Part 1 - Training a CNN on SVHN

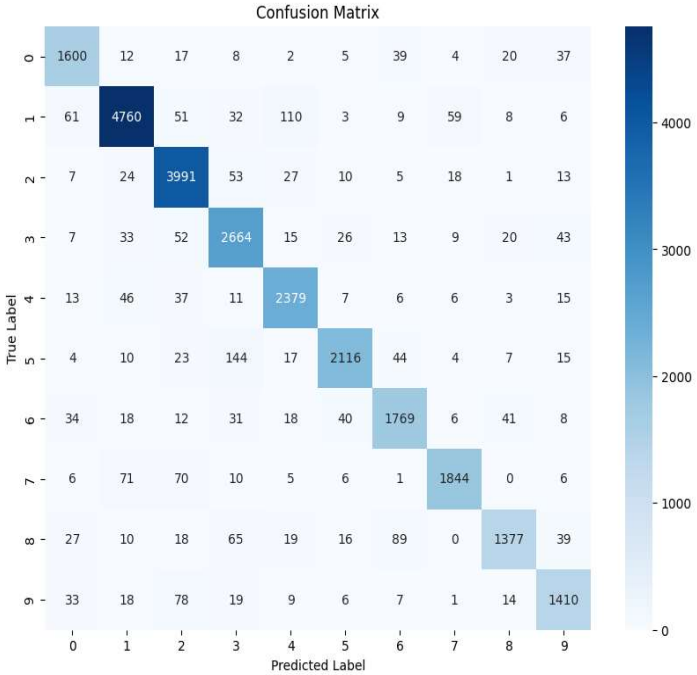
we trained a Convolutional Neural Network (CNN) on the SVHN (Street View House Numbers) dataset using PyTorch, achieving a test accuracy of 91.85% which exceeds the target of 90%. The dataset is preprocessed with normalization to the range [-1, 1] and split into training (80%), validation (20%), and test sets. The model architecture consists of three convolutional blocks with Batch Normalization, ReLU activation, Max Pooling, and Dropout for regularization, followed by two fully connected layers. Training is performed using Stochastic Gradient Descent (SGD) with a learning rate of 0.01 and momentum of 0.9, alongside CrossEntropyLoss for optimization. Over 10 epochs, the model shows consistent improvement: training accuracy increases from 41.31% to 88.20%, while validation accuracy rises from 75.85% to 91.86%. The final test accuracy of 91.85% indicates strong generalization, with no significant overfitting due to the effective use of dropout and batch normalization. Key factors contributing to the model's performance include proper normalization, regularization techniques, and a well-designed architecture. Potential improvements could involve adding data augmentations, using advanced optimizers like Adam, or experimenting with deeper architectures to further enhance accuracy.

```
Epoch 1/10, Loss: 1.6631, Train Accuracy: 41.31%
Validation Loss: 0.7560, Validation Accuracy: 75.85%
Epoch 2/10, Loss: 0.9498, Train Accuracy: 69.24%
Validation Loss: 0.5459, Validation Accuracy: 83.92%
Epoch 3/10, Loss: 0.7506, Train Accuracy: 76.65%
Validation Loss: 0.4805, Validation Accuracy: 85.57%
Epoch 4/10, Loss: 0.6345, Train Accuracy: 80.48%
Validation Loss: 0.3766, Validation Accuracy: 89.23%
Epoch 5/10, Loss: 0.5644, Train Accuracy: 82.91%
Validation Loss: 0.3520, Validation Accuracy: 89.68%
Epoch 6/10, Loss: 0.5013, Train Accuracy: 84.83%
Validation Loss: 0.3409, Validation Accuracy: 90.14%
Epoch 7/10, Loss: 0.4676, Train Accuracy: 86.00%
Validation Loss: 0.3153, Validation Accuracy: 90.97%
Epoch 8/10, Loss: 0.4366, Train Accuracy: 86.95%
Validation Loss: 0.3065, Validation Accuracy: 91.16%
Epoch 9/10, Loss: 0.4127, Train Accuracy: 87.78%
Validation Loss: 0.2959, Validation Accuracy: 91.48%
Epoch 10/10, Loss: 0.3964, Train Accuracy: 88.20%
Validation Loss: 0.2914, Validation Accuracy: 91.86%
Test Accuracy: 91.85%
```

Analysis:

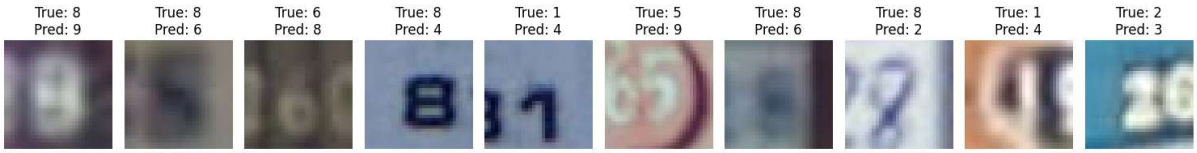
Confusion Matrix Analysis

The confusion matrix provides a comprehensive breakdown of the model's classification performance, illustrating the frequency of correct and incorrect predictions for each class. The **diagonal elements represent correctly classified instances**, while off-diagonal values highlight misclassifications, where the true and predicted labels do not match. High values along the diagonal indicate that the model performs well on those digits, whereas larger off-diagonal values suggest frequent misclassifications. In this matrix, the model exhibits strong performance on digits like **1 (4760 correct classifications)** and **2 (3991 correct classifications)** but struggles more with digits such as **8 and 9**, which are often confused due to their structural similarities. Additionally, misclassifications between **1 and 4, or 5 and 3**, suggest that overlapping features may be causing ambiguity. Another noticeable pattern is the occasional confusion between **8 and 6**, which could be due to their circular shapes. The matrix



was generated using sklearn.metrics.confusion_matrix, which compares the model's predicted labels with the true labels to quantify the classification errors. By analyzing these patterns, we can identify areas where the model needs improvement, such as incorporating more robust feature extraction or refining training techniques to better differentiate visually similar digits.

Misclassified Images Analysis



The 10 misclassified images highlight specific cases where the model struggles, primarily due to the visual similarity between certain digits. For example, a **true label of 1 is misclassified as 4 twice**, and two instances of **8 are misclassified as 6** . These errors are likely due to the structural resemblance of these digits, where the model fails to distinguish subtle differences in curvature or contrast. Some misclassifications, such as **6 predicted as 8**, may stem from variations in font style, distortions, or poor image quality. Additionally, digits like **8 misclassified as 9** suggest challenges in recognizing incomplete strokes or faded sections. Since the SVHN dataset consists of house number images captured in real-world scenarios, factors like background noise, lighting conditions, and partial occlusions further contribute to classification errors. These misclassified images were extracted by identifying instances where the predicted labels did not match the ground truth during evaluation. Visualizing these errors provides valuable insights into the model's weaknesses.

