Visualizing Impact of Uncertainty and Adversarial Attack on Deep Classifier Models

CS396: Undergraduate Project - II (UGP - 2)

Student : Ayush Kumar (200246)
Yashwant Mahajan (201156)

Supervisor: Dr. Soumya Dutta

Problem Addressed

In this project we address the problem of understanding the quality, confidence/uncertainty and robustness associated with deep classifier models based on convolutional neural networks used in computer vision and visual computing applications. This project also focuses on studying the susceptibility of these models to different types of adversarial attacks, which is crucial for safety-critical applications. The ultimate goal is to develop a visualization tool that can help users comprehend the impact of uncertainty estimation techniques and adversarial attacks on the performance of deep classifiers.

Datasets Used

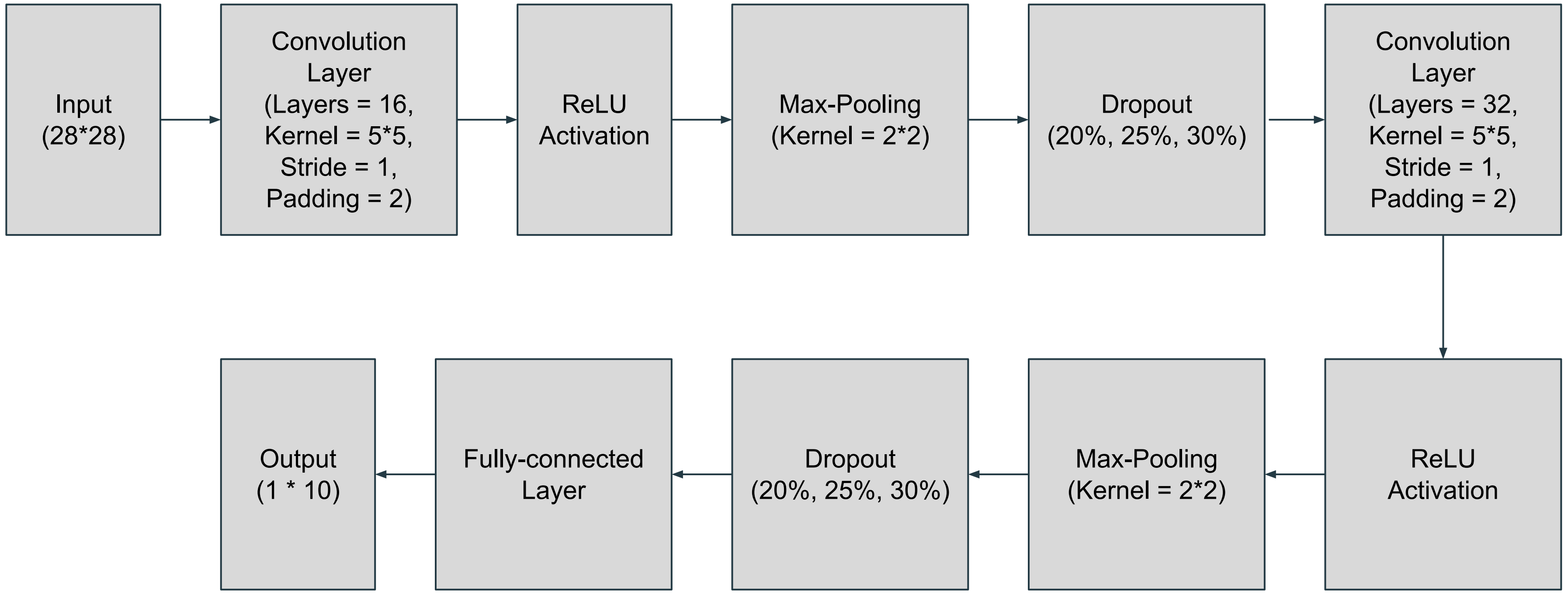
1) MNIST (Modified National Institute of Standards and Technology)
2) STL-10 (Self-Taught Learning 10)

MNIST

What is MNIST and Why did we used MNIST ?

- MNIST stands for "Modified National Institute of Standards and Technology" and refers to a widely used dataset in the field of machine learning and computer vision.
- The MNIST dataset consists of a collection of 70,000 grayscale images of handwritten digits (0-9), with 60,000 images used for training and 10,000 images used for testing.
- Size of each image is 28X28
- The small size and simplicity of the dataset make it convenient for experimentation and quick testing our algorithms for Confidence & Adversarial Attack.

MNIST MODEL ARCHITECTURE


Input (28*28)
Convolution Layer (Layers = 16, Kernel = 5*5, Stride = 1, Padding = 2)
ReLU Activation
Max-Pooling (Kernel = 2*2)
Dropout (20%, 25%, 30%)
Convolution Layer (Layers = 32, Kernel = 5*5, Stride = 1, Padding = 2)
ReLU Activation
Max-Pooling (Kernel = 2*2)
Dropout (20%, 25%, 30%)
Fully-connected Layer
Output (1 * 10)

MNIST MODEL ARCHITECTURE

The architecture of this model is a Convolutional Neural Network (CNN) with two convolutional layers and one fully connected layer.
The first convolutional layer has 16 output channels, uses a kernel size of 5x5, a stride of 1, and a padding of 2. It is followed by a ReLU activation function and a max-pooling layer with a kernel size of 2x2.
The second convolutional layer has 32 output channels, uses a kernel size of 5x5, a stride of 1, and a padding of 2. It is also followed by a ReLU activation function and a max-pooling layer with a kernel size of 2x2.
After the convolutional layers, the output is flattened into a one-dimensional tensor using the view() function. The fully connected layer consists of a linear transformation that takes the flattened output as input and produces 10 output classes, which represents the prediction of the model for each of the 10 possible classes in the dataset. Additionally, a dropout layer with a dropout probability of 0.2/0.25/0.3 is included after each convolutional layer. This helps to induce uncertainty while testing the model on input images.

Why this Architecture?

The architecture of the model is relatively simple and efficient, which makes it a good choice for small image classification tasks such as MNIST dataset. However, for more complex tasks or larger datasets, a more sophisticated model architecture may be necessary to achieve better performance.

Confidence

Confidence in deep learning algorithms refers to the level of certainty or reliability that a model has in its predictions. It is a measure of how confident the model is in its output, indicating the model's belief in the correctness of its predictions for a given input.

A higher confidence score implies that the model is more certain or confident in its prediction, while a lower confidence score indicates lower certainty.

Various methods to estimate Model Confidence

- **Uncertainty Quantification with Dropout (Monte Carlo Dropout)**. It's a technique to estimate the uncertainty of a neural network model by randomly dropping out neurons during inference, which helps in improving the model's confidence scores.
- **Ensemble method** : -
    a. **Different Models** - In this method we build & train multiple models altogether to get different output.
    b. **Different Dataset %age** - In this we train the same model multiple times but every time we train it on different dataset percentage (say 50%) picked randomly out of complete train dataset to get different output.

We have used **Monte Carlo Dropout** Method instead of **Ensemble** Method as it's more computationally efficient while still providing similar benefits in terms of model performance and robustness.

Confidence

Confidence of our CNN model is defined as the variance in predictions made by our model.

Confidence of a single image is calculated by running the image through the model 50 times and computing the variance of probabilities of each class.

