INT305 ASSESSMENT 2: Handwritten Digits Recognition

Mingyuan Li, *2145618*

***Abstract*—Handwritten digit recognition is a critical task in computer vision, with diverse applications including postal code recognition and digital document analysis. This study focuses on handwritten digit recognition on the MNIST dataset using CNN and explores the potential of a lightweight model, GECCO, for improving accuracy while reducing model size. The study proposes three modifications to enhance GECCO's performance: dropout regularization after ReLU, a batch-based attention mechanism for feature representation, and L2 regularization in the fully connected layer. These modifications are compared with baseline models, including CNN, SpinalNet, ResNet, and Efficient-CapsNet. Results show that the modified GECCO outperforms traditional CNN models, achieving higher accuracy, lower error rates, and reduced parameter size.**

***Index Terms*—Convolutional Neural Network, MNIST, Classification**

I. Introduction

HANDWRITTEN digit recognition is a fundamental problem in computer vision and machine learning, with widespread applications ranging from postal code identification to digital document processing [1]. The MNIST dataset [2], a benchmark for this task, consists of 70,000 grayscale images of handwritten digits (0–9), standardized to a resolution of 1×28×28 pixels. This dataset provides a rich and well-structured resource for evaluating and comparing the performance of various machine learning or deep learning models in accurately identifying digits, making it a cornerstone for research and experimentation in this domain [3].

Convolutional Neural Network (CNN) has emerged as the dominant approach for handwritten digit recognition, significantly outperforming traditional machine learning methods [4] [5]. Early CNN architectures, such as LeNet-5 [6], laid the foundation for digit recognition by using multiple convolutional layers followed by pooling and fully connected layers. More recently, deeper and more sophisticated architectures, such as AlexNet [7], VGGNet [8], and ResNet [9], have been adapted and optimized for the MNIST dataset, achieving state-of-the-art (SOTA) performance. These modern CNNs typically incorporate advanced techniques like dropout, batch normalization, and data augmentation to improve generalization and prevent overfitting [10]. The success of CNN in handwritten digit recognition is primarily due to their ability to automatically capture relevant spatial features from the images, allowing them to achieve high accuracy without manual feature engineering [11].

Despite their success, CNN in handwritten digit recognition faces several challenges. One issue is the trade-off between model complexity and performance, where deeper networks often require significantly more parameters and computational resources, making them less efficient. For example, large architectures like ResNet can achieve high accuracy but are computationally expensive and prone to overfitting, especially when dealing with relatively simple datasets like MNIST. There is a growing demand for methods that can achieve higher accuracy while maintaining smaller model sizes [12] [13], such as techniques that involve pruning [14], knowledge distillation [15], or lightweight architectures like MobileNet [16], which aim to reduce parameter count and improve computational efficiency without compromising accuracy.

To achieve higher accuracy with smaller model parameters, it is essential to consider lightweight solutions that can deliver high performance and robustness, even with shallow models. In this context, selecting Fein-Ashley's model GECCO [17] as a starting point is particularly meaningful. This model offers a highly efficient, lightweight architecture that vectorizes an image in a fully connected manner and subsequently inputs the resultant vector into a single-layer graph convolutional network (GCN). GECCO has demonstrated SOTA performance on several datasets, including the SAR ATR dataset [18], the MSTAR dataset [19], and the CXR dataset [20]. However, despite these successes, the model's performance on the MNIST dataset is less impressive, revealing a notable gap in its effectiveness. Therefore, a clear research gap exists: developing methods to address the limitations of GECCO when applied to MNIST to enhance its performance on this particular dataset.

In this study, a basic CNN model with two convolutional layers was selected as the baseline to provide a more comprehensive and convincing comparison of experimental results. To enhance the performance of the GECCO model, three effective modifications were proposed and implemented: (1) introducing dropout regularization immediately after the ReLU activation function; (2) incorporating a batch-based attention mechanism to improve feature weighting and representation; and (3) applying L2 regularization in the fully connected layer to further constrain the model parameters and prevent overfitting. This study compares five models with the modified GECCO: CNN [21], GECCO [17], SpinalNet [22], ResNet [9], and Efficient-CapsNet [23]. The analysis focuses on the final accuracy and the model parameter sizes. Eventually, the proposed method achieves higher accuracy, lower error rates, and reduced model parameter size when evaluated on the MNIST dataset, demonstrating its effectiveness.

This report is organized as follows: Section II outlines the two main aspects of the CNN architecture: the utilization of

Manuscript submitted Dec. 11, 2024

convolutional kernels for feature extraction and the application of loss functions to guide the learning process. Section III outlines the experimental setup, including details of the dataset, the hardware environment, and the framework used. Section IV introduces the baseline considered, followed by the proposed improved approach in Section V. The experimental results, evaluations and comparison are presented in Section VI. Finally, Section VII concludes the paper with a summary and discussion of future work.

II. Problem Specifications

This section provides a detailed description and analysis of the convolutional kernel and the loss function utilized within the framework, supported by mathematical equations.

A. Convolutional kernel

Convolutional kernels are fundamental to the operation of CNN, serving as the primary mechanism for feature extraction. These kernels traverse input images to identify and learn patterns such as edges, textures, and more complex representations at higher layers [24]. By convolving over the input, kernels produce feature maps that encapsulate critical spatial information for subsequent processing in the network. Below, it is meaningful to provide a detailed discussion of convolutional kernels in the context of a specific framework.

To delve into the intricacies of the convolutional kernel, it is imperative to begin with an understanding of the convolution operation itself. The convolution operation can be described universally using the following mathematical formulation:

$$x_{ij}^{n} = \sum_{c}\sum_{a=0}^{k-1}\sum_{b=0}^{k-1} \omega_{ab}^{cn} y_{(i+a)(j+b)}^{c} \quad (1)$$

In this equation:

- x_{ij}^{n} is the output value at coordinates (i, j) in the n-th feature map.
- ω_{ab}^{cn} represents the weight at position (a, b) in the kernel corresponding to the n-th feature map and c-th input channel.
- $y_{(i+a)(j+b)}^{c}$ is the pixel value at position $(i + a, j + b)$ in the c-th input channel.
- k is the size of the convolutional kernel (e.g., $k = 3$ for a 3x3 kernel).