Classification Report Analysis

The classification report provides a detailed breakdown of the model's performance using **precision, recall, and F1-score** for each digit class. **Precision** measures how often the model's predictions for a class are correct, while **recall** indicates how well the model identifies all actual instances of that class. The **F1-score** balances these two metrics to give an overall measure of accuracy for each digit. For example, digit **1** has **high precision (0.95) and recall (0.93)**, indicating that the model rarely misclassifies other digits as 1 and correctly identifies most instances of 1. However, some digits, such as **3 and 9, have lower precision (0.89 and 0.89, respectively)**, suggesting that they are frequently confused with visually similar digits. The **macro average (0.91)** provides the unweighted mean across all classes, while the **weighted average (0.91)** considers the number of samples per class, reflecting the overall model performance. The total **accuracy of 92%** indicates strong overall classification performance. This report was generated using

Classification Report:

	precision	recall	f1-score	support
0	0.89	0.92	0.90	1744
1	0.95	0.93	0.94	5099
2	0.92	0.96	0.94	4149
3	0.88	0.92	0.90	2882
4	0.91	0.94	0.93	2523
5	0.95	0.89	0.92	2384
6	0.89	0.89	0.89	1977
7	0.95	0.91	0.93	2019
8	0.92	0.83	0.87	1660
9	0.89	0.88	0.88	1595
accuracy			0.92	26032
macro avg	0.91	0.91	0.91	26032
weighted avg	0.92	0.92	0.92	26032

sklearn.metrics.classification_report, which computes these metrics based on the true and predicted labels. These insights suggest that while the model is effective, further refinements—such as better handling of ambiguous digits like 3, 8, and 9—could improve classification accuracy.

Discussion of Model Weaknesses and Improvement Tips

The model exhibits several weaknesses that stem from both the dataset and the model architecture. One major issue is the confusion between visually similar digits, such as **1 and 4 or 8 and 6**, as observed in the confusion matrix and misclassified images. This is likely due to the **limited depth of the model**, which may not capture subtle differences in shape and curvature. Additionally, the lack of extensive **data augmentation** limits the model's exposure to variations in the input data, such as **rotated, blurred, or distorted digits**, reducing its ability to generalize. **Class imbalance also plays a role**, as underrepresented classes like **8 (recall 0.83, F1-score 0.87) and 9 (recall 0.88, F1-score 0.88)** have slightly lower performance compared to more frequent classes like **1 (recall 0.93, F1-score 0.94) and 2 (recall 0.96, F1-score 0.94)**. To address these weaknesses, **data augmentation techniques** like **rotation, scaling, and contrast adjustments** could help improve generalization. Using a **deeper or more sophisticated architecture**, would allow the model to learn more discriminative features, reducing errors in ambiguous cases. Furthermore, handling class imbalance through **oversampling, class-weighted loss functions** could ensure that the model performs equally well across all digits. Finally, **experimenting with advanced optimizers like Adam**, and **extending the training duration** with proper regularization could further enhance the model's performance. By addressing these issues, the model's **overall accuracy, robustness, and ability to distinguish between visually similar digits** can be significantly improved.

Part 2: Adversarial Attacks on our Model

in this part We attacked our model (from part 1) using the Fast Gradient Sign Method (FGSM), where we add a small, calculated noise to the original images based on the model's gradient. By scaling this perturbation with epsilon = 0.1, we generate adversarial examples that the model struggles to classify correctly.

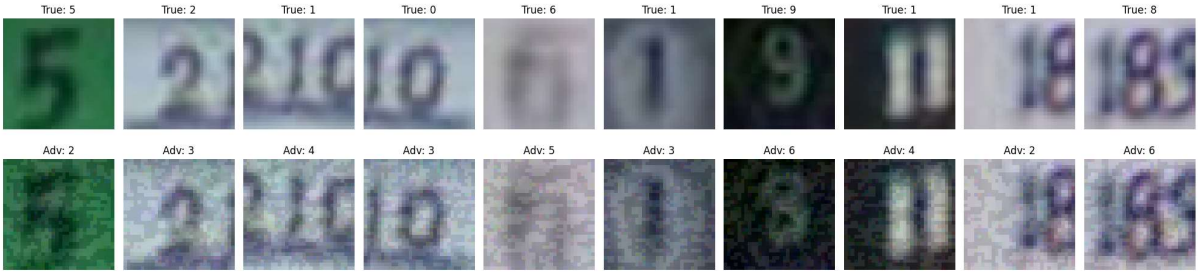
The output

```
Accuracy after FGSM attack with epsilon=0.1: 15.30%
```

This shows a drastic drop in our model's accuracy compared to its performance on clean images, confirming the model's vulnerability to adversarial perturbations. We see such a significant reduction because FGSM deliberately exploits our model's gradient information to generate small yet highly effective perturbations.

