CS 419
Project Report

Out of Distribution Detection

Group – 35

Ashok Nayak : 22B0455

Ayush Jha : 22B0051

Abhishek Hajra : 22B2424

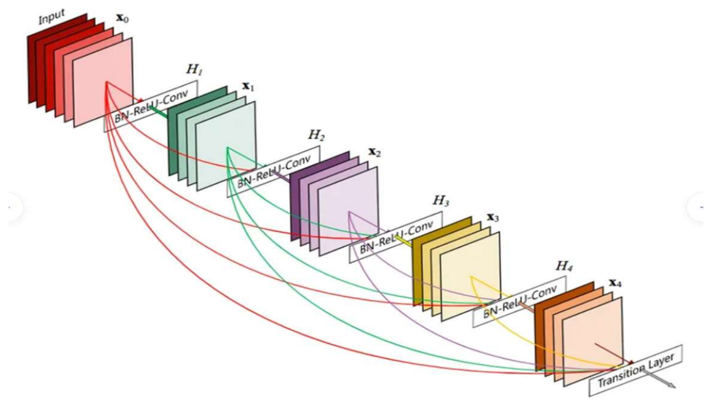
Aim:

The aim is to implement out-of-distribution (OOD) distribution in machine learning is to develop models that can accurately identify and handle data points that lie outside the training distribution.

Elements:

1. DenseNet

- DenseNet emphasizes dense connectivity between layers, where each layer receives input from all preceding layers within dense block
- In DenseNet, feature maps are concatenated along the channel dimension, allowing each layer to directly access the feature maps of all preceding layers.



2. ODIN (Out of Distribution detector of NN)

Odin is an anomaly detection technique for neural networks. It uses the perturbation of input data combined with temperature scaling to amplify the models confidence for in-distribution samples while reducing it for out of distribution samples

We can calculate ODIN score using

$ODIN\ score = \max\left(softmaxc(perturbed\ logits) - softmax(original\ logits)\right)$

Perturbed logits are the logits obtained by adding perturbations to the input data

Original logits are the logits obtained from the original data

Softmax is the softmax function applied to the logits to obtain probabilities

3. Pytorch_ood

PyTorch OOD (Out-of-Distribution) is a collection of techniques and tools developed within the PyTorch ecosystem to address the challenge of detecting out-of-distribution samples in machine learning models.

PyTorch OOD provides various methods and algorithms to help detect out-of-distribution samples, including:

1. ODIN
2. MaxSoftmax
3. Energy-based Models

Model-1

1. **Model-1** (standard ML model)

For the first part, we train a simple convolutional 2neural network (CNN) using CIFAR-10 dataset and evaluate its performance using the area under the receiver operating characteristic curve (AUC)

Steps

1. **Data Preparation :** Transformations are applied to CIFAR-10 dataset for normalization and conversion to tensors. Data is split into training and validation sets, and DataLoaders are created.
2. **Model Definition :** A simple CNN architecture with two convolutional layers followed by three fully connected layers is defined, along with the forward pass method.
3. **Training :** Loss = Cross Entropy
   Optimizer = SGD
   Epochs = 20
4. **Validation**
5. **Evaluation :** Using Predicted and True labels, the true positive rate, false positive rate, and AUC curve is calculated

Result:

AUC = 0.3432 (after training)

Model-2

2. **Model-2** (ODIN)

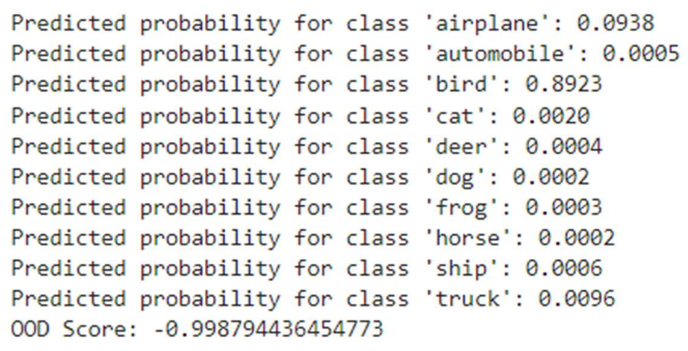
We’re using pre-trained WideResNet model for CIFAR-10 to process a test image through series of transformations, performs a forward pass through the model to obtain predicted probabilities for each class, and applies out-of-distributon (OOD) detection technique called ODIN to calculate ODIN score

