Artificial neural networks and their applications project
Assessment of Deep NNs on different Out of Distribution datasets in deep CNN

Made by Suleimanov Nikhat and Kirill Golubev

Introduction

In recent years, the advent of Deep Learning has revolutionised many fields of technology, leading to significant breakthroughs in image and speech recognition, natural language processing, and beyond. Among the various architectures developed, Deep Neural Networks and specifically Deep Convolutional Neural Networks have emerged as particularly powerful tools for image analysis. This project focuses on the exploration of DNNs through the lens of out-of-distribution detection—an area of critical importance for the deployment of AI systems in unpredictable, real-world environments.

Deep Neural Networks are sophisticated machine learning models composed of multiple layers of neurons that mimic the neural structures of the human brain. These networks are capable of learning highly complex and abstract patterns from large volumes of data. Deep Convolutional Neural Networks are a specialized kind of DNN, are particularly adept at processing pixel data and extracting hierarchical features, making them the architecture of choice for image classification tasks.

This project will mostly be focused on creating a deep CNN, using NN concatenation concept to implement out-of-distribution detection

Terms and concepts

The concept of ***Neural Network Concatenation*** involves the integration of multiple neural network architectures to leverage and combine their learned features for enhanced prediction performance.

Monte Carlo Dropout is a technique used in neural networks, to estimate uncertainty in predictions. Dropout is a regularization technique commonly used during training to prevent overfitting by randomly dropping units (along with their connections) from the neural network during each training iteration. Monte Carlo Dropout extends this idea to making predictions on new, unseen data. Instead of using dropout only during training, Monte Carlo Dropout applies dropout during inference as well. This involves making multiple predictions for a given input while stochastically dropping units according to the dropout probabilities. By averaging these predictions, Monte Carlo Dropout can provide a measure of uncertainty associated with the predictions.

Temperature scaling is a technique used to adjust the output probabilities of a softmax function in neural networks. In a softmax function, the output is a probability distribution over multiple classes, with each class assigned a probability between 0 and 1, and the sum of all probabilities equals 1.

The ***softmax function*** is defined as follows:

$$P(y = j|x) = \frac{e^{z_j/T}}{\sum_{k=1}^{K} e^{z_k/T}}$$

Where:

$P(y=j|x)$ - is the probability of class

j - given input x

zj is the unnormalized logit for class j

T is the temperature parameter.

In temperature scaling, the softmax function's temperature parameter T is adjusted to scale the logits before computing the probabilities. Higher temperatures lead to more "uniform" probability distributions, while lower temperatures sharpen the probabilities, making the model more confident in its predictions.

The temperature scaling process involves training a neural network as usual, then applying temperature scaling to its output probabilities. This can be achieved by adjusting the temperature parameter through a simple optimization process such as maximum likelihood estimation or cross-validation, aiming to calibrate the model's confidence levels. Temperature scaling is particularly useful in scenarios where the model's predicted probabilities do not align well with the true uncertainties, such as when the model is overconfident or underconfident in its predictions. By adjusting the temperature, the model's uncertainty can be better calibrated, leading to more reliable probability estimates.

The main idea behind ***Principal Component Analysis*** is to transform the original variables of a dataset into a new set of variables, called principal components, which are linear combinations of the original variables. These principal components are ordered in such a way that the first few capture the maximum variance present in the data. This means that by keeping only the first few principal components, we can retain most of the information in the original dataset while reducing its dimensionality.

Traditional fully connected layers (also known as dense layers) in neural networks expect input data to be in a one-dimensional format. Each neuron in a dense layer connects to all neurons in the previous layer, which assumes a flat, linear array of features. ***Flattening the image to a one-dimensional*** from a 2-D or 3-D format (for grayscale or color images, respectively) to a 1-D array makes it compatible with these layers.

Confusion Matrix:

For a classification problem with N classes, the confusion matrix is a $N \times N$ matrix where each row represents the instances in an actual class, while each column represents the instances in a predicted class. Here is a basic example for a binary classification problem (two classes: Positive and Negative):

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Precision is how accurate the model predictions are:

$$\text{Precision} = \frac{TP}{(TP+FP)}.$$

The harmonic mean of Precision and Recall. It is used as a measure of a test’s accuracy.

$$\text{F1 Score} = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}.$$

Implementation

A model of deep CNN was found and later used in NN concatenation. Base architecture comprises convolutional, pooling and dense layers tailored for image data. Input 28x28 images are passed through 3 convolutional blocks with kernels of size 3x3 and appropriate padding to retain spatial dimensions. Along with conv blocks, 2x2 max pooling reduces dimensions while preserving key activations. Leaky ReLU activation is used for convolutional layers and ReLU for dense layers. Batch normalization is performed on each layer. Final classification layer is a softmax activated dense layer for 10-way probabilistic prediction. An Adam optimizer with cross-entropy loss is used for end-to-end model training.

