INTERNSHIP REPORT

Sarosij Bose

Summer Intern at LISA Lab, IISc

1st June - 2nd August, 2021

UNDER THE GUIDANCE OF

Kunal Narayan Chaudhury

Associate Professor

Department of Electrical Engineering

Indian Institute of Science, Bangalore

Submitted on: 2nd August, 2021.

Contents

Motivation

Fazlyab et al. first introduced a novel technique where the non-linear activation functions present in any Deep neural network could be regulated to obtain a very tight upper (almost exact) Lipschitz bound. However, this approach had some drawbacks:-

- SDPs are known to be computationally expensive and non-industry scalable. Hence, questions arise about it’s real-world usefulness.
- Experimental results are shown only for very limited and shallow fully connected networks on a dataset like MNIST where images are small and pre-processed.

As a result of the combination of these two points, the work scales poorly for deep networks and can’t be used for other architectures like CNNs.

We attempt to understand and try to tackle these problems step-by-step. Hence, we extensively tested and reported our observations based on the results obtained from different types of fully connected networks. Next, we show how to convert a CNN into a fully connected network and show our observations and results on these converted CNNs.

These extensions of the original work including a novel implementation gives us a very robust idea of the exact limits to which the proposed work can be stretched and generalized to other networks (here CNNs). We also discuss about the conversion from Convolutional neural networks to Fully connected networks, the methods present and which one is better for conversion with the pros and cons of both.

In the exact same year, a team of researchers from Google also attempted to solve this same problem but for CNNs exclusively. The proposed method could produce a range of singular values for each individual convolutional layer.

However, again this method had some drawbacks:-

- The researchers do not account for any non-linearity present in the network like simple activation functions.
- As a result, it was able to produce only trivial upper Lipschitz bounds obtained by taking the product of the maximum of the range of singular values produced for each individual layer.

We can see that the above two approaches have their own set of advantages and disadvantages. On combination however, we can try to create an algorithm that can give us a tight Lipschitz value without the accompanying excessive computational costs.

Such an implementation would allow us to build a real-time algorithm which could be implemented on real-world images and not small, curated images from datasets like MNIST.

We also briefly discuss about the various other applications where Lipschitz regularization is extensively used like in Control Theory, Reinforcement Learning and GANs. Then, we show a possible research direction for one of these applications in brief.

Finally, we bring together our experimental efforts all into one place: we plot the empirical values and observe how they tend to saturate much below both the trivial and tight bounds. This provides us a graphical perspective of how much still needs to be done and how close we are really to the exact Lipschitz constant.

Methodology

Reproducibility of LipSDP results for FCNs

Here, the change of the ratio of the Trivial and Non-trivial Lipschitz bounds are explored with respect of various changes in the network. The various parameters being experimented with are as given below:-

- Number of hidden fc layers.
- No. of neurons per layer.

The primary motivation here is to observe the behaviour of fully connected networks by trying to reproduce their results by changing network parameters.

The tables below show the results obtained on the various networks and their corresponding parameters used.

Table 1:-

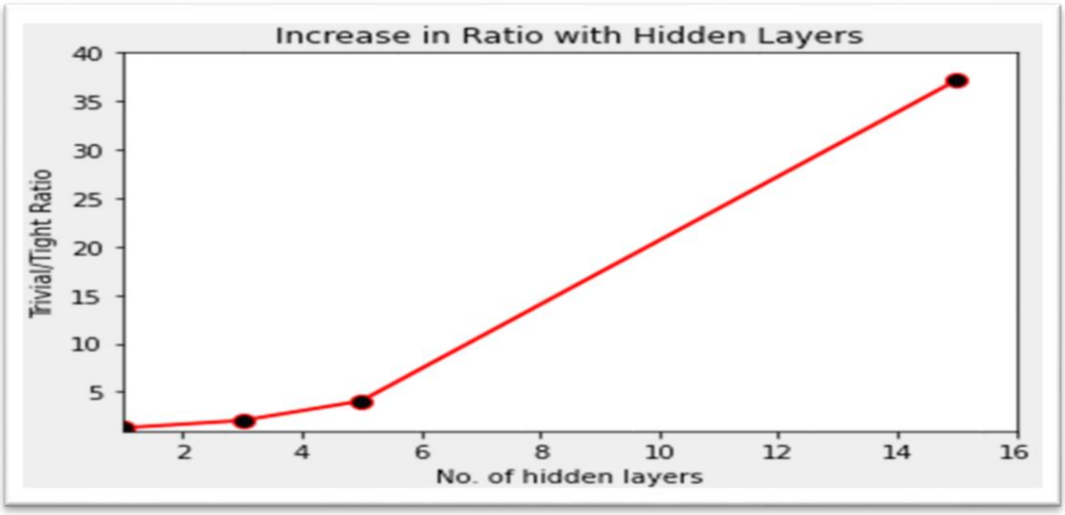
- **Model Information = 64 units per hidden layer. [784, L1, L2, …., 10]**
- **Model Test accuracy: 96.9 % (Avg. test accuracy taken of all 3 cases as given below.)**

SL No.	No. of Hidden layers L(i)	Trivial Bound	Tight Bound	Tight Bound Type, Split Size	Ratio (Trivial/Tight)
1.	3	1081.011	406.003	Neuron, None	2.66
2.	5	10675.460	2619.196	Neuron, None	4.07
3.	15	19360863876.347	520644601.471	Neuron, 3	37.18

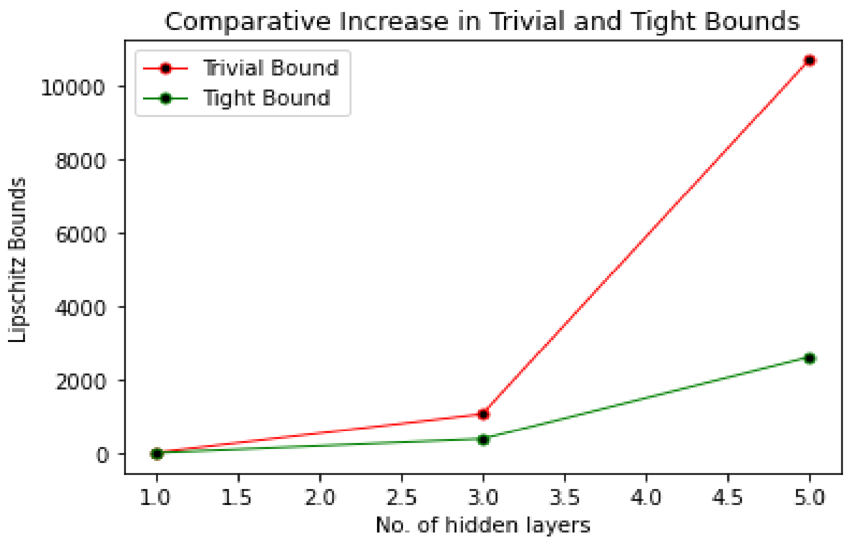
ADDITIONAL NOTES:-

1. Tight Bound results for SL No. 3 with split size 2 and 4 are 1582867748.892 and 596179800.627 corresponding to ratios of 12.23 and 32.47. This is a bit unexpected since the ratio with split size 4 is expected to be higher than that of 3.
2. Tight Bound results for SL No. 3 with split size 5 returns an incorrect value of ‘inf’ showing that either the model has scalability issues or some issues arose with the solver.

Following are some graphs putting Table 1 results into perspective.

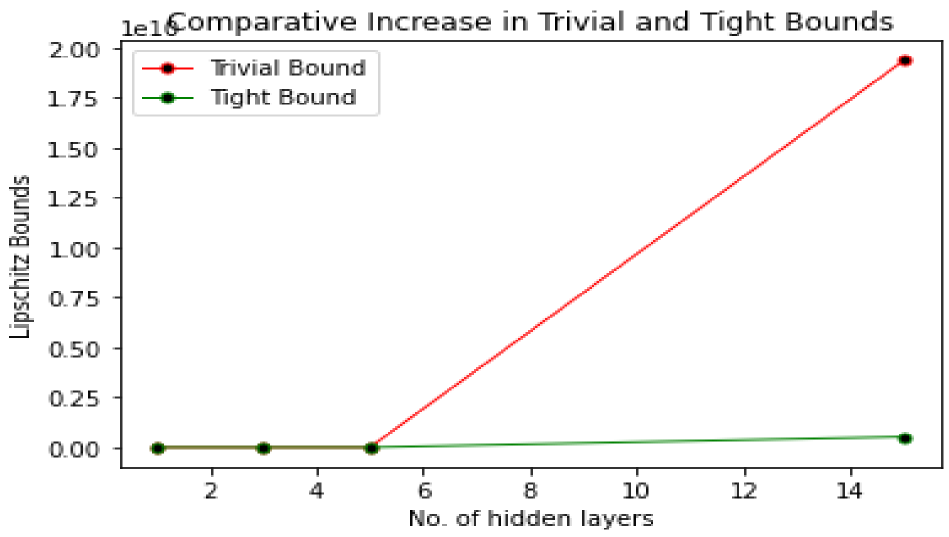


Graph 1: Graph of the variation in Ratio with no. of hidden layers.



Graph 2: Graph of variation in Trivial and Tight Bounds with increase in hidden layers

The above graph omits the last reading (SL No. 3) in Table 1. The graph including that reading is given below in **Graph 3** where it can be seen how the first three readings (as in **Graph 1**) become completely squashed below due to the exploding trivial Lipschitz bound for 15 hidden layers**.**



Graph 3: Graph including the last reading for 15 hidden layers. The first 3 readings are completely flattened showing the magnitude in increase.

