Deep Learning for Computer Vision (CS776A)
Indian Institute of Technology Kanpur
Assignment 1

Member Names: Kartick Verma, Madhav Maheshwari
Member Emails: kartickv22@iitk.ac.in, madhavm22@iitk.ac.in
Member Roll Numbers: 22111029, 22111037
Date: February 3, 2023

Objective

To implement and train a multi-layer perceptron (MLP) on the CIFAR10 dataset with data augmentation (Two hidden layers with 64 neurons each MLP model).

DataSet

CIFAR-10 Dataset

The CIFAR-10 (1) collection comprises 60,000 32x32-pixel colored photos that are categorized into 10 groups, each with 6,000 images. Out of these, 50,000 are designated for training and 10,000 for testing purposes. The test batch contains 1,000 randomly picked photos from each category, while the training set is divided into 5 batches, each consisting of 10,000 images. The distribution of images from each class among the training batches may vary, but in total, they comprise 5,000 pictures from every class. **reference/ some images**

Figure 1: CIFAR-10 Dataset

Data Augmentation

Data augmentation refers to a set of methods used to artificially expand the quantity of data by creating new data points from the existing information. This can be done through small modifications to the data or utilizing deep learning models to generate novel data points.

Data augmentation is an effective way to enhance the results and performance of machine learning models. By creating new and diverse examples from the training dataset, the model's accuracy and efficiency are improved. A well-rounded and ample dataset in a machine learning model leads to improved performance.

Image Transformation

Image transformation refers to a process of modifying an image to improve its appearance, or to change its shape, orientation, or size. Image transformations can be applied to digital images for a variety of purposes, including image enhancement, image compression, image resizing, and data augmentation in machine learning. Some common image transformations include cropping, rotation, scaling, flipping, and color adjustments like brightness, contrast, and saturation. These transformations can be performed using various algorithms, ranging from simple mathematical operations to complex computer vision techniques. The aim of image transformations is to make the image more suitable for a specific task or to make it easier to process or analyze.

Image transformation methods we have used in this assignment are described below:

Image Enhancement

Image enhancement is the process of adjusting the pixel values in an image to improve its visual quality. The goal is to increase the contrast and make the image more vivid and appealing.
To enhance an image, for each pixel 'i' in the image, the following formula is used:

$$i' = \frac{(i - min)}{(max - min)} * 255 \tag{1}$$

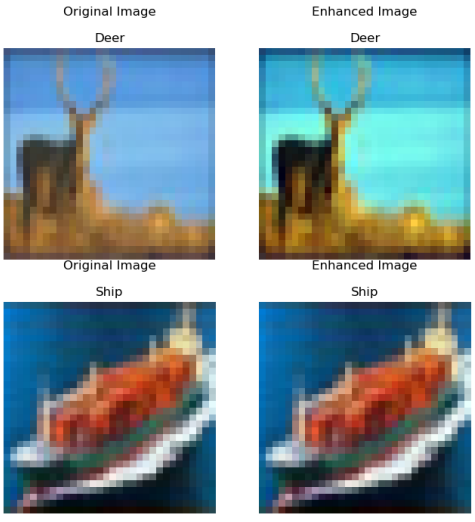


Figure 2: Example of Image Enhacement

where 'min' and 'max' are the minimum and maximum pixel values in the image. The effect of this transformation is to stretch the range of the pixel values, making the image appear brighter or darker depending on the values of 'min' and 'max'.

Posterization of Image

Posterization is a process of reducing the number of colors in an image to a smaller set of colors, creating a stylized effect. The process is commonly used in digital art to create a distinct visual style. To posterize an image, the following steps are taken:

1. Select a desired minimum and maximum pixel value in the range of [0-255].
2. Calculate the range 'r' by subtracting the selected minimum and maximum pixel value.
3. Get a divider for the colors using $divider = \frac{255}{r}$.
4. Get the level of colors by $i = \frac{i}{divider}$.
5. Finally, apply the color palette on pixel by $i = i + min$.

