CS776A : Assignment 1

Gunj Hundiwala, 22111024 Vivek Kumar Gautam, 22111069
{gunjmehul22,vivekg22}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

1 Dataset

Cifar 10: It is a dataset of 50,000 training images and 10,000 testing images, divided into 10 classes: truck, aeroplane, car, cat, deer, bird, cat, dog, and frog. Each image in the dataset is represented as a 3072-sized row, with every subsequent 1024 values corresponding to Red,Pixel values for the green and blue channels that can be altered to produce a 32*32 RGB image.

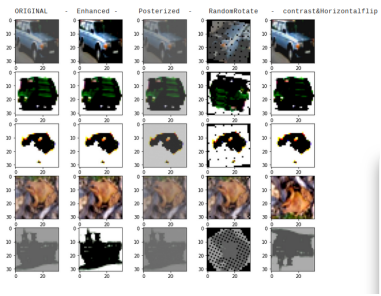


Figure 1: Data After Transformations

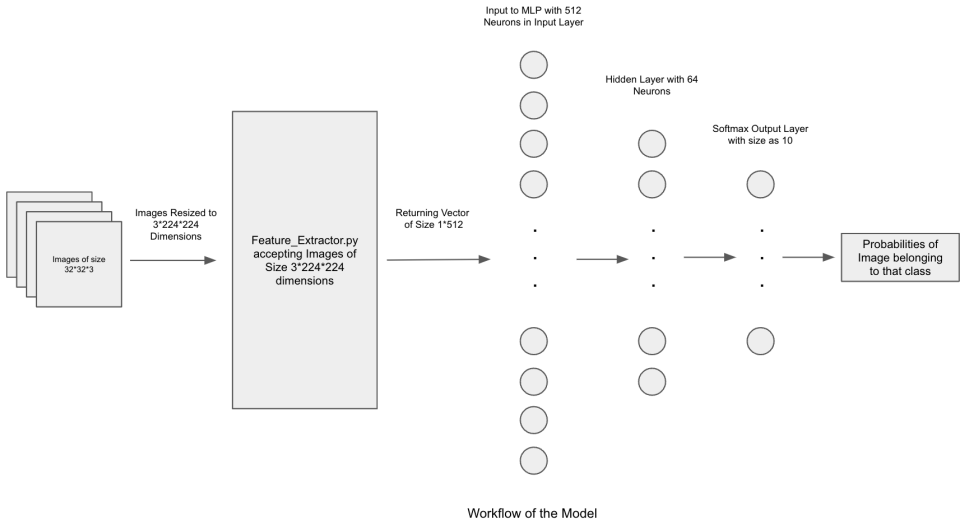
Augmented Cifar 10: We made a 50,000*3072-element Numpy array of zeros to use as the basis for the augmented dataset. Using the functions defined later in the Report, images from the Original dataset are transformed (Image enhancement, Posterization, Random Rotate and contrast+horizontal flip). One by one, these images are stored in the Numpy Array. Following the storage of all 50,000 photographs, the original data set is added to this array to build a final Numpy array with 100,000 images.

Due to Constraint on uploading File size, I have uploaded the dataset and Extracted features on the Google Drive. Steps to download these files and storing in provided with ReadMe.txt file.
LINK TO THE DRIVE

2 Workflow of the Model

About MLP Model: Our MLP Model consists of two hidden layers of size 64 each, an Input layer of size 512 and a Softmax output Layer of size 10.

Working: We loaded the dataset from memory. To construct Numpy arrays of Unaugmented and Augmented Dataset, this image dataset is what we used. To allow for feeding into the FeatureExtractor.py file, each image in these arrays is resized to 3*224*224 dimensions. Because the feature extraction method required a lot of time, the features are saved in the memory after being extracted from the images. These Features are now used as a 512-byte input to our MLP model. The model, after performing



Workflow of the Model

a forward pass, outputs an array of size 10, where each Number represents the likelihood that the image belongs to a specific class.Taking the Argmax (i.e., Class with the highest probability) of this array will provide the Final label. Refer to fig. 2 for more information.