Table 2:-

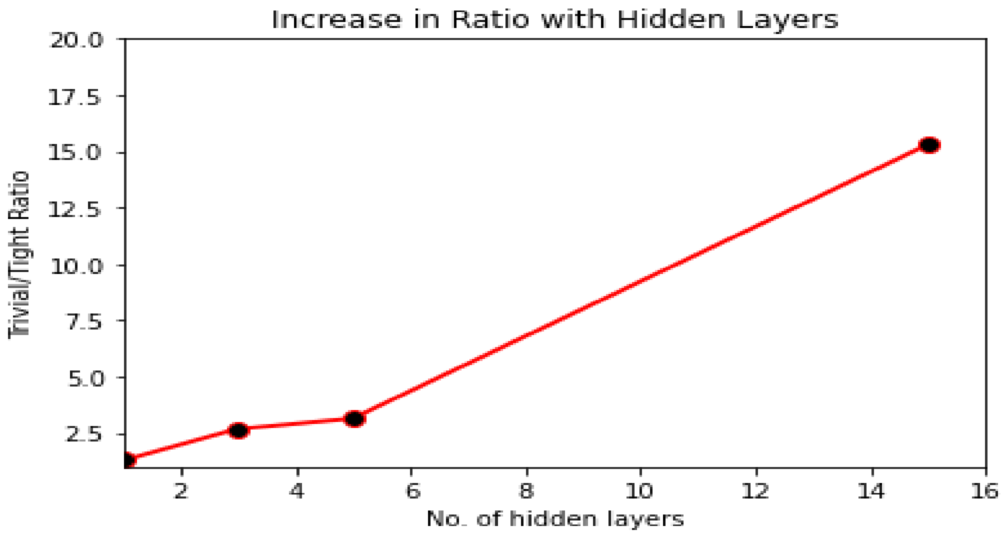
- **Model Information = 128 units per hidden layer.[784, L1,L2…, 64, 10]**
- **Model Test Accuracy = 97.2% (Avg accuracy taken over all the 3 models below combined).**

SL No.	**No. of Hidden layers L(i)**	**Trivial Bound**	**Tight Bound**	**Tight Bound Type, Split Size**	**Ratio (Trivial/Tight)**
1.	3	2778.205	1035.224	Neuron, None	2.68
2.	5	60621.952	19197.798	Neuron, 3	3.15
3.	10	450507912.597	29412230.181	Neuron, 2	15.31

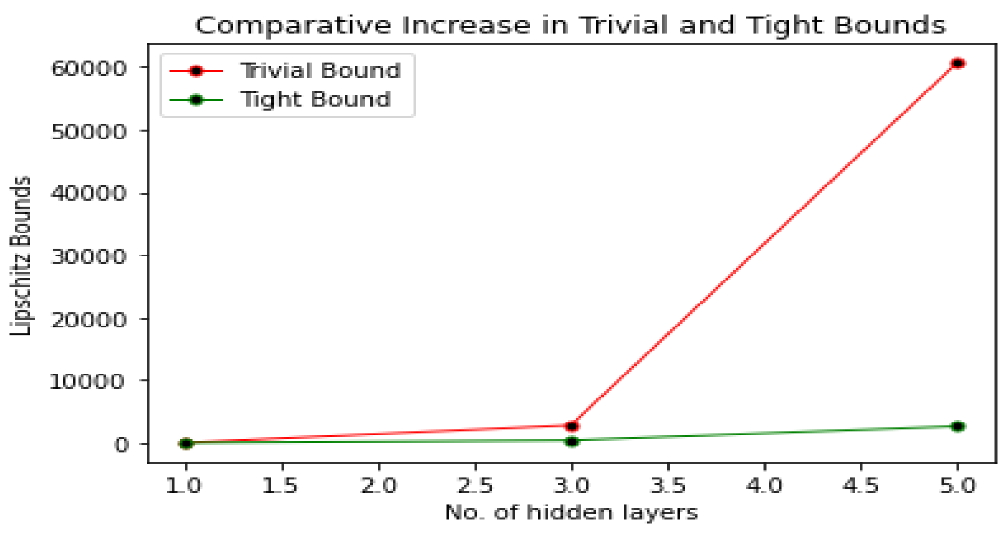
ADDITIONAL NOTES:-

1. Experiments with a 15-hidden layer network could'nt be carried out due to RAM constraints.
2. On running with higher split values for SL No. 3, the LipSDP program simply failed to return any value at all. Higher ratio would have been yielded with a subnetwork of size say 5 than 2.

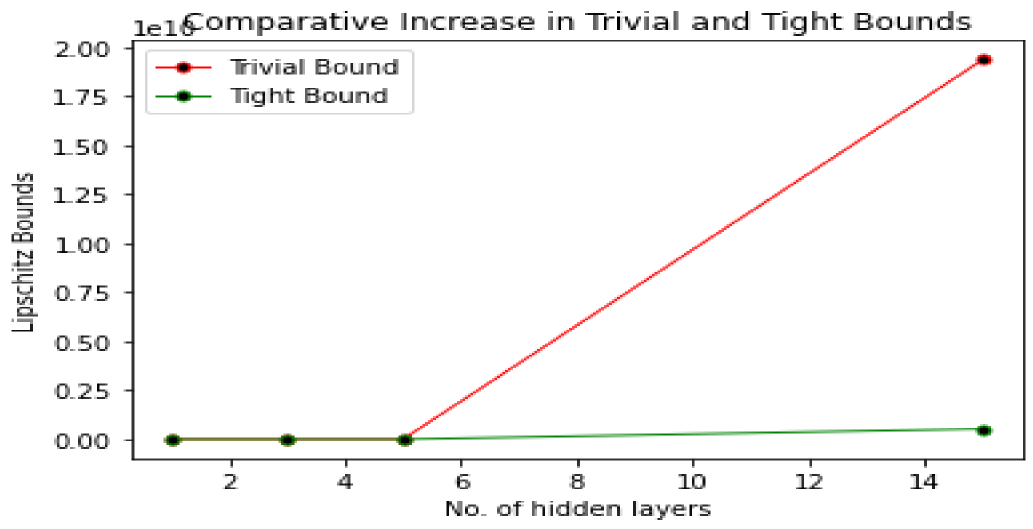
Following are some graphs putting Table 2 results into perspective. The same pattern has been followed as above which is: increase in ratio's, graph without the last reading and graph with the last reading.



Graph 4: Increase in trivial/tight ratio with number of layers.



Graph 5: Graph of variation in Trivial and Tight Bounds with increase in hidden layers.



Graph 6: With the last reading included into the graph. This graph looks extremely similar to Graph 5 above.

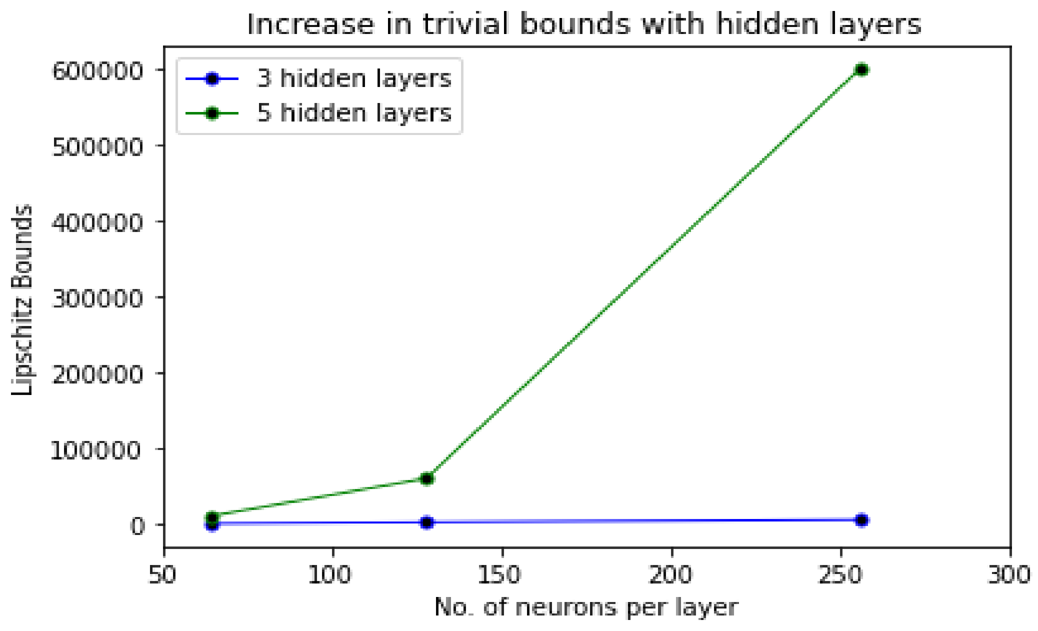
<u>Table 3:-</u>

- **Model Information = 256 units per hidden layer. [784, L1, L2, ….., 128, 64, 10]**
- **Model Accuracy = 97.25% (Avg. accuracy of the 2 models below combined.)**

SL No.	No. of hidden layers L(i)	Trivial Bound	Tight Bound	Tight Bound Type/Split Size	Ratio (Trivial/Tight)
1.	3	6225.828	3206.889	Neuron, None	1.94
2.	5	599695.715	NA	NA	NA

ADDITIONAL NOTES:-

1. Tight bound for 5 hidden layer network could not be found out due to memory constraints.



Graph 7: Increase of Trivial Lipschitz bounds with no of units per hidden layer.

Conversion of CNNs to FCNs

There are 2 main methods of converting a CNN into a fully-connected network:-

- Unrolling convolution [3]
- Creation of Toeplitz matrices [4]

Out of these two, the latter approach was chosen here.

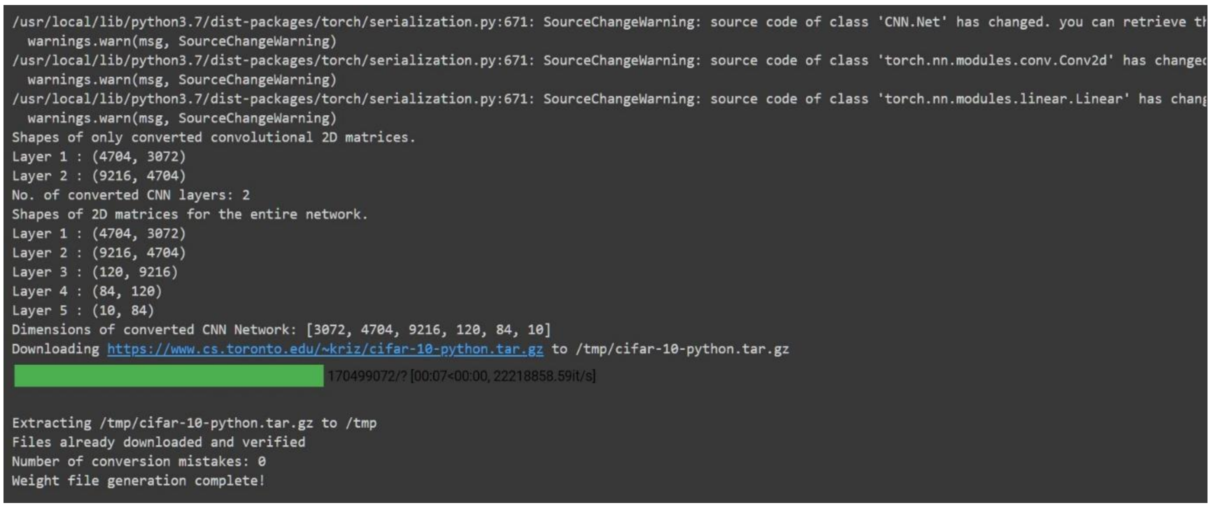
The main intuition behind both these approaches are finding a method by which the convolution operation can be expressed in a matrix-matrix operation format. This representation allows us to treat convolution in a manner represented by the following formula:-

$$x^0 = x, \quad x^{k+1} = \phi(W^k x^k + b^k) \text{ for } k = 0, \cdots, \ell - 1, \quad f(x) = W^\ell x^\ell + b^\ell.$$

The above formula shows the mathematical representation for a typical feed-forward network consisting of l layers.