The summation over c accounts for the contribution from all input channels when performing the convolution, emphasizing the kernel's ability to aggregate multi-channel information. The kernel slides over the input image (or feature map) in steps defined by the stride, applying this operation at each valid location.

1) Example of Convolution in Practice: To illustrate the convolution process, consider a convolutional kernel of size 3×3 applied to a 6×6 input image, producing a feature map. This operation can be summarized in Equation 2:

$$\begin{bmatrix} x_{11}^c & x_{12}^c & x_{13}^c & x_{14}^c & x_{15}^c & x_{16}^c \\ x_{21}^c & x_{22}^c & x_{23}^c & x_{24}^c & x_{25}^c & x_{26}^c \\ x_{31}^c & x_{32}^c & x_{33}^c & x_{34}^c & x_{35}^c & x_{36}^c \\ x_{41}^c & x_{42}^c & x_{43}^c & x_{44}^c & x_{45}^c & x_{46}^c \\ x_{51}^c & x_{52}^c & x_{53}^c & x_{54}^c & x_{55}^c & x_{56}^c \\ x_{61}^c & x_{62}^c & x_{63}^c & x_{64}^c & x_{65}^c & x_{66}^c \end{bmatrix} \cdot \begin{bmatrix} w_{11}^{cn} & w_{12}^{cn} & w_{13}^{cn} \\ w_{21}^{cn} & w_{22}^{cn} & w_{23}^{cn} \\ w_{31}^{cn} & w_{32}^{cn} & w_{33}^{cn} \end{bmatrix}$$
$$= \begin{bmatrix} z_{11}^n & z_{12}^n & z_{13}^n & z_{14}^n \\ z_{21}^n & z_{22}^n & z_{23}^n & z_{24}^n \\ z_{31}^n & z_{32}^n & z_{33}^n & z_{34}^n \\ z_{41}^n & z_{42}^n & z_{43}^n & z_{44}^n \end{bmatrix} \quad (2)$$

In this example:

- Input Image (x): The 6×6 matrix represents the pixel values in a single channel (c).
- Convolutional Kernel (w): The 3×3 weights correspond to the n-th kernel for the same channel.
- Feature Map (z): The resulting 4×4 matrix is the feature map generated by the convolution.

This example illustrates how a convolutional kernel processes a larger input image, producing a feature map with reduced spatial dimensions. The reduction is determined by the kernel size, which defines the receptive field; the stride, which sets the step size for sliding; and padding, which controls edge preservation. These parameters work together to balance computational efficiency with spatial information retention.

2) Importance of Convolutional Kernels in CNN: Convolutional kernels are not static but learnable components optimized during training. Each kernel learns to detect specific patterns, ranging from simple edges at lower layers to complex features at deeper layers. By stacking multiple layers of kernels, CNN achieves their ability to model intricate data relationships, making them effective in applications such as image recognition, object detection, and semantic segmentation [25] [26].

B. Loss function

The loss function is a critical component in the training of CNN, as it quantifies the difference between predicted outcomes and true labels [27]. This feedback guides the optimization process, allowing the model to adjust its parameters and improve its performance over time [28]. Loss functions can vary based on the task, with cross-entropy loss being widely used for classification problems due to its effectiveness in measuring probability-based outputs.

In this framework, the cross-entropy loss function is utilized, defined as follows:

$$L = -\sum_{i=1}^{N} y^{(i)} \cdot \log \hat{y}^{(i)} \quad (4)$$

Where:

- N represents the number of classes in the dataset.
- $y^{(i)}$ is the true label for class i ($y^{(i)} = 1$ if the data belongs to class i, and 0 otherwise).
- $\hat{y}^{(i)}$ is the predicted confidence (probability) for class i.

This equation calculates the weighted negative log probability of the true class, emphasizing incorrect predictions while rewarding accurate ones.

1) Implementation in PyTorch: In PyTorch, the **cross-entropy loss function** is implemented as a combination of two key components: **log-softmax** and **negative log-likelihood loss (nll_loss)**. This modular design ensures computational efficiency and numerical stability. Here's how each component contributes:

Log-Softmax Transformation. The log-softmax function applies a softmax operation followed by a logarithm. This step converts raw model outputs (logits) into log probabilities, ensuring better handling of numerical precision when dealing with very large or small values. The log-softmax function is defined as:

$$\text{log-softmax}(y) = \log \frac{e^{y_k}}{\sum_j e^{y_j}} \quad (5)$$

- y: The raw prediction scores (logits) from the model.
- y_k: The k-th score, where e^{y_k} represents the exponential of the score.
- $\sum_j e^{y_j}$: The sum of exponentials across all scores, acting as a normalizing term.

By taking the logarithm of the softmax output, this operation simplifies the computation of the loss while maintaining numerical stability.

Negative Log-Likelihood Loss (nll_loss). Following the log-softmax transformation, the negative log-likelihood loss calculates the final loss by penalizing the log probability assigned to the true class. It is expressed as:

$$\text{nll\_loss}(x, t) = -\log(x_t) \quad (6)$$

- x: The log probabilities computed by log-softmax.
- t: The index of the true class label.

2) Importance of Loss function in CNN: The loss function guides CNN by measuring the difference between predictions and true labels, enabling the model to learn and improve through optimization [29]. It ensures effective pattern recognition and minimizes errors, with cross-entropy loss being ideal for classification tasks.

III. Experiments Setup

A. Dataset

MNIST [2]: it consists of handwritten digits, with the input feature space formatted as $28 \times 28 \times 1$. The training set includes 60,000 data points, while the test set contains 10,000 examples.

B. Hardware

All experiments described in this paper were conducted using both CPU and GPU. The implementation of all involved models is flexible and can be executed using a CPU, GPU, or a combination of both. Specifically, for the computational tasks, this study utilized Google Colab's Nvidia T4 GPU [30], a Tensor Core GPU with 16 GB GDDR6 memory, and Kaggle's Nvidia P100 GPU [31], equipped with 16 GB HBM2 memory. The combination of these GPUs and platforms ensured efficient performance for deep learning workloads and parallel task processing capabilities for training and evaluation.

