Masters Thesis

Submitted June 2020, in partial fulfillment of
the conditions for the award of the degree **MSc Computer Science.**

Biswadeep Sen
MCS201821

Supervised by Prof K. V. Subrahmanyam

Department of Computer Science
Chennai Mathematical Institute

Abstract

Checking the robustness of various classifiers under adversarial attacks has been a very important problem in Machine Learning.Performance of various state of the art models like CNNs have been tested against adversarial attacks.In this project we try to check the robustness of SO(2) equivariant neural networks under adversarial attacks like FGSM and PGD.

We know that CNNs do very well in image classification tasks. So a natural question to ask is what the layers are learning. It is known that the first few layers learn some simple filters, which are common across most data sets. But the deeper layers learn more sophisticated features of the images. For the last section of this report we look at the features learned by a CNN in the last layer. We ask if we can use those features(for classification tasks) in a simple way using SVD.

Acknowledgements

I would like to thank my supervisor, Professor KV Subrahmanyam, for introducing me to the field of Computer Vision and robustness of classifiers, and for all the insightful discussions that we have had. I also wish to thank my senior Muthuvel Murugan, whose help I sought whenever I had a doubt.

Contents

Chapter 1

Basic Definitions and terminology

In this section we provide descriptions of the two different types of classifiers that we use in this project.

1.1 Convolutional Neural Networks

Convolutional Neural Networks are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer.

ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet architecture.(Ref [5])

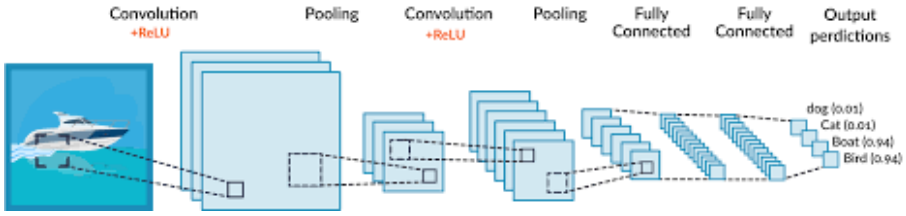


Figure 1.1: A typical CNN architecture

1.2 SO(2) Equivariant Neural Network

This is a Neural Network model working in the Fourier domain of the group SO(2), which learns features of 2-D images invariant/equivariant to the group SO(2). Equivariance here would mean that if the networks inputs are rotated by an angle θ, then the corresponding activations at layer l, gets transformed by T^l_θ, from some fixed set of transformations $T^l_{\theta\theta\in SO(2)}$. For non linearity, tensor product of the activations is used. Here our motivation is to learn features of images which are invariant to rotations about the center. So we use the Fourier coefficients of the image as the input to a classifier. The subsequent layers are also a collection of Fourier coefficients. And the connections of our network ensure that the maps between adjacent layers are S^1 equivariant maps. The subsequent layers of the classifier are so connected that if the image is rotated by an angle θ, the feature at a neuron of type n gets multiplied by $e^{in\theta}$. A classifier, which is SO(2) equivariant performs well, without augmentation, when the train set is unrotated, and the test set is rotated on MNIST dataset.(Ref [3])

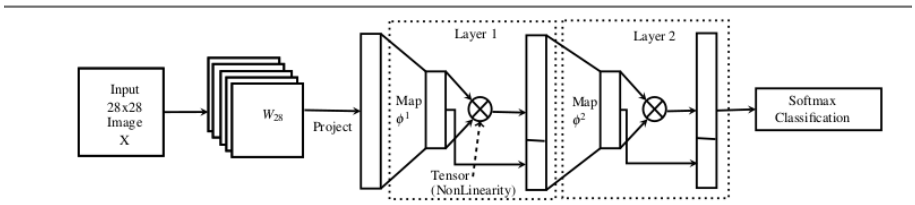


Figure 1.2: SO(2)-Equivariant NN

Chapter 2

Robustness of SO(2)-NN Classifier

State-of-the-art classifiers, especially deep networks, have shown impressive classification performance on many challenging benchmarks in computer vision tasks. An equally important property of a classifier that is often overlooked is it's robustness when data samples are perturbed by noise. The robustness of a classifier is especially fundamental when it is deployed in real-world, uncontrolled, and possibly hostile environments. In such cases it is important for classifiers to show good robustness properties. In other words, a sufficiently small perturbation of a data-point should ideally not result in altering the estimated label of a classifier.State-of-the-art deep neural networks have recently been shown to be very unstable to adversarial perturbations of the data. In particular, despite the highly accurate performances of these classifiers, well-sought perturbations of the data can easily cause it to wrongly classify the test images, since data points often lie very close to the decision boundary of the classifier.