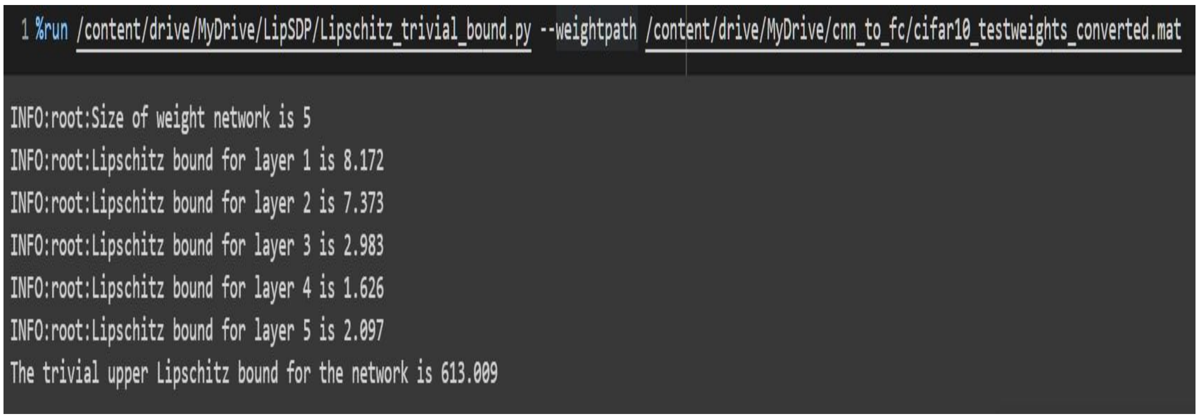
The figure given in [4] further provides an excellent illustration of the entire conversion in a graphic manner.

After conversion of the CNN, the model is passed on as saved weights to obtain the tight and trivial bounds like any normal fully connected network.



```
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:671: SourceChangeWarning: source code of class 'CNN.Net' has changed. you can retrieve th
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:671: SourceChangeWarning: source code of class 'torch.nn.modules.conv.Conv2d' has changed
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:671: SourceChangeWarning: source code of class 'torch.nn.modules.linear.Linear' has chang
  warnings.warn(msg, SourceChangeWarning)
Shapes of only converted convolutional 2D matrices.
Layer 1 : (4704, 3072)
Layer 2 : (9216, 4704)
No. of converted CNN layers: 2
Shapes of 2D matrices for the entire network.
Layer 1 : (4704, 3072)
Layer 2 : (9216, 4704)
Layer 3 : (120, 9216)
Layer 4 : (84, 120)
Layer 5 : (10, 84)
Dimensions of converted CNN Network: [3072, 4704, 9216, 120, 84, 10]
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to /tmp/cifar-10-python.tar.gz
 170499072/? [00:07<00:00, 22218858.59it/s]

Extracting /tmp/cifar-10-python.tar.gz to /tmp
Files already downloaded and verified
Number of conversion mistakes: 0
Weight file generation complete!
```

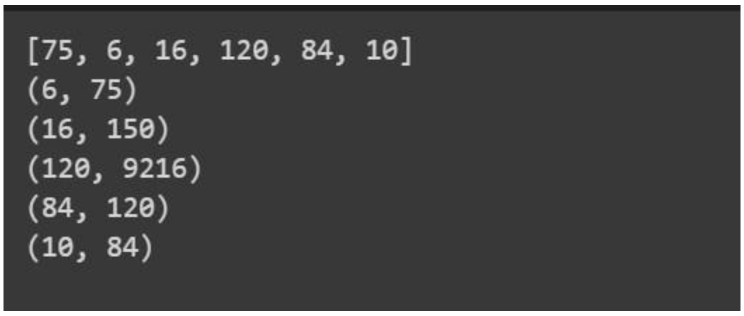
Fig 1: Conversion of the CNN into a fully connected network. All the converted shapes and layers are shown.



```
1 %run /content/drive/MyDrive/LipSDP/Lipschitz_trivial_bound.py --weightpath /content/drive/MyDrive/cnn_to_fc/cifar10_testweights_converted.mat

INFO:root:Size of weight network is 5
INFO:root:Lipschitz bound for layer 1 is 8.172
INFO:root:Lipschitz bound for layer 2 is 7.373
INFO:root:Lipschitz bound for layer 3 is 2.983
INFO:root:Lipschitz bound for layer 4 is 1.626
INFO:root:Lipschitz bound for layer 5 is 2.097
The trivial upper Lipschitz bound for the network is 613.009
```

Fig 2: Trivial bound of the converted CNN.

The reason for not choosing the 1st method was the inconsistent dimensionality problem at the intersection of a fully-connected layer and a convolutional layer.



```
[75, 6, 16, 120, 84, 10]
(6, 75)
(16, 150)
(120, 9216)
(84, 120)
(10, 84)
```

Fig 3: Improper weight dimensions at the junction of CNN and FC layers.

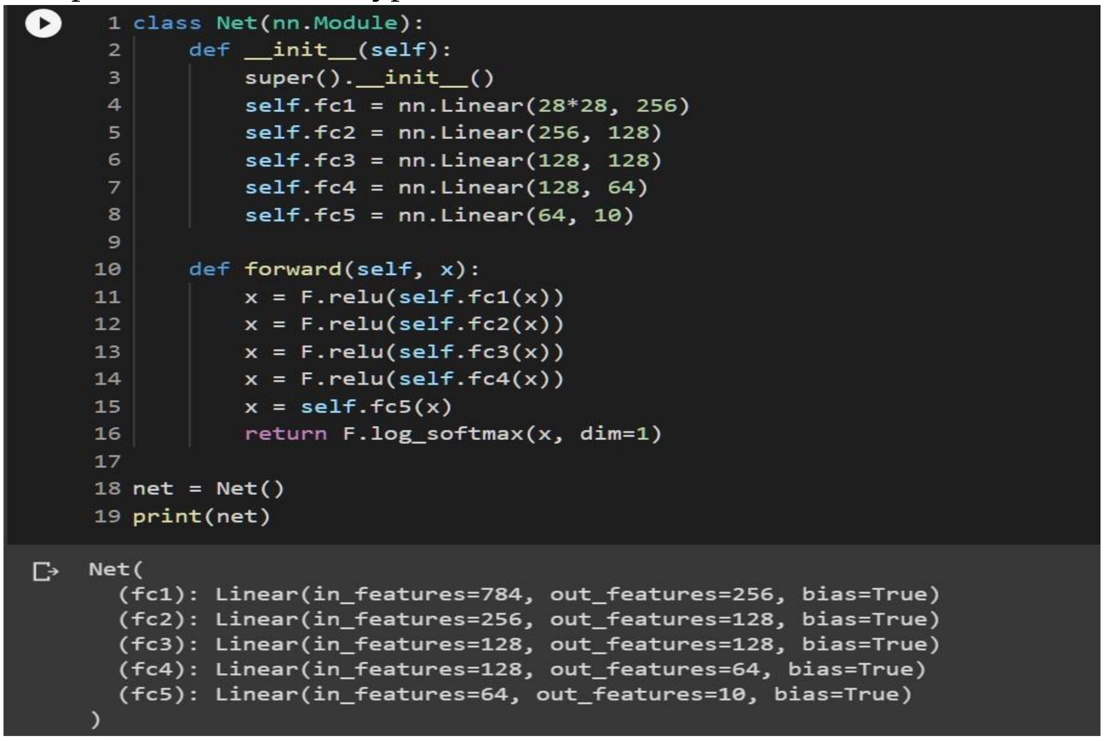
The LipSDP bound could not be measured since it could'nt fit into the RAM. This shows the limits of the algorithm as the tight bound for a small CNN can't be found out within 12 GB of RAM.

Empirical, Tight and Trivial bound analysis

1. *Fully connected Network type: 1.*

- Model Architecture:-

The picture for the first type of network architecture used is shown below.



```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)
        return F.log_softmax(x, dim=1)

net = Net()
print(net)
```

```
Net(
  (fc1): Linear(in_features=784, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=64, bias=True)
  (fc5): Linear(in_features=64, out_features=10, bias=True)
)
```

Fig 4: Fully connected network type 1 architecture.

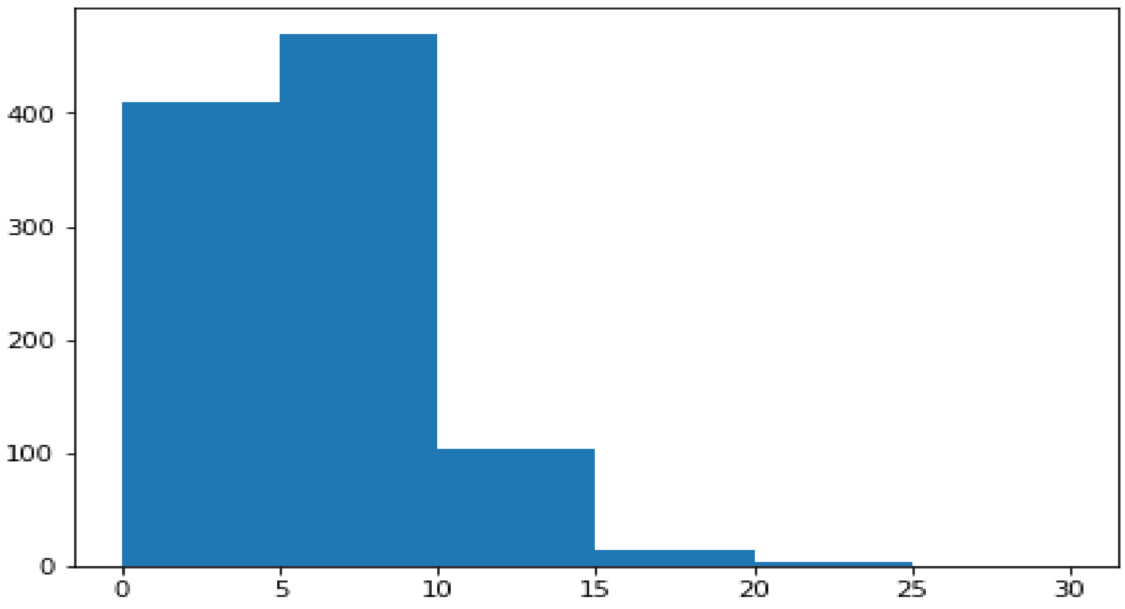
- Testing and histogram creation Information:-

  Test accuracy: 97.3%

  No. of images in test split: 10,000.
  No. of pairs of images taken: 1000
  Maximum value of empirical bound: 24.106
  Value of trivial Lipschitz bound: 9856.117
  Value of LipSDP bound: Could'nt fit into RAM.