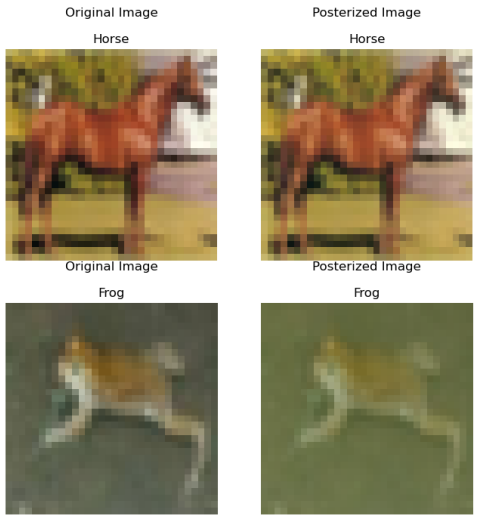


Figure 3: Example of Image Posterization

The result of this transformation is an image with a limited color palette, creating a stylized and simplified visual appearance.

Random Rotate [-180°, 180°]

Random rotation is a process of randomly rotating an image by a certain degree in either direction (2). The degree of rotation is randomly selected from the range of [-180°, 180°]. The goal of this transformation is to increase the diversity of the training data and reduce overfitting in a machine learning model. By randomly rotating the image, the model will see a wider range of variations in the input data and generalize better.

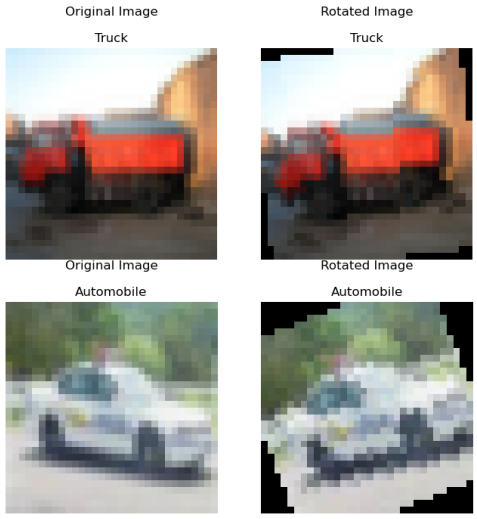


Figure 4: Example of Image Rotation

Contrast & Horizontal flipping

Contrast and horizontal flipping is a two-step transformation process where the contrast of an image is first changed and then the image is flipped horizontally.

Changing contrast of an image

The contrast of an image can be changed by multiplying each pixel value in the image by a certain factor α. The factor α is randomly selected from the range (0.5, 2.0). The formula used to change the contrast of an image is:

$$x'(i,j,c) = \alpha * (x(i,j,c) - 128) + 128 \tag{2}$$

Horizontal flipping

Horizontal flipping is the process of flipping an image along the vertical axis. The image is flipped with a probability of 0.5, meaning that half of the images will be flipped and half will remain unchanged.

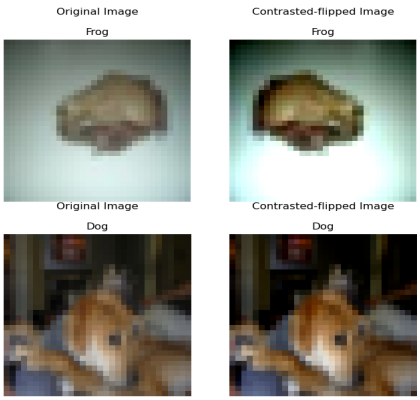


Figure 5: Example of Image Contrast and Horizontal Flip

The result of these two transformations is a diverse set of images with different contrasts and orientations, allowing a machine learning model to better generalize to new data. **images and reference**

Augmented Dataset

In order to improve the accuracy of a machine learning model, it is common practice to augment the training set with additional data. This helps to prevent overfitting, where the model performs well on the training data but fails to generalize to new data. In the previous part of our assignment, we implemented four image transformation functions - image enhancement, posterization of image, random rotate and contrast horizontal flipping.

