Evaluation of Semi-Supervised Learning Techniques Using Convolutional Neural Network

Praveen Rao Nittoor
pnittoor@asu.edu

Varun Rao Veeramaneni
vveeram2@asu.edu

Amit Singh
asing227@asu.edu

Avinash Reddy Samala
asamala2@asu.edu

Abstract

Semi-supervised learning is a combination of supervised and unsupervised learning which makes use of unlabeled data along with small amount of labeled data for training of a machine learning model. In this paper, we test and compare classification accuracies, confusion matrices and plot loss function of the following semi-supervised learning algorithms, viz.: Entropy Minimization, CutMix, MixUp using convolutional neural network. We analyze and report the performance by applying all the three techniques together and by varying the number of labelled and unlabeled samples for each technique using CIFAR 10 dataset.

Index Terms – Convolutional Neural Network, CutMix, MixUp, Entropy Minimization, Padding, Pooling

1. Introduction

Unprecedented growth is seen in the area of computer vision with the capability of machine able to use intelligence. Convolutional neural network (CNN) is one such deep learning algorithm used mainly for Image Classification. CNN is like any other neural network except that it has extra hidden layers which are called convolutional layers. A filter matrix or matrices are initialized for each layer and the convoluted matrix is calculated. The convoluted matrix generated can be made of the same dimensions of the image matrix using padding. These convoluted matrices can be further processed using the maximum or average pooling techniques to generate a pooling layer. Pooling is done to reduce dimensionality and preserve spatial invariance. Both the convoluted and the pooling layer form a layer of the CNN. The layers can be constructed hierarchically, to train the model, to learn the hierarchy of features and the low-level features of the image. Entropy minimization is to motivate minimum entropy regularization aiming at making most of the unlabeled data when beneficial in a controlled manner.

The method of training the model with similar but different set of data is called data augmentation. The similar data is usually obtained by rotating, zooming, sharpening and saturating the original data. Data augmentation is done to increase the accuracy of the model by reducing the possibility of overfitting. Data augmentation does not improve accuracy when the new data produced does not make any sense and instead adds the complexity. So, the techniques used for data augmentation should be carefully selected. We have used CutMix (Yun et al., 2019) and MixUp (Zhang et al., 2017) as the data augmentation techniques in this paper to show the improvements in accuracy, confusion matrix, error rates.

2. Related Work

2.1. Convolutional Neural Network

In a fully connected neural network, each layer processes the output of all other previous layers. The difference between actual output for a given input and the predicted output for the same input is termed as 'loss'. To optimize a model, this loss is to be minimized. Before that, there is a need to learn the neuron's parameters i.e.; weights and biases. Then a loss function can be used to quantify the loss. The gradient of this loss function is taken to determine where the loss can be minimized. Gradient Descent algorithm is used to iterate through different combinations of weights to find the best combination which minimizes the error.

A fully connected neural network typically don't work well on images. Because if each pixel is an input, as more layers are added, the number of parameters increases exponentially. For example, consider a $32\ x\ 32$ image, that is 32 wide and 32 high. A single fully connected neuron in the first hidden layer of a regular neural network would have $32\ x32\ =\ 1024$ weights. But for a $200\ x\ 200$ image, there are 40000 weights. The number of parameters this large can quickly lead to over-fitting. One work around is to use the smaller images, but clearly there will be a loss in information.

The property that makes one image distinguishable from another is its spatial structure. A convolutional network can successfully capture these spatial dependencies. In image recognition applications, *Convolutional Neural Network* [3] algorithm takes an image as input, assigns parameters (weights and biases) to various objects in the image and distinguish one image from the other. Due to the reduction in number of parameters and reusability of weights, a

convolutional network gives a better fitting to the image dataset.

The functionality of the Convolution[3] is the extraction of high-level features such as edges, from the input image. Generally, the first convolution layer captures the low-level features such as edges, gradient orientation, color, etc. When the layers are added, the architecture adapts to the high-level features, giving a network which has complete knowledge of images in the dataset.

The results of this operation can be classified into two types — one in which the dimensions of convolved feature is reduced as compared to the input by applying Valid Padding, and the other in which the dimensions are either increased or remains the same by applying Same Padding [3].