  Ratio (Trivial/empirical): 408.86
  Ratio (LipSDP/empirical): NA

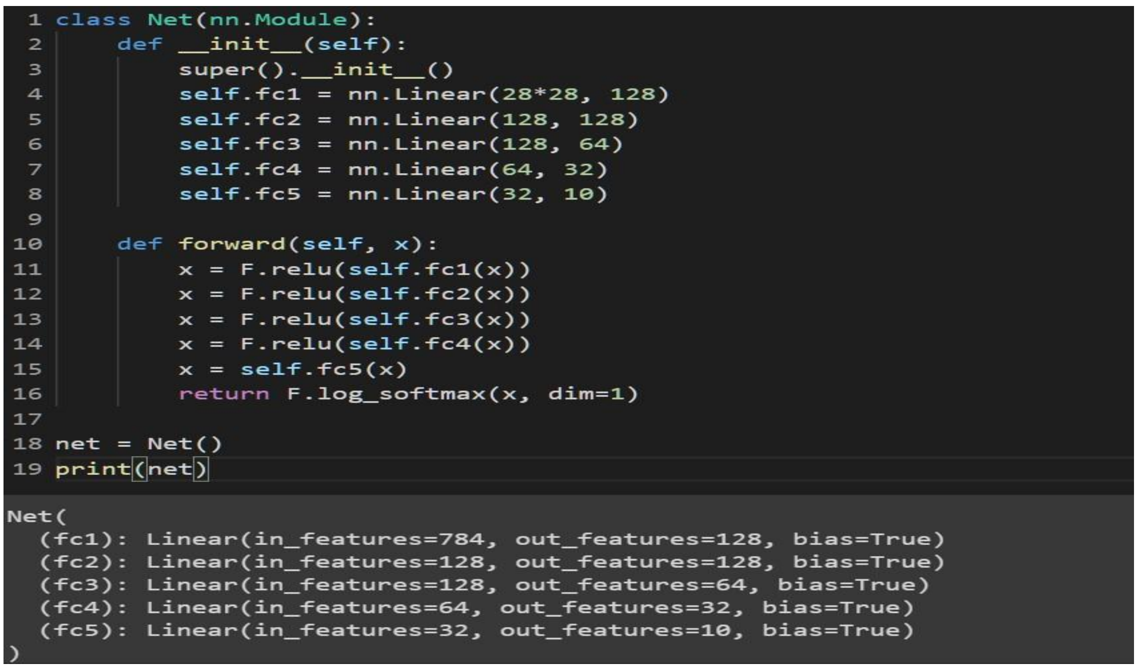
- Histogram:-



Hist 1: The histogram of empirical bounds for FC Network shown in Fig 4. The x-axis corresponds to the values of bounds obtained and the y-axis shows the frequency of the bounds within each bracket.

2. *Fully connected Network type: 2.*

- Model Architecture:-



```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 32)
        self.fc5 = nn.Linear(32, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)
        return F.log_softmax(x, dim=1)

net = Net()
print(net)
```

```
Net(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=64, bias=True)
  (fc4): Linear(in_features=64, out_features=32, bias=True)
  (fc5): Linear(in_features=32, out_features=10, bias=True)
)
```

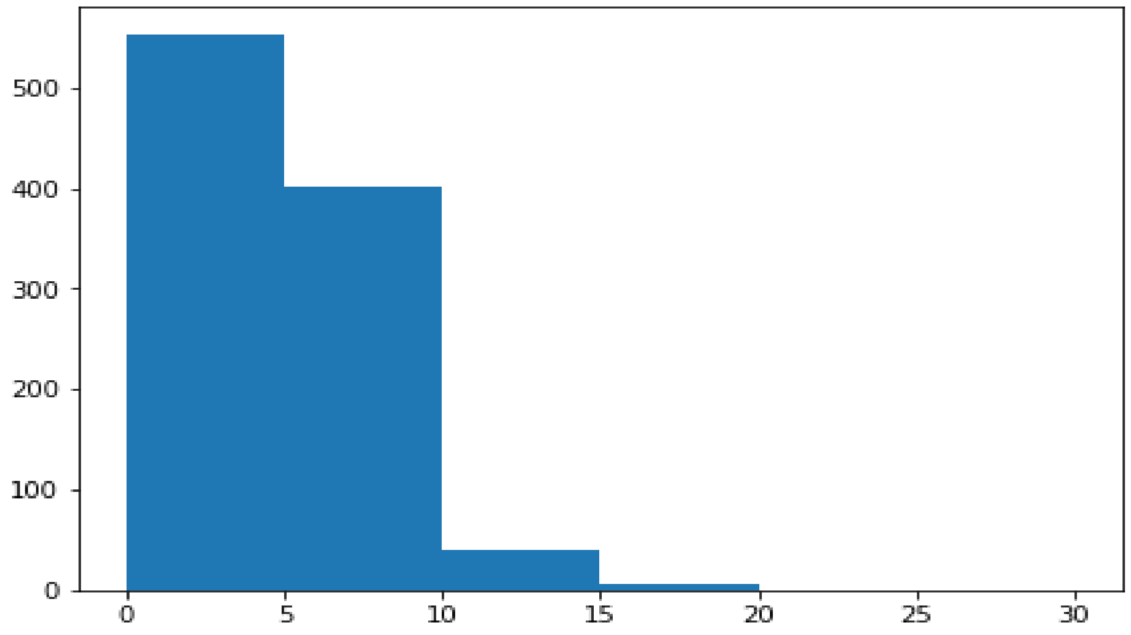
Fig 5: Fully connected network type 2 architecture.

- Testing and histogram creation Information:-

  Test accuracy of network: 96.8%
  No. of pairs of images taken: 1000
  Maximum value of empirical bound: 18.910
  Value of trivial Lipschitz bound: 2041.604
  Value of LipSDP bound: 800.502

  Ratio (Trivial/empirical): 107.96
  Ratio (LipSDP/empirical): 42.33

- Histogram:-



Hist 2: The histogram of empirical bounds for FC Network shown in Fig 5. The x-axis corresponds to the values of bounds obtained and the y-axis shows the frequency of the bounds within each bracket.

ADDITIONAL NOTES:-

1. The MNIST Dataset was used for training and evaluation for both the models.
2. Each image in MNIST has a dimension of 28*28 and there are a total of 60000 train images + 10000 test images in the dataset.
3. The output of the model was taken to be the softmax tensor matrix generated for a pair of images.

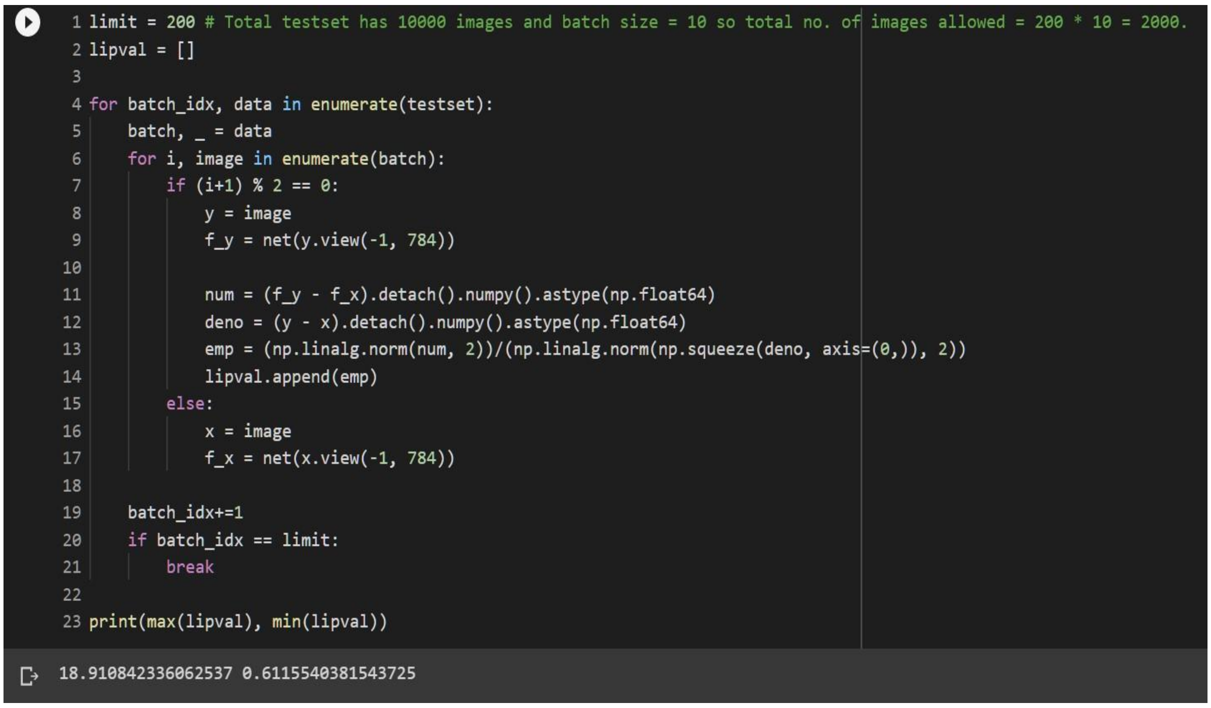
4. The reason for doing the experiment for 2 networks instead of 1 was due to the varying nature of the histograms observed. I felt it was necessary to include these two different types.
5. All the experiments were run on the test batch of images.

