculus to compute the gradients at each layer. Chain rule helps to express the derivative of a function $f(x) = g(h(x))$ which is the composition of two differentiable functions g and h. Equation (2.7) shows a simple reminder of this rule.

$$\frac{df}{dx} = \frac{df}{dg}.\frac{dg}{dx} \tag{2.7}$$

In summary, the process of updating the weights is a recursion of performing forward propagation through the network on a batch of data to obtain the loss, then backpropagating the obtained loss to calculate the gradients and finally using the gradients to update the weights.

Evaluation

Among several techniques used for evaluating classification models, we use one of the simplest and most popular methods, called prediction accuracy. Prediction accuracy is defined as:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{2.8}$$

2.2.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is an important class of deep neural networks with the assumption of getting images as input. This assumption enables the implementation of some properties into the architecture, making the forward function

more efficient and significantly reducing the number of parameters.

Assume a dataset with images of size $200 \times 200 \times 3$ (width, height, number of channels, respectively). A fully-connected neuron in a regular neural network would have $200 \times 200 \times 3 = 120,000$ weights. Clearly, a fully-connected structure is not scalable to these sorts of large images. This issue is tackled in CNNs by having three dimensions (width, height, depth) neurons. Also, each neuron in a CNN, unlike a fully-connected manner, is only connected to a small region of the previous layer. For example, each image in the CIFAR-10 dataset is an input volume of activations with a height and weight of 32 and a depth of 3. The last layer (output layer) for an image from CIFAR-10 is a volume with dimensions of $1 \times 1 \times 10$ made by reducing the full image into a vector of class scores. Figure 2.8 is a sample visualization of a CNN.

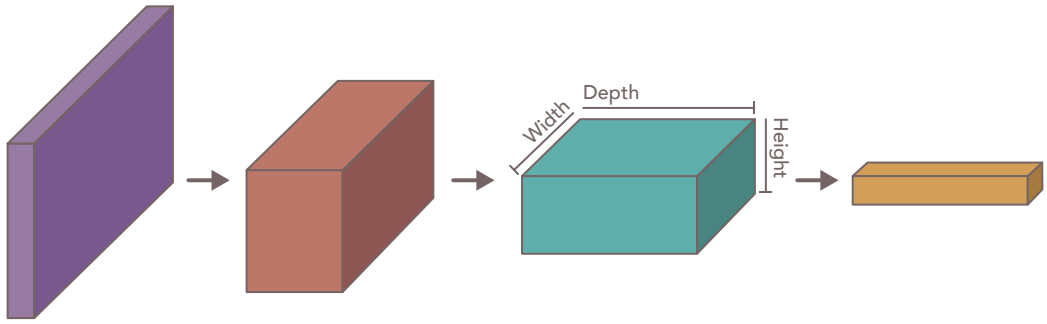


Figure 2.8: In Convolutional Neural Networks, the input of each layer is arranged as a 3 dimensional volume transferred to another 3 dimensional output which will be the input for the next layer [3].

In a classification task, a sequence of layers in a CNN transfers an image from its original pixel values to the final class score. There are three main types of CNN layers. We will explain these layers through an example network, consisting of layers [Input - Convolutional - ReLU - Pooling - Fully-connected] for a classification task on the CIFAR-10 dataset.

- **Input layer** consists of the image raw pixel values. In our example, data points are of the size $32 \times 32 \times 3$ so the input layer is of dimensions $32 \times 32 \times 3$.

- **Convolutional layer** calculates the dot product between the input and the

weights of the neurons connected to them. For example, if we use 8 filters, the output volume of this layer would be of dimensions $32 \times 32 \times 8$.

- **ReLU layer** applies the ReLU activation function elementwise, without changing the volume size.

- **Pooling layer** performs downsampling along width and height (spatial dimensions), changing the volume to a size such as $16 \times 16 \times 8$.

- **Fully-Connected layer** is similar to the fully-connected layer in regular networks which all neurons from the fully-connected layer connect to all units in the previous layer. The fully-connected layer will output the class score with a volume of size $1 \times 1 \times 10$.

There are several known CNN architectures such as AlexNet, VGG-16, DenseNet, etc., among which ResNet and WideResnet are used in our experiments.

- ResNet: Residual Network [14] has special "skip connections" which enables training much deeper networks. As shown in Figure 2.9 the model can skip training of some layers using skip connections. In this way, an identity mapping is applied, and then its result (x) will be added to the outputs of the stacked layers ($\mathcal{F}(x) + x$). These operations do not increase the computational cost or the number of parameters. As a result, the model will backpropagate through an identity function, and the gradient value will be maintained in the first layers of the network. Using skip connections can help the model to tune the number of layers while training.

- WideResNet: Wide Residual Networks were proposed [15] to prevent Diminishing Feature Reuse in Residual networks, which happens due to the possibility of the model skipping all residual blocks and ending up not learning anything or just learning little useful representations. WRN-n-k is a denotation of a

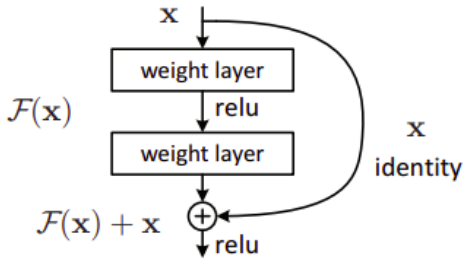


Figure 2.9: Residual learning: a building block [14].

WideResNet with a total number of n convolutional layers and a widening factor of k. $k = 1$ will be the ResNet block. Figure 2.10 [15] shows the comparison of training curves between ResNet and WideResNet networks.

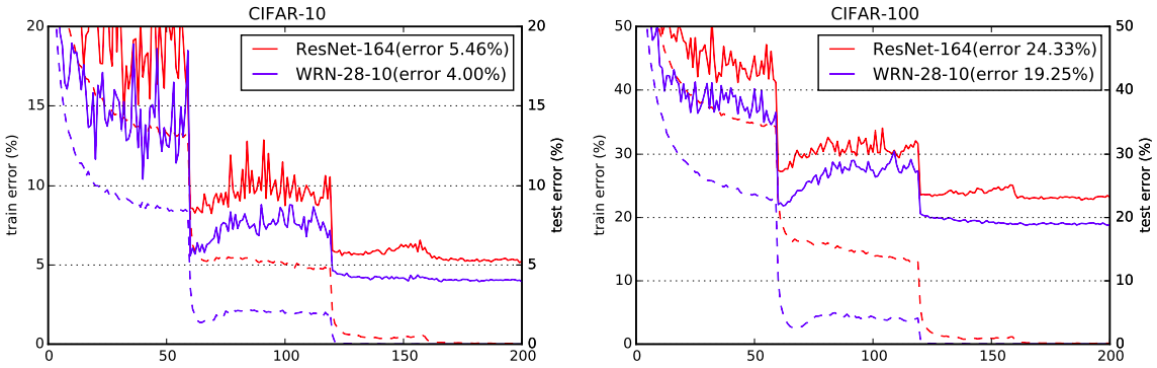


Figure 2.10: Training curves for residual and wide residual networks on CIFAR-10 and CIFAR-100. Solid lines denote test error (y-axis on the right), dashed lines denote training loss (y-axis on the left) [15]. X-axes in both show the epoch number.

In later chapters, we will run experiments using these models and show how they will improve classification accuracy with the help of data augmentation techniques.

2.3 Data Augmentation

Convolutional Neural Networks (CNN) have shown growing learning ability in complex representational spaces. These deep neural networks perform better when they are trained on large and varied datasets. However, more complex models, larger training datasets, and more parameters lead to a higher chance of overfitting, which raises the importance of regularization.

We reviewed some regularization techniques that have been introduced to overcome the mentioned problem. Another technique is Data Augmentation, a data-space solution approaching the problem by enhancing the size and quality of the training datasets by adding slightly different copies of existing data or synthetically created data to the dataset to extract more information from them. Data augmentation is vastly used in vision tasks due to its ease of implementation and effectiveness [11, 15–18].

As mentioned before, issues such as occlusion, viewpoint, lighting, background, and more are extremely common in image classification, image recognition, and other vision tasks. These issues can affect the model's performance. Data Augmentation aims to add these invariances to the dataset by enlarging the dataset in principled ways so that the model can extract more information from augmented data. It should be noted that while we want the augmented data to be reasonably dissimilar to the original data, we still want them to be a correct representation of the original label. Some of these label-preserving transformations are called data warping augmentations. Other augmentations, called oversampling, create new instances with the help of some techniques such as generative adversarial networks (GANs), feature space augmentation, and mixing images. These two groups are not mutually exclusive. We will talk more about these augmentations in Section 2.4.1.

2.4 Related Work

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) started in 2010 to evaluate algorithms for image classification and object detection at a large scale [19]. This competition led to the development of many significant models and algorithms. In 2012, the winner of the ILSVRC was ALexNet [11], the first deep convolutional neural network which had a crucial impact on further researches in the usage of deep CNNs in classification tasks. The 2014's winner was the Inception network, an innovative CNN with kernels of different sizes processing in parallel [20]. A Residual

neural network (ResNet) with 152 layers was the winner of ILSVRC in 2015 [14]. These networks have residual blocks to extend the depth of the network using the core idea of identity shortcut connection that skips one or more layers. In 2016 DenseNet, following the same idea as ResNet residual blocks, used dense blocks to achieve a trainable model with more than 200 layers [21]. MobileNet was proposed in 2017 as a lightweight deep neural network with far fewer parameters than most neural networks [22]. MobileNet showed competitive performance on the ImageNet dataset with the possibility of being used on low-power devices such as mobile phones embedded vision applications [4].