The first part of the architecture defines a CNN that includes two convolutional layers, each followed by a max pooling layer, and includes dropout and fully connected layers:

1. Convolutional Layer (conv1):
    - Input Channels: 1 (for grayscale images)
    - Output Channels: 32
    - Kernel Size: 5x5
    - Stride: 1
    - Padding: 2
    - This layer creates 32 feature maps and uses padding to preserve the dimensionality of the input image.
2. Max Pooling Layer (maxpool1):
    - Kernel Size: 2x2
    - Stride: 2
    - This layer reduces the spatial dimensions (height and width) of the input feature maps by half.
3. Second Convolutional Layer (conv2):
    - Input Channels: 32
    - Output Channels: 64
    - Kernel Size: 5x5
    - Stride: 1
    - Padding: 2
    - This layer further processes the feature maps from the first pooling layer, increasing the depth to 64.
4. Second Max Pooling Layer (maxpool2):
    - Kernel Size: 2x2
    - Stride: 2
    - Further reduces the spatial dimensions of the feature maps.
5. Dropout (dropout):
    - Dropout Rate: 0.5
    - Randomly zeroes some of the elements of the input tensor with probability 0.5 at each step during training time, which helps prevent overfitting.
6. Fully Connected Layers (fc1, fc2):
    - fc1: Transforms the flattened output of the last pooling layer to 1000 features.
    - fc2: Reduces the dimension from 1000 to the number of classes (10).

Second part is modular, using "nn.sequential", which is an ordered container of modules. The network is structured into a series of convolutional blocks followed by max pooling and fully connected layers:

1. Convolutional Blocks, created by a helper function make_cnn_block:
    - Starts from 1 channel, progressively increasing the channels through each block (16, 32, 64, 128).

- Each block likely applies a convolution followed by a ReLU activation.

2. Max Pooling Layers, reducing the spatial dimension after specific blocks:
3. Fully Connected Network:
   - Layer 1: Linear transformation from 512 to 256 neurons, followed by ReLU.
   - Layer 2: Linear transformation from 256 to 128 neurons, followed by ReLU.
   - Layer 3: Final layer reducing from 128 to 10, the number of classes.

Results

Through training on the fashion MNIST dataset, the network achieved high accuracy, demonstrating its effectiveness in handling well-distributed, in-distribution data. Loss and accuracy throughout the epochs are shown on the image 1 and 2. Validation accuracy reached 0.9213.

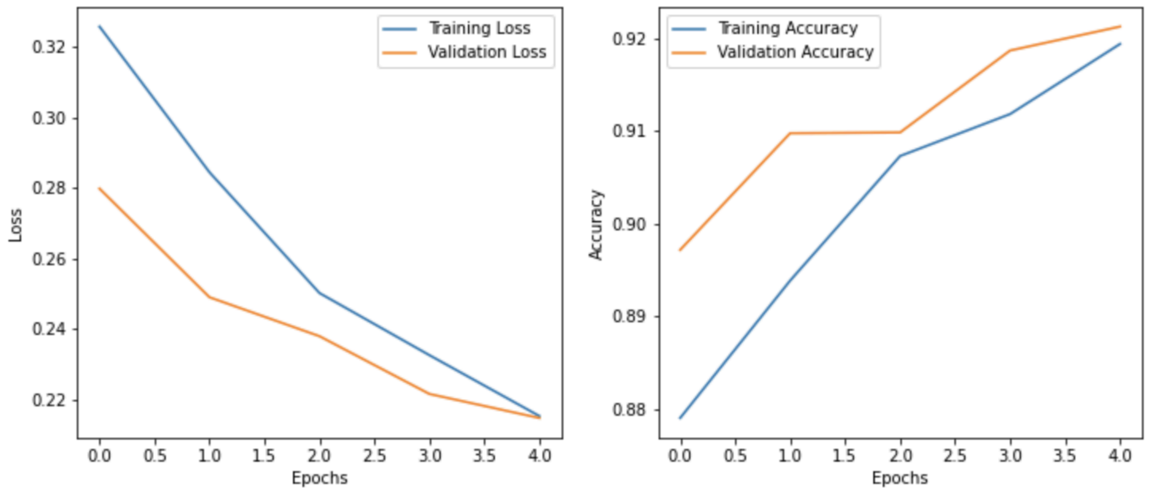
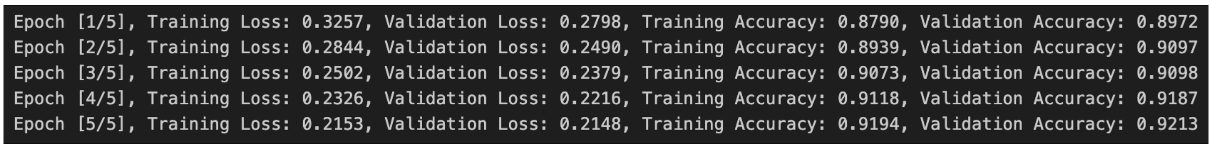


