DS-265 DEEP LEARNING FOR COMPUTER VISION

Assignment #1

Sai Teja Kuchi (21317)
kuchisai@iisc.ac.in

1 Training a Single-Layer classifier

Training a linear model on MNIST dataset using learning-rate(α) = 1e−4, regularization-coefficient(λ) = 1e−3, batch-size = 32 and epochs = 50, we get the following results. Validation accuracy = 85.56%

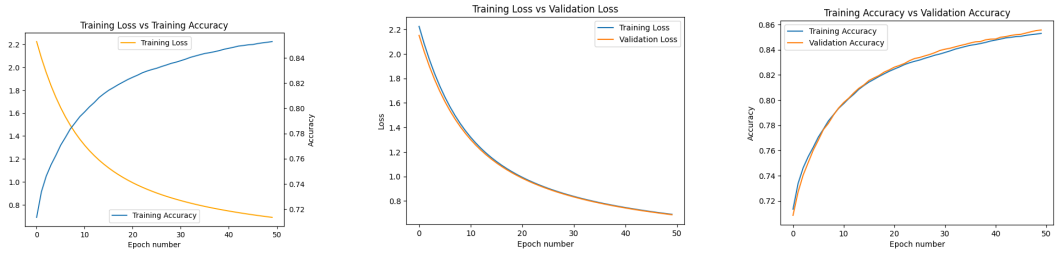

Figure 1: Using only 80% of data for training and rest 20% for validation.

From the above results, we can observe that both the training and validation loss keep decreasing with the number of epochs and there is no gap between them, which suggests there is no over-fitting of model on the training data.

1.a Finding optimal hyperparameter values

Using regularization-coefficient(λ) = 1e−3 and epochs = 50.
Tuning learning-rate (λ) and batch-size using [1e−2, 1e−3, 1e−4] and [32, 64, 128] values respectively to observe their effect on training, we get the below results.

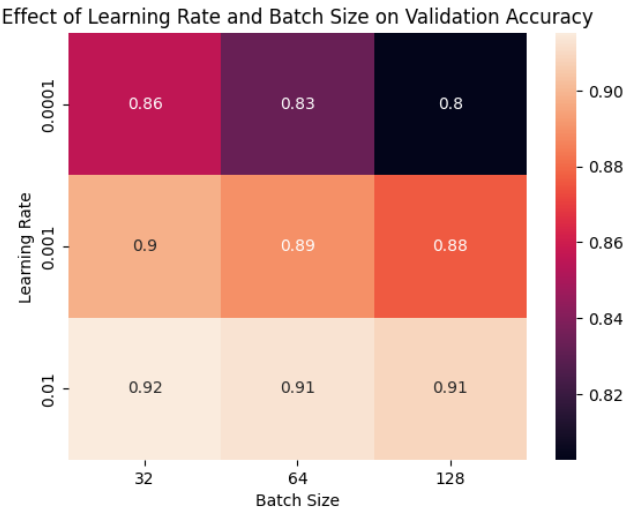

Figure 2: Effect of learning-rate and batch-size on training data.

- From (2), we can observe that as batch-size increases for fixed learning-rate there is slight decrease in validation-accuracy. Smaller the batch-size, the more frequent the

model gets updated which leads to faster convergence but this also has a side-effect of increase in the overall training time which is not captured here.

- Larger learning-rate for fixed batch-size leads to faster convergence by quick updates but it might diverge from the actual optimal solution. Generally a decay strategy is preferred where we start with high learning rate followed by decaying it to smaller value so that we don't diverge and get stable results.

Further, tuning all the hyperparameters, i.e learning rate (α), regularization-coefficient(λ), epochs, batch-size using the data [1e-1, 1e-2, 1e-3], [1e-1, 1e-2, 1e-3], [50, 100], [32, 64] respectively we get the table result (1). (Results for other hyperparameter values is present in the jupyter-notebook.)

Learning Rate	Regularization Coefficient	Epoch Count	Batch Size	Validation Accuracy
0.010	**0.001**	**100**	**64**	**91.51%**
0.010	0.001	100	32	91.42%
0.010	0.001	50	32	91.41%
0.100	0.001	50	64	91.30%
0.100	0.001	100	64	91.27%

Table 1: Displaying top-5 values sorted by validation accuracy.

