# Report - Supervised and unsupervised training

Code is available at : <https://github.com/abhinav3/CIFAR-Autoencoder-Classification>

[Report - Supervised and unsupervised training](#_7tqm8b5jzn2)

[1. Dataset used for classifier training :](#_a4sf1499acec)

[Training dataset :](#_rglln8qiv3a2)

[2. Experimentation](#_g8z4p1rpef19)

[Dataset Exploration](#_5mpvell8umro)

[Data Preprocessing](#_jff3kwu001nh)

[A). AutoEncoder Training](#_lfd34avoz7oi)

[Deciding the Autoencoder Architecture](#_ed0g3l9aqtc6)

[Deciding the Autoencoder Loss function (BCELoss vs MSELoss)](#_l5jprnu3ha7w)

[B). CNN Classifier Training:](#_66ohjy9w7ubv)

[Ensemble Classifier Model Architecture](#_6k8ibsta0b1l)

[Deciding the CNN Classifier Architecture](#_ns72emvex81n)

[3. Results](#_guimsowb7ocq)

[Class wise accuracy:](#_p1er1pw4xo7l)

[Confusion matrix](#_k30nlnfmbnue)

[4. Result Analysis :](#_m9hggybvtdyg)

[Contribution](#_doqstqwr763)

[5. Notebooks descriptions and Weights files](#_bwtxdm1dnmqm)

[Notebooks:](#_kgv4jnlrhq63)

[Weights (refer weights folder):](#_hteeqbti6apm)

[6. Sources I referred:](#_niqwk4gr4b6k)

## 1. Dataset used for classifier training :

CIFAR10 has 10 classes and 60k images total (6k images per class).

### Training dataset :

* I’ve used 3k images (50%) for training for ***bird, deer and truck***.
* And I’ve used 5k images for training for the rest of the classes.
* The leftover count of 1k images were used for validation for each class.

## 2. Experimentation

### Dataset Exploration

[This notebook](https://github.com/abhinav3/CIFAR-Autoencoder-Classification/blob/master/Notebook2-AutoEncoder-MSELoss-Compressionfactor2.ipynb) ([Link](https://github.com/abhinav3/CIFAR-Autoencoder-Classification/blob/master/Notebook2-AutoEncoder-MSELoss-Compressionfactor2.ipynb)) contains the dataset exploration and analysis. Each image is of 3\*32\*32 shape. The CIFAR10 dataset images pixel values are having following stats for the mean and standard deviation :

Cifar10\_mean = [ 0.4735, 0.4640, 0.4295]

Cifar10\_std = [ 0.2200, 0.2165, 0.2151]

The 10 classes with labels are as following :

0 : airplane

1 : automobile

2 : bird

3 : cat

4 : deer

5 : dog

6 : frog

7 : horse

8 : ship

9 : truck

### Data Preprocessing

* I ***normalize the data by subtracting their mean and dividing by their standard deviation*** values as obtained above.
* I take ***horizontal flip (with 0.5 likelihood)*** and ***random rotation (by 10 degrees)*** as data augmentation techniques.

### A). AutoEncoder Training

#### Deciding the Autoencoder Architecture

1. **Deciding the encoding kernel size in autoencoder** : I’ve experimented with kernels sizes varying 3 till 7. Bigger the kernel size I used for the convolutional layer in encoding network, smaller is the latent vector size and hence more is information loss. That's why the quality of generated images are better while using smaller size kernels and hence a little bigger size latent vector. **Hence I fixed the encoding kernel size to be 3.**
2. **Deciding the decoding kernel size in autoencoder** : For Upscaling in the decoder network, **4x4 kernels gives the best reconstruction results.** 
   1. 2x2 kernels can only learn nearest pixel upscaling.
   2. 3x3 kernels can do bilinear but will require asymmetric padding.
   3. But 4x4 can do bilinear again without asymmetrical padding.
3. **Deciding the number of hidden layers in Autoencoder :**

Here are the accuracy scores of the classifier on the autoencoder embeddings for different values of hidden layers (any hence compression factors) in autoencoder.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Classification Accuracy after 2 epochs | Classification Accuracy after 10 epochs | Classification Accuracy after 30 epochs |
| 3 hidden layers  (Latent vector size is (48, 4, 4)) | 34% | 44% | 45% |
| 2 hidden layers  (Latent vector size is (48, 8, 8)) | 42% | 46% | 55% |
| 1 hidden layer  (Latent vector size is (48, 16, 16)) | 50% | 60% | 62% |

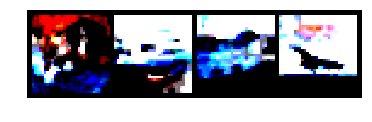
I find that more is the number of hidden layers used while reducing the size of input image, more is the information loss and hence lesser is the classification loss.