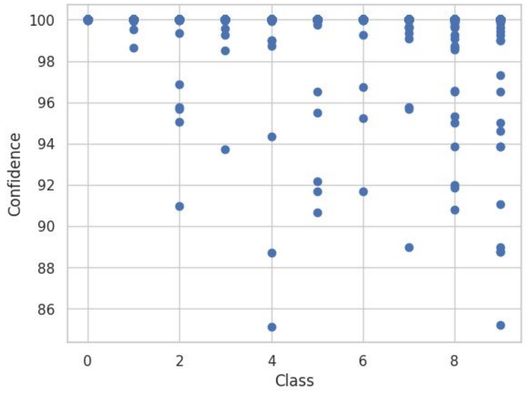
Pseudo Code

```
1. Initialize a tensor of size 50 X 10  'softmax_value'
2. For i in range(50):
    a. Run the image through the model and obtain the output probabilities
    b. Compute the softmax value for the correct class
    c. Append the softmax value to 'softmax_value'
3. Compute the variance  of each class in ‘softmax_value’
4. Subtract the softmax_value from torch.ones(1,10) and you get variance of each class for that image.
```

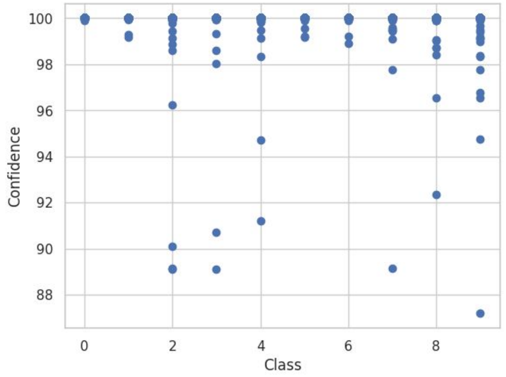
Use-Case

1. **Medical Diagnosis:** A medical diagnosis system can offer confidence scores alongside predictions of various medical conditions, which physicians can use to evaluate the model's reliability and make informed decisions about patient care. For instance, if the model predicts a rare disease with a low confidence score, the physician may confirm the diagnosis before taking any further steps.
2. **Fraud Detection:** In a fraud detection system, a machine learning model can assign confidence scores to its predictions of whether a transaction is fraudulent or not. If the model has a high confidence score for a predicted fraud, it can trigger an automatic alert for further investigation or even block the transaction. On the other hand, if the model has a low confidence score, it may require human intervention for additional verification before taking any action.

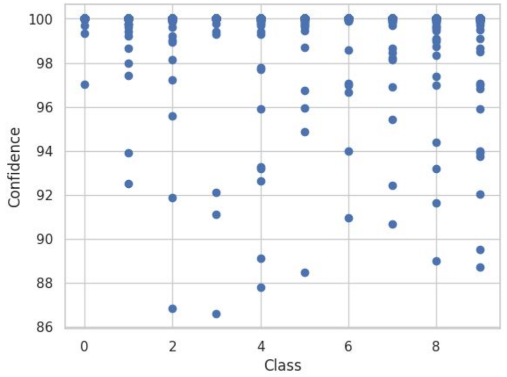
Confidence vs Class



Dropout Probability=0.2



Dropout Probability=0.25



Dropout Probability=0.3

Accuracy

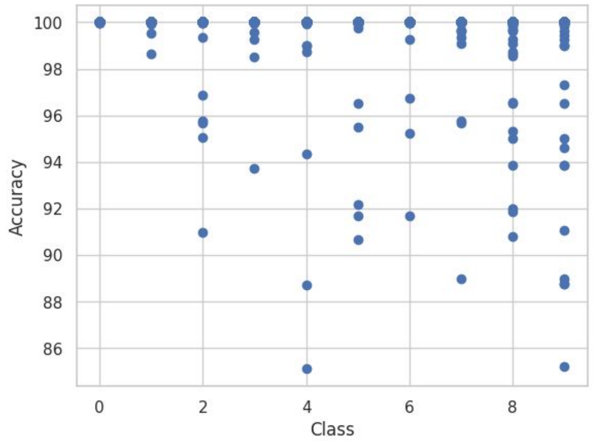
Accuracy of our CNN model is defined as the percentage of correctly classified images in the test dataset.

Accuracy of a single image is calculated by running the image through the model 50 times and computing the average softmax value for the correct class.

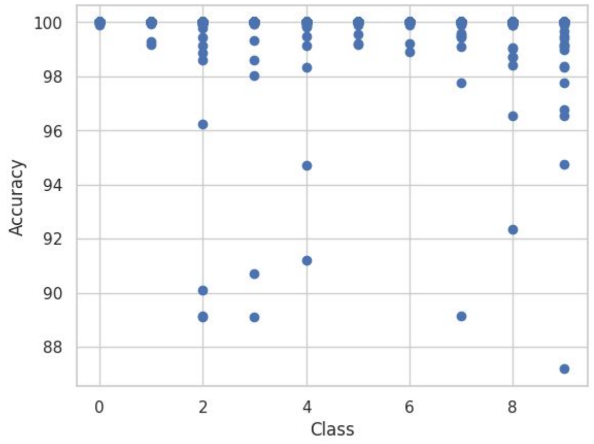
Pseudo Code

```
1. Initialize a variable 'softmax_sum' to 0
2. For i in range(50):
    a. Run the image through the model
    b. Compute the probabilities using softmax value for the correct class
    c. Add the softmax value to 'softmax_sum'
3. Compute the average softmax value by dividing 'softmax_sum' by 50
4. The accuracy of the image is the value computed in step 3
```

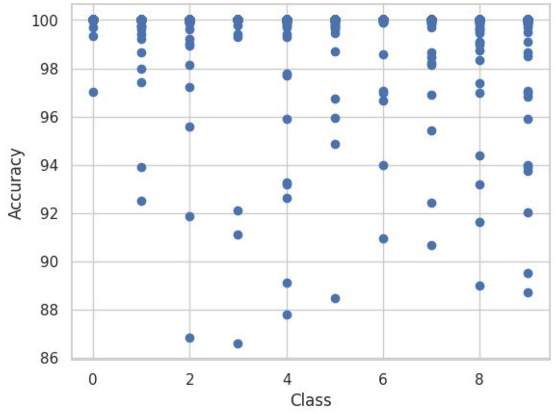
Accuracy vs Class



Dropout Probability=0.2

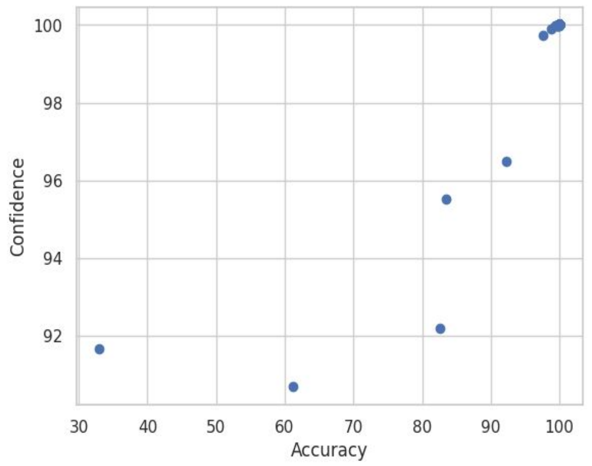


Dropout Probability=0.25

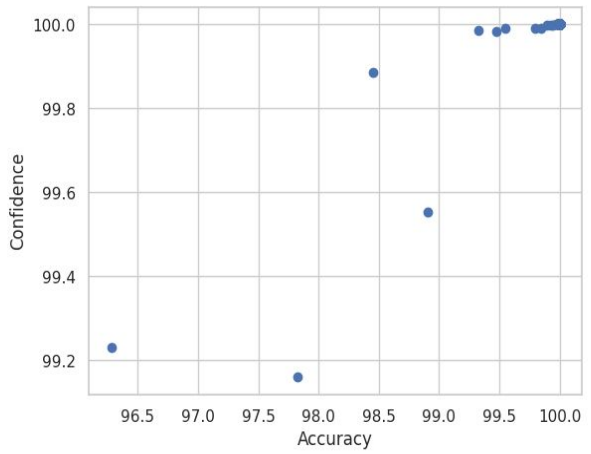


Dropout Probability=0.3

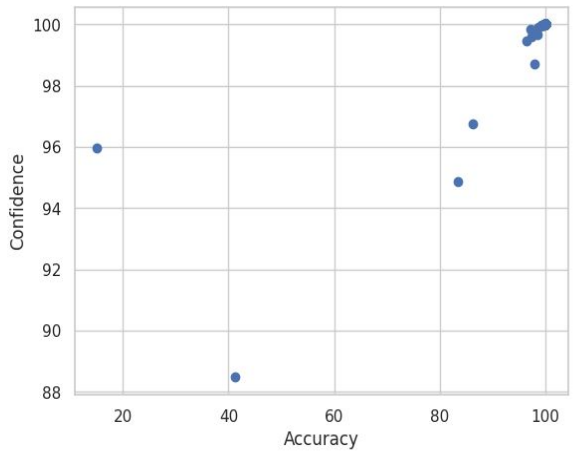
Confidence vs Accuracy



Dropout Probability=0.2



Dropout Probability=0.25



Dropout Probability=0.3

Adversarial Attack

What is Adversarial Attack?