In this chapter we test the robustness of the SO(2) classifier under adversarial attacks on the MNIST dataset.For the experiments we train the classifier on MNIST dataset and test it on MNIST with different types of adversarial noise(FGSM,PGD).Then for the next set of experiments we train the classifier on images with adversarial noise as well and then check how well it classifies test images with adversarial noise.

2.1 Adversarial attacks and Robust Classifiers

Adversarial attacks on neural network models are certain, deliberate changes to inputs that causes a highly accurate model to wrongly classify the image but these images are unlikely to fool humans.The modified image is called an adversarial image, and when submitted to a classifier is misclassified, while the original one is correctly classified.

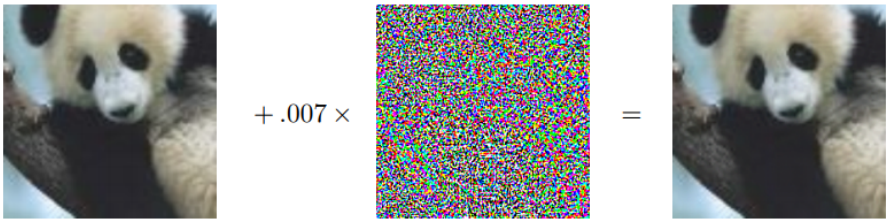


Figure 2.1: Example of adversarial attack

A classifier is called **Robust** if a sufficiently small perturbation of a data-point does not result in altering the estimated label of a classifier.In other words adding adversarial noise to test images should not drastically change the accuracy of the classifier.

2.2 Some Common Attacks

Most successful attacks are gradient-based methods. Namely the attackers modify the image in the direction of the gradient of the loss function with respect to the input image.We use two such attacks for this chapter.

2.2.1 Fast Gradient Step Method (FGSM)

Goodfellow et al. in the paper "Towards Explaining and Harnessing Adversarial Examples"(Ref [2]) propose Fast Gradient Sign Method (FGSM) that adversarially perturbs an input image x in the following manner

$$x' = x + sign(\nabla_x J(\theta, x, y))$$

. Here $J(\theta, x, y)$ is the loss function used to train the network, x is the input and y is the target label and θ are the model parameters.

2.2.2 Projected Gradient Descent Method(PGD)

This method is a multi-step variant of FGSM. Madry et al. in the paper "Towards Deep Learning Models Resistant to Adversarial Attacks"(Ref [1]) introduced this method. Given any model, these attacks produce adversarial perturbation for every test image x from a small ℓ_∞ -ball around it, namely, each pixel value x_i is perturbed within $[x_i - \epsilon, x_i + \epsilon]$. PGD attack does so by solving an inner optimization by projected gradient descent over ℓ_∞-ball of radius ϵ around x, to approximate the optimal perturbation.
In other words we are taking a step in the direction of the gradient of the loss function and then projecting back into the set.We find the closest point in our set to the projected point and keep repeating the process.Adversarial training with PGD perturbations improves the adversarial robustness of models and it is one of the best known defenses to make models robust to perturbations of bounded ℓ_∞ norm on MNIST and CIFAR-10 datasets.

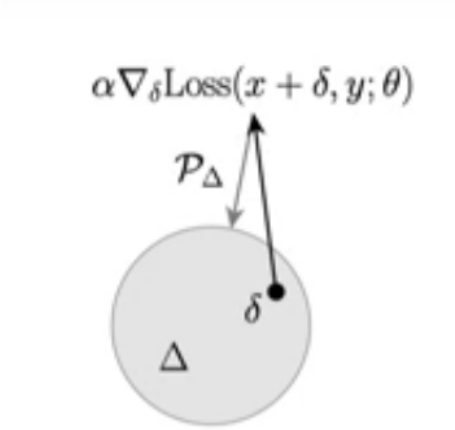


Figure 2.2: Projected Gradient Descent

2.3 Experiments

All the adversarial examples that were we generated for these experiments were generated from the CNN described in github repository of the paper "Towards Deep Learning Models

Resistant towards adversarial attacks"([1] in references).The convolutional neural network consists of two convolutional layers (each followed by max-pooling) and a fully connected layer.