Like the Convolutional Layer, the Pooling layer[3] reduces the spatial size of the Convolved Feature. This results in a drop in the computational power that is required to process the data. Moreover, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the effective training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Filter. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Filter. Max Pooling is better than Average Pooling because both are used to reduce the dimensionality, but Max Pooling has an additional advantage of suppressing noise. [3]

2.2. Activation Function - ReLU:

Activation functions are the ones that triggers the neural networks. These activation functions can be linear or non-linear. Linear activation functions are useful only when there are only two classes but not for the multi class problem. To ensure that there is some non-linearity in the network, the activation function should be non-linear. The rectified linear unit is computationally less expensive compared to other activation functions, viz. sigmoid, hyperbolic tangent and step function. It is more and well-known activation function.

A rectified linear unit is defined as outputting a zero for any value of 'x' that is less than zero. For any value of 'x' greater than or equal to zero, the function returns an 'x'. ReLU can be represented as follows:

$$f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$$

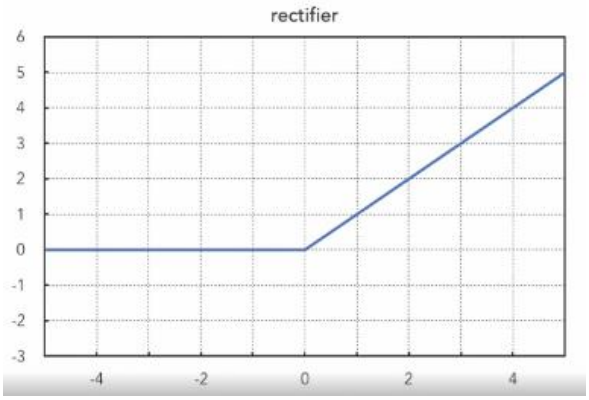

Fig 2.2.1 Graphical representation of Rectified Linear Unit

The rectified linear unit is computationally less expensive compared to sigmoid function. It is more and well-known activation function.

2.3. Regularization

Sometimes, the network model may lead to overfitting -- when a model works very well for the data that it is trained on, but the model provides a poor prediction for any data it hasn't seen before.

The purpose of regularization is to reduce overfitting. Regularization allows to apply penalties on layer parameters or layer activity during optimization. These penalties are incorporated in the loss function that the network optimizes.

In Keras, there are 3 keyword arguments: kernel_regularizer, bias_regularizer and activity_regularizer that can be applied to the layer. Or any function that takes in a weight matrix and returns a loss contribution tensor can also be used as a regularizer.

2.4. CIFAR - 10 Dataset

The CIFAR - 10 dataset consists of 60000 colored images, each of 32x32 size. These images are divided into 10 classes with each class containing 6000 images. [5]

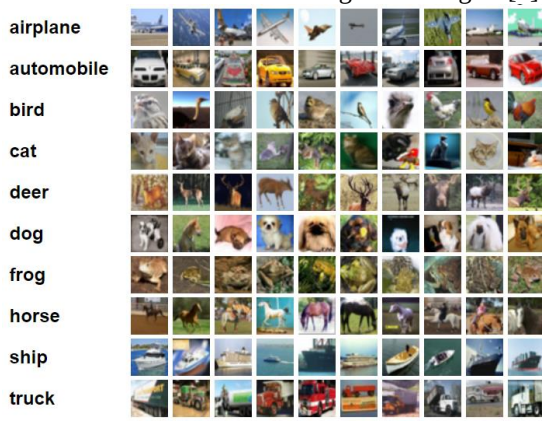

Fig 2.3.1 Sample of CIFAR - 10 dataset

Of these 60000 images, 50000 are training dataset and 10000 are test dataset. The CIFAR - 10 dataset consists of five training batches and one test batch, each with 10000 images.

The test batch contains 1000 images of each class, but each training batch may contain more than or less than 1000 images from a particular class.

2.5. Keras 2.0

Keras is a high-level neural network's API written in python, and capable to run on top of TensorFlow, CNTK (Cognitive Toolkit) or Theano.

Some of the important components of Keras are:
model.compile()
compile(self, optimizer, loss, metrics = None, ...)
Before training a model, there is a need to configure the learning process, which is done via compile method. This method receives 3 arguments.
optimizer: This is the algorithm that given a set of parameters, returns one with a smaller function.
loss: This is an objective function for measuring the accuracy or performance error of a neural network.
metrics: List of metrics. Usually set to 'accuracy'.