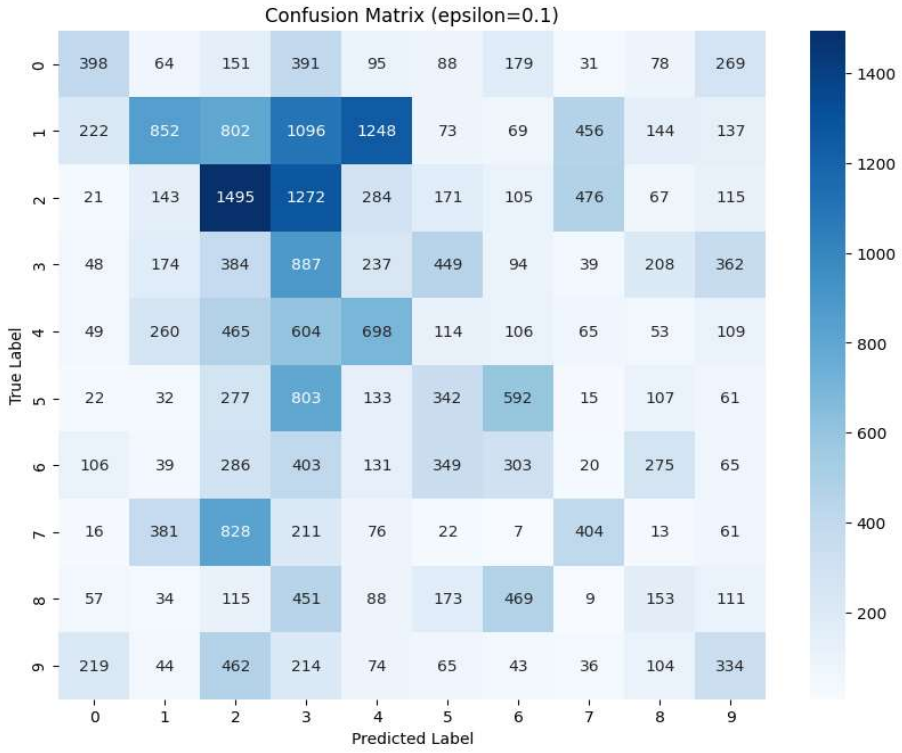
Visualization

1. Visualization of images that were identified correctly before the attack and misidentified afterward



The visualization demonstrates how the **Fast Gradient Sign Method (FGSM) attack** manipulates correctly classified images, leading to misclassification. In the **top row**, we see original images that the model **correctly classified before the attack**, with their respective true labels. The **bottom row** displays the adversarially perturbed versions, now labeled with their **incorrect predictions** after the attack. These adversarial examples were generated using **epsilon = 0.1**, introducing subtle pixel alterations that are often imperceptible to humans but significantly impact the model's decision-making. Some images appear **Blurring or noisy**, indicating the applied perturbation, which shifts the model's confidence in its classifications, causing incorrect outputs. This highlights the model's **susceptibility to adversarial attacks**, where even small, targeted modifications to pixel values can lead to **drastic misclassification**. The significant drop in accuracy after the attack emphasizes the need for **adversarial robustness strategies**, such as **adversarial training, defensive distillation, or gradient masking**, to make the model more resistant to these subtle but effective perturbations.

2.Creating a confusion matrix after the attack



The confusion matrix after the **FGSM attack (epsilon=0.1)** clearly demonstrates the significant drop in classification performance. The **off-diagonal values have increased**, meaning that the model is now frequently misclassifying digits, while the **diagonal values (correct**

classifications) have decreased compared to the clean dataset. For example, before the attack, digit **1 was classified correctly 4760 times**, but after the attack, its correct classifications **dropped to 852**, with **1096 misclassified as 3, 1248 as 4, and 802 as 2**, showing how adversarial perturbations disrupt the model's ability to differentiate between similar digits. Similarly, digit **2, which was correctly classified 3991 times in the clean dataset, now only has 1495 correct classifications**, while **1272 instances are misclassified as 3** and **476 as 7**, indicating that the perturbations make it resemble these other digits more closely.

Another noticeable degradation occurs with digit **8**, where its correct classifications fell to **153**, while **469 were misclassified as 6, 451 as 3, and 173 as 5**. This suggests that the attack significantly altered the pixel distribution of digit **8**, causing it to appear more like other curved digits such as **6 and 3**. Similarly, digit **9, which was originally recognized correctly 1410 times, now only has 334 correct classifications**, with **462 instances confused with 2 and 219 with 0**, reinforcing the impact of perturbations on model predictions.

This confusion pattern highlights **which digits are most susceptible to adversarial attacks**, particularly **those with similar shapes, such as 1 and 4, or 7 and 2**. The attack forces the model to make incorrect predictions even for images it previously classified correctly. To **counteract this vulnerability**, techniques like **adversarial training**, where the model is trained on adversarial examples, could help it learn to recognize perturbations and make more robust decisions.

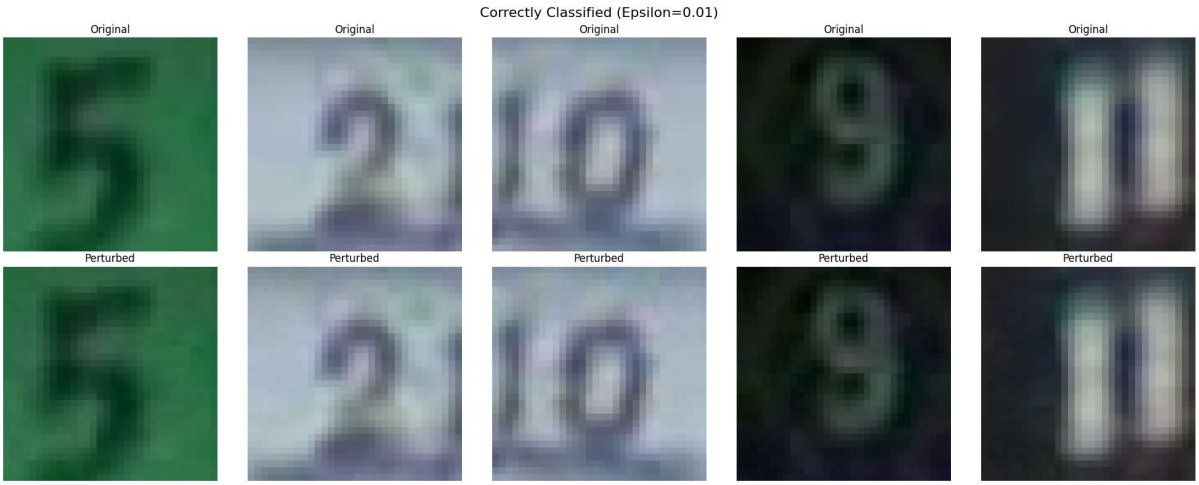
3.Accuracy as a function of epsilon

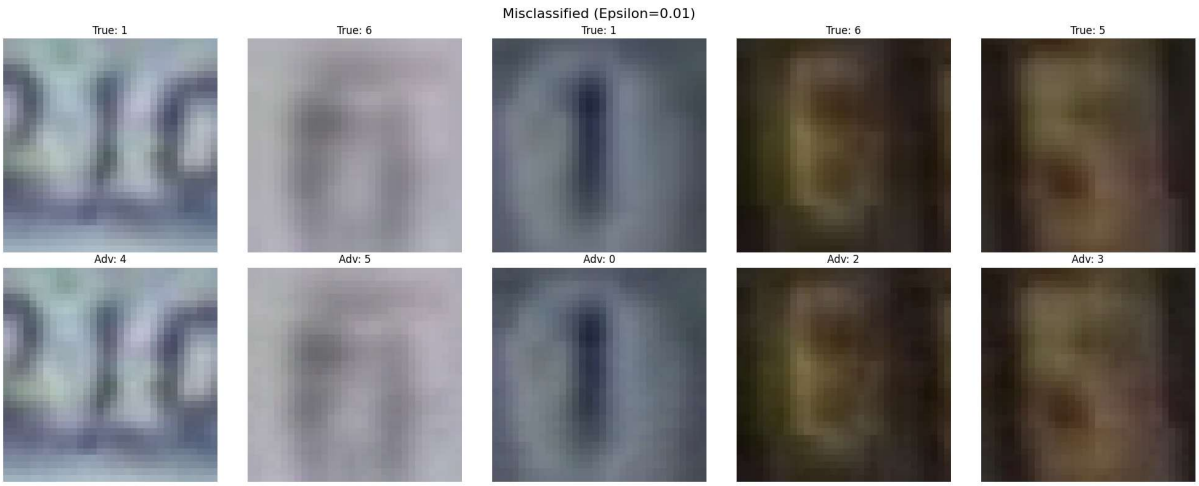
BUT FIRST:

Let's Start by visualize both correctly classified and misclassified perturbed images for each epsilon

For epsilon = 0.01.

