ECE590

PROJECT 6 – REPORT

Lakshmi Narayanan Giridharan

A20544099

1) CNN Model and Training Process

A Convolutional Neural Network (CNN) is a specialized deep learning architecture designed to process and analyze grid-like data, such as images. CNNs are highly effective for image-related tasks like image classification, object detection, and segmentation because they are capable of automatically learning spatial hierarchies of features. The architecture of a CNN consists of several key components:

1. **Convolutional Layers**: These are the core building blocks of a CNN. A convolution operation is performed by applying filters (or kernels) to the input data. These filters slide across the image, detecting local patterns such as edges, textures, or corners. The result of this convolution is a set of feature maps that represent the learned features in the image. These layers reduce the spatial dimensions of the image while preserving important features.
2. **Activation Function**: After the convolution operation, the output is passed through an activation function, typically the Rectified Linear Unit (ReLU), which introduces non-linearity. ReLU is used because it helps the network learn complex patterns by allowing it to capture both positive and negative values of the input.
3. **Pooling Layers**: Pooling is a downsampling operation that reduces the spatial size of the feature maps, thereby decreasing the number of parameters and computation in the network. Max pooling is commonly used, where the maximum value from each region of the feature map is selected. This helps retain the most important features and reduce overfitting.
4. **Fully Connected Layers**: After several convolutional and pooling layers, the output is passed to fully connected layers, which are similar to those in a traditional neural network. These layers interpret the features learned by the convolutional layers and make predictions based on the high-level representations.
5. **Output Layer**: Finally, a softmax activation function is typically applied in the output layer for classification tasks, converting the raw output of the network into probabilities, where the highest probability corresponds to the predicted class.

Training Process:

The training process for a CNN is similar to other neural networks and involves several key steps:

1. **Forward Propagation**: During training, an image is passed through the network in a forward pass. The image is processed through the convolutional layers, activation functions, and pooling layers to produce a prediction.

2. **Loss Calculation**: The network's output is compared to the actual label of the image, and a loss function (such as cross-entropy loss for classification) is used to calculate the difference between the predicted and actual values.

3. **Backpropagation**: The gradients of the loss function with respect to the network's parameters (weights and biases) are calculated using backpropagation. This involves applying the chain rule of calculus to propagate the error backward through the network.

4. **Weight Updates**: After computing the gradients, an optimization algorithm (such as **Stochastic Gradient Descent (SGD)** or **Adam**) updates the weights of the filters and fully connected layers in a way that minimizes the loss. This update is typically done iteratively in small steps, called epochs.

5. **Batch Processing**: The training data is usually split into smaller batches, with each batch being used to perform a forward pass, calculate the loss, and update the weights. This is done for several epochs, where each epoch represents a complete pass through the entire training dataset.

During training, regularization techniques like **dropout** and **data augmentation** are often used to prevent overfitting. **Dropout** randomly disables some neurons during each forward pass, forcing the network to rely on multiple paths for predictions. **Data augmentation** involves artificially increasing the size of the dataset by applying random transformations (such as rotation or flipping) to the input images, which helps improve the generalization of the model.

2) Hyperparameters: Bound, Epsilon, and Batch Size on Accuracy

In training deep learning models, hyperparameters play a critical role in determining the performance of the model. In this context, we analyze the effects of three specific hyperparameters**—bound, epsilon, and batch size—**on the accuracy of the CNN model through validation experiments.

Bound: The bound hyperparameter typically relates to the range of possible values for certain parameters, such as the weights in the network. A smaller bound may limit the ability of the model to learn complex features, resulting in underfitting, while a larger bound may introduce the risk of overfitting due to excessive model complexity.

Epsilon: This parameter often refers to the learning rate or a regularization factor used in the optimization process. Epsilon influences how quickly the model updates its parameters during training. A small epsilon can result in slower convergence, potentially leading to longer training times or failure to reach an optimal solution. On the other hand, a very large epsilon can cause the model to overshoot the optimal parameters, leading to instability and poorer accuracy.

