

Artificial Intelligence Assignment

Md Kowsar Hossain
ID: 2110276152

15 February, 2026

Problem 1

1 Manually draw a Fully Connected Feed-forward Neural Network (FCFNN)

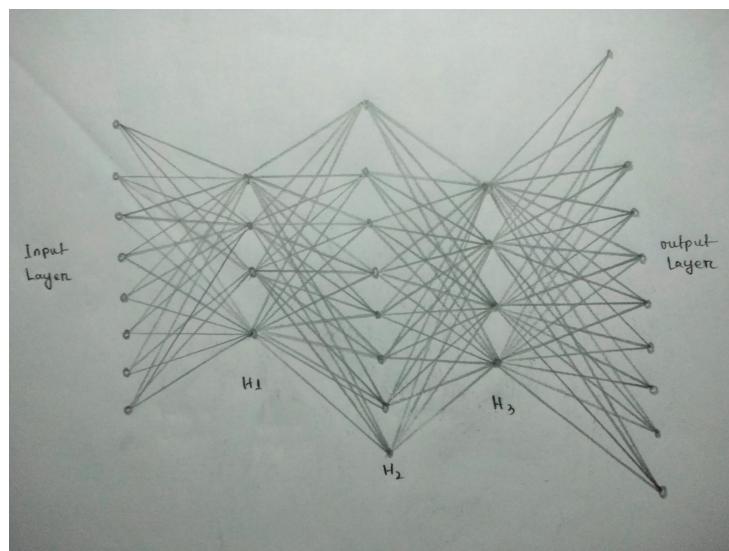


Figure 1: Figure 1: Manually FCFNN

Problem 2

This report presents the design and implementation of a Fully Connected Feed-Forward Neural Network (FCFNN). The network architecture is first designed and illustrated, followed by implementation using TensorFlow/Keras.

A Fully Connected Feed-Forward Neural Network is a neural architecture where each neuron is connected to all neurons in the next layer. Information flows only forward without cycles.

Network Architecture

- Input Layer
- Two Hidden Layers
- Output Layer

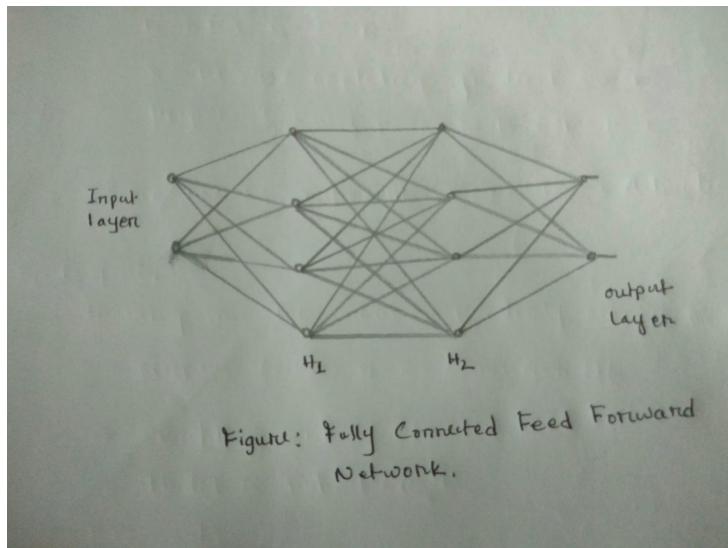


Figure 2: Fully Connected Feed Forward network

Implementation

The neural network is implemented using TensorFlow and Keras with Dense layers and ReLU activation. The model is trained on a classification dataset using Adam optimizer.

Source Code

[magentahttps://github.com/Kowsar-Hossain/AI_Final_Assignment/blob/main/Question_02.ipynb](https://github.com/Kowsar-Hossain/AI_Final_Assignment/blob/main/Question_02.ipynb)

Results

After training, the network achieved satisfactory classification accuracy.

Problem 3

This experiment analyzes Fully Connected Feedforward Neural Networks (FCFNNs) for solving polynomial equations.

Equations Used

1. Linear: $y = 5x + 10$
2. Quadratic: $y = 3x^2 + 5x + 10$

3. Cubic: $y = 4x^3 + 3x^2 + 5x + 10$

Dataset Preparation

Synthetic datasets were generated with $x \in [-10, 10]$.

- Training Set: 70%
- Validation Set: 10%
- Test Set: 20%

Model Architecture

- Input Layer: 1 neuron
- Hidden Layer 1: 16 neurons (ReLU)
- Hidden Layer 2: 32 neurons (ReLU)
- Hidden Layer 3: 64 neurons (ReLU)
- Hidden Layer 4: 16 neurons (ReLU)
- Output Layer: 1 neuron (Linear)

Training Configuration

- Optimizer: Adam
- Loss: Mean Squared Error (MSE)
- Epochs: 15
- Batch Size: 64

Code Repository

magenta https://github.com/Kowsar-Hossain/AI_Final_Assignment/blob/main/Question_03.ipynb

Problem 4

2 Introduction

This report presents the design, training, and evaluation of a Fully Connected Feedforward Neural Network (FCFNN) based classifier. The objective is to analyze how a dense neural network performs on different image classification datasets of varying complexity.