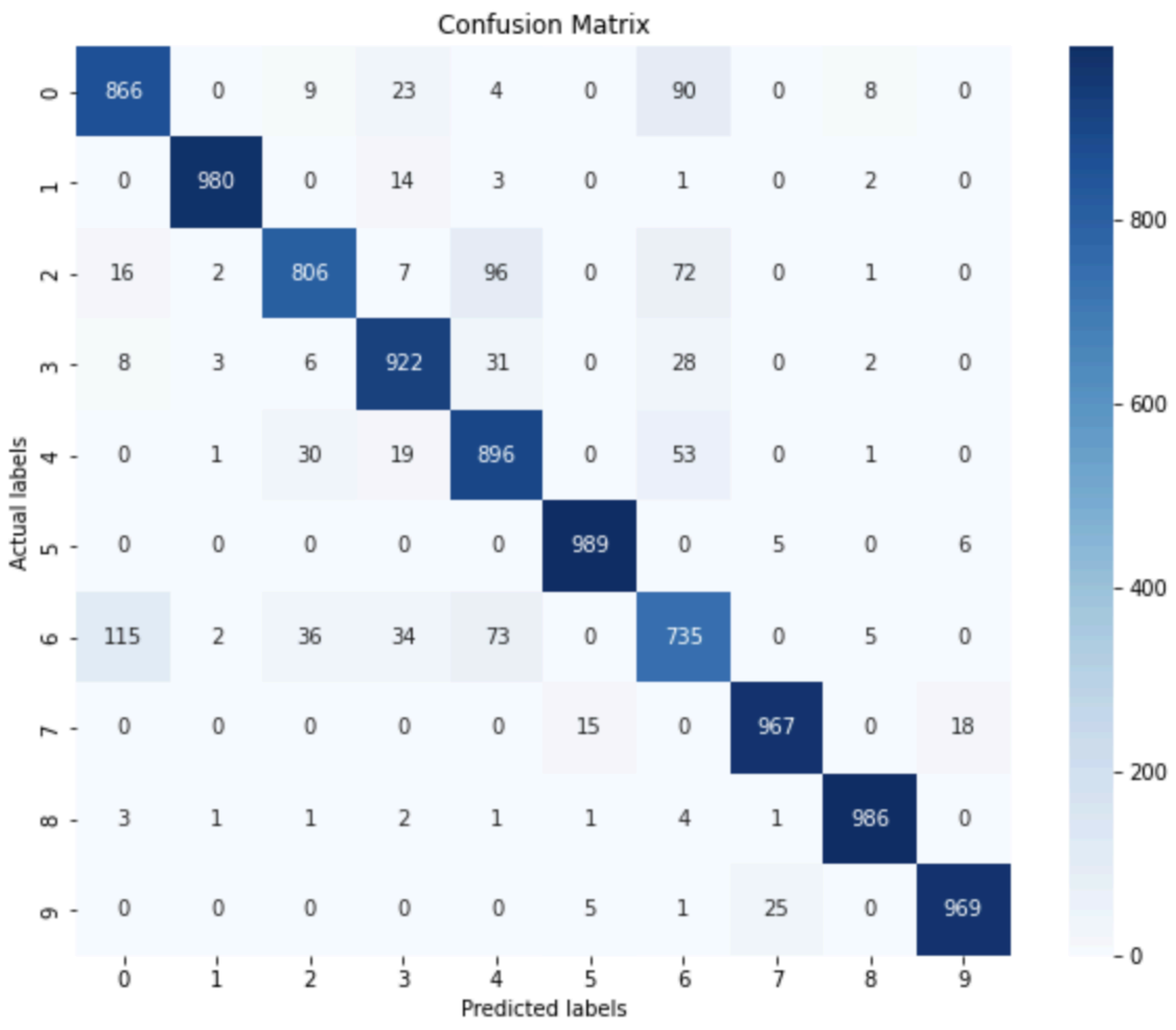
Image 1



```
Epoch [1/5], Training Loss: 0.3257, Validation Loss: 0.2798, Training Accuracy: 0.8790, Validation Accuracy: 0.8972
Epoch [2/5], Training Loss: 0.2844, Validation Loss: 0.2490, Training Accuracy: 0.8939, Validation Accuracy: 0.9097
Epoch [3/5], Training Loss: 0.2502, Validation Loss: 0.2379, Training Accuracy: 0.9073, Validation Accuracy: 0.9098
Epoch [4/5], Training Loss: 0.2326, Validation Loss: 0.2216, Training Accuracy: 0.9118, Validation Accuracy: 0.9187
Epoch [5/5], Training Loss: 0.2153, Validation Loss: 0.2148, Training Accuracy: 0.9194, Validation Accuracy: 0.9213
```

