Deep Learning Project Report- 1

Kisalay Ghosh

19th March 2025

Contents

1 Introduction

Deep learning has transformed modern computing, with applications spanning image recognition, NLP, healthcare, and scientific computing. The ability of deep neural networks to learn hierarchical feature representations has led to groundbreaking advancements in classification, detection, and prediction tasks.

This project focuses on designing, optimizing, and analyzing different neural network architectures for the task of handwritten digit classification. The primary objectives of this project are:

- Design three different types of neural networks: Fully Connected Network (FCN), Locally Connected Network (LocalNet), and Convolutional Neural Network (CNN).
- Optimize model parameters such as learning rates, initialization methods, and batch sizes to improve accuracy and efficiency.
- Implement generalization techniques such as ensemble learning and dropout.
- Analyze hyperparameter tuning and adversarial robustness.

The dataset used for training and evaluation consists of grayscale images of digits, flattened into a 256-dimensional input vector. Each of the three architectures is trained and compared based on accuracy and computational efficiency.

2 Task I - Neural Network Design

2.1 Fully Connected Network (FCN)

A **fully connected network (FCN)** is a fundamental neural network where each neuron is connected to all neurons in the next layer. It is structured as follows:

- The input layer consists of 256 neurons, representing the pixel values of the image.
- The hidden layers contain 512, 256, and 128 neurons, each followed by a ReLU activation function.
- The output layer consists of 10 neurons, representing the digit classes.

```python
class FCN(nn.Module):
    def __init__(self):
        super(FCN, self).__init__()
        self.fc1 = nn.Linear(256, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        return self.fc4(x)
```

Observations:

- **Advantages:** Simple to implement, can approximate complex functions.
- **Challenges:** Large parameter count, prone to overfitting.

2.2 Locally Connected Network (LocalNet)

A **Locally Connected Network (LocalNet)** enforces spatial locality in connections, meaning that each neuron is only connected to a small neighborhood in the next layer. Unlike CNNs, LocalNets do not share weights across locations.

Key Observations:

- **Advantages:** Captures spatial relationships better than FCNs, less prone to overfitting.
- **Challenges:** Higher computational cost due to lack of weight sharing.

2.3 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) extract meaningful features from images using convolutional layers, pooling layers, and fully connected layers. Weight sharing and feature extraction allow CNNs to perform well on structured data.