Methodology of calculating the empirical bound:-

The formula for calculating the Lipschitz bound for a pair of data input can be expressed mathematically as follows:-

$$|f(x) - f(y)| \leq L|x - y| \qquad \text{for all } x, y \in \mathbb{R}.$$

The manner in which the empirical bound is calculated is illustrated with the code snippet attached below:-



```python
limit = 200 # Total testset has 10000 images and batch size = 10 so total no. of images allowed = 200 * 10 = 2000.
lipval = []

for batch_idx, data in enumerate(testset):
    batch, _ = data
    for i, image in enumerate(batch):
        if (i+1) % 2 == 0:
            y = image
            f_y = net(y.view(-1, 784))

            num = (f_y - f_x).detach().numpy().astype(np.float64)
            deno = (y - x).detach().numpy().astype(np.float64)
            emp = (np.linalg.norm(num, 2))/(np.linalg.norm(np.squeeze(deno, axis=(0,)), 2))
            lipval.append(emp)
        else:
            x = image
            f_x = net(x.view(-1, 784))

    batch_idx+=1
    if batch_idx == limit:
        break

print(max(lipval), min(lipval))
```

```
18.910842336062537 0.6115540381543725
```

Fig 6: Code and sample output for empirical bound.

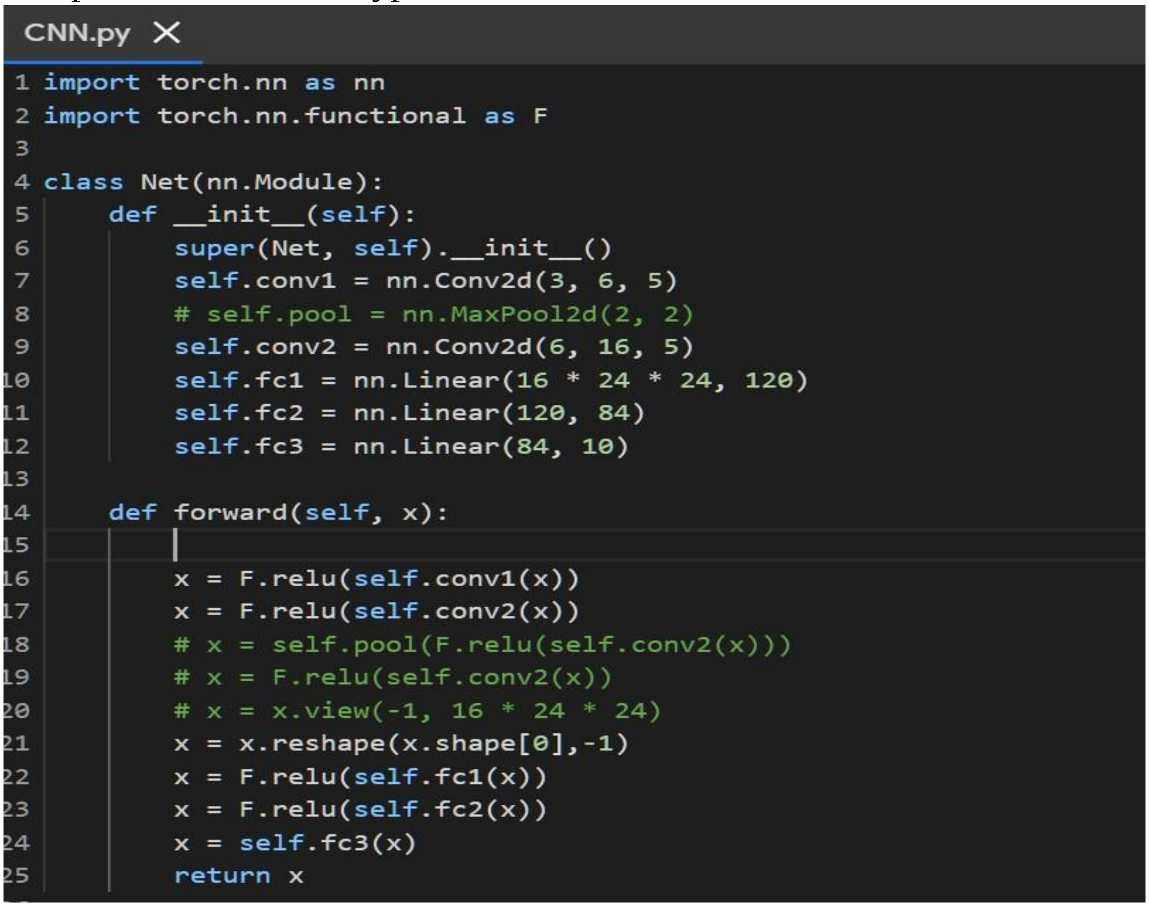
The explanation for the above code is summarized below:-

1. Since the batch size is 10, the no. of batches taken is 200. Hence total no. of pairs = 200*10/2 = 1000.
2. For every consecutive pair of images, the image and their respective network outputs are stored.

3. When i is an even number, or a pair has been stored, the empirical bound is calculated according to the basic Lipschitz formula.
4. All these values are then appended to a list which is later used for creating the histogram.

- ***CNN Model Architecture:-***

The picture for the first type of network architecture used is shown below.



CNN.py

```python
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        # self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 24 * 24, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):

        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        # x = self.pool(F.relu(self.conv2(x)))
        # x = F.relu(self.conv2(x))
        # x = x.view(-1, 16 * 24 * 24)
        x = x.reshape(x.shape[0],-1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Fig 7: CNN Architecture.

- Testing and histogram creation Information:-

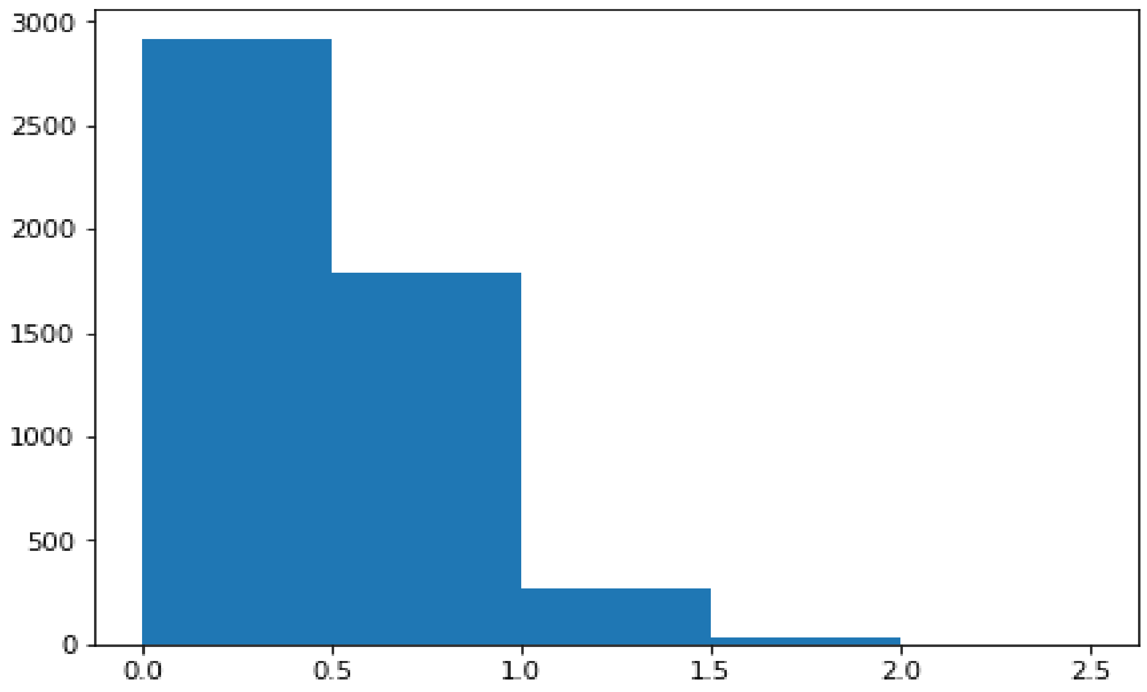
  Test accuracy: 54%

  No. of images in test split: 10,000.
  No. of pairs of images taken: 1000
  Maximum value of empirical bound: 2.25
  Value of trivial Lipschitz bound: 733.248
  Value of LipSDP bound: Could’nt fit into RAM.

Ratio (Trivial/empirical): 325.88
Ratio (LipSDP/empirical): NA

NOTE: There is a significant decrease in the empirical value obtained from CNNs compared to FC networks.

- Histogram:-



Hist 3: The histogram of empirical bounds for CNN shown in Fig 7. The x-axis corresponds to the values of bounds obtained and the y-axis shows the frequency of the bounds within each bracket.

ADDITIONAL NOTES:-

1. The CIFAR-10 Dataset was used for training and evaluation for the CNN model.
2. Each image in CIFAR-10 has a resolution of 32*32 and there are a total of 50000 train images + 10000 test images in the dataset.
3. The output of the model was again taken to be the softmax tensor matrix generated for a pair of images.
4. The test split was used only for the above analysis.

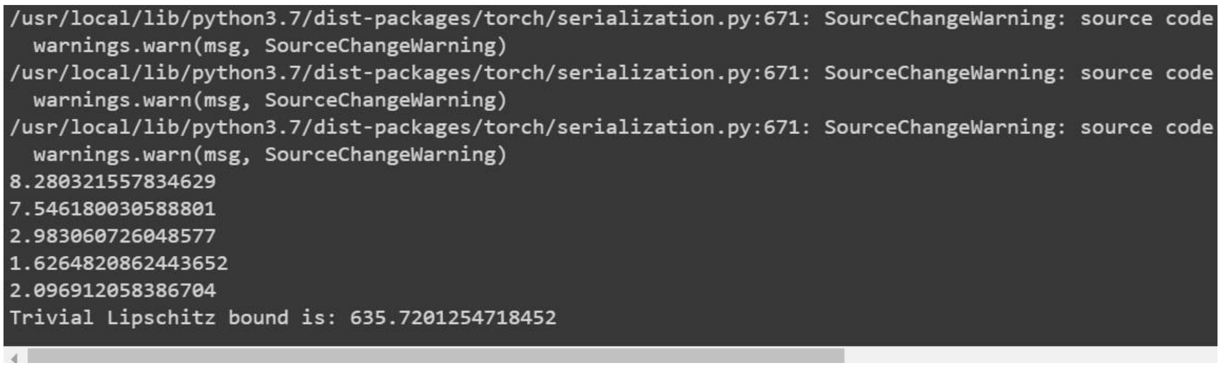
Comparison of Google’s method against LipSDP

Some results from the experiment:-

1. Value of trivial bound using Google’s proposed method: 635.720
2. Value of actual trivial bound: 613.009

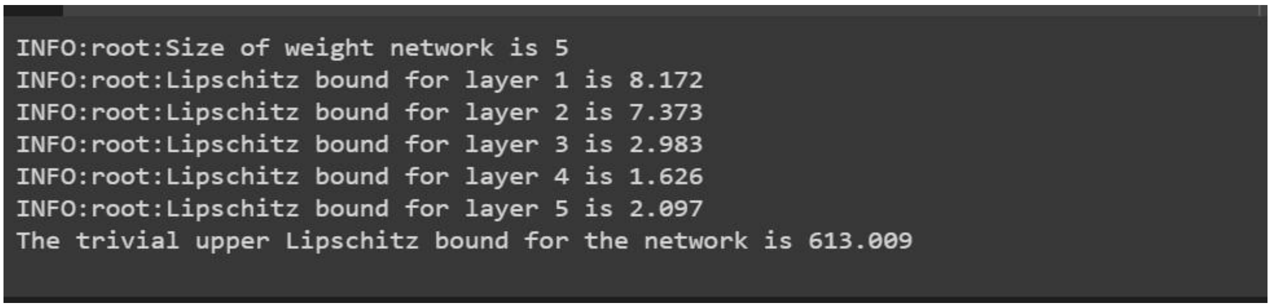
As it can be seen, the value obtained from both of these methods do not *exactly match*.

If we do a layer-by-layer analysis however, then the corresponding values are given below:-



