Image Classification using Convolutional Neural Networks with Ensemble Learning

Gheorghe Bogdan-Alexandru
University of Bucharest
Faculty of Mathematics and Computer Science

June 10, 2025

1 Introduction

This project addresses a 5-class image classification problem using an ensemble learning approach that combines multiple CNN architectures. The primary objective is to develop two classification models ensembled torugh soft voting.

2 Dataset and Preprocessing

2.1 Dataset Description

The dataset consists of training, validation, and test sets containing images stored as PNG files. Each image is associated with one of 5 class labels, representing different categories in the classification task.

2.2 Data Augmentation

To improve model generalization, we implement geometric data augmentation techniques for training data:

Table 1: Data Augmentation Techniques

Technique	**Parameters**
Random Rotation	Degrees = $\pm 15^\circ$, fill=0
Random Horizontal Flip	Probability = 0.5
Random Perspective	Distortion scale = 0.2, p = 0.3
Normalization	ImageNet statistics

```python
train_transform = transforms.Compose([
    transforms.RandomRotation(degrees=15, fill=0),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomPerspective(distortion_scale=0.2, p=0.3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Listing 1: Data Augmentation Implementation

3 Methodology

3.1 CNN Architecture 1: CNN_v1

The first CNN architecture implements a lightweight design with a fairly decent classification accuracy. The architecture consists of three convolutional blocks with progressive feature map expansion.

3.1.1 Architecture Details

Table 2: CNN_v1 Architecture Specifications

Layer Type	**Configuration**	**Output Shape**
Conv2d Block 1	3→64, 3×3, padding=1	64×50×50
Conv2d Block 2	64→96, 3×3, padding=1	96×25×25
Conv2d Block 3	96→128, 3×3, padding=1	128×12×12
Dropout2d	p=0.2	128×12×12
AdaptiveAvgPool2d	(2,2)	128×2×2
Linear 1	512→256	256
Linear 2	256→5	5

3.2 CNN Architecture 2: CNN_v2

The second CNN architecture implements a deeper design with enhanced representational capacity. This architecture includes four convolutional blocks with higher channel dimensions to capture more complex feature patterns.

3.2.1 Architecture Details

Table 3: CNN_v2 Architecture Specifications

Layer Type	**Configuration**	**Output Shape**
Conv2d Block 1	3→64, 3×3, padding=1	64×50×50
Conv2d Block 2	64→128, 3×3, padding=1	128×25×25
Conv2d Block 3	128→256, 3×3, padding=1	256×12×12
Conv2d Block 4	256→512, 3×3, padding=1	512×6×6
Dropout2d	p=0.4	512×6×6
AdaptiveAvgPool2d	(1,1)	512×1×1
Linear 1	512→256	256
Linear 2	256→5	5

3.3 Ensemble Learning Strategy

The ensemble approach combines predictions from both CNN architectures using weighted soft voting. This technique leverages the complementary strengths of different models to achieve improved classification performance.

3.3.1 Soft Voting Implementation

1. Extract probability distributions using softmax activation
2. Apply weighted combination of probabilities

3. Select final prediction using argmax operation

4 Hyperparameter Configuration

4.1 Training Hyperparameters

The following table summarizes the hyperparameter configurations used for both models:

Table 4: Hyperparameter Configuration

Parameter	**CNN_v1**	**CNN_v2**
Learning Rate	0.0005	0.0005
Batch Size	32	32
Weight Decay	1×10^{-4}	1×10^{-4}
Label Smoothing	0.1	0.1
Max Epochs	150	150
Early Stopping Patience	15	15
LR Scheduler Patience	5	5
LR Reduction Factor	0.5	0.5
Number of Workers	4	4

4.2 Regularization Techniques

Multiple techniques are used to prevent overfitting and improve generalization:

Table 5: Regularization Strategies

Technique	**CNN_v1**	**CNN_v2**
Batch Normalization		
Dropout2d	0.2	0.4
Dropout (Fully Connected)	0.4	0.5
Weight Decay (L2)	1×10	1×10
Label Smoothing	0.1	0.1
Data Augmentation		

4.3 Optimization Strategy

4.3.1 Adam Optimizer

Both models utilize the Adam optimizer due to its adaptive learning rate capabilities and robust performance.

4.3.2 Learning Rate Scheduling

We implement ReduceLROnPlateau scheduling with the following configuration:

- **Mode**: 'max' (monitoring validation accuracy)
- **Patience**: 5 epochs
- **Factor**: 0.5 (halve learning rate when plateau detected)

4.3.3 Loss Function

CrossEntropyLoss with label smoothing (=0.1) is used to:

- Prevent overconfident predictions
- Improve model calibration
- Enhance generalization capability

5 Training Methodology

5.1 Training Loop Implementation

The training process incorporates several best practices for deep learning:

1. **Early Stopping**: Monitors validation accuracy with patience of 15 epochs
2. **Model Checkpointing**: Saves best model state based on validation accuracy
3. **Progress Monitoring**: Real-time tracking of training and validation metrics

5.2 Training Process