In this part, we will use these transformation functions to create an augmented training set. The process involves randomly selecting one of the four transformation functions for each image in the original training set and applying it to that image. After transforming each image, we will then combine the transformed images with the original training set to create the augmented training set.

The **image enhancement** function increases the contrast of an image by stretching the range of the pixel values. This makes the image appear brighter or darker depending on the values of the minimum and maximum pixel values in the image.

The **posterization** of image function reduces the number of colors in an image to a smaller set of colors, creating a stylized effect.

The random **rotate** function randomly rotates an image by a certain degree in either direction. The degree of rotation is randomly selected from the range of [-180°, 180°].

The **contrast & horizontal flipping** function is a two-step transformation process. First, the contrast of the image is changed by multiplying each pixel value in the image by a certain factor α, which is randomly selected from the range (0.5, 2.0).

By using these transformation functions, we can create an augmented training set with twice the number of examples as the original unaugmented training set. This diversity in the training data will help to increase the accuracy of the machine learning model and prevent overfitting.

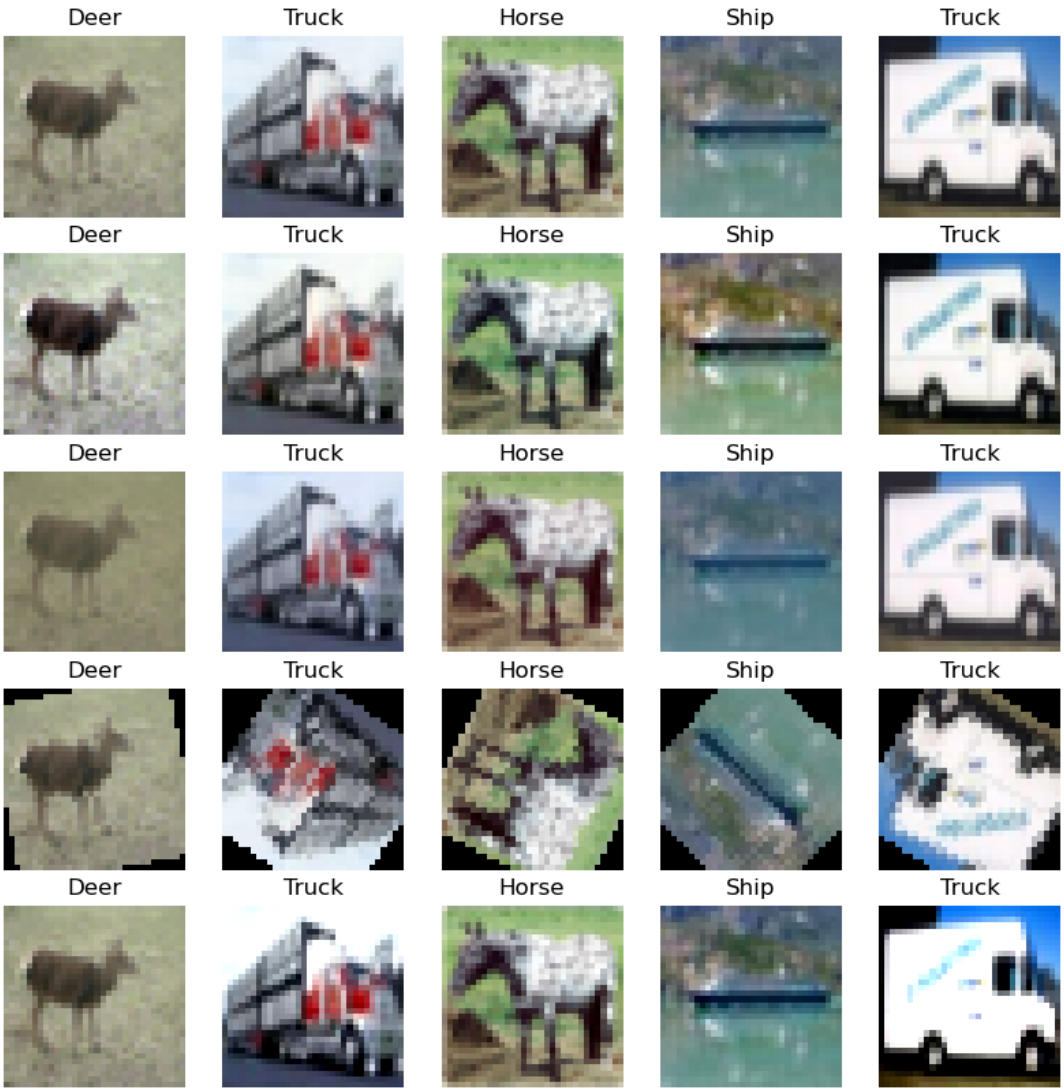


Figure 6: Augmented DataSet with all four Image Transformation

Feature Extraction

Feature extraction is the process of transforming an input data set into a set of features, or characteristics that are representative of the input data set. The goal of feature extraction is to reduce the complexity of the data set and to preserve the most important information in the data for further analysis. In the context of image processing, feature extraction involves transforming an image into a set of numerical values that represent the key characteristics of the image, such as color, texture, and shape. These features can then be used to classify or cluster images, perform image recognition, or train machine learning models. Feature extraction is an important step in many computer vision and image processing tasks, as it enables the effective analysis and manipulation of images.