C. Framework

The implementation was performed in Python using Nvidia's Compute Unified Device Architecture (CUDA), a parallel computing framework and API designed for high-performance computations. For building and training the neural networks, we employed the TensorFlow [32] and Pytorch [33] libraries, ensuring compatibility and efficiency on the Colab platform.

IV. Baseline

In this study, it is reasonable to consider Mohamed's research [21] as a baseline because it used a basic CNN as its backbone. It is composed of two sequential convolutional blocks and a fully connected classification head. Each convolutional block includes two convolutional layers with 3×3 kernels, followed by batch normalization, ReLU activation, and a max-pooling layer with a stride of 2. The first block processes input data with 1 channel, producing 32 feature maps, while the second block expands the feature representation to 64 channels. After the convolutional blocks, the output is flattened and passed to a fully connected network comprising three linear layers with hidden dimensions of 1152 and 576, interleaved with ReLU activations and dropout layers to prevent overfitting. The final layer outputs predictions for 10 classes. This design balances feature extraction and computational efficiency, making it suitable for a variety of vision tasks. The intuitive visualization of the CNN structure is presented in Fig. 1.

The model is trained over 25 epochs using a supervised learning approach with separate training and validation phases in each epoch. The training process employs the cross-entropy loss as the criterion, which is minimized via backpropagation using an optimizer. A learning rate scheduler adjusts the learning rate dynamically after each epoch to optimize convergence. During the training phase, gradients are computed and weights are updated using the optimizer, while in the validation phase, the model operates in evaluation mode without gradient updates. The performance is tracked using two metrics: average loss and classification accuracy, computed across the dataset sizes for each phase. At the end of each epoch, the model's weights are deep-copied if the validation accuracy surpasses the current best. The training process, including accuracy and loss, is shown in Fig. 2. The confusion matrix of the training results is presented in Fig. 3.

Additionally, this study compiled a series of classification examples to evaluate the CNN model's performance. These include instances of misclassified digits, as shown in Fig. 4, and correctly recognized digits in Fig. 5.

V. Method

A. GECCO Structure

The GECCO model (Fig. 6) begins by vectorizing a batch of images, transforming them into a 2D tensor for efficient

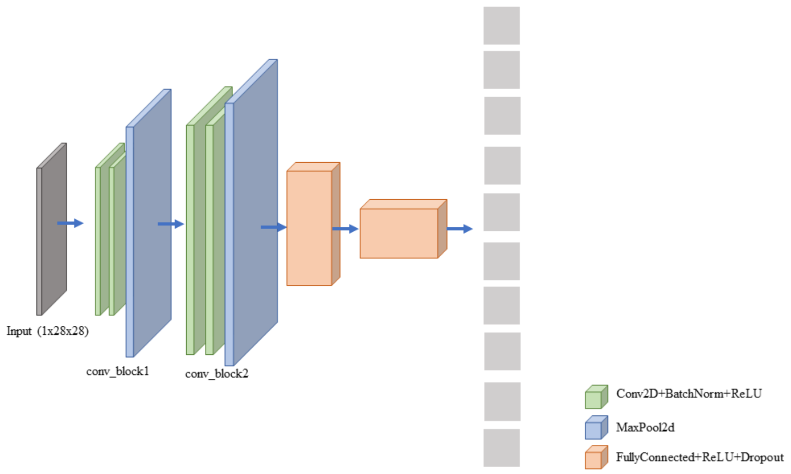


Fig. 1: Baseline CNN Structure

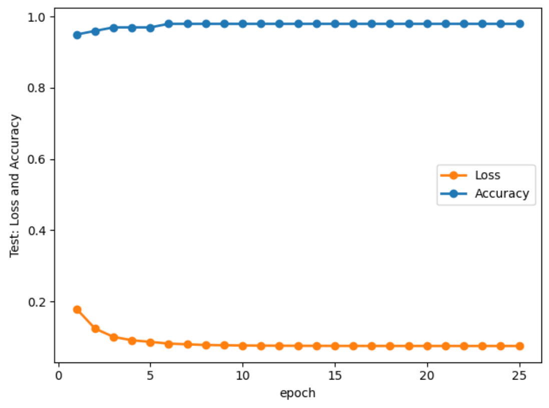


Fig. 2: Baseline Training Curve: Loss and Accuracy

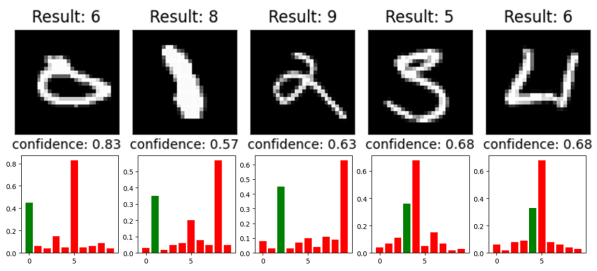
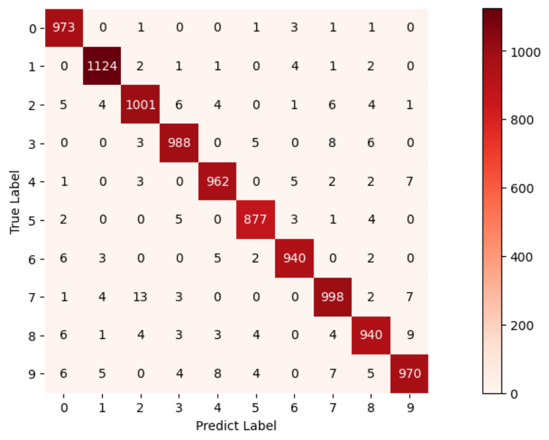


Fig. 4: Example of misclassified digits in the baseline results. Green bars indicate the correct labels, with their height corresponding to the capsule length.



