DEEP LEARNING FOR COMPUTER VISION

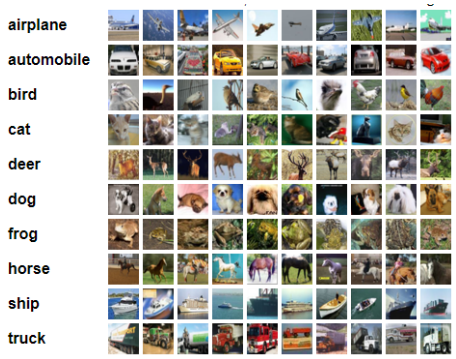
ASSIGNMENT NO 1

Member Names: Avnish Tripathi, Divyesh Tripathi
Member Emails: avnisht22@iitk.ac.in, divyeshdt22@iitk.ac.in
Member Roll Numbers: 22111014, 22111020
Date: February 3, 2023

MLP Model for CIFAR10 Classification

1 CIFAR-10 Dataset

A portion of the 80 million tiny picture dataset is the CIFAR-10 dataset. It has 60,000 32x32 RGB-colored images divided into 10 classes. 10,000 test photos are kept in one batch, while 50,000 training images are stored in 5 batches. Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton collected it. The distribution from which the training and test photos are selected is the same.



2 Data Augmentation

Data augmentation refers to techniques that are used to add slightly edited versions of existing data or create synthetic data by using existing data, thereby

increasing the actual amount of data available.

It is used to make your model's training sets more diverse by using random, yet realistic data transformation, such as flipping or rotating images.

By creating fresh and distinctive instances for training datasets, data augmentation helps machine learning models perform better and produce better results. A machine learning model performs better and is more accurate if the dataset is large and sufficient. By producing variables that the model could encounter in the real world, data augmentation approaches allow machine learning models to be more resilient.

Below are a few data augmentation techniques we employed for this assignment.

1. **Image Enhancement :**
   Here for each pixel i of particular channel in the image x we have used this formula: ((imin)/(maxmin))x255 Where, min and max are the minimum and maximum pixel values of that channel in the image x.

2. **Posterization of Image**
   We are reducing the number of colours in an image to create a work of visual art.
   First we selecting a minimum and maximum pixel value in the range of [0-255] which are min=3 and max=225.

   • Now, for each pixel i in image x:

   i) Calculate the range r by subtracting selected minimum and maximum pixel value
   ii) Get a divider for the colors using divider = 255/r
   iii) Get the level of colors by i = i/divider
   iv) Finally, apply the color palette on pixel by i = i + min

3. **Random Rotate**

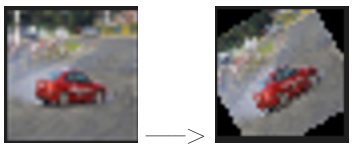
We begin by importing the three modules/libraries numpy, cv2, and math. The opencv-python, or cv2 package, is only utilised in this case for the purpose of reading and displaying images. We will first use the math.radians() function to translate the degrees into radians as we implement the function. The output image was then specified as a numpy array with all values set to 0. It was then declared to be the same size as the original (or input) image.

Since the input and output image sizes are same, their centres will also be identical. The centre point of the output image is determined in the following phase.

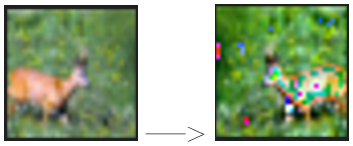
The intensity values for the output image must now be filled from the new coordinates derived using the following equation in order to rotate correctly without any gaps.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

In this we have selected an angle randomly for rotation of an image and performed rotation operation by selected angle on that image.



4. **Contrast Horizontal flipping of Image**
   First we have randomly selected in range (0.5, 2.0) and contrasted that image with the factor of and then flip the image horizontally with a probability of 0.5



Following data augmentation on the original dataset, the enhanced dataset has a total of 100,000 samples.