2.3.1 Experiment 1

Experiment:

For the first experiment we train the SO(2) classifier on standard upright MNIST data and then test it on MNIST test dataset with added PGD noise.We choose two values of ϵ which are 0.3 and 0.5.

Observation:

The SO(2) classifier was able to classify the test images correctly with accuracy 10.8 percent with $\epsilon = 0.3$ and 11.07 percent with $\epsilon = 0.5$. The standard CNN gives an accuracy of 0.00 percent on this PGD test data in both cases.

Inference:

When trained on standard MNIST training dataset SO(2) classifier does a poor job in classifying MNIST test images with PGD noise added.It's accuracy is almost comparable to random guessing.

2.3.2 Experiment 2

Experiment:

For this experiment we train the SO(2) classifier on MNIST training data with added PGD noise and then test it on MNIST test dataset with added PGD noise.We choose two values of ϵ which are 0.3 and 0.5

Observation:

The SO(2) classifier was able to classify the test images correctly with accuracy 97.05 percent with $\epsilon = 0.3$ and 96.73 percent with $\epsilon = 0.5$. On the other hand adversarially trained CNN is able to classify with accuracy 96.64 percent and 95.4 percent.

Inference:

When trained on MNIST train dataset with PGD noise added, SO(2) classifier is able to classify PGD added test images with very high accuracy.

2.3.3 Experiment 3

Experiment:

For this experiment we train the SO(2) classifier on standard upright MNIST data with added PGD noise and then test it on rotated MNIST test dataset with added PGD noise.We choose two values of ϵ which are 0.3 and 0.5.

Observation:

The SO(2) classifier was able to classify the test images correctly with accuracy 88.89 percent with $\epsilon = 0.3$ and 85.5 percent with $\epsilon = 0.5$. On standard CNN it is 10.29 percent and 9.86 percent respectively.

Inference:

When trained on MNIST train dataset with PGD, SO(2) classifier is able to classify PGD added test images with rotation with reasonable accuracy.

Table 2.1: Robustness under PGD attack

Dimension	ϵ	train:std,test:adv	train:adv,test:adv	train:adv,test:rot-adv
`CNN`	0.3	0.00	96.64	10.29
`CNN`	0.5	0.00	95.4	9.86
`SO(2)-Classifier`	0.3	10.8	97.05	88.89
`SO(2)-Classifier`	0.5	11.7	96.73	85.5

2.3.4 Experiment 4

Experiment:

For this experiment we train the SO(2) classifier on standard upright MNIST data and then test it on MNIST test dataset with added FGSM noise.We choose two values of ϵ which are 0.3 and 0.5

Observation:

The SO(2) classifier was able to classify the test images correctly with accuracy 11.10 percent with $\epsilon = 0.3$ and 10.7 percent with $\epsilon = 0.5$. The standard CNN gives accuracy 10.05 percent and 3.15 percent.

Inference:

When trained on standard MNIST train dataset SO(2) classifier does a poor job in classifying MNIST test images with FGSM noise added.It's accuracy is almost comparable to random guessing.

2.3.5 Experiment 5

Experiment:

For this experiment we train the SO(2) classifier on upright MNIST data with added FGSM noise and then test it on MNIST test dataset with added FGSM noise.We choose two values of ϵ which are 0.3 and 0.5.

Observation:

The SO(2) classifier was able to classify the test images correctly with accuracy 95.34 percent with $\epsilon = 0.3$ and 94.26 percent with $\epsilon = 0.5$. The standard CNN gives accuracy 97.8 percent and 96.6 percent.

Inference:

When trained on MNIST train dataset with FGSM noise SO(2) classifier is able to classify FGSM added test images with very high accuracy.

