Implement the Classification and Clustering Algorithms to Handwritten Digit Identification

Shuaibing Li, Yu-Jie Lin, Wenling Zheng, Jinwei Yin and Duong Bao Loc Phan*

(Dated: December 19, 2024)

This study explores the performance of deep learning models, including Baseline CNN, ResNet, DenseNet, RNN, and LSTM, on two tasks: CIFAR-10 image classification and ECG signal analysis. The ResNet and DenseNet models outperformed the Baseline CNN in image classification, achieving accuracies of 89.8% and 81.5%, respectively. For ECG signal analysis, the LSTM model showed improved accuracy over training epochs, while the Baseline RNN did not converge. Performance metrics such as accuracy, F1-score, and sensitivity were used to compare the models, highlighting the strengths of ResNet and LSTM for their respective tasks.

Keywords: CNN, ResNet, DenseNet, RNN, LSTM

I. INTRODUCTION

Neural networks are widely used in various image and data analysis tasks due to their ability to model complex patterns. Among them, Convolutional Neural Networks (CNNs) stand out for their specialization in image recognition tasks, as they incorporate convolutional layers to process grid-like data structures, such as images. Building upon the CNN architecture, Residual Neural Networks (ResNet) and DenseNet were introduced to address the gradient vanishing problem that arises when training deep networks with many layers. On the other hand, Recurrent Neural Networks (RNNs), with their ability to maintain hidden states, excel in sequential data tasks. The improved RNN model, known as Long Short-Term Memory (LSTM), is particularly effective in overcoming vanishing and exploding gradient issues in long-term dependency modeling.

CNNs are particularly effective for extracting spatial features from images. The core component of CNNs is the convolutional layer, which applies a set of filters to the input data to extract meaningful features. To reduce the spatial dimensions of feature maps while retaining critical information, pooling layers are typically added after convolutional layers. Although adding more layers and neurons can improve model performance, excessively deep architectures may lead to extremely small gradients during backpropagation, degrading performance. To address this, the Residual Block was introduced in ResNet. A residual block typically consists of two convolutional layers, each followed by batch normalization and an activation function. Instead of learning the complete mapping, it learns the residual mapping by directly adding the input to the output through a skip connection. This simple yet effective structure allows the model to mitigate gradient-related issues while improving training efficiency[1].

Similarly, DenseNet improves on CNNs by introducing dense connections. Unlike ResNet, where each layer connects only to its immediate predecessor, DenseNet ensures that every layer connects to all preceding layers. This promotes feature reuse and facilitates better gradient flow during training. DenseNet is built using dense blocks, where the output of each layer is concatenated with all previous layer outputs. To control network complexity and reduce redundant information, transition layers are used between dense blocks. These layers apply 1×1 convolutions to reduce feature map dimensions and perform down-sampling through average pooling[2].

In contrast to CNNs, RNNs are designed for sequential data processing. They use hidden states to capture dependencies between different time steps, making them particularly suitable for tasks such as time-series analysis. RNNs share the same weights across all time steps, reducing model complexity. At each time step t, an RNN updates its hidden state h_t based on the current input x_t and the previous hidden state h_{t-1}, as follows:

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b)$$

Here, W_x and W_h are the weight matrices for the input and hidden states, respectively, while b is the bias term, and σ is the activation function. The output y_t at time t is computed as:

$$y_t = \mathrm{softmax}(W_y h_t + b_y)$$

where W_y is the weight matrix for the output layer, and $b_y$is the output bias. The sharing of weights across time steps ensures computational efficiency.

However, traditional RNNs struggle with long-term dependencies due to vanishing and exploding gradient problems. To address these challenges, LSTMs were developed. LSTM introduces a memory cell that can store information over long periods. It uses three gates to regulate information flow: the forget gate, which determines what information to discard from the memory cell; the input gate, which decides what new information to add; and the output gate, which determines the output at each time step. This architecture allows LSTMs to effectively manage long-term dependencies and improve performance on sequential data tasks[3].

* Penn State University.

In this study, we implement two projects to explore the applications of these models. First, we use a baseline CNN model, ResNet, and DenseNet for CIFAR-10 classification to evaluate their performance on image recognition tasks. Second, we apply RNN and LSTM models to analyze electrocardiogram (ECG) signals using the MIT-BIH Arrhythmia Database. We compare the models' performance using metrics such as accuracy, F1-score, sensitivity, and specificity, providing insights into their strengths and limitations for different types of data.