Image 2

The Confusion Matrix, which is shown on image 3, for the MNIST dataset revealed that while most digits were recognized with high precision and recall. While certain classes like ‘6’ exhibited lower accuracy.



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	866	0	9	23	4	0	90	0	8	0
1	0	980	0	14	3	0	1	0	2	0
2	16	2	806	7	96	0	72	0	1	0
3	8	3	6	922	31	0	28	0	2	0
4	0	1	30	19	896	0	53	0	1	0
5	0	0	0	0	0	989	0	5	0	6
6	115	2	36	34	73	0	735	0	5	0
7	0	0	0	0	0	15	0	967	0	18
8	3	1	1	2	1	1	4	1	986	0
9	0	0	0	0	0	5	1	25	0	969

Image 3

PCA Analysis (image 4) indicated that the first few principal components captured a significant proportion of the variance within the dataset. Around 90 components cover about 90% variance, while 190 components explain around 95%. As components increase, variance reaches 100%, aligning with the original 784 dimensions.

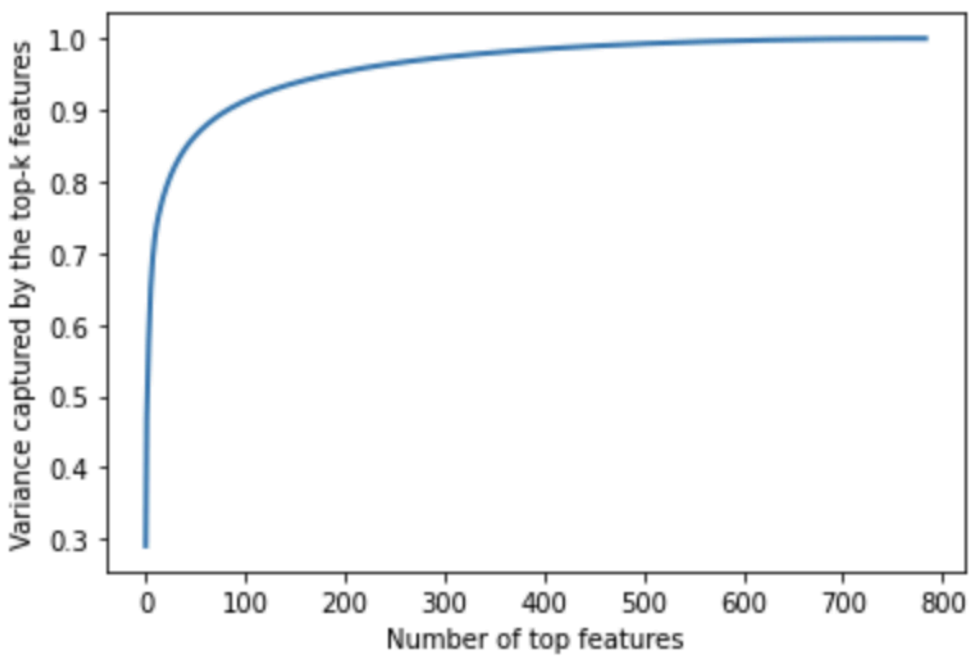


Image 4

The following visualization illustrates images before PCA and post-PCA application, showcasing the preservation of key information in the reduced dimensionality.