Some of the common optimizers are:
SGD (Stochastic Gradient Descent): Includes support for momentum, learning rate decay, and Nesterov momentum.
RMSprop: Divides the learning rates by an exponentially decaying average of the squared gradients. It is a good choice for recurrent neural networks.
Adam: It is the algorithm for first-order gradient-based optimization of stochastic objective functions.

Some of the common loss functions are:
mean_squared_error: It computes the element-wise square defense between the two tenses.
categorical_crossentropy: Computes the categorical cross entropy between predictions and targets, and this is often used when the target has multiple classes.
binary_crossentropy: Computes the binary cross entropy between predictions and targets, and this is often used when the target has 2 classes.

3. Implementation

3.1. Baseline Convolutional Neural Network

In the implementation of baseline CNN, we have used 6 convolutional layers, 3 pooling layers, filters of size 3x3 at all convolutional layers, max pooling of size 2x2 at all pooling layers, with no dropout and ran for 50 epochs.

The *sequential* API in Keras allows to create model in a layer-by-layer way. The convolution layers and pooling layers are added using *model.add()* component of Keras.

In configuring the learning model, compile method *model.compile()* is used. As the dataset consists of multiple classes, the loss function *'categorical_crossentropy'* is used and the optimizer *'SGD'* (Stochastic Gradient Descent) is used with *learning rate* 0.001 and *momentum* 0.9.

3.2. Data Augmentation - CutMix

Regional dropout methods have been proposed to increase the efficiency of a model using CNN to predict the class labels. They increase their efficiency by dropping less informative/discriminative parts of the image and have better localization properties. Previous methods use cut out where the region of information is removed from the image and is trained, we employ a method opposed to deletion. We replace the part removed with another image in the training set and train the model. CutMix uses regularization effect of dropouts and efficiently uses training pixels which surpasses the state-of-the-art technologies on CIFAR-10 dataset [5].

In previous employed methods, a region of data in the image is removed and filling it with random noise or blackening it out, which leads to loss of information. CNN are generally data hungry and in order to improve its efficiency, we take the removed region and insert part image of some other training image to fill the region. The ground truths are mixed proportional to the ratios of the image, so our methods have the efficiency of regional dropouts for localization and no uninformative pixel resides in the image making out training efficient. CutMix generates new samples by cutting and pasting within mini batches, leading to performance improvements in computer vision tasks(Yun et al., 2019).

Algorithm:
Let $x \in \mathbb{R}^{W \times H \times C}$, y be the training set and its labels. We have to generate a CutMix set of the training samples (x', y') by combining two samples $(x_A, y_A), (x_B, y_B)$.
We have

$$x' = M \odot x_A + (1 - M) \odot x_B$$

$$y' = \lambda y_A + (1 - \lambda) y_B$$

where $\odot$ denotes element-wise multiplication with $M \in \{0,1\}^{W \times H}$ is a binary mask indicating where to remove and fill from the two images. λ is the ratio of two sampled points from the beta distribution Beta(α,α).
M is sampled from the bounding box $B(r_x, r_y, r_w, r_h)$

$$r_x \sim Unif(0, W) \qquad r_y = W\sqrt{1-\lambda}$$

$$r_w \sim Unif(0, H) \qquad r_h = H\sqrt{1-\lambda}$$

The binary mask M is filled with 0 if it is within the boundary box B, otherwise it is 1.

3.3. Data Augmentation - MixUp

Most of the neural network applications share two major properties: first, the neural network is trained to reduce the average error over the training dataset, which is known as Empirical Risk Minimization (ERM) principle [6] and second, the size of the neural networks ascends linearly with the number of samples in training dataset. While ERM allows neural networks in memorizing the training data in the presence of strong regularization, there is a drawback of drastic change in the prediction when neural network is evaluated on the data that id different from the training distribution.

The above drawback marked the introduction of a method of choice to train on similar but different examples compared to the training data. This method is called *data augmentation*. [7]. This data augmentation technique pushes the model to behave linearly in-between training examples. This linear tendency reduces the undesirable oscillations while predicting samples(Zhang et al., 2017).

In this paper, we introduced a simple MixUp augmentation to construct virtual training examples

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j$$
$$\tilde{y} = \lambda y_i + (1-\lambda)y_j$$

where x_i, x_j are raw input vectors and y_i, y_j are one-hot label encodings.