Adversarial attacks are a type of attack that involves crafting inputs to a neural network that are specifically designed to mislead or deceive the model. Adversarial attacks on deep learning models are particularly insidious because they can often be generated using imperceptible perturbations to the input data, and can cause even highly accurate models to produce incorrect or unexpected outputs.

Use-Case:-

1. **Self-driving cars**: An attacker could use adversarial attacks to create misleading traffic signs or road markings that cause the self-driving car to behave unexpectedly.
2. **Facial recognition**: An attacker could use adversarial attacks to create modified images that are misclassified by the facial recognition system, allowing them to bypass security measures.
3. **Medical diagnosis**: An attacker could use adversarial attacks to generate images that deceive a medical diagnosis system, leading to incorrect diagnoses and potentially harmful treatments.

Adversarial Attack Types

1. **FGSM (Fast Gradient Sign Method)**
2. **L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno)**
3. **IAA (Imperceptible Adversarial Attack)**

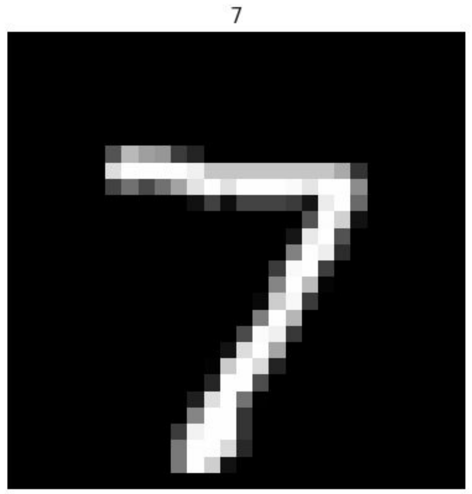
Good & Bad Images

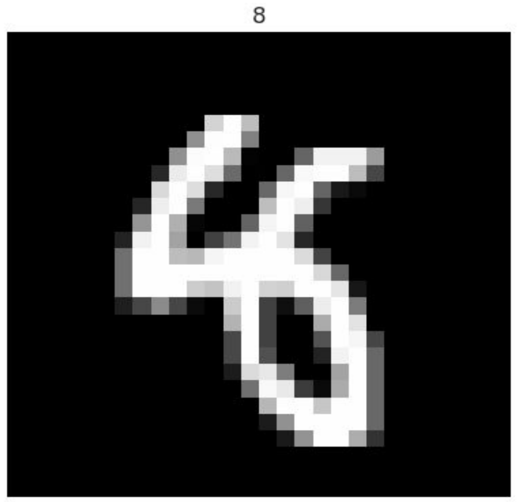
Good Images

Good Images are images that are robust to adversarial attacks and are difficult to fool with small perturbations to their pixel values.

Bad Images

Bad Images are images that are vulnerable to adversarial attacks and can be easily fooled with small perturbations to their pixel values.





Fast Gradient Sign Method

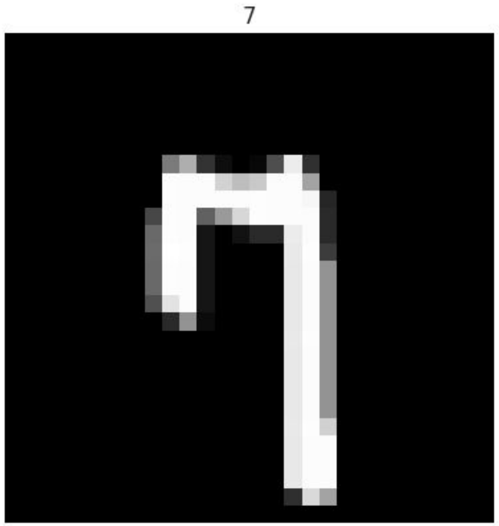
FGSM (Fast Gradient Sign Method) is a simple and efficient technique for generating adversarial examples in machine learning. It involves adding a small perturbation to the input data in the direction of the gradient of the model's loss function, scaled by a small value epsilon. This perturbation can cause the model to misclassify the input, highlighting vulnerabilities in the model's robustness.

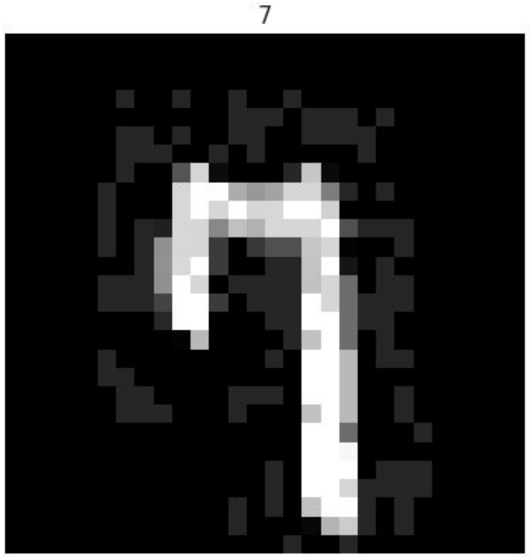
FGSM

Pseudo Code

```
1. Sets the model to evaluation mode.
2. Enables gradients for the input image.
3. Performs a forward pass to obtain the model's output and intermediate variable x.
4. Calculates the loss
5. Performs a backward pass to compute gradients.
6. Zeros out the gradients of the model.
7. Generates the adversarial example by adding the scaled sign of the data gradients to
   the original image.
8. Clips the perturbed image to be within the valid image pixel range (0 to 1).
9. Returns the perturbed image as the output.
```

Original vs FGSM





Limited-memory Broyden-Fletcher-Goldfarb-Shanno

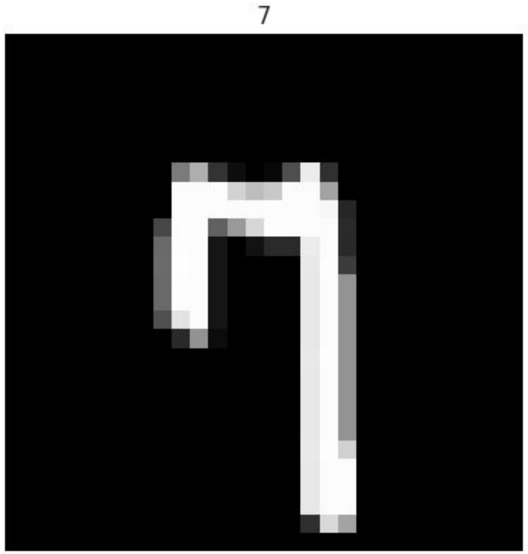
L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) is an efficient optimization algorithm used for crafting adversarial examples in machine learning. It iteratively finds optimal perturbations to maximize misclassification while minimizing perceptibility. LBFGS is widely used for evaluating model robustness and vulnerability to adversarial attacks.

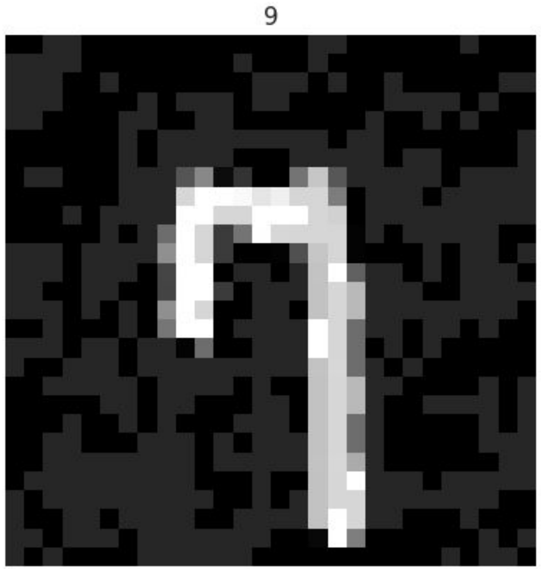
L-BFGS

Pseudo Code

1. Sets the model to evaluation mode.
2. Enables gradients for the input image.
3. Initialize the attack with a copy of the input image.
4. Define the L-BFGS optimizer
5. Use a closure function to compute the loss and gradient
6. Iteratively optimizes the input image to generate an adversarial example
7. Clips the perturbed image to be within the valid image pixel range (0 to 1).
8. Returns the perturbed image as the output.

