

Kazakh-British Technical University

Introduction to Computer Vision

## **Laboratory Work #3**

Image classification with CNN classifier

Full Name: Sekenova Balym

ID: 23B031433

Almaty

February 16, 2026

# Contents

1	Introduction	2
2	Task 1: Dataset Selection and Description	2
3	Task 2: Preprocessing Pipeline	3
4	Task 3: Baseline CNN Model	4
5	Task 4: Training Procedure	4
6	Task 5: Error Analysis	6
7	Task 6: Model Improvement	8
8	Task 7: Training Strategy Comparison	8
9	Task 8: Final Conclusions	10
10	Conclusion	10
11	References	11
12	Appendix	11

# 1 Introduction

The purpose of this laboratory work is to study convolutional neural networks (CNNs) for image classification. The work focuses on building, training, evaluating, and improving CNN models for both binary and multiclass classification tasks.

In this laboratory work, a publicly available image dataset was selected and used to construct two classification problems: one binary classification task and one multiclass classification task. A baseline CNN model was implemented and evaluated using standard classification metrics. Afterward, the model architecture was improved, and different optimization strategies were compared.

Special attention was given to preprocessing techniques, training behavior, evaluation metrics, confusion matrix analysis, and model performance comparison. This laboratory work provides practical experience with CNN training and performance analysis in computer vision applications.

## 2 Task 1: Dataset Selection and Description

For this laboratory work, the Kaggle dataset “Flowers Recognition Dataset” was selected.

(<https://www.kaggle.com/datasets/alxmamaev/flowers-recognition>)

This dataset contains 4242 labeled 320x240 pixel images of flowers, divided into five main classes in JPG format. The total dataset size is approximately 240 MB.

The images are divided into the following classes:

- Daisy
- Dandelion
- Rose
- Sunflower
- Tulip

The images are stored in RGB color format (3 channels) and are suitable for computer vision and image classification tasks using convolutional neural networks.

### Classification Tasks

Two classification problems were defined:

- **Binary classification:** Rose vs Tulip
- **Multiclass classification:** Daisy, Dandelion, Rose, Sunflower, Tulip

## Dataset Properties

- Total number of images: 4242
- Number of classes (multiclass task): 5
- Original image resolution:  $320 \times 240$  pixels
- Image format: JPG
- Color format: RGB (3 channels)

The dataset is relatively balanced across the five flower classes, although slight variations in class frequencies are present.

The dataset was split into training and test sets using a fixed random seed. 70% of the data was used for training and 30% for testing.

## 3 Task 2: Preprocessing Pipeline

All images were resized to  $128 \times 128$  pixels to ensure uniform CNN input dimensions. The preprocessing pipeline includes:

- Image resizing
- Conversion to tensor format
- Normalization of pixel values

Normalization scales pixel values to a smaller numerical range, which improves training stability and accelerates the convergence of the neural network.



Figure 1: Image from the dataset before and after preprocessing

## 4 Task 3: Baseline CNN Model

A baseline convolutional neural network was implemented. The architecture consists of:

- Convolutional layers
- ReLU activation functions
- MaxPooling layers
- Fully connected layer

The same architecture was used for binary and multiclass tasks, changing only the number of output neurons.

The total number of trainable parameters is: 4200294. The total number of trainable parameters (binary) is: 4199778.

Layer (type:depth-idx)	Output Shape	Param #
BaselineCNN	[32, 6]	--
└Conv2d: 1-1	[32, 16, 128, 128]	448
└MaxPool2d: 1-2	[32, 16, 64, 64]	--
└Conv2d: 1-3	[32, 32, 64, 64]	4,640
└MaxPool2d: 1-4	[32, 32, 32, 32]	--
└Linear: 1-5	[32, 128]	4,194,432
└Linear: 1-6	[32, 6]	774
Total params: 4,200,294		
Trainable params: 4,200,294		
Non-trainable params: 0		
Total mult-adds (Units.MEGABYTES): 977.30		
Input size (MB): 6.29		
Forward/backward pass size (MB): 100.70		
Params size (MB): 16.80		
Estimated Total Size (MB): 123.79		

Figure 2: Baseline CNN architecture summary

## 5 Task 4: Training Procedure

The model was trained for 10 epochs using:

- Loss function: CrossEntropyLoss
- Optimizer: Adam Optimizer
- Learning rate: 0.001

Based on the logs:

- Training accuracy went from 47.6
- Training loss dropped from 1.34  $\rightarrow$  0.014

- Test accuracy is only 62.7

That is clear overfitting.