Accuracy after FGSM attack with epsilon=0.01: 77.43%

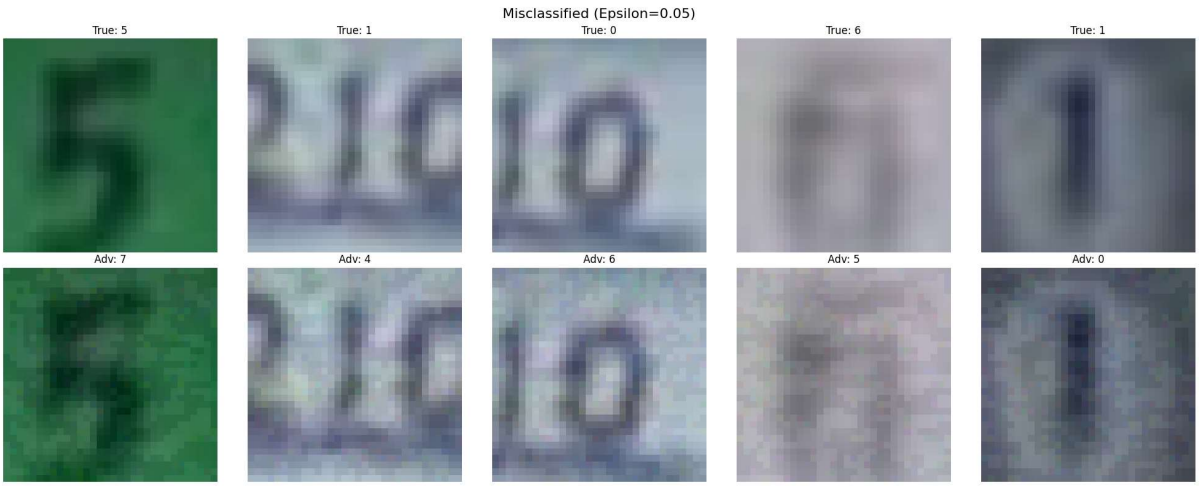




The **FGSM attack at epsilon = 0.01** introduces subtle perturbations that are **nearly imperceptible to the human eye** but significantly impact the model's performance, reducing accuracy to **77.34%**. While most images remain correctly classified, others are misclassified, revealing the model's sensitivity to even minor pixel shifts. However, these slight changes are enough to push inputs across the model's decision boundaries, leading to incorrect predictions (e.g., **1 misclassified as 4 or 0, 6 as 5 or 2, and 5 as 3**). This demonstrates how neural networks rely on precise pixel patterns, making them vulnerable to adversarial noise.

epsilon = 0.05.

Accuracy after FGSM attack with epsilon=0.05: 35.07%



At **epsilon = 0.05**, the added noise becomes more noticeable, and misclassifications increase significantly, bringing accuracy down to **35.05%**. Some distortions are still subtle, but they increasingly interfere with the model's feature recognition, leading to errors such as a **0 being misclassified as 6** and a **previously correct digit as 0 now misclassified as 6 (third picture)**. While the model struggles, a **human eye can still recognize most digits with some effort**, though the perturbations are beginning to affect readability.

Accuracy after FGSM attack with epsilon=0.1: 15.30%

At **epsilon = 0.1**, the adversarial perturbations become much stronger, creating **visible distortions** that significantly mislead the model, reducing accuracy to **15.30%**. While **a human can still recognize most digits with careful observatio**n, the added noise is enough to confuse the model, as seen in cases like **2 misclassified as 3 and 0 misclassified as 3**. At this level, adversarial noise **overwhelms the model's learned features**, making it highly susceptible to incorrect predictions.

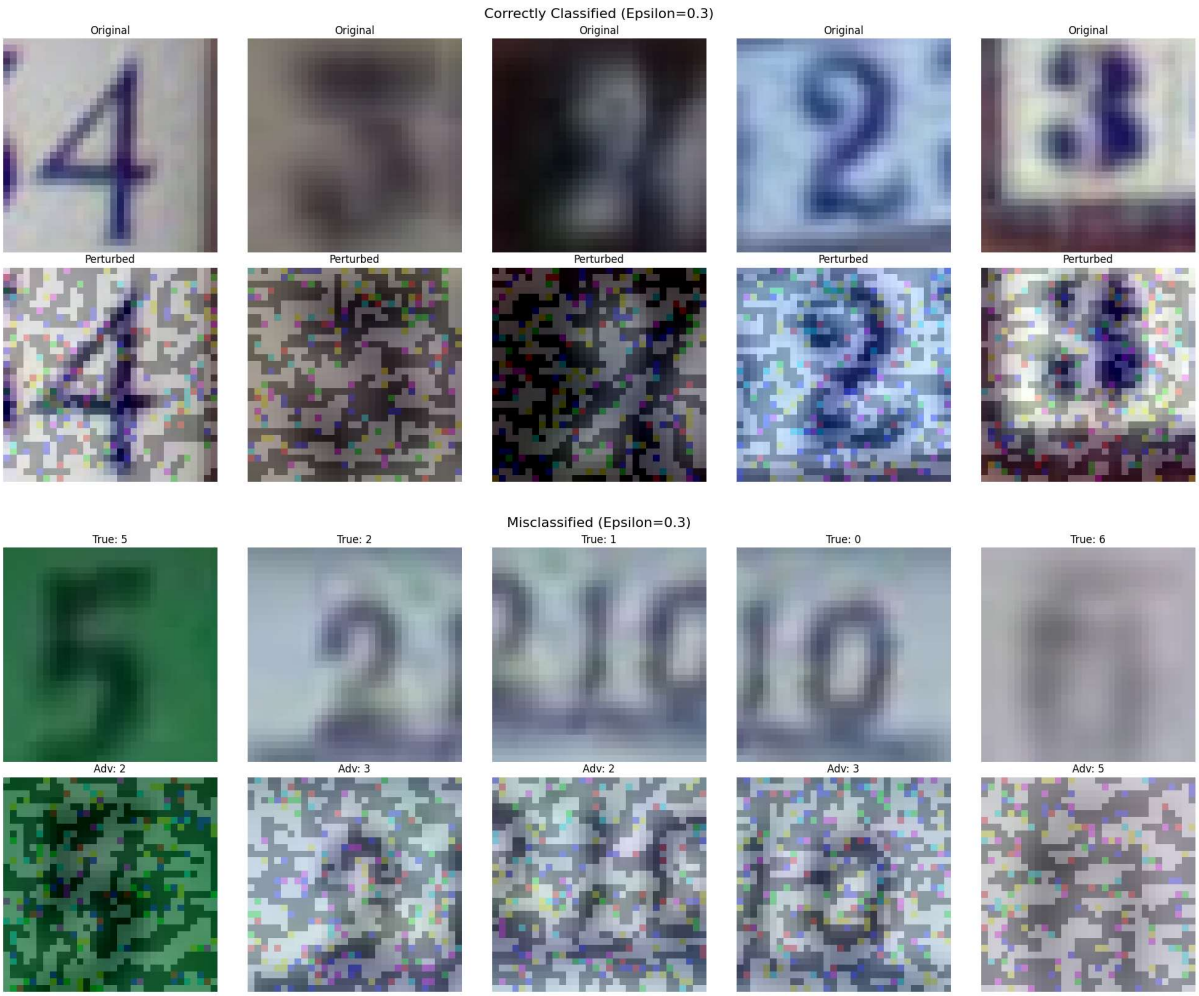
For epsilon = 0.2.