True \ Predict	0	1	2	3	4	5	6	7	8	9
0	973	0	1	0	0	1	3	1	1	0
1	0	1124	2	1	1	0	4	1	2	0
2	5	4	1001	6	4	0	1	6	4	1
3	0	0	3	988	0	5	0	8	6	0
4	1	0	3	0	962	0	5	2	2	7
5	2	0	0	5	0	877	3	1	4	0
6	6	3	0	0	5	2	940	0	2	0
7	1	4	13	3	0	0	0	998	2	7
8	6	1	4	3	3	4	0	4	940	9
9	6	5	0	4	8	4	0	7	5	970

Fig. 3: Baseline Confusion Matrix

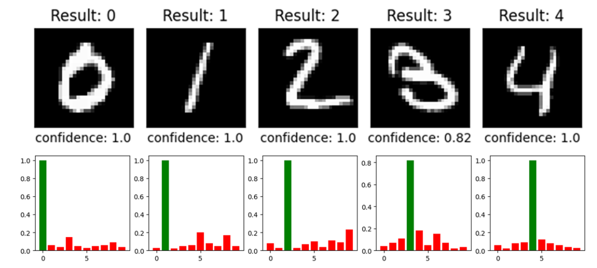


Fig. 5: Example of correct classified digits in the baseline results. Green bars indicate the correct labels, with their height corresponding to the capsule length.

processing. Each image is flattened from a 3D tensor $X \in \mathbb{R}^{B\times H\times W}$ into a vector $X_1 \in \mathbb{R}^{B\times(H\cdot W)}$, allowing the model to bypass traditional convolutional layers, reducing computational complexity. This vectorized representation is then passed through a fully connected layer, followed by a ReLU activation to produce X_2. The output is then used to construct a graph, where each image in the batch is treated as a node

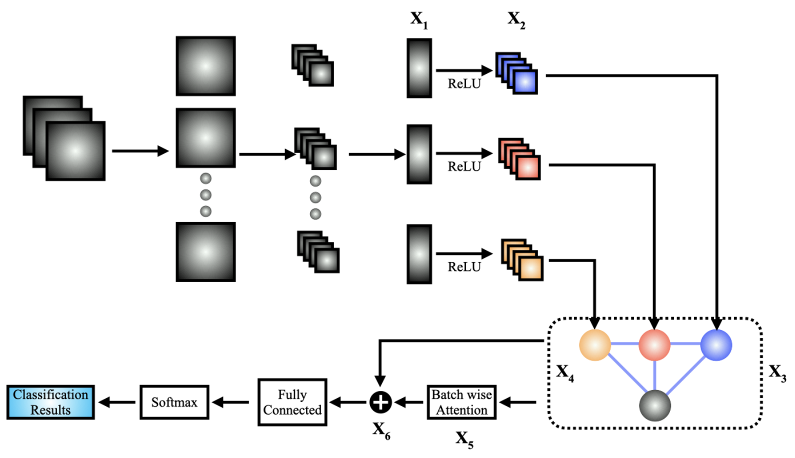


Fig. 6: Original GECCO model structure

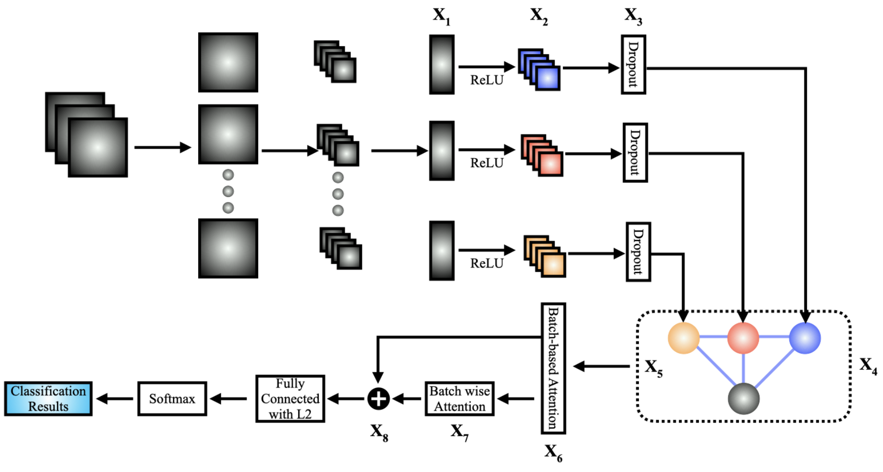


Fig. 7: My method: Improved GECCO model structure

with features. A GCN X_3 is applied to capture relationships between images in the batch, updating node representations based on aggregated neighbor features. The resulting feature matrix X_4 is processed through GCN max-pooling, completing the graph convolutional operation with X_5. Finally, it introduces a residual connection, $X_6 = X_5 + X_4$, which stabilizes and accelerates learning. This residual is then passed through a fully connected layer and a softmax function for classification, resulting in final classification.

B. *Optimization 1: Introducing Dropout Regularization*

As shown in Fig. 7, the first optimization involved integrating dropout regularization X_3 immediately after the ReLU activation function in the GECCO model. Dropout randomly deactivates neurons during training, reducing overfitting by preventing co-adaptation of features. This enhancement led to a significant improvement in the model's generalization ability, as evidenced by a little increase in accuracy on the MNIST dataset. Additionally, the dropout mechanism provided better robustness in handling noisy or varied input data.

C. *Optimization 2: Adding Batch-Based Attention Mechanism*

The introduction of dropout regularization successfully reduced the model's parameters but did not result in a substantial improvement in accuracy or error rate. To further enhance performance, a batch-based attention mechanism X_6 was

TABLE I: Hyperparameters Used in Training

Model	Optimizer	Batch Size	Learning Rate	Epoch
Baseline [21]	ReLU	32	0.1	25
GECCO [17]	Sigmoid and ReLU	64	0.1	50
Enhanced GECCO	Sigmoid and ReLU	64	0.1	50
SpinalNet [22]	ReLU	64	0.1	20
ResNet [9]	ReLU	64	0.1	25
Efficient-CapsNet [23]	ReLU	64	0.1	100

incorporated into the model [34]. This mechanism dynamically adjusted the importance of features across different batches, allowing the network to prioritize the most relevant features while suppressing less significant ones. By improving the discriminative power of the model, this modification significantly enhanced feature representation and addressed a key limitation of the original GECCO structure, which treated all features uniformly.

D. Optimization 3: Incorporating L2 Regularization

While the attention mechanism improved feature weighting, it introduced a potential risk of overfitting due to increased model complexity. To address this, L2 regularization was applied to the fully connected layer, penalizing large weights and encouraging simpler, more generalized models. This final modification resolved the overfitting issue and further reduced the model size, achieving a balanced trade-off between accuracy and efficiency.