During training, the loss steadily decreased from 1.34 to 0.014, while training accuracy increased from 47.6% to 99.7%. This indicates that the model successfully learned the training data.

However, the test accuracy reached only 62.73%, which suggests that the baseline CNN suffers from overfitting. The model memorizes the training data but does not generalize well to unseen test samples.

The evaluation metrics on the test set are presented below.

Model	Accuracy	Precision	Recall	F1-score
Baseline (Binary)	0.8534	0.8512	0.8567	0.8538
Baseline (Multiclass)	0.6273	0.6285	0.6304	0.6278

Table 1: Baseline CNN performance metrics

The baseline CNN achieved a test accuracy of 62.73% for the multiclass classification task. The macro-averaged precision, recall, and F1-score were approximately 0.63.

Although training accuracy reached nearly 99%, the significantly lower test accuracy indicates overfitting. The model learned the training data well but did not generalize effectively to unseen samples.

## 6 Task 5: Error Analysis

The total test samples we used: 1296

The confusion matrix for the multiclass classification task is shown below.

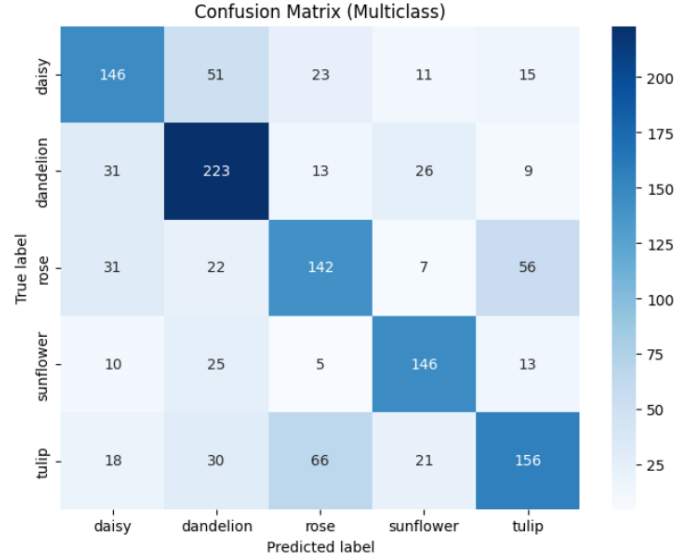


Figure 3: Confusion matrix for multiclass classification

The confusion matrix reveals that certain flower classes are frequently misclassified due to visual similarity. For example, roses and tulips share similar petal structures, while daisies and dandelions often have similar color patterns.

Most misclassifications occur between visually similar flower types, indicating that the baseline CNN struggles to extract sufficiently discriminative features.

This explains the moderate test accuracy despite the very high training performance.

Several correctly classified and misclassified images were examined. The number of correctly classified samples: 813