In the previous section we discussed different regularization techniques such as those focusing on models architectures leading to introduce a series of more complex models including AlexNet [11], VGG-16 [23], ResNet [14], Inception-V3 [24], and DenseNet [21], and also some of the functional solutions helping deep learning models in tasks with smaller datasets. Dropout regularization, batch normalization, transfer learning and pretraining were of this type.

Dropout seemed to be less effective in convolutional layers in comparison with fully-connected layers [25]. This inefficiency has roots in two main factors: First, convolutional layers with far fewer parameters than fully-connected ones need less regularization. Second, when some pixels of an image data are dropped, it is likely that their information still is passed on by their neighbouring pixels, which are still active and usually share the same information with them. As a result, dropout does not have the model averaging effect same as in the fully-connected layers but instead makes convolutional layers robust to noisy inputs [1]. Many variations of dropout were introduced in order to increase its effectiveness in convolutional layers. SpatialDropout [25] randomly drops the whole feature map instead of individual pixels to combat the issue of passing the same information by neighbouring pixels. Considering a probability, max-drop [26] discards the maximal activation of feature maps or channels. This approach performed better than dropout in convolutional

layers, but in CNNs with batch normalization, standard dropout outperformed both max-drop and SpatialDropout. PatchShuffle [27] shuffles the pixels in local patches while keeping the structures of the original images. This approach increased CNN generalization, especially with small datasets and can be applied along with dropout for better results.

In the following section, we will expand on the history of data augmentation as the main focus of our research and another regularization technique that addresses the overfitting from the perspective of the dataset.

2.4.1 Data Augmentation

This section will review the history of data augmentation techniques related to our focus, i.e. improving Image Recognition models. Image Recognition models predict an output label for each input image. However, results are extendable between other Computer Vision tasks such as Object Detection or Semantic Segmentation [28, 29].

As mentioned in Section 2.3, data augmentation techniques can be divided into two groups: data warping augmentations and oversampling augmentations. Data warping augmentations are a group of label-preserving transformations such as adversarial training, neural style transfer, geometric and colour transformations, and Random Erasing. However, even data warping augmentations could be a non-label preserving transformation in some applications, such as using Random Erasing in handwritten digit recognition task. In that case, there is not any difference between an '8' and a '6' if the top part of the '8' is randomly cropped out. Therefore, some domain expertise and manual intervention may be needed depending on the dataset and task. Later we will discuss more on Random Erasing.

Data augmentation is vastly used in practice when training convolutional neural networks. In LeNet5 [30] data warping augmentations, including squeezing, scaling, and horizontal shearing, is used to improve one of the first use of CNNs in handwritten digit classification.

The two most popular data warping augmentations in deep CNNs training are RandomCrop and HorizontalFlip. RandomCrop extracts random sub-patch from the input image and, depending on the cropping reduction threshold, might not be a label-preserving transformation. HorizontalFlip randomly flips the input image horizontally. It is not a label-preserving transformation on datasets such as MNIST or The Street View House Numbers (SVHN), which involves text recognition.

Bengio et al. show in [31] that data augmentation is more beneficial for deep architectures than shallow ones. Oversampling augmentations are mainly to help the model not be biased in labelling instances as the majority class. Random Oversampling (ROS) and Intelligent Oversampling are two of these techniques approaching class imbalance [32]. In the 2012 ILSVRC, AlexNet [11] was enhanced by applying random crop, horizontal flip, mirroring, and PCA colour augmentation to randomly adjust colour and intensity. These augmentations increased the training dataset size by a magnitude of 2048.

Goodfellow et al. introduced Generative Adversarial Networks (GANs) in 2014 [33] as an approach to Generative Modeling using deep learning methods. In Generative Modeling, the model is used unsupervised to generate new samples with the same characteristics as the input data by automatically learning the patterns in the original set. GANs train generative models using two sub-models in a supervised manner: the generator model trained to generate new examples and the discriminator model classifying generated examples as real (from the input data) or fake (generated). When the generator model starts generating plausible examples, it will fool the discriminator model about half the time. That is when the adversarial training of both models ends. Many works were published as GAN extensions [34–36].

Neural Style Transfer [37] is a great tool for Data Augmentation by manipulating the CNN representations of the images so that the style of one image is transferred to another one while keeping its original content. Neural Architecture Search (NAS) [38] proposed a novel approach to meta-learning architectures using Reinforcement

Learning to train a recurrent neural network to design architectures with the highest accuracy. In some later works [39, 40] meta-learning concepts from NAS is used in Data Augmentation.

One of the interesting Data Augmentation techniques is Random Erasing (RE) [18] which is inspired by the dropout regularization technique except being applied on the input data instead of being used in the network architecture. RE randomly chooses a rectangular patch of an image and erases its pixels with 0s, 255s, mean pixel values, or random values. According to Table 2.1, using RE with random values outperforms other versions.

Types of erasing value	**Baseline**	**RE-R**	**RE-M**	**RE-0**	**RE-255**
Test error value (%)	5.17 ± 0.18	$\mathbf{4.31 \pm 0.07}$	4.35 ± 0.12	4.62 ± 0.09	4.85 ± 0.13

Table 2.1: Test errors (%) on CIFAR-10 based on ResNet18 (pre-act) [16] with four types of erasing value. In all experiments, the standard data augmentations (random crop + random flip) are applied first. Baseline: Baseline model, RE-R: Random Erasing model with random value, RE-M: Random Erasing model with mean value of ImageNet 2012, RE-0: Random Erasing model with 0, RE-255: Random Erasing model with 255 [18].

RE forces the model to pay attention to all parts of the image and not only a part of it. It also combats the visual challenge of occlusion, when some parts of the object are unclear, by preventing the model from overfitting to a specific visual feature in the image. RE is easy to implement without the need for parameter learning. RE reached some improvements over the previous baselines in image classification, object detection and person reidentification domains. Table 2.2 shows results of using RE for image classification task.

A similar study called Cutout Regularization [1] conducted contemporary with RE. We follow Cutout in our research because of better performance and easier implementation than RE. The following section elaborates more on Cutout regularization as the main technique inspiring our proposed approach.

Model	CIFAR-10	CIFAR-100
ResNet-18-PreAct	5.17 ± 0.18	24.50 ± 0.29
ResNet-18-PreAct + RE	$\mathbf{4.31 \pm 0.07}$	$\mathbf{24.03 \pm 0.19}$
WRN-28-10	3.80 ± 0.07	18.49 ± 0.11
WRN-28-10 + RE	$\mathbf{3.08 \pm 0.05}$	$\mathbf{17.73 \pm 0.15}$

Table 2.2: Test errors (%) with ResNet-18-PreAct [16] and WideResNet [15] on CIFAR-10 and CIFAR-100 [41]. In all experiments, the standard data augmentations (random crop + random flip) are applied first. RE: Random Erasing [18].

2.4.1.1 Cutout

Cutout [1] is one of the popular regularization techniques in Convolutional Neural Networks, typically used as a baseline in image classification experiments. Cutout augments each data point and creates different occluded versions of them by randomly covering a rectangular area in them.

To some extent, Cutout is similar to dropout in adding noise to the dataset, but there are two main differences. First, in Cutout, a contiguous part of the image is dropped out instead of scattered pixels. Second, dropping units in Cutout happens only in the input layer rather than intermediate layers, resulting in removing the masked parts from all subsequent feature maps. In contrast, when a feature is removed from one feature map in dropout and its variations, it may not be removed from the other feature maps. This happens because each feature map is considered individually. These inconsistencies among feature maps result in a noisy representation of the images making the model more robust to noisy inputs. Based on these differences, Cutout is more of a data augmentation technique than a dropout variant as it generates novel images for the network.

Table 2.3 from the Cutout paper shows the test error results of ResNet18 [16] and WRN-28-10 [15] models on CIFAR-10 and CIFAR-100 datasets with and without applying Cutout. Cutout has increased their accuracies in a setting with other regu-

larization techniques such as data augmentation, dropout, and batch normalization.

Model	CIFAR-10	CIFAR-100
ResNet18	4.72 ± 0.21	22.46 ± 0.31
ResNet18 + Cutout	$\mathbf{3.99 \pm 0.13}$	$\mathbf{21.96 \pm 0.24}$
WRN-28-10	3.87 ± 0.08	18.80 ± 0.08
WRN-28-10 + Cutout	$\mathbf{3.08 \pm 0.16}$	$\mathbf{18.41 \pm 0.27}$

Table 2.3: Test error rates (%) on CIFAR datasets. In all experiments, the standard data augmentations (random crop + random flip) are applied first. Results are averaged over five runs [1].