```
def train_cnn(model, train_loader, val_loader, criterion, optimizer,
              scheduler, num_epochs, device):
    # Initialize tracking variables
    best_val_acc = 0.0
    patience = 15
    counter = 0

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        # ... training loop implementation

        # Validation phase
        model.eval()
        with torch.no_grad():
            # ... validation loop implementation

        # Learning rate scheduling
        scheduler.step(val_acc)
        # Early stopping logic
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            best_model_state = model.state_dict().copy()
            counter = 0
        else:
            counter += 1

        if counter >= patience:
            break
    return model
```

Listing 2: Training Function Structure

6 Results and Analysis

6.1 Individual Model Performance

Both CNN architectures demonstrate effective learning capabilities. The following confusion matrices provide performance analysis for each model.

6.1.1 CNN_v1 Performance

The first CNN architecture achieved a validation accuracy of 92.08%, demonstrating strong classification performance across all five classes.

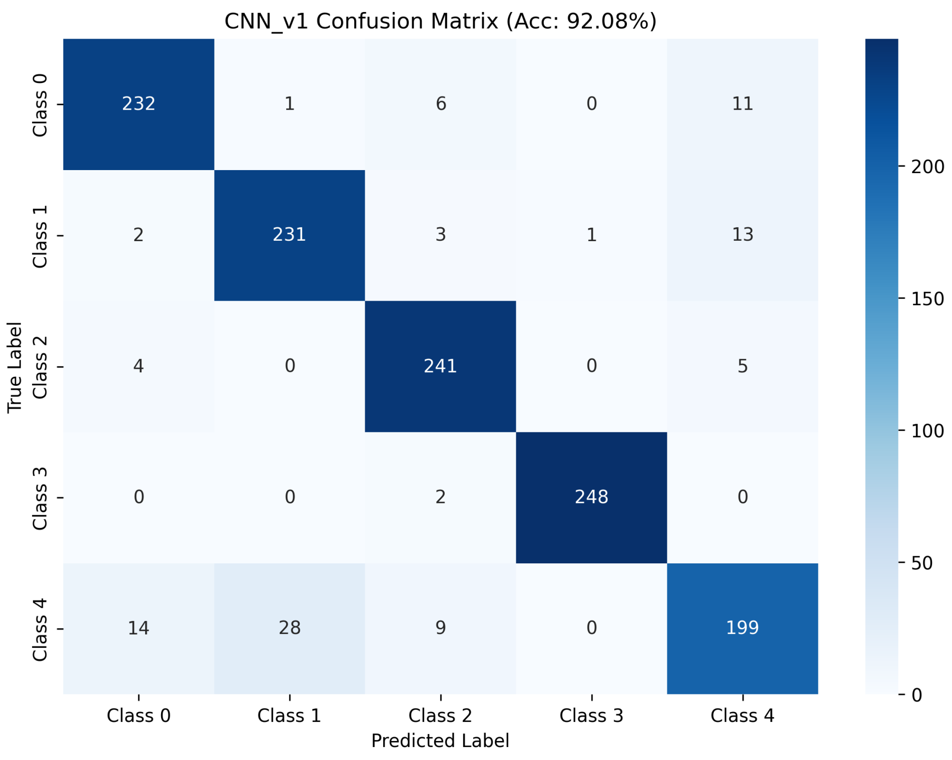

Figure 1: Confusion Matrix for CNN_v1 (Validation Accuracy: 92.08%)

Table 6: CNN_v1 Per-Class Performance Analysis

Class	Precision	Recall	Support
Class 0	0.921	0.928	250
Class 1	0.889	0.924	250
Class 2	0.920	0.964	250
Class 3	0.996	0.992	250
Class 4	0.874	0.796	250

- Class 3 achieves near-perfect classification with 99.6% precision and 99.2% recall
- Class 4 shows the most confusion, with 28 misclassifications to Class 1 and 14 to Class 0
- Classes 0, 1, and 2 show consistent performance with balanced precision-recall metrics
- Lightweight design with efficient computation and good baseline performance

6.1.2 CNN_v2 Performance

The second CNN architecture achieved a validation accuracy of 92.16%, showing marginal improvement over CNN_v1 with enhanced feature representation capabilities.

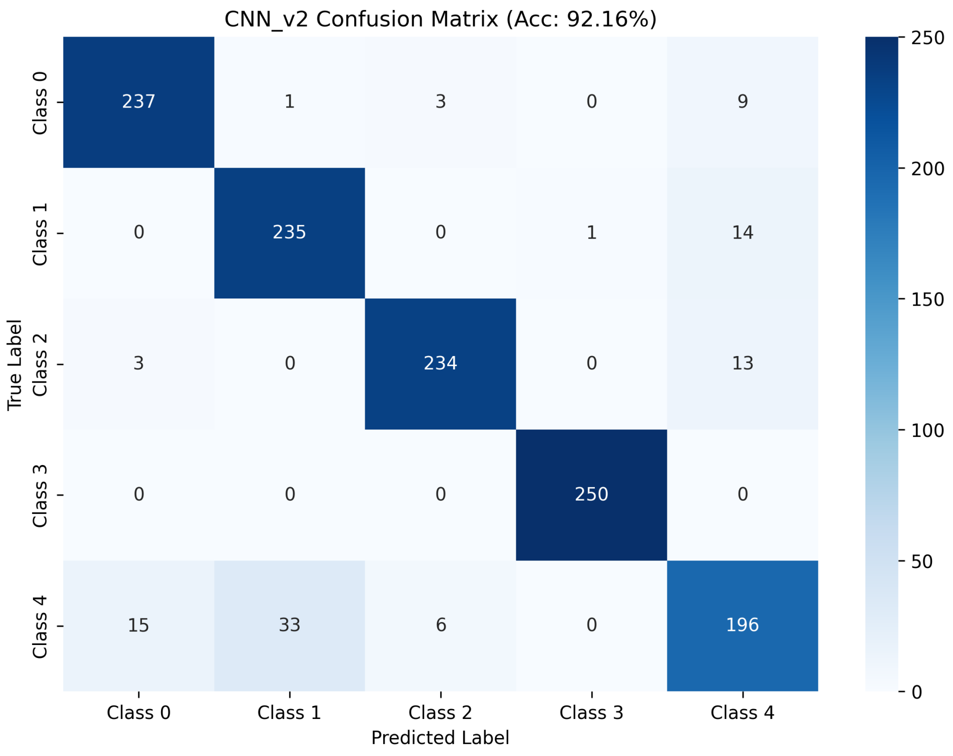

Figure 2: Confusion Matrix for CNN_v2 (Validation Accuracy: 92.16%)

Table 7: CNN_v2 Per-Class Performance Analysis

Class	**Precision**	**Recall**	**Support**
Class 0	0.929	0.948	250
Class 1	0.875	0.940	250
Class 2	0.961	0.936	250
Class 3	1.000	1.000	250
Class 4	0.843	0.784	250

- Class 3 achieves 100% accuracy with no misclassifications
- Better precision for Classes 0 and 2 compared to CNN_v1