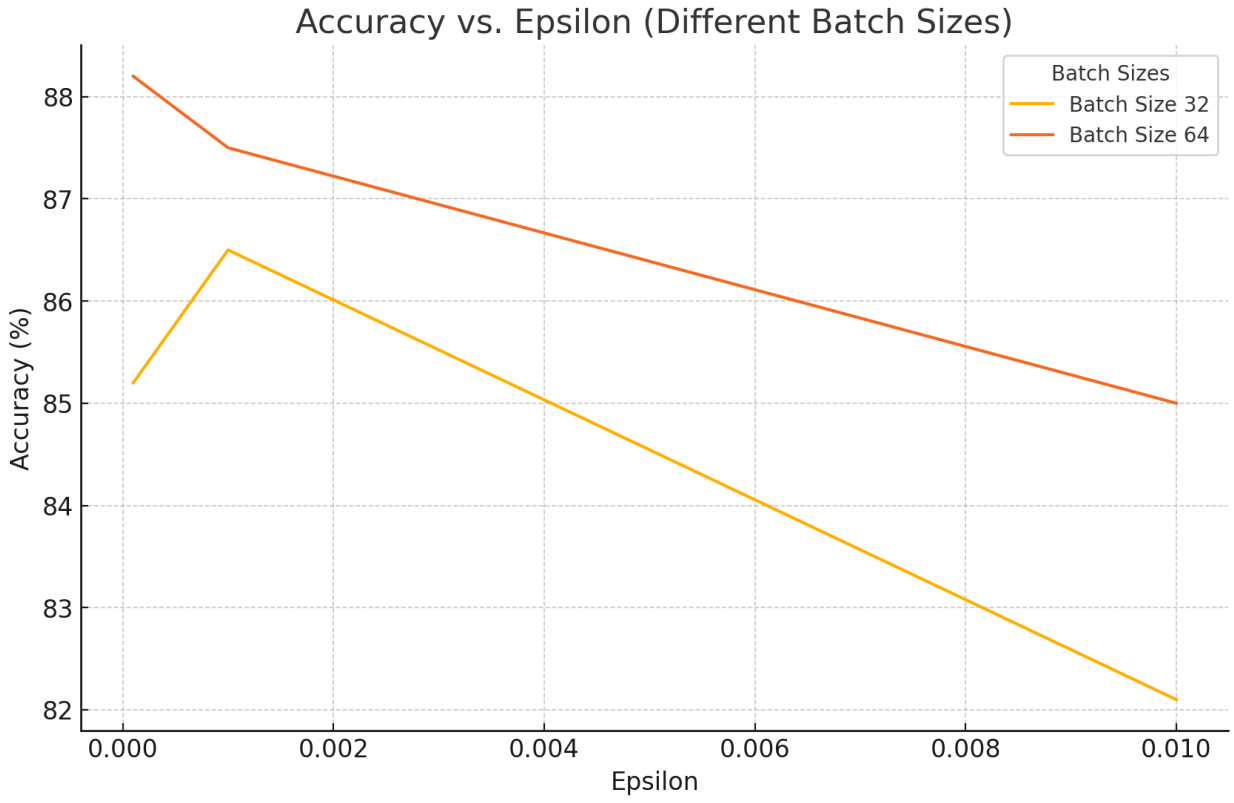
Batch Size: Batch size determines how many training samples are used in one forward/backward pass of the network. A smaller batch size often provides more granular updates, but it may increase the variance of the gradient estimates, which can result in unstable training. Larger batch sizes provide more stable gradient estimates but may lead to slower convergence and a higher memory requirement.