3 Functions Used/Defined:

- **For transformations:**
  - **imEnhance():**This function is used to perform image enhancement by stretching the image pixel values in the smaller range of minimum to maximum pixel value of the image.
  - **posterize():** This function performs image posterization for generating augmented data. This function basically reduces the no of level of pixels in the image.
  - **randomRotate():** With the help of this function, you can rotate pictures at an angle between 180 and -180 degrees. Since the positions of the pixels in the rotated image are determined using approximations, some pixels may be left unfilled, giving the appearance of an image with holes.
  - **contrastHflip():** This function changes the contrast of the image randomly using the equation given in the question. It also flips the image horizontally with 50% probability.
- **resize224():**This is user defined function, which resizes the 32*32 dimension image to 224*224 dimensional image. This function interpolates every single pixel from 32*32 image to 7*7 size window of the 224*224 image. As a result we get an enlarged image.
- **Helper Functions:**

- **relu():** This function performs Relu Function on the data, i.e. it converts every -ve value to 0.
- **relugrad():** This function returns 1 if the value is more than 0, else return 0.
- **cross_entropy_loss():** This function calculates the Multi-Class cross entropy loss, using the following Equation:

$$Loss = \frac{-1}{m} \Sigma_{c=1}^{C} y_c . log(\hat{y}_c)$$

where y is one-hot vector and C is No. of Classes
$\hat{y}$ is the output of the Softmax Layer
m is the size of Mini Batch

- **softmax():** This function calculates the Probabilities of an image belonging to the class. The formula used is :

$$Softmax(x) = \frac{e^{x_i}}{\Sigma_i e^{x_i}}$$

- **accuracy():** This functions returns the no.s of Correct predicted labels. The no. of correct Labels are later divided in train() function to calculate overall accuracy.

- **Functions in class 'NeuralNetwork':**
  - **__init__():** This is used to Initialize the weights and biases for the model.
  - **forward():** This function performs the forward pass on the images Features. Working of Forward pass is explained in the following sections.
  - **backward():**This function Performs Backward Propagation of the error. Derivation of the gradients and Exact functioning is explained in the following sections.
  - **summary():** This Function prints the Size of Weights and biases in the model. In short, its beneficial for knowing the amount of learn-able parameters of the model.
  - **minmaxweights():** This Function prints the range of weights, i.e. minimum and maximum value of the weights. This function is used to know amount of updates happening, and also in debugging.
  - **saveweights():** This model is used to save weights in the memory, in 'params' folder. This function also accepts a parameter 'sub', which stands for subscript. This Sub parameter is added to the name of the files being saved. It becomes handy if we are working on more than one model.
  - **loadweights():** This Function accepts weights and biases and assigns them to model parameters
- **train():** This Function is the backbone which connects all the components above. This Function is used to train the Model.

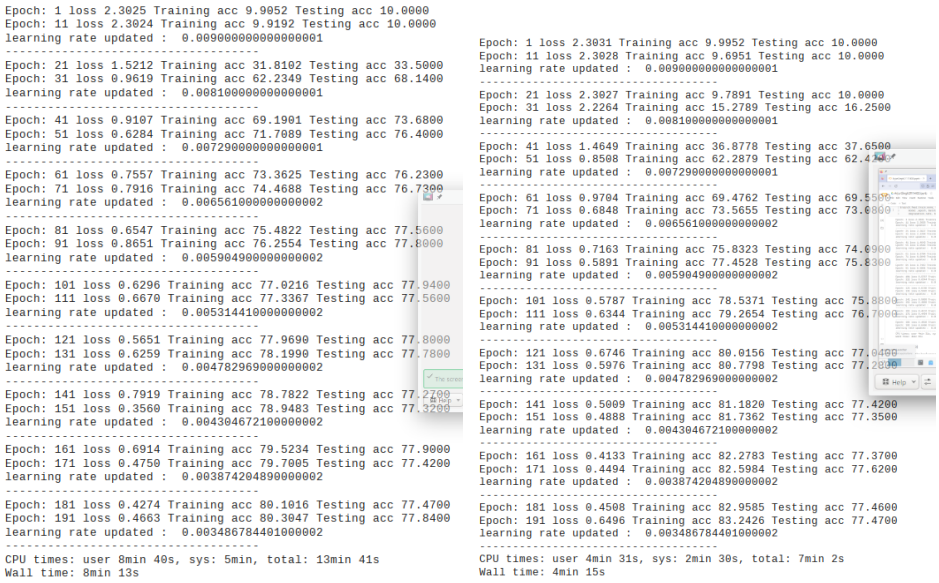


Figure 2: (a) Running training Loop on Augmented Dataset (b) Running training Loop on Unaugmented Dataset

4 Training

- **Notions Used :**
  - **Input :** 512 Sized Vector Input
  - z_1 **:** Weighted Output of Input layer
  - $a1$ **:** Activation(relu) applied on z_1
  - z_2 **:** Weighted Output of first hidden layer
  - $a2$ **:** Activation(relu) applied on z_2
  - z_3 **:** Weighted output of second hidden layer
  - $a3$ **:** Activation(relu) applied on z_3
  - $\hat{y}$: Predicted labels using Softmax layer
  - y **:** True Labels
  - **relu'() :** Gradient of relu function
  - W_1 **:** Weight matrix of input layer. Size : $512 * 64$
  - b_1 **:** Biases of Input layer. Size : $1 * 64$
  - W_2 **:** Weight matrix of input layer. Size : $64 * 64$
  - b_2 **:** Biases of Input layer. Size : $1 * 64$
  - W_3 **:** Weight Matrix of Hidden layer. Size : $64 * 10$
  - b_3 **:** Biases of Hidden layer. Size : $1 * 10$z
- **Setting up Hyperparameters:** The hyperparameters of both the trained models is set as following :
  - learning_rate = **0.01**
  - degradation_rate = **0.9**