Original vs L-BFGS





Imperceptible Adversarial Attack (IAA)

This is an iterative method used for crafting adversarial examples in machine learning. Here we find the priority of pixels on the basis of their imperceptibility to human eyes. We then perturb a batch of pixels by a small value and repeat this process until we reach a maximum total perturbation.

IAA Implementation

$$Gap(X^*) \approx P_t - \log(\textstyle\sum e^{kP_i})/k$$

$$SD(x_i) = \sqrt{\frac{\sum_{x_k \in S_i} (x_k - \mu)^2}{n^2}}.$$

$$Sen(x_i) = 1/SD(x_i).$$

$$PerturbPriority(x_i) = \frac{\nabla_{x_i} Gap(X^*)}{Sen(x_i)},$$

. Overall, we formulate the problem as follows:

$$\underset{X^*}{\text{argmax}} \;\; Gap(X^*)$$

(

$$s.t. \;\; D(X^*, X) \leq D_{max}$$

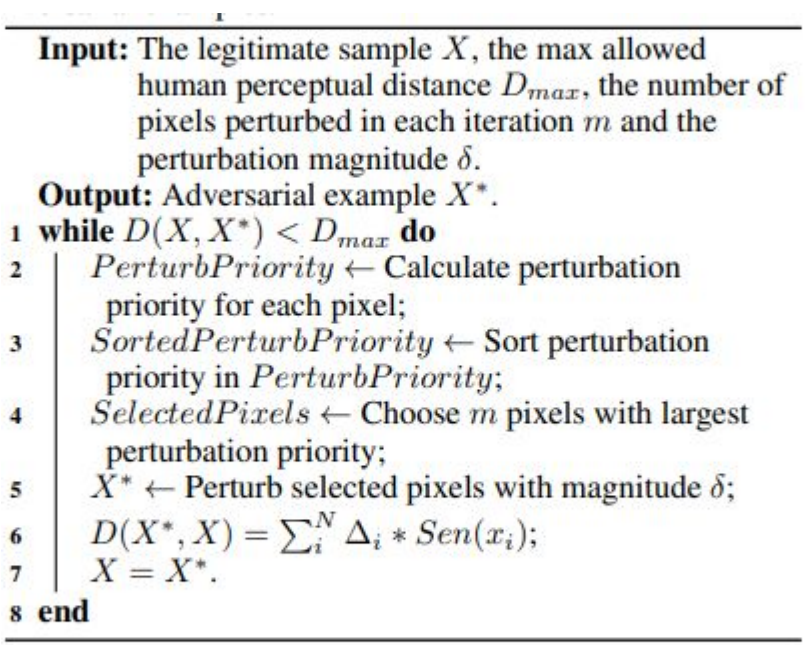
We should perturb pixels at high variance zones rather than low variance ones. We compute the variance of a pixel based on the standard deviation SD(xi) among a 3X3 region.

Accordingly, we introduce perturbation sensitivity to measure how much “attention” will be drawn by adding per “unit” perturbation on a pixel – Sen(xi)

For robust attack, it is necessary to reduce the max probability among other classes as well. Gap(X*)

Considering that we have to choose pixels with less perturbation sensitivity to human eyes and at the same time increase the objective function we define a new quantity called perturbation priority to estimate the effect of perturbing a pixel.

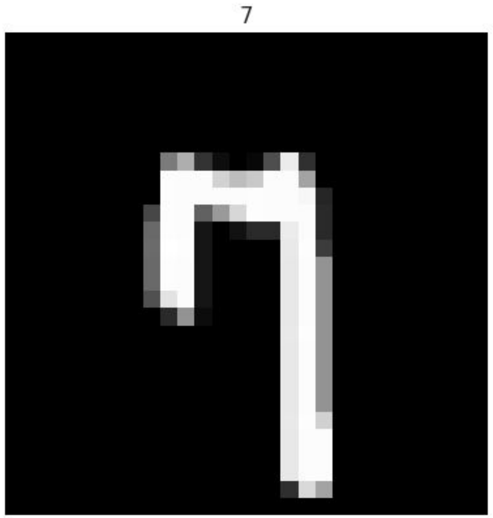
IAA Implementation

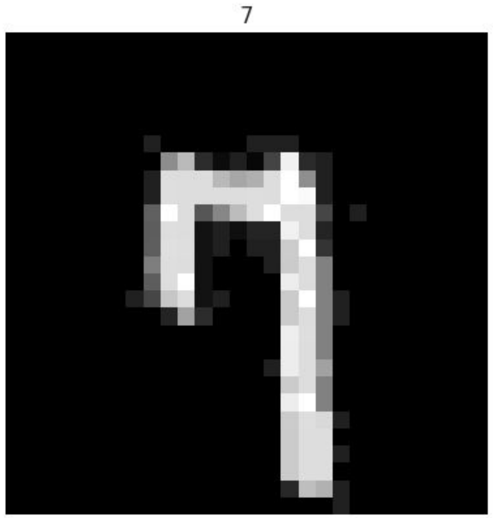


```
Input: The legitimate sample $X$, the max allowed human perceptual distance $D_{max}$, the number of pixels perturbed in each iteration $m$ and the perturbation magnitude $\delta$.
Output: Adversarial example $X^*$.
1 while $D(X, X^*) < D_{max}$ do
2     $PerturbPriority \leftarrow$ Calculate perturbation priority for each pixel;
3     $SortedPerturbPriority \leftarrow$ Sort perturbation priority in $PerturbPriority$;
4     $SelectedPixels \leftarrow$ Choose $m$ pixels with largest perturbation priority;
5     $X^* \leftarrow$ Perturb selected pixels with magnitude $\delta$;
6     $D(X^*, X) = \sum_i^N \Delta_i * Sen(x_i)$;
7     $X = X^*$.
8 end
```

To evaluate the human perceptual effect of a perturbation added to a pixel, we can multiply the magnitude of the perturbation by its sensitivity

Original vs IAA





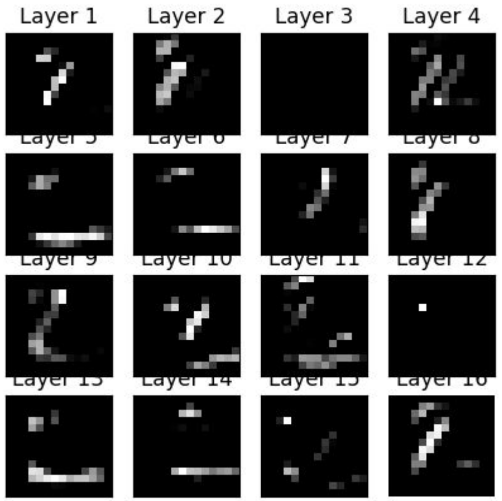
Activation Layers

In our CNN model activation layers refer to the intermediate layers that capture and highlight specific features or patterns in input images or feature maps. These activation layers play a crucial role as they help in learning and representing meaningful features for the MNIST dataset.

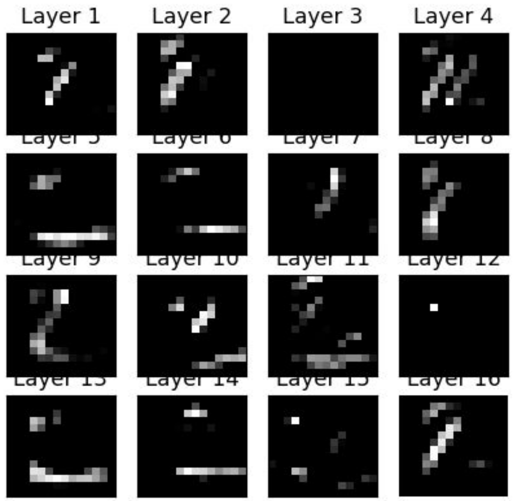
In our MNIST CNN Model we have : -

1. 16 layers in Convolution Layer 1.
2. 32 Layers in Convolution Layer-2.