Using the best hyperparameters from the above table (1), i.e Learning Rate = 1e-2, Regularization Coefficient = 1e-3, epochs = 100 batch-size = 64, we get,

- Train Accuracy = 91.73%
- Validation Accuracy = 91.47%
- Test Accuracy = 91.81%
- Best performing class is class-0 with an test accuracy of 97.86%
- Worst performing class is class-5 with an test accuracy of 86.66%

We can observe that the validation accuracy is close to train accuracy and it also performs well on the test dataset which it has not seen. Thus it's not overfitting on the training-data.

1.b Training linear classifier from scikit-learn

Results from implemented model

- Train Accuracy = 91.73%
- Test Accuracy = 91.81%

Results from scikit-learn **LogisticRegression model**

- Train Accuracy = 93.65%
- Test Accuracy = 92.57

We observe that the implemented linear models results are close to that of the LogisticRegression model. The implemented models results can be further improved by training on more epochs but need to make sure that it doesn't overfit on the training-data by making use of techniques like early-stopping etc.

1.c Training on subset of data

Using the best hyperparameters from the above table (1), i.e Learning Rate = 1e-2, Regularization Coefficient = 1e-3, epochs = 100 batch-size = 64, we get,

Figure 3: Using only 10% of data for training and rest 90% for validation.

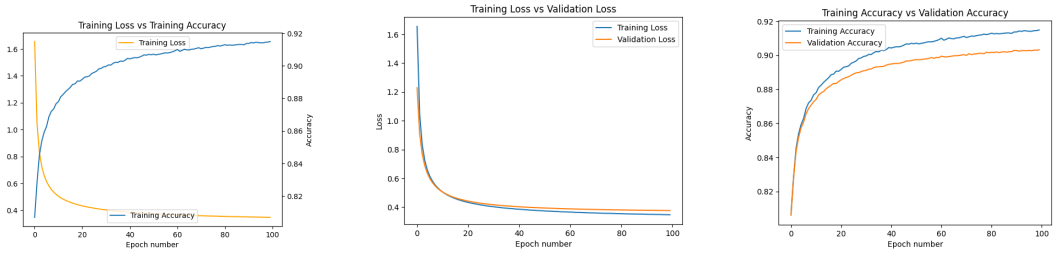

Figure 4: Using only 20% of data for training and rest 80% for validation.

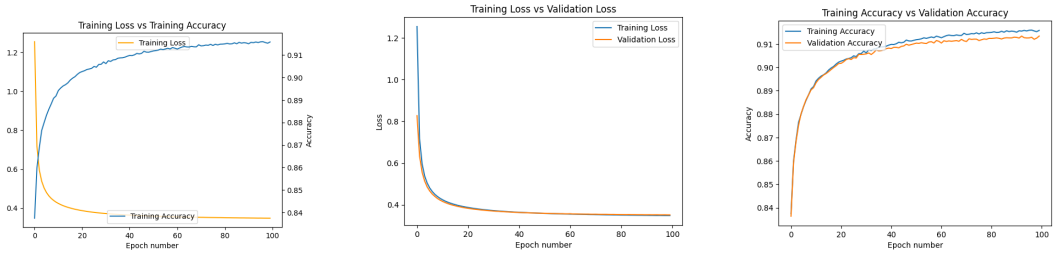

Figure 5: Using only 50% of data for training and rest 50% for validation.

From (3), (4), (5), we observe that the model is not overfitting.