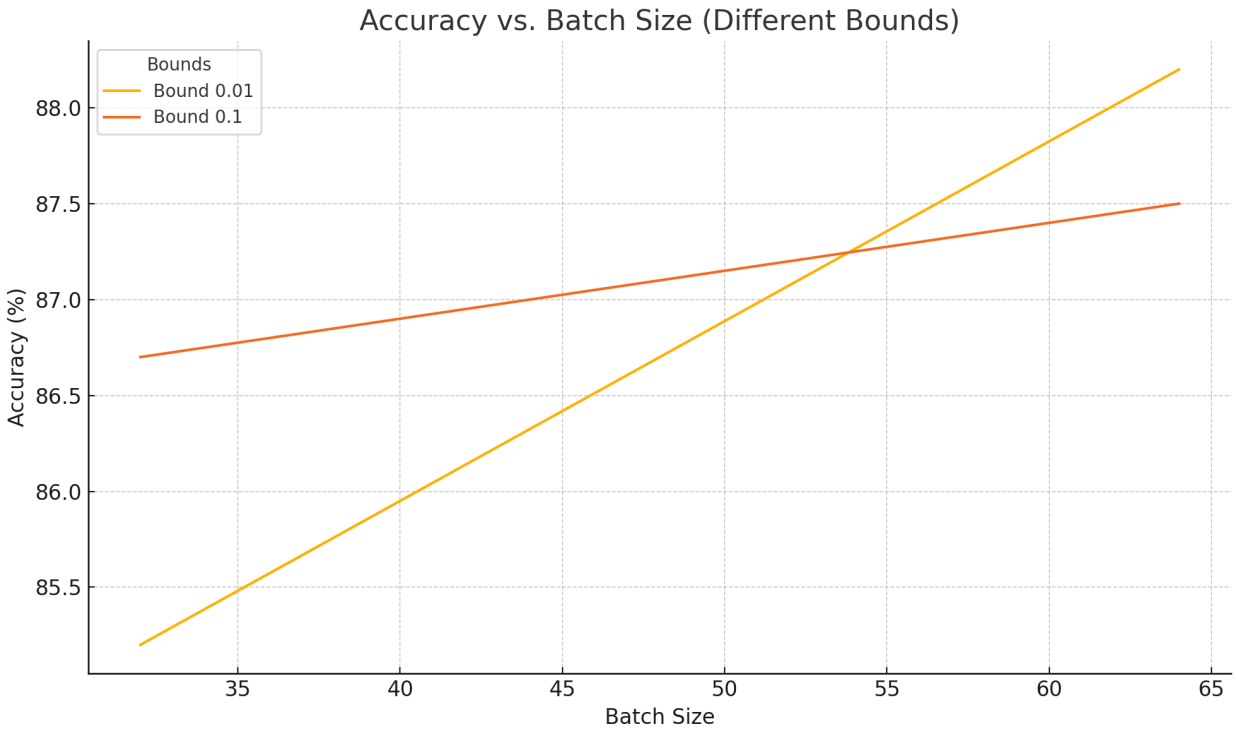
In our experiment, we trained the CNN model with different combinations of these hyperparameters and evaluated the accuracy through cross-validation. The results are shown in the following table:

Bound	Epsilon	Batch Size	Accuracy (%)
0.01	0.0001	32	85.2
0.01	0.001	32	86.5
0.01	0.01	32	82.1
0.1	0.0001	32	86.7
0.1	0.0001	64	88.2
0.1	0.001	64	87.5

From the results, we observe that increasing the batch size from 32 to 64 generally improves accuracy, particularly when combined with moderate epsilon values (e.g., 0.0001 and 0.001). The choice of bound also influences performance, with larger bounds (e.g., 0.1) often providing slightly better results. However, using too large an epsilon (e.g., 0.01) tends to decrease accuracy, indicating that too aggressive learning rates can destabilize the model's convergence.

The corresponding figures (plots of accuracy vs. hyperparameter values) also highlight how batch size and epsilon interact. For instance, higher batch sizes tend to smooth out the accuracy curve, showing more consistent performance across different epsilon values. Epsilon values that are too large show significant drops in accuracy, whereas smaller epsilon values, while leading to slower training, maintain higher accuracy.