II. EXPERIMENT

A. CIFAR-10 Classification

1. *Dataset and Preprocessing*

We use the images from the CIFAR-10 dataset, which consists of 60,000 32x32 color images across 10 classes. First, we preprocess our data by implementing data augmentation. In the real world, objects in images are not always perfectly aligned, and data augmentation helps simulate these variations, improving the generalization of our model. This approach allows our model to focus on the content of the image rather than its exact position.

For data augmentation, we first add a 4-pixel-wide border around the original image, increasing the size to 40x40. Then, we randomly crop a 32x32 region from the padded image. Next, we apply horizontal flipping to the image with a 50% probability. This enables the model to see slightly different versions of the same image during training. Finally, we normalize the data by shifting the mean to 0 and scaling the standard deviation to 1. All of these preprocessing steps are performed with a batch size of 128.

2. *Models*

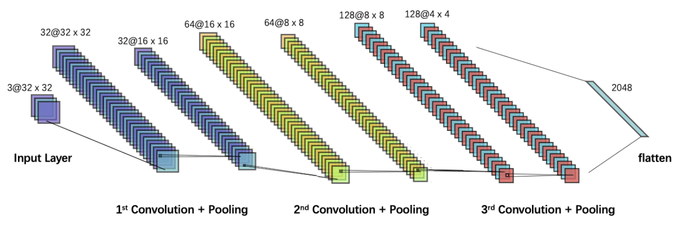


FIG. 1: Convolution layers for baseline model

a. Baseline CNN Model Our baseline CNN model consists of convolutional layers (FIG. 1) and fully connected layers. The architecture includes three convolutional blocks with similar structures. The input layer has a size of 32x32 with 3 RGB channels, followed by a convolutional layer with a 3x3 kernel, stride 1, and padding 1. Additionally, we apply a ReLU activation function and a 2x2 max-pooling layer. The second and third convolutional layers have the same kernel size, stride, and padding, and each is followed by a 2x2 max-pooling layer. After the convolutional blocks, the output is flattened from a shape of 128x4x4 into a vector, which is then connected to two fully connected layers. The first fully connected layer has 512 neurons, with ReLU activation and a 0.5 dropout rate, while the final layer consists of 10 neurons for classification.

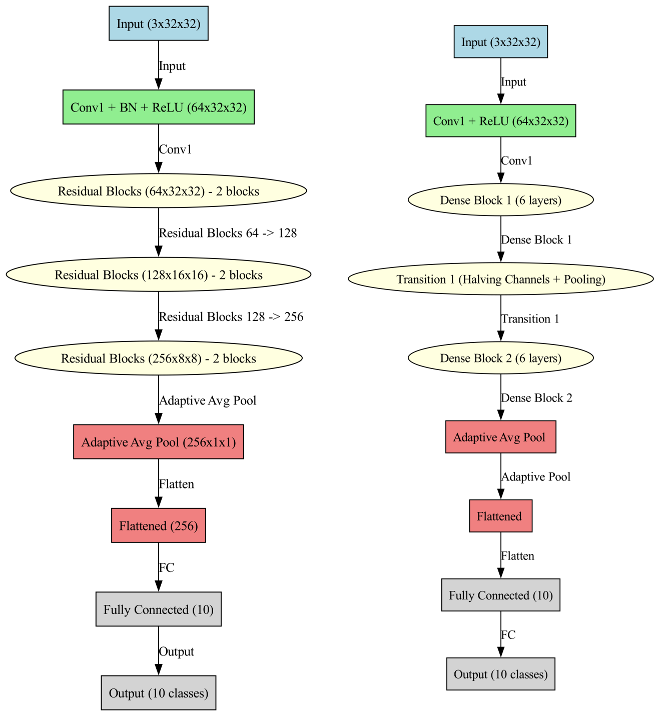


FIG. 2: Structure of ResNet (left) and DenseNet (right) Model.

b. ResNet Model The input image is of size 32x32x3 and is passed through a convolutional block. This block applies 64 filters of size 3x3, with stride 1 and padding 1, followed by a ReLU activation. The output size remains 32x32 due to the padding. Batch normalization is then applied to normalize the activations, improving model training stability. The output of the batch normalization is input into the Residual Block structure. Data passes through three levels of residual blocks, with each level containing two residual blocks. Each residual block applies two 3x3 convolutions, including batch normalization and ReLU activation. The input of each block is added to its output via a shortcut connection. If the dimensions differ (due to stride or channel changes), a 1x1 convolution is used in the shortcut:

- **Level 1:** Two residual blocks, each with 64 input and output channels, and stride 1 (no spatial reduction).