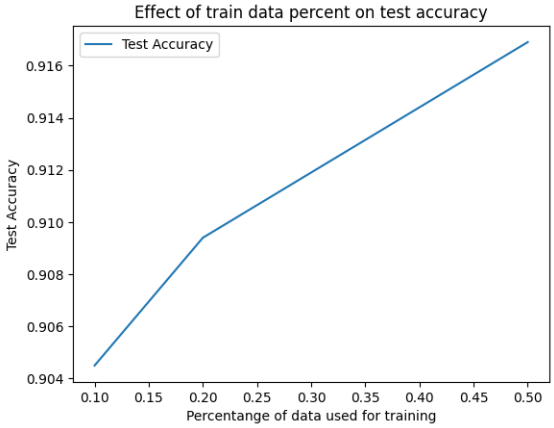

Figure 6: Training Percent Vs Test Accuracy

From (6), we observe that even with small percent of data used for training, we achieve good test accuracy and increasing the amount of data used for training further improves the test accuracy. However there will be a point where increasing the data further will not improve the results significantly and worse will lead to overfitting of the model.

2 Scalar Backpropagation

x	y	z	forward-value	grad-x	grad-y	grad-z
2	4	1	-0.23823101469115085	2.1417102890343505e-16	0.43720979194276516	-0.30697227565888
9	14	3	-1.3132617097862371	-2.5313688314302287e-15	-1.148344174369598	2.672161875217988e
128	42	666	-0.3132616875182231	1.3244281103803985e-14	-0.42245219681098717	-0.0
52	14	28	-0.31326168751822264	-6.626324387577346e-15	-0.4224521968109866	-0.0

3 Modular Vector Backpropagation

Training a 2 layer MLP model on MNIST dataset using hidden-channels = 256, learning-rate(α) = 1e$-$3, regularization-coefficient(λ) = 1e$-$3, batch-size = 32 and epochs = 1000. Early stopping incase the validation loss either starts increasing or difference between the validation loss of 2 consecutive epochs after atleast training for 250 epochs is very small, we get the following results.
Validation accuracy = 95.61%

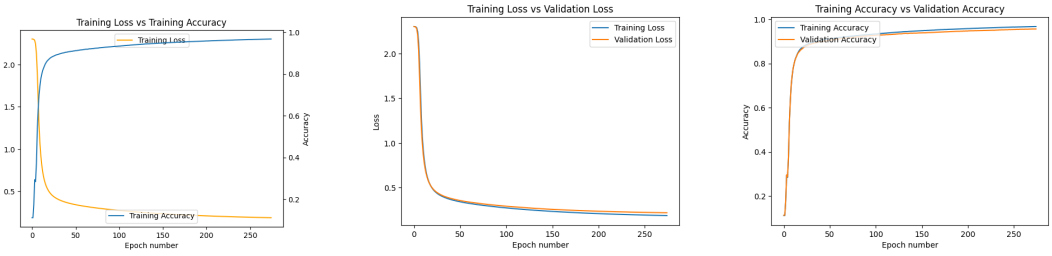

Figure 7: Using only 80% of data for training and rest 20% for validation.

From the above results, we can observe that both the training and validation loss keep decreasing with the number of epochs and there is very small gap between them, and as soon as the validation loss started increasing at epoch = 274, we stopped training it further to prevent from overfitting of the model.

3.a Finding optimal hyperparameter values

Using regularization-coefficient(λ) = 1e−3 and epochs = 1000. Early stopping incase any of the condition described above is satisfied.
Tuning learning-rate (λ) and batch-size using [1e−2, 1e−3] and [32, 64] values respectively to observe their effect on training, we get the below results.

Learning Rate	Batch Size	Validation Accuracy
0.010	32	97.56%
0.010	64	97.48%
0.001	32	95.60%
0.001	64	95.28%

Table 2: Tuning learning-rate and batch-size

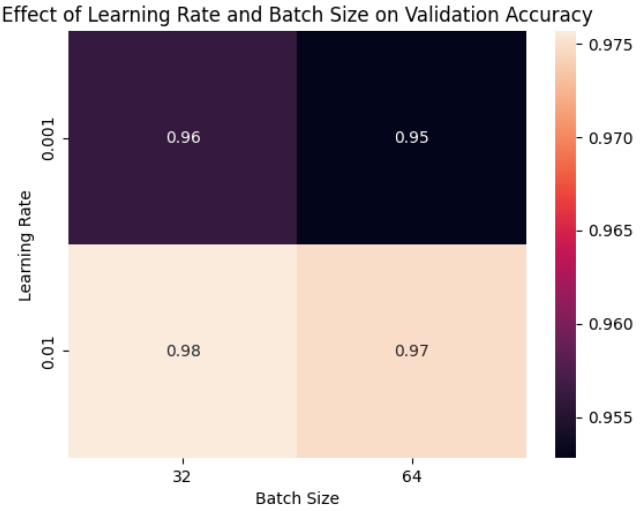

Figure 8: Effect of learning-rate and batch-size on training data.