2.3.6 Experiment 6

Experiment:

For this experiment we train the SO(2) classifier on standard upright MNIST data with added FGSM noise and then test it on rotated MNIST test dataset with added FGSM noise.We choose two values of ϵ which are 0.3 and 0.5.

Observation:

The SO(2) classifier was able to classify the test images correctly with accuracy 85.56 percent with $\epsilon = 0.3$ and 83.52 percent with $\epsilon = 0.5$. On standard CNN it is 11.29 percent and 12.25 percent respectively.

Inference:

When trained on MNIST train dataset with FGSM, SO(2) classifier is able to classify FGSM added test images with rotation with reasonable accuracy.It does far better than standard CNN.

Table 2.2: Robustness under FGSM attack

Dimension	ϵ	train:std,test:adv	train:adv,test:adv	train:adv,test:rot-adv
`CNN`	0.3	10.5	97.8	11.29
`CNN`	0.5	3.15	96.6	12.25
`SO(2)-Classifier`	0.3	11.1	95.34	85.56
`SO(2)-Classifier`	0.5	10.7	94.26	86.52

2.4 Conclusion

Form the above experiments we can conclude that when trained on standard MNIST dataset the SO(2)-equivariant neural network does a poor job in classifying adversarial test images.Much like standard CNNs the SO(2) Equivariant Neural network can also be made resistant to such attacks by training it on adversarial images as evident from the high accuracy of our experiments.Also when rotated MNIST dataset is used as test set with added adversarial noise the SO(2) equivariant Neural Network is able to classify it with reasonably high accuracy which is a huge improvement over Standard CNNs.

Chapter 3

Explainability of penultimate layer of CNN

3.1 Motivation:

We know that CNNs do very well in image classification tasks. So a natural question to ask is what the layers are learning. It is known that the first few layers learn some simple filters, which are common across most most data sets. But the deeper layers learn more sophisticated features of the images. In this chapter we look at the features learned by a CNN in the penultimate layer. We ask if if can use those features in a simple way - For each class we collect the features learned in the penultimate layer and compute the top singular feature vector for each class. For test sample we compute the feature vector we get in the penultimate layer and compute the closest one dimensional singular feature vector and find that this too works as a good classifier with a minor drop in accuracy.

3.2 Singular Value Decomposition:

In linear algebra, the singular value decomposition (SVD) is a factorization of a real or complex matrix.

SVD states that any matrix A can be factorized as:

$$A = USV^T$$

$$\underset{m \times n}{\mathrm{A}} = \underset{m \times m}{\mathrm{U}} \; \underset{m \times n}{\mathrm{S}} \; \underset{n \times n}{\mathrm{V^T}}$$

$$\begin{pmatrix} x_{11} & x_{12} & & x_{1n} \\ & & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix} = \begin{pmatrix} u_{11} & & u_{m1} \\ & \ddots & \\ u_{1m} & & u_{mm} \end{pmatrix} \begin{pmatrix} \sigma_1 & \ddots & 0 \\ & \sigma_r & \ddots \\ 0 & & 0 \end{pmatrix} \begin{pmatrix} v_{11} & & v_{1n} \\ & \ddots & \\ v_{n1} & & v_{nn} \end{pmatrix}$$

such that:

- The matrices U,S and V are unique
- U and V are column orthonormal.
  - $U^TU = V^TV = I$
  - columns are orthogonal unit vectors
- S is a diagonal matrix with positive entries(singular values) and sorted in descending order

$$\begin{pmatrix} \sqrt{\lambda_1} & & & & \\ & \sqrt{\lambda_2} & & & \\ & & \ddots & & \\ & & & \sqrt{\lambda_r} & \\ & & & & \ddots \; 0 \end{pmatrix} \equiv \begin{pmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_r & \\ & & & & \ddots \; 0 \end{pmatrix}$$

S

σ_2: singular value

The singular value decomposition finds the best-fitting k-dimensional subspace for $k = 1, 2, 3, ...$, for a set of n data points. Here, "best" means minimizing the sum of the squares of the perpendicular distances of the points to the subspace, or equivalently, maximizing the sum of squares of the lengths of the projections of the points onto this subspace.Also the top singular value is the best one dimensional approximation to the data, the top two, are the best 2-d plane approximating the data.