- epoch = **200**
- batch_size = **64**

- **Forward Pass:**
  Forward pass is performed using following equations :

$$z1 = Input.W1 + b1$$

$$a1 = relu(z1)$$

$$z2 = a1.W2 + b2$$

$$\hat{y} = softmax(z2)$$

  Where, a1 is activation out of the first hidden layer
  $\hat{y}$ is the softmax output of the model

- **Backward Pass:**

  Backward Propagation is the technique, which is used to update weight matrices. Here we start calculating involvement of every weight in increasing loss value. We use Gradient Descent to decrease loss. Value of gradients, points in the opposite direction to that of local minima, so we update our weights in the opposite directions of the Gradients. After Repeating this step for several iterations, we finally arrive at local minima.

4.1 Derivation of Gradients

The gradient of softmax activation can be derived as follows:

$$\frac{\partial Loss}{\partial W_2} = \frac{\partial Loss}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial W_2}$$

where, $\hat{y}_c$ is softmax output for c^{th} class

$$= -\sum_{c=1}^{C} y_c \frac{\partial(log(\hat{y}_c))}{\partial \hat{y}_c} . \frac{\partial \hat{y}_c}{\partial z_2} \frac{\partial z_2}{dW_2}$$

$$= -\sum_{c=1}^{C} \frac{y_c}{\hat{y}_c} . \frac{\partial \hat{y}_c}{\partial z_2} \frac{\partial z_2}{dW_2}$$

$$= -\left[\frac{y_i}{\hat{y}_i} . \frac{\partial \hat{y}_i}{\partial z_2} \frac{\partial z_2}{dW_2} + \sum_{c=1, c \neq i}^{C} \frac{y_c}{\hat{y}_c} \frac{\partial \hat{y}_c}{\partial z_2} \frac{\partial z_2}{dW_2} \right]$$

$$= \left[-\frac{y_i}{\hat{y}_i} . \hat{y}_i(1 - \hat{y}_i) - \sum_{c=1, c \neq i}^{C} \frac{y_c}{\hat{y}_c} . (\hat{y}_c \hat{y}_i) \right] a_1, since \frac{\partial \hat{y}_i}{\partial z_2} = \hat{y}_i(1 - \hat{y}_i) and \frac{\partial z_2}{dW_2} = a_1$$

$$= -\left[y_i - \hat{y}_i y_i - \sum_{c=1, c\neq i}^{c} y_c \hat{y}_i\right] a_1$$

$$= \left[\hat{y}_i \left(y_i + \sum_{c=1, c\neq i}^{C} y_c\right) - y_i\right] a_1$$

$$= \left[\hat{y}_i . 1 - y_i\right] a_1, since \sum_{c=1}^{C} y_c = 1$$

$$= \left[\hat{y}_i - y_i\right] a_1$$

The gradient of ReLU function is 1 if the input is positive else zero. Now, we will use these results to calculate the gradients of weights and biases.

The loss function is $L(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$ where $\hat{y}$ represents the predicted label, y represents the actual label, and C represents the number of classes.

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

$$\frac{\partial L}{dW_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial W_2} = \left[\hat{y}_i - y_i\right] a_1$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{db_2} = \left[\hat{y}_i - y_i\right]$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \frac{da_1}{dW_1}$$

$$\frac{\partial L}{\partial W_1} = Input(\hat{y} - y).W_2.relu'(a_1)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial b_1}$$

$$\frac{\partial L}{\partial b_1} = (\hat{y} - y).W_2.relu'(a_1)$$

The model parameters, i.e. weights and biases are updated using the gradient descent method. Its update equations are as follows:

$$W_2 = W_2 - learning\_rate * \partial W_2$$

$$b_2 = b_2 - learning\_rate * \partial b_2$$

$$W_1 = W_1 - learning\_rate * \partial W_1$$

$$b_1 = b_1 - learning\_rate * \partial b_1$$

here we are taking steps in the opposite directions of the gradient.

