DEEP LEARNING FOR COMPUTER VISION

ASSIGNMENT NO 1

Member Names: Allan Robey, Kush Shah
Member Emails: allanrobey22@iitk.ac.in, kushshah22@iitk.ac.in
Member Roll Numbers: 22111007, 22111033
Date: February 3, 2023

MLP MODEL for CIFAR10 CLASSIFICATION

1 DATASET: CIFAR-10 Dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.The dataset is divided into five training batches and one test batch, each with 10000 images. Here are the classes in the dataset, as well as 10 random images from each

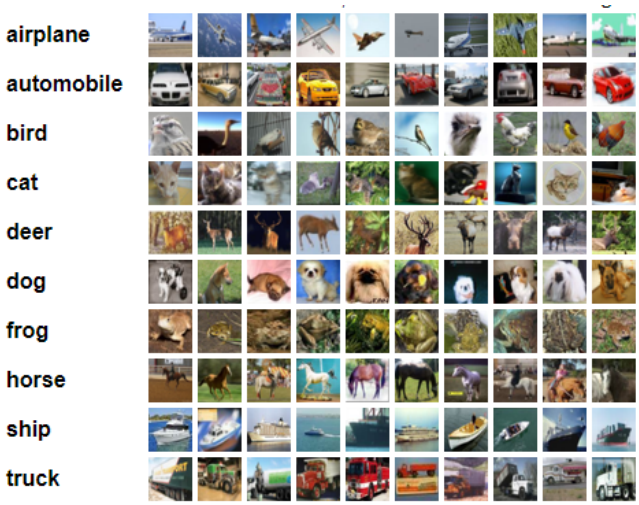

Figure 1: CIFAR-10 DATASET

2 DATA AUGMENTATION

Data augmentation uses existing data set to create modified copies of datasets, which are then used to artificially increase the training set.The purpose of data augmentation is to create new, diverse samples from the original data, which can help the deep learning model perform better.In order to increase the variety of the training set and decrease overfitting, data augmentation generates new training samples from the existing data. The model has more examples to learn from as a result of enriching the training data, which increases accuracy. In order to make the model more resistant to changes in the data distribution, data augmentation can also be employed to create examples that imitate domain shift.

The different augmentation methods that have been incorporated in this assignment are as mentioned below.

1. **Enhancement of Image** - A function has been defined that takes in an image array, which is a 1-dimensional array representing an RGB image of shape (32,32,3). The image array is converted to a numpy array and the minimum and maximum values for each color channel (red, green, blue) are computed. Then, for each color channel, the values of the channel are normalized such that the minimum value is mapped to 0 and the maximum value is mapped to 255. Finally, the image is reshaped back to its original shape (32,32,3) and returned.

Figure 2: Enhancement of Image

2. **Posterization of Image** - The function takes in an image array, which a 1-dimensional array representing an RGB image of shape (32,32,3). The image array is converted to a numpy array. The minimum and maximum values for posterization are set to 7 and 253, respectively, and a divider is calculated based on the range of these values. Then, each pixel value in the image is divided by the divider, and added with the minimum value for posterization. The resulting pixel value is then clamped to the range [0, 255]. Finally, the image is reshaped back to its original shape (32,32,3) and returned.

Figure 3: Posterization of Image

3. **Random Rotation of Image** - The function takes in an image and rotates it by a randomly generated angle between -180 and 180 degrees. The angle is first converted from degrees to radians. Then, a blank image of the same size as the input image is created to store the rotated image. The center of the image is found and used as the pivot point for rotation. The rotated image is created by iterating over the rows and columns of the blank image and finding the corresponding pixel value in the input image after rotation. The resulting image is then returned.

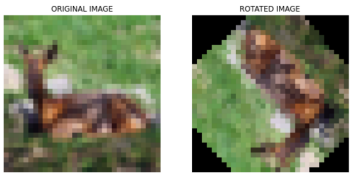

Figure 4: Rotation of an Image

4. **Contrast and Horizontal Flipping of Image** - The function takes a 3-dimensional image array (height, width, channels) as input and returns a processed image array with contrast enhancement and horizontal flipping with a 50 percent chance.The contrast enhancement part of the function modifies the image by multiplying each pixel value by a random factor between 0.5 and 2.0, effectively brightening or darkening the image. The function then clips the pixel values to the range [0, 255].The horizontal flipping part of the function has a 50 percent chance of flipping the image horizontally.The final image is returned as a 3-dimensional array in the same format as the input image.