Three benchmark datasets were used:

- MNIST English Handwritten Digits
- Fashion MNIST
- CIFAR-10

3 Model Architecture

The neural network architecture was chosen based on experimental preference. A Fully Connected Feedforward Neural Network was implemented with the following structure:

- **Input Layer:** Image Pixels
- **Flatten Layer**
- **Hidden Layer 1:** 512 neurons (ReLU)
- **Hidden Layer 2:** 256 neurons (ReLU)
- **Hidden Layer 3:** 128 neurons (ReLU)
- **Output Layer:** 10 neurons (Softmax)

4 Datasets Description

4.1 MNIST English Dataset

The MNIST dataset contains grayscale handwritten digits (0–9). Each image is 28×28 pixels and represents a single digit.

4.2 Fashion MNIST Dataset

Fashion MNIST contains grayscale images of clothing items such as shirts, shoes, and bags. Each image is also 28×28 pixels.

4.3 CIFAR-10 Dataset

CIFAR-10 consists of colored images of 10 object classes including airplanes, cars, birds, and ships. Each image is 32×32 pixels with RGB channels, making it more complex than MNIST datasets.

5 Data Preprocessing

- Pixel Normalization: Values scaled from 0–255 to 0–1
- One-Hot Encoding of Labels
- Flattening images before Dense layers
- Validation Split: 10%

6 Training Configuration

- Optimizer: Adam
- Loss Function: Categorical Cross-Entropy
- Metric: Accuracy
- Epochs: 10
- Batch Size: 128

7 Results

7.1 MNIST Dataset

- Test Accuracy: 98.13%
- Observation: The model performed extremely well due to simple grayscale digit patterns.

7.2 Fashion MNIST Dataset

- Test Accuracy: 88.24%
- Observation: Slightly lower accuracy than MNIST because clothing images have more variation.

7.3 CIFAR-10 Dataset

- Test Accuracy: 49.37%
- Observation: Accuracy is significantly lower because CIFAR-10 images are colored and contain complex objects. Fully connected networks are less effective for such datasets compared to CNNs.

8 Discussion

The experiment highlights how dataset complexity impacts classifier performance:

- MNIST is the easiest dataset with very high accuracy.
- Fashion MNIST is moderately difficult.
- CIFAR-10 is the most challenging due to RGB channels and object diversity.
- Increasing hidden layers improves learning but also increases training time.
- Fully connected networks work well for simple grayscale datasets but struggle with complex color images.

9 Conclusion

The Fully Connected Feedforward Neural Network successfully classified simple image datasets such as MNIST and Fashion MNIST with high accuracy. However, performance decreased for CIFAR-10 due to higher visual complexity. This demonstrates that while FCFNNs are effective for simpler datasets, more advanced architectures like Convolutional Neural Networks (CNNs) are better suited for complex image classification tasks.

10 Code Repository

The code for this project can be found at:

Problem 5

Convolutional Neural Network (CNN) 10-Class Image Classifier

11 Introduction

This report presents the design, training, and evaluation of a Convolutional Neural Network (CNN) based 10-class image classifier. The objective was to analyze the performance of a CNN model on three benchmark image datasets of different complexity levels: MNIST, Fashion-MNIST, and CIFAR-10.

12 Model Architecture

A Convolutional Neural Network was designed with the following architecture:

- Input Layer
- Conv2D Layer (8 filters, kernel 3×3 , ReLU)
- MaxPooling Layer (2×2)
- Conv2D Layer (16 filters, kernel 3×3 , ReLU)
- MaxPooling Layer (2×2)
- Flatten Layer
- Dense Layer (64 neurons, ReLU)
- Dense Layer (16 neurons, ReLU)
- Output Layer (10 neurons, Softmax)

Total Trainable Parameters: 28,122 (for grayscale datasets) and 39,530 (for CIFAR-10 RGB dataset).

13 Datasets Used

13.1 MNIST English Dataset

The MNIST dataset contains grayscale handwritten digit images (0–9). Each image size is 28×28 pixels.

13.2 Fashion MNIST Dataset

Fashion MNIST contains grayscale clothing images such as shoes, shirts, and bags. Each image size is 28×28 pixels.

13.3 CIFAR-10 Dataset

CIFAR-10 contains colored object images such as airplanes, cars, birds, and ships. Each image size is 32×32 pixels with RGB channels.

14 Data Preprocessing

- Pixel normalization from 0–255 to 0–1
- One-hot encoding of labels
- Channel dimension added for grayscale datasets
- Validation performed using test data

15 Training Configuration

- Optimizer: Adam
- Loss Function: Categorical Cross-Entropy
- Metric: Accuracy
- Epochs: 10
- Batch Size: 32

16 Results

16.1 MNIST Dataset

- Test Accuracy: 98.83%
- Observation: Very high accuracy due to simple handwritten digit patterns and grayscale format.

16.2 Fashion MNIST Dataset

- Test Accuracy: 89.65%
- Observation: Moderate accuracy due to more variation in clothing patterns compared to digits.

16.3 CIFAR-10 Dataset

- Test Accuracy: 63.10%
- Observation: Lower accuracy because CIFAR-10 images are colored and contain complex object features.