- **Multi Step Learning Rate :** As value of Learning Rate is very important. As the model is trained more and more, the initialized learning rate becomes too large for the model, as a result, model's weights start to oscillate. As a solution, I have used the concept of Multi-Step-Learning-Rate i.e. the value of learning rate is degraded by some amount after every 20 Epochs. For this Assignment I found degradation rate as 0.9 to be effective.
  Learning Rate is Modified using formula:

$$Learning\_rate = Learning\_rate * Degradation\_rate$$

$$Where, 0 < Degradation\_rate \leq 1$$

5 Results

After Training for 200 Epochs, Testing Accuracy achieved on Unaugmented Dataset is 77.57% and on Augmented Dataset is 77.36%. Highest Accuracy achieved during training is 78.04% on Unaugmented dataset model and 78.21% on Augmented Dataset i.e. after training for 200 epochs Augmented Dataset model is working 0.15% - 0.2 % data. According to Training Stats, Augmented Dataset is performing really well compared to Unaugmented Dataset model. After 1 Epoch, Augmented Dataset has testing Accuracy of 45% but Unaugmented Dataset only has accuracy of 28%. This trend continues and Augmented Dataset shows 75.5% Testing accuracy compared to 73.8% Testing Accuracy of Unaugmented Datset after 11 Epochs.

```
Maximum Test Accuracy achieved while training in Unaugmented Dataset:  78.04
Maximum Test Accuracy achieved while training in Augmented Dataset:  78.21
```

Figure 3: Testing Accuracy after 200 Epochs

6 Conclusions

After carefully examining the graphs, we came to the following conclusions: :

- Training Accuracy on Augmented Dataset is higher than Training Accuracy on Unaugmented Dataset until epoch 75 and then the order reverses. The final training accuracy of Unagmented dataset is higher than the final trainig accuracy of augmented dataset. Initially, Unaugmented model Accuracy is lower than Augmented model when making predictions on the Test set, indicating that Unaugmented Dataset leads to Overfitting and underperforms when analysing unseen Data.

- The Amount of Difference between Augmented and Unaugmented Dataset Testing Accuracy is not that much, even though Difference between Augmented and Unaugmented Dataset Training Accuracy is around 3%.

- We can also observe that after our model has just run for few epochs (≤ 20) Per Epoch loss for Augmented Data Model is less, Training accuracy is comparatively more and also Testing Accuracy of Augmented Dataset Model is more compared to that of Unaugmented Dataset model. This shows that augmentation can be really useful if no.s of Epochs are less. Augmentation can be Useful for Developing Prototype Models. Augmented Dataset's Testing Accuracy is similar to Unaugmented Dataset. While there is a significant difference between augmented and unaugmented dataset training accuracy, there isn't much difference between augmented and unaugmented dataset testing accuracy.

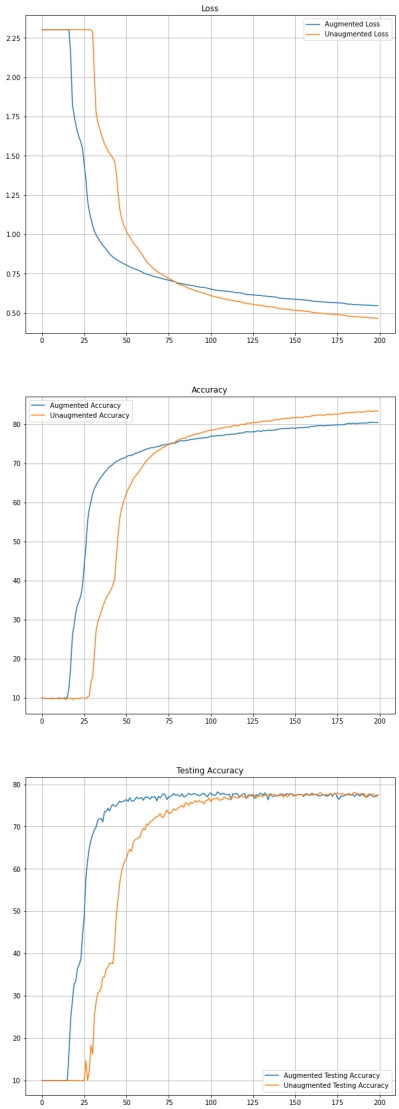


Figure 4: Loss (a) , Training Accuracy (b) and Testing Accuracy (c) of both Augmented and Unaugmented Data Model

Possible Explanation for Above mentioned Conclusions :

- Since Augmented dataset is more prone to distorted images, it becomes more robust to changes.
- Since no.s of data points are twice for Augmented Dataset. This leads to twice the amount of updates (if batch size is kept same), as a result it converges faster in less no.s of epochs.

7 References

- https://wch.github.io/latexsheet/
- https://www.adeveloperdiary.com/data-science/deep-learning/neural-network-with-softmax-in-python/
- https://pytorch.org/docs/stable/index.html
- https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1
- https://www.cs.toronto.edu/ kriz/cifar.html
- https://numpy.org/doc/stable/
- https://www.adeveloperdiary.com/data-science/deep-learning/neural-network-with-softmax-in-python/