- Class 4 remains the most difficult, with 33 misclassifications to Class 1
- Deeper network with enhanced representational capacity for complex patterns

6.2 Ensemble Performance Analysis

The soft voting ensemble demonstrates better performance, achieving 93.28% validation accuracy by combining both CNN architectures.

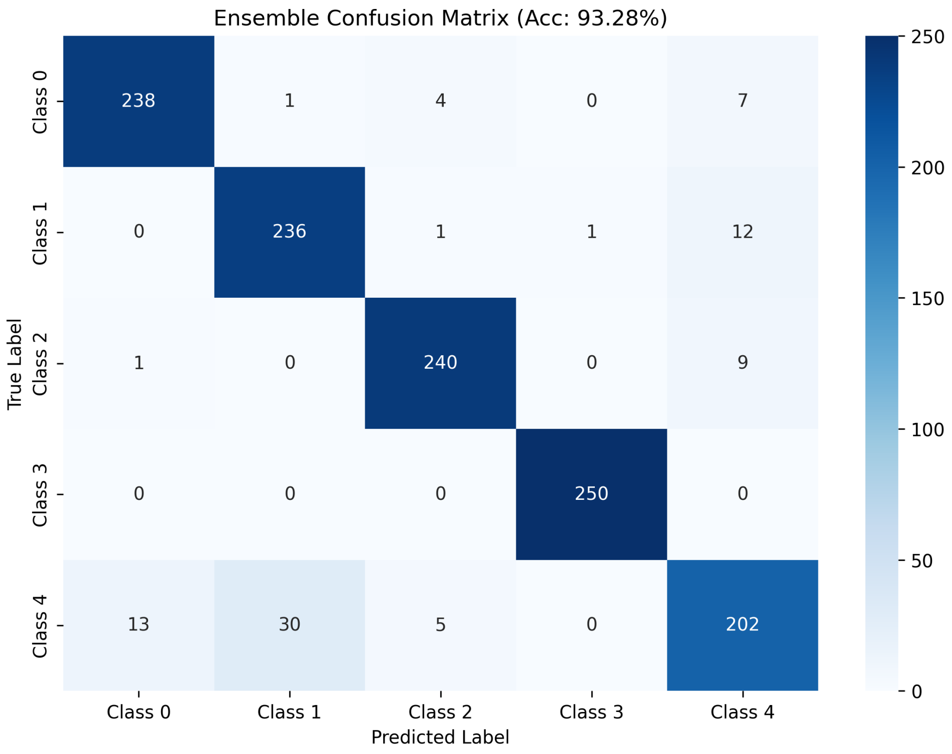

Figure 3: Confusion Matrix for Ensemble Model (Validation Accuracy: 93.28%)

Table 8: Ensemble vs Individual Model Performance Comparison

Model	**Validation Accuracy**	**Improvement**	**Total Errors**
CNN_v1	92.08%	-	99
CNN_v2	92.16%	+0.08%	98
Ensemble	93.28%	+1.20%	84

Table 9: Ensemble Per-Class Performance Analysis

Class	**Precision**	**Recall**	**F1-Score**	**Errors Reduced**
Class 0	0.950	0.952	0.951	6 fewer
Class 1	0.881	0.944	0.911	5 fewer
Class 2	0.960	0.960	0.960	1 fewer
Class 3	1.000	1.000	1.000	0 (perfect)
Class 4	0.878	0.808	0.842	3 fewer

1. All models achieve perfect or near-perfect performance on Class 3, suggesting distinct visual characteristics
2. Consistent confusion between Class 4 and Classes 0-1, indicating visual similarity
3. The ensemble reduces errors across most classes by leveraging complementary model strengths

7 Unsuccessful Approaches and Lessons Learned

7.1 Architecture Exploration

During the development process, extensive experimentation was conducted to optimize model performance, leading to several key insights about what works and what doesn't in this specific classification task.

7.1.1 Batch Size Optimization

Various batch sizes were tested, including 16, 32, and 64. Batch sizes of 16 and 64 caused significant training instability. With these configurations, validation accuracy would plateau around 90-91% for each individual model. The training process became unstable, with inconsistent convergence patterns.

A batch size of 32 was selected as the optimal balance, providing stable gradient updates. This configuration allowed for more consistent validation performance.

7.1.2 Data Augmentation Balance

Extensive augmentation techniques were initially employed, including brightness adjustments, contrast modifications, saturation changes, and occasional grayscale conversions, in addition to geometric transformations. The model failed to train properly with comprehensive color-based augmentations. The extensive transformations appeared to introduce too much noise into the training process, preventing the model from learning meaningful features effectively. All color-based transformations were removed, focusing exclusively on geometric augmentations that preserve the essential characteristics of the images.