- From (8), similar to single layer model as batch-size increases for fixed learning-rate there is slight decrease in validation-accuracy.
- As the model became more complex than the simple linear model with the addition of ReLU and more layers, we are able to learn more complex and non-linear patterns from the training data. Thus we observe an overall increase in accuracies of the model compared to previous models. This can be observed from the below section.

Using the best hyperparameters, i.e Learning Rate = 1.00e-02, Regularization Coefficient = 1.00e-03, epochs = 1000 (with early-stopping), batch-size = 32, we get,

- Train Accuracy = 99.99%
- Validation Accuracy = 97.57%
- Test Accuracy = 97.87%
- Best performing class is class-1 with an test accuracy of 98.94%
- Worst performing class is class-9 with an test accuracy of 96.53%

We observe there is slight overfitting of model on training data as the training percent is close to 100. This can be fixed by using more complex model architecture.

3.b Training on subset of data

Using the best hyperparameters, i.e Learning Rate = 1.00e-02, Regularization Coefficient = 1.00e-03, epochs = 1000 (with early-stopping), batch-size = 32, we get,

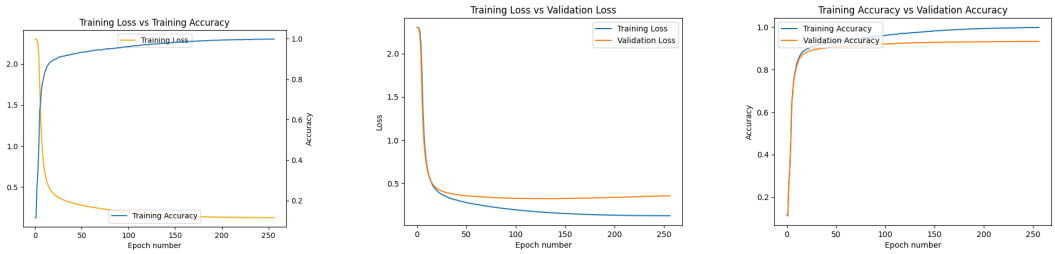

Figure 9: Using only 10% of data for training and rest 90% for validation.

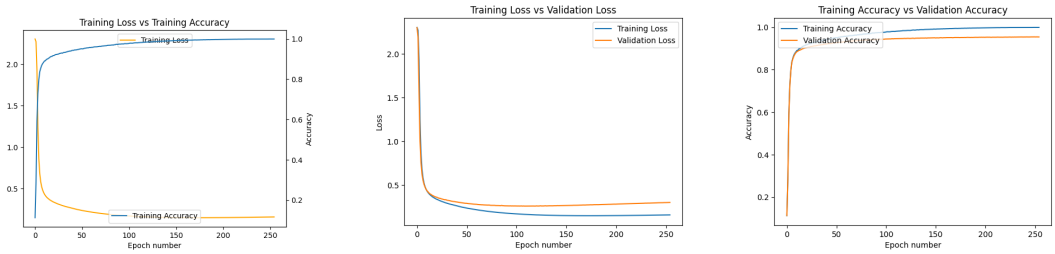

Figure 10: Using only 20% of data for training and rest 80% for validation.

From (9), (10), (11), we observe that the model is overfitting on the training data as the training loss keeps decreasing further. By increasing the percent of data used for training we observe that gap between the 2 loss (train and validation) curves is reducing, as complex models requires more amount data to not overfit for large training epochs.