Cutout Motivation

The primary motivation comes from the object occlusion problem, one of the common issues in the Computer Vision domain. In order to understand what features the CNN detects, they [1] visualized feature maps for input images after applying Cutout on them. The visualization results show that masked regions in Cutout will be dropped out from all the feature maps, which means no trace of the occluded parts remains in the final representation. As a result, the model cannot see the occluded parts in each epoch and is forced to learn from all image contexts rather than focusing only on certain visual features. This is how Cutout helps to regularize the model and increase the test accuracy.

Cutout originally intended to remove the prominent visual features of the images to help the model consider the less important ones. In order to accomplish this purpose, similar to maxdrop [26], they extracted and sorted the most activated features from the activation maps (i.e. feature maps) of the images in each epoch. Then they upsampled each feature map to the input resolution and made a binary mask out of them by setting the mean feature map value as a threshold. These binary masks were laid over the original images in the next epoch before passing them through the network. Despite the good results of this earlier version, the authors of the paper [1]

chose to randomly mask a fixed-size part of the images in all their experiments due to the same satisfying results with less complexity and computational cost. Figure 2.11 shows the first and final versions of Cutout.

(a) (b)

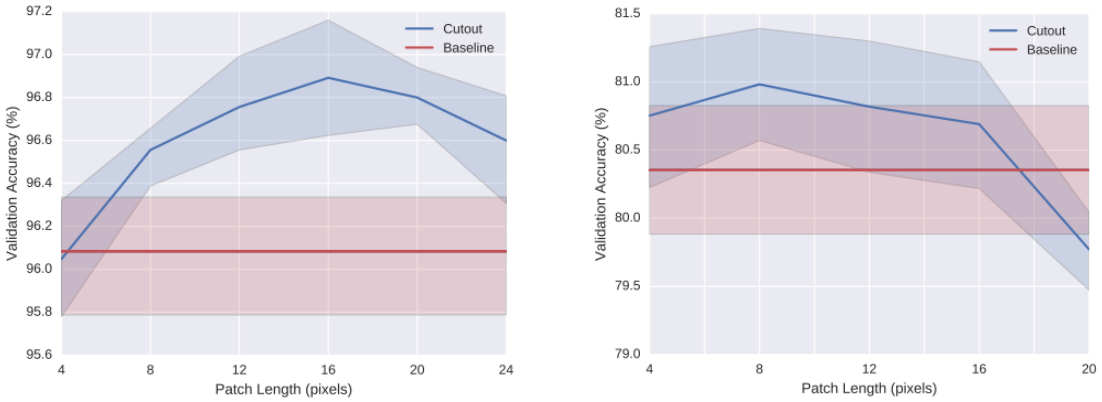
Figure 2.11: (a) An early version of Cutout on CIFAR-10 dataset. This version mostly masks part-level features of the image, such as heads, legs, or wheels. (b) Final version of Cutout applied on CIFAR-10 dataset [1].

Cutout Implementation Details

Cutout is applied on a normalized dataset as a Dataloader's transform, following RandomCrop and HorizontalFlip augmentations on each data point in each epoch. In order to mask an image randomly, Cutout chooses one point inside the image (including the borders) as the center of the fixed-size black mask. This way of masking may result in laying some parts of the mask outside of the image, which is shown to be crucial in achieving high performance because the model gets the chance to see bigger and more diverse areas of the image in some epochs. The original paper claims that an alternative approach is to apply border-constrained masks, i.e. the whole mask should lay inside the image, with a probability of 50% so that model can see the whole original image in some epochs.

The paper performs a grid search to find the optimal side length for the square mask. As it is shown in Figure 2.12 the model accuracy has a parabolic behaviour with respect to the Cutout size. The accuracy increases with the increase in Cutout size up to an optimal point, and then accuracy decreases until it gets below that of

the baseline model. This grid search is done on the validation set of CIFAR datasets resulted in choosing a mask size of 16×16 and 8×8 pixels when training on CIFAR-10 and CIFAR-100 datasets, respectively. This results can be observed in Figures 2.12a and 2.12b which are obtained from CIFAR-10 and CIFAR-100 datasets respectively.



(a) CIFAR-10 (b) CIFAR-100

Figure 2.12: Cutout patch length with respect to validation accuracy with 95% confidence intervals (average of five runs). Tests run on CIFAR-10 and CIFAR-100 datasets using WRN-28-10 and standard data augmentation. Baseline indicates a model trained with no Cutout [1].

According to the above experiment, with the increase of the number of classes, the optimal mask size decreases. The reasoning can be that when we have more classes, i.e. the more fine-grained detection is required, the nuanced details are more useful than the context of the image in identifying the classes.

Algorithm 1 shows the Cutout method with arguments of X as the input tensor image of size $c \times h \times w$ ($channel \times height \times width$), N_h as the number of the masks, and L is the mask size. *Tensor* is a multi-dimensional matrix containing elements of a single data type. M is a tensor of ones that keeps N_h masks created in the for loop (starting at line 2). C_y and C_x are the mask center chosen randomly as a point inside or on the image's border, and x_is and y_is are the mask four corners calculated based on the chosen center (C_x, C_y) and the mask size ($L \times L$). As Cutout uses zero masks, it sets pixels' values in the mask location to zero (line 9). Finally, M is expanded to be the same size as the original image, and by elementwise multiplication, masked

pixels are set to zero in the original image.

Algorithm 1 Cutout

Input: $(X \in \mathbb{R}^{c \times h \times w}, N_h, L)$
$M \leftarrow$ tensor of ones sized $c \times h \times w$
for $i \leftarrow 1$ to N_h **do**
    $C_y \leftarrow \text{random}(0, h)$
    $C_x \leftarrow \text{random}(0, w)$
    $y_1 \leftarrow \max(C_y - \lfloor L/2 \rfloor, 0)$
    $y_2 \leftarrow \min(C_y + \lfloor L/2 \rfloor, h)$
    $x_1 \leftarrow \max(C_x - \lfloor L/2 \rfloor, 0)$
    $x_2 \leftarrow \min(C_x + \lfloor L/2 \rfloor, w)$
    $M[(x, y)] \leftarrow 0$, $x_1 \leq x < x_2$ and $y_1 \leq y < y_2$ and *all channels*
end for
return $X * M$

While loading the data, Cutout is performed on the CPU as a data transform along with other augmentations. "Hiding" the Cutout computational cost by running it on the CPU, parallel with the main training task running on the GPU, increases the performance virtually free [1].

As explained above, DeVries and Taylor claim that removing features with high activations neither improves the performance nor the computational cost compared to randomly masking out the images. That is why they follow the latter approach. As another attempt to substitute the random approach of randomly masking the images, we propose to use the model itself to find and mask the most prominent part of the images. We conjecture that by choosing the size and location of the masks more wisely, we can improve the performance of Cutout regularization. Next chapter will expand on our idea.

Chapter 3

Our Approach

This chapter will first introduce MANGO, the approach from which our core idea comes, and then explain our system design and modifications to the original Cutout.

3.1 MANGO: Model Agnostic explaNation for imaGe classificatiOn

MANGO[1] is a model-agnostic tool developed to answer this important question: In a classification task, which parts of the image are relevant to the model's prediction? Or, in other words, which parts of the image, if not seen by the model, would mostly affect its decision? [42] Considering MANGO as a model-agnostic approach means it can be used for any trained machine learning algorithm because MANGO will use them as a black-box to address the question above.

Given an image, MANGO tries to answer the above questions by building a tree with nodes containing a properly masked version of the original image. Nodes at the same level have a mask of the same size, and the mask size will decrease while moving down the tree. We will elaborate on how MANGO decides on expanding a tree at each level by following the example image in Figure 3.1 representing an Indian Runner duck.

A pre-trained ResNet50 model on ImageNet is used to see the changes in the model

[1]The initial framework was developed by Jason Cannon and Mario Nascimento at Huawei Canada (Distributed Data Storage and Mgmt Lab)

Figure 3.1: An Indian Runner duck[1].

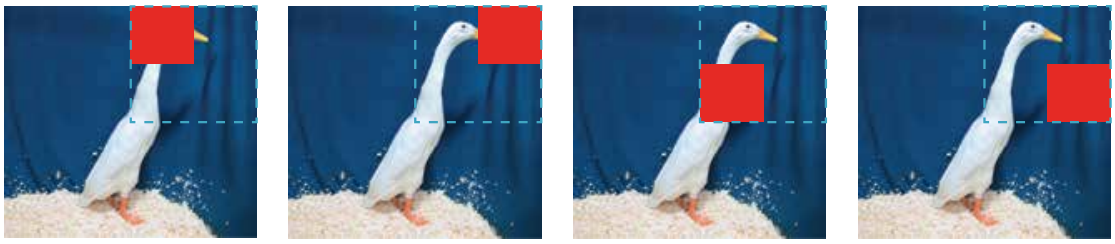
accuracy while covering some parts of the image. Note that, MANGO being model agnostic, any other model could be plugged in instead. The model initially predicts the Figure 3.1 to be a "goose" with 95% confidence. As we cover different parts of the image, the classifier's confidence decreases by different amounts. For instance, Figure 3.2 shows that covering the up-right quarter of the image, i.e. Figure 3.2b, causes the largest drop in confidence. This drop in confidence might be due to the higher importance of this part to the classifier compared to the other parts. Having this assumption, we want to see if there is any subpart in the up-right corner which matters even more than the whole up-right corner to the model and so forth.