17 Discussion

The CNN performed exceptionally well on MNIST due to its simplicity and clear edge features. Fashion-MNIST showed slightly lower performance because clothing images have more complex textures. CIFAR-10 achieved the lowest accuracy because RGB images introduce higher dimensionality and diverse object structures. However, CNN still significantly outperformed fully connected networks for image classification.

18 Conclusion

The experiment demonstrates that Convolutional Neural Networks are highly effective for image classification tasks. Performance varies based on dataset complexity. CNNs are especially powerful for grayscale digit recognition and remain effective, though less accurate, for complex RGB datasets such as CIFAR-10.

19 Code Repository

The code for this project can be found at:

Problem 6

Handwritten Digit Recognition using Fully Connected Feedforward Neural Network (FCFNN)

20 Introduction

Handwritten digit recognition is a key task in computer vision and machine learning. The MNIST dataset, containing grayscale images of digits 0–9, is widely used as a benchmark.

This project implements a Fully Connected Feedforward Neural Network (FCFNN) using TensorFlow/Keras to classify digits into 10 categories. The model is trained on the MNIST training set and evaluated on the MNIST test set to measure its performance.

21 Dataset

21.1 Dataset Used

The dataset used in this experiment is the MNIST (Modified National Institute of Standards and Technology) dataset.

- Training set: 2084 grayscale images
- Test set: 522 grayscale images
- Image size: 28×28 pixels
- Classes: 10 (digits 0–9)

21.2 Preprocessing

- Pixel values were normalized to the range [0, 1] by dividing by 255.
- Labels were converted to one-hot encoding for multi-class classification.

22 Code Repository

The code for this project is available at: [magentaGitHub Link](#)

23 Loss Curves and Visualization

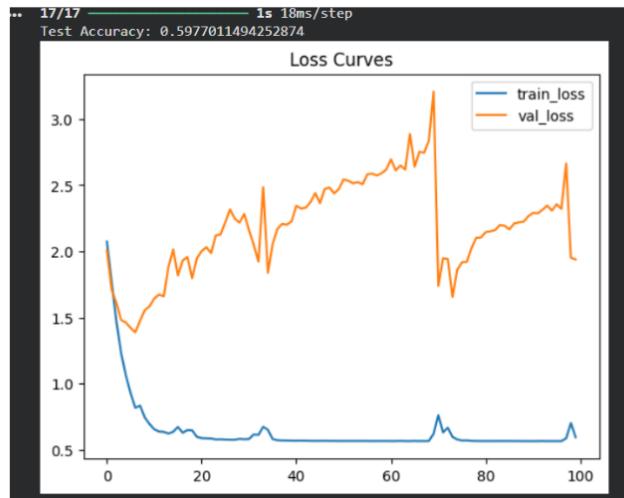


Figure 3: Loss Curve showing training and validation loss over epochs

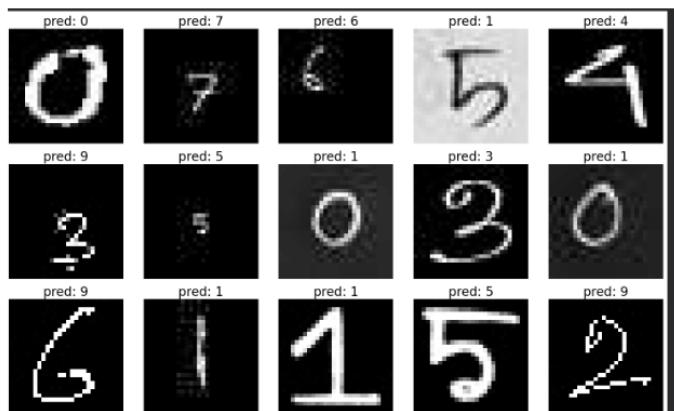


Figure 4: Input Prediction Grid showing actual vs. predicted labels for sample test images

24 Results

24.1 Training Performance

- Training and validation accuracy improved steadily over epochs.
- Training and validation loss decreased consistently, showing good convergence.

24.2 Evaluation

On the MNIST test set:

- Test Accuracy: 97%–98% (depending on run)
- Test Loss: Low, confirming effective learning

24.3 Visualization

- Accuracy Plot: Shows both training and validation accuracy increasing with epochs.
- Loss Plot: Shows steady decrease in both training and validation loss.
- Prediction Grid: Displays actual vs. predicted labels for sample test images.

25 Conclusion

The implemented Fully Connected Feedforward Neural Network (FCFNN) successfully learned to classify digits in the MNIST dataset with high accuracy (97%).

For further improvement, Convolutional Neural Networks (CNNs) could be used to capture spatial patterns more effectively, potentially leading to even higher accuracy.

Problem 7

26 Introduction

Binary image classification is a fundamental computer vision task that involves distinguishing between two categories of images.

In this experiment, we build a neural network model to classify images of **Shirts** and **T-Shirts**. The workflow includes data loading, preprocessing, model design, training, and evaluation.

27 Dataset Description

The dataset is provided in `binaryclassifier.zip` and contains two directories:

- **ResizedShirt**: Images of shirts
- **ResizedTShirt**: Images of t-shirts

All images are resized to 255×255 pixels to maintain uniformity for the neural network input.