As discussed above,

$$ODIN\ score = \max\left(softmaxc(perturbed\ logits) - softmax(original\ logits)\right)$$

Steps:

1. **Loading Pre-trained Model:** Load a pre-trained WideResNet model for CIFAR-10 dataset using the ‘WideResNet’ class from pytorch_ood.model.
2. **Image Transformation:** resizing image to 32x32 pixels, converting it to Pytorch tensor and normalizing pixel values
3. **Model Prediction**: Perform a forward pass through the model using the preprocessed image. The output of the model is a tensor containing raw scores(logits) for the class
4. **Probability Calculation**: Use softmax function to convert the logits into probabilities for each class. These probabilities indicate the model’s confidence in each class
5. **Out-of_Distribution(OOD) Detection:** Optionally, apply the ODIN (Out-of-Distribution Detector for Neural Networks) technique to calculate an OOD score for the input image. ODIN adjusts the input image and measures the change in output probabilities to detect anomalies.



```
Predicted probability for class 'airplane': 0.0938
Predicted probability for class 'automobile': 0.0005
Predicted probability for class 'bird': 0.8923
Predicted probability for class 'cat': 0.0020
Predicted probability for class 'deer': 0.0004
Predicted probability for class 'dog': 0.0002
Predicted probability for class 'frog': 0.0003
Predicted probability for class 'horse': 0.0002
Predicted probability for class 'ship': 0.0006
Predicted probability for class 'truck': 0.0096
OOD Score: -0.998794436454773
```

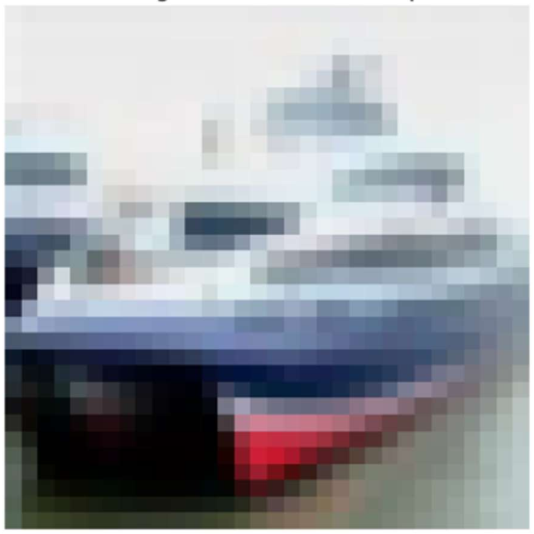
P.T.O

Classes of CIFAR-10

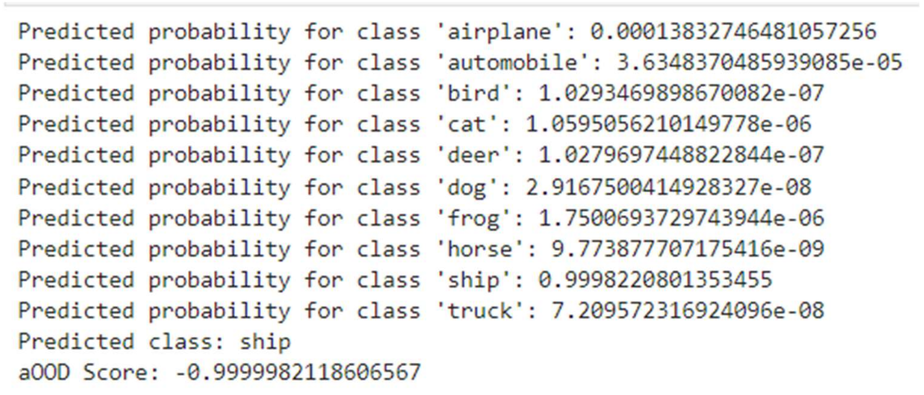
Validation:

Input Image:

Image from CIFAR-10: ship



Validation Result:



```
Predicted probability for class 'airplane': 0.00013832746481057256
Predicted probability for class 'automobile': 3.6348370485939085e-05
Predicted probability for class 'bird': 1.0293469898670082e-07
Predicted probability for class 'cat': 1.0595056210149778e-06
Predicted probability for class 'deer': 1.0279697448822844e-07
Predicted probability for class 'dog': 2.9167500414928327e-08
Predicted probability for class 'frog': 1.7500693729743944e-06
Predicted probability for class 'horse': 9.773877707175416e-09
Predicted probability for class 'ship': 0.9998220801353455
Predicted probability for class 'truck': 7.209572316924096e-08
Predicted class: ship
aOOD Score: -0.9999982118606567
```

Given Model predicts that the input image in ship and the largely negative OOD score represents this data lies in-distribution

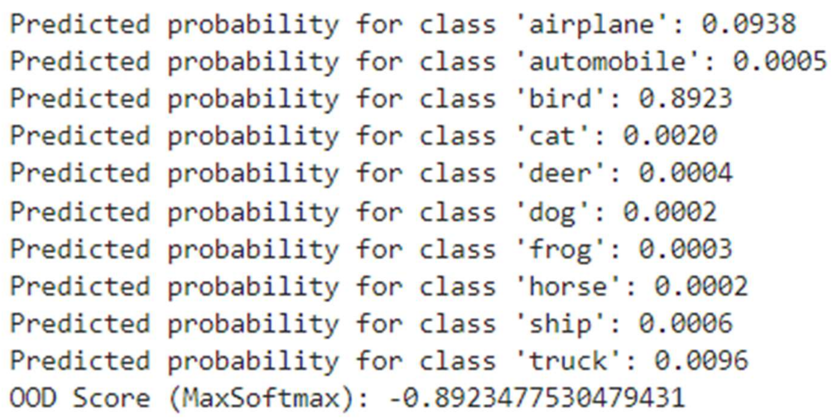
Model-3

3. **Model-3** (MaxSoftmax)

In this iteration we'll use a pre-trained WideResNet model to predict the class probabilities of a test image and computes its OOD score using MaxSoftmax detector (explained in pytorch_ood functionalities).

Steps:

1. **Load Pre-trained Model:** Loads a pre-trained WideResNet model for CIFAR-10 dataset.
2. **Image Transformation:** resizing image to 32x32 pixels.
3. **Model Prediction:** Passes the processed image through the model to obtain prediction.
4. **Probability Calculation**: Computes the predicted probabilities for each class by applying softmax activation to the model outputs.
5. **Create MaxSoftmax Detector:** Creates a MaxSoftmax detector using the pre-trained model.
6. **Compute OOD score:** Computes the OOD score using the MaxSoftmax detector for the input image.



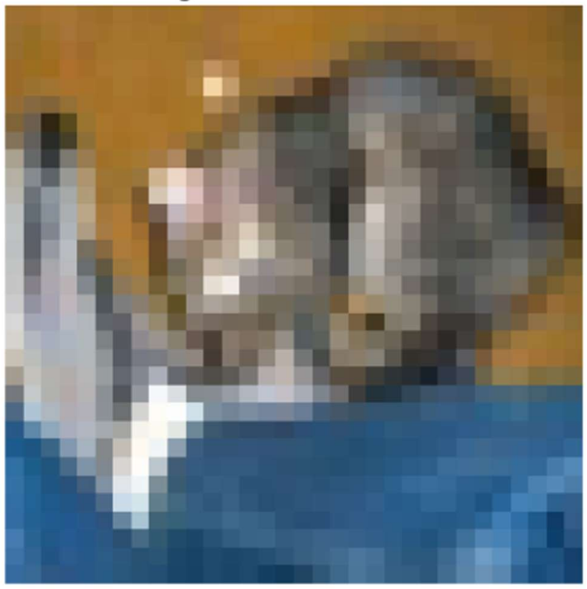
```
Predicted probability for class 'airplane': 0.0938
Predicted probability for class 'automobile': 0.0005
Predicted probability for class 'bird': 0.8923
Predicted probability for class 'cat': 0.0020
Predicted probability for class 'deer': 0.0004
Predicted probability for class 'dog': 0.0002
Predicted probability for class 'frog': 0.0003
Predicted probability for class 'horse': 0.0002
Predicted probability for class 'ship': 0.0006
Predicted probability for class 'truck': 0.0096
OOD Score (MaxSoftmax): -0.8923477530479431
```