- **Level 2:** Two residual blocks with 128 input and output channels, and stride 2 in the first block (reduces spatial dimensions by half).

- **Level 3:** Two residual blocks with 256 input and output channels, and stride 2 in the first block (further reduces spatial dimensions).

Next, we add an adaptive average pooling layer, reducing the feature map size to 1x1. The flattened output is then passed through a fully connected layer to obtain the final output (FIG. 2).

c. DenseNet Model In our experiment, the image (3x32x32) is passed through the initial convolution layer. We use two Dense Block with similar stucture. Each Block processed the data through 6 dense layers and adds feature maps to the output. Then it follows the transition layer to reduce the number of feature maps by half and downsamples the spatial size. The output of Dense Blocks will input the CLobal Pooling Layer to reduce the feature map size to 1x1. Then fully connected layer is added to get the final output of given classes (FIG. 2).

3. Results and Analysis

In our experiment, we set the learning rate to 0.001, the number of epochs to 50, and the batch size to 256. We then examined how the accuracy of the training and testing datasets changed with each epoch (FIG. 3). We observed that, as expected, the accuracy of all three models improved as the number of training epochs increased. For the Baseline CNN model, the training and testing accuracies were relatively close to each other. However, for the ResNet and DenseNet models, the testing accuracy was lower than the training accuracy.

Additionally, we analyzed the confusion matrix (FIG. 4) and calculated various performance metrics, including average loss, average recall, accuracy, precision, and F1-score. Among these models, the ResNet model performed the best, achieving an accuracy of 89.8%, while the Baseline CNN model had the lowest accuracy, at 81.5%.

Metric	**CNN**	**ResNet**	**DenseNet**
Accuracy (%)	81.5	89.8	85.35
Average Loss	0.546	0.4243	0.728
Average Precision	0.8184	0.899	0.8776
Average Recall	0.815	0.898	0.8535
Average F1-Score	0.8142	0.8977	0.8572

TABLE I: Model performance across different metrics.

B. ECG Signal Processing

1. Dataset and Preprocessing

We read the ECG signals and their annotations from MIT-BIH Arrhythmia Database. We split the ECG signals into 100-length segments with no overlap and convert the labels into not-hot encoding. We implement 80/20 splitting for training and test dataset.

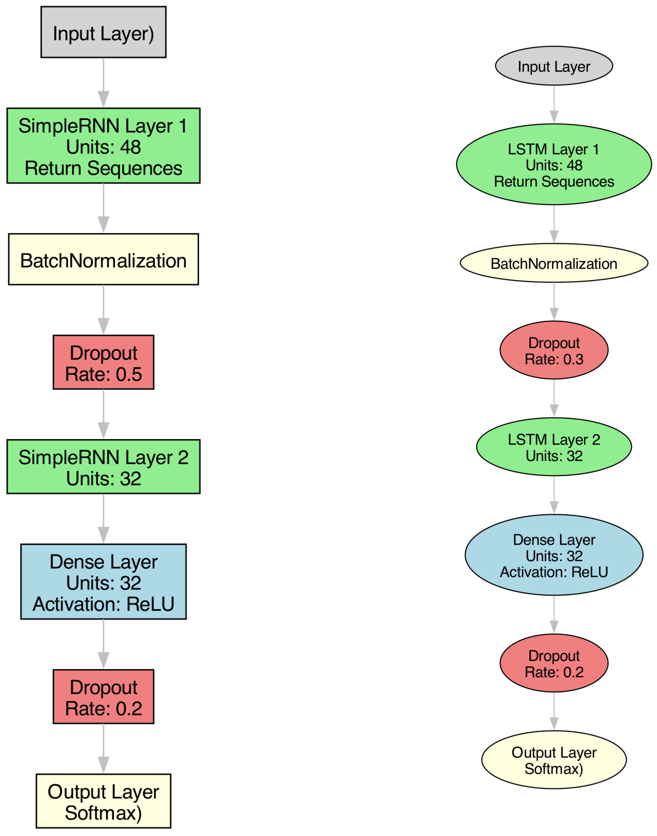
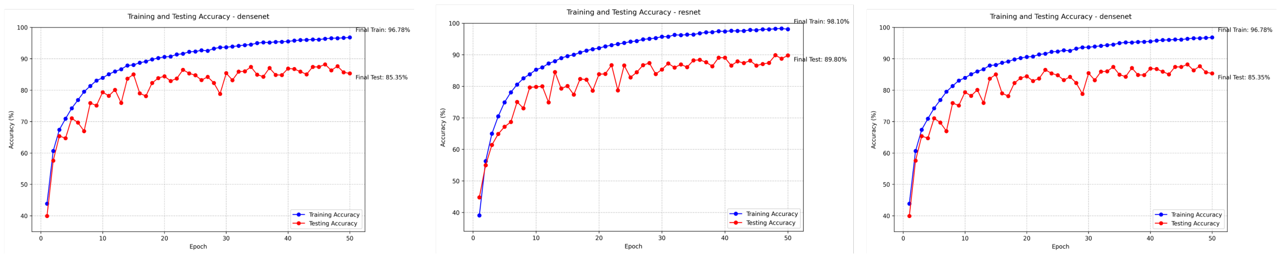