Accuracy after FGSM attack with epsilon=0.2: 6.01%

At **epsilon = 0.2**, the adversarial perturbations become highly visible, introducing **strong distortions** that significantly degrade both model performance and human readability. The accuracy drops drastically to **6.01%**, as the added noise overwhelms the model's ability to recognize digits. Many perturbed images appear as **heavily pixelated or grainy versions** of their originals, making classification increasingly difficult. While some digits can still be recognized with effort, the distortions are now severe enough to **mislead (in most cases) a human observer**.

For epsilon = 0.3.

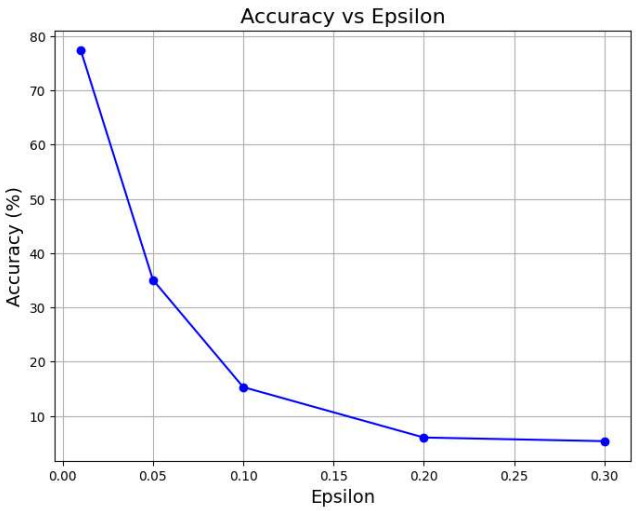
Accuracy after FGSM attack with epsilon=0.3: 5.34%



At **epsilon = 0.3**, we are surprised to see that some images, including certain **2s and 4**, are still correctly classified despite the extreme distortion. The perturbations are so severe that many digits appear heavily **pixelated, noisy, or broken apart**, making it **difficult even for a human to recognize them**. Yet, somehow, the model still manages to correctly classify a few of these heavily altered images. This raises the question of **why certain perturbed digits remain recognizable to the model while others are completely misclassified**. One possibility is that the model has learned **some robust feature representations** that remain detectable even under strong perturbations, while other digits rely on more fragile features that are easily disrupted. It would be interesting to investigate whether specific digits or patterns are inherently more resilient to adversarial noise, and whether this is due to **model biases, dataset properties, or deeper structural patterns in the network's learned features**.

3.Accuracy as a function of epsilon

With increasing epsilon values, we see a **clear downward trend** in the model's accuracy—from **77.43% at epsilon = 0.01** down to just **5.34% at epsilon = 0.3**. The **"Accuracy vs. Epsilon"** plot highlights this steep decline, confirming that **larger perturbations significantly degrade performance**. Looking at the perturbed images for **epsilon = 0.01 and epsilon = 0.05**, the changes are **barely noticeable**, and a human observer can still easily recognize the digits. However, by **epsilon = 0.1**, the added noise **starts interfering more aggressively** with digit shapes, causing the model's accuracy to **drop to 15.30%**, and at this point, humans may need to **focus more carefully** to classify some digits correctly. At **epsilon = 0.2 and especially at epsilon = 0.3**, the perturbations **become so strong** that many images **look heavily distorted and grainy**, making it increasingly difficult even for a human to distinguish the correct digits. This confirms that **while the model degrades quickly with adversarial noise, human vision is much more resilient**, though at higher epsilon values, even humans may struggle. Ultimately, this plot exposes the **model's vulnerability to adversarial attacks**, reinforcing the need for **defensive strategies** such as **adversarial training** to maintain robustness as epsilon increases.