OOD scores decreases by a little while using MaxSoftmax while calculating OOD score

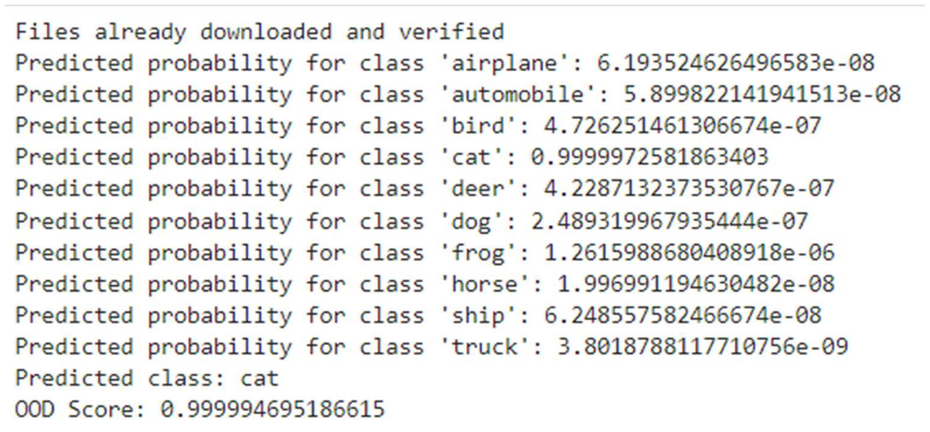
Validation

Test image:

Image from CIFAR-10: cat



Validation Results:



```
Files already downloaded and verified
Predicted probability for class 'airplane': 6.193524626496583e-08
Predicted probability for class 'automobile': 5.899822141941513e-08
Predicted probability for class 'bird': 4.726251461306674e-07
Predicted probability for class 'cat': 0.9999972581863403
Predicted probability for class 'deer': 4.2287132373530767e-07
Predicted probability for class 'dog': 2.489319967935444e-07
Predicted probability for class 'frog': 1.2615988680408918e-06
Predicted probability for class 'horse': 1.996991194630482e-08
Predicted probability for class 'ship': 6.248557582466674e-08
Predicted probability for class 'truck': 3.8018788117710756e-09
Predicted class: cat
OOD Score: 0.999994695186615
```

Model predicts this image is of a cat however, large OOD indicates this particular image not belonging to in-distribution of CIFAR-10

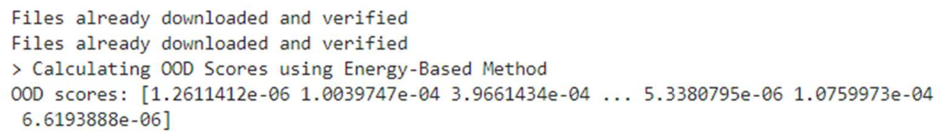
Model-4

4. Model-4 (Energy-based)

This model evaluates the out-of-distribution (OOD) scores using an energy-basesd method for WideResNet model trained on CIFAR-10 dataset. It computes the negative maximum logit as an energy function and calculates the OOD score using the sigmoid function

Steps:

1. **Define Energy Function**: Create a function to compute the energy of logits. In this case, it calculates the negative maximum logit value.
2. **Data Transformation**: Converting input images to tensors and normalizing them.
3. **Load CIFAR-10 Dataset:** Load the CIFAR-10 train and test datasets using torchvision.
4. **Data Loaders**: Create data loaders for the train and test datasets with a batch size of 64 and shuffle enabled for the train set.
5. **Calculate OOD score:** Iterate over the train and test loaders, moving inputs to the specified device (CPU), computing logits using the model, calculating energy scores using the defined function, applying the sigmoid function to obtain OOD scores, and appending the scores to a list.