With these three optimizations, the enhanced GECCO model achieved higher accuracy, a lower error rate, and reduced parameter size on the MNIST dataset.

VI. Experiment and Evaluation

This section presents the experimental results and evaluations for the proposed models, including GECCO, Enhanced GECCO, and the variants SpinalNet, ResNet, and Efficient-CapsNet. The results are analyzed in terms of their performance on the MNIST classification task, with a focus on quantitative metrics. The evaluation primarily considers the training dynamics, visualized through loss/accuracy curves, and classification performance, summarized via confusion matrices.

A. GECCO

To replicate the results of the original study, this experiment uses the same hyperparameters as the GECCO [17], as shown in Table I. Fig.8 presents the training loss and accuracy curves over the course of the training process. As seen in the figure, the training loss decreases steadily, and it reaches a stable value after 35, suggesting that the model has converged. Additionally, the training accuracy increases consistently, with the accuracy plateauing at approximately 99.02% after epoch 37. Noteworthily, the significant spike in loss of the epoch 5 may be attributed to the model encountering particularly challenging data points during this epoch.

Furthermore, Fig.9 shows the confusion matrix for the GECCO's predictions on the test set. The matrix represents the number of samples classified correctly (on the diagonal) and incorrectly (off-diagonal). Overall, the model performs well across the majority of digit classes, as evidenced by the high number of correct classifications along the diagonal. The number of parameters in the model is 50800, which contributes to its capacity to learn the complex patterns in the dataset. The final error rate on the test set is 1.95, and some misclassifications can be observed between digits that are visually similar, such as 3 and 5. These results suggest that while the model performs well overall, there are certain digit pairs that remain challenging to distinguish.

B. Enhanced GECCO

To achieve better model performance, this experiment tested the enhanced GECCO, which incorporates the proposed modifications, while maintaining the same hyperparameters as the original GECCO model, as shown in Table I. The enhanced

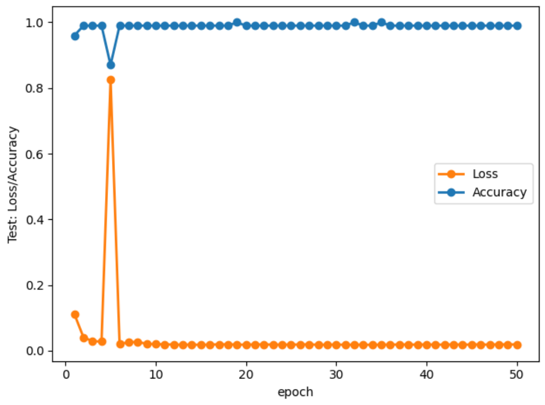


Fig. 8: GECCO Training Curve: Loss and Accuracy

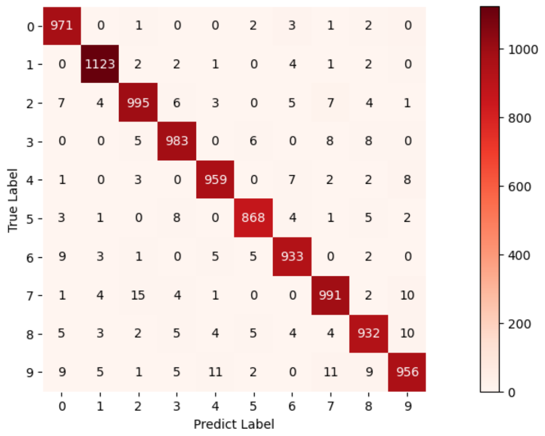


Fig. 9: GECCO Confusion Matrix

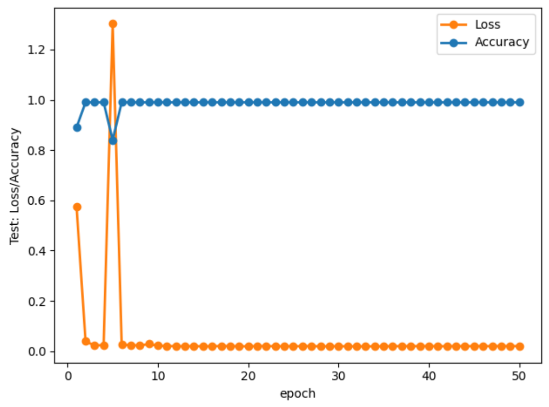


Fig. 10: Enhanced GECCO Training Curve: Loss and Accuracy

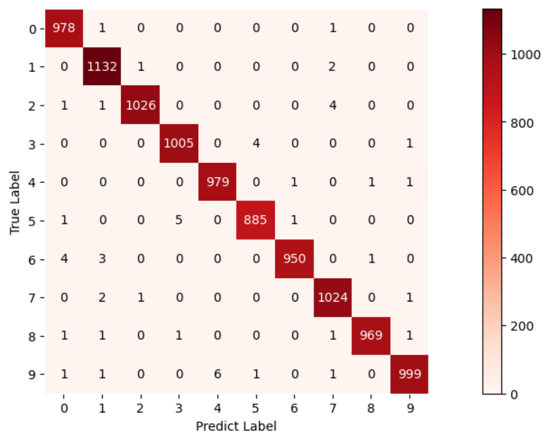


Fig. 11: Enhanced GECCO Confusion Matrix

version demonstrates superior performance compared to the original GECCO, with a smaller model size and reduced error percentage. Specifically, the model's error rate is reduced by 0.2%, achieving a test accuracy of 99.42%, as shown in the training curve Fig.10. This reduction in error percentage highlights the effectiveness of the proposed modifications in improving classification accuracy while maintaining computational efficiency. The smaller model size ensures faster inference times, making the enhanced GECCO method not only more accurate but also more suitable for resource-constrained environments. Still, there is a significant spike in loss, which may be attributed to the model encountering particularly challenging data points during this epoch.

C. SpinalNet

For SpinalNet [22], this study attempts to fine-tune the model using the hyperparameters shown in Table I. These values were selected after several rounds of fine-tuning to identify the best-performing hyperparameters for optimal convergence and accuracy on the dataset. Fig.12 shows the training and validation loss curves, where the model exhibits smooth convergence, with the validation accuracy reaching 99.39% after 12 epochs. The confusion matrix in Fig.13 highlights that most digits are correctly classified.