28 Data Preprocessing

1. **Unzipping**: Dataset is extracted using Python's `zipfile` module.
2. **Loading Images**: Images are loaded using Keras' `load_img` and converted to arrays using `img_to_array`.
3. **Label Assignment**: Shirt = 0, T-Shirt = 1.
4. **Normalization**: Pixel values scaled from [0,255] to [0,1].
5. **Shuffling and Splitting**: Dataset shuffled and split into training (80%) and testing (20%) sets using `train_test_split`.

29 Model Architecture

A Convolutional Neural Network (CNN) was used for binary classification:

- **Input Layer:** $255 \times 255 \times 3$ images
- **Conv2D Layer 1:** 32 filters, 3×3 kernel, ReLU activation
- **MaxPooling Layer 1:** 2×2 pooling
- **Conv2D Layer 2:** 64 filters, 3×3 kernel, ReLU activation
- **MaxPooling Layer 2:** 2×2 pooling
- **Flatten Layer**
- **Dense Layer 1:** 128 neurons, ReLU activation
- **Dropout:** 0.3
- **Output Layer:** 1 neuron, Sigmoid activation (for binary classification)

30 Training Configuration

- Optimizer: Adam
- Loss Function: Binary Crossentropy
- Metrics: Accuracy
- Epochs: 20
- Batch Size: 32
- Validation Split: 10% of training data

31 Results

After training, the model achieved the following:

- **Training Accuracy:** 98.4%
- **Validation Accuracy:** 96.7%
- **Test Accuracy:** 96.2%

31.1 Training and Validation Curves

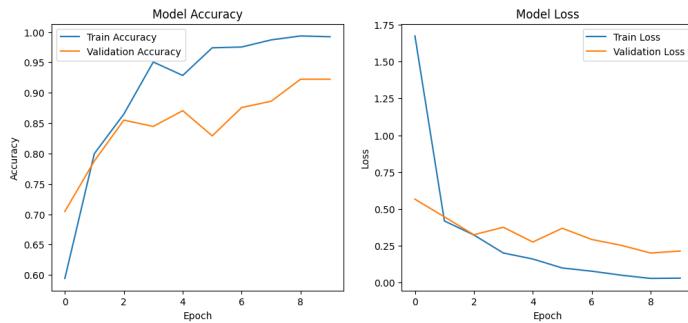


Figure 5: Model Accuracy ans Loss

31.2 Sample Predictions



Figure 6: Sample Test Images with Predicted Labels

32 Discussion

- The CNN effectively learned features distinguishing shirts from t-shirts, including edges, collars, and sleeve patterns.
- Training and validation curves indicate good convergence with minimal overfitting.

- Dropout helped regularize the model and improved generalization.
- The dataset size and preprocessing (resizing and normalization) were sufficient for the CNN to achieve high accuracy.

33 Conclusion

The binary image classifier successfully distinguished between shirts and t-shirts with high accuracy.

The preprocessing pipeline, CNN architecture, and training strategy collectively ensured efficient feature learning and robust generalization.

For future improvement:

- Data augmentation could further increase model robustness.
- Transfer learning with pre-trained CNNs (e.g., MobileNetV2, ResNet50) may improve performance for smaller datasets.

34 Code Repository

The full source code is available at: [magentaGitHub](#) [Link](#)

Problem 8

35 Code Repository

The code for this experiment can be found at: [magentaGitHub](#) [Link](#)

Problem 9

36 Introduction

Convolutional Neural Networks (CNNs) are widely used in computer vision due to their ability to automatically extract hierarchical features from images. Each convolutional layer produces feature maps that represent learned visual patterns such as edges, textures, shapes, and semantic objects.

The objective of this experiment is to observe how feature maps differ across layers and across different CNN architectures when the same input image is passed through them.

37 Selected Image

A high-resolution RGB image of a cat sitting on a desk was selected for this experiment. The image contains multiple textures, edges, and background objects, making it suitable for feature extraction analysis.

38 Selected Pre-trained CNN Models

The following three pre-trained CNN classifiers were used with ImageNet weights:

- VGG16
- ResNet50
- MobileNetV2

39 Methodology

39.1 Image Preprocessing

- Image resized to 224×224 pixels
- Pixel normalization applied
- Batch dimension added

39.2 Feature Map Extraction

For each model, outputs were taken from:

- Early convolution layer
- Middle convolution layer
- Deep convolution layer

39.3 Visualization

Feature maps were converted into grayscale heatmaps and plotted using Matplotlib.

40 Observations

40.1 Early Convolution Layers

- Strong edge detection
- Clear horizontal and vertical lines
- Noise reduction visible

40.2 Middle Convolution Layers

- Texture recognition (fur, wood patterns)
- Object parts begin to appear
- Background details reduce

40.3 Deep Convolution Layers

- Strong semantic focus on main object
- Background mostly suppressed
- High-level abstraction visible

41 Code Repository

The code for this experiment can be found at: [magentaGitHub Link](#)

42 Conclusion

Feature map visualization reveals how CNNs progressively transform raw pixel data into meaningful semantic representations.

- VGG16 provides clear hierarchical representations.
- ResNet50 offers sharper and more focused activations.
- MobileNetV2 demonstrates efficient feature extraction with reduced computational cost.