(a) goose 85% (b) goose 10% (c) goose 70% (d) goose 40%

Figure 3.2: First level of the MANGO tree.

According to Figure 3.3, covering just the bottom-left corner of the initial corner results in the highest accuracy drop. Even though the accuracy increases a bit from 10% to 20%, still 20% accuracy has a significant difference with other accuracies from Figure 3.3.

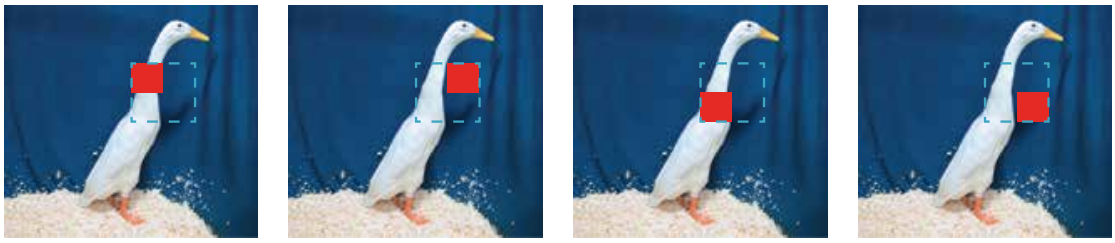
[1]Image from https://wildacres.ca/indian-runner-ducks/



(a) goose 70% (b) goose 60% (c) goose 20% (d) goose 85%

Figure 3.3: Second level of the MANGO tree.

As shown in Figure 3.4, in the next level of the tree, uncovering some parts of a previously covered part of the image results in confidence gain for the model, which means uncovering more is not helpful for our purpose. The complete MANGO tree for this example is depicted in Figure 3.5.



(a) goose 80% (b) goose 80% (c) goose 70% (d) goose 80%

Figure 3.4: Third level of the MANGO tree.

Considering a confidence threshold to determine when to uncover more parts of a masked section can change the MANGO result. Without any thresholds, the most affecting part of the image ends up being the red area in Figure 3.2b whereas it seems what distracted the model to predict the Indian Runner duck as a goose is specifically its long neck. As a result, it seems to be helpful to have a threshold in some examples. Figure 3.6 shows the relation between model confidence and the depth of the tree. The confidence will decrease by increasing the depth to a minimum point where the model starts gaining confidence. Different thresholds will determine which point in the diagram is chosen to be the minimum confidence.

The conjecture about these steps is that when the model starts to gain confidence, the (sub-)parts getting uncovered are the most effective part of the image with respect

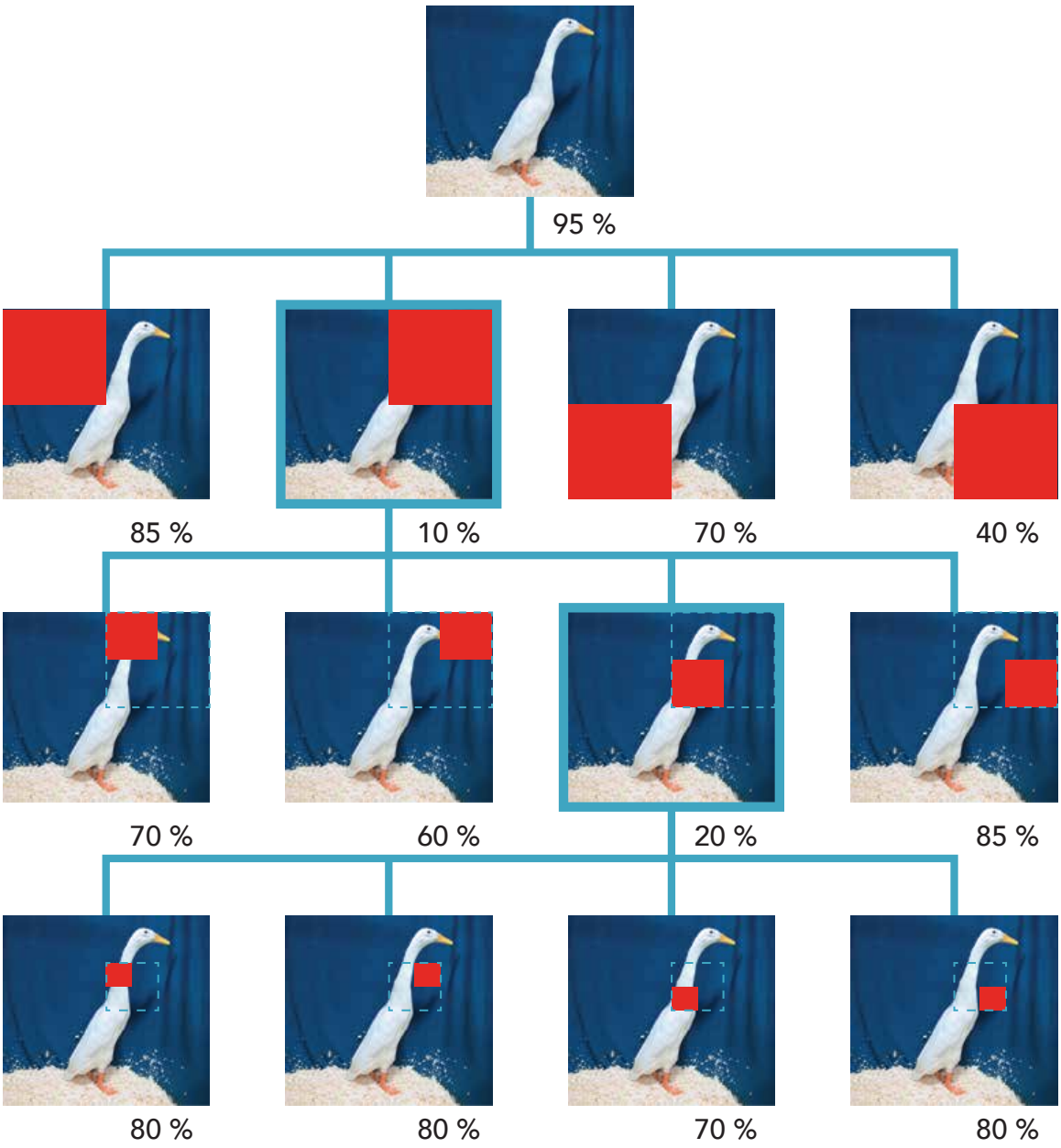


Figure 3.5: The tree build by MANGO algorithm.

to the model's first prediction (which is "goose" in our example). This is the main idea we are following in our proposed approach, i.e. using the model itself to find the most determining part of the image and cover it. We conjecture that by applying the idea taken from MANGO to replace the randomness used in the Cutout to choose the location of the masks, we can train a more generalized model with higher accuracy. In the following section, we will expand on our approach and the reasoning behind

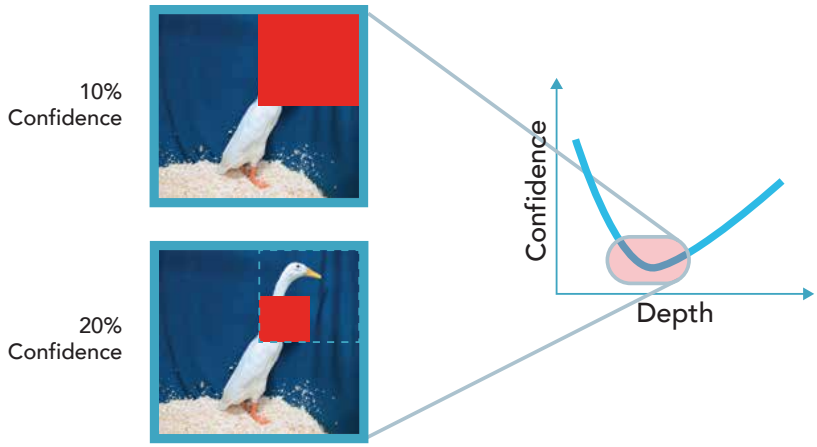


Figure 3.6: Confidence with respect to the depth of the tree. The confidence will decrease by increasing the depth to a minimum point where uncovering more from the masked part will increase the confidence.

our assumption of being effective.

3.2 Main Approach

As explained in the previous chapter, the authors of [1] claim that relying on randomness in choosing the mask locations is helpful enough without any extra cost. Despite their claim, we hypothesize that we can improve the accuracy by changing the random approach in Cutout for masking images to an approach using the MANGO idea to ask the model which part to mask.

According to 2.4.1.1, Cutout is set as a data transformation of a data loader and in each epoch is applied to each data point while loading the data with the data loader. We define a new function described in Algorithm 2 and instead of applying Cutout, we apply this new method on the data points.