3.2.1 SVD for Dimensionality Reduction:

A popular application of SVD is for dimensionality reduction. Data with a large number of features, such as more features (columns) than observations (rows) may be reduced to a smaller subset of features that are most relevant to the problem we want to predict. The result is a matrix with a lower rank that is a close approximation the original matrix. To do this we can perform an SVD operation on the original data and select the top k largest singular values in S. These columns can be selected from S and the rows selected from V^T. An approximate B of the original vector A can then be reconstructed.

$$B = USV^T[k]$$

The truncated SVD to A can be found by setting all but the first k largest singular values equal to zero and using only the first k columns of U and V.
Theorem: For any other matrix C of rank at most k: $||A - B|| \leq ||A - C||$ where $||.||$ is the Frobenius norm. (Ref [4])

3.3 Using SVD for predicting the Final output of a Neural Network:

For this problem we use SVD to predict the output of the final layer of a Neural Network on different datasets.
The detailed flowchart for the algorithm is given below:

- We select a dataset(divided into train and test sets) and a Neural Network Classifier(either CNN or SO(2)-classifier)
- We train the chosen classifier using the training data to a considerable accuracy.
- We pass each element of the training data though the trained classifier and save the output given by the penultimate layer.

- We collect the output of all the elements of the training data and according to their classes we store them in separate arrays.
- We treat each of these arrays containing the output of the penultimate layer for elements of each classes as a matrix and we do SVD on them with number of components n.So now we have n SVD vectors for each class.
- We take the output of the penultimate layer for each element of the test set and we take the matrix multiplication of this feature vector with each of the SVD vectors of each class and take the L^2 norm of the resulting one-dimensional vector.
- Whichever SVD vector gives the highest L^2 norm we classify the test set element as the class of that SVD vector.

3.4 Experiments:

We implemented the algorithm described above and ran it on three different datasets MNIST,Fashion-MNIST and CIFAR-10.The classifiers that we used are Convolutional Neural Network and SO(2) classifier.

3.4.1 CNN on MNIST

:

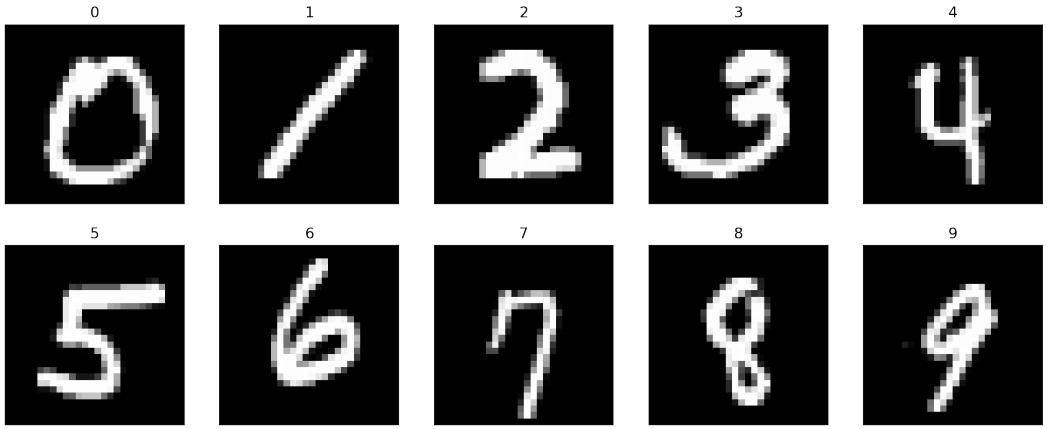


Figure 3.1: images/Sample MNIST images

Experiment:

A convolutional neural network was taken and it was trained on the training set of the MNIST dataset.The detailed architecture of the CNN is given below.
Upon training it achieved 98.02 percent accuracy on the test set.Then as described above we took the penultimate layer output for each image of both the train set and the test set. The output of the penultimate layer for each of the classes was then stored in an array and then we reduce the dimension of this array by using SVD with number of components $n = 1, 2, 50, 100$.
Then the dot product of each of these dimension reduced vectors was taken with the penultimate layer output of the test images and the product corresponding to whichever class gives the maximum value,we classify the test image to be of that class.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2359552
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 2,380,938
Trainable params: 2,380,938
Non-trainable params: 0
```

Figure 3.2: Description of the CNN used

Observation:

Using the algorithm described above we are able to correctly predict the test labels with 97.26 percent accuracy with n = 1,97.31 with $n = 2$ and 97.33 with $n = 50$ and 97.33 with $n = 100$.

Inference:

SVD is able to predict with accuracy comparable to the CNN on MNIST dataset.The features of the penultimate layer of a well trained CNN can be used as simple linear separators achieving a reasonable accuracy on the MNIST dataset.

Here is a pictorial representation of the SVD output of the penultimate layers for each class :

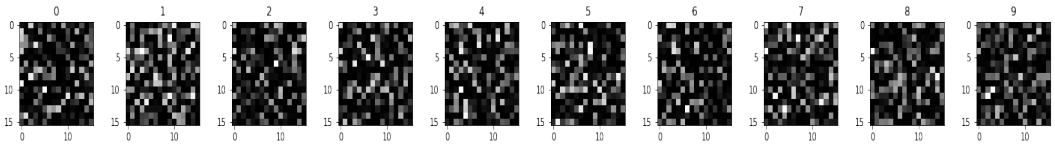


Figure 3.3: SVD vectors for each class

3.4.2 CNN on fashion MNIST

:

Figure 3.4: Sample fashion MNIST images

Experiment:

A convolutional neural network was taken and it was trained on the training set of the fashion MNIST dataset.The detailed architecture of the CNN is given below.

Upon training it achieved 90.4 percent accuracy on the test set.Then as described above we took the penultimate layer output for each image of both the train set and the test set. The output of the penultimate layer for each of the classes was then stored in an array and then we reduce the dimension of this array by using SVD with number of components $n = 1, 2, 50, 100$.

Then the dot product of each of these dimension reduced vectors was taken with the penultimate layer output of the test images and the product corresponding to whichever class gives the maximum value,we classify the test image to be of that class.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2359552
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 2,380,938
Trainable params: 2,380,938
Non-trainable params: 0
```

Figure 3.5: Description of the CNN used

Observation:

Using the algorithm described above we are able to correctly predict the test labels with 79.74 percent accuracy with n = 1,79.92 with n = 2 and 79.99 with n = 50 and 79.99 with $n = 100$.

Inference:

The features of the penultimate layer of a well trained CNN can be used as simple linear separators achieving a reasonable accuracy - however there is a significant drop, of about

10 percent in the accuracy, as compared to the original CNN in the Fashion MNIST dataset.

3.4.3 CNN on CIFAR 10

:

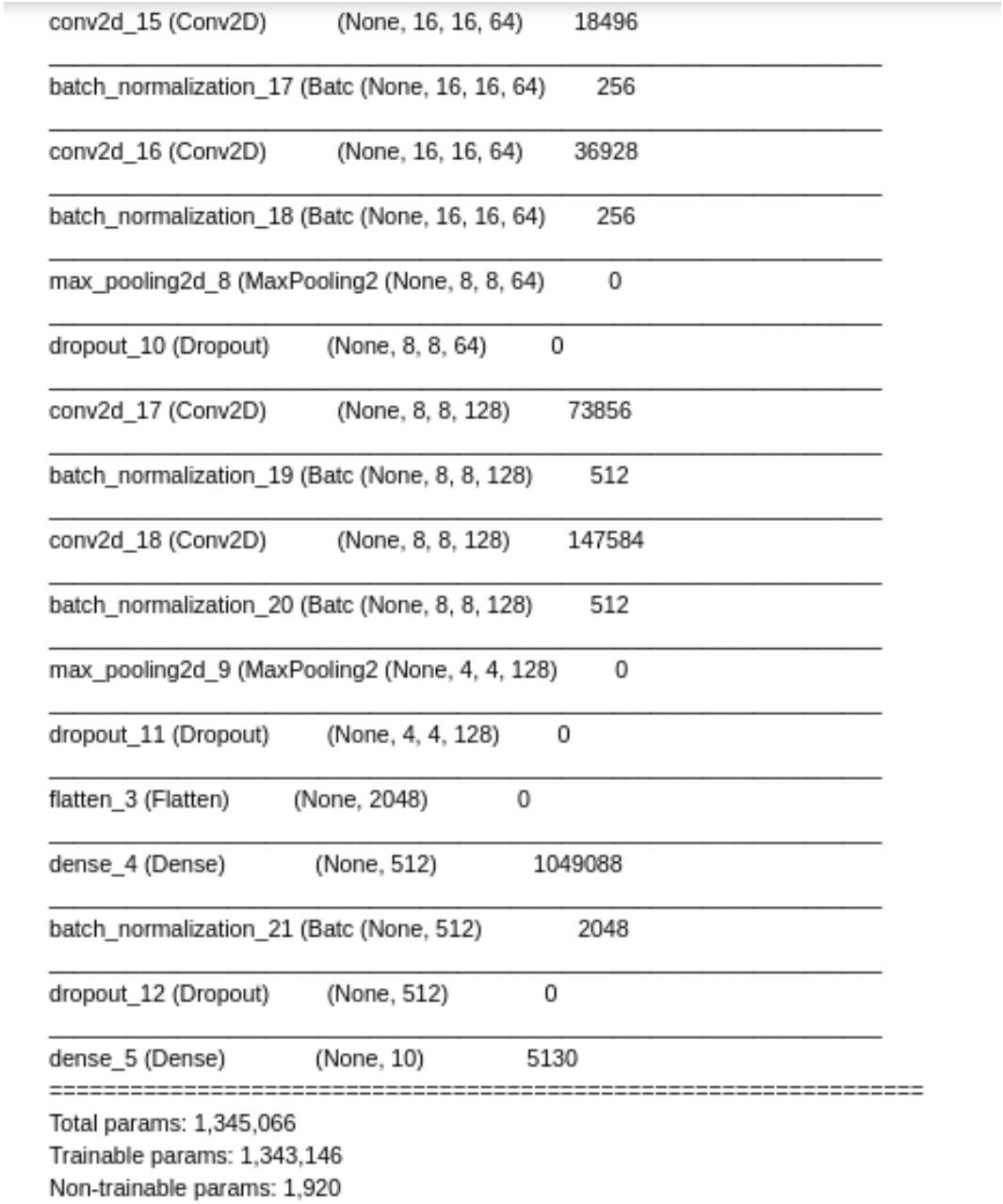
Figure 3.6: Sample CIFAR 10 images

Experiment:

A convolutional neural network was taken and it was trained on the training set of the CIFAR-10 dataset.The detailed architecture of the CNN is given below.

Upon training it achieved 80.71 percent accuracy on the test set.Then as described above we took the penultimate layer output for each image of both the train set and the test set. The output of the penultimate layer for each of the classes was then stored in an array and then we reduce the dimension of this array by using SVD with number of components $n = 1, 2, 50, 100$.

Then the dot product of each of these dimension reduced vectors was taken with the penultimate layer output of the test images and the product corresponding to whichever class gives the maximum value,we classify the test image to be of that class.



conv2d_15 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_17 (Batc	(None, 16, 16, 64)	256
conv2d_16 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_18 (Batc	(None, 16, 16, 64)	256
max_pooling2d_8 (MaxPooling2	(None, 8, 8, 64)	0
dropout_10 (Dropout)	(None, 8, 8, 64)	0
conv2d_17 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_19 (Batc	(None, 8, 8, 128)	512
conv2d_18 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_20 (Batc	(None, 8, 8, 128)	512
max_pooling2d_9 (MaxPooling2	(None, 4, 4, 128)	0
dropout_11 (Dropout)	(None, 4, 4, 128)	0
flatten_3 (Flatten)	(None, 2048)	0
dense_4 (Dense)	(None, 512)	1049088
batch_normalization_21 (Batc	(None, 512)	2048
dropout_12 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 10)	5130

Total params: 1,345,066
Trainable params: 1,343,146
Non-trainable params: 1,920

Figure 3.7: Description of the CNN used

Observation:

Using the algorithm described above we are able to correctly predict the test labels with 77.84 percent accuracy with n = 1,77.84 with n = 2 and 77.91 with n = 50 and 77.91 with n = 100.