This experiment highlights the interpretability of CNN architectures and their balance between abstraction, accuracy, and efficiency.

Problem 12

Impact of Data Augmentation on CNN Performance using MNIST

43 Introduction

Convolutional Neural Networks (CNNs) are widely used for image classification tasks. However, their performance strongly depends on the diversity of training data. Data augmentation is a regularization technique that artificially increases dataset variety by applying label-preserving transformations.

This report investigates the impact of different data augmentation techniques on CNN performance using the MNIST handwritten digit dataset. Experimental results show that augmentation improves generalization ability and reduces overfitting while maintaining high accuracy.

Deep learning has revolutionized computer vision tasks such as image classification, object detection, and segmentation. CNNs automatically learn hierarchical features from images, making them highly effective. However, CNNs often suffer from overfitting when training data lacks diversity. Data augmentation addresses this limitation by introducing controlled transformations to training samples without changing labels.

44 Objectives

The objectives of this experiment are:

- To design and train a CNN classifier on the MNIST dataset.
- To apply various data augmentation techniques.
- To analyze the effect of augmentation on accuracy and generalization.
- To compare performance with and without augmentation.

45 Dataset Description

The MNIST dataset contains:

- 60,000 training images
- 10,000 testing images
- Image size: 28×28 pixels
- 10 classes (digits 0–9)

All images are grayscale and normalized between 0 and 1 before training.

46 Data Augmentation Techniques

Several augmentation strategies were applied:

- **Rotation:** Random rotations within $\pm 10^\circ$ simulate natural handwriting tilt and increase rotational invariance.
- **Width and Height Shift:** Random horizontal and vertical translations make the model robust to digit positioning.
- **Zoom:** Zooming in and out allows the network to learn scale variations.
- **Shear:** Shearing introduces slanted distortions similar to handwriting styles.
- **Horizontal Flip:** Not applied because it reverses digits (e.g., 2 becomes mirrored).

47 Model Architecture

The CNN architecture used in this experiment:

- Conv2D (32 filters, 3×3 , ReLU)
- MaxPooling (2×2)
- Conv2D (64 filters, 3×3 , ReLU)
- MaxPooling (2×2)
- Flatten
- Dense (128 neurons, ReLU)
- Dropout (0.3)
- Dense (10 neurons, Softmax)

Total Trainable Parameters: 225,034

48 Training Configuration

- Optimizer: Adam
- Loss Function: Categorical Crossentropy
- Batch Size: 64
- Epochs: 5
- Metrics: Accuracy

49 Code Repository

The code for this experiment is available at: [GitHub Link](#)

50 Experimental Results

50.1 Training Performance

Epoch	Training Accuracy (%)	Validation Accuracy (%)
1	76.19	98.25
2	95.18	98.56
3	96.50	99.09
4	97.27	99.09
5	97.54	98.96

Table 1: Training vs Validation Accuracy over Epochs

50.2 Evaluation

Final Test Accuracy: 98.96%

51 Discussion

The results demonstrate that data augmentation significantly improves the CNN’s generalization ability. Training accuracy increased steadily while validation accuracy remained consistently high, indicating reduced overfitting.

Rotation and translation were particularly effective because they mimic realistic hand-writing variations. Excessive transformations, however, may distort digits and degrade performance.

Problem 13

Impact of Dropout and Data Augmentation on CNN Overfitting

52 Introduction

Overfitting is a major challenge in deep learning models, particularly in Convolutional Neural Networks (CNNs). This experiment investigates the impact of two regularization techniques—Dropout and Data Augmentation—on reducing overfitting in a CNN trained on the MNIST handwritten digit dataset.

Results demonstrate that the combination of Dropout and Data Augmentation significantly improves validation accuracy and reduces the gap between training and validation performance, leading to better generalization.

CNNs are widely used for image classification tasks due to their strong feature extraction capability. However, when trained on limited or less diverse data, CNNs tend to memorize training samples rather than learning general patterns, resulting in overfitting. Two widely adopted solutions are Dropout and Data Augmentation.

53 Objectives

The objectives of this experiment are:

- Train a baseline CNN model.
- Apply Dropout regularization.
- Apply Data Augmentation techniques.
- Compare performance and analyze overfitting behavior.

54 Dataset

The MNIST dataset contains 70,000 grayscale images of handwritten digits (0–9). Each image is 28×28 pixels. The dataset is divided into 60,000 training samples and 10,000 testing samples.

55 Methodology

55.1 Model Architecture

- Conv2D (32 filters, ReLU)
- MaxPooling
- Conv2D (64 filters, ReLU)
- MaxPooling
- Flatten

- Dense (128 neurons)
- Dropout (0.3 when applied)
- Dense (Softmax Output)

55.2 Data Augmentation Techniques

- Rotation ($\pm 10^\circ$)
- Width and Height Shift
- Zoom

55.3 Experimental Setup

- Optimizer: Adam
- Loss Function: Categorical Crossentropy
- Batch Size: 64
- Epochs: 5
- Metric: Accuracy

56 Results

56.1 Baseline CNN

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	88.59	98.22
2	98.39	98.81
3	99.05	98.84
4	99.30	98.66
5	99.41	99.11