Figure 5:Contrast and Horizontal Flipping of Image

The augmented dataset comprises of 1,00,000 images after performing the data augmentation tasks on the original dataset.

3 EXTRACTION OF FEATURES

The process of identifying and extracting significant details or features from an image that may be utilised as inputs for machine learning algorithms is known as feature extraction in deep learning and computer vision. Feature extraction seeks to simplify the data so that algorithms can discover relevant patterns and representations more quickly. The feature extractor.py file provided with the assignment has been used on the original (unaugmented) CIFAR-10 dataset and on the augmented dataset to obtain 1-dimensional input vectors.

A function has been defined that takes a batch of data as input, and returns a feature vector extracted from that data. The function processes the data in batches of 100 images at a time.The first step in the function is to determine the number of batches.

For each batch, the function extracts the corresponding portion of the data, resizes each image to (224,224,3), and transposes the image from (height, width, channels) to (channels, height, width). The resized images are then converted to float32 format, and each pixel value is divided by 255 to normalize the data.

The feature vectors are extracted using the feature-extraction method of the obj object, which is an instance of the BBResNet18 class.It returns a 512-dimensional feature vector.

4 ARCHITECTURE OF THE MULTI-LAYER PERCEPTRON

An artificial neural network called a multi-layer perceptron (MLP) has multiple layers of nodes, including an input layer, one or more hidden layers, and an output layer. The input layer receives the feature vector as input, the hidden layers process it and create intermediate representations, and the output layer generates the final prediction. In an MLP, the feedforward process involves transmitting the input data through each layer and computing each node's output based on the weights of the connections between the nodes. Up until the final layer's output is produced, this output serves as the next layer's input.

Backpropagation is the method by which the weights of connections between nodes in an MLP are updated to reduce the error between the desired output and the prediction made by the network. By calculating the gradient of the error with respect to each weight and using mini-batch gradient descent to update the weights in a way that lowers the error, we were able to do this.

The error is first calculated at the output layer, and then it is propagated backward through the hidden layers to calculate the gradients of the error with respect to each weight. The weights are updated using these gradients again and again until the inaccuracy is reduced to an acceptable level.In short, we have performed forward propagation then the loss has been calculated and then the loss has been backpropagated to update the weights and biases.

The MLP model for CIFAR-10 dataset classification has the following architecture:

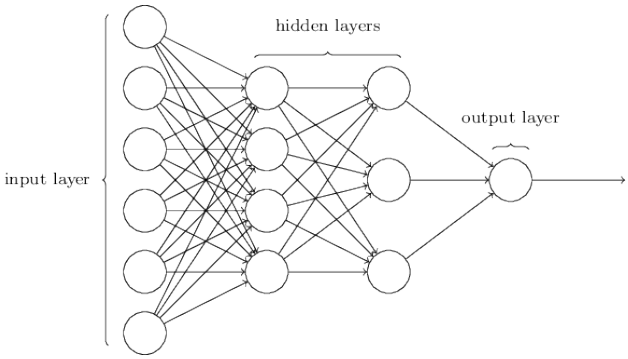

Figure 2: MLP Network Architecture

4.1 Nodes

There will be 512 nodes in the input layer as 512-dimensional feature vector is produced after the image has been run through the ResNet18 feature extractor. There are 2 hidden layers each comprising of 64 neurons. In the CIFAR10 dataset there are ten classes, thus the output layer comprises of 10 nodes.