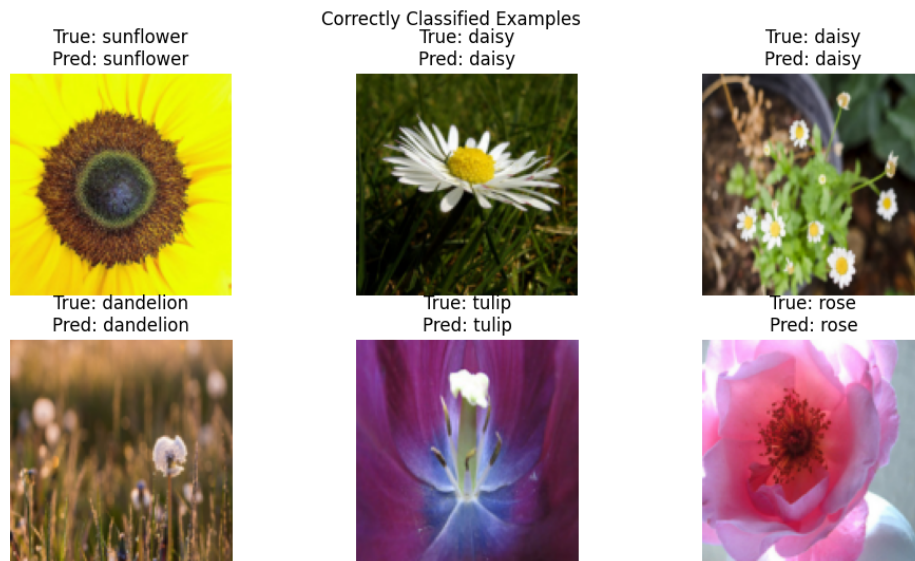


Figure 4: Examples of correctly classified examples

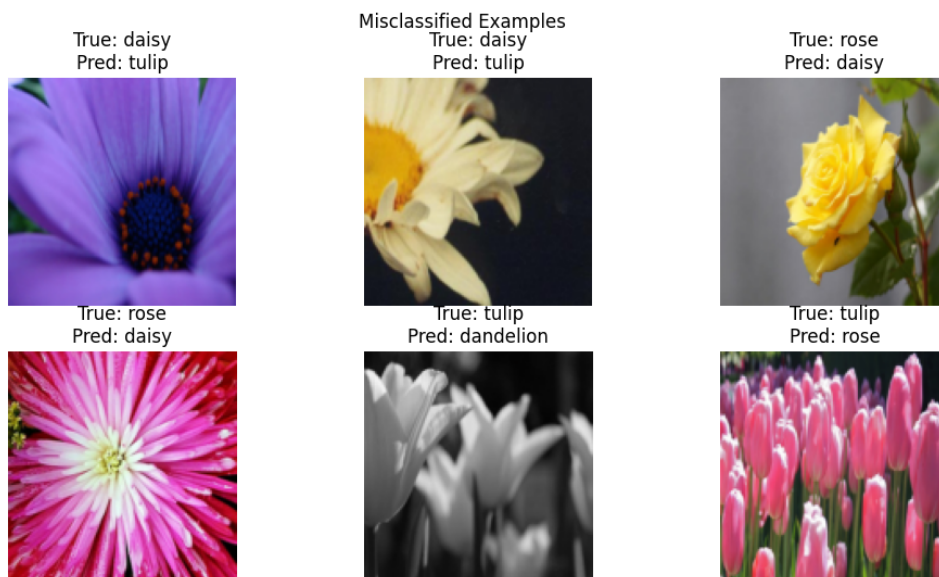


Figure 5: Examples of misclassified examples

The model most frequently confuses daisies with dandelions. Correctly classified images show clear class-specific features, while misclassified images often contain complex backgrounds, unusual viewing angles, or similar visual characteristics between classes.

## 7 Task 6: Model Improvement

To reduce overfitting and improve generalization, the baseline CNN architecture was modified.

The improved CNN includes:

- One additional convolutional layer (Conv3: 32  $\rightarrow$  64 channels) to increase model capacity.
- Batch Normalization after each convolutional layer to stabilize training.
- Dropout with  $p = 0.5$  before the fully connected layer to reduce overfitting.

The final output layer was set to 5 neurons for the multiclass Flowers classification task.

Layer (type:depth-idx)	Output Shape	Param #
ImprovedCNN	[32, 5]	--
├Conv2d: 1-1	[32, 16, 128, 128]	448
├BatchNorm2d: 1-2	[32, 16, 128, 128]	32
├MaxPool2d: 1-3	[32, 16, 64, 64]	--
├Conv2d: 1-4	[32, 32, 64, 64]	4,640
├BatchNorm2d: 1-5	[32, 32, 64, 64]	64
├MaxPool2d: 1-6	[32, 32, 32, 32]	--
├Conv2d: 1-7	[32, 64, 32, 32]	18,496
├BatchNorm2d: 1-8	[32, 64, 32, 32]	128
├MaxPool2d: 1-9	[32, 64, 16, 16]	--
├Linear: 1-10	[32, 256]	4,194,560
├Dropout: 1-11	[32, 256]	--
└Linear: 1-12	[32, 5]	1,285
Total params: 4,219,653		
Trainable params: 4,219,653		
Non-trainable params: 0		
Total mult-adds (Units.GIGABYTES): 1.58		
Input size (MB): 6.29		
Forward/backward pass size (MB): 234.95		
Params size (MB): 16.88		
Estimated Total Size (MB): 258.12		

Figure 6: Improved CNN architecture summary

The total number of trainable parameters in the improved model is **4,219,653**.