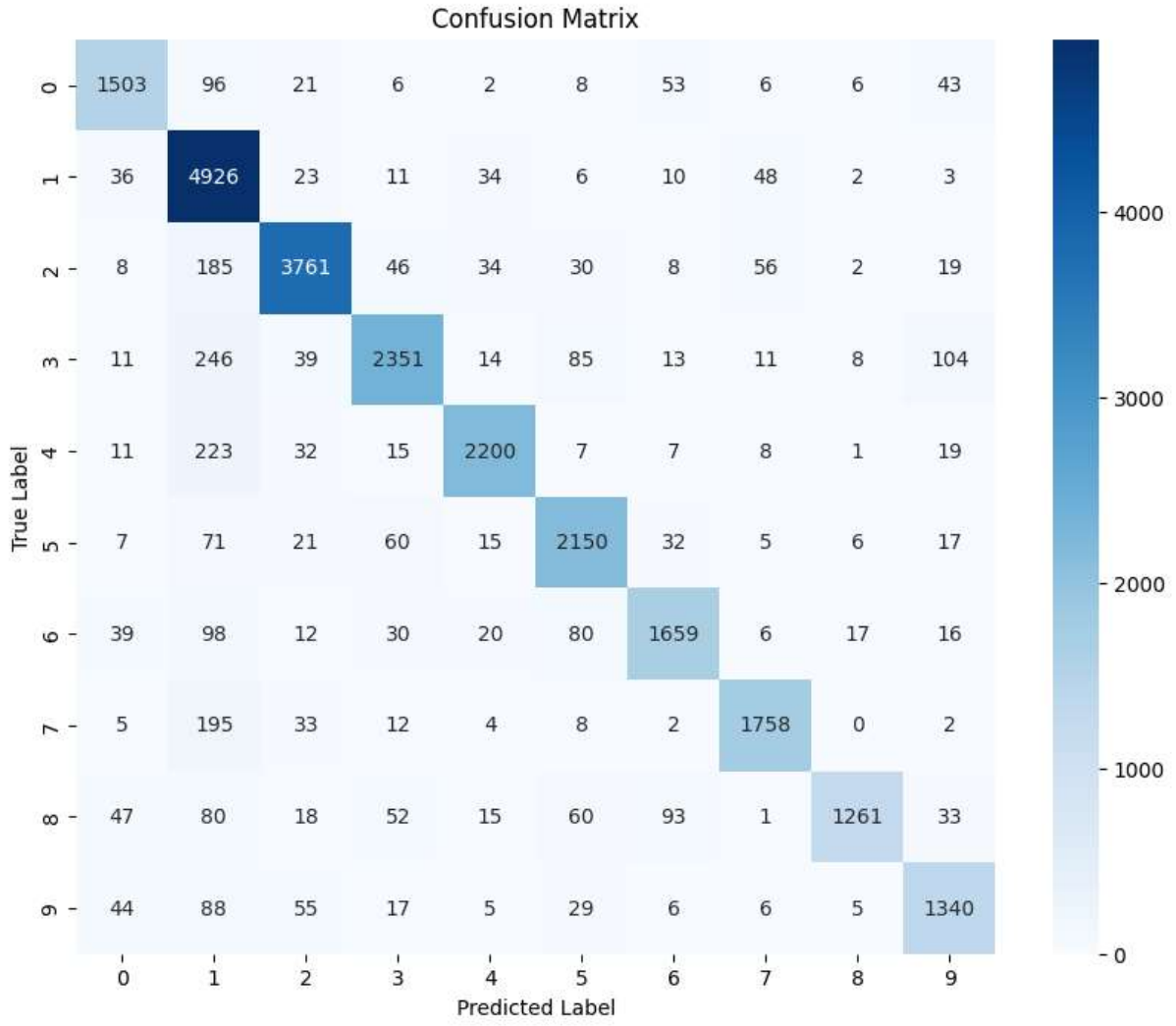


Part 3: Training our model using adversarial training

In this part, we perform adversarial training by including both the original images and FGSM-perturbed images (with epsilon=0.1) in each training batch. We generate adversarial examples on the fly, after computing gradients on the original loss and then combine the original loss with the adversarial loss using a weighted approach controlled by alpha=0.8. This setup ensures the network sees challenging adversarial variants at every iteration, gradually making it more resistant to attacks. We train for up to 50 epochs and rely on early stopping with a patience of 5 epocs to prevent overfitting and rely on **CosineAnnealingLR** to schedule a smooth reduction of the learning rate. We also apply **gradient clipping** (max norm=1.0) to avoid large updates that can destabilize training. Throughout the process, we monitor validation accuracy each epoch, resetting a patience counter only when the model's validation accuracy improves. Ultimately, this adversarial training procedure raises the model's accuracy on perturbed data to about **76.34%**, a significant improvement compared to the model's performance on adversarial examples without adversarial training.

```
Accuracy after FGSM attack with epsilon=0.1: 76.34%
Accuracy on perturbed data after adversarial training (epsilon=0.1): 76.34%
```

Visualization



The visualizations provide insights into the model's performance and areas for improvement. The **confusion matrix** illustrates the classification accuracy by displaying correct and incorrect predictions for each digit, where darker shades represent higher counts. The model performs well on certain digits, such as "1" with **4926** correct classifications, but struggles with others, particularly **3 misclassified as 9 and 9 misclassified as 5**, which indicates issues with visually similar digits. Off-diagonal elements highlight these misclassifications, emphasizing that digits with similar shapes tend to be confused. The second visualization presents **ten misclassified images**, displaying the true label and the model's incorrect prediction. Many of these errors arise from **blurry, distorted, or poorly illuminated images**, making classification difficult. Some digits, like **9 mistaken for 5 or 3 mistaken for 9**, resemble each other in shape, leading to

frequent confusion. While adversarial training with **FGSM perturbations (ε = 0.1)** improved robustness, raising accuracy on adversarial samples to **76.34%**, it did not eliminate all misclassifications, especially those caused by natural variations in digit appearance. The model still struggles with inherent distortions and ambiguous cases, suggesting that additional improvements, such as **more data augmentation** (e.g., rotations, contrast adjustments) or **higher epsilon values for stronger adversarial training**, could enhance its robustness further. Despite these errors, adversarial training successfully increased resilience to small perturbations, but further refinements in architecture and preprocessing techniques are necessary to minimize natural misclassifications.

Part 4: Contrastive Learning

Dry Questions

1. When training an unsupervised contrastive learning model such as SimCLR, would we prefer to have a large or small batch size?

When training an unsupervised contrastive learning model such as SimCLR, we prefer to use a large batch size. This is because the NT-Xent Loss relies on both positive and negative pairs, and a larger batch size provides a more diverse set of negative pairs (unrelated examples) for each positive pair. This diversity helps the model learn more meaningful embeddings by effectively distinguishing between similar and dissimilar samples, ultimately improving generalization. Empirical evidence from the SimCLR paper shows that larger batch sizes significantly enhance the quality of the learned embeddings. However, large batch sizes require substantial computational resources, so techniques like memory banks or multi-GPU training are often used as practical solutions to overcome these limitations.

2. In general, what possible evaluation metrics could be used in this task (unsupervised representation learning) to measure our model's performance?

In the context of unsupervised representation learning for our task, the most relevant evaluation methods are:

Linear Evaluation Protocol: A linear classifier is trained on top of the embeddings using a labeled subset of the data. The classifier's performance, measured through metrics like accuracy or F1-score, reflects the quality of the representations.

k-Nearest Neighbors (k-NN) Accuracy: This evaluates the embeddings by classifying samples based on their nearest neighbors in the embedding space, providing an intuitive measure of how well similar samples are grouped.

Clustering Metrics: Metrics like Normalized Mutual Information (NMI) or Adjusted Rand Index (ARI) assess how well the learned embeddings capture the underlying structure of the data by comparing clustering results to ground truth labels.

Visualization: Techniques such as t-SNE or UMAP allow for a visual inspection of the embedding space, helping determine if similar samples form distinct clusters.

These methods are particularly suited for evaluating unsupervised representation learning models, as they emphasize the quality and structure of the learned embeddings.

3. When creating embeddings for images in the test set, how does the process differ from what we do in training?

When creating embeddings for images in the test set, the process differs from training in the following key ways:

No Data Augmentation:

During training, data augmentations (e.g., cropping, flipping, color jittering) are applied to create multiple augmented views of the same image to help the model learn robust representations.

In the test phase, no augmentations are applied to ensure the embeddings represent the original image.

No Contrastive Loss:

In training, the embeddings are used to compute the contrastive loss (e.g., NT-Xent Loss) by comparing positive and negative pairs.

In the test phase, we directly extract the embeddings without computing any loss.

Frozen Model Parameters:

During training, the model parameters are updated using backpropagation.

In the test phase, the model operates in evaluation mode (model.eval()), and the parameters remain frozen, ensuring that batch normalization statistics and dropout are handled appropriately.

Single View per Image:

In training, multiple augmented views of each image are used to compute positive pairs.

In the test phase, each image is passed through the encoder only once to obtain its embedding.