FIG. 5: Structure of RNN (left) and LSTM (right) Model.

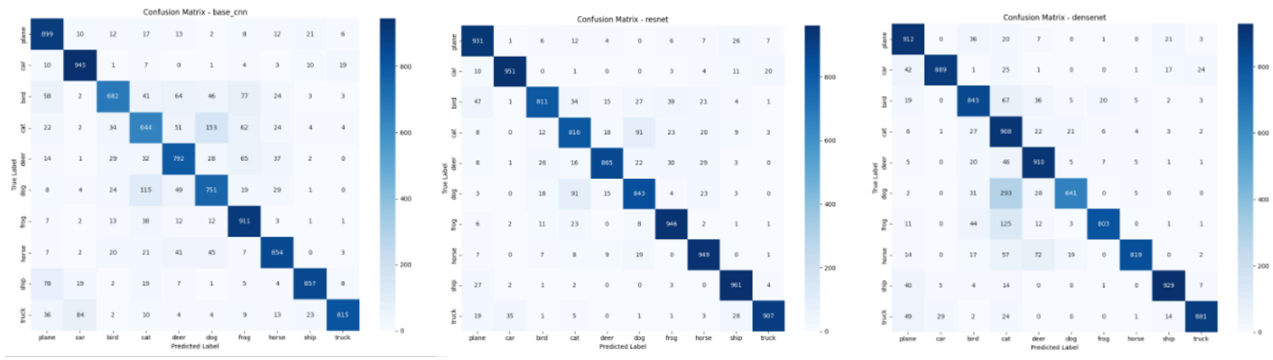
2. Models

a. Baseline RNN Model We processes sequential data using an RNN with 48 units and output the hidden state for each time step. We then applied Batch Normalization layer and reduces overfitting by dropping 50% of the neurons. Then it connects with second RNN layer, which process the sequence with 32 units and outputs only the final hidden state. Then we connected with a fully connected layer with 32 units and ReLU activation to further process the data. Next, droupping 20% of neurons and finally connect wit a fully connected layer that outputs the class probabilities (FIG. 5).



(a) CNN (b) ResNet (c) DenseNet

FIG. 3: Training and testing accuracy for CIFAR-10 Classification.



(a) CNN (b) ResNet (c) DenseNet

FIG. 4: Confusion Matrix for CIFAR-10 Classification.

b. LSTM Model The model begins with an Input Layer, which receives sequential data where each time step contains multiple features. The first layer is an LSTM Layer with 48 units, designed to process the time series data and return outputs for all time steps. This is essential for capturing temporal dependencies in the sequence. Following this, a BatchNormalization Layer normalizes the outputs from the first LSTM layer, ensuring the data has a consistent mean and variance. This step improves the stability and efficiency of training. To reduce overfitting, a Dropout Layer is applied next, randomly dropping 30% of neurons during training. Afterward, a second LSTM Layer with 32 units further processes the data, but it returns only the final output of the sequence. This focuses the model on the most important features learned from the previous LSTM layer. The model then moves to a Dense Layer with 32 units and a ReLU activation function, enabling the network to learn complex, non-linear feature representations from the LSTM layer outputs. Another Dropout Layer is introduced, this time dropping 20% of the neurons to further mitigate the risk of overfitting. Finally, the Output Layer uses the Softmax activation function, converting the final outputs into probabilities corresponding to each class in the classification task. This enables the model to make predictions based on these probabilities (FIG. 5).