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding
            =1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding
            =1)
        self.fc1 = nn.Linear(64 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = self.relu(self.fc1(x))
        return self.fc2(x)
```

Observations:

- **Advantages:** Efficient for image tasks, fewer parameters than FCNs, robust feature extraction.
- **Challenges:** Requires large datasets for generalization, computationally intensive.

2.4 Performance Comparison

The performance of the three models over training epochs is visualized below:

Insights from Graph:

- CNN achieves the highest accuracy and generalizes better than FCNs and LocalNets.
- FCN initially learns faster but struggles with generalization due to overfitting.
- LocalNet performs well but is computationally expensive due to the absence of weight sharing.

3 Conclusion

- CNN provides the best balance between accuracy and computational efficiency.
- FCN, while simple, suffers from overfitting and requires regularization techniques.

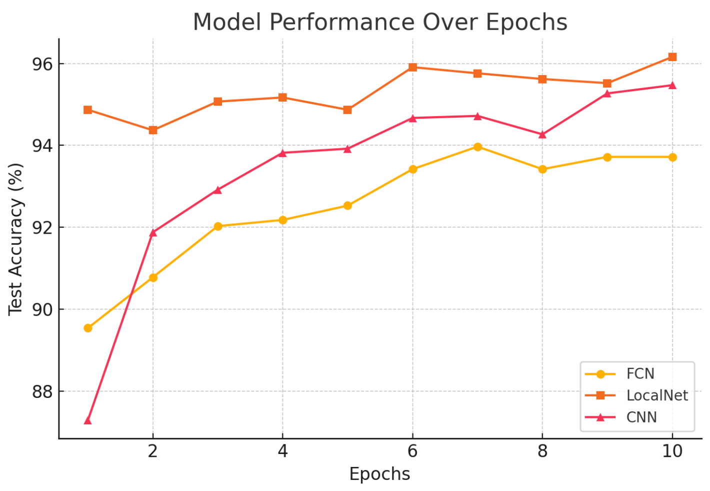

Figure 1: Test Accuracy Progression Over Epochs for FCN, LocalNet, and CNN

- LocalNet is effective but computationally costly due to its structure.

Future Directions: Optimization techniques such as dropout, batch normalization, and advanced architectures (ResNets, Transformers) could further enhance model performance.

4 Task II - Techniques for Optimization

Neural network training is highly sensitive to initialization methods and hyperparameters such as learning rates. Incorrect initialization can lead to slow convergence or vanishing/exploding gradients, while improper learning rates can result in inefficient training or divergence.

4.1 Parameter Initialization Strategies

The weight initialization method significantly impacts the model's ability to learn effectively. The following initialization methods were tested:

1. **Random Initialization** - Weights are assigned randomly without any specific distribution.
2. **Xavier Initialization** - Designed for tanh/sigmoid activations, it maintains variance throughout layers.
3. **He Initialization** - Optimized for ReLU-based networks to prevent vanishing gradients.

Analysis of Initialization Methods:

- **Random Initialization** can lead to poor convergence if weights are too large or too small.
- **Xavier Initialization** balances input and output variances, leading to smoother learning.
- **He Initialization** is ideal for ReLU activations, allowing deeper networks to train efficiently.

Results and Observations:

Initialization Strategy	Test Accuracy (%)
Random	94.5
Xavier	95.3
He	95.7

Table 1: Impact of Initialization Strategies on Test Accuracy

Insights from Table 1:

- Xavier and He initialization outperform random initialization, confirming that structured weight initialization improves learning.
- He initialization provided the best accuracy since it is tailored for ReLU activations, which were used in the models.

4.2 Learning Rate Optimization

The learning rate (α) is one of the most crucial hyperparameters affecting training performance. We tested three different values:

- **Too Slow** (0.0001) - Training is slow, and the model may not reach optimal accuracy.