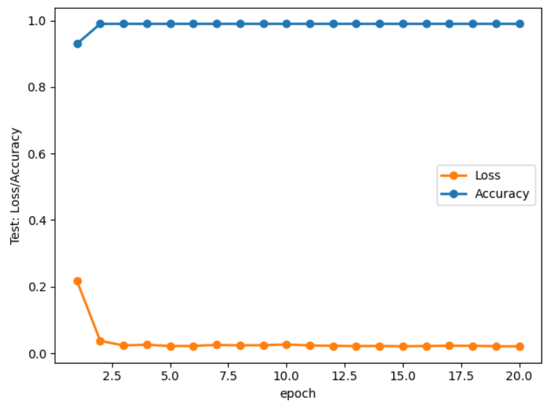


Fig. 12: SpinalNet Training Curve: Loss and Accuracy

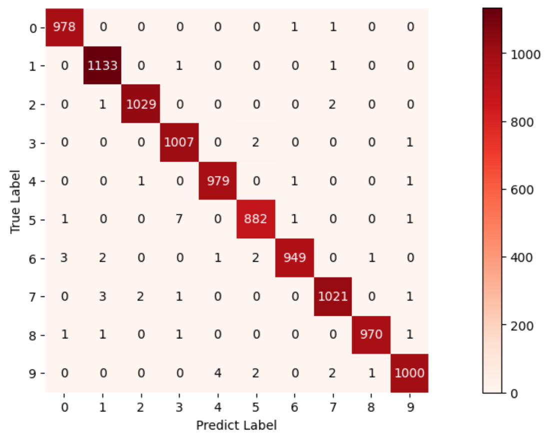


Fig. 13: SpinalNet Confusion Matrix

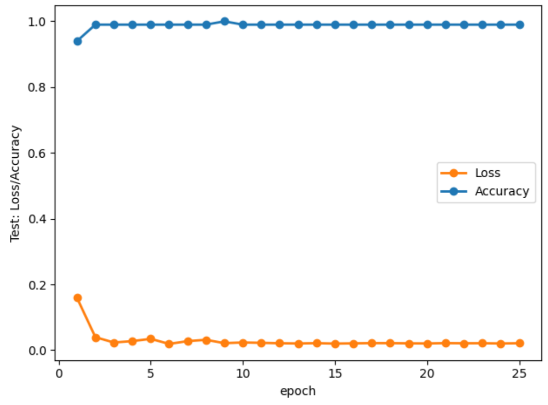


Fig. 14: ResNet Training Curve: Loss and Accuracy

D. ResNet

For ResNet [9], the model is trained with the hyperparameters outlined in Table I, which were optimized through grid search. Fig.14 displays the loss and accuracy curves, indicating a consistent reduction in loss and a steady increase in accuracy, with the model reaching 99.41% validation accuracy by epoch 11. The confusion matrix in Fig.15 further confirms good performance.

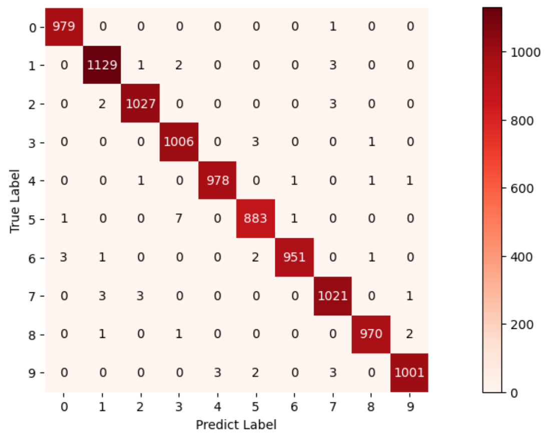


Fig. 15: ResNet Confusion Matrix

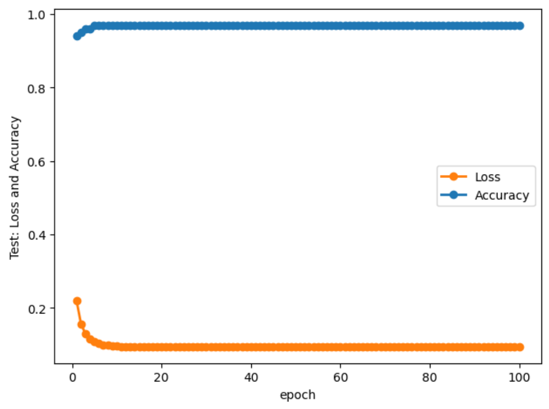


Fig. 16: Efficient-CapsNet Training Curve: Loss and Accuracy

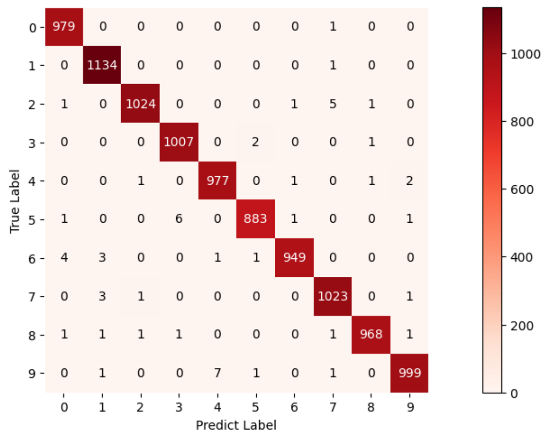


Fig. 17: Efficient-CapsNet Confusion Matrix

E. Efficient-CapsNet

In Efficient-CapsNet [23], I explore a different set of hyperparameters, as shown in Table I, fine-tuned through a combination of random search and cross-validation. Fig.16 illustrates the training and validation curves, where it shows a validation accuracy of 99.34% at epoch 45. The confusion matrix in Fig.17 indicates that while most digits are accurately predicted.