**Hence I use Autoencoder with just 1 hidden layer with latent vector size of (48, 16, 16).**

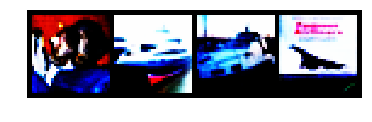
#### Deciding the Autoencoder Loss function (BCELoss vs MSELoss)

I experimented with two types of losses : Cross Entropy Loss (BCELoss) and Mean Square Loss (MSELoss). The reconstruction images using **MSELoss were much better than that of cross entropy loss.**

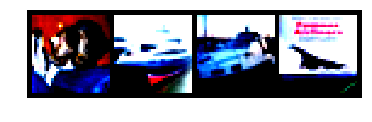
Here is a comparison :



a). Reconstructed image (BCELoss)



b). Reconstructed image (MSELoss)



c). Original images

We can see a visible difference b/w the reconstructed images (Look at the very last image for red parts). So the reconstruction using BCELoss looks more blobby and unlike the original.

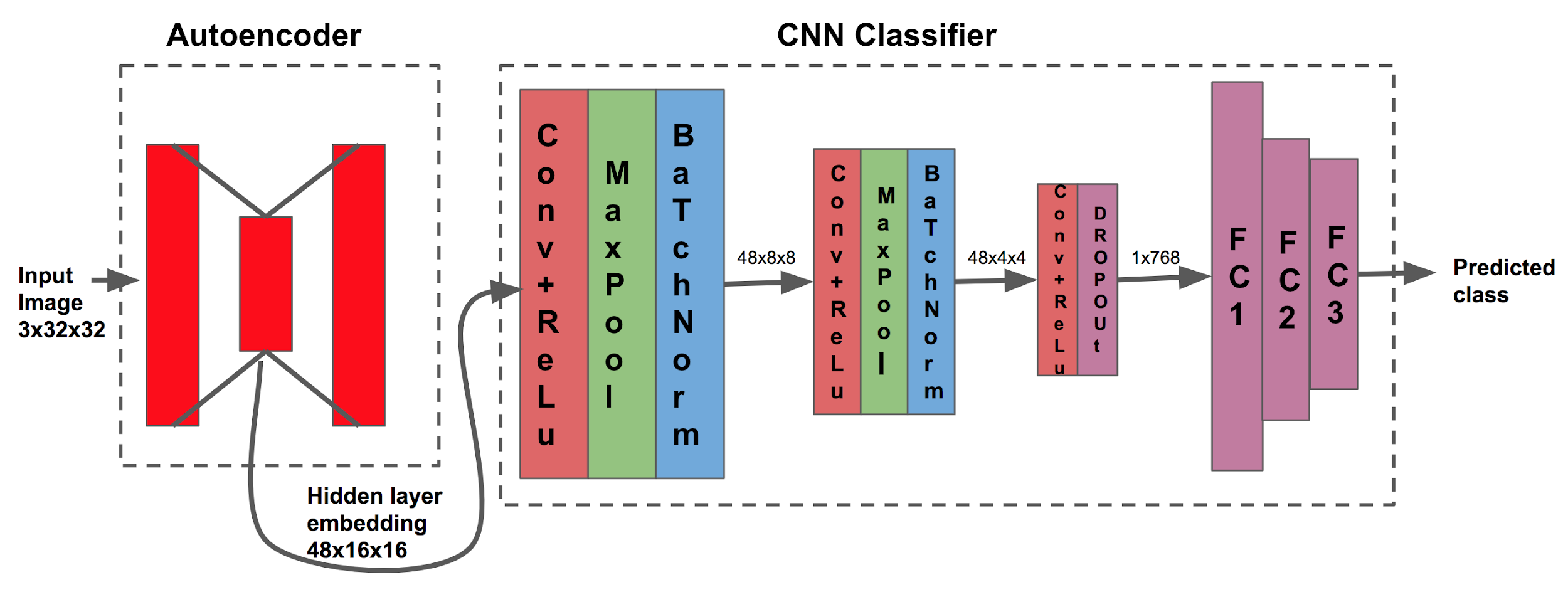
MSELoss tends to predict more continuous range of values while reconstruction. It’s more commonly used in all reconstruction works like mask generations etc. A possible explanation can be that BCELoss is asymmetric while MSELoss is symmetric. (Ref : [stackoverflow](https://stats.stackexchange.com/questions/245448/loss-function-for-autoencoders)).