3 Feature Extraction

Every computer vision assignment must start with feature extraction. The process of identifying an image's distinctive qualities is crucial for analysis and classification.

We apply the pre-trained CNN architecture of ResNet 18 on the ImageNet Dataset in this project to extract features. It receives batches of dimension (3, 224, 224) image input and outputs data characteristics of dimension (512,1).

This method uses a deep neural network with the BBResNet18 architecture to extract features from a set of photos. To effectively manage enormous amounts of data, the code processes the data in batches. First, the resize function uses the OpenCV library's cv2.resize function to resize the photos in the batch to (224, 224). After converting the values in each batch to the range [0, 1], the get features function calls the feature extraction method on the BBResNet18 object to extract features from each batch. Each batch's extracted features are added to a list before being returned as the final product.

4 Network Architecture

A 512-dimensional feature vector is produced after the image has been run through the ResNet18 feature extractor. As a result, there will be 512 nodes in the input layers. Additionally, we utilise the CIFAR10 dataset, which has ten classes, therefore the result will have 10 nodes. There are 64 nodes in the each hidden layer.And there are 2 hidden layers used in our neural network.

Figure 2: Network Architecture

4.1 Weight dimension and initialization

In order to initialise all of the weights with extremely small integers and the biases with zeros, the weights are initialised randomly by using np.random.randn and dividing by 1000 (i.e. 1e3). To solve the vanishing gradient problem, we divided the weights by this value. The following are the dimensions of weights and biases:

1. W1 = $(512, 64)$
2. B1 = $(1, 64)$
3. W2 = $(64, 64)$
4. B2 = $(1, 64)$
5. W3 = $(64, 10)$
6. B3 = $(1, 10)$

5 Forward Propagation

In Forward Propagation the input data is fed into the network in a forward direction. Each hidden layer receives the data input, processes it in accordance with the activation function, and then transfers it to the following layer.Each layer's node calculates the weighted total of their input before sending it to the activation layer, which adds nonlinearity to the network.

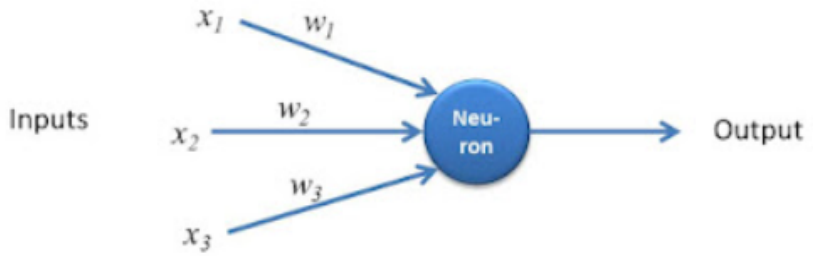


Figure 3: Working of single node in MLP

Let A_0 be the network's input (512, 1), Z_i be the i^{th} layer's output, and A_i be the result of applying an activation function to that layer's output.

So the forward propagation for model looks like this:

$$
\begin{aligned}
Z_1 &= A_0 * W_1 + B_1 \\
A_1 &= \mathrm{Re}\,LU\,(Z_1) \\
Z_2 &= A_1 * W_2 + B_2 \\
A_2 &= \mathrm{Re}\,LU\,(Z_2) \\
Z_3 &= A_2 * W_3 + B_3 \\
O &= \mathrm{Softmax}\,(Z_3)
\end{aligned}
$$

6 Activation Function

Because they introduce non-linearity into the model and aid the neural network in learning more complicated data properties, activation functions are essential for neural networks. activate function determine For the hidden layer, we use ReLU activation, and for the output layers, we use softmax activation.