According to Algorithm 2, given the inputs of an image tensor (X), training model ($MODEL$), number of the branches at each level (N), initialized mask size (L), and minimum mask size (l_{min}) MANGO will return $target$, a copy of the input image which might be partially masked. The decision of whether to mask the input image or not and where to put the mask is made in the while loop starting at line 3. First,

Algorithm 2 MANGO

Input: $(X \in \mathbb{R}^{c \times h \times w}, Model, N, L, l_{min})$
1: $probability, label \leftarrow$ MODEL(X)
2: $target \leftarrow X$
3: **while** $L \geqslant l_{min}$ **do**
4: $Masked\_Imgs \leftarrow$ Matrix of size $N \times c \times h \times w$ containing masked versions of the image with masks of size $L \times L$
5: $b \leftarrow 0$
6: **for** $n \leftarrow 1$ to N **do**
7: $probability_n \leftarrow$ MODEL$(Masked_Imgs_n)[label]$
8: **if** $probability_n < probability$ **then**
9: $probability \leftarrow probability_n$
10: $b \leftarrow n$
11: $target \leftarrow Masked\_Imgs_n$
12: **end if**
13: **end for**
14: **if** b is not 0 **then**
15: $L \leftarrow L/N$
16: **else**
17: **return** $target$
18: **end if**
19: **end while**
20: **return** $target$

the label and the probability of the label are predicted using the given $MODEL$ (line 1). Creating child nodes, calculating the model confidence for the root label for each child, and deciding on which branch to expand at the next level, is done in the while loop.

In all our experiments, the initial mask size L is set to 16, and it will get updated at each level. The while loop will go on to a depth where mask size L at that level gets to l_{min}. N child nodes with a square-shaped masked part of size $L \times L$ are created in each iteration. In order to create the child nodes, N tensors of ones with the same shape as the input image will be created. Then according to the size of the mask in that depth of the tree, a square part of each of these N tensors will be filled with the colour of the mask, which will be 0 (or black) following the Cutout. Then the output probability of the model for each masked child given the root *label*

is calculated in line 7. The child with minimum probability will be the branch (b) to be expanded in the next iteration with a mask of size $(L/N) \times (L/N)$. This loop will go on till none of the children has a smaller probability than the parent node, or we reach the l_{min} mask size. In the first case, the parent node of that level which also was last assigned to $target$, will be returned (lines 16, 17). The returned $target$ can even be the original image without any masked part. In the second case, the last image assigned to $target$, which is the one with minimum confidence up until the depth we went down, will be returned (line 20).

As with Cutout, our approach can be used with other regularization techniques for better performance. However, we consider our proposed approach of potential because it allows us to use any powerful pre-trained models leading to possible improvements.

According to the above explanations, some parameters or factors are important in the algorithm's performance. In the rest of the section, we will go over these factors, see the possible choices, and argue which are intuitively best to choose.

What is the Model() inside Algorithm 2 and how is it chosen?

The MODEL() can be either a pre-trained model or a newly initialized model. In both cases, the model is updated and passed to the MANGO function. In each call to MANGO, the latest update of the training model decides where to put the masks on each image.

If we use a fixed pre-trained model without any updating, in almost all epochs for a specific image data point, the model's decision on the location and size of the mask will be the same. As a result, for almost all the epochs, a fixed masked version of the data points will be used as the training data leading to a very low data diversity which is against one of the main goals of using data augmentation. However, passing the training model to the MANGO function every epoch after updating it produces many more different versions of each data point, increases data quantity and diversity, and hopefully results in higher accuracy and a more generalized model.

How is the threshold being applied while comparing confidences?

As explained in the previous section, a confidence threshold can be considered when comparing the probability of children to their parents. Adding such a threshold will change the if condition in line 8 of Algorithm 2 to a condition like:

$$probability_n < \tau \times probability \tag{3.1}$$

Where τ is the threshold which can be a fixed number or can vary depending on the level of the tree. Later we will show that running experiments with a fixed threshold of 1 and a minimum limit on mask size is efficient enough for our purpose. That is why we do not bring τ in our algorithm.

Why do we limit the height of the tree?

The *while* condition in line 3 is the condition limiting the height of the tree with the help of variable l_{min} as the minimum size of the masks. The reason behind setting a limit for the mask size, and thus the depth we get to, is that by minimizing the mask length, the ratio of the masked area to the image area $((l_{min} \times l_{min})/(w \times h))$ is minimized as well. If this ratio becomes too small to not be distinguishable in the original image, the masking can lose its effectiveness.

This issue specifically may happen when applying zero masks on CIFAR datasets where images have a small size of 32×32 and in most of the images, the main parts of the images are black or dark-coloured after normalizing. As a result, such a small masked area is not distinguished or does not change most images. Figure 3.7 shows a random batch of images from both CIFAR-10 and CIFAR-100 datasets after applying the standard augmentations (random crop and horizontal flip) and being normalized using standard deviation and per-channel mean.

On the other side, increasing the depth adds to the algorithm cost by making the *while* loop longer, resulting in more calls to the model for each new child node at the new level.

(a) CIFAR-10 (b) CIFAR-100

Figure 3.7: A sample batch of CIFAR datasets after applying Random Crop, Horizontal Flip and Normalization.

Due to all the above reasonings, we will limit the depth of the tree instead of letting the algorithm go down to really small and impractical mask sizes such as 1×1 (one pixel) for CIFAR images.

What is the mask colour?

Another important parameter is the colour of the masks. Even though Cutout only relies on zero masks, i.e. masks filled with zero value pixels, which will be an all-black mask, we hypothesize that MANGO can benefit from having masks of different

colours.

Each mask can still be mono-colour, but instead of being black, the colour can be chosen randomly for each image each time running MANGO. This approach will decrease the chance of a zero mask getting lost between many zero pixels of a normalized CIFAR image. However, there is still a possibility of the mask and the part of the image mask being laid on to be matched or be close in colours causing the same previous problems lessening the effect of the mask. Following the idea of the most efficient version of Random Erasing from Table 2.1, we can choose a random colour between 0 and 255 for each pixel of the mask to combat the mentioned issue. As a result, the probability of all mask's pixels matching with the masked area is almost zero, solving the issue mentioned above. We will refer to this approach as "Rand-Per-Pixel".

Intuitively, masking a portion of the image with a mask that contrast the original pixels' colours the most, seems to be the best practise. In order to apply this idea, we can calculate the average pixel values ranging from 0 (black) to 255 (white) for each channel of Red, Green and Blue, and then for each mask convert the pixel values below the average to 255 and those above the average to 0. This approach will result in having 8 different pixel values which will result in having pixels of colour black (R =0, G = 0, B = 0), red (R =255, G = 0, B = 0), green (R =0, G = 255, B = 0), blue (R =0, G = 0, B = 255), yellow (R =255, G = 255, B = 0), magenta (R =255, G = 0, B = 255), cyan (R =0, G = 255, B = 255), and white (R =255, G = 255, B = 255). We will refer to this approach as "Faredge" method. Figure 3.8 shows a random batch from CIFAR-10 dataset which is masked by MANGO with each of above different colouring approaches.

(a) (b)

Figure 3.8: A sample batch of CIFAR-10 dataset being masked by MANGO using a Resnet18 model and following the (a)Rand-Per-Pixel and (b) Faredge masking.

How does the number of branches at each level affect the algorithm performance?

N is the parameter determining the number of child nodes or branches at each level. Increasing N will increase the cost of the algorithm exponentially. If number of the branches is multiplied by a constant variable k, then number of the nodes at level l of the tree will be k^l times more. Moreover, having more branches means having a smaller mask size at each level which needs close observation in order not to pass the threshold for the mask size and height of the tree (mentioned in previous paragraphs).

On the other hand, increasing N can be beneficial in dividing the image (or a sub-part of the image) into smaller areas and increasing the mask coverage diversity, which will increase the chance of covering and only covering the most determining part of the image. Figure 3.9 shows an example of setting $N = 9$.

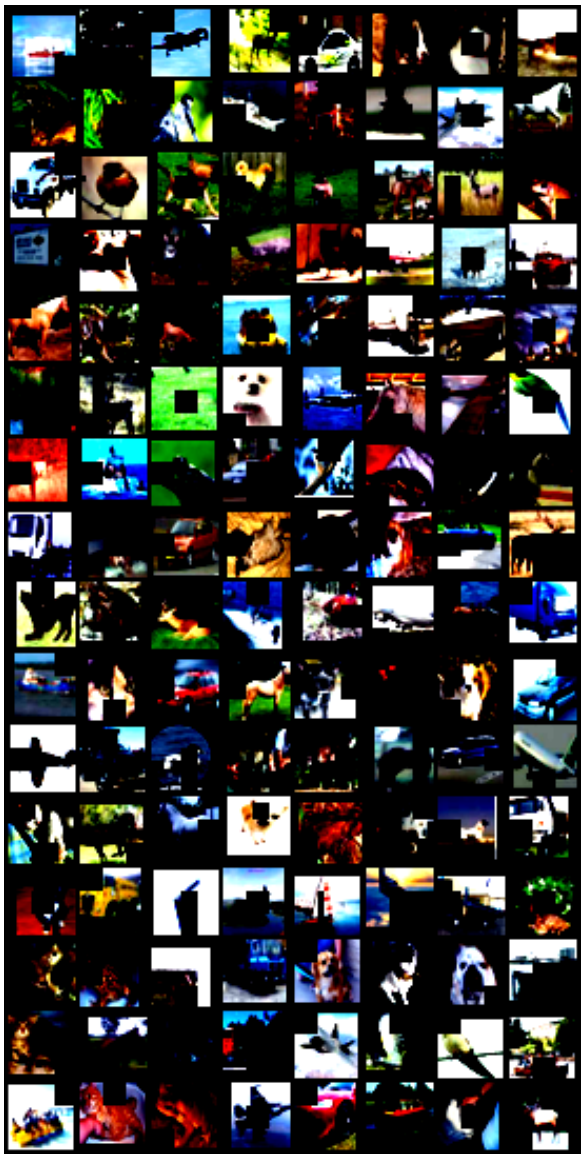


Figure 3.9: Applying MANGO with $N = 9$ on a batch of CIFAR-10 dataset using Resnet18 model.