F. Overall Comparative

TABLE II: Performance of Models

Model	Average Accuracy	Best Accuracy	Parameters
Baseline [21]	97.45	97.68	149418
GECCO [17]	99.32	99.10	50800
Enhanced GECCO	99.54	99.59	49900
SpinalNet [22]	99.65	99.72	3646000
ResNet [9]	99.41	99.48	46833226
Efficient-CapsNet [23]	99.45	99.55	161000

In conclusion, this study evaluates and compares the performance of the mentioned models based on accuracy and complexity. As shown in Table II, the models' classification average and best accuracy on the MNIST dataset are reported. While accuracy remains a primary factor in assessing model performance, it is equally important to consider model complexity, which directly influences computational cost and efficiency. Table II also summarizes the model complexity in terms of the number of parameters and model size. It can be observed that the improved GECCO maintains a significantly low number of model parameters while achieving substantial performance improvements, approaching the SOTA.

VII. Conclusion

In conclusion, this study demonstrates that the enhanced GECCO model outperforms traditional CNN-based approaches for handwritten digit recognition on the MNIST dataset, achieving higher accuracy, reduced error rates, and smaller model sizes. The proposed modifications—dropout regularization, a batch-based attention mechanism, and L2 regularization—prove effective in enhancing the performance of GECCO. These findings underscore the potential of lightweight models in achieving SOTA performance with reduced computational complexity. More broadly, the success of such modifications suggests promising avenues for optimizing neural network architectures for resource-constrained environments, which could be applied to other tasks in computer vision and machine learning.

Future work could focus on applying the enhanced GECCO model to more complex datasets, such as CIFAR-10 or ImageNet, to evaluate its performance on more diverse and challenging benchmarks. Additionally, the model could be extended to tackle advanced computer vision tasks, such as object detection and image segmentation, where its lightweight design and efficiency could offer significant advantages in both accuracy and computational cost.

Acknowledgments

Gratitude is extended to Google Colab and Kaggle for providing free cloud computing services for this project.

Appendix

Below are the modified codes implemented in the enhanced GECCO model:

Listing 1: Enhanced GECCO Model Class

```
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = Att(IMG_SIZE*IMG_SIZE,
            featurelength)
        self.conv2 = ResGatedGraphConv(
            featurelength, featurelength)

        self.fc1 = nn.Linear(featurelength //
            2, OUT)
        self.bn = nn.BatchNorm1d(
            featurelength)

        self.dropout = nn.Dropout(0.15)
        self.pool = nn.MaxPool1d(2)

        # Adding batch-based attention
            mechanism
        self.attention_fc = nn.Linear(
            featurelength, featurelength)
        self.attention_softmax = nn.Softmax(
            dim=1)

    def forward(self, x, train):
        batch_size = x.shape[0]

        x_1 = torch.reshape(x, [batch_size,
            IMG_SIZE*IMG_SIZE])

        edge_index = torch.tensor([[i for i
            in range(batch_size)] for j in
            range(batch_size)]).to(device)
        edge_index = edge_index.reshape(1,
            -1)
        edge_index = torch.cat((edge_index,
            torch.flip(edge_index, [0])), dim
            =0)

        x_2 = self.conv1(x_1)

        if train:
            x_2 = self.dropout(x_2)

        x_3 = F.relu(x_2)

        if train:
            x_3 = self.dropout(x_3)

        x_4 = self.conv2(x_3, edge_index)

        if train:
            x_4 = self.dropout(x_4)

        x_4 = F.relu(x_4)
        x_4 = self.pool(self.bn(x_4))

        attention = torch.matmul(x_4, torch.
            transpose(x_4, 0, 1))
        attention = sigmoid(attention)
        attention = attention / torch.sum(
            attention, dim=1).unsqueeze(1)

        # Batch-based attention mechanism
        attention_weights = self.
            attention_softmax(self.
            attention_fc(x_4))
        x_5 = x_4 * attention_weights

        x_5 = torch.matmul(attention, x_4)

        x_6 = x_5 + x_4
        output = self.fc1(x_6)
        return output
```

Listing 2: Enhanced GECCO Modified Train Class

```
    if source == 'checkpoint':
        model, optimizer, old_epoch, epoch_loss =
            load_checkpoint(device,
            checkpoint_path)
    elif source == 'new':
        model = new_model(device)
        optimizer = optim.Adam(model.parameters()
            , lr=1e-3, weight_decay=1e-4)
    else:
        assert 0
```

References

[1] L. Rathore and R. Yadav, “Advancements in Handwritten Digit Recognition: A Literature Review of Machine Learning and Deep Learning Approaches,” IJRASET, vol. 11, no. 8, pp. 1879–1883, Aug. 2023, doi: 10.22214/ijraset.2023.55482.

[2] Li Deng, “The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web],” IEEE Signal Process. Mag., vol. 29, no. 6, pp. 141–142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.

[3] D. Ciresan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI: IEEE, Jun. 2012, pp. 3642–3649. doi: 10.1109/CVPR.2012.6248110.

[4] S. Anwar, N. Barnes, and L. Petersson, “A Systematic Evaluation: Fine-Grained CNN vs. Traditional CNN Classifiers,” Electronics, vol. 12, no. 23, p. 4877, Dec. 2023, doi: 10.3390/electronics12234877.

[5] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar, “A review of convolutional neural networks in computer vision,” Artif Intell Rev, vol. 57, no. 4, p. 99, Mar. 2024, doi: 10.1007/s10462-024-10721-6.

[6] G. Wei, G. Li, J. Zhao, and A. He, “Development of a LeNet-5 Gas Identification CNN Structure for Electronic Noses,” Sensors, vol. 19, no. 1, p. 217, Jan. 2019, doi: 10.3390/s19010217.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in Advances in Neural Information Processing Systems, F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[8] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Apr. 10, 2015, arXiv: arXiv:1409.1556. doi: 10.48550/arXiv.1409.1556.

[9] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778. Accessed: Dec. 10, 2024. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[10] C. F. G. D. Santos and J. P. Papa, “Avoiding Overfitting: A Survey on Regularization Methods for Convolutional Neural Networks,” ACM Comput. Surv., vol. 54, no. 10s, pp. 1–25, Jan. 2022, doi: 10.1145/3510413.

[11] D. Beohar and A. Rasool, “Handwritten Digit Recognition of MNIST dataset using Deep Learning state-of-the-art Artificial Neural Network (ANN) and Convolutional Neural Network (CNN),” in 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India: IEEE, Mar. 2021, pp. 542–548. doi: 10.1109/ESCI50559.2021.9396870.