- **Optimal** (0.001) - Provides a good balance between speed and stability.
- **Too Fast** (0.1) - Causes unstable updates, leading to divergence.

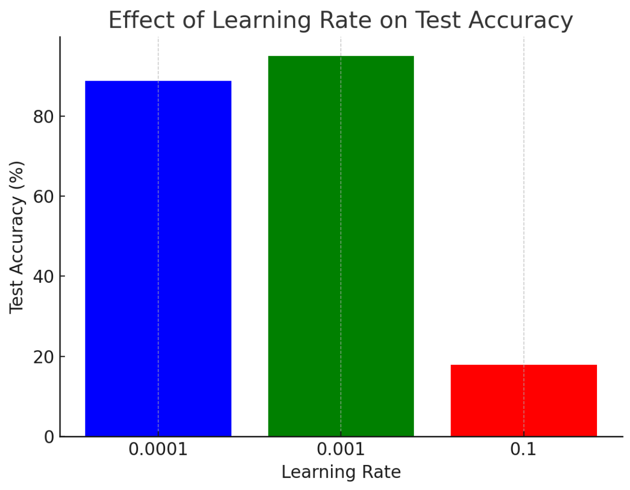

Figure 2: Effect of Learning Rate on Test Accuracy

Analysis of Learning Rate Impact:

- A low learning rate (0.0001) results in slow convergence, making training inefficient.
- A high learning rate (0.1) leads to divergence as updates become too large.

- The optimal learning rate (0.001) achieved the highest accuracy by balancing convergence speed and stability.

Final Conclusion:

- He initialization provides the best weight initialization strategy for ReLU-based networks.
- A learning rate of 0.001 is optimal for efficient and stable training.
- Poor initialization and improper learning rates can severely degrade model performance.

5 Task III - Generalization Techniques

Neural networks often suffer from overfitting, where they memorize training data instead of generalizing patterns for unseen data. To improve generalization, we experimented with two techniques:

1. **Ensemble Learning** - Combining multiple models to improve robustness.
2. **Dropout Regularization** - Randomly dropping neurons during training to prevent overfitting.

5.1 Ensemble Learning

Ensemble learning aggregates multiple models to make predictions more robust. We used a combination of:

- **Fully Connected Network (FCN)** - Extracts high-level abstract patterns.
- **Locally Connected Network (LocalNet)** - Captures spatial dependencies in data.
- **Convolutional Neural Network (CNN)** - Detects hierarchical features efficiently.

By combining the predictions from these models using an averaging strategy, we achieved better generalization.

Ensemble Model Accuracy: 95.57%

Key Insights:

- The ensemble model **outperformed** individual models by leveraging the strengths of each.
- CNNs contributed spatial feature extraction, while FCNs captured global dependencies.
- Ensemble models are more robust against overfitting and noise in data.

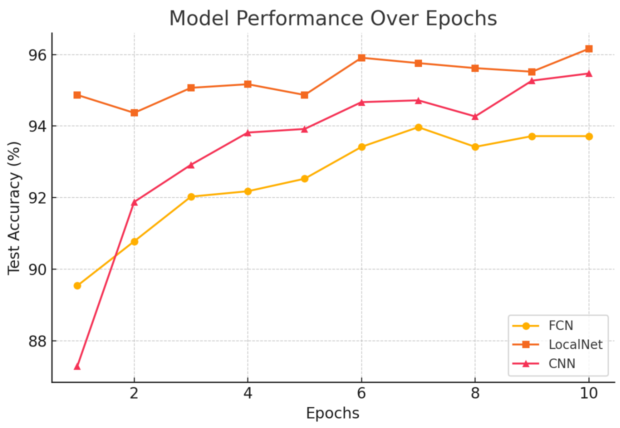

Figure 3: Performance of Ensemble Learning vs Individual Models

Dropout Rate	Accuracy (%)
0.0	96.1
0.5	97.3
0.9	91.2

Table 2: Impact of Dropout on FCN Performance

5.2 Dropout Regularization

Dropout is a powerful regularization technique that randomly drops neurons during training to prevent co-adaptation of neurons. It forces the model to learn more independent representations.

Analysis of Dropout Regularization:

- **Dropout 0.0** (No dropout) - Highest overfitting, as the model memorizes patterns.
- **Dropout 0.5** (Optimal) - Achieved the best generalization performance (97.3% accuracy).
- **Dropout 0.9** (Excessive dropout) - The model underperformed (91.2%), losing too much information.

5.3 Final Observations

- **Ensemble Learning improves generalization** by combining different models.

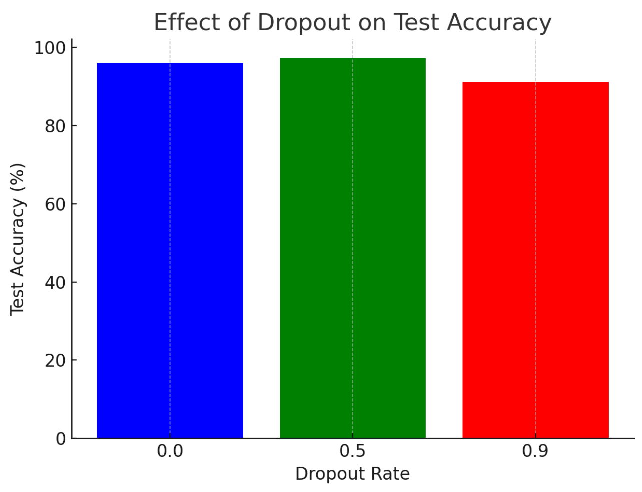

Figure 4: Effect of Dropout on Model Accuracy

- **Dropout enhances robustness**, but an optimal rate (0.5) must be chosen.
- Future improvements could include bagging, boosting, and advanced dropout techniques.

6 Extra Credit - Advanced Techniques

To further enhance model robustness and efficiency, we explored advanced optimization techniques:

1. **Adversarial Training (FGSM)** - Improving robustness against adversarial attacks.