4.2 Notations

The different notations used are as mentioned below:

1. X: It represents the input data of dimensions batch-size*512.
2. W1: It represents the weight between input layer and hidden layer 1 of dimensions 512*64
3. W2: It represents the weight between hidden layer 1 and hidden layer 2 of dimensions 64*64
4. W3: It represents the weight between hidden layer 2 and output layer of dimensions 64*10
5. B1: It represents the baises of hidden layer 1 (64)
6. B2: It represents the baises of hidden layer 2 (64)
7. B3: It represents the baises of output layer (10)
8. Z1: It represents the weighted outputs of hidden layer 1.
9. Z2: It represents the weighted outputs of hidden layer 2.
10. Z3: It represents the weighted outputs of output layer.
11. A1: It represents the activated outputs of hidden layer 1
12. A2: It represents the activated outputs of hidden layer 2
13. A3: It represents the activated outputs of the output layer

4.3 Initialization of Weight and Biases

The model has 2 hidden layers and an output layer. The parameters being initialized are the weights W1, W2, W3 and the biases B1, B2, B3. The weights and biases are randomly initialized using the numpy function "np.random.rand" and divided by the normalization constant "norm=1e3" to handle the vanishing gradient problem. The biases are initialized as zero vectors.

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 ACTIVATION FUNCTION

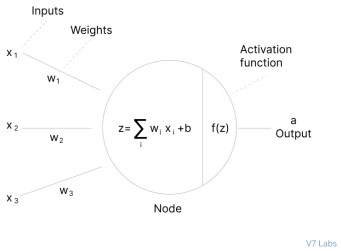

Figure 6: Activation Function

Activation functions are essential for our MLP model as it will help introduce non-linearity's into neural networks, they are helpful because this enables the neural networks to learn powerful operations such as image processing.An activation function is a function that outputs a smaller value for tiny inputs and a higher value if its inputs are greater than a threshold.If the inputs are large enough, the activation function "fires", otherwise it does nothing.To put it another way, an activation function is comparable to a barrier that verifies if an incoming value exceeds a critical threshold.We have used 2 activation functions as mentioned below in our MLP model -

5.1 ReLU - Rectified Linear Unit

In our MLP model, we have incorporated ReLU activation function in both the hidden layers.ReLU is a standard activation function which produces zero

when the input is negative input's directly if the input is positive.ReLU is computationally efficient adds non-linearity to the neural network, allowing the model to learn complex relationships between inputs and outputs.

$$\text{ReLU}(x) = \max(0, x)$$

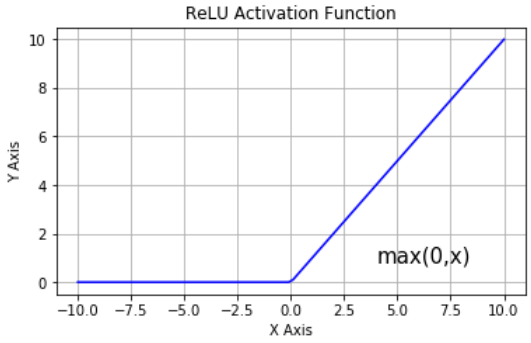

Figure 7: ReLU - Activation Function

5.2 Softmax

We have used softmax activation function in the output layer of our model as it will convert a vector of 10 real numbers into a probability distribution of 10 possible outcomes.It has been employed as it will normalise the output to a probability distribution over expected output classes.

$$\text{softmax}\,(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

6 FORWARD PROPAGATION

Forward propagation is the initial-step in the multi-layer perceptron.In forward propagation, the input data is passed through the network layer by layer until it reaches the output layer. At each layer, the inputs are transformed using the weights and biases by computing the weighted sums and an activation function is applied to it to produce the output for the next layer. The output from the final layer is the predicted result of our neural network.