6.1 Softmax

When dealing with multiclass classification issues, the last layer of the network employs the softmax activation function. It changes every value between 0 and 1, which is equivalent to a probability for that class.

$$
\mathrm{softam}\,x\,(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}
$$

6.2 ReLU - Rectified Linear Unit

The most popular activation function in deep learning is ReLU. When the input is zero, it behaves as a dead neuron; otherwise, it produces the same result. In order to achieve better results and address the issue of disappearing gradients, it is also utilised in hidden layers.

$$
\mathrm{relu}(x) = \max(0, x)
$$

7 Loss Function

How far an estimated value is from its real value is determined by the loss function. It is a technique for assessing how well our model performed. In our model, we employ cross-entropy loss. It evaluates the effectiveness of categorization models, the results of which are probabilities.

$$
\mathrm{Loss}_{CE} = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log\left(\hat{y}_i\right)
$$

8 Back Propagation

The artificial neural network uses backpropagation, a learning algorithm, to increase prediction accuracy. It is a method used in mathematics to determine the gradient of the loss in relation to weights and biases.

Since the second layer's output is the model's prediction, $A_2 = \hat{Y}$ Therefore...

$$\text{Loss}_{CE} = -\frac{1}{m}\sum_{i=1}^{m} y^i \cdot \log\left(A_2^i\right)$$

The Softmax activation gradient is provided as

$$\frac{\partial a^i}{\partial z^j} = \begin{cases} -a^i a^j & \text{if } i \neq j \\ a^i\left(1 - a^i\right) & \text{if } i = j \end{cases}$$

8.1 Derivation of Gradient

The softmax activation derivative has been calculated as follows:

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

gradient(ReLU) = 1 if the input is +ve ,else 0

Following is a derivation of the gradient of loss with regard to the weights and biases of the network layers:

The gradients between hidden layer 2 and output layer

$$
\begin{aligned}
dZ_3 &= \frac{\partial L}{\partial Z_3} = dZ_3 = A_3 - Y \\
dW_3 &= \frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial W_3} = \frac{1}{m} A_2^T dZ_3 \\
dB_3 &= \frac{\partial L}{\partial B_3} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial B_3} = \frac{1}{m} np \cdot \text{sum}\left(dZ_3,\ \text{axis}\ = 0\right) \\
g_1' &= \frac{\partial A_2}{\partial Z_2} = 1 \text{ if } A_2^i > 0 \text{ otherwise } 0
\end{aligned}
$$

The gradients between hidden layer 1 and hidden layer 2.

$$
\begin{aligned}
dZ_2 &= \frac{\partial L}{\partial Z_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2} = dZ_3 W_3^T * g_1' \\
dW_2 &= \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial W_2} = \frac{1}{m} A_1^T dZ_2 \\
dB_2 &= \frac{\partial L}{\partial B_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial B_2} = \frac{1}{m} \text{np. sum}\left(dZ_2, axis = 0\right) \\
g_0' &= \frac{\partial A_1}{\partial Z_1} = 1 \text{ if } A_1^i > 0 \text{ otherwise } 0
\end{aligned}
$$

The gradients between hidden layer 1 and input layer.

$$
\begin{aligned}
dZ_1 &= \frac{\partial L}{\partial Z_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1} = dZ_2 W_2^T * g_0 \\
dW_1 &= \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial W_1} = \frac{1}{m} x_0^T dZ_1 \\
dB_1 &= \frac{\partial L}{\partial B_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial B_1} = \frac{1}{m} \text{np. sum}\left(dZ_1, axis = 0\right)
\end{aligned}
$$

The gradient of loss with respect to weight and biases can be determined using the preceding derivation as follows:

$$dZ_2 = \frac{\partial L}{\partial Z_2} = dZ_2 = A_2 - Y$$
$$dW_2 = \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2} = \frac{1}{m}A_1^T dZ_2$$
$$dB_2 = \frac{\partial L}{\partial B_2} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial B_2} = \frac{1}{m}np \cdot \text{sum}\,(dZ_2,\ \text{axis}\ = 0)$$
$$g_1' = \frac{\partial A_1}{\partial Z_1} = 1 \text{ if } A_1^i > 0 \text{ otherwise } 0$$
$$dZ_1 = \frac{\partial L}{\partial Z_1} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1} = dZ_2 W_2^T * g_1'$$
$$dW_1 = \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial W_1} = \frac{1}{m}A_0^T dZ_1$$
$$dB_1 = \frac{\partial L}{\partial B_1} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial B_1} = \frac{1}{m}\,\text{np.sum}\,(dZ_1, axis = 0)$$