Generating feature vector

Images are passed to the feature extraction function to generate feature vectors. The feature extraction function processes each image, extracts the relevant features and returns

the feature vector for that image. These feature vectors can then be used for training machine learning models.
The process of creating feature vectors from the CIFAR images involves the following steps:

Resizing the images: The first step is to resize the images from their original size of (3×32×32) to (3×224×224). This is done using image processing library CV2 (6). This library have inbuilt functions for image resizing that can be used to change the size of the images.

Feature Extraction: The next step is to extract features from the resized images. For this, the feature extraction function of BBResNet18 (7) class is used. This function expects each image to be in the form of a numpy array of dtype numpy.float32 and shape [None, 3, 224, 224], where None represents a variable size. The function then returns a numpy array of dtype numpy.float32 and shape [None, 512].

Multi-layer Perceptron (MLP)

Model Overview

We have developed a Multi-layer Perceptron (MLP) with an input layer, two hidden layers, and an output layer. The input layer has 512 neurons, which represents the vector of input features. The two hidden layers each have 64 neurons and use the ReLU activation function on each neurons. The output layer has 10 neurons and uses the softmax activation function, a standard activation function for multi-class classification problems. The loss function used in our model is Cross Entropy, which measures the dissimilarity between the predicted probabilities and the true label distribution.

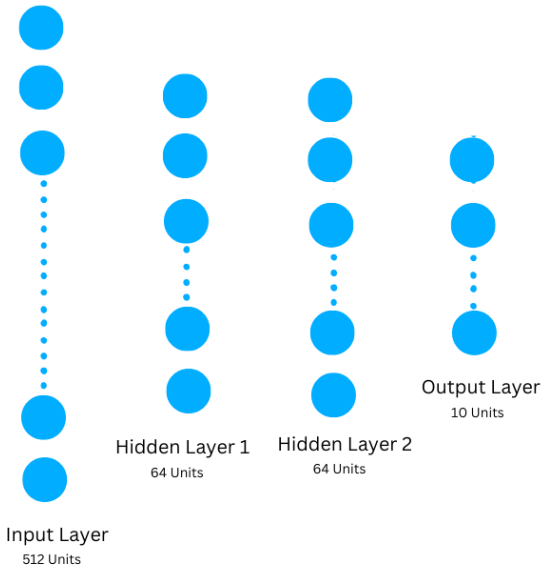


Figure 7: Network Architecture

Weight Dimensions and Initialization

We have initialized the weights and biases of our model using the **np.random.random** and **np.zeros** functions. The input of our model consists of a 100000 x 512 dimensional vector. The shape of the weights and bias at each layer are indicated in the table given below:

Layer	Weights Shape	Bias Shape
Hidden Layer 1	(512,64)	(1,64)
Hidden Layer 2	(64,64)	(1,64)
Output Layer	(64,10)	(1,10)

Table 1: Dimensions of weights and bias at each layer.