```
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:671: SourceChangeWarning: source code
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:671: SourceChangeWarning: source code
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.7/dist-packages/torch/serialization.py:671: SourceChangeWarning: source code
  warnings.warn(msg, SourceChangeWarning)
8.280321557834629
7.546180030588801
2.983060726048577
1.6264820862443652
2.096912058386704
Trivial Lipschitz bound is: 635.7201254718452
```

Fig 8: Layerwise trivial Lipschitz bounds acc. to Google’s procedure.

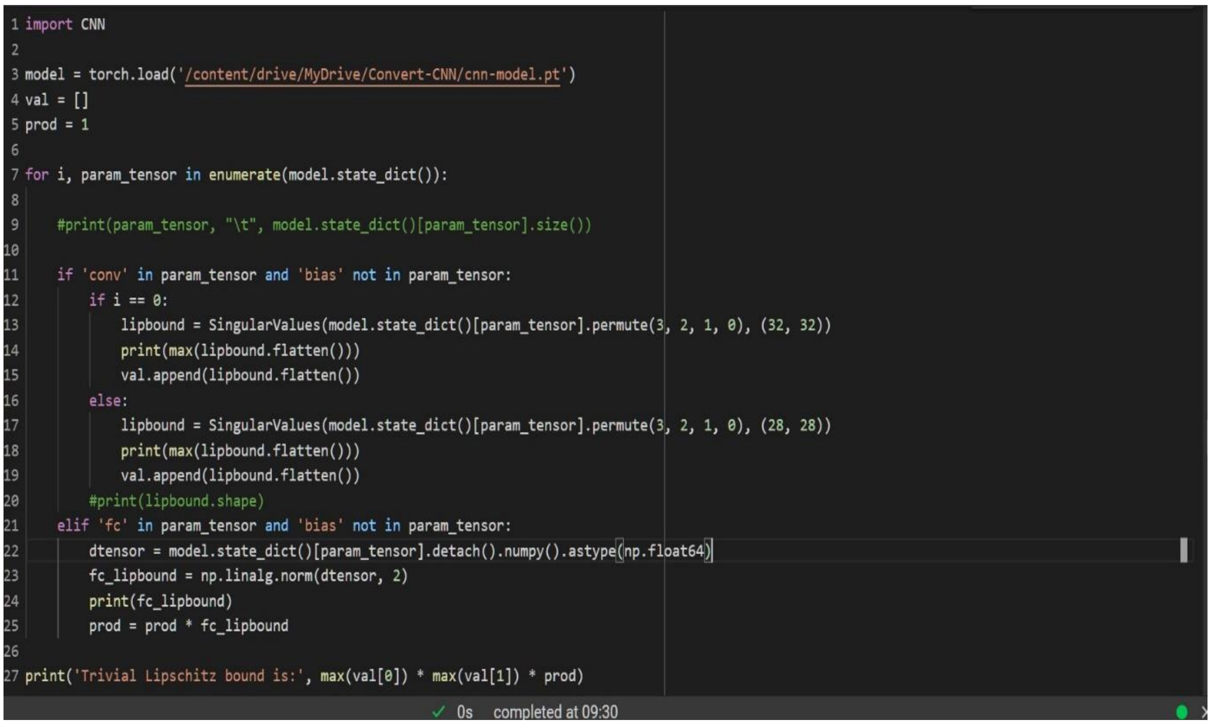


```
INFO:root:Size of weight network is 5
INFO:root:Lipschitz bound for layer 1 is 8.172
INFO:root:Lipschitz bound for layer 2 is 7.373
INFO:root:Lipschitz bound for layer 3 is 2.983
INFO:root:Lipschitz bound for layer 4 is 1.626
INFO:root:Lipschitz bound for layer 5 is 2.097
The trivial upper Lipschitz bound for the network is 613.009
```

Fig 9: Layerwise trivial bounds according to normal procedure.

Now, if we look carefully, the difference can be seen in the *first 2 convolutional layers* (since the network has 2 conv layers + 3 fc layers , see Fig 1). The bound of the first convolutional layer is **8.28 by Google’s method and 8.17 by my method**. Similarly, bound of the second convolutional layer is **7.54 by Google’s method and 7.37 by my method**. These are very small differences but they amplify when multiplied to obtain the trivial bound.

Below attached is a screenshot of how I implemented Google’s method. The entire implementation is done carefully and after going through the discussions on Openreview.net.

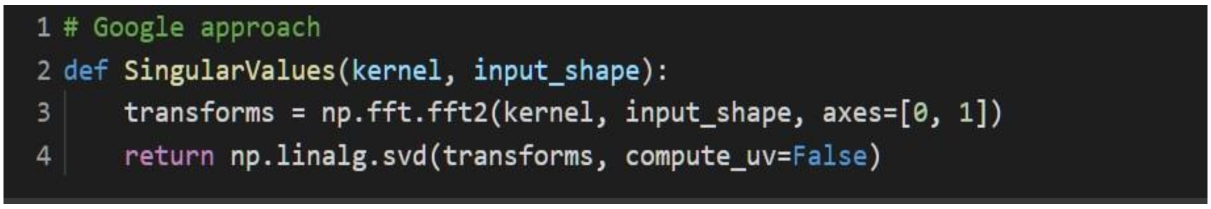


```python
import CNN

model = torch.load('/content/drive/MyDrive/Convert-CNN/cnn-model.pt')
val = []
prod = 1

for i, param_tensor in enumerate(model.state_dict()):

    #print(param_tensor, "\t", model.state_dict()[param_tensor].size())

    if 'conv' in param_tensor and 'bias' not in param_tensor:
        if i == 0:
            lipbound = SingularValues(model.state_dict()[param_tensor].permute(3, 2, 1, 0), (32, 32))
            print(max(lipbound.flatten()))
            val.append(lipbound.flatten())
        else:
            lipbound = SingularValues(model.state_dict()[param_tensor].permute(3, 2, 1, 0), (28, 28))
            print(max(lipbound.flatten()))
            val.append(lipbound.flatten())
        #print(lipbound.shape)
    elif 'fc' in param_tensor and 'bias' not in param_tensor:
        dtensor = model.state_dict()[param_tensor].detach().numpy().astype(np.float64)
        fc_lipbound = np.linalg.norm(dtensor, 2)
        print(fc_lipbound)
        prod = prod * fc_lipbound

print('Trivial Lipschitz bound is:', max(val[0]) * max(val[1]) * prod)
```

```
✓ 0s    completed at 09:30
```

Fig 10: Implementation of Google’s procedure.

The **SingularValues** function is the direct transform method proposed by Google as shown below:-



```python
# Google approach
def SingularValues(kernel, input_shape):
    transforms = np.fft.fft2(kernel, input_shape, axes=[0, 1])
    return np.linalg.svd(transforms, compute_uv=False)
```

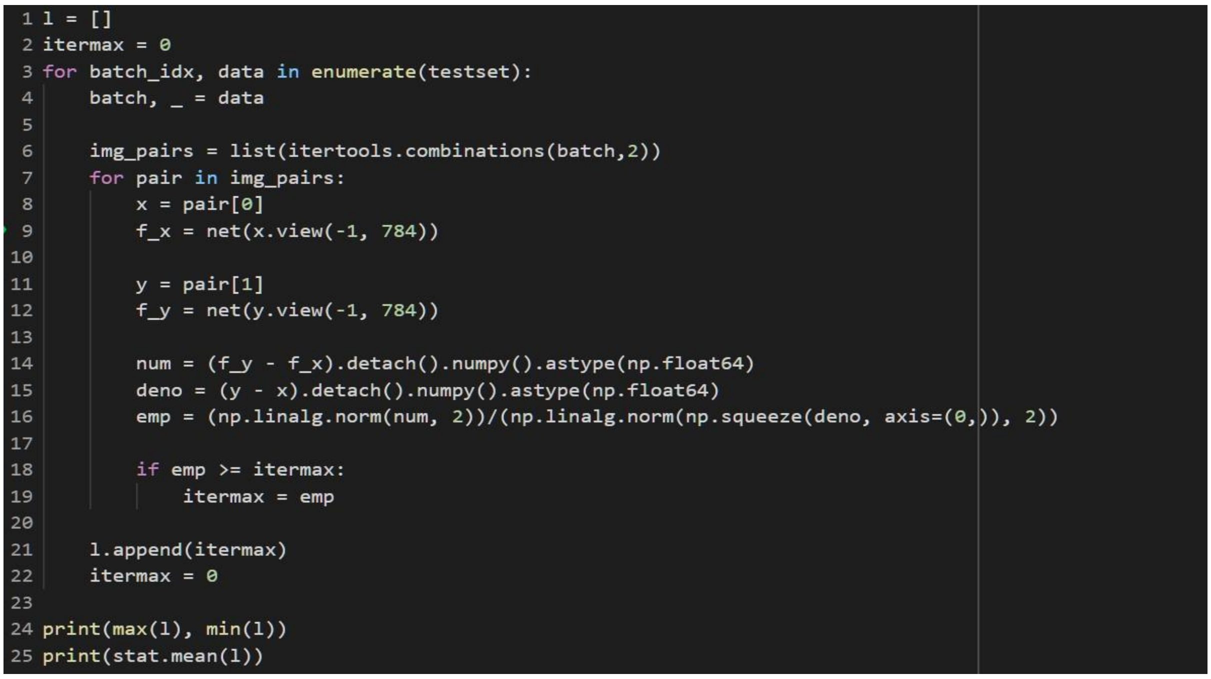
Fig 11: SingularValues function for returning range of singular values for each individual convolutional layer.

Graphical Analysis

Here, the values of the trivial bound, the tight (LipSDP) bound and the average of the maximum empirical Lipschitz values are shown in one place. The way the values are calculated as follows:-

- For each group of images (10, 30, etc), generate all possible **unique** pairs of images. The total no. of such pairs can be obtained by using the combination formula: nCr. For example, 10 images taken at a time, total no. of possible pairs: 10C2 = 45.
- Using the example above, since there are 45 unique image pairs, the total no. of generated empirical Lipschitz bounds will also be 45. Now, the **maximum** of those 45 values are stored.
- Now, iterate over the entire pool of images (or dataset) and for each set of 10 images, generate the **local maximum**. If there are say a total of 10, 000 images then the total no. of such local maximum's will be 10, 000/ 10 = 1000.
- Take the average over the entire set of 1000 maximum's obtained which is then plotted. Keep repeating this procedure for different size of the set of images.

The code snippet for the above procedure is shown below:-