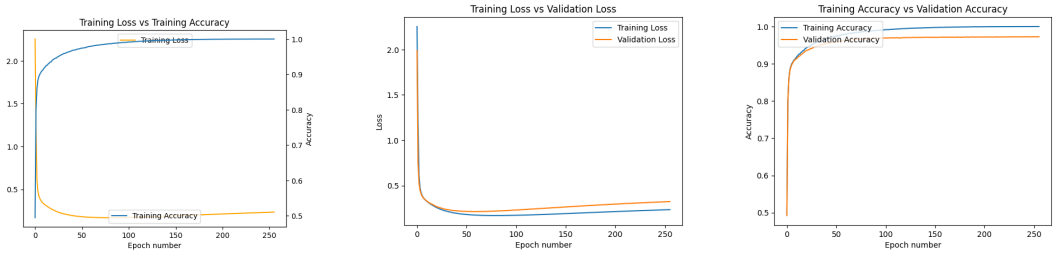

Figure 11: Using only 50% of data for training and rest 50% for validation.

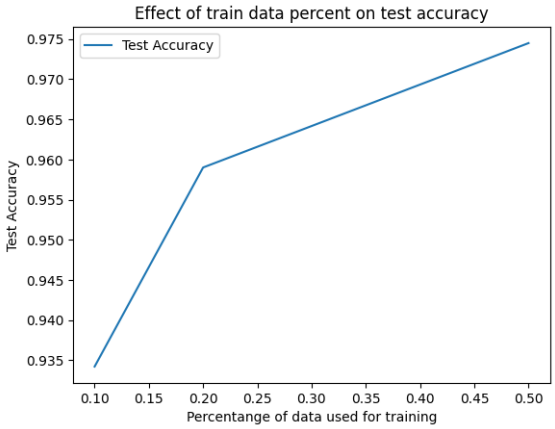

Figure 12: Training Percent Vs Test Accuracy

From (12), we observe that even with small percent of data used for training, we achieve good test accuracy and increasing the amount of data used for training further improves the test accuracy. However there will be a point where increasing the data further will not improve the results significantly and worse will lead to overfitting of the model.

4 Convolution Module

Using the gaussian filter (13) for blurring the first 5 images (14), we get (18)