We conjecture that increasing the number of branches might be helpful in terms of defining more fine-grained masks as far as it does not surpass our computational resources and does not push our limits for the size of the masks.

3.3 Summary of Our Approach

This chapter proposed an approach using the main idea behind the MANGO algorithm to see if replacing it with the randomness in Cutout can beat its efficiency. Instead of randomly masking as Cutout does, we use a model to identify which portion of the image affects the model decision the most, and then we will mask out those regions while training. The goal behind our approach is to encourage the model to pay attention to the entire image and not only a subset of it. Also, it combats the visual challenge of occlusion. We also propose to apply some modifications to the size and colour of the masks to see if they can increase the accuracy of the original MANGO.

In the next chapter, we will design, run, and analyze experiments to evaluate our approach and to see what modifications can make it better and whether we can surpass Cutout's efficiency or not.

Chapter 4

Experiments

4.1 Setup

We run our experiments on a TITAN RTX GPU and report the average result over 5 runs. We follow all the settings mentioned in the original Cutout paper [1] for the sake of fair comparison. In the two following sections, we explain our settings for the datasets and the models we use in detail.

4.1.1 Datasets

We use two well-known natural image datasets of CIFAR-10 and CIFAR-100 in our experiments and evaluations. CIFAR datasets are collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton from 80 million Tiny Images dataset [43][41] consisting of 60000, 32×32 colour images used to train and test different Machine Learning models.

CIFAR-10 consists of 10 classes, each with 5000 training and 1000 testing images. Figure 4.1 shows 10 randomly chosen pictures from each class in this dataset. CIFAR-10 divides into five training batches and one testing batch, each of size 10000 images. Test batches include 1000 randomly selected images from each of 10 classes, but training batches may not include the same number of images from each class. However, the total number of images from each class in all batches is the same.

CIFAR-100 has 100 classes grouped into 20 superclasses giving each image a fine

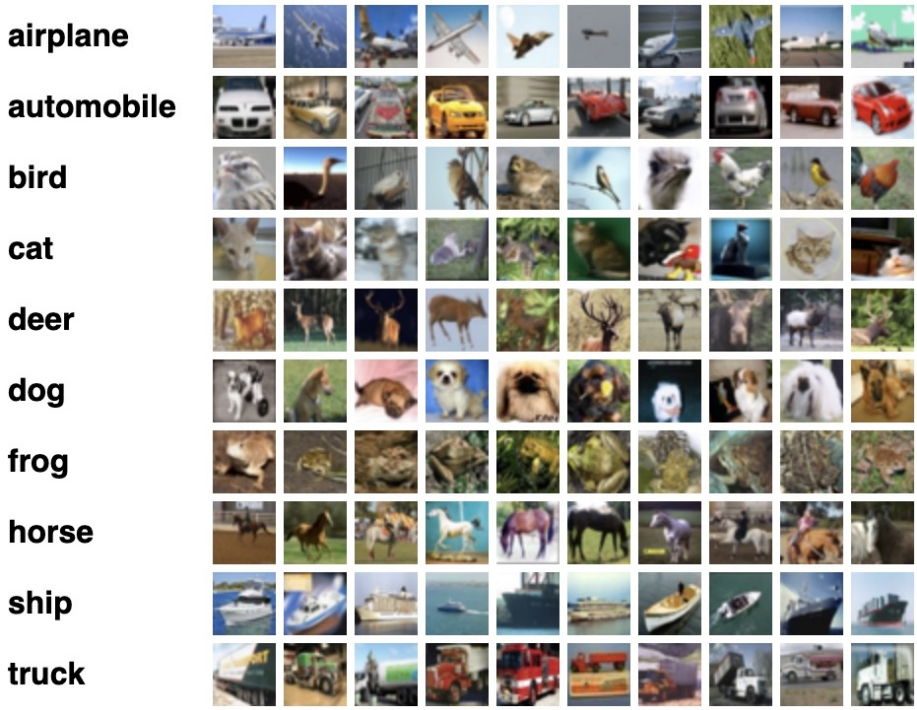


Figure 4.1: CIFAR-10 classes with 10 random examples taken from each class [44].

label (actual class) and a coarse label (superclass). Each class of CIFAR-100 has 500 training images, and 100 testing images [41, 44]. Because of the visual similarity between subclasses of each superclass, CIFAR-100 needs a more fine-grained recognition compared to CIFAR-10. Table 4.1 lists 20 superclasses and 100 subclasses in CIFAR-100.

The Criteria for assigning a label to an image of CIFAR datasets were as follows:

- The class name should stand high on the list of probable answers to the question of "What is in this picture"?
- Labellers rejected any line drawings as the images should be photo-realistic.
- Only one prominent instance of the class object should be in the image.
- The viewpoint from which the object is being seen may be unusual, or the object could be partially occluded, but its identity should remain clear to the labeller.

Following the Cutout settings, in our experiments and evaluations, images are first

Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Table 4.1: CIFAR-100 superclasses and subclasses.

zero-padded by 4 pixels meaning adding 4 zero (black) pixels on each side to make them a 40×40 image, and then we randomly crop a 32×32 part of it. Next, we apply the Random Horizontal Flip on the images with the probability of 50%. Finally, the datasets are normalized using the standard deviation and mean value per channel.

4.1.2 Networks

We use two architectures in our experiments on CIFAR datasets: ResNet18, which is a deep residual network with a depth of 18 [14], and WRN-28-10 which is a wide residual network with a depth of 28 [15] and a widening factor of 10. Also, we apply dropout with a drop probability of $p = 0.3$. All the settings for the training process are the same as the training phase in Cutout [1], i.e. the models are trained for 200 epochs using batches of size 128. The learning rate starts from 0.1 and is divided by 5 after epochs 60, 120, and 160. As for the loss function, we use cross-entropy loss. Also, Stochastic Gradient Descent is used for optimization with Nesterov momentum of 0.9 and weight decay of 5e-4. In the testing phase, we calculate the prediction accuracy using Equation (2.8).

4.2 Results

4.2.1 Original MANGO

In all experiments, we apply MANGO on each batch of training data with the probability of 50%. Our preliminary results suggest that by doing so, we enhance the time efficiency without affecting the accuracy (taking the idea from [1] as mentioned in Section 2.4.1.1). We present the results of applying MANGO on the training data in Table 4.2. The results are averaged over 5 runs. As shown before in Table 2.3, Cutout beats the base models on CIFAR datasets using WRN-28-10 and ResNet18 models. According to Table 4.2, MANGO slightly outperforms Cutout by decreasing the error of ResNet18 by 0.08% and 0.40% when applied on CIFAR-10 and CIFAR-100 datasets, respectively. Also, the result for the WRN-28-10 model is improved by 0.04% on CIFAR-10 and by 0.40% on CIFAR-100.

One drawback of MANGO compared to Cutout is the time efficiency. As shown in Table 4.2, MANGO is more than twice slower than Cutout. This is due to the several calls to model to pass the test data in order to find the mask location. Depending

on our application, this extra cost might be tolerable considering the improvement in the results.

Model	**CIFAR-10**	**CIFAR-100**	**Time / Epoch**
ResNet18 (Baseline)	4.81 ± 0.13	22.00 ± 0.09	**20.2**
ResNet18 + Cutout	3.89 ± 0.28	22.00 ± 0.14	22.8
ResNet18 + MANGO	$\mathbf{3.81 \pm 0.11}$	$\mathbf{21.60 \pm 0.28}$	53.1
WRN-28-10 (Baseline)	3.83 ± 0.02	18.80 ± 0.17	**83.0**
WRN-28-10 + Cutout	3.02 ± 0.11	18.20 ± 0.17	**83.0**
WRN-28-10 + MANGO	$\mathbf{2.98 \pm 0.02}$	$\mathbf{17.80 \pm 0.05}$	204.2

Table 4.2: Comparing test error rates (%) and the average time per epoch (s) for ResNet18 and WRN-28-10 models running on the CIFAR-10 dataset using baseline, Cutout, and MANGO. Baseline indicates a model trained with standard augmentations of mirror + crop without using Cutout or MANGO. Cutout implementation is based on https://github.com/uoguelph-mlrg/Cutout. Results are with 95% confidence intervals and averaged over five runs.