(x_i, y_i) and (x_j, y_j) are the random examples of training dataset and $\lambda \in [0,1]$. Thus, MixUp extends the distribution of training dataset by assimilating the prior knowledge that linear interpolations of feature vectors should lead to linear interpolations of the associated targets. [4]

3.4. Entropy Minimization

In the probabilistic framework, semi-supervised learning can be modeled as a missing data problem, which can be directed by generative models such as mixture models using EM algorithm.[2]

Here, we assume that classes are well separated without any overlapping in order to take complete advantage of unlabeled data. The conditional entropy H(Y|X) gives the measure of class overlap, which doesn't vary with parameters of the model. This measure is related to the usefulness of unlabeled data where labeling is inexact. [2]

The conditional entropy of class labels conditioned on the observed variables

$H(Y|X, Z) = -E_{XYZ}[\log P(Y|X,Z)]$

where Ex is the expectation with respect to X.

Computing H(Y|X,Z) requires a model of P(X,Y,Z) whereas the choice of the diagnosis is motivated by the possibility to limit modeling to conditional probabilities. There is a need of additional modeling by applying the plug-in principle, which replaces the expectation with respect to (X,Z) by the sample average. This substitution, which can be interpreted as "modeling" P(X,Z) by its empirical distribution, yields

$$H_{emp}(Y|X, Z; \mathcal{L}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} P(w_k|x_i, z_i)\log P(w_k|x_i, z_i)$$

4. Experimentation

In this section, we discuss the various tests conducted by varying the number of convolutional layers added, number of pooling layers added, the type of data augmentation technique used and percentage of labeled and unlabeled data. We have reported the classification test accuracy and plotted for training accuracy, test accuracy, cross entropy against epochs. We have plotted confusion matrix (fig 4.10) for each result but not shown in the report due to space constraints.

Fig 4.1 shows how the test and train accuracies vary with the number of epochs when a baseline CNN is used without any data augmentation.

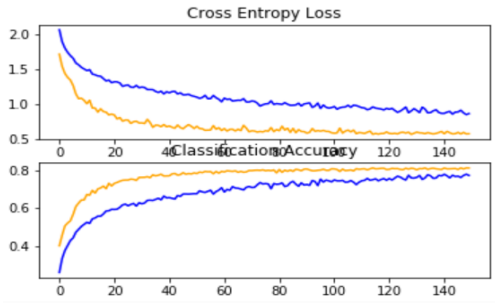

Fig 4.1 Cross entropy loss and Classification Accuracy using Baseline CNN

Fig 4.2 shows how the test and train accuracies vary with the number of epochs when data augmentation technique CutMix is used. The orange graph shows training accuracy and blue graph shows the test accuracy. It also shows how cross entropy loss vary with the number of epochs.

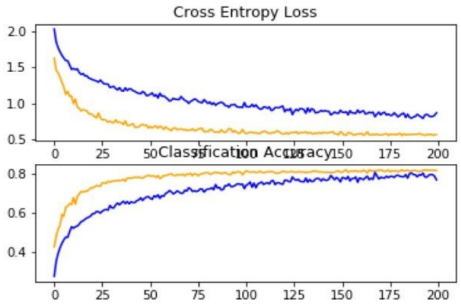

Fig 4.2 Cross entropy loss and Classification Accuracy using data augmentation method - CutMix

Fig 4.3 shows how the test and train accuracies vary with the number of epochs when data augmentation technique MixUp is used. The orange graph shows training accuracy and blue graph shows the test accuracy. It also shows how cross entropy loss vary with the number of epochs. We have also performed experiments by varying the number of epochs for the same technique.

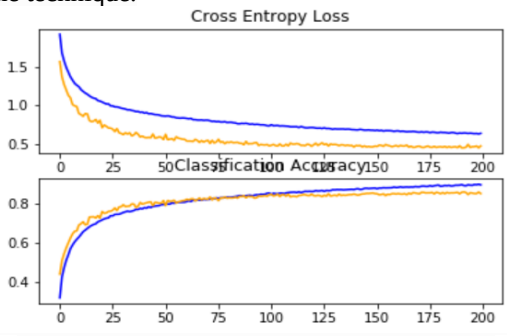

Fig 4.3 Cross entropy loss and Classification Accuracy using data augmentation method - MixUp

Fig 4.4 to fig 4.9 shows the plot for test and training accuracies and cross entropy loss by varying the percentage of labeled and unlabeled data against epochs. These figures are the plots by applying both the data augmentation techniques CutMix and MixUp.