Original

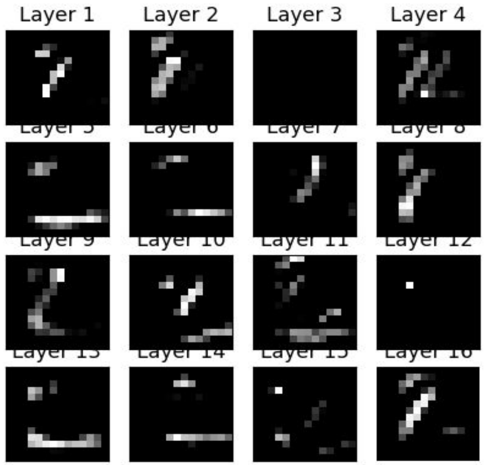


IIA

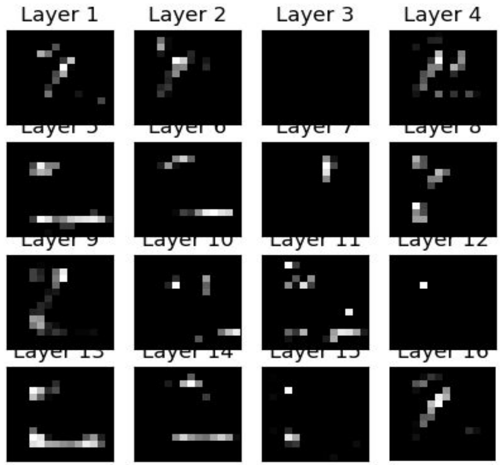


Convolutional Layer 1

L-BFGS



FGSM



STL-10

What is STL-10 and Why did we used STL-10 ?

- STL-10 (Self-taught Learning 10) is an image recognition dataset consisting of 10 classes, each containing 5000 labeled training images and 800 test images.
- STL-10 dataset consist of color images of 10 classes (airplanes, birds, cars, deer, dogs, cats, horses, monkeys, ships, and trucks).
- The images in this dataset are of size 96x96 and are taken from a larger dataset of unlabeled images known as the "unlabeled dataset".

- STL-10 is a good choice because it is larger than MNIST and CIFAR-10, but smaller than the massive ImageNet dataset. This means that it is more challenging than smaller datasets, but not so large that it requires massive computational resources to work with.
- Hence more sophisticated models such as AlexNet can now be used for estimating confidence & robustness.

STL-10 MODEL ARCHITECTURE

AlexNet is a deep convolutional neural network architecture that was designed for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It consists of 8 layers, including 5 convolutional layers and 3 fully connected layers. The architecture can be summarized as follows:

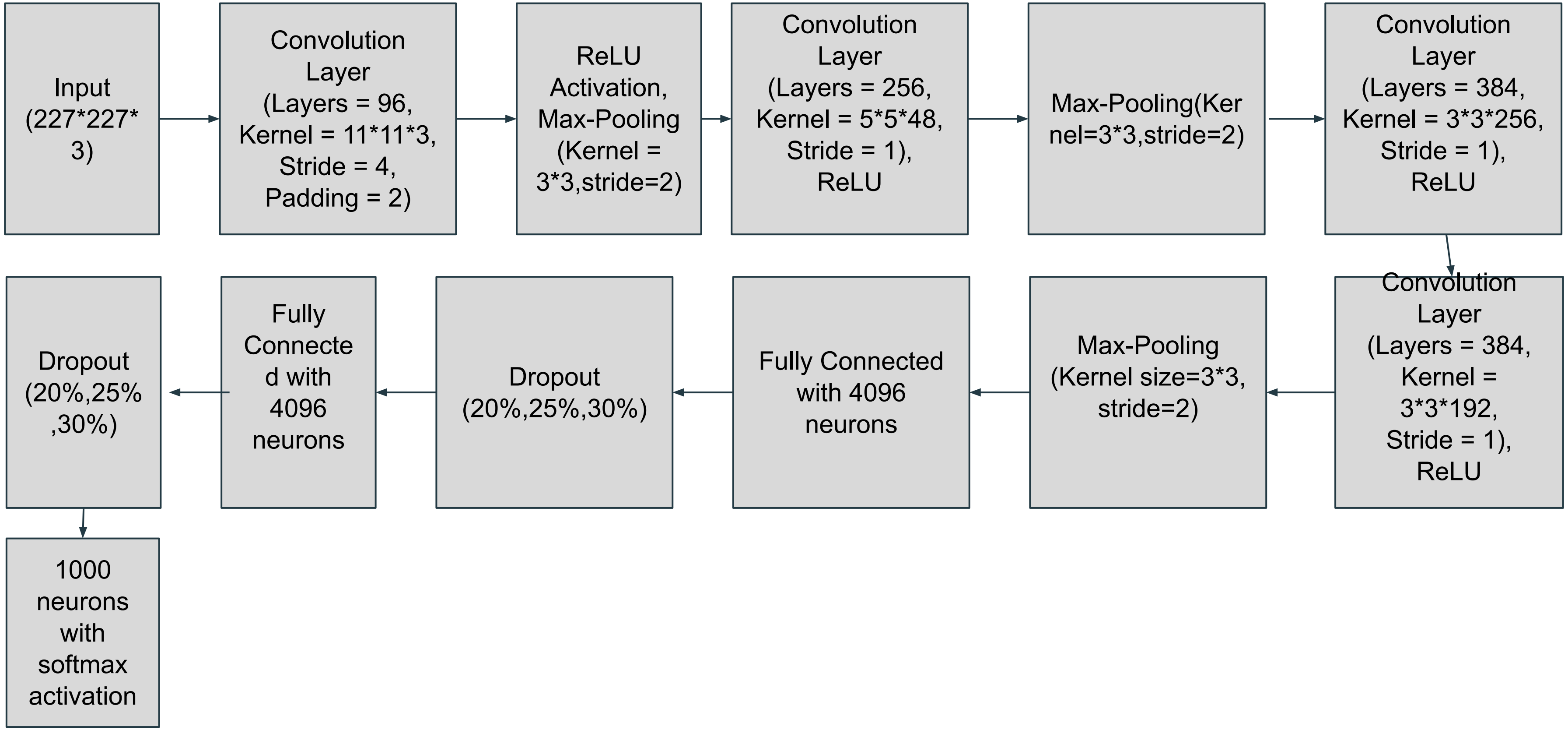
- Input layer: 227 x 227 x 3 image
- Convolutional layer 1: 96 filters of size 11 x 11 x 3, with a stride of 4 and ReLU activation function
- Max pooling layer 1: 3 x 3 max pooling with a stride of 2
- Convolutional layer 2: 256 filters of size 5 x 5 x 48, with a stride of 1 and ReLU activation function
- Max pooling layer 2: 3 x 3 max pooling with a stride of 2
- Convolutional layer 3: 384 filters of size 3 x 3 x 256, with a stride of 1 and ReLU activation function
- Convolutional layer 4: 384 filters of size 3 x 3 x 192, with a stride of 1 and ReLU activation function
- Convolutional layer 5: 256 filters of size 3 x 3 x 192, with a stride of 1 and ReLU activation function
- Max pooling layer 3: 3 x 3 max pooling with a stride of 2
- Fully connected layer 1: 4096 neurons with ReLU activation function
- Dropout layer 1: with a rate of 0.5

- Fully connected layer 1: 4096 neurons with ReLU activation function
- Dropout layer 1: with a rate of 0.5

-- Fully connected layer 3: 1000 neurons with softmax activation function (output layer for classification)

Note that the original AlexNet model was trained on the ImageNet dataset, which has 1000 classes, and hence the output layer

STL-10 MODEL ARCHITECTURE


Input (227*227*3)
Convolution Layer (Layers = 96, Kernel = 11*11*3, Stride = 4, Padding = 2)
ReLU Activation, Max-Pooling (Kernel = 3*3,stride=2)
Convolution Layer (Layers = 256, Kernel = 5*5*48, Stride = 1), ReLU
Max-Pooling(Kernel=3*3,stride=2)
Convolution Layer (Layers = 384, Kernel = 3*3*256, Stride = 1), ReLU
Convolution Layer (Layers = 384, Kernel = 3*3*192, Stride = 1), ReLU
Max-Pooling (Kernel size=3*3, stride=2)
Fully Connected with 4096 neurons
Dropout (20%,25%,30%)
Fully Connected with 4096 neurons
Dropout (20%,25%,30%)
1000 neurons with softmax activation

Why this Architecture?