Hence reconstruction image were having more clear edges, corners and shapes while using MSELoss than BCELoss.

### B). CNN Classifier Training:

#### Ensemble Classifier Model Architecture

* In the ensemble classifier model, the pre trained autoencoder is used to first obtain a (48x16x16) size latent vector encoding from a 3x32x32 input image.
* This latent vector feeds as i/p to the classifier which predicts a class label.
* Data Augmentation is used with **Random** **horizontal flipping (0.5 likelihood)**.
* Data normalization is also used by using CIFAR10 mean and std. dev. as already discussed.
* A **mini batch size of 64** was used based on the hardware availability and general standards.
* Below is the architecture of our end to end pipeline.



#### Deciding the CNN Classifier Architecture

1. **Deciding the loss function :** Since its a **classification** task, I go ahead with the **cross entropy loss** with sigmoid layer at last.
2. **Deciding the optimizer :** I tried with three different optimizers functions and here are the classification accuracies for different epochs of training :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Val Accuracy after 2 epochs | Val Accuracy after 10 epochs | Val Accuracy after 30 epochs | Val Accuracy after 100 epochs |
| Adadelta | - | - | - | 51% |
| Stochastic Gradient Descent with momentum | 54% | 56% | 58% | 59% |
| Adam | 56% | 61% | 63% | 66% |

Hence I go ahead with the Adam optimizer.

Adding a **Dropout layer** just before the last convolution layer boosts the classification accuracy by 6-7% and adding **BatchNormalization** layers after each conv->Relu->pool block gives a further boost of 6%.

Here are the accuracies :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Val Accuracy after 2 epochs | Val Accuracy after 10 epochs | Val Accuracy after 30 epochs | Val Accuracy after 100 epochs |
| Vanilla CNN layers | 56% | 61% | 63% | 66% |
| Adding Dropout layer (with 0.1 prob) | 62% | 66% | 67% | 70% |
| Dropout + BatchNormalization | 68% | 73% | 74% | 77% |

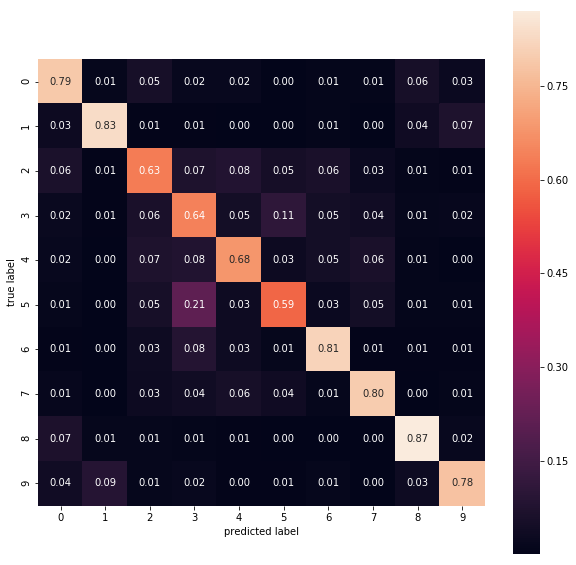
The reason for the boost in accuracy is that **dropout and batch normalization try to mitigate the bias and noise learnt in the network.** It’s a form of **regularization and helps in avoiding overfitting.** Batchnorm further helps in **better and faster convergence by moving the layer activations around it’s mean.**

## 3. Results

### Class wise accuracy:

|  |  |
| --- | --- |
| Accuracy of plane | 79% |
| Accuracy of car | 83% |
| Accuracy of bird | 63% |
| Accuracy of cat | 64% |
| Accuracy of deer | 68% |
| Accuracy of dog | 59% |
| Accuracy of frog | 81% |
| Accuracy of horse | 79% |
| Accuracy of ship | 87% |
| Accuracy of truck | 77% |