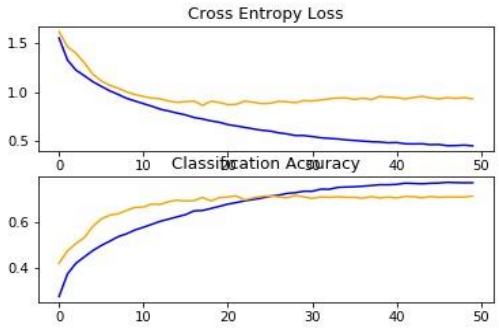

Fig 4.4 Cross entropy loss and Classification Accuracy for 20% unlabeled and 80% labeled data

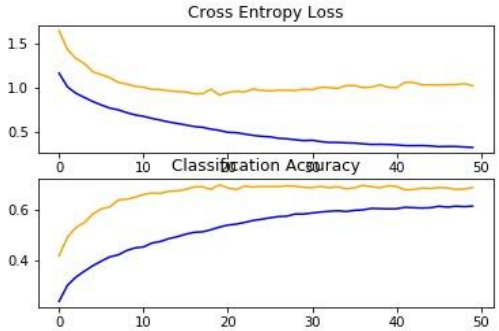

Fig 4.5 Cross entropy loss and Classification Accuracy for 40% unlabeled and 60% labeled data

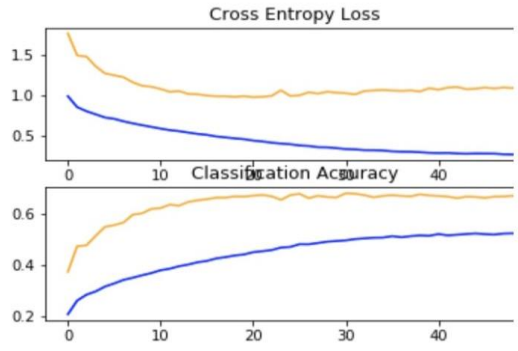

Fig 4.6 Cross entropy loss and Classification Accuracy for 50% unlabeled and 50% labeled data

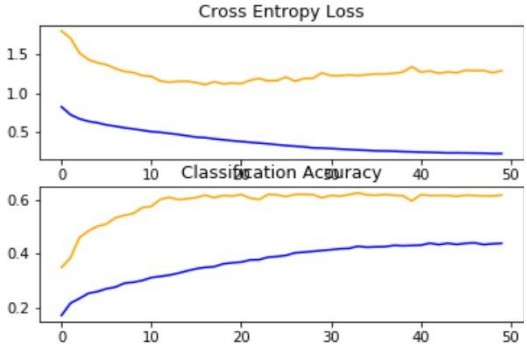

Fig 4.7 Cross entropy loss and Classification Accuracy for 60% unlabeled and 40% labeled data

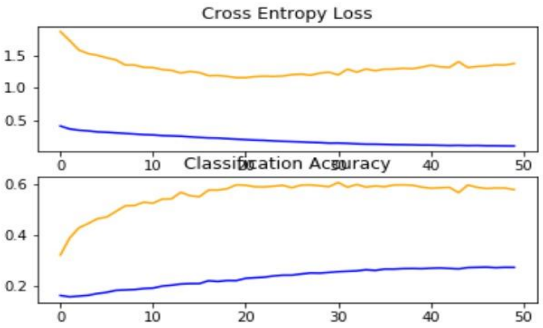

Fig 4.8 Cross entropy loss and Classification Accuracy for 80% unlabeled and 20% labeled data