Original Images:
Trouser
Pullover
Sandal
Dress
Sneaker
Ankle boot
Pullover
Sandal
Coat
Ankle boot
Shirt
Coat
Sneaker
Trouser
Dress
T-shirt/top
Trouser
Coat
Trouser
Shirt
Bag
Pullover
Pullover
Shirt
T-shirt/top
Dress
Bag
Sandal
Shirt
Ankle boot
Reduced Images:
Trouser
Pullover
Sandal
Dress
Sneaker
Ankle boot
Pullover
Sandal
Coat
Ankle boot
Shirt
Coat
Sneaker
Trouser
Dress
T-shirt/top
Trouser
Coat
Trouser
Shirt
Bag
Pullover
Pullover
Shirt
T-shirt/top
Dress
Bag
Sandal
Shirt
Ankle boot

Image 5

t-Distributed Stochastic Neighboring Entities is applied on the first five thousand rows to reduce the number of dimensions. The following image 5 shows significant improvement in classification wherein each class is properly clustered and separated from one another.

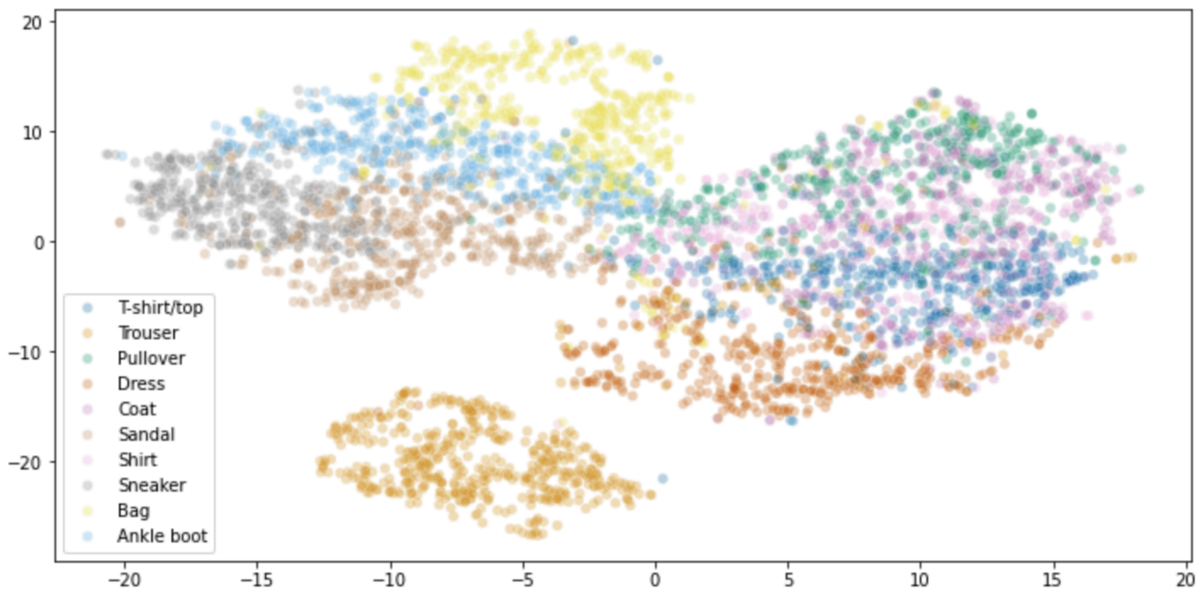
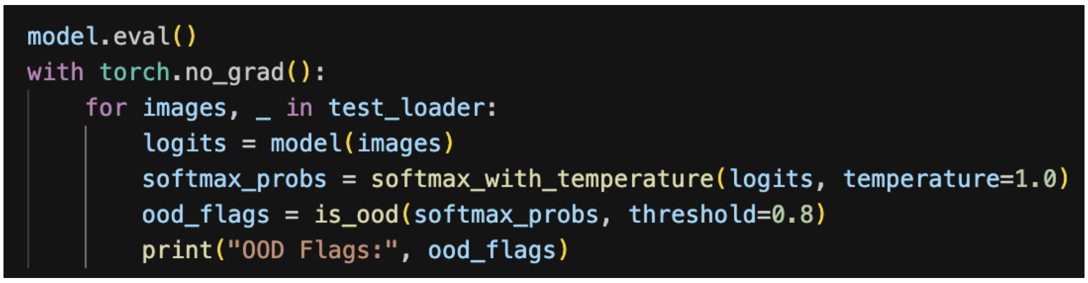