[12] S. Ahlawat, A. Choudhary, A. Nayyar, S. Singh, and B. Yoon, “Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN),” Sensors, vol. 20, no. 12, p. 3344, Jun. 2020, doi: 10.3390/s20123344.

[13] B. Shi, Z. Wu, M. Mao, X. Wang, and T. Darrell, “When Do We Not Need Larger Vision Models?,” Jul. 18, 2024, arXiv: arXiv:2403.13043. doi: 10.48550/arXiv.2403.13043.

[14] M. Zahedi, Mohammad. S. Abazari, and A. Savadi, “Pruning and Mixed Precision Techniques for Accelerating Neural Network,” in 2023 13th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, Islamic Republic of: IEEE, Nov. 2023, pp. 085–090. doi: 10.1109/ICCKE60553.2023.10326309.

[15] X. Wang, Y. Sun, X. Chen, and H. Xu, “DDEP: Evolutionary pruning using distilled dataset,” Information Sciences, vol. 659, p. 120048, Feb. 2024, doi: 10.1016/j.ins.2023.120048.

[16] D. Sinha and M. El-Sharkawy, “Thin MobileNet: An Enhanced MobileNet Architecture,” in 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York City, NY, USA: IEEE, Oct. 2019, pp. 0280–0285. doi: 10.1109/UEMCON47517.2019.8993089.

[17] J. Fein-Ashley, S. Wickramasinghe, B. Zhang, R. Kannan, and V. Prasanna, "A Single Graph Convolution is All You Need: Efficient Grayscale Image Classification," in 2024 IEEE International Conference on Image Processing (ICIP), Abu Dhabi, United Arab Emirates: IEEE, Oct. 2024, pp. 849–855. doi: 10.1109/ICIP51287.2024.10647347.
[18] T. D. Ross, J. J. Bradley, L. J. Hudson, and M. P. O'Connor, "SAR ATR: so what's the problem? An MSTAR perspective," presented at the AeroSense '99, E. G. Zelnio, Ed., Orlando, FL, Aug. 1999, pp. 662–672. doi: 10.1117/12.357681.
[19] E. R. Keydel, S. W. Lee, and J. T. Moore, "MSTAR extended operating conditions: a tutorial," presented at the Aerospace/Defense Sensing and Controls, E. G. Zelnio and R. J. Douglass, Eds., Orlando, FL, Jun. 1996, pp. 228–242. doi: 10.1117/12.242059.
[20] A. Jacobi, M. Chung, A. Bernheim, and C. Eber, "Portable chest X-ray in coronavirus disease-19 (COVID-19): A pictorial review," Clinical Imaging, vol. 64, pp. 35–42, Aug. 2020, doi: 10.1016/j.clinimag.2020.04.001.
[21] "Handwritten-Digit-Recognition/handwritten_digit_recognition.ipynb at main · Eng-Abdelrahman-M/Handwritten-Digit-Recognition." Accessed: Dec. 11, 2024. [Online]. Available: https://github.com/Eng-Abdelrahman-M/Handwritten-Digit-Recognition/blob/main/handwritten_digit_recognition.ipynb
[22] H. M. D. Kabir et al., "SpinalNet: Deep Neural Network With Gradual Input," IEEE Trans. Artif. Intell., vol. 4, no. 5, pp. 1165–1177, Oct. 2023, doi: 10.1109/TAI.2022.3185179.
[23] V. Mazzia, F. Salvetti, and M. Chiaberge, "Efficient-CapsNet: capsule network with self-attention routing," Sci Rep, vol. 11, no. 1, p. 14634, Jul. 2021, doi: 10.1038/s41598-021-93977-0.
[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.
[25] B. U. Mahmud and G. Y. Hong, "Semantic Image Segmentation using CNN (Convolutional Neural Network) based Technique," in 2022 IEEE World Conference on Applied Intelligence and Computing (AIC), Sonbhadra, India: IEEE, Jun. 2022, pp. 210–214. doi: 10.1109/AIC55036.2022.9848977.
[26] J. Zamora Esquivel, A. Cruz Vargas, P. Lopez Meyer, and O. Tickoo, "Adaptive Convolutional Kernels," presented at the Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, 2019, pp. 0–0. Accessed: Dec. 11, 2024. [Online]. Available: https://openaccess.thecvf.com/content_ICCVW_2019/html/NeurArch/Esquivel_Adaptive_Convolutional_Kernels_ICCVW_2019_paper.html
[27] Y. Tian, D. Su, S. Lauria, and X. Liu, "Recent advances on loss functions in deep learning for computer vision," Neurocomputing, vol. 497, pp. 129–158, Aug. 2022, doi: 10.1016/j.neucom.2022.04.127.
[28] R. Jaiswal and B. Singh, "A Comparative Study of Loss Functions for Deep Neural Networks in Time Series Analysis," in Big Data, Machine Learning, and Applications, vol. 1053, M. D. Borah, D. S. Laiphrakpam, N. Auluck, and V. E. Balas, Eds., in Lecture Notes in Electrical Engineering, vol. 1053. , Singapore: Springer Nature Singapore, 2024, pp. 147–163. doi: 10.1007/978-981-99-3481-2_12.
[29] G. Habib and S. Qureshi, "Optimization and acceleration of convolutional neural networks: A survey," Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 7, pp. 4244–4268, Jul. 2022, doi: 10.1016/j.jksuci.2020.10.004.
[30] "Google Colab." Accessed: Dec. 11, 2024. [Online]. Available: https://colab.research.google.com/
[31] "Kaggle: Your Home for Data Science." Accessed: Dec. 11, 2024. [Online]. Available: https://www.kaggle.com/
[32] "Keras: Deep Learning for humans." Accessed: Dec. 11, 2024. [Online]. Available: https://keras.io/
[33] "PyTorch." Accessed: Dec. 11, 2024. [Online]. Available: https://pytorch.org/
[34] M. Li, Y. Wang, J. Huang, E. Purwanto, and K. L. Man, "Patch-Based Multi-Level Attention Mechanism for Few-Shot Multi-Label Medical Image Classification," in 2023 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Jiangsu, China: IEEE, Nov. 2023, pp. 84–91. doi: 10.1109/CyberC58899.2023.00024.