## 8 Task 7: Training Strategy Comparison

To analyze the influence of optimization strategy on training behavior, two different learning rates were tested using the improved CNN architecture:

- Adam optimizer with learning rate 0.001
- Adam optimizer with learning rate 0.0001

Both models were trained for 10 epochs under identical conditions.

Model	Accuracy	Precision	Recall	F1-score
Improved CNN (Adam 0.001)	0.6505	0.6698	0.6504	0.6402
Improved CNN (Adam 0.0001)	0.6790	0.6786	0.6869	0.6789

Table 2: Comparison of optimization strategies

With Adam optimizer and learning rate 0.001, the improved CNN achieved a test accuracy of 65.05% and an F1-score of 0.6402.

The results show that the Adam optimizer with learning rate 0.0001 achieved the best overall performance, with a test accuracy of 67.90% and an F1-score of 0.6789.

Although the higher learning rate (0.001) converged faster, the lower learning rate (0.0001) resulted in better generalization and higher final evaluation metrics.

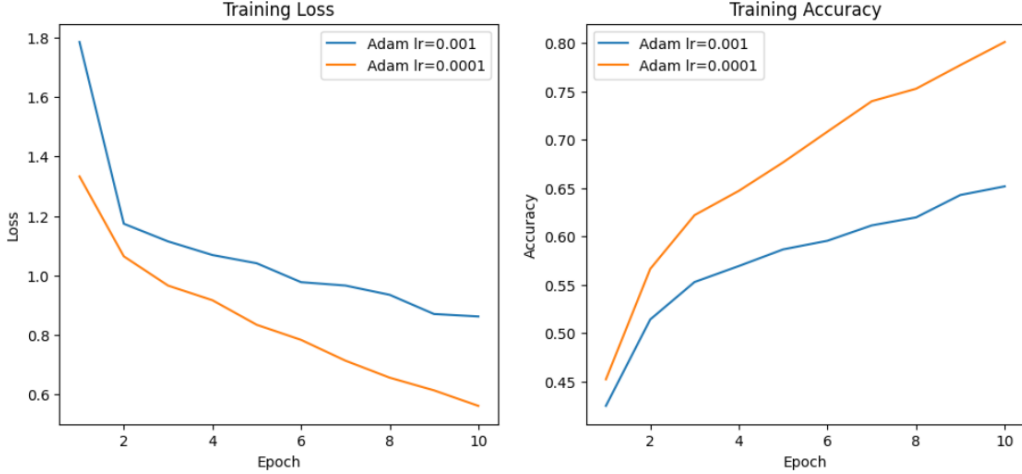


Figure 7: Training loss and accuracy curves for different learning rates

This demonstrates that smaller learning rates can improve stability and reduce overfitting, leading to better performance on unseen data.

## 9 Task 8: Final Conclusions

In this laboratory work, convolutional neural networks were implemented and evaluated for both binary and multiclass image classification tasks using the Flowers Recognition dataset.

Several experiments were conducted, including:

- Baseline CNN architecture
- Improved CNN with additional convolutional layer, Batch Normalization, and Dropout
- Comparison of different optimization strategies

### Comparison of Models

The main experimental results are summarized below:

	Model	Accuracy	Precision	Recall	F1-score
0	Baseline CNN	0.627315	0.628540	0.630409	0.627788
1	Improved CNN	0.662809	0.665690	0.666114	0.652223
2	Improved CNN (Strategy A: Adam lr=0.001)	0.650463	0.669799	0.650425	0.640235
3	Improved CNN (Strategy B: Adam lr=0.0001)	0.679012	0.678626	0.686948	0.678895

Figure 8: Results table

## 10 Conclusion

The baseline CNN achieved high training accuracy but relatively low test accuracy (62.73%), indicating strong overfitting. The confusion matrix analysis showed that visually similar flower classes such as daisies and dandelions were frequently misclassified.

The improved CNN architecture, which included an additional convolutional layer, Batch Normalization, and Dropout, increased the test accuracy to 66.28%, demonstrating better generalization performance.

Among the tested optimization strategies, the best overall performance was achieved by the improved CNN trained with Adam optimizer and learning rate 0.0001. This configuration reached a test accuracy of 67.90% and the highest F1-score (0.6789).