In the following sections of this chapter, we will evaluate different versions of the MANGO using the ResNet18 model on the CIFAR-10 dataset. We concentrate only on one pair of model and dataset in these experiments, assuming that different variations will affect qualitatively the same other models and datasets.

4.2.2 Colored Mask MANGO

As mentioned previously in Section 3.2, we follow two approaches for choosing the colour of the mask: Rand-Per-Pixel and Faredge. The results of colouring the masks with both mentioned methods while applying the MANGO on the CIFAR-10 dataset using the ResNet18 model are shown in Table 4.3. The results are averaged over 5 runs.

According to Table 4.3, using the Rand-Per-Pixel or Faredge approach to colour the masks is not improving the result of MANGO. Among these two approaches, Rand-Per-Pixel has slightly better performance than Faredge, but both approaches

have considerably more cost than Cutout. One reason for the difference between the two colouring approaches could be the full coverage of the Rand-Per-Pixel mask over the masking area compared to the Faredge coloured mask. As it is shown in Figure 3.8 when using Faredge, the masked area tends to maintain the content just with different colouring, whereas Rand-Per-Pixel hides the content of the masked area completely. We can justify the superiority of the black masks in Cutout and original MANGO over Faredge masks with the same reasoning.

Method	**Error**	**Time / epoch**
Original MANGO	$\mathbf{3.81 \pm 0.11}$	**53.1**
Rand-Per-Pixel	3.97 ± 0.11	71.4
Faredge	4.11 ± 0.12	142.0

Table 4.3: Test error rates (%) of two mask colouring approaches in MANGO and the average time per epoch (s) compared with the original MANGO. Experiments are on the CIFAR-10 dataset using ResNet18, with 95% confidence intervals averaged over five runs.

4.2.3 MANGO with Different Number of Branches

In all above mentioned experiments the variable l_{min} from Algorithm 2 is set to be 8 and the threshold τ from Equation 3.1 is set to be 1. As described in Section 3.2, we want the ratio of the masked area over the image area to stay at a reasonable amount. Setting the $l_{min} < 8$ results in the possibility of having a mask of size 4×4 or less, which, considering the small size of CIFAR images, makes the mentioned ratio very small (less than 1.5%). Also, having a threshold with the same idea of MANGO explained in Section 3.1 will encourage the algorithm to increase the depth and thus decrease the mask size. As a result, we set the threshold $\tau = 1$, and we control the depth of the tree by thresholding the mask size and using the variable $l_{min} = 8$. In this section, we will analyze an experiment that is the only one with $l_{min} = 3$.

As mentioned in Section 3.2, MANGO may benefit from having more branches at

each level. In order to verify this, we run experiments on the CIFAR-10 dataset using ResNet18 and setting the $N = 9$ from Algorithm 2. As we want to divide each side of the image (i.e. h and w) equally, instead of dividing each side to 2 as it is in the original MANGO, we divide each side to 3, which results in $N = 9$. The l_{min} is set to 3 and the initial mask size is $L = 11$. Table 4.4 shows the result of this version of MANGO averaged over 5 runs. As the results suggest, increasing the number of branches did not help the model's performance or time efficiency. The nuanced difference in error between this version and the original MANGO could be due to the different mask sizes. As mentioned in Section 2.4.1.1, according to [1] the optimum mask size for CIFAR-10 dataset is 16×16 and in the original MANGO the masks are either 16×16 or 8×8. Thus, the original version shares more similarity with the Cutout's optimum mask size than the latter version where masks are either 11×11 or 10×10 in the first level and 5×5 or 3×3 in the second level. The increase in time is due to the increase in the number of the nodes at each level which will result in a higher cost for MANGO with $N = 9$ as explained in Section 3.2.

Method	**Error**	**Time / Epoch**
Original MANGO	$\mathbf{3.81 \pm 0.11}$	**53.1**
MANGO with $N = 9$	3.89 ± 0.08	91.2

Table 4.4: Test error rate (%) and the average time per epoch (s) of MANGO with $N = 9$ compared with the original MANGO. Experiments are on CIFAR-10 dataset using ResNet18. Results are with 95% confidence intervals averaged over five runs.

4.2.4 Comparing MANGO Variants

As it is shown in Figure 4.2, all versions of MANGO have comparable performance to Cutout. Even though all versions of MANGO showed comparable results in our experiments, none of them have superiority over the original one. We plot Cutout versus original MANGO in Figure 4.3 during all 200 epochs of training and also in the last 40 epochs in order to see the final improvements of MANGO over Cutout

more clearly. According to these two figures, with little difference, MANGO reaches a higher accuracy than Cutout.

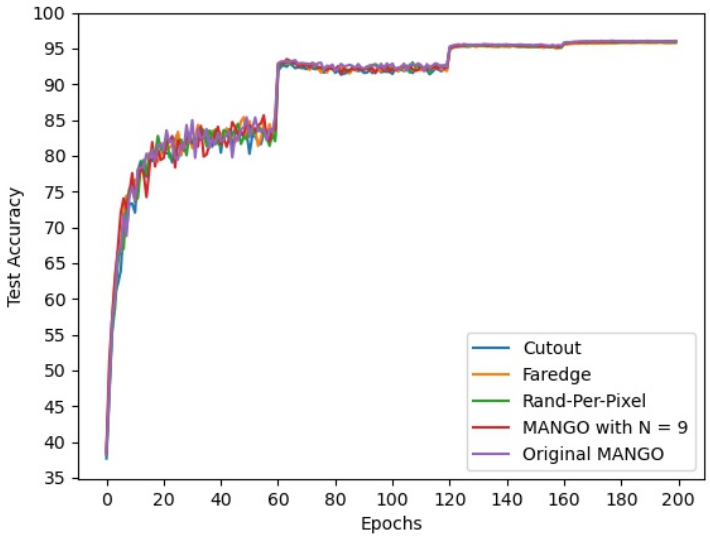


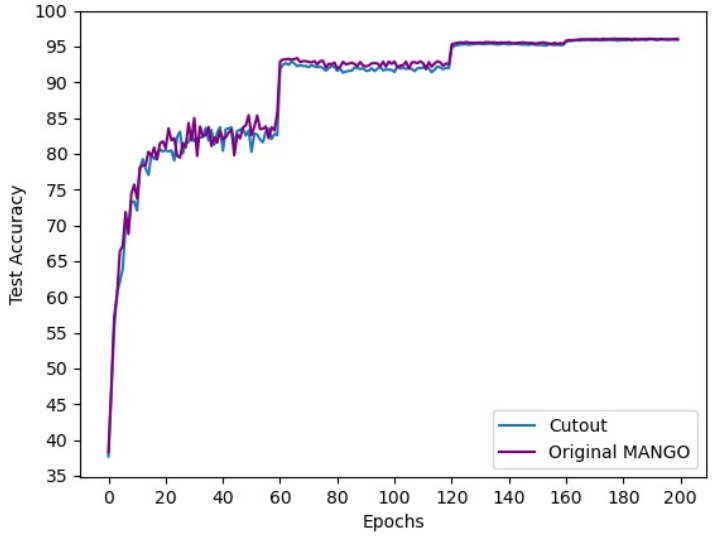
Figure 4.2: Accuracy of all versions of MANGO VS Cutout using CIFAR-10 dataset and ResNet18. Results averaged over 5 runs.

4.2.5 MANGO's Effect on Activations

In this section, we plot and compare the magnitude of feature activations to clarify and compare the effects of the different augmentation approaches on the data.

In Figure 4.4 we use a ResNet18 model trained on CIFAR-10 to calculate and compare the magnitude of feature activations in all residual blocks of the model, averaged over all test samples from CIFAR-10 and sort them in descending order. The plotting is done in three settings: applying Cutout, applying MANGO, and applying none of them, represented as the baseline model. We apply the standard data augmentations (mirror + crop) before any other augmentation method in all settings. The configurations for MANGO are the same as those mentioned in Section 4.1, and configurations for Cutout are the same as the default settings from [1].