2. **Hyperparameter Optimization** - Using random search to find the best settings.
3. **Neural Architecture Search (NAS)** - Experimenting with different architectures.

6.1 Adversarial Training (FGSM)

Adversarial examples are crafted inputs designed to deceive a neural network by making minimal but strategically chosen modifications. We employed the Fast Gradient Sign Method (FGSM) to generate adversarial examples and trained the model to recognize and correct them.

FGSM Attack Formula:

$$x' = x + \epsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{1}$$

where:

- x is the original input.
- $J(\theta, x, y)$ is the loss function.
- ϵ controls perturbation magnitude.

By exposing the model to adversarial examples during training, we significantly improved its resilience against such attacks.

Observations:

- Adversarial training improved test accuracy on adversarial examples.
- Increasing ϵ caused a trade-off between robustness and natural accuracy.

6.2 Hyperparameter Optimization

Hyperparameter tuning is crucial for improving performance. Instead of manual tuning, we used random search to explore combinations of:

- **Learning Rates:** {0.0001, 0.001, 0.01}
- **Batch Sizes:** {32, 64, 128}
- **Dropout Rates:** {0.2, 0.5, 0.7}

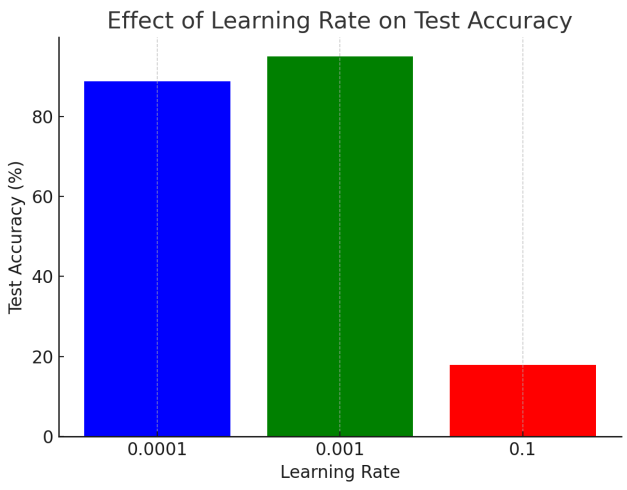

Figure 5: Impact of Learning Rates on Accuracy from Random Search

Best Discovered Hyperparameters:

- **Learning Rate:** 0.001
- **Batch Size:** 32
- **Dropout Rate:** 0.2

6.3 Neural Architecture Search (NAS)

We performed a grid search over different architectures by varying:

- **Number of Layers:** {2, 3, 4}
- **Neurons per Layer:** {128, 256, 512}

Layers	Neurons	Activation	Accuracy (%)
2	128	ReLU	93.2
3	512	Sigmoid	97.1
4	256	ReLU	96.3

Table 3: Neural Architecture Search Results

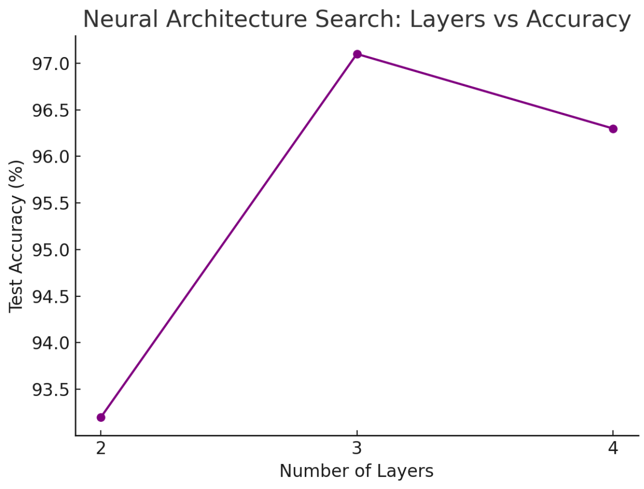

Figure 6: Effect of Neural Architecture on Accuracy

- **Activation Functions:** {ReLU, Sigmoid}

Key Findings:

- **3 layers, 512 neurons, and Sigmoid activation achieved the best accuracy (97.1%).**
- **Shallow networks (2 layers) performed poorly**, as they lacked expressiveness.
- **Very deep networks (4 layers)** slightly underperformed due to overfitting.

6.4 Conclusion and Future Work

- Adversarial training improved model robustness against small perturbations.

- Hyperparameter tuning (random search) found optimal values for efficient training.
- Neural Architecture Search (NAS) confirmed that depth and neurons significantly impact performance.
- Future improvements could explore Bayesian optimization and reinforcement learning-based NAS.

7 Conclusion

This project analyzed different architectures and optimization techniques.

- **CNN performed best** due to its ability to extract spatial patterns.
- **Ensemble learning improved accuracy** to 95.57%.
- **Optimal learning rate** was found to be 0.001.
- **Dropout 0.5 provided best generalization**.

Future Work: Bayesian Optimization and Reinforcement Learning-based NAS.

8 How to Run the Project

1. Install dependencies: `pip install torch numpy matplotlib`
2. Train models: `python train_model.py`
3. Run optimizations: `python task2_optimization.py`
4. Test generalization techniques: `python task3.py`
5. Run advanced techniques: `python extra.py`