Inference:

The features of the penultimate layer of a well trained CNN can be used as simple linear separators achieving a reasonable accuracy - however there is a drop, of about 4 percent in the accuracy, as compared to the original CNN on the CIFAR 10 dataset.

Table 3.1: Accuracy for different dimensions of SVD on CNN

Dimension	MNIST	Fashion MNIST	CIFAR-10
n=1	97.26	79.74	77.84
n=2	97.31	79.92	77.84
n=50	97.33	79.92	77.91
n=100	97.33	79.99	77.91
w/o SVD	98.02	90.4	80.71

3.4.4 SO(2) Equivariant Neural Network on MNIST

:

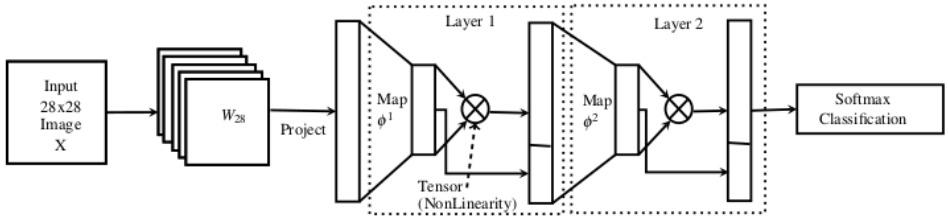


Figure 3.8: Structure of the SO(2) Equivariant Neural Network

Experiment:

We took the SO(2) classifier and trained it on the MNIST dataset achieving an accuracy of 98.84 percent.Then as described above we took the penultimate layer output for each image of both the train set and the test set. The output of the penultimate layer for each of the classes was then stored in an array and then we reduce the dimension of this array by using SVD with number of components $n = 1, 2, 50, 100$.

Then the dot product of each of these dimension reduced vectors was taken with the penultimate layer output of the test images and the product corresponding to whichever class gives the maximum value,we classify the test image to be of that class.

Observation:

Using the algorithm described above we are able to correctly predict the test labels with 94.42 percent accuracy with n = 1,94.47 with $n = 2$ and 94.48 percent with $n = 50$ and 94.48 with $n = 100$.

Table 3.2: Accuracy for different dimensions of SVD on SO(2)

Dimension	MNIST
`n=1`	94.42
`n=2`	94.47
`n=50`	94.48
`n=100`	94.48
`w/o SVD`	98.02

Inference:

SVD is able to predict with accuracy comparable to the SO(2) classifier on MNIST dataset.The features of the penultimate layer of a well trained SO(2) classifier can be used as simple linear separators achieving a reasonable accuracy on the MNIST dataset.

Chapter 4

Summary,Future Work and References

4.1 Summary

So in the first part of this project we tested the robustness of the SO(2) equivariant Neural Network.There we saw that it performs poorly against images with added PGD and FGSM noise.But much like Convolutional Neural Networks this can be resolved by training the classifier on these adversarial images and it can achieve nearly state of the art accuracy on the MNIST dataset.Also when the test set contains MNIST images with rotations along with added noise then the SO(2) equivariant NN performs significantly better than standard CNN.

In the second part of the project we address the question what are the final layers of a CNN and SO(2) equivariant NN learning?So the accumulate final layer outputs of the images in the training set and do SVD on them with number of components = 1,2,50,100.We show that these SVD vectors provide a reasonably accurate estimate of how the final layer of the network behaves and we use it to do classification tasks with just a minor drop in accuracy.

4.2 Future Work:

Some future work in this direction might be to test the robustness of the SO(2) Equivariant Neural Network against other adversarial attacks like DeepFool,JSMA etc.

We can try to use the SVD method proposed here on all the layers of a neural network including Convolutional and Max pooling layers and calculate it's accuracy for classification tasks.

4.3 References:

[1] Aleksander Madry et al - Towards Deep Learning Models Resistant to Adversarial attacks,ICLR 2018

[2] Goodfellow et al - Explaining and Harnessing Adversarial Examples,ICLR 2014

[3] Muthuvel Murugan, KV Subrahmanyam -SO(2)-Equivariance in Neural networks using tensor nonlinearity,BMVC 2019

[4] Blum,Hopcroft and Kannan - Foundations of Data Science,2013

[5] https://cs231n.github.io/convolutional-networks/