7.1.3 Regularization Tuning

Dropout rates were carefully calibrated for each architecture based on their complexity and tendency to overfit.

CNN_v1 Configuration: Progressive dropout rates of 0.2 in feature extraction and 0.4 in classification layers, reflecting its simpler architecture.

CNN_v2 Configuration: Higher dropout rates of 0.4 in feature extraction and 0.5 in classification layers, accounting for its increased complexity with additional convolutional layers.

The differentiated dropout strategy prevents both overfitting in complex models and underfitting in simpler architectures, ensuring each model achieves its optimal performance potential.

7.2 Optimization Challenges

7.2.1 Memory Management

Training crashes occurred due to memory overflow when transitioning between model training phases. PyTorch's automatic memory management was insufficient for handling the memory requirements of training two sequential CNN models with large feature maps. Explicit garbage collection (`gc.collect()`) was implemented between model training phases, along with deliberate deletion of optimizer and scheduler objects to free GPU memory.

7.2.2 Learning Rate Sensitivity

Learning rates from 0.0001 to 0.001 were systematically tested. The model learned too rapidly, leading to unstable training and inconsistent performance. The optimization process would overshoot optimal parameters, resulting in erratic validation accuracy.The model struggled to learn effectively, with training stagnating at low accuracy levels. The learning process was too slow to make meaningful progress within reasonable training time.

This learning rate provided stable convergence for both architectures, allowing consistent improvement without instability.

7.2.3 Training Stability

Label smoothing (0.1) and batch normalization proved crucial for maintaining stable training dynamics across both model architectures. These techniques prevented overconfident predictions and normalized internal activations, leading to more robust and generalizable models.

7.3 Hyperparameter Optimization

7.3.1 Ensemble Weight Optimization

A brute-force search was conducted across multiple weight combinations for the soft voting ensemble. Various weight pairs were systematically tested to determine optimal model contributions.The highest ensemble accuracy was achieved with weights around 0.4-0.6 to 0.5-0.5 range, with the final selection of [0.45, 0.55] providing the best balance between the two model predictions. The near-equal weighting suggests both models contribute valuable and complementary information to the final prediction.