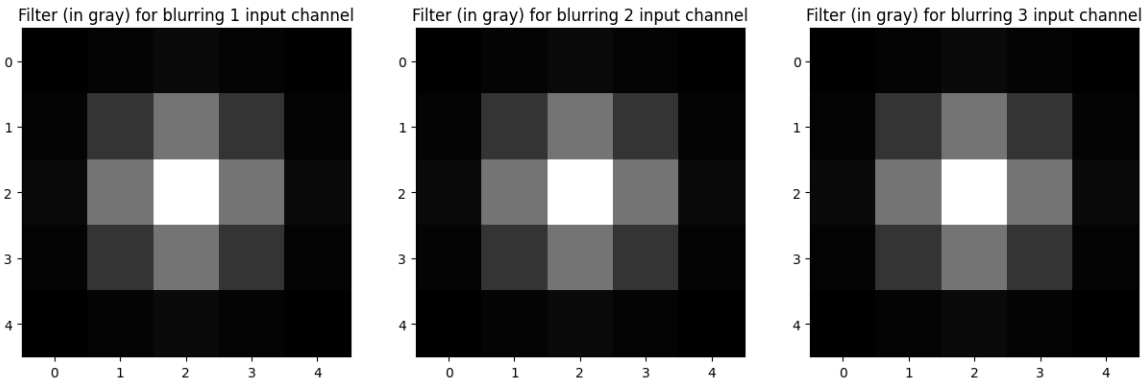

Figure 13: Gaussian Filter Data with $\sigma = 0.8$ and size = 5 x 5

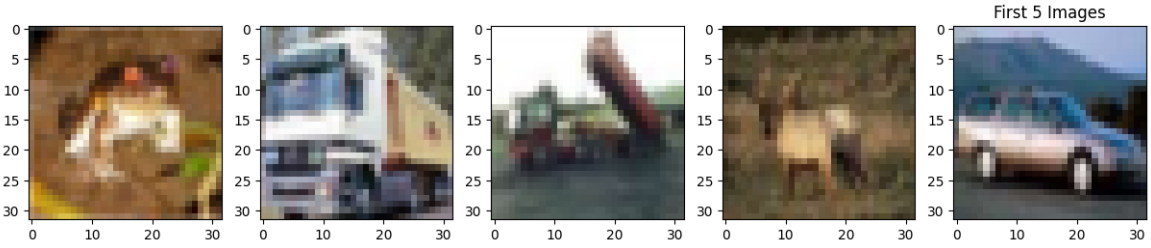

Figure 14: First 5 images of training data

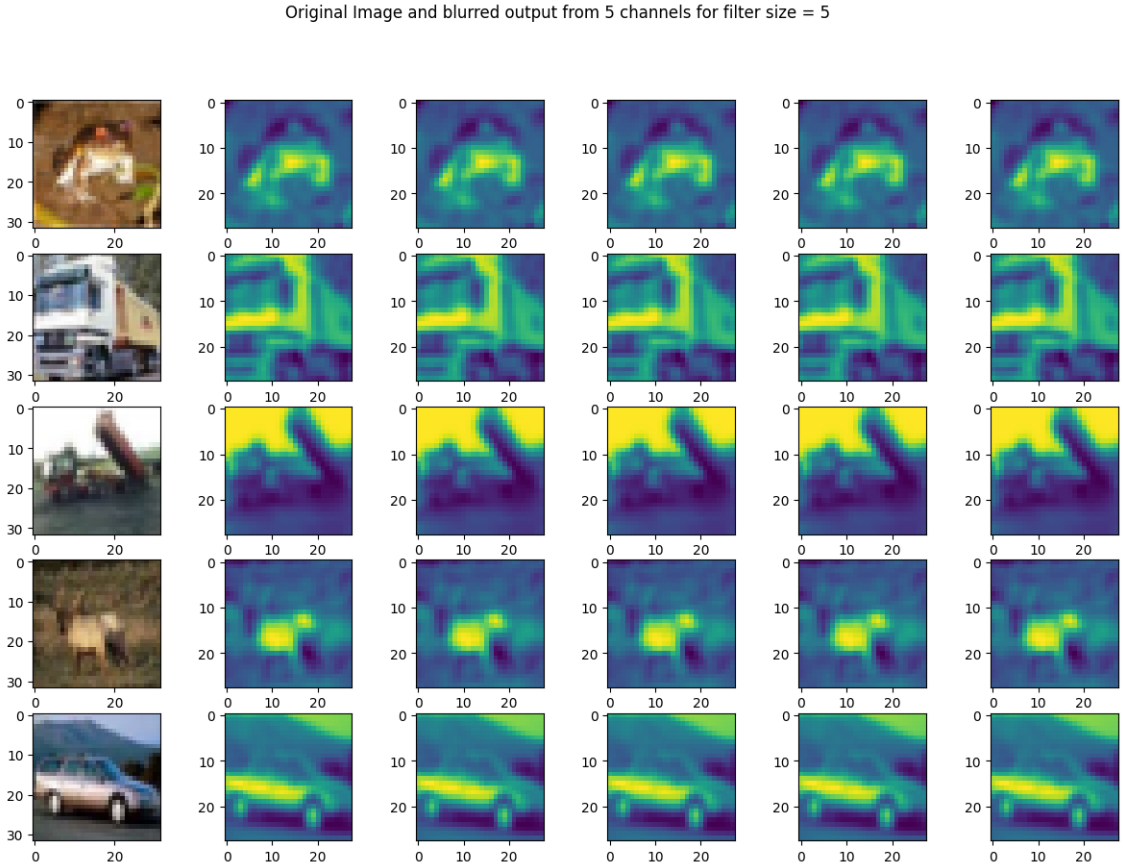

Figure 15: Original and Blur images across 5 output channels

5 Convolution Layer with L2 Loss

Training a convolution model on CIFAR-10 dataset using weights $\in \mathcal{U}[-1, 1]$, learning-rate(α) = 1e$-$5, regularization-coefficient(λ) = 1e$-$3, batch-size = 4 and epochs = 50. We observe that even after using small learning rate and L2 regularization, the gradients are still exploding as seen from (16).

```
epoch = 0, Loss :- 1.4109762388603082e+30
epoch = 1, Loss :- 2.8346672692352706e+60
epoch = 2, Loss :- 8.01510335313089e+90
epoch = 3, Loss :- 1.8459701758464843e+121
epoch = 4, Loss :- 4.3035175057837027e+151
epoch = 5, Loss :- inf
epoch = 6, Loss :- inf
epoch = 7, Loss :- inf
epoch = 8, Loss :- inf
epoch = 9, Loss :- inf
/tmp/ipykernel_216056/1432655735.py:84: RuntimeWarning: o
  grad_bias[chan_index] += grad_of_output_size[batch_inde
/tmp/ipykernel_216056/1432655735.py:80: RuntimeWarning: o
  grad_weights[chan_index, :, :, :] += r_field_data * \
/tmp/ipykernel_216056/1432655735.py:48: RuntimeWarning: i
  r_field_data * self.weights[chan_index, :, :, :])
/tmp/ipykernel_216056/1432655735.py:80: RuntimeWarning: i