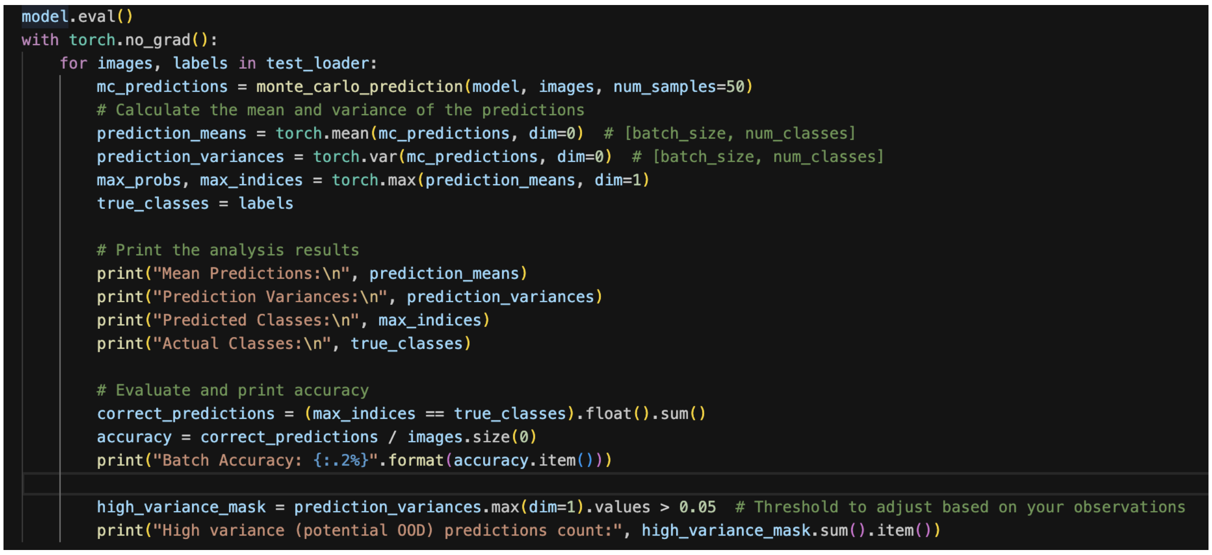
Image 6

Different model evaluations:



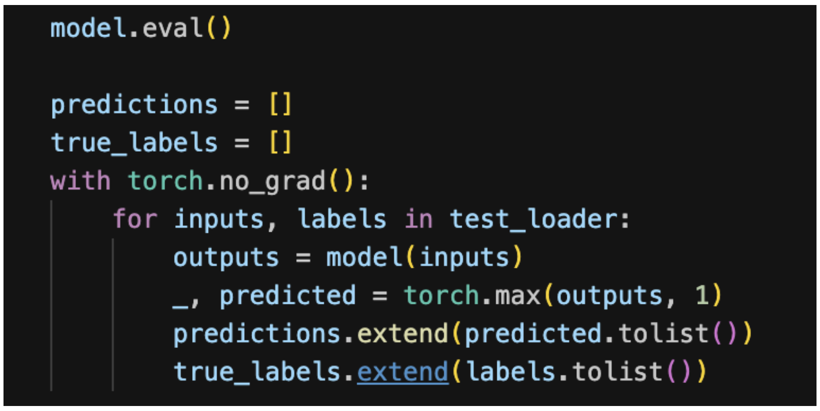
```
model.eval()
with torch.no_grad():
    for images, _ in test_loader:
        logits = model(images)
        softmax_probs = softmax_with_temperature(logits, temperature=1.0)
        ood_flags = is_ood(softmax_probs, threshold=0.8)
        print("OOD Flags:", ood_flags)
```

Image 7 - evaluation to check “softmax with temperature scaling”