9 Mini Batch Gradient Descent

Using Batch Gradient Descent, curves can be made more supple. When the dataset is large, SGD can be utilised. Batch Gradient Descent quickly approaches minima. In case of bigger datasets, SGD converges quicker. But we are unable to use the vectorized implementation on SGD because we only use one example at a time. The computations may get slower as a result. A combination of Batch Gradient Descent and SGD is used to solve this issue.

We don't use the entire dataset all at once or one example at a time. We use a batch of training examples with a fixed number that is smaller than the actual dataset and refer to it as a mini-batch. By doing this, we are able to benefit from both of the earlier variants.

$$w3 = w3 - \ \text{learningrate}\ * dw3$$
$$b3 = b3 - \ \text{learningrate}\ * db3$$
$$w2 = w2 - \ \text{learningrate}\ * dw2$$
$$b2 = b2 - \ \text{learningrate}\ * db2$$
$$w1 = w1 - \ \text{learningrate}\ * dw1$$
$$b1 = b1 - \ \text{learningrate}\ * db1$$

10 Model Training and Hyper-parameters

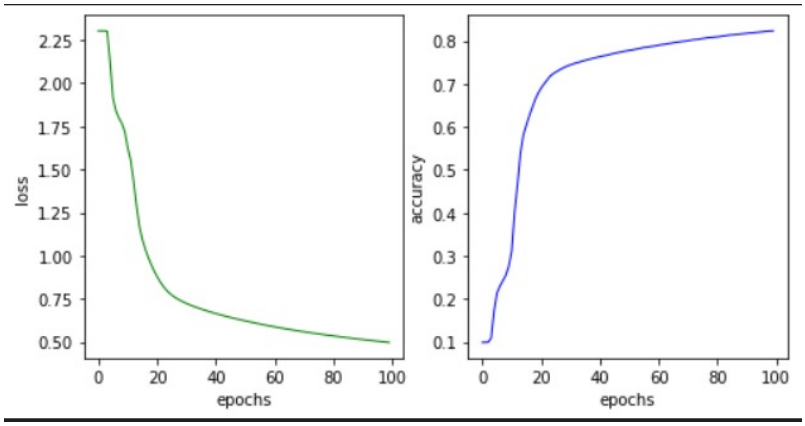
In the deep learning model lifecycle, the model training phase is where we attempt to fit the best weights and bias to a deep learning algorithm in order to minimise a loss function over the prediction range. Building the best mathematical representation of the relationship between data features and a target label (in supervised learning) or between the features themselves is the goal of model

training (unsupervised learning). Since they specify how to optimise the deep learning algorithms, loss functions are a crucial component of model training. Different types of loss functions are used depending on the aim, type of data, and technique.

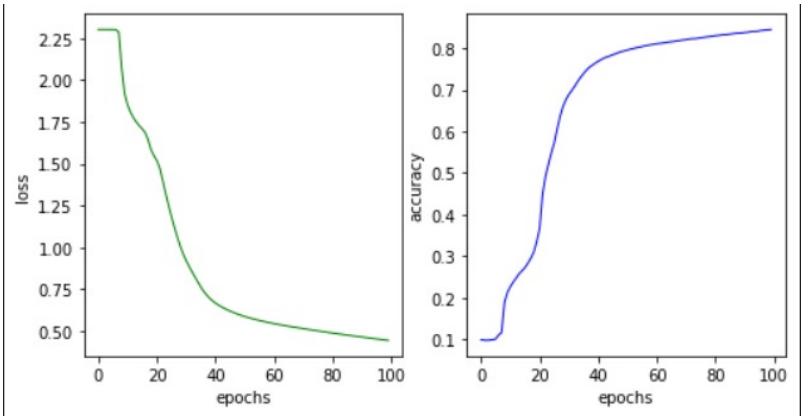
Two models are trained for this assignment, one on the base dataset and the other on the enhanced dataset. The following hyper-parameters are same for both model training scenarios:

1. To enable the production of that result once more, all weights are initialised to = 0.
2. Number of epochs = 100.
3. Batch size = 64.
4. Mini-batch gradient descent is used for updating the weights and bias.
5. Initial learning rate is = 0.03, and after every 10 epochs, it is reduced by a factor of = 0.9.
6. Loss is calculated using cross entropy loss.

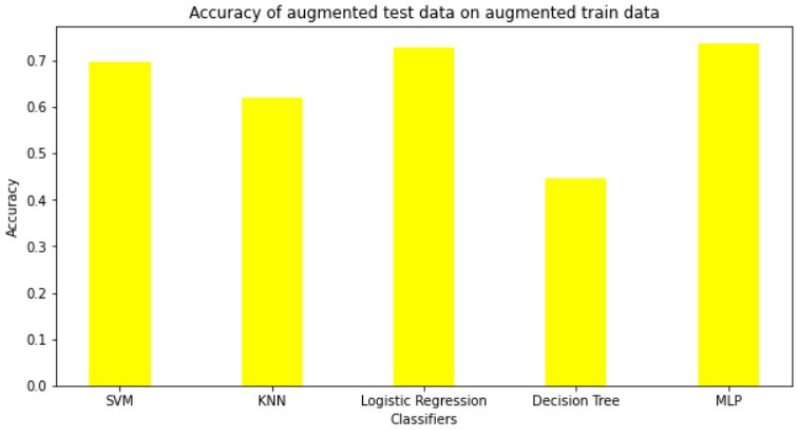
11 Result



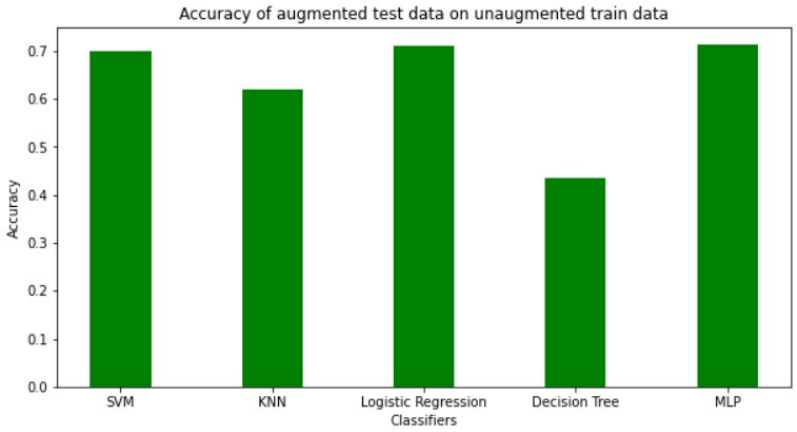
(a) Training Loss and Accuracy for augmented model and original model.



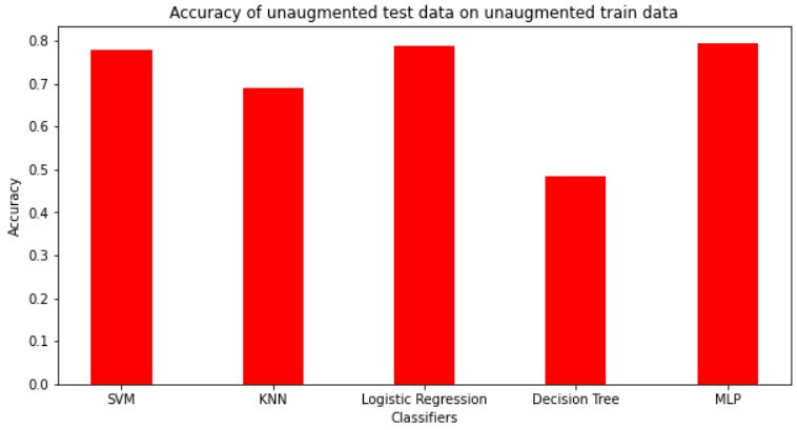
(b) Test Loss and Accuracy for augmented and original model.