Alexnet is a popular CNN architecture known for its superior performance on image classification tasks. It is particularly suited for large-scale datasets with complex visual features, such as STL-10, which has a large number of high-resolution images with diverse content.

Alexnet architecture, with its multiple convolutional and max-pooling layers, has the capability to learn complex features and patterns from such datasets, making it an ideal choice for image classification tasks on large and complex datasets like STL-10.

Also, since Alexnet is quite popular & widely used, it’s pre-trained model is easily available saving us a lot of computational time.

Why this Architecture?

The architecture of the model is relatively simple and efficient, which makes it a good choice for small image classification tasks such as MNIST dataset.

However, for more complex tasks or larger datasets, a more sophisticated model architecture may be necessary to achieve better performance.

Confidence & Accuracy

Same as that defined for MNIST dataset.

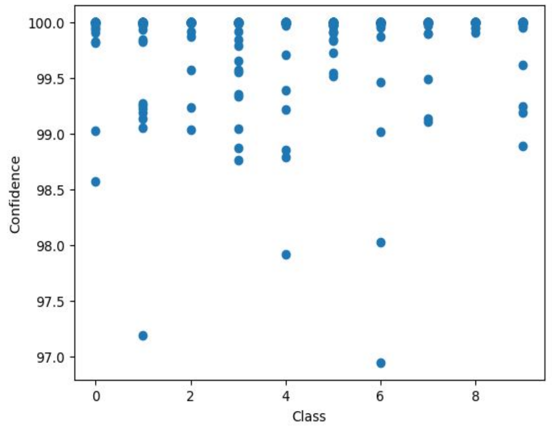
<u>Confidence</u>

Confidence of our CNN model is defined as the variance in predictions made by our model.
Confidence of a single image is calculated by running the image through the model 50 times and computing the variance of probabilities of each class.

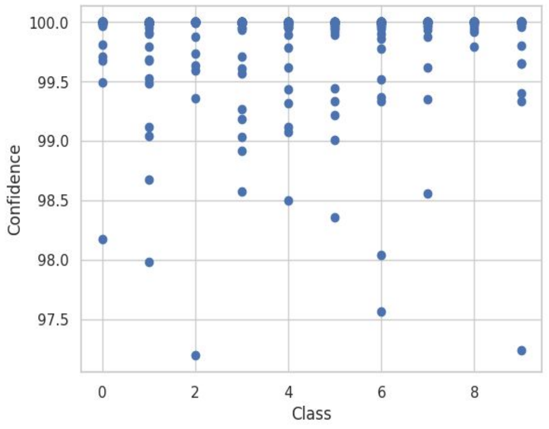
<u>Accuracy</u>

Accuracy of our CNN model is defined as the percentage of correctly classified images in the test dataset.
Accuracy of a single image is calculated by running the image through the model 50 times and computing the average softmax value for the correct class.