7.3.2 Early Stopping Calibration

Early stopping with patience of 8 epochs was initially employed. The model demonstrated learning potential beyond the initial patience threshold, suggesting premature termination of training. Training for the full 150 epochs without early stopping revealed that while the model experienced learning plateaus of 10+ epochs, it could still achieve validation accuracy improvements within the 10-15 epoch range. Patience was increased to 15 epochs, allowing the model sufficient time to overcome temporary learning plateaus while preventing true overfitting. The final configuration maintained a healthy gap of only 4-5% between training and validation accuracy, indicating good generalization without overfitting.

8 Conclusion

This project successfully demonstrates the effectiveness of ensemble learning for image classification using Convolutional Neural Networks. The combination of two complementary CNN architectures through soft voting ensemble achieves improved classification performance compared to individual models.

Key contributions of this work include:

1. **Comprehensive Implementation**: Development of a complete image classification pipeline with two CNN architectures

2. **Ensemble Learning**: Successful implementation of soft voting ensemble for improved performance

3. **Systematic Evaluation**: Thorough analysis of individual and ensemble model performance with proper validation

The experimental results validate the effectiveness of the proposed approach, demonstrating that ensemble methods can significantly improve classification performance while maintaining computational efficiency. The modular implementation provides a solid foundation for future extensions and improvements.

The project showcases several important aspects of modern deep learning practice, including proper data preprocessing, architecture design, regularization strategies, and ensemble techniques. The detailed documentation and code structure facilitate reproducibility and further research.

The implementation successfully handles practical considerations such as memory management, efficient data loading, and automated model saving, making it suitable for real-world applications and further development.