Activation Functions

Softmax

The softmax activation function is used in the final layer of a neural network to obtain a probability distribution over multiple classes. It maps the raw activations to a set of values that sum to 1, representing the predicted probabilities of each class.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3}$$

Relu

The ReLU activation function is used in neural networks to introduce non-linearity into the model. It replaces negative values in the input with 0 and passes the positive values unchanged, creating an activation threshold for the neurons in the network. This activation function helps to alleviate the vanishing gradient problem in deep networks and allows the model to learn more complex representations of the data.

$$Relu(x) = max(0, x) \tag{4}$$

Loss Function

The cross-entropy loss is calculated by comparing the predicted class probabilities with the one-hot encoded representation of the true class label and summing the differences over all classes. The goal is to minimize this loss during training, leading to a model that can accurately predict the class labels.

$$Loss = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(\hat{y_i}) \tag{5}$$

Forward Propogation

The Forward Propagation in our MLP follows the steps:

1. Taking the input data from the first layer

2. Calculating the weighted sum of inputs for each node in the subsequent layers
3. Passing the result to the activation layer to introduce non-linearity into the network
4. Repeating these steps until the output is generated from the last layer.

Calculation

Let X_0 be the input vector and Z_i, A_i, W_i, B_i be the input, output of the activation function, weights, and bias respectively at the ith layer.

At Hidden Layer 1:

$$Z_1 = X_0 \cdot W_1 + B_1$$
$$A_1 = relu(Z_1)$$

At Hidden Layer 1:

$$Z_2 = A_1 \cdot W_2 + B_2$$
$$A_2 = relu(Z_2)$$

At Output Layer :

$$Z_3 = A_2 \cdot W_3 + B_3$$
$$A_3 = softmax(Z_3)$$

Backward Propogation(3)

Derived derivative of Cross-Entropy Loss with Softmax to complete the Backpropagation.

$$\frac{dL}{dZ_3^i} = \frac{d}{dZ_3^i}\left[-\sum_{k=1}^{c} Y^k \log\left(A_3^k\right)\right] = -\sum_{k=1}^{C} Y^k \frac{d\left(\log\left(A_3^k\right)\right)}{dZ_3^i}$$
$$= -\sum_{k=1}^{C} Y^k \frac{d\left(\log\left(A_3^k\right)\right)}{dA_3^k} \cdot \frac{dA_3^k}{dZ_3^i}$$
$$= -\sum_{k=1}^{C} \frac{Y^k}{A_3^k} \cdot \frac{dA_3^k}{dZ_3^i}$$
$$= -\left[\frac{Y^i}{A_3^i} \cdot \frac{dA_3^i}{dZ_3^i} + \sum_{k=1,k\neq i}^{C} \frac{Y^k}{A_3^k}\frac{dA_3^k}{dZ_3^i}\right]$$
$$= -\frac{Y^i}{A_3^i} \cdot A_3^i\left(1 - A_3^i\right) - \sum_{k=1,k\neq i}^{C} \frac{Y^k}{A_3^k} \cdot \left(A_3^k A_3^i\right)$$
$$= -Y^i + Y^i A_3^i + \sum_{k=1,k\neq i}^{C} Y^k A_3^i$$
$$= A_3^i\left(Y^i + \sum_{k=1,k\neq i}^{C} Y^k\right) - Y^i = A_3^i \cdot \sum_{k=1}^{C} Y^k - Y^i$$
$$= A_3^i \cdot 1 - Y^i, \text{since} \sum_{k=1}^{C} Y^k = 1$$
$$= A_3^i - Y^i = A_3 - Y$$

The above derivation is useful to find the gradient of loss with respect to weight and biases.