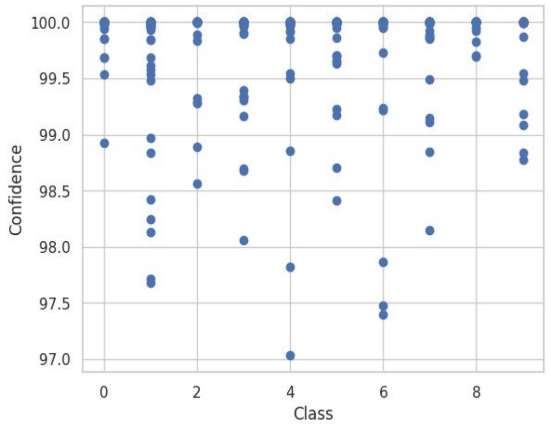
Confidence vs Class



Dropout Probability=0.2



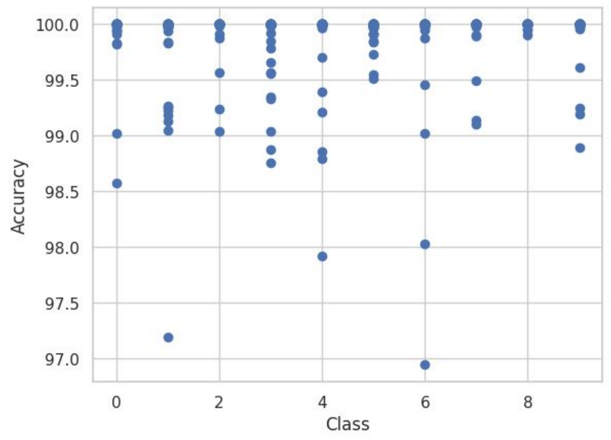
Dropout Probability=0.25



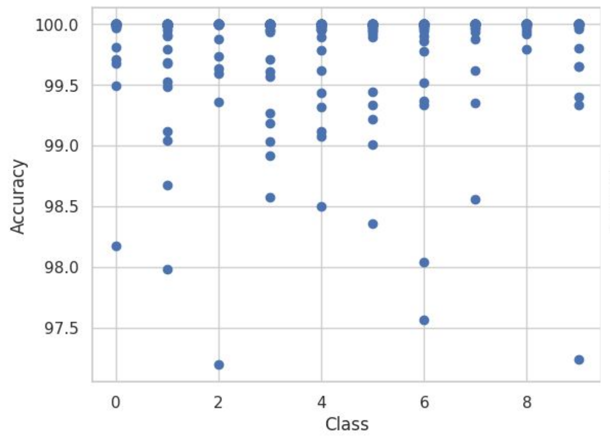
Dropout Probability=0.3

MC Dropout method is used for estimating Model Confidence

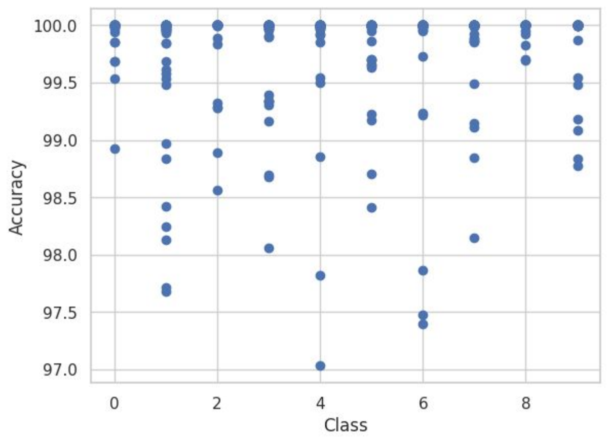
Accuracy vs Class



Dropout Probability=0.2

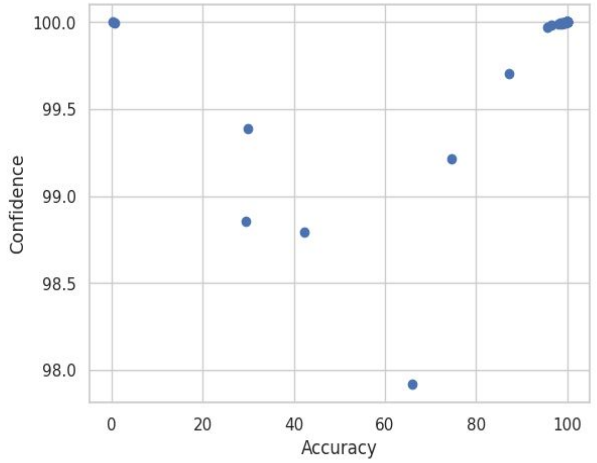


Dropout Probability=0.25

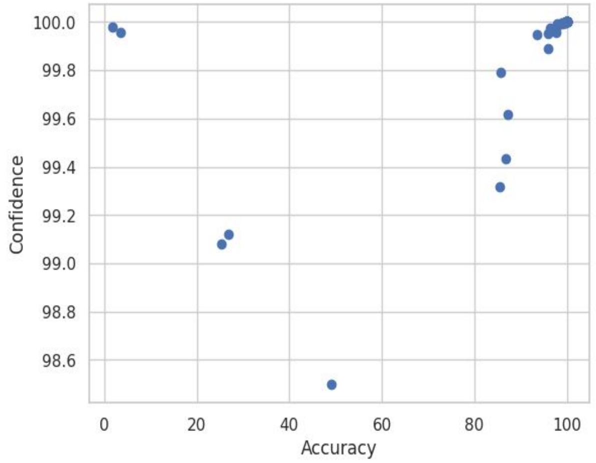


Dropout Probability=0.3

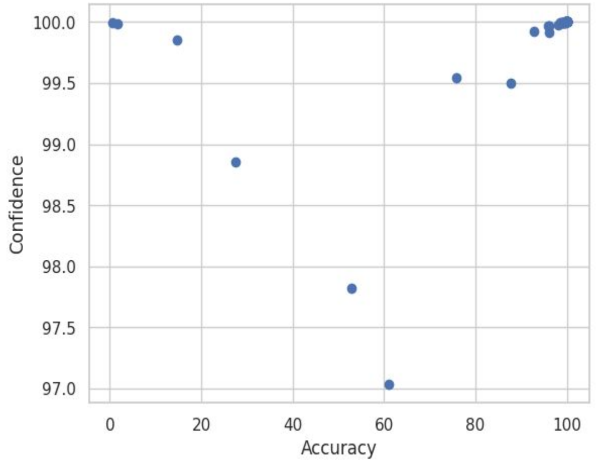
Confidence vs Accuracy



Dropout Probability=0.2



Dropout Probability=0.25



Dropout Probability=0.3

Adversarial Attack Types (STL-10)

1. **FGSM (Fast Gradient Sign Method)**
2. **L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno)**
3. **IAA (Imperceptibility Adversarial Attack)**

Good & Bad Images

Good Images

Good Images are images that are robust to adversarial attacks and are difficult to fool with small perturbations to their pixel values.

Bad Images

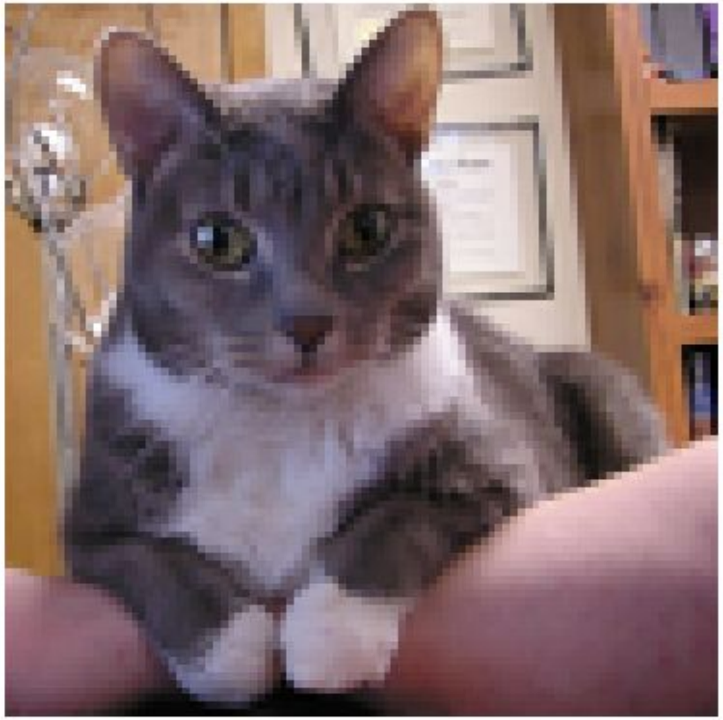
Bad Images are images that are vulnerable to adversarial attacks and can be easily fooled with small perturbations to their pixel values.