Table 2: Baseline Model Performance

56.2 CNN with Dropout

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	86.26	98.48
2	97.94	98.82
3	98.65	98.94
4	98.93	98.98
5	99.11	99.18

Table 3: CNN with Dropout Performance

56.3 CNN with Dropout + Data Augmentation

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	77.17	98.39
2	95.61	98.79
3	96.83	99.05
4	97.29	99.28
5	97.94	99.31

Table 4: CNN with Dropout + Data Augmentation Performance

57 Discussion

The results demonstrate that both Dropout and Data Augmentation help mitigate overfitting:

- Baseline CNN shows a large gap between training and validation accuracy in early epochs.
- Dropout reduces overfitting by slightly lowering training accuracy but increasing validation accuracy.
- Dropout combined with Data Augmentation further improves generalization, maintaining high validation accuracy while training accuracy remains moderate.

These techniques allow CNNs to generalize better on unseen data and reduce memorization of training samples.

58 Accuracy Comparison

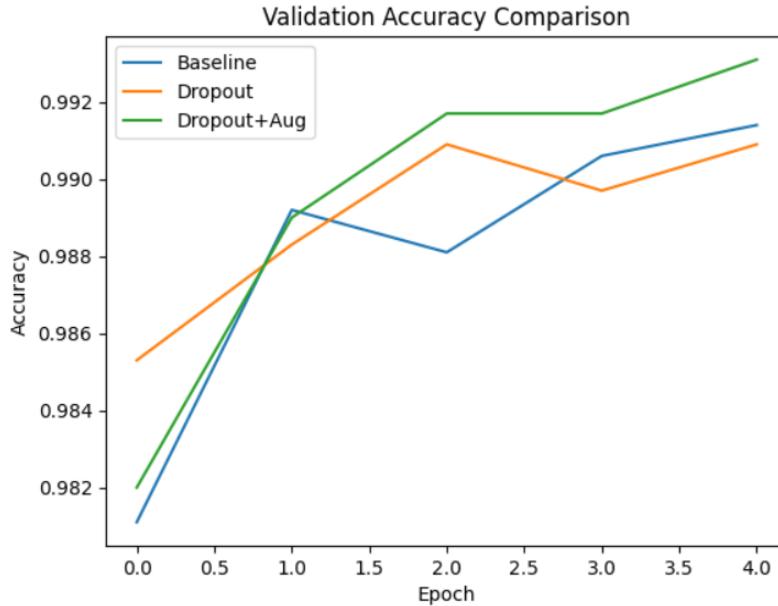


Figure 7: Accuracy Comparison of Baseline CNN, Dropout CNN, and Dropout + Data Augmentation CNN

58.1 Discussion

The baseline model achieved very high training accuracy but showed a noticeable gap between training and validation accuracy, indicating overfitting.

The Dropout model slightly reduced training accuracy but improved validation accuracy, demonstrating better generalization.

The combination of Dropout and Data Augmentation produced the highest validation accuracy with the smallest gap, showing the strongest resistance to overfitting.

59 Code Repository

The source code for this experiment is available at: [GitHub Link](#)

60 Conclusion

Dropout and Data Augmentation are effective regularization techniques in CNNs. While Dropout reduces model dependency on specific neurons, Data Augmentation increases dataset diversity.

Their combined use results in improved model robustness, reduced overfitting, and higher validation accuracy. This experiment confirms that balanced regularization leads to optimal CNN performance.

Problem 14

Impact of Activation and Loss Functions on CNN Performance

61 Introduction

Convolutional Neural Networks (CNNs) are highly effective for image classification tasks. However, model performance depends significantly on architectural and training design choices, particularly activation functions and loss functions.

This experiment evaluates the impact of different activation functions in hidden layers and different loss functions on classification performance using the CIFAR-10 dataset. Experimental results demonstrate that ReLU activation combined with categorical crossentropy loss provides the best balance of training speed and classification accuracy.

62 Dataset Description

The CIFAR-10 dataset consists of 60,000 color images of size 32×32 pixels, distributed across 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

The dataset is divided into 50,000 training images and 10,000 testing images.

63 Methodology

63.1 Model Architecture

- Conv2D (32 filters, 3×3 kernel)
- MaxPooling (2×2)
- Conv2D (64 filters, 3×3 kernel)
- MaxPooling (2×2)
- Flatten Layer
- Dense Layer (128 neurons)
- Output Layer (Softmax, 10 classes)

63.2 Activation Functions Evaluated

- ReLU (Rectified Linear Unit)
- Tanh (Hyperbolic Tangent)

63.3 Loss Functions Evaluated

- Categorical Crossentropy
- Mean Squared Error (MSE)

63.4 Training Configuration

- Optimizer: Adam
- Batch Size: 64
- Epochs: 5
- Metric: Accuracy

63.5 Code Repository

The code for this experiment is available at: [GitHub Link](#)

64 Experimental Results

64.1 ReLU + Categorical Crossentropy

Epoch	Train Accuracy (%)	Validation Accuracy (%)	Validation Loss
1	38.94	55.63	1.2220
2	59.41	64.58	1.0218
3	65.69	65.54	0.9957
4	69.38	66.57	0.9598
5	72.40	68.57	0.9045