Calculation between Hidden Layer 2 and Output Layer

$$\frac{\partial L}{\partial Z_3} = A_3 - Y$$
$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial W_3} = \frac{1}{m} np \cdot \text{dot}\left(A_2^T, \frac{\partial L}{\partial Z_3}\right)$$
$$\frac{\partial L}{\partial B_3} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial B_3} = \frac{1}{m} np \cdot \text{sum}\left(\frac{\partial L}{\partial Z_3},\ \text{axis } = 0\right)$$

Calculation between Hidden Layer 1 and Hidden Layer 2

Now Calculate gradient g_1',

$$g_1' = \frac{\partial A_2}{\partial Z_2} = 1 \text{ if } A_2^i > 0 \text{ otherwise } 0$$
$$\frac{\partial L}{\partial Z_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2} = np \cdot \text{dot}\left(\frac{\partial L}{\partial Z_3}, W_3^T\right) * g_1'$$
$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial W_2} = \frac{1}{m} np \cdot \text{dot}\left(A_1^T, \frac{\partial L}{\partial Z_2}\right)$$
$$\frac{\partial L}{\partial B_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial B_2} = \frac{1}{m} \text{np. sum}\left(\frac{\partial L}{\partial Z_2}, axis = 0\right)$$

Calculation between Input Layer and Hidden Layer 1

Now Calculate gradient g_0',

$$g_0' = \frac{\partial A_1}{\partial Z_1} = 1 \text{ if } A_1^i > 0 \text{ otherwise } 0$$
$$\frac{\partial L}{\partial Z_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1} = np \cdot \text{dot}\left(\frac{\partial L}{\partial Z_2}, W_2^T\right) * g_0'$$
$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial W_1} = \frac{1}{m} np \cdot \text{dot}\left(X_0^T, \frac{\partial L}{\partial Z_1}\right)$$
$$\frac{\partial L}{\partial B_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial B_1} = \frac{1}{m} \text{np. sum}\left(\frac{\partial L}{\partial Z_1}, axis = 0\right)$$

Let lr be the learning rate.

Updation Of weights and bias at Output Layer

$$W_{3new} = W_{3old} - lr * \frac{\partial L}{\partial W_3}$$
$$B_{3new} = B_{3old} - lr * \frac{\partial L}{\partial B_3}$$

Updation Of weights and bias at Hidden Layer 2

$$W_{2new} = W_{2old} - lr * \frac{\partial L}{\partial W_2}$$
$$B_{2new} = B_{2old} - lr * \frac{\partial L}{\partial B_2}$$

Updation Of weights and bias at Hidden Layer 1

$$W_{1new} = W_{1old} - lr * \frac{\partial L}{\partial W_1}$$

$$B_{1new} = B_{1old} - lr * \frac{\partial L}{\partial B_1}$$

Model Training and Hyper-parameters

After experimenting with different hyperparameters, we determined that the below set of hyperparameters was the optimal configuration for our model. Both the original and augmented datasets were trained using an MLP model with the following identical hyperparameters:

1. The training was conducted for 40 epochs.
2. The batch size was set to 32.
3. Mini-batch gradient descent was employed to update the weights and bias.
4. The learning rate was set to 0.01.
5. Cross entropy loss was used to determine the loss calculation.

Result and Conclusion

After testing the accuracy of various classifiers, such as Multi-Layer Perceptron (MLP), Support Vector Machine (SVM) (4), Logistic Regression(4), K Nearest Neighbors(4), and Decision Tree Classifier(4), we plotted a graph to compare their performance on both the original and test datasets. We have concluded the following points:

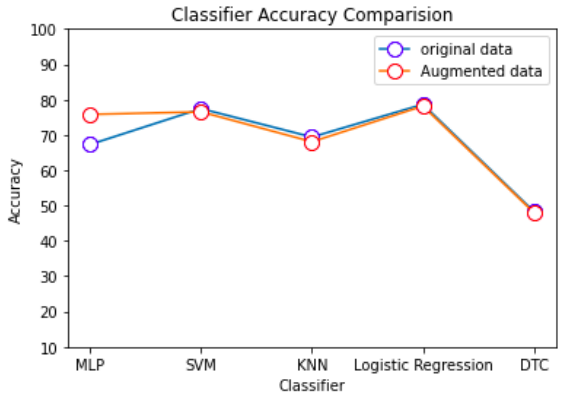
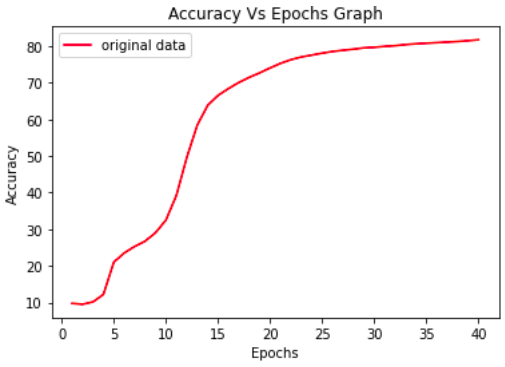
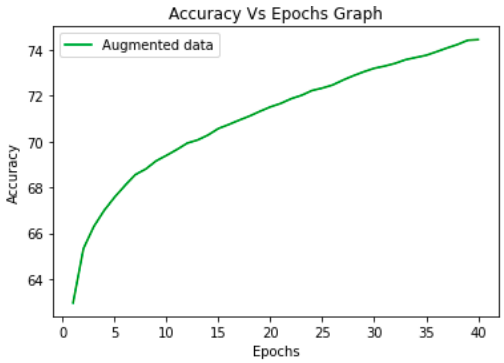


Figure 8: Accuracy Vs Classifier

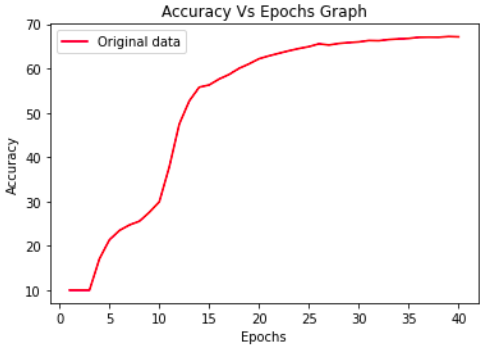
1. Logistic Regression performed well on both the original and augmented datasets.
2. All other classifiers performed similarly on original and augmented datasets, with the exception of MLP, which performed better on the augmented data.
3. In the MLP Model data augmentation plays a crucial role in improving the accuracy of a model, especially with limited original data.(5)



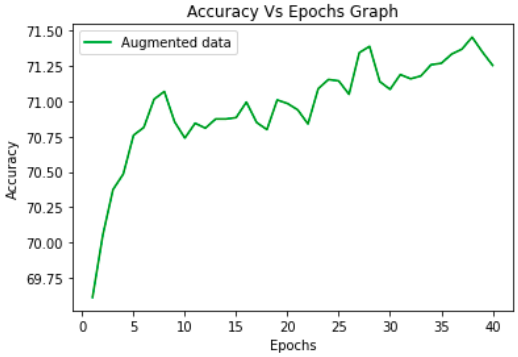
(a) Accuracy at the time of training



(b) Accuracy at the time of training



(a) Accuracy at the time of testing



(b) Accuracy at the time of testing

4. The choice of classifier and data preprocessing significantly impact the accuracy of a model.

References

References

[1] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton *CIFAR-10 Dataset*, https://www.cs.toronto.edu/~kriz/cifar.html

[2] Gautam Agrawal *Rotating Image By Any Angle(Shear Transformation)*, https://gautamnagrawal.medium.com/rotating-image-by-any-angle-shear-transformation-using-only-nu

[3] Abhisek Jana *Understanding and implementing Neural Network with SoftMax in Python from scratch)*, https://www.adeveloperdiary.com/data-science/deep-learning/neural-network-with-softmax-in-python/

[4] Scikit learn *scikit-learn Machine Learning in Python)*, https://scikit-learn.org/stable/

[5] Matplotlib *(Matplotlib: Visualization with Python)*, https://matplotlib.org/

[6] OpenCV *(CV2: Open Source Computer Vision)*, https://docs.opencv.org/4.x/

[7] pytorch, https://pytorch.org/get-started/locally/