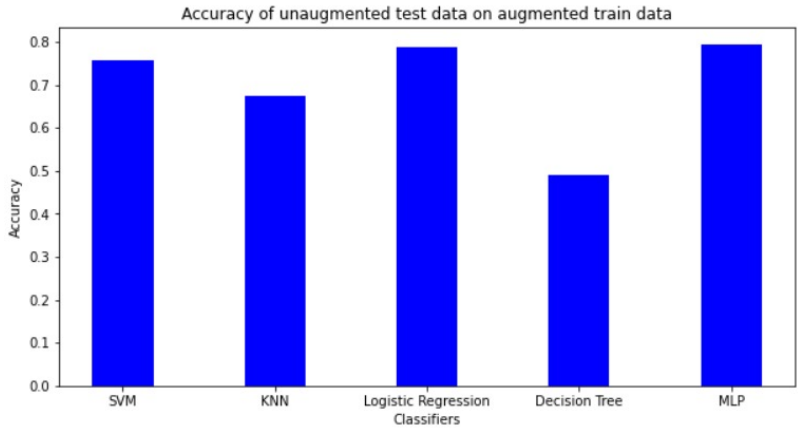


(c) Accuracy of augmented test data on augmented train data.



(d) Accuracy of augmented test data on unaugmented train data.





Model	Training Accuracy	Training Loss	Test Accuracy	Test Loss
Original Dataset	84.52%	0.4438	79.5%	0.511
Augmented Dataset	82.29%	0.4975	73.74%	0.509

Table 1: Comparative Study of both Classification models

12 Conclusion

Based on the available data, it can be said that the machine learning algorithms perform better on the unaugmented dataset than they do on the augmented dataset. This shows that the algorithms' accuracy suffered as a result of the augmentation procedure. With accuracy scores exceeding 0.7, the algorithms SVM, Logistic Regression, and MLP nonetheless did well on both datasets. The KNN and DT algorithms, on the other hand, significantly decreased in accuracy, with scores on the expanded dataset falling below 0.7. This analysis gives a general knowledge of the effect of data augmentation on the performance of various algorithms, while the results may vary depending on the particular situation and dataset.

13 References

[1] References for Understanding the concepts of Neural Network- towardsdatascience.com/building-a-neural-network-with-a-single-hidden-layer-using-numpy

[2] Naive Rotation of an Image

[3] The Complete Guide to Neural Network multi-class Classification from scratch

[4] The CIFAR-10 dataset https://www.cs.toronto.edu/ kriz/cifar.html

[5] What is Data Augmentation? Techniques, Benefit Examples https://research.aimultiple.com/data-augmentation/amp/

[6] How Does the Gradient Descent Algorithm Work in Machine Learning? https://www.analyticsvidhya.com/blog/2020/10/ how-does-the-gradient-descent-algorithm-work-in-machine-learning/

[7] Model Training https://c3.ai/glossary/data-science/model-training/#: :text=What %20 is %20 Model%20Training%3F, function%20over %20 the %20 prediction%20range.

[8] Neural Networks and Deep Learning https://www.coursera.org/learn/neural-networks-deep-learning?specialization=deep-learning

[9] Training with Image Data Augmentation in Keras https://stepup.ai/train_data_augmentation_keras/

[10] Neural-network-with-softmax-in-python https://www.adeveloperdiary.com/data-science/deep-learning/