EX1 - Advanced Practical Course In Machine Learning

Daniel Afrimi
203865837

November 12, 2020

1 Dataset Insights

From watching the train data set, I saw that lot of cat images labeled as ”car”.

In addition the data is imbalance, it means that there are a disproportionate ratio of observations in each class. most of the images are cars, while there are much less images of trucks and cats. Figure 1 shows the imbalance classes in the data set.

In Figures 2 we can see wrong label for a cat.

2 Pretrained Model Evaluation and Results

The given pretrained model tested on the ’dev’ dataset, and its predicting only the label 0, i.e the model predicts all of the images as ”cars”.

The accuracy on each class showed that the pretrained model has 100% on cars images, and 0% on the other classes. this is occurs due the fact that most of the images are cars and there are wrong labels. this is cause the model to learn how cars looks like, but it doesn't learn how to classify cats or trucks.

Figure 4 shows the confusion matrix of the pre trained net. each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class
The code for this section can be found in tester.py

3 Learning Rate

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated.

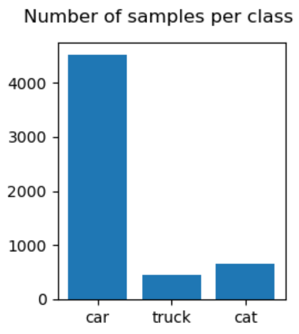


Figure 1: Imbalance Data

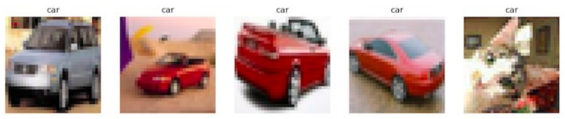


Figure 2: The right image is cat, but labeled as car

Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.

Figure 3 shows the with large/small lr, the net does not converge.

4 Model Training

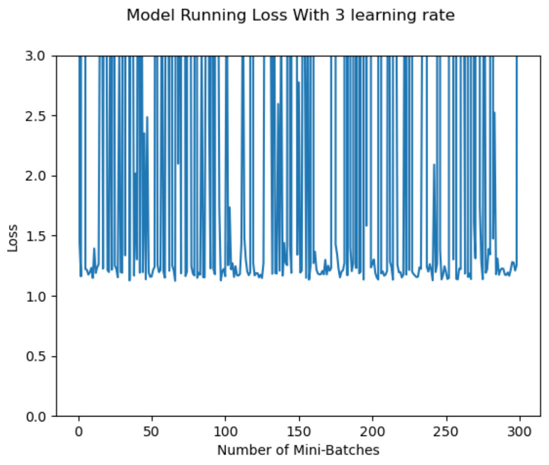
The model trained for 60 epochs (not to high to avoid overfit), with lr equal to 0.001 (for convergence) and with batch size of 32.

Because of the imbalanced data, I used the WeightSampler Pytorch Object, which instead of sampling uniformly, it will sample according to custom distribution.

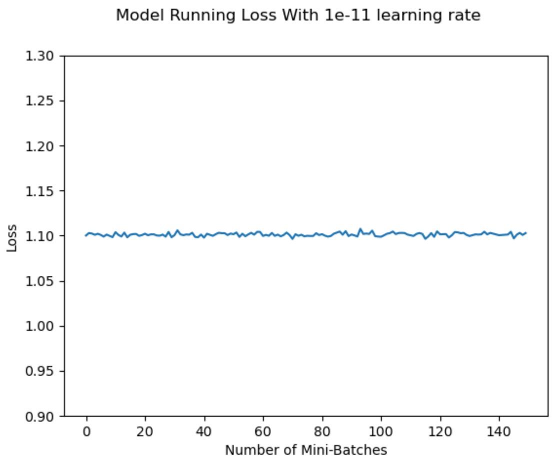
The custom distribution was defined by calculating how many samples there are for each class, then define a weight for each label, i.e according to this formula

$$\frac{\text{numbers of samples for class i}}{\text{total labels}}$$

In addition, I used transfroms functions which do augmentation (flipping, rotating), color changes of the image and more. This is for increasing the data for better results.



(a) learning rate is 3



(b) learning rate is 1e-11

Figure 3: Net does not converge with large or small learning rates

	Classes		
	Car	Cat	Truck
Car	100%	0%	0%
Cat	100%	0%	0%
Truck	100%	0%	0%

Figure 4: Confusion Matrix of the pre-trained model

By the way, the cats labels were fixed manually :)

The code for this section can be found in trainer.py

5 Adversarial example

The idea is rather than working to minimize the loss by adjusting the weights based on the backpropagated gradients, the attack adjusts the input data to maximize the loss based on the same backpropagated gradients.
In other words, the attack uses the gradient of the loss w.r.t the input data, then adjusts the input data to maximize the loss. the attack backpropagates the gradient back to the input data to calculate the gradient according to the image (instead of the weights).

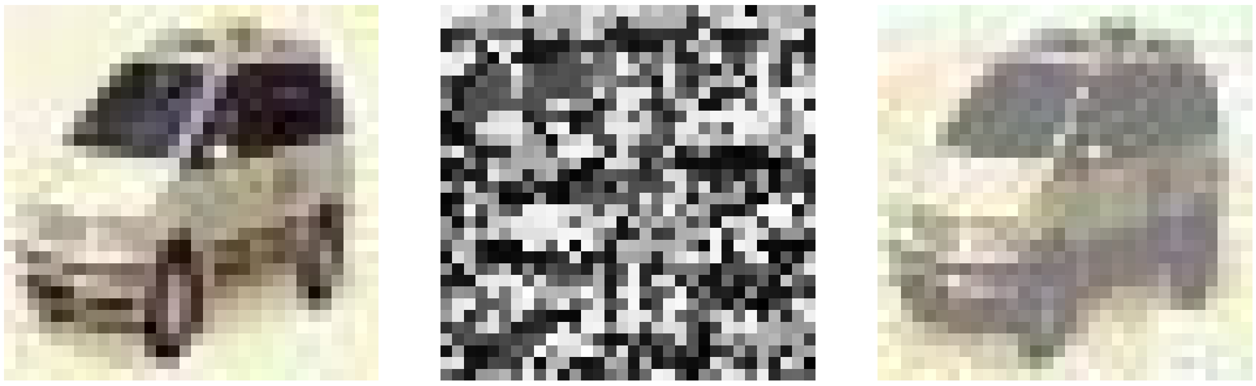
I generated the attack image, by taking a trained model and a test set. In the attack i took examples that the model predict on them well (i.e equals to the ground truth).

Than calculating the gradient of the loss function according the the image and than instead of taking a step against the direction of gradient (for minimize the loss), the net takes step in the same direction (maximizie the loss).

This is done by taking the sign of the grdient (the noise signal), and add it to the image ($\epsilon \cdot$ noise)

Figure 5 shows an example for the attack.
With epsilon = 0.05, multiply the original image the net label the outcome as cat, although this is a car.

The code for this section can be found in adversarialExample.py



(a) Original Image - 'Car' (b) Noise Signal (grey scale (c) Distorted Image - 'Cat'

Figure 5: The Original Image + 0.05 · Noise Signal = Distorted Image. The Distorted Image was predicted on the trained model as "Cat", although the Original Image predicted as 'Car'