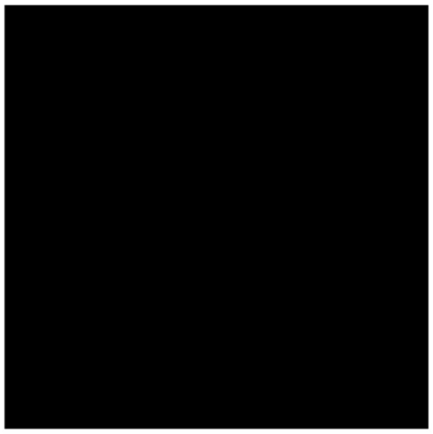


```
Files already downloaded and verified
Files already downloaded and verified
> Calculating OOD Scores using Energy-Based Method
OOD scores: [1.2611412e-06 1.0039747e-04 3.9661434e-04 ... 5.3380795e-06 1.0759973e-04
 6.6193888e-06]
```

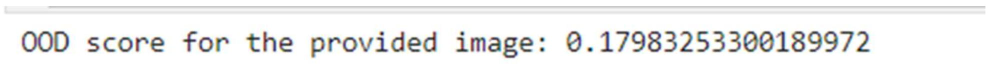
Due to use of Sigmoid in Energy-based approach, OOD score ranges from 0 to 1.

Validation

Test image: (black image with no features)



Validation Results:



```
OOD score for the provided image: 0.17983253300189972
```

Large OOD score represents this data point doesn’t lie in-distribution hence it’s an outlier.

P.T.O

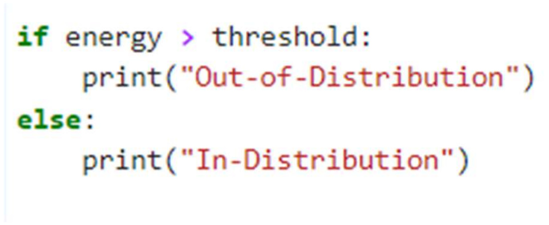
Model-5

5. **Model-5** (Energy-based Fine-tunned)

This Model fine-tunes a pre-trained WideResNet model on the CIFAR-10 dataset using energy-bounded learning. It defines an energy function to compute logits' energy, optimizes the model parameters using both logits and energy losses

Size:

1. **Define Energy Function :** Create a function to compute the energy of logits, which calculates the negative maximum logit value.
2. **Data Transformation**: Converting input images to tensors and normalizing them.
3. **Load CIFAR-10 Dataset:** Load the CIFAR-10 train and test datasets using torchvision
4. **Set Parameters:** Define parameters such as minimum and maximum margins for energy, the energy coefficient lambda, and optimizer settings.
5. **Fine-tune Model:** Train the model for 3 epochs using energy-bounded learning. Iterate over the training data, calculate the energy loss and logits loss, and update the model parameters using backpropagation.
6. **Check Energy Against Threshold:** Compare a computed energy value (not defined in the snippet) against the threshold to determine if the image is out-of-distribution or in-distribution.



```
if energy > threshold:
    print("Out-of-Distribution")
else:
    print("In-Distribution")
```

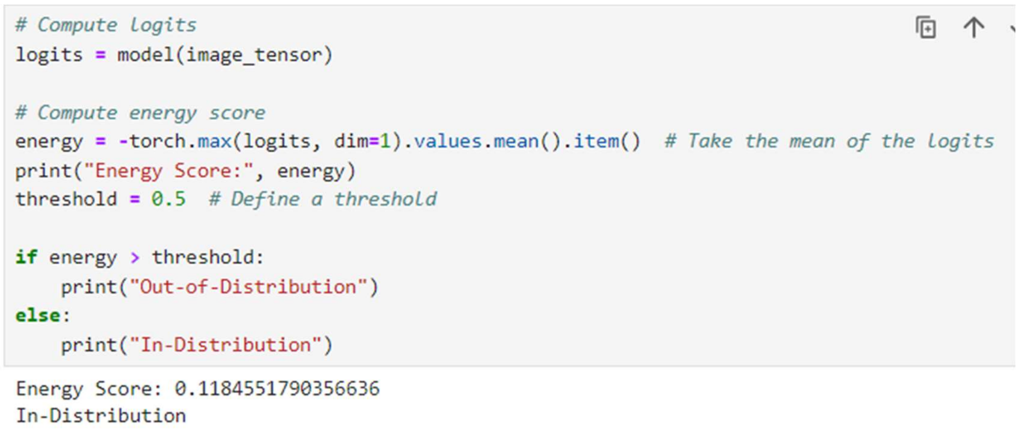
P.T.O

Validation:

We set a Threshold by ourselves and give test images as input

Based on the threshold model can comment if its in-distribution or out-of-distribution.