airplane

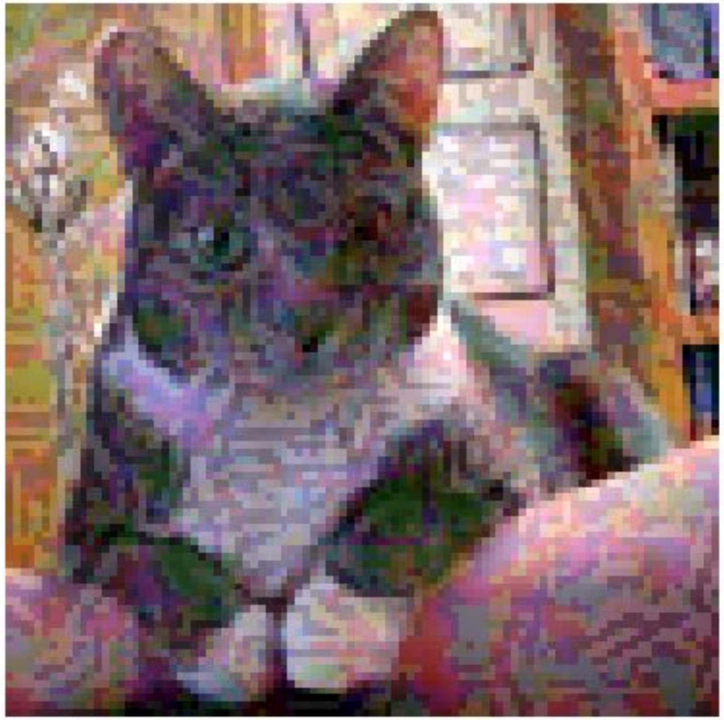
deer

Original vs FGSM

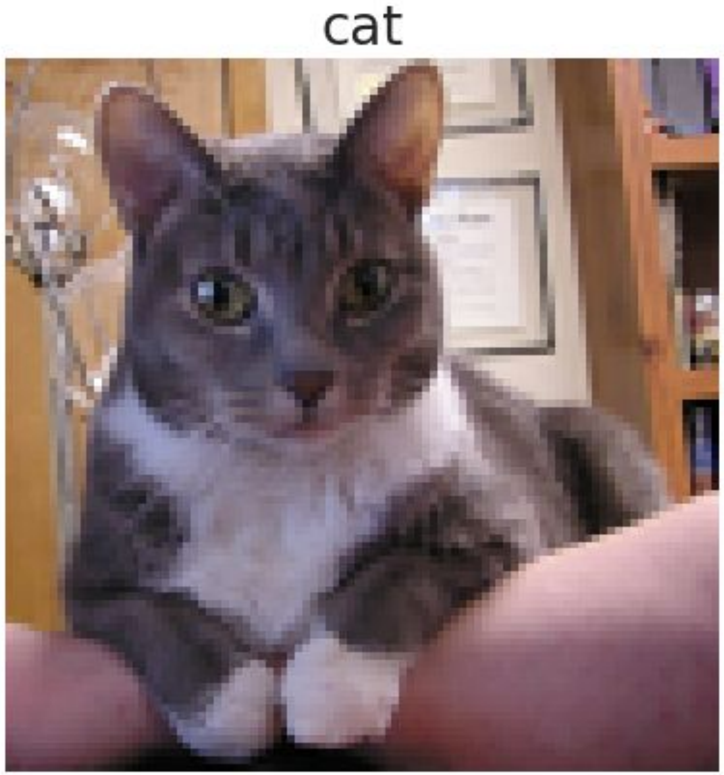
cat

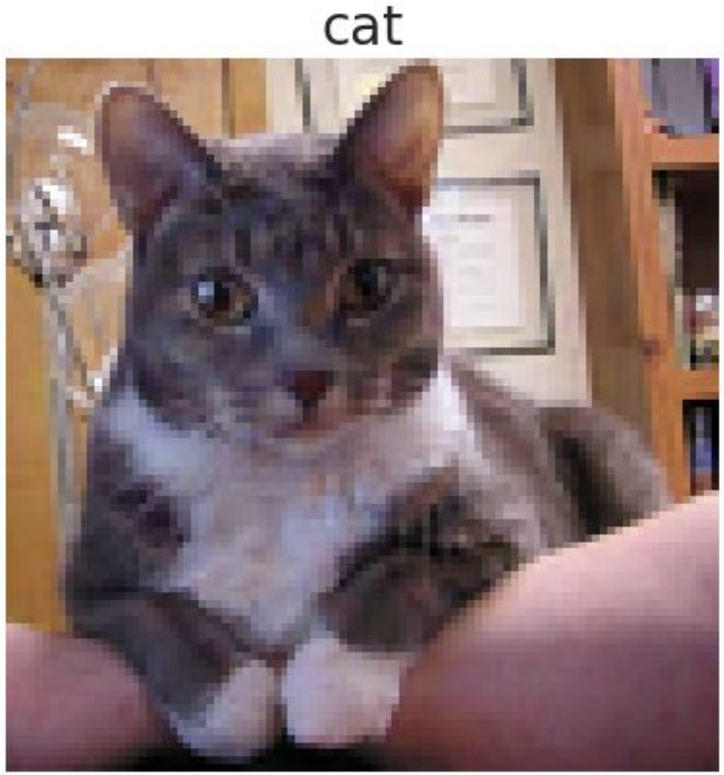


horse



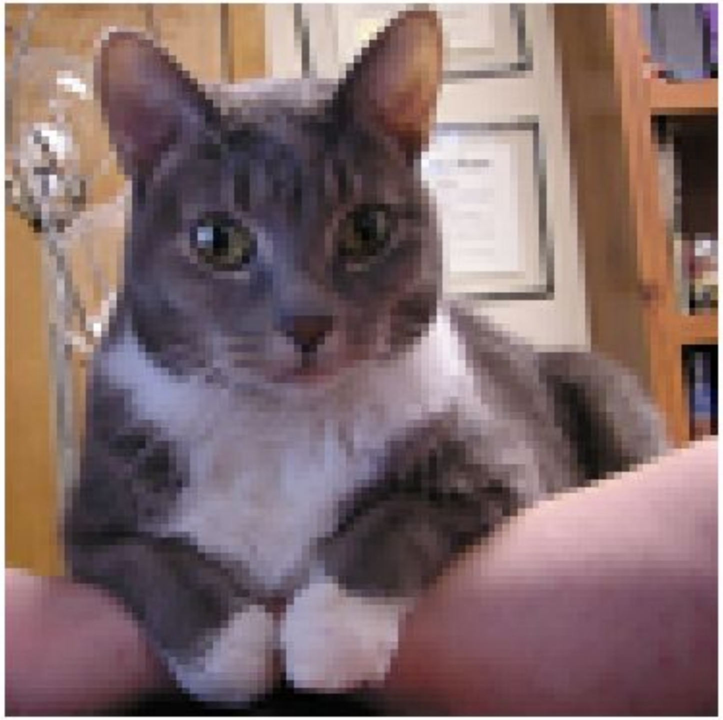
Original vs L-BFGS



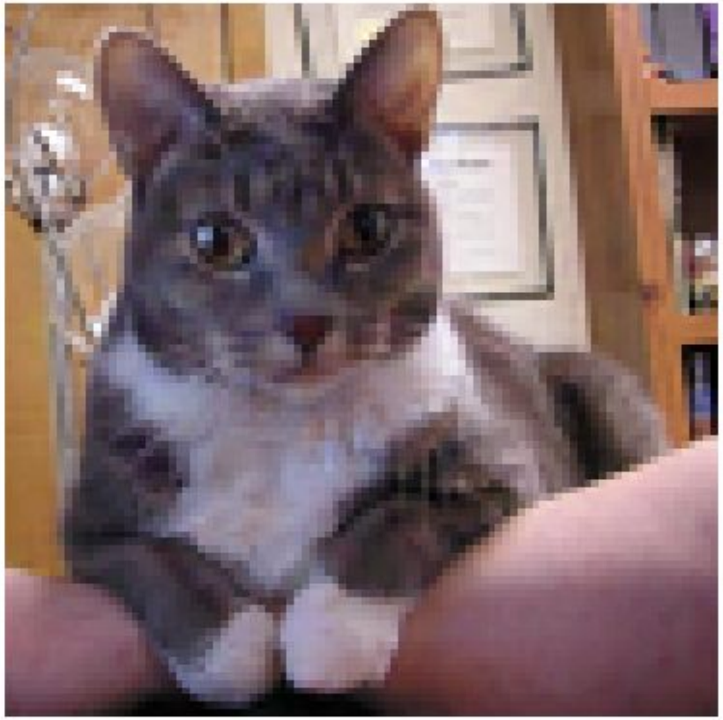


Original vs IAA

cat



cat



Activation Layers

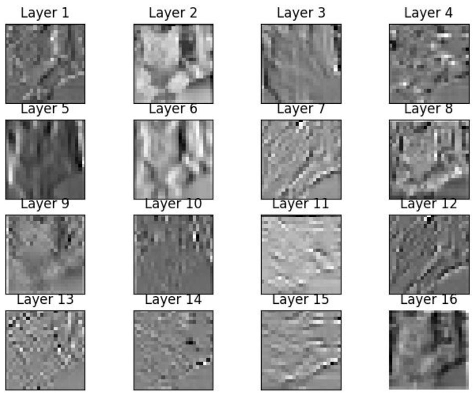
In our AlexNet model activation layers refer to the intermediate layers that capture and highlight specific features or patterns in input images or feature maps. These activation layers play a crucial role as they help in learning and representing meaningful features for the STL-10 dataset.

In our AlexNet Model we have : -

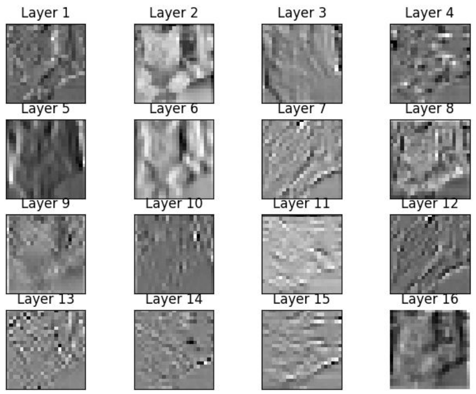
1. Conv1: 96 filters of size 11x11
2. Conv2: 256 filters of size 5x5
3. Conv3: 384 filters of size 3x3
4. Conv4: 384 filters of size 3x3
5. Conv5: 256 filters of size 3x3

16 Layers of Convolutional Layer-1

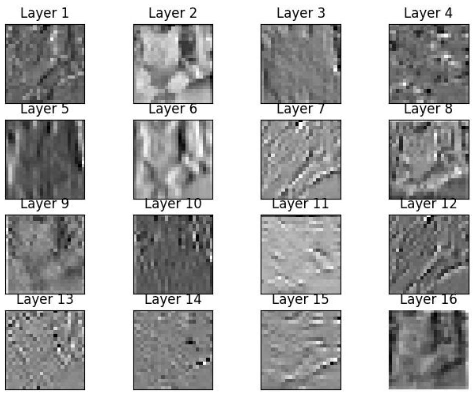
Original



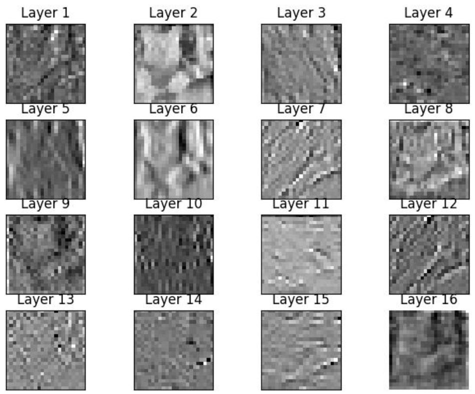
IIA



L-BFGS



FGSM



Conclusion / Outcomes

- From Confidence vs Accuracy graph it’s clear that both are linearly dependent on each other meaning with increase in Confidence, Accuracy Increases and vice-a-versa although there were some outliers.

- Implemented IAA method which generated perturbations on input data in a way that is difficult to detect by humans, and is also more robust compared to other existing methods like FGSM, L-BFGS etc.

- Built a Visualization Tool / Website for visualization that shows the behavior of the classifier under uncertainty and adversarial attack.

Future Scope/Work

- Techniques such as ensemble method etc. for estimating uncertainty can also be considered and explored.
- Other methods for generating Adversarial Image can also be implemented to see some interesting pattern, outlier etc.
- Different datasets like CIFAR-10, ImageNet-100 etc. can be used.
- Improve the current website by implementing Backend & Databasing

Thank You