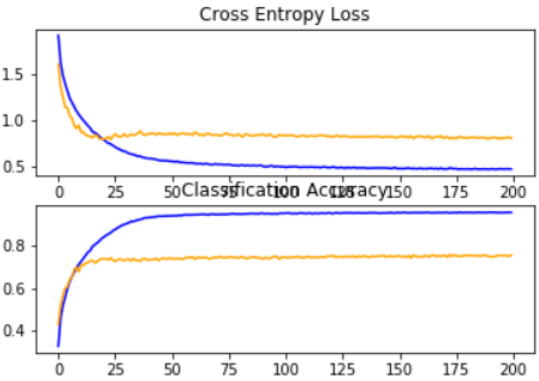

Fig 4.9 Cross entropy loss and Classification Accuracy for 100% labeled data and varying epochs

```
[[763  13  79  20  14   3  12   6  62  28]
 [ 39 763  17  24  10   3  11   6  35  92]
 [ 75   4 607  72  98  63  42  28   7   4]
 [ 23   3 102 561  67 142  65  22   8   7]
 [ 37   3  93  79 600  22  60  87  18   1]
 [ 16   3  99 229  52 539  10  46   3   3]
 [  8   4  73  88  67  19 730   8   2   1]
 [ 27   4  53  59  56  65   7 722   1   6]
 [100  37  25  25  10  12   7   3 764  17]
 [ 53  78  16  30  10   7  16  20  38 732]]
```

Fig 4.10 Showing confusion matrix for one of the Experiment.

5. Results

Here in this section we report the test accuracy for different augmentation techniques by varying the number of convolutional layers, pooling layers, data augmentation techniques and percentage of labeled and unlabeled data.

Table 5.1 shows the test accuracy for baseline Convolutional Neural Network and by applying MixUp and CutMix technique individually.

Fig 5.2 shows the test accuracy by applying both the data augmentation techniques MixUp and CutMix. From theory we learnt that using semi supervised data the accuracy increased we have found that it decreases as the amount of unlabeled data increases.

Number of epochs	Baseline CNN Accuracy	CutMix Accuracy	MixUp Accuracy
50	81.19%	77.21%	81.24%
100	82.03%	80.13%	83.59%
150	82.86%	81.38%	84.15%
200	83.59%	81.44%	85.21%

Table 5.1 Test accuracy for varying epochs and different data augmentation techniques.

Percentage of Unlabeled data	Percentage of Labeled data	Test Accuracy
20	80	71.30%
40	60	68.56%
50	50	66.50%
60	40	61.87%
80	20	58.08%

Table 5.2 Test accuracy for epochs = 50 and varying percentages of labeled and unlabeled data

Table 5.3 shows test accuracy by varying the number of epochs for 100 percent labeled data and by applying both the data augmentation techniques.

Number of epochs	Test Accuracy
50	73.72%
100	74.87%
150	75.13%
200	75.50%

Table 5.3 Test accuracy for 100% labeled data and varying epochs

Table 5.4 shows the test accuracy for baseline Convolutional Neural Network by varying the number of convolutional and pooling layers.

Number of Conv Layers	Number of Pooling Layers	Test Accuracy
2	2	65.24%
4	2	60.88%
6	2	73.72%

Table 5.4 Test accuracy for epochs = 50, 2 pooling layers and varying number of convolution layers

6. Conclusion

We have explained how the Convolutional Neural Network is designed and works for semi-supervised learning. The data augmentation techniques CutMix and MixUp along with entropy minimization technique are detailed and the implementation of the above techniques applying each technique individually and applying all the semi-supervised learning techniques to train the CIFAR 10 dataset. And experimentation on these semi-supervised learning techniques is carried out by varying the percentage of labeled and unlabeled data. From these experiments, it can be concluded that the classification accuracy is reduced when all the semi-supervised learning techniques are applied collectively

7. Division of Work

We have chosen to test for classification accuracy, confusion matrix and plot loss function for a convolutional neural network using technique like entropy minimization and data augmentation techniques like MixUp and CutMix using CIFAR-10 dataset. Praveen and Varun worked together to implement Baseline Convolutional Neural Network and application of all the semi-supervised learning techniques in training the data and analysis of results. Amit worked on data augmentation technique - MixUp and analysis of results. I worked on implementation of CutMix and the analysis of its results.

8. Self-Peer Evaluation Table

V. Varun Rao	S. Avinash Reddy	N. Praveen Rao	Amit Singh
20	20	20	20

References

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

[2] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization," in Advances in neural information processing systems, 2005, pp. 529–536.

[3] Sumit Saha, "A Comprehensive Guide to Convolutional Neural Networks", 2018. [Online].Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[4] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," arXiv preprint arXiv:1710.09412, 2017.

[5] A. Krizhevsky, G. Hinton et al., "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep.,2009. [Online].Available: https://www.cs.toronto.edu/~kriz/cifar.html

[6] V. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and its Applications, 1971.

[7] P. Simard, Y. LeCun, J. Denker, and B. Victorri. Transformation invariance in pattern recognitiontangent distance and tangent propagation. Neural networks: tricks of the trade, 1998.