```python
l = []
itermax = 0
for batch_idx, data in enumerate(testset):
    batch, _ = data

    img_pairs = list(itertools.combinations(batch,2))
    for pair in img_pairs:
        x = pair[0]
        f_x = net(x.view(-1, 784))

        y = pair[1]
        f_y = net(y.view(-1, 784))

        num = (f_y - f_x).detach().numpy().astype(np.float64)
        deno = (y - x).detach().numpy().astype(np.float64)
        emp = (np.linalg.norm(num, 2))/(np.linalg.norm(np.squeeze(deno, axis=(0,)), 2))

        if emp >= itermax:
            itermax = emp

    l.append(itermax)
    itermax = 0

print(max(l), min(l))
print(stat.mean(l))
```

Fig 12: Code outlining the above approach.

Below, **Table 4** shows the process illustrated above for different sizes of set of images.

Table 4:-

SL No.	Set size (n)	Avg. emp value (Ln)	Max. of max emp value	Time taken
1.	10	10.279830587380388	21.996063734099092	NS
2.	30	13.707607288527434	21.170323473052314	NS
3.	50	15.246720297071317	20.64324099543047	NS
4.	100	17.03347227003784	24.768061532731902	NS
5.	250	19.07323968615581	24.768061532731902	NS
6.	500	21.005331100841303	23.033706625601297	24 min 32 s
7.	1000	21.927596934984948	24.378693337100803	49 min 11 s
8.	2000	23.889638399197523	27.115284256708122	1 hr 40 min
9.	5000	27.59053004717982	27.930989694770535	4 hr 49 min

*NS = Not significant.

The corresponding plot for the table along with the values of trivial and LipSDP bound is shown below:-

- Trivial bound = 997.543
- LipSDP bound = 406.720

Both of these as well as the above values are obtained from a pre-trained fully connected network. The MNIST dataset was used here.

The **red line denotes the value of the Trivial bound**, the **green line denotes the value of the LipSDP bound**. The **blue line shows the plotted values listed in the table above**. The blue line appears to be squashed due to the distortion induced due to the two bounds. A plot consisting only of the empirical values has also been provided along with the plot consisting all three values below.

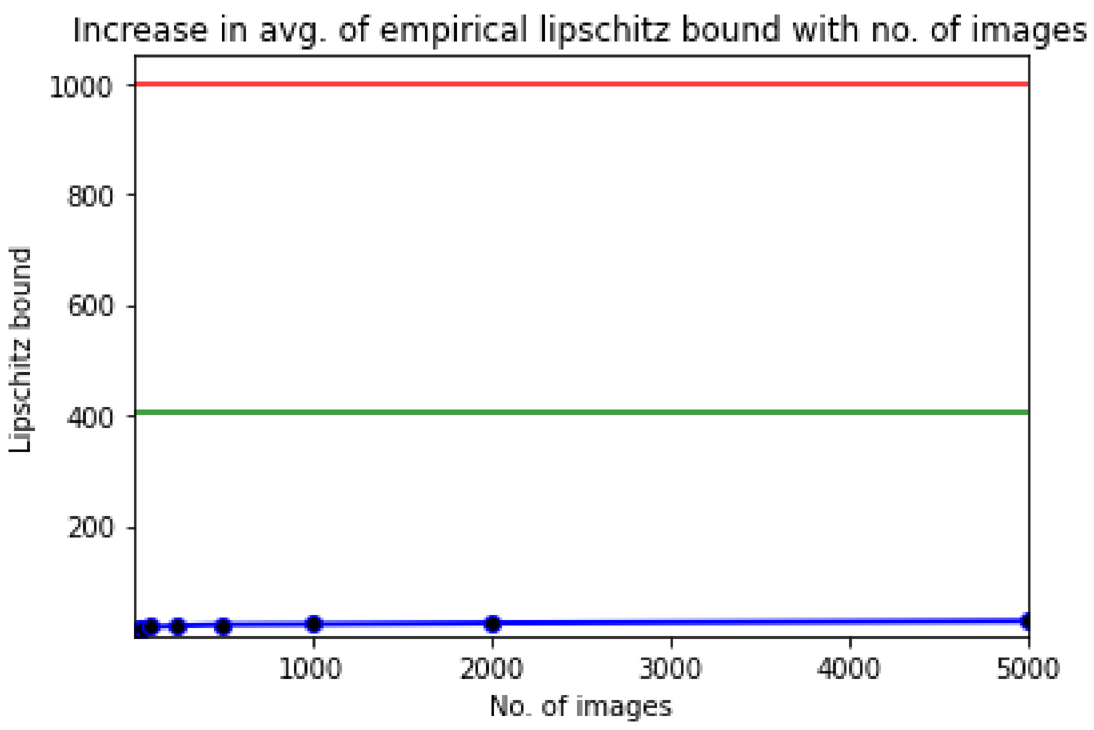


Fig 12: The above plot shows both the bounds as well as the obtained average empirical values in the same plot.

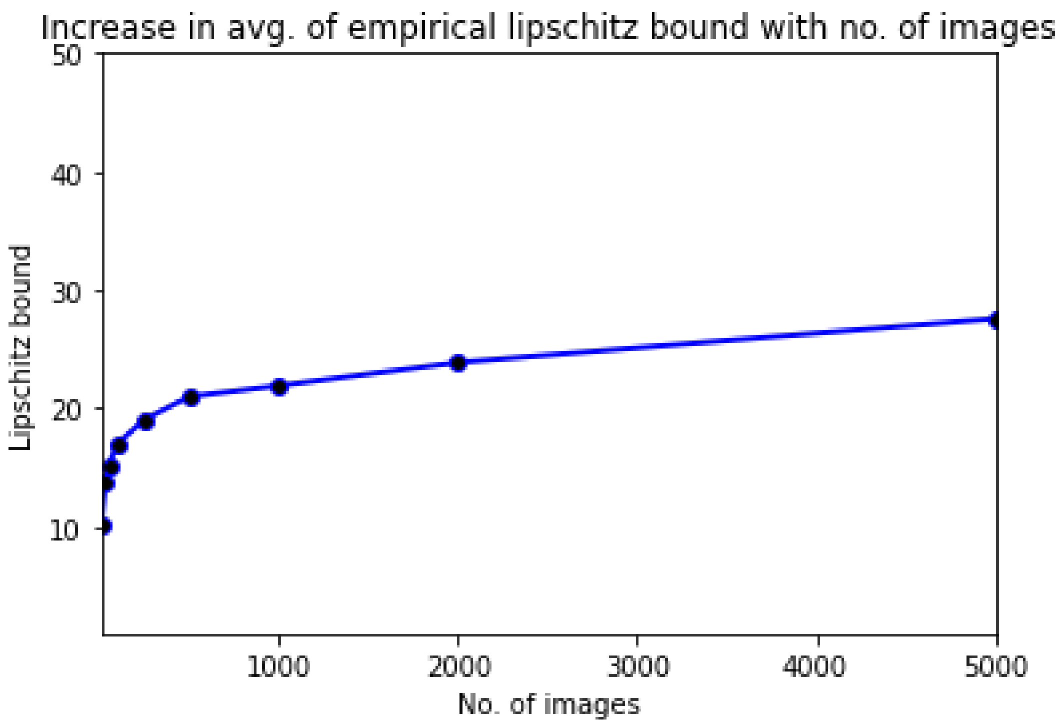


Fig 13: The above plot shows only the values present in Table 4 after changing the perspective.

The same set of the above experiments was carried out on the CIFAR-10 Dataset as described below.

Some notes on the implementation:-

- The same codebase with light tweaks (like shape, channels etc) was used.
- The test-split was used. For more details on CIFAR-10 please see page 18 ADDITIONAL NOTES.

Table 5 lists the values obtained on CIFAR-10 using the same approach outlined above.

Table 5:-

SL No.	Set size (n)	Avg. emp value (Ln)	Max. of max emp value	Time taken
1.	10	4.187693842854425	14.728758172579719	NS
2.	30	6.095645335213012	16.919025989393045	NS
3.	50	7.121934926404491	14.990914609526389	NS
4.	100	8.715434570219498	17.319284407435813	NS
5.	250	11.005277727588096	15.88900128808256	NS
6.	500	12.54003731701458	19.285789917640848	38 min 18 s
7.	1000	14.760203781110514	19.25229735093593	1 hr 14 min
8.	2000	17.909963354143944	19.285789917640848	2 hr 29 min

*NS = Not significant.

The corresponding plot for the table along with the values of trivial and LipSDP bound is shown below:-

- Trivial bound = 727.091
- LipSDP bound = Could'nt be obtained due to RAM constraints.

Both of these as well as the above values are obtained from a pre-trained fully connected network. The **green line shows the trivial bound** and the **blue line represents the plotted points from Table 5**.

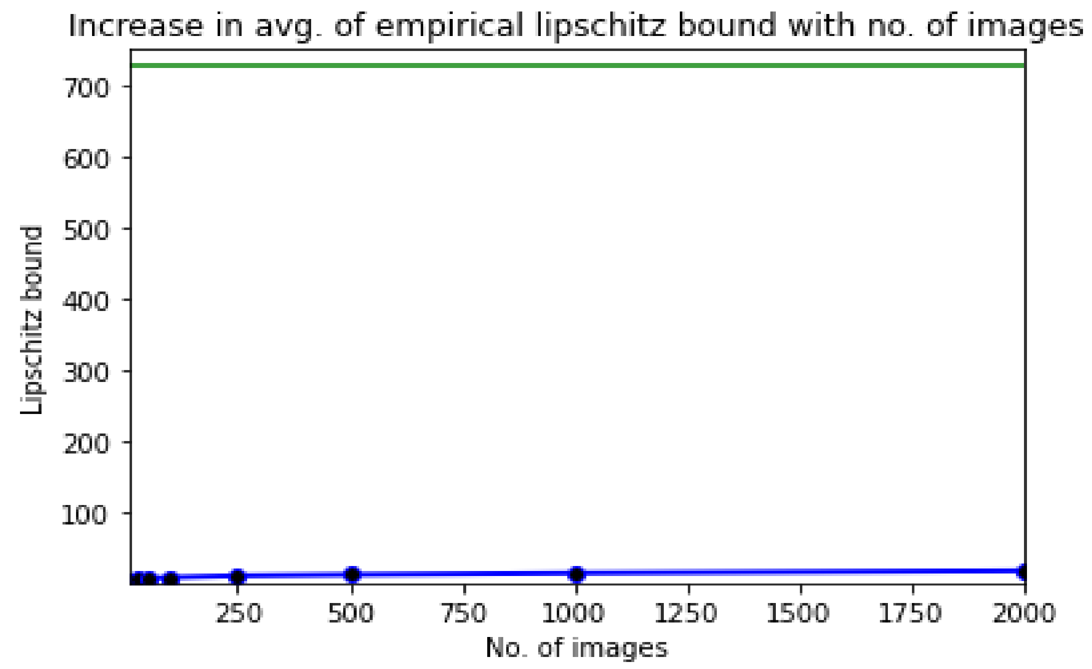


Fig 14: The above plot shows both the bounds as well as the obtained average empirical values in the same plot.

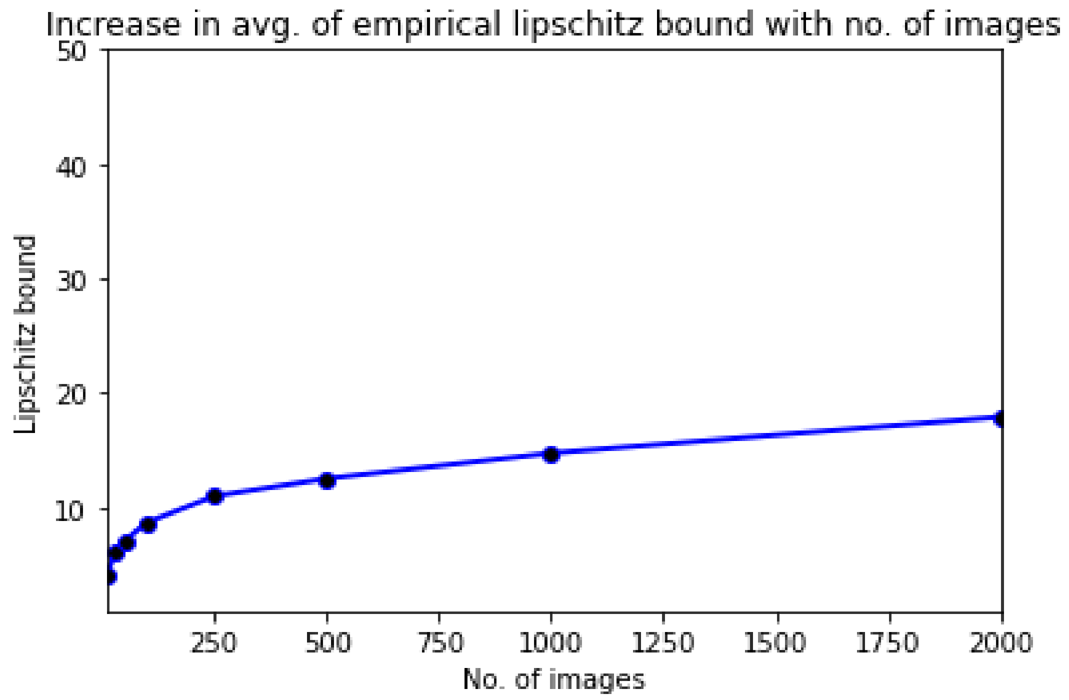


Fig 15: The above plot shows only the values present in Table 5 after changing the perspective.

Other Applications

We also looked at some other applications of Lipschitz regularization to deep models in other fields like:-

- **Feedback control in loops**. Used in robot navigation and in general control theory.
- **Reinforcement Learning** for trajectory control. Useful in cases like Self-driving where it is undesirable if the learning agent picks up unsafe practices, for ex: car crashing against a wall etc.

The primary motivation here is to regulate the **input space** as opposed to the central theme of regulating the **output space of a neural network** explored here in this report. This opens up another vast arena of research especially in classic control theory where this subject has seen significant interest.

Very briefly, we shall touch upon the aspects we looked at while studying literature in control theory dealing with regularization of systems.

There are several approaches commonly used in **Model Predictive Control (MPC)**. MPC is a powerful technique of designing controller systems when compared to more classical approaches like **Linear Control (LC)** and it's varieties like LQ, LQR etc. The difference arises in the fact that in LC, environmental constraints are framed linearly, hence tightness is quite poor and unsuitable, ex: quality control in precision manufacturing etc. On the other hand in MPC, the constraints are quadratic and the bounds can be tight or relaxed depending on the situation. Either way, MPC ensures far better robustness of the model. However, this comes at a cost since Non-linear programming is computationally expensive and cannot be readily deployed in low-resource devices. On the other hand, LC is fast and can be used for real-time systems but one has to compromise on its accuracy leading to potentially unsafe systems.