  grad_weights[chan_index, :, :, :] += r_field_data * \
/tmp/ipykernel_216056/2151166081.py:37: RuntimeWarning: i
  c_layer.bias = c_layer.bias - alpha * grad_bias
epoch = 10, Loss :- nan
```

Figure 16: Exploding gradients

To fix the above problem, there are different ways one can proceed like using proper weight initialization (Xavier), batch normalization, gradient clipping, etc. In this case we use the 1st method as our weights need to be from uniform distribution, so gradient clipping is used instead.

Training on first 100 images with learning-rate(α) = 1e$-$2, regularization-coefficient(λ) = 1e$-$3, epochs = 50.Clipping the gradients weights and biases between -1 and 1.
Tuning using batch-sizes = [1, 4, 8], we get the following results,

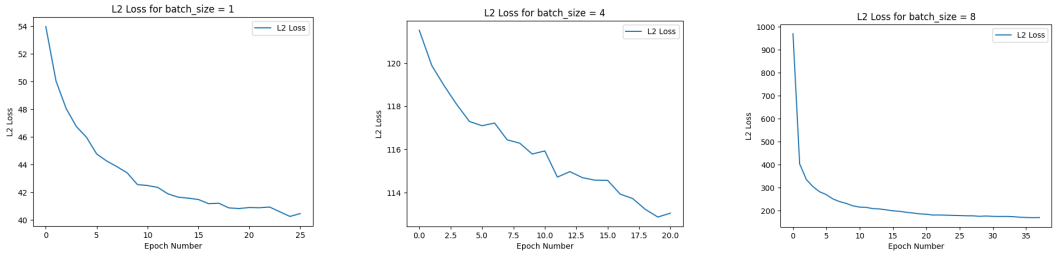

Figure 17: Effect of batch size on training.

From (17), as batch size decreases, the L2 loss between the 2 models on the batch of the data decreases aswell, as we are updating the weights frequently when the batch-size is smaller but this has an side-effect of longer training-times which is not captured here.

Batch Size	L2 Norm
1	**429.59033361927476**
4	561.2261602968755
8	617.4761607720582

Table 3: L2 Norm of output from 2 models.

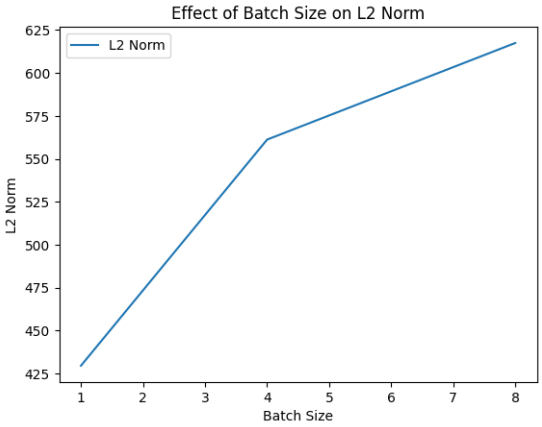

Figure 18: L2 Norm of output from 2 models.

We can observe that as batch-size increases the L2-Norm between the output of 2 models also increases.