3. Results and Analysis

In our experiment, we set the learning rate to 0.005, the number of epochs to 50, and the batch size to 256. We then analyzed how the accuracy of the training and testing datasets evolved over the course of the epochs. As expected, the LSTM model showed an improvement in accuracy with each additional training epoch. However, the accuracy of the Baseline RNN model did not converge.

We also calculated various performance metrics, including accuracy, F1-score, sensitivity, and specificity. Among all the models, ResNet outperformed the Baseline RNN, achieving an accuracy of 94.1%, while the Baseline CNN model had an accuracy of 69%.

Metric	**RNN**	**LSTM**
Accuracy	0.6909	0.9419
F1 Score	0.7988	0.9572
Sensitivity	0.6219	0.9144
Specificity	0.6940	0.9431

TABLE II: Model performance comparison between RNN and LSTM.

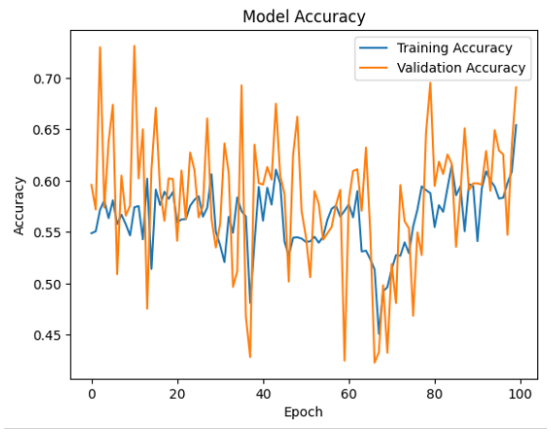


FIG. 6: Accuracy fo RNN

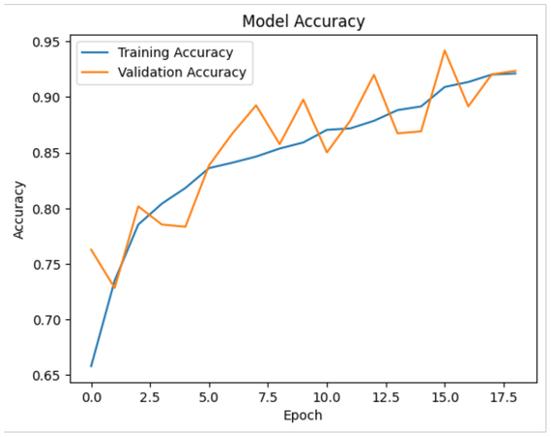


FIG. 7: Accuracy for LSTM

III. CONCLUSION

In this study, we explored the application of various deep learning models, including Baseline CNN, ResNet, DenseNet, RNN, and LSTM, in two distinct tasks: image classification on the CIFAR-10 dataset and ECG signal analysis using the MIT-BIH Arrhythmia Database. Our experiments demonstrated that ResNet and DenseNet significantly outperformed the Baseline CNN model in terms of image classification accuracy. Specifically, ResNet achieved an accuracy of 89.8

In the second part of the experiment, we applied RNN and LSTM models to analyze ECG signals. While the LSTM model demonstrated steady improvements in accuracy over training epochs, the Baseline RNN model struggled to converge, reflecting its limitations for sequential data analysis. The LSTM model, with its ability to capture long-term dependencies, showed superior performance, achieving a higher accuracy compared to the Baseline RNN.

Throughout the study, we evaluated the models using various performance metrics, including accuracy, F1-score, sensitivity, specificity, and recall. The results suggest that for image recognition tasks, ResNet and DenseNet are highly effective, while LSTM excels in sequential data tasks like ECG signal analysis. The findings highlight the strengths of each model for their respective tasks, offering valuable insights for future applications in image recognition and time-series analysis.

In conclusion, our experiments demonstrate that deep learning models such as ResNet, DenseNet, and LSTM have strong potential for real-world applications in both image classification and time-series data analysis. Further improvements can be made to address the convergence issues of RNNs and enhance the performance of these models in complex tasks.

ACKNOWLEDGMENTS

We sincerely appreciate Dr. Chunyan Li for her guidance and support throughout this study. We would like to express our gratitude for her insightful classes and invaluable assistance.

[1] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 770–778.

[2] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, Densely connected convolutional networks, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017) pp. 4700–4708.

[3] A. Graves and A. Graves, Long short-term memory, Supervised sequence labelling with recurrent neural networks , 37 (2012).