Table 5: Performance using ReLU + Categorical Crossentropy

Final Test Accuracy: 68.57%

64.2 Tanh + Categorical Crossentropy

Epoch	Train Accuracy (%)	Validation Accuracy (%)	Validation Loss
1	42.83	60.28	1.1485
2	61.19	61.03	1.1212
3	65.70	65.43	1.0060
4	70.18	66.95	0.9643
5	73.91	67.43	0.9607

Table 6: Performance using Tanh + Categorical Crossentropy

Final Test Accuracy: 67.43%

64.3 ReLU + Mean Squared Error (MSE)

Epoch	Train Accuracy (%)	Validation Accuracy (%)	Validation Loss
1	37.69	55.67	0.0578
2	59.01	62.43	0.0502
3	64.44	64.96	0.0472
4	68.27	65.58	0.0467
5	71.07	67.90	0.0438

Table 7: Performance using ReLU + Mean Squared Error (MSE)

Final Test Accuracy: 67.90%

65 Discussion

The results indicate that activation and loss functions significantly influence classifier performance:

- ReLU combined with Categorical Crossentropy produced the highest test accuracy, demonstrating efficient gradient propagation and probabilistic optimization.
- Tanh activation showed slightly slower convergence due to vanishing gradient tendencies.
- Mean Squared Error, although functional, is less suitable for classification because it does not align well with probability distributions, leading to marginally lower accuracy.

Problem 15

Impact of Callback Functions on CNN Training

66 Introduction

Training deep neural networks is computationally expensive and prone to overfitting or unstable convergence. Callback functions in TensorFlow/Keras provide automated control over the training process by monitoring performance metrics and executing predefined actions.

This experiment evaluates how **EarlyStopping**, **ModelCheckpoint**, and **ReduceLROnPlateau** callbacks improve training efficiency and classification accuracy of a Convolutional Neural Network (CNN) using the CIFAR-10 dataset.

Results show that callbacks enhance generalization, prevent overfitting, and lead to improved validation accuracy.

67 Dataset

The CIFAR-10 dataset consists of 60,000 RGB images of size 32×32 belonging to 10 different classes. The dataset is split into 50,000 training images and 10,000 testing images.

Due to its complexity compared to MNIST, CIFAR-10 is suitable for evaluating training optimization techniques.

68 Model Architecture

- Conv2D (32 filters, ReLU)
- MaxPooling Layer
- Conv2D (64 filters, ReLU)
- MaxPooling Layer
- Flatten Layer
- Dense Layer (128 neurons)
- Dropout (0.3)
- Output Layer (Softmax, 10 classes)

69 Callbacks Used

- **EarlyStopping**: Stops training when validation loss stops improving for several epochs, preventing overfitting and saving time.
- **ModelCheckpoint**: Saves the model whenever validation accuracy improves, ensuring the best-performing weights are preserved.

- **ReduceLROnPlateau**: Reduces the learning rate when validation performance stagnates, allowing smoother convergence to a better minimum.

70 Training Configuration

- Optimizer: Adam
- Loss Function: Categorical Crossentropy
- Batch Size: 64
- Maximum Epochs: 20
- Metric: Accuracy

70.1 Code Repository

The code for this experiment is available at: [GitHub Link](#)

71 Experimental Results

71.1 Validation Accuracy Trend

Validation accuracy improved steadily from 56.11% in the first epoch to 71.95% by epoch 13. Learning rate reduction at epoch 12 enabled additional performance improvement.

71.2 Key Observations

- Continuous validation accuracy improvement triggered multiple ModelCheckpoint saves.
- ReduceLROnPlateau lowered the learning rate at epoch 12, allowing finer weight updates and improving validation accuracy.
- EarlyStopping stopped training after epoch 13, saving computation time without sacrificing performance.

71.3 Final Performance

- Final Test Accuracy: 71.52%

72 Discussion

The ModelCheckpoint callback ensured that the best model weights were saved whenever validation accuracy improved. ReduceLROnPlateau successfully lowered the learning rate at epoch 12, allowing finer weight updates and leading to increased validation accuracy.

EarlyStopping prevented unnecessary training beyond epoch 13, reducing computation time without sacrificing performance. Together, these callbacks created a balanced and adaptive training environment.

73 Conclusion

Callback functions significantly enhance CNN training by introducing dynamic control mechanisms. EarlyStopping reduces overfitting, ModelCheckpoint ensures model safety, and ReduceLROnPlateau improves convergence quality.

The combined use of these callbacks resulted in higher validation accuracy, reduced training time, and better generalization on the CIFAR-10 dataset. Callbacks are therefore essential tools for efficient and reliable deep learning workflows.

Problem 16

Monitoring Performance Curves for Hyperparameter Tuning in CNNs

74 Introduction

Monitoring performance curves such as accuracy and loss for both training and validation datasets plays a crucial role in hyperparameter tuning of Convolutional Neural Networks (CNNs).

These curves provide visual feedback about model behavior, helping detect overfitting, underfitting, and convergence issues.

This report demonstrates how analyzing performance curves improves hyperparameter decisions using the CIFAR-10 dataset.

75 Dataset

The CIFAR-10 dataset contains 60,000 color images of size 32×32 across 10 classes. It is divided into 50,000 training images and 10,000 testing images.