According to Figure 4.4 in the first layers of the network, we have more differences in the magnitude of the activations compared to the deeper layers. Using MANGO



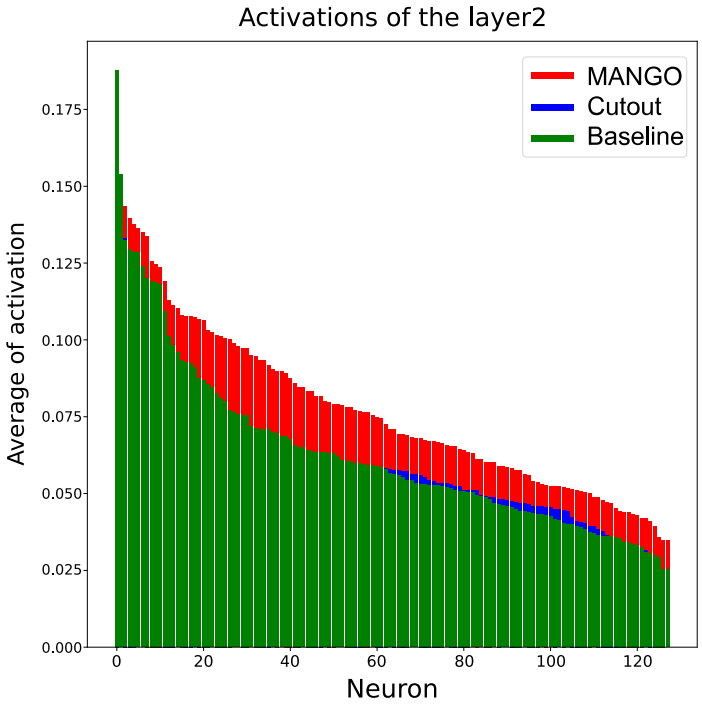
(a)

(b)

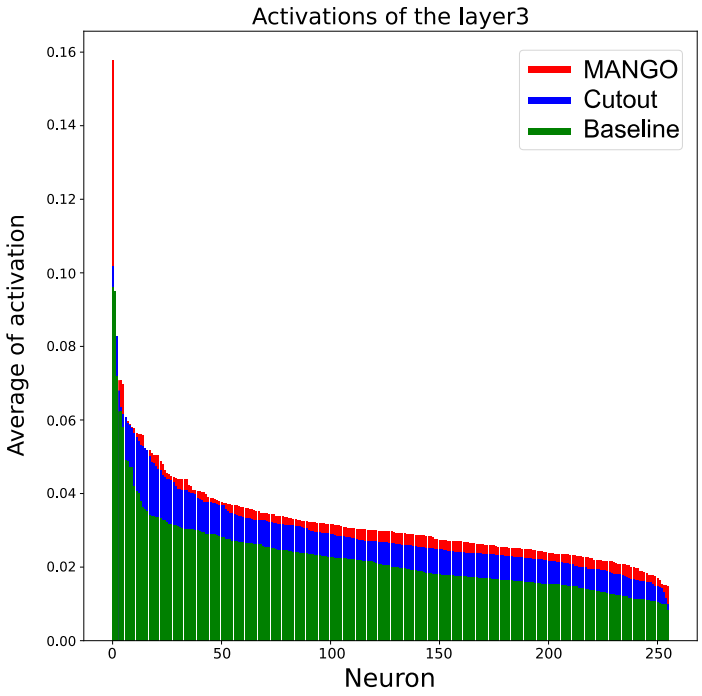
Figure 4.3: (a) Accuracy of MANGO VS Cutout run over CIFAR-10 dataset using ResNet18 and averaged over 5 runs. (b) Last 40 epochs of results from Figure 4.3a.

strengthens the activations more than Cutout and both more than the base model. In [1], the authors claim that the greater improvement of activations in shallow layers compared to the deeper layers proves the fact that Cutout encourages the model to consider more diverse features instead of focusing only on a small subset of features.

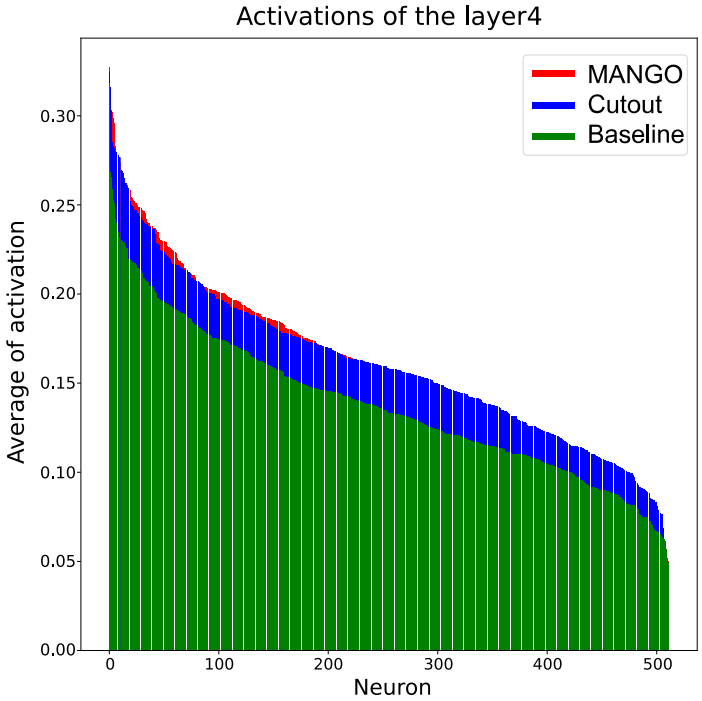
This might be due to the fact that shallower layers in deep neural networks usually detect the more general features of the images, e.g. corners, colours conjunctions etc., whereas the last layers detect more invariant class-specific features [45]. Figure 4.4 shows that MANGO makes the difference between the activations of the first and last layer even more than the Cutout. With the same reasonings from [1] we can say MANGO performs better than Cutout in encouraging the model to take into account more diverse features, thus helping it to generalize better.



(a) Average of activations magnitude in the 2nd residual block.



(b) Average of activations magnitude in the 3rd residual block.



(c) Average of activations magnitude in the 4th residual block.

Figure 4.4: Feature activations' magnitude averaged over all test samples, sorted by descending value. A baseline ResNet18, a ResNet18 trained with Cutout, and a ResNet18 trained with MANGO are compared at 3 residual blocks of different depths.

Chapter 5

Conclusion and Future Work

5.1 Thesis Summary

In this thesis, we proposed MANGO, a new data augmentation method that follows the idea of the Cutout regularization technique. In our method, we use the model to find the location of the mask on each training data point based on the effect of the covered area on the model's prediction result. By doing so, we aimed at encouraging the model to pay attention to the full context of the image and not only focus on prominent features. Our experiments on CIFAR datasets using WRN-28-10 and ResNet18 networks proves that MANGO improves the model's performance in the image classification task by increasing the network generalization and making it more robust to issues such as occlusion.

Our experiments show that Cutout is more efficient in time with slightly less accuracy than MANGO. This observation can validate that a computationally cheaper and conceptually simpler approach of randomly choosing the mask locations as Cutout does, is efficient enough.

5.2 Future Work

Our experiments only focus on the Image Classification domain. It will be interesting to see how MANGO will perform in other vision tasks such as Object Detection or Person Re-identification (re-ID). Also, it might be helpful to parameterize some

elements in the MANGO algorithm, such as the shape or number of the masks and find an optimal setting for them. For instance, we will have two masked areas in circle and triangle shapes on one image. In all these possible future studies, the challenge of making the MANGO more cost-efficient remains to be addressed.

Bibliography

[1] T. DeVries and G. W. Taylor, *Improved regularization of convolutional neural networks with cutout*, 2017. arXiv: 1708.04552 [cs.CV].

[2] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[3] *Cs231n convolutional neural networks for visual recognition*. [Online]. Available: https://cs231n.github.io/.

[4] C. S. Nielsen, *Improving image classification through generative data augmentation*, 2019. DOI: 10.11575/PRISM/10182. [Online]. Available: https://prism.ucalgary.ca/handle/1880/110365.

[5] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[7] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: 1502.03167 [cs.LG].

[8] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[9] L. Shao, F. Zhu, and X. Li, "Transfer learning for visual categorization: A survey," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 5, pp. 1019–1034, 2014.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[12] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3712–3722.

[13] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?" In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 201–208.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[16] K. He, X. Zhang, S. Ren, and J. Sun, *Identity mappings in deep residual networks*, 2016. arXiv: 1603.05027 [cs.CV].

[17] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, *Cutmix: Regularization strategy to train strong classifiers with localizable features*, 2019. arXiv: 1905.04899 [cs.CV].

[18] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 13 001–13 008.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[25] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 648–656.

[26] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," in *Asian conference on computer vision*, Springer, 2016, pp. 189–204.

[27] G. Kang, X. Dong, L. Zheng, and Y. Yang, *Patchshuffle regularization*, 2017. arXiv: 1707.07103 [cs.CV].

[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[29] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[31] Y. Bengio, F. Bastien, A. Bergeron, N. Boulanger–Lewandowski, T. Breuel, Y. Chherawala, M. Cisse, M. Côté, D. Erhan, J. Eustache, X. Glorot, X. Muller, S. Pannetier Lebeuf, R. Pascanu, S. Rifai, F. Savard, and G. Sicard, "Deep learners benefit more from out-of-distribution examples," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA: PMLR, 2011, pp. 164–172. [Online]. Available: http://proceedings.mlr.press/v15/bengio11b.html.

[32] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, 321–357, 2002, ISSN: 1076-9757. DOI: 10.1613/jair.953. [Online]. Available: http://dx.doi.org/10.1613/jair.953.

[33] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[34] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[35] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[36] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[37] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[38] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[39] J. Lemley, S. Bazrafkan, and P. Corcoran, "Smart augmentation learning an optimal data augmentation strategy," *Ieee Access*, vol. 5, pp. 5858–5869, 2017.

[40] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation policies from data," *arXiv preprint arXiv:1805.09501*, 2018.

[41] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[42] C.-H. Chang, E. Creager, A. Goldenberg, and D. Duvenaud, "Explaining image classifiers by counterfactual generation," *arXiv preprint arXiv:1807.08024*, 2018.

[43] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[44] A. Krizhevsky, *Cifar-10 and cifar-100 datasets*. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[45] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, Springer, 2014, pp. 818–833.