Focus on Downstream Tasks:

In training, the goal is to learn embeddings that separate similar and dissimilar pairs.

In the test phase, the embeddings are typically used for downstream tasks such as classification, clustering, or retrieval.

4. For each of the following image augmentations, explain whether or not we would like to use them in the SimCLR framework:

- Randomly cropping a fixed-size window in the image.

Use in SimCLR? Yes.

Reason: Random cropping creates spatial variations of the image, which forces the model to focus on shared semantic features rather than precise locations of objects. This is one of the most effective augmentations used in SimCLR for generating positive pairs.

- Enlarging the image to 128x128.

Use in SimCLR? No.

Reason: Enlarging the image introduces scale inconsistencies and interpolation artifacts, which can negatively affect the training process. Maintaining a consistent input size is critical for ensuring meaningful embeddings.

- Random rotation of the image.

Use in SimCLR? Yes (with limitations).

Reason: Small random rotations help the model learn rotational invariance, which can improve robustness. However, large rotations may distort the semantic meaning of certain images (e.g., a rotated house number might no longer resemble its original class).

- Adding Gaussian noise.

Use in SimCLR? Yes.

Reason: Adding Gaussian noise introduces subtle variations in pixel values, which helps the model learn to focus on higher-level semantic features instead of noise-sensitive low-level features. However, the magnitude of noise should be carefully controlled to avoid overly distorting the image.

- Randomly changing the image's dimensions.

Use in SimCLR? No.

Reason: Randomly altering dimensions (e.g., stretching or shrinking non-proportionally) distorts the aspect ratio of the image, which can introduce unnatural artifacts and reduce the model's ability to learn meaningful representations.

- Randomly converting the image to grayscale.

Use in SimCLR? Yes.

Reason: Grayscale conversion removes color information, encouraging the model to focus on texture, structure, and other semantic features. This is particularly useful in datasets where color is not a critical feature.

Model and Training Overview

We trained a Convolutional Neural Network (CNN) to generate image embeddings by modifying a pretrained ResNet-50. Specifically, we removed the original classification layer and replaced it with a 128-dimensional output layer to produce compact vector representations of input

images. We then employed an **unsupervised contrastive learning** approach (NT-Xent loss) to train the model.

Under this paradigm, for each image we apply two different data augmentations, and the model learns to **pull embeddings of these augmented views (positive pairs) closer together** while pushing apart embeddings from different images (negative pairs). The temperature hyperparameter in the loss helps moderate how sharply positive examples are separated from negatives in the embedding space.

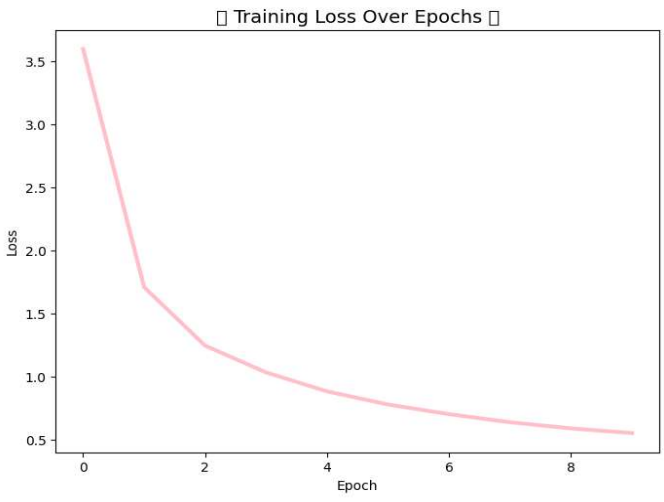
```
🎯 Epoch [1/10], Loss: 3.5943
🎯 Epoch [2/10], Loss: 1.7104
🎯 Epoch [3/10], Loss: 1.2473
🎯 Epoch [4/10], Loss: 1.0354
🎯 Epoch [5/10], Loss: 0.8851
🎯 Epoch [6/10], Loss: 0.7814
🎯 Epoch [7/10], Loss: 0.7050
🎯 Epoch [8/10], Loss: 0.6410
🎯 Epoch [9/10], Loss: 0.5922
🎯 Epoch [10/10], Loss: 0.5552
```

Training Loss Curve:

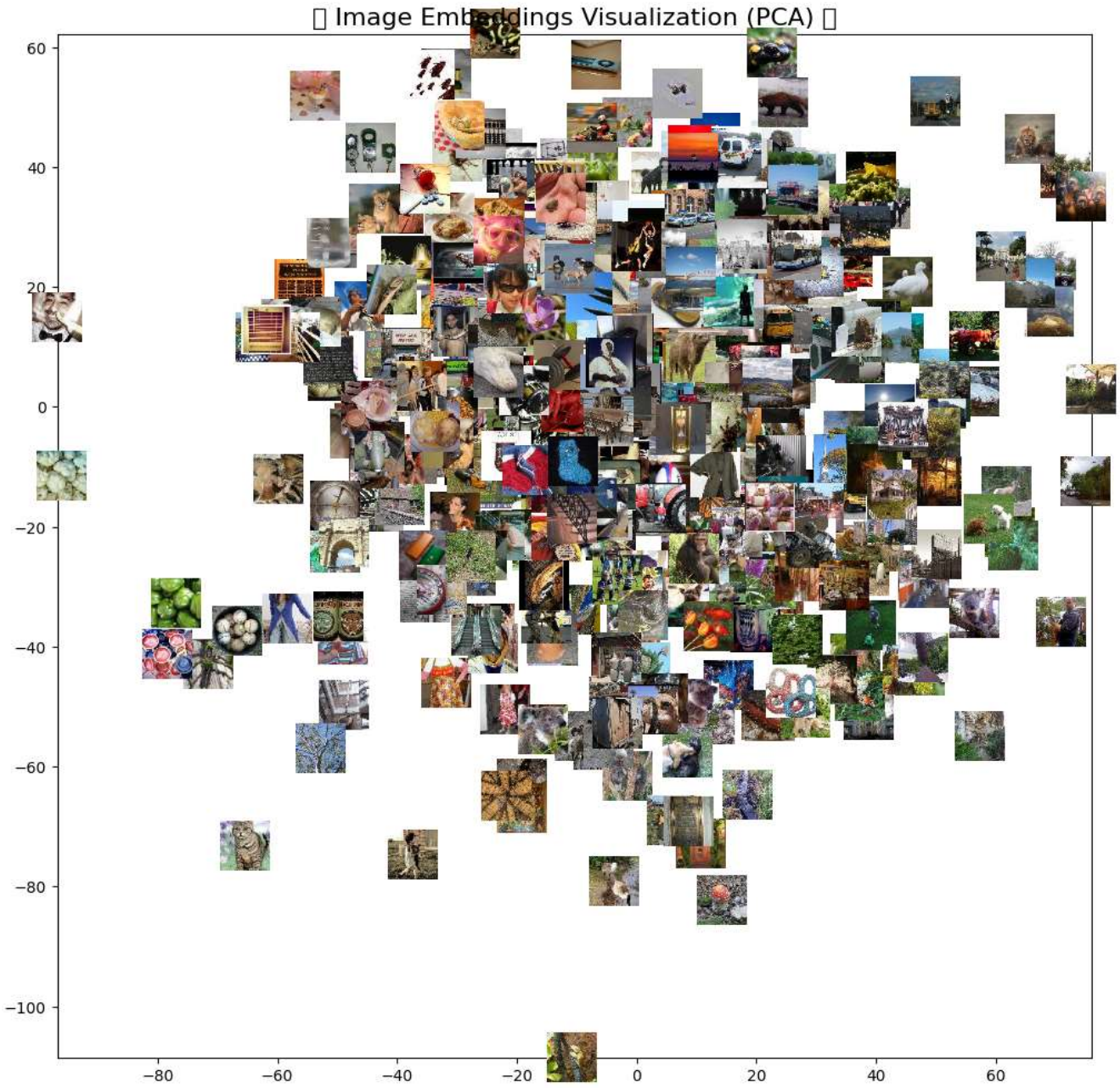
The **training loss** started at about **3.59** and made a significant drop to around **1.71** by the second epoch. By the third epoch, it was already near **1.25**—well under our target threshold of **3.0**. Over the course of ten epochs, the loss continued to decline, ultimately arriving at **0.55**.