The following equations have been used to perform the forward propagation in our model:

$$Z_1 = X * W_1 + B_1$$
$$A_1 = \mathrm{Re}\,LU\,(Z_1)$$
$$Z_2 = A_1 * W_2 + B_2$$
$$A_2 = \mathrm{Re}\,LU\,(Z_2)$$
$$Z_3 = A_2 * W_3 + B_3$$
$$A_3 = \mathrm{Softmax}\,(Z_2)$$

7 LOSS FUNCTION

Our model's loss function is an integral component. It indicates how closely our network's output resembles the actual true value. During training, our primary goal is to lower the loss value by adjusting the model's parameters.We have used Cross-entropy loss which is also known as the negative log-likelihood loss in our MLP model as it is commonly used loss function in supervised learning for classification problems. It measures the difference between the predicted probability distribution and the true probability distribution of the target class.

$$\mathrm{Loss}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log \hat{y}_{i,j}$$

where:N is the number of samples, C is the number of classes, $y_{i,j}$ is the ground truth label for sample i and class j $\hat{y}_{i,j}$ is the predicted probability for sample i and class j,log is the natural logarithm The sum over j runs over all classes, The sum over i runs over all samples, The negative sign is included to turn the log likelihood into a loss (maximizing the likelihood is equivalent to minimizing the negative log likelihood)

8 BACK PROPAGATION

Backpropagation is the process of computing the gradients of the loss function with respect to the weights and biases of the neural network. The gradients are used to update the weights and biases during training to minimize the loss function and improve the prediction accuracy.

In backpropagation, the gradients are calculated starting from the output layer and working backwards through the network. The gradients of the loss function with respect to the weights and biases at each layer are computed using the chain rule of differentiation. The gradients are then used to update the weights and biases using an optimization algorithm such as gradient descent as follows:

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

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

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

The gradient of Softmax activation is given 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 derivative of softmax activation has been computed as follows:

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

The gradient of ReLU function is 1 if the input is positive else zero.

The gradient of loss with respect to the weights and biases of the network layers can be derived as follows

Computing the gradients between output layer and hidden layer 2

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

Computing the gradients between hidden layer 2 and hidden layer 1

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

Computing the gradients between hidden layer 1 and input layer

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

9 MODEL TRAINING AND TESTING SET-UP

The model has been trained and tested with different combinations of hyper-parameters before finalizing the set of the model.

No of Epoch	Batch Size	Learning rate	Accuracy on Aug. Data	Accuracy on Org. Data
300	64	0.01	69.43%	75.51%
200	64	0.01	69.70%	76.04%
200	32	0.01	71.10%	77.32%
100	32	0.01	70.90%	77.20%
100	64	0.01	71.68%	77.76%
100	128	0.01	73.52%	79.71%

Table 1: Comparative Study of different hyperparameters combination

The MLP model has been trained and tested on the original dataset and on the augmented dataset. Hyper-parameters such as number of epochs, batch size and learning rate has been finalized in accordance with the above mentioned table:

1. EPOCH: The model has been trained on 100 epochs
2. BATCH SIZE: 128

3. LEARNING RATE: The initial learning rate was set to 0.01 and it has been reduced by the factor of 0.2 after every 10 epochs.

4. WEIGHTS AND BIASES: The weights have been intialized using np.random.rand() and the biases were initialized to zeros.

5. LOSS: Cross-Entropy Loss

6. GRADIENT DESCENT: Mini-batch gradient descent

7. ACCURACY: The accuracy has been calculated by matching the true labels with the one-hot encoded labels of prediction i.e number of labels correctly identified out of the total number of items

10 RESULTS AND CONCLUSION

The performance of MLP model on Original (Unaugmented) Dataset and Augmented Dataset is as show below:

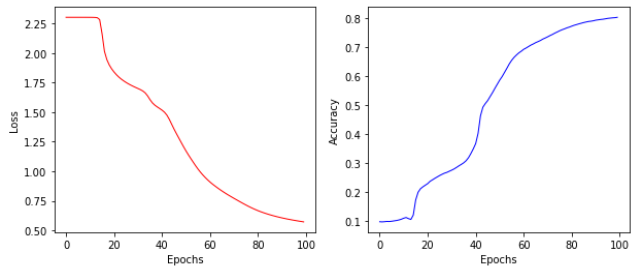

Figure 8: Epoch V/S Loss and Epoch V/S Accuracy Graph for Unaugmented Dataset

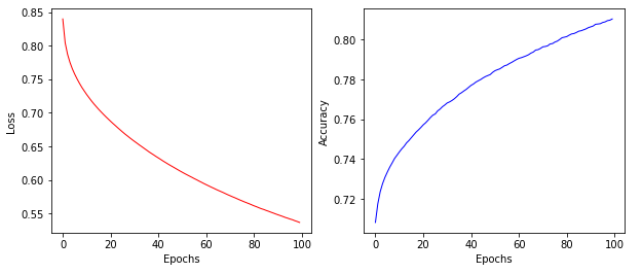

Figure 9: Epoch V/S Loss and Epoch V/S Accuracy Graph for Augmented Dataset

Dataset	Training Accuracy	Training Loss	Test Accuracy	Test Loss
Unaugmented Dataset	80.2664%	0.5658	78.67%	0.6687
Augmented Dataset	81.20%	0.5347	72.695%	0.875

Table 2: Comparison of MLP Model on Unaugmented(Original) Dataset and Augmented Dataset

The performance of different Classification Algorithms and MLP model on Original (Unaugmented) Dataset and Augmented Dataset is as show below:

Classification Algo.	Accuracy on Aug. Dataset	Accuracy on Org. Dataset
SVM	80.2664%	0.5658%
KNN	80.2664%	0.5658%
Logistic Regression	80.2664%	0.5658%
Decision Tree	80.2664%	0.5658%
MLP Model	80.2664%	0.5658%

Table 3: Comparison of MLP Model and different Classification Algorithms

Based on the data provided, it can be concluded that the performance of the machine learning algorithms on the unaugmented dataset was generally better compared to their performance on the augmented dataset. The accuracy scores of the algorithms on the unaugmented dataset range from 69.38 to 78.67 percent, with the Support Vector Machine (SVM) and the Multi-layer Perceptron (MLP) models performing the best. On the other hand, the accuracy scores on the augmented dataset range from 43.09 percent to 72.695 percent, showing a decrease in the performance of all the algorithms.

The MLP model was one of the best performers on both the unaugmented and augmented datasets, with accuracy scores of 78.67 and 72.695 percent, respectively. This suggests that the MLP model is relatively robust to the effects of data augmentation and can still produce good results even when the data has been modified.

The K-Nearest Neighbors (KNN) and Decision Tree (DT) algorithms showed a significant decrease in performance on the augmented dataset. The accuracy score for KNN dropped from 69.38 percent on the unaugmented dataset to 68.05 percent on the augmented dataset, while the score for DT decreased from 48.34 to 43.09 percent. These results indicate that these algorithms are more sensitive to the effects of data augmentation and may not be suitable for use in all situations.

In comparison, the performance of the Logistic Regression algorithm was relatively stable, with a decrease in accuracy of only 7.13 percent from the

unaugmented to the augmented dataset. The SVM algorithm also performed well on both datasets, with a drop in accuracy of only 8.56 percent.

In conclusion, the MLP model demonstrated good performance on both the unaugmented and augmented datasets, making it a strong candidate for use in situations where data augmentation may be necessary. The KNN and DT algorithms showed a decrease in performance when applied to augmented data, and therefore may not be the best choice in these situations. The Logistic Regression and SVM algorithms also performed well on both datasets, making them suitable alternatives in a variety of circumstances.

11 References

1. CIFAR-10 DATASET-https://shrishailsgajbhar.github.io/post/Image-Processing-Image-Rotation-Without-Library-Functions
2. Naive Rotation of an Image-https://shrishailsgajbhar.github.io/post/Image-Processing-Image-Rotation-Without-Library-Functions
3. The Complete Guide to Neural Network multi-class Classification from scratch - https://towardsdatascience.com/the-complete-guide-to-neural-networks-multinomial-classification-4fe88bde7839
4. Creating a Neural Network from Scratch in Python: Multi-class Classification - https://stackabuse.com/creating-a-neural-network-from-scratch-in-python-multi-class-classification/
5. Deep Neural Network for Classification from scratch using Python - https://medium.com/@udaybhaskarpaila/multilayered-neural-network-from-scratch-using-python-c0719a646855
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. Classification Algorithms - https://scikit-learn.org/stable/modules/svm.html
8. 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/