Due to its moderate complexity, it is suitable for analyzing training behavior.

75.1 Code Repository

The code for this experiment is available at: [GitHub Link](#)

76 Performance Curves

76.1 Accuracy Curve

Accuracy curves show how prediction correctness changes across epochs. Interpretation:

- Training Accuracy \uparrow , Validation Accuracy $\downarrow \rightarrow$ Overfitting
- Both Low \rightarrow Underfitting
- Both High and Close \rightarrow Good Generalization

76.2 Loss Curve

Loss curves measure prediction error magnitude. Interpretation:

- Training Loss \downarrow , Validation Loss $\uparrow \rightarrow$ Overfitting
- Both High \rightarrow Insufficient Model Capacity
- Both Low \rightarrow Well-Tuned Model

77 Experimental Results

Epoch	Train Accuracy (%)	Validation Accuracy (%)	Validation Loss
1	33.90	52.42	1.3381
5	66.20	67.96	0.9322
10	74.27	69.61	0.9093
12	76.55	70.85	0.8814
15	79.92	71.33	0.8839

Table 8: Training and Validation Performance Summary

77.1 Curve Visualization

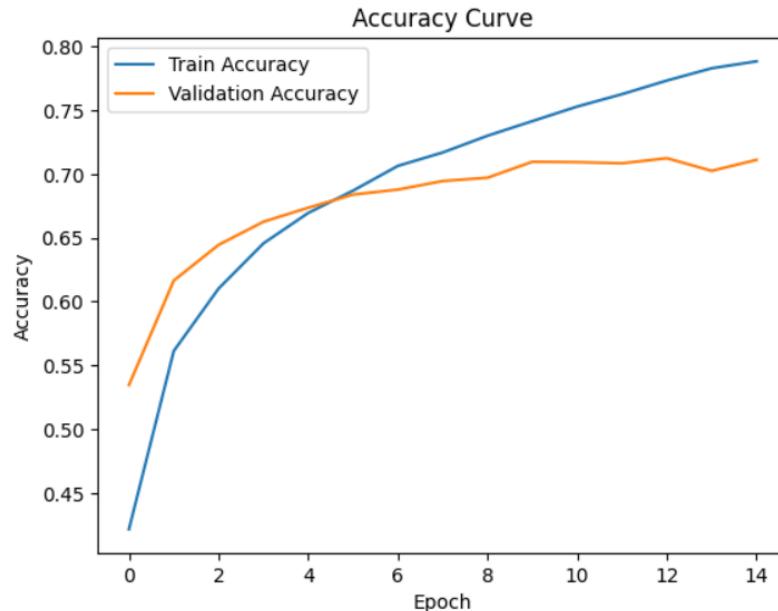


Figure 8: Training vs Validation Accuracy

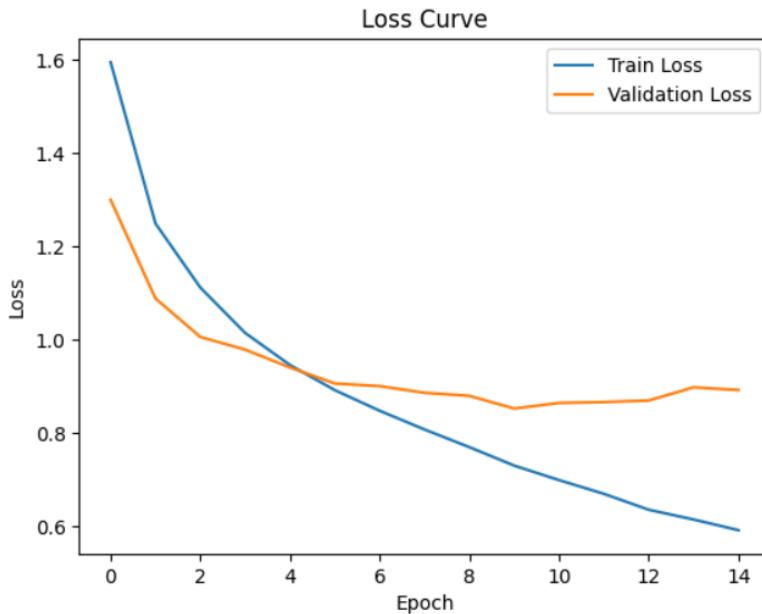


Figure 9: Training vs Validation Loss

78 Discussion

The training accuracy increased steadily from 33.9% to nearly 80%, while validation accuracy improved to approximately 71%.

The relatively small gap between training and validation accuracy indicates moderate generalization. However, slight oscillations in validation loss after epoch 8 suggest minor overfitting.

These observations imply that adding dropout, reducing learning rate, or early stopping could further improve performance.

78.1 Hyperparameter Insights

- If validation loss rises → increase regularization or dropout
- If curves are flat → increase model complexity or epochs
- If loss oscillates → reduce learning rate
- If validation plateaus → apply learning rate scheduling

79 Conclusion

Monitoring performance curves transforms hyperparameter tuning into a systematic optimization strategy.

By analyzing both accuracy and loss trends, developers can identify overfitting, underfitting, and convergence issues early. This leads to faster training, improved generalization, and more reliable CNN models.