In [6], Aswani et al., discusses **Learning based Model Predictive Control (LBMPC).** Again, there are two approaches to deal with this situation: either hand-craft the non-linear constraints or use a deep neural network to learn the parameters. The authors went on to prove that the latter approach is **Lipschitz continuous and hence eligible for Lipschitz regularization**. This particular work was published in 2013, before any of the modern regularization approaches had been discovered.

This opens up the possibility of trying to apply LipSDP to the above scenario and design a very-tight, robust and safe controller system.

Discussion

LipSDP results for FC Networks

1. No particular trend was observed wrt. to time with the increase in number of layers or units per layer.
2. This could have been for various reasons one of which could be uneven comparisons: an unsplitted network will take more time than a split network inspite of running on a smaller network.
3. An example in Table 2: For SL No. 1, total time taken for finding out the tight bound was **4775.6 seconds** whereas for SL No.3, total time taken was **2211.7 seconds**.
4. Therefore, running time for the latter network is half that of the former inspite being 3x as large.
5. From the data, ratio for a fixed number of hidden layers tend to remain fixed. (2 to 2.6 for 3 hidden layer networks, 3-4 for 5 hidden layer networks and so on). However, whether this holds for larger networks or not needs to be explored.
6. The Trivial Lipschitz Bound tends to double in magnitude with a **2.5x** increase with number of units per layer. Also, the tight Lipschitz bound tend to increase by **3x** on doubling the no. of units per layer.

Conversion between CNN and FC Networks

1. Two approaches are mainly used however, the first approach results in a dimensionality conflict in the junction between convolutional layers and fully connected layers.
2. The formation and usage of Toeplitz matrices lead to more interesting properties some of which are used in [5] where the properties of Toeplitz matrices are extensively used.
3. The converted CNNs lead to huge filter matrices. These huge matrices consume large memory for computing tight bounds.
4. Point 3 is where Google’s approach comes in handy since they propose a direct method which does’nt require storing those matrices. However, that approach comes with a set of limitations as well.

Histogram analysis

1. Nature of histogram varies with change in the dimensions of Fully connected networks like in the no. of units per layer or the no. of layers present.
2. The value of empirical bound is way less (order of 100-500x) than the software obtained bound as expected. The value of empirical bound is way less (in order of 100-500x) than the software obtained bound as expected.
3. However, no drastic increase in the ratio was observed w.r.t to FC Networks. This maybe is due to the fact that the CNN used here is very small.
4. Compared to our earlier used FC network, **which had 5 layers,** in contrast this network has only **2 convolutional layers**. Inspite of that, the former network gave a (trivial/empirical) ratio of **107** whereas our tiny CNN gave a ratio of **366** which is amazing considering the difference in sizes.

Google, Trivial bound and LipSDP

1. Google's trivial bound does not exactly match to the dot with our earlier trivial bound implementation. The reason perhaps can be attributed to different environements in which the experiments are carried out but which is still something debatable.
2. The ratio between Google's trivial bound and the tight bound comes out to be roughly around **21x**.
3. As pointed out earlier, the main difference between Google's approach and LipSDP is that the former does not account for the **effect the non-linearity has on the model's output**. This results in an inherently inferior approach which will always produce bounds significantly higher than the actual (empirical) bound.
4. Again though, Google's approach is important as it showed a **fast approach** where there was no need of **explicitly storing the kernel matrices** and just the kernel and the input image size per layer sufficed for the task.

Observations from Graphs

1. Our calculated **Average of Maximum Lipschitz Empirical Values** (**ALMEV**) tend to slowly converge towards the maximum of the entire distribution as the number of images are increased.
2. The graphs bring out how much still needs to be done before the bounds obtained from various approaches come close to the actual empirical Lipschitz constant. The ratio still hovers between **20x-500x**.
3. Regarding point 1, the average tends to increase simply because the maximum (of the maximum’s) saturates whereas the minimum (of the maximum’s) slowly climbs up. This is interesting atleast w.r.t to datasets like MNIST where the quality, resolution etc of the images are equal (not the underlying values).

References

[1]https://papers.nips.cc/paper/2019/file/95e1533eb1b20a97777749fb94fdb944-Paper.pdf

[2] https://openreview.net/pdf?id=rJevYoA9Fm

[3] https://hal.inria.fr/inria-00112631/document

[4] https://stackoverflow.com/questions/16798888/2-d-convolution-as-a-matrix-matrix-multiplication

[5] https://ojs.aaai.org/index.php/AAAI/article/view/16824

[6] https://doi.org/10.1016/j.automatica.2013.02.003

----X----