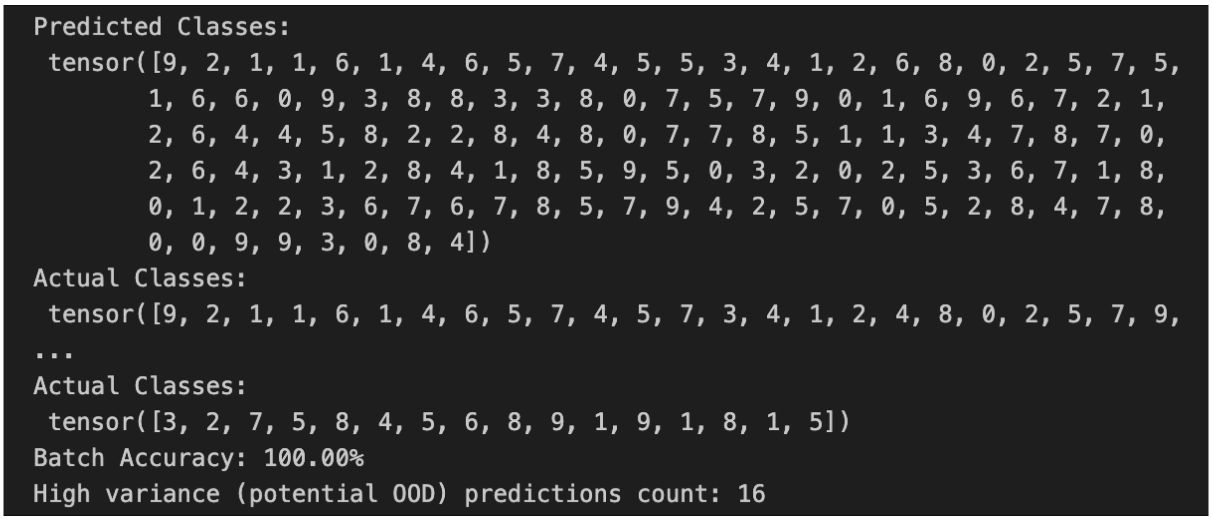
```python
model.eval()
with torch.no_grad():
    for images, labels in test_loader:
        mc_predictions = monte_carlo_prediction(model, images, num_samples=50)
        # Calculate the mean and variance of the predictions
        prediction_means = torch.mean(mc_predictions, dim=0)  # [batch_size, num_classes]
        prediction_variances = torch.var(mc_predictions, dim=0)  # [batch_size, num_classes]
        max_probs, max_indices = torch.max(prediction_means, dim=1)
        true_classes = labels

        # Print the analysis results
        print("Mean Predictions:\n", prediction_means)
        print("Prediction Variances:\n", prediction_variances)
        print("Predicted Classes:\n", max_indices)
        print("Actual Classes:\n", true_classes)

        # Evaluate and print accuracy
        correct_predictions = (max_indices == true_classes).float().sum()
        accuracy = correct_predictions / images.size(0)
        print("Batch Accuracy: {:.2%}".format(accuracy.item()))

        high_variance_mask = prediction_variances.max(dim=1).values > 0.05  # Threshold to adjust based on your observations
        print("High variance (potential OOD) predictions count:", high_variance_mask.sum().item())
```

Image 8 - evaluation to check “Monte Carlo Dropout”



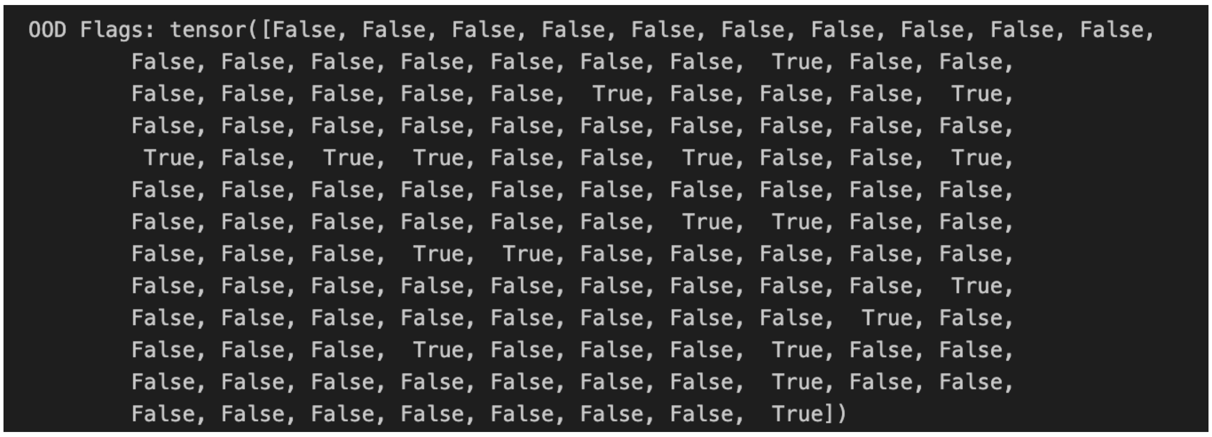
```python
model.eval()

predictions = []
true_labels = []
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        predictions.extend(predicted.tolist())
        true_labels.extend(labels.tolist())
```