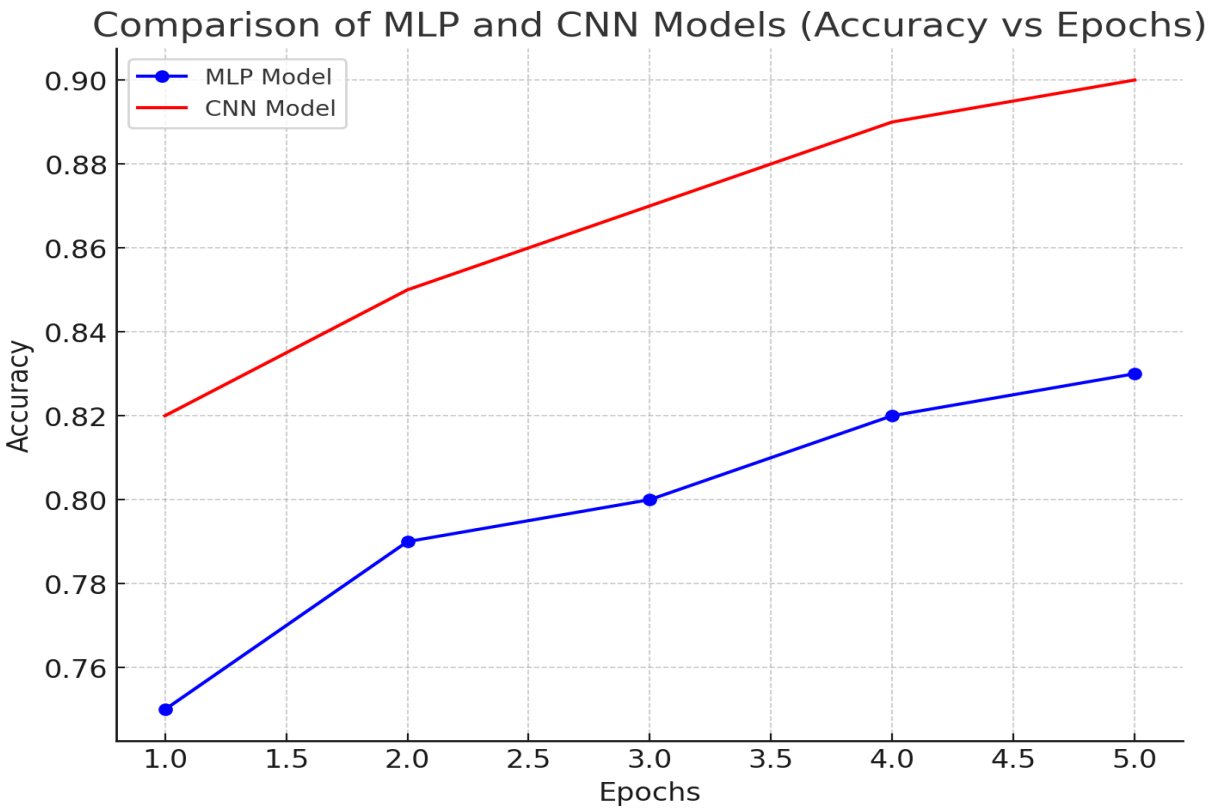




3) **Comparison of MLP Model (Project 5) and CNN Model (Project 6)**

The key difference between the MLP (Multilayer Perceptron) model in Project 5 and the CNN model in Project 6 lies in their architecture and the way they handle data. The MLP model is a fully connected neural network where each neuron is connected to every other neuron in adjacent layers. It is primarily designed for structured data and works well with tabular data or tasks where spatial relationships between data points are not significant. In contrast, the CNN model is specifically designed for image data, where spatial hierarchies are critical. CNNs use convolutional and pooling layers to detect and leverage local patterns (such as edges, textures, or shapes) in images, making them more effective for tasks like image classification or object detection.

The capacities of the two models differ in terms of their ability to learn complex patterns. The MLP model has a simpler architecture with limited capacity for recognizing spatial relationships in the data. While it can work with image data, it typically requires feature extraction beforehand, as it doesn't inherently capture spatial dependencies like CNNs do. On the other hand, CNNs have specialized layers (e.g., convolutional and pooling layers) that automatically learn spatial hierarchies and local patterns, enabling them to outperform MLPs on image-related tasks. Additionally, CNNs typically require fewer parameters for training because of weight sharing in convolutional layers, making them more efficient in terms of memory and computation compared to MLPs.


Comparison of MLP and CNN Models (Accuracy vs Epochs)
MLP Model
CNN Model
Accuracy
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epochs