### Confusion matrix



Class labels = { 0 : 'plane', 1 : 'car', 2 : 'bird', 3 : 'cat', 4 : 'deer', 5 : 'dog', 6 : 'frog', 7 : 'horse', 8 : 'ship', 9 : 'truck' }

## 4. Result Analysis :

Best accuracy of our ensemble autoencoder classifier network on the 10000 test images : ~77%

Analysing the confusion matrix we infer that “Cats” and “dogs” predictions are confused with each other. Similar is the case between “trucks” and “cars”, “planes” and “ships” etc.

I’ve trained a vanilla classifier end to end with similar layers as well which has reached overall accuracy of ~ 80%.

**So, we can say that by using unsupervised method to get the latent encodings and then using that encoding to train our classifier has almost similar accuracy to that of a similar architectured end to end fully supervised classifier.**

### Contribution

**Plausible benefits and contribution** of using an autoencoder network with a classifier (our ensemble model) could be :

1. It gives us a compressed representation of an input image. So it helps in data size reduction (less storage space required).
2. It helps in denoising. So only the relevant activated features would be fed to the classifier. Thus our classifier is less likely to learn noise.

## 5. Notebooks descriptions and Weights files

### Notebooks:

1. “[DataFilteringAndManipulation.ipynb](https://github.com/abhinav3/CIFAR-Autoencoder-Classification/blob/master/DataFilteringAndManipulation.ipynb)” is the jupyter notebook for filtering the 50% data points for birds, deer and truck as pytorch DataLoader doesn’t provide us with any way to control the number of samples I wish to extract. I’ve added similar script is there for test set as well.
2. “[Notebook1-AutoEncoder-Compressionfactor16.ipynb](https://github.com/abhinav3/CIFAR-Autoencoder-Classification/blob/master/Notebook1-AutoEncoder-Compressionfactor16.ipynb)” compares the autoencoder reconstructions using BCELoss vs MSELoss.
3. “[Notebook2-AutoEncoder-MSELoss-Compressionfactor2.ipynb](https://github.com/abhinav3/CIFAR-Autoencoder-Classification/blob/master/Notebook2-AutoEncoder-MSELoss-Compressionfactor2.ipynb)” is the final notebook to be used for training the autoencoder with 48x16x16 latent vector size (compression factor is 2).
4. “[Ensemble\_model\_classifier\_dropout\_bn-16\_16.ipynb](https://github.com/abhinav3/CIFAR-Autoencoder-Classification/blob/master/Ensemble_model_classifier_dropout_bn-16_16.ipynb)” is the final notebook where we take the output of an autoencoder hidden layer and then train our CNN classifier on it.
5. “[Vanilla\_model\_classifier.ipynb](https://github.com/abhinav3/CIFAR-Autoencoder-Classification/blob/master/Vanilla_model_classifier.ipynb)” is the notebook for fully supervised CNN classifier for comparison purpose.

### Weights ([refer weights folder](https://github.com/abhinav3/CIFAR-Autoencoder-Classification/tree/master/weights)):

1. “model\_BCELoss().pkl” and “model\_MSELoss().pkl” are the weights files for the autoencoder trained using BCELoss and MSELoss respectively.
2. “autoencoder\_16\_16.pkl” is the weight file where autoencoder gives out a 48x16x16 latent encoding vector.
3. “Final\_ensemble\_cnn\_classifier\_dropout\_batchnorm.pkl” is the weight file for the final CNN classifier.
4. “Vanilla\_cnn\_classifier.pkl” is the weight file for fully supervised CNN classifier.

## 6. Sources I referred:

1. Ref Autoencoder: <https://www.cs.toronto.edu/~lczhang/360/lec/w05/autoencoder.html>
2. Making a pytorch CIFAR10 classifier : <https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html>
3. Claim : Adadelta giving better result : <https://www.datascience.com/blog/transfer-learning-in-pytorch-part-one>
4. Why MSELoss is better for our case : <https://stats.stackexchange.com/questions/245448/loss-function-for-autoencoders>
5. Research paper using Autoencoder for anomaly detection and classification: <https://arxiv.org/pdf/1802.06360.pdf>