Image 9 - default evaluation



```
Predicted Classes:
 tensor([9, 2, 1, 1, 6, 1, 4, 6, 5, 7, 4, 5, 5, 3, 4, 1, 2, 6, 8, 0, 2, 5, 7, 5,
        1, 6, 6, 0, 9, 3, 8, 8, 3, 3, 8, 0, 7, 5, 7, 9, 0, 1, 6, 9, 6, 7, 2, 1,
        2, 6, 4, 4, 5, 8, 2, 2, 8, 4, 8, 0, 7, 7, 8, 5, 1, 1, 3, 4, 7, 8, 7, 0,
        2, 6, 4, 3, 1, 2, 8, 4, 1, 8, 5, 9, 5, 0, 3, 2, 0, 2, 5, 3, 6, 7, 1, 8,
        0, 1, 2, 2, 3, 6, 7, 6, 7, 8, 5, 7, 9, 4, 2, 5, 7, 0, 5, 2, 8, 4, 7, 8,
        0, 0, 9, 9, 3, 0, 8, 4])
Actual Classes:
 tensor([9, 2, 1, 1, 6, 1, 4, 6, 5, 7, 4, 5, 7, 3, 4, 1, 2, 4, 8, 0, 2, 5, 7, 9,
...
Actual Classes:
 tensor([3, 2, 7, 5, 8, 4, 5, 6, 8, 9, 1, 9, 1, 8, 1, 5])
Batch Accuracy: 100.00%
High variance (potential OOD) predictions count: 16
```

Image 10



```
OOD Flags: tensor([False, False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False,  True, False, False,
        False, False, False, False, False,  True, False, False, False,  True,
        False, False, False, False, False, False, False, False, False, False,
         True, False,  True,  True, False, False,  True, False, False,  True,
        False, False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False,  True,  True, False, False,
        False, False, False,  True,  True, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,  True,
        False, False, False, False, False, False, False, False,  True, False,
        False, False, False,  True, False, False, False,  True, False, False,
        False, False, False, False, False, False, False,  True, False, False,
        False, False, False, False, False, False, False,  True])
```

Image 11

The OOD Detection mechanisms, particularly through Monte Carlo Dropout, showed that the network’s confidence in its predictions dropped significantly when faced with CIFAR-10 images. This drop in confidence, coupled with higher prediction variances, highlighted the network's awareness of its uncertainty in these cases (Image 10 and 11).

Model was also tested directly on CIFAR-10 dataset to check the percentage of low-confidence predictions. Final percentage of low-confidence predictions: 77.59%.

Conclusion

The project underscored in some areas and tests, which is likely possible due to the small number of epochs. The number itself was chosen because of the low powered laptop, which was used during training. But overall, the model can be ready for final fine tuning and training to achieve better results. In any case, there was a lot of information and experience gained from this study towards the necessity for ongoing improvements in neural network architectures, training methodologies, and OOD handling mechanisms to build neural networks that are both powerful and reliable across a broad spectrum of real-world applications.

Main references

[1] Web-page - Classification of Fashion MNIST Dataset Using DNN and CNN, link: <https://medium.com/@ksaraswat_97923/fashion-mnist-image-classification-deep-learning-using-dnn-and-cnn-675f786d1b0d>

[2] Pub - Performance Analysis of Out-of-Distribution Detection on Various Trained Neural Networks, link: <https://arxiv.org/pdf/2103.15580>

[3] Pub - Cifar-10 (canadian institute for advanced research), link: <http://www.cs.toronto.edu/ ~kriz/cifar.html>

[4] Pub - Towards evaluating the robustness of neural networks," in 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017