This **steady downward trend** suggests that the network effectively learned to identify shared features among augmented versions of the same image. Early epochs typically see a swift reduction in contrastive loss, as the model grasps fundamental visual features that differentiate positive from negative pairs. After about three epochs, the drop becomes more gradual, indicating the model is refining its representation and focusing on subtler distinctions.

While the loss curve demonstrates successful training, it's worth noting that unsupervised contrastive learning can sometimes hinge on **low-level cues** (e.g., color, texture) rather than deeper semantic concepts. Further fine-tuning—either by extending training, carefully tuning hyperparameters (like the NT-Xent temperature), or modifying the augmentation strategy—could potentially yield even more meaningful representations.



🖼️ Visualizing embeddings on test data.

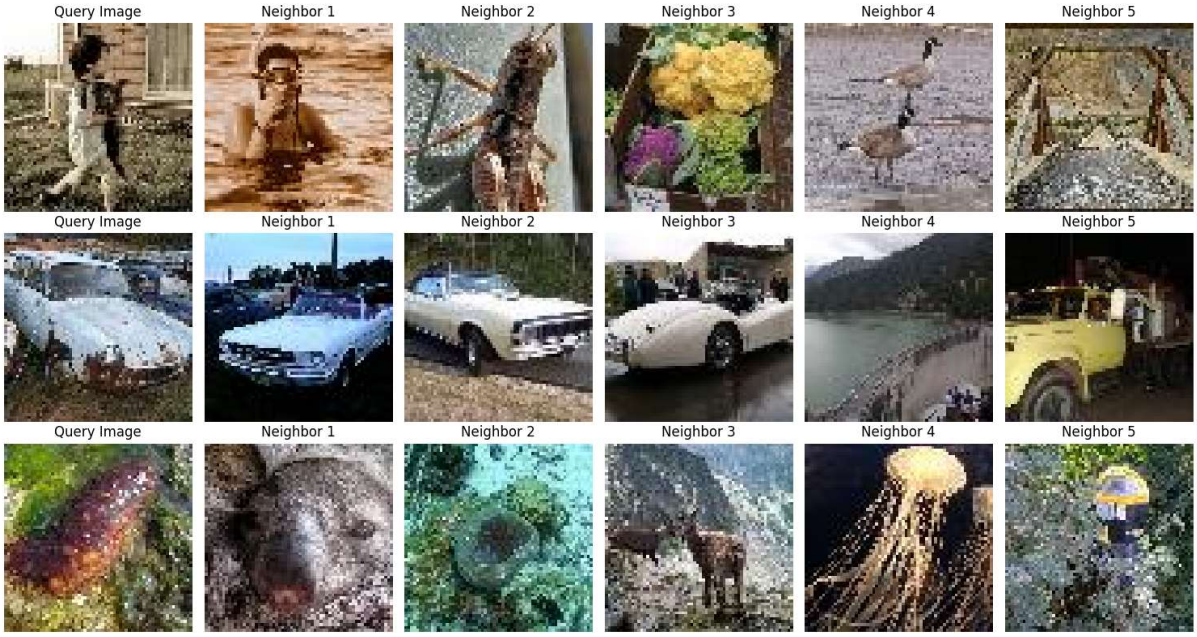


Embeddings Visualization:

Our **embedding space** shows promising results for some categories, such as **cars**, which do often cluster together, suggesting that the model has captured key visual features for those objects; however, for other classes, like **people** or **animals**, the nearest neighbors can be surprising—matching a child carrying a bucket with scenes that share **color** or **texture** rather than truly **semantic** content. This indicates that while the embeddings incorporate some **domain knowledge**, they are also over-reliant on background similarities and **aggressive**

augmentations, which can distort or mask essential cues. Because **purely unsupervised** contrastive learning only enforces closeness between augmented views of the same input, it does not necessarily drive the model to form robust **class-level** clusters for diverse categories. Consequently, fine-tuning elements such as **NT-Xent temperature**, **augmentation strategies**, or introducing **weak supervision** might improve the embedding quality, ensuring that the model consistently captures more **meaningful** visual similarities across all image types.

For some batch of the test loader, take 3 images in the batch. For each image, find and display the 5 images that have the closest embeddings to them. Do the chosen images make sense? If not, what could have possibly gone wrong with your model?



Nearest-Neighbor Retrieval:

We set out to examine how our **purely unsupervised** embeddings capture meaningful visual similarities by retrieving five nearest neighbors for each query image, and while we were **pleased** to see instances where **cars** are successfully matched with other **cars**, many of the results were still **surprising**—for example, a child carrying a bucket might be paired with insects or unrelated scenes purely due to shared **color** or **texture** rather than object identity. This behavior reflects the **instance-level** nature of our contrastive learning approach: it only enforces that **augmented views** of the *same image* appear close together in the embedding space, offering no direct incentive for grouping broader **semantic** categories. Consequently, the model can become **over-reliant** on background details (e.g., water, rust, foliage) and **aggressive augmentations**, which can distort or mask crucial features. Other factors, such as **imbalanced data**, **limited training time**, or an **untuned NT-Xent temperature**, may exacerbate these issues, leading to retrievals that favor superficial similarity rather than actual object identity. To address these challenges, one could **refine** data augmentations, **fine-tune**

hyperparameters, or introduce **weak supervision**, each of which can nudge the model toward more **robust** and **conceptually meaningful** representations across a wider range of image categories.