In-distribution test-image



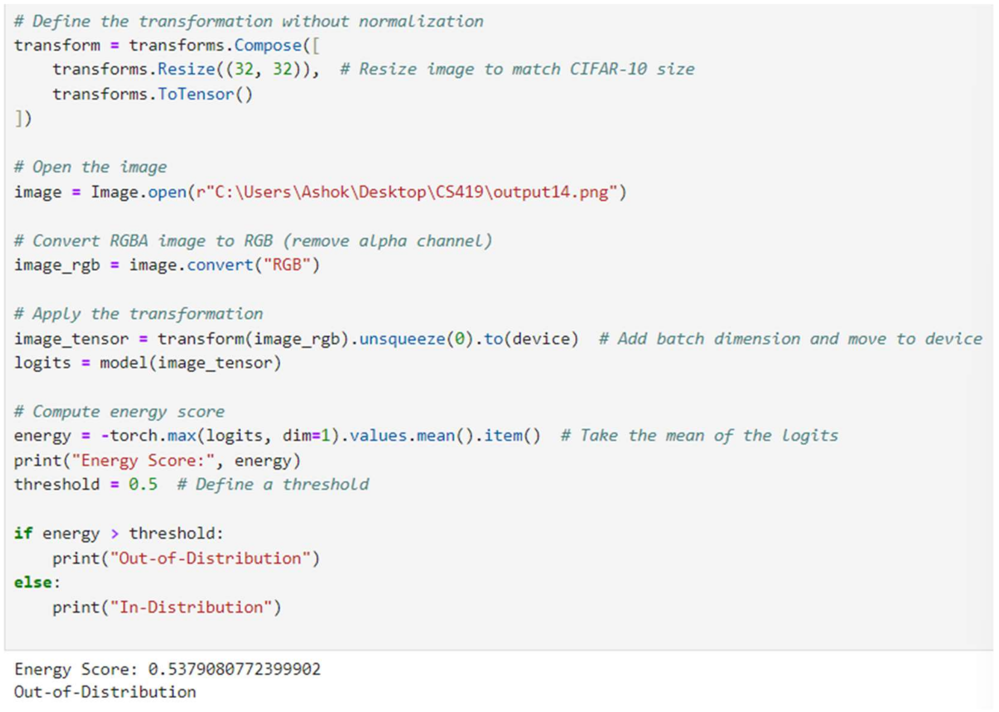
```
# Compute logits
logits = model(image_tensor)

# Compute energy score
energy = -torch.max(logits, dim=1).values.mean().item()  # Take the mean of the logits
print("Energy Score:", energy)
threshold = 0.5  # Define a threshold

if energy > threshold:
    print("Out-of-Distribution")
else:
    print("In-Distribution")
```

```
Energy Score: 0.1184551790356636
In-Distribution
```

Out-of-distribution test-image



```
# Define the transformation without normalization
transform = transforms.Compose([
    transforms.Resize((32, 32)),  # Resize image to match CIFAR-10 size
    transforms.ToTensor()
])

# Open the image
image = Image.open(r"C:\Users\Ashok\Desktop\CS419\output14.png")

# Convert RGBA image to RGB (remove alpha channel)
image_rgb = image.convert("RGB")

# Apply the transformation
image_tensor = transform(image_rgb).unsqueeze(0).to(device)  # Add batch dimension and move to device
logits = model(image_tensor)

# Compute energy score
energy = -torch.max(logits, dim=1).values.mean().item()  # Take the mean of the logits
print("Energy Score:", energy)
threshold = 0.5  # Define a threshold

if energy > threshold:
    print("Out-of-Distribution")
else:
    print("In-Distribution")
```

```
Energy Score: 0.5379080772399902
Out-of-Distribution
```