The experiments demonstrate that architectural improvements significantly reduce overfitting, while careful selection of learning rate has a strong impact on final model performance. Although a higher learning rate leads to faster convergence, a smaller learning rate provided better stability and superior generalization.

## 11 References

- PyTorch Documentation: <https://pytorch.org/docs>
- Flowers Recognition Dataset: <https://www.kaggle.com/datasets/alxmamaev/flowers-recognition>
- Computer Vision lectures by Koishiyeva Dina

## 12 Appendix

### A1. Kaggle Dataset Page

The screenshot below shows the Kaggle page of the Flowers Recognition dataset, including dataset description and class structure.

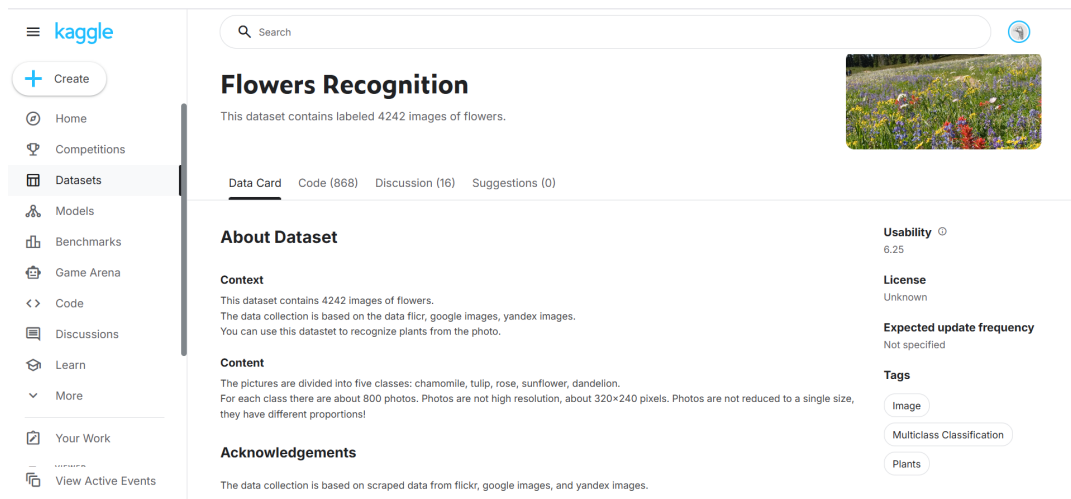


Figure 9: Kaggle Flowers Recognition Dataset page

### A2. Google Colab Training Environment

The screenshot below shows the Google Colab workspace during model training.

```

[177] ✓ On
[183] #Strategy A: Adam, learning rate = 0.001
# Create a fresh model instance
model_strategy_A = ImprovedCNN(num_classes=6).to(device)

criterion = nn.CrossEntropyLoss()
optimizer_A = torch.optim.Adam(model_strategy_A.parameters(), lr=0.001)

num_epochs = 10 # Keep this fixed for fair comparison

print("Training with Strategy A: Adam, lr = 0.001")
history_A = train_model_with_history(model_strategy_A, train_loader, criterion, optimizer_A, num_epochs)

# Evaluate Strategy A
print("Evaluation for Strategy A:")
acc_A, prec_A, rec_A, f1_A = evaluate_model(model_strategy_A, test_loader)

[ ]
#Strategy B: Adam, learning rate = 0.0001
# Create a fresh model instance
model_strategy_B = ImprovedCNN(num_classes=6).to(device)

```

```

Training with Strategy A: Adam, lr = 0.001
Epoch [1/10] - Loss: 1.7851, Accuracy: 0.4247
Epoch [2/10] - Loss: 1.1737, Accuracy: 0.5141
Epoch [3/10] - Loss: 1.1142, Accuracy: 0.5528
Epoch [4/10] - Loss: 1.0685, Accuracy: 0.5693
Epoch [5/10] - Loss: 1.0407, Accuracy: 0.5866
Epoch [6/10] - Loss: 0.9771, Accuracy: 0.5955
Epoch [7/10] - Loss: 0.9658, Accuracy: 0.6114
Epoch [8/10] - Loss: 0.9346, Accuracy: 0.6197

```

Figure 10: Training logs in Google Colab

### A3. Model Architecture Output

The following screenshot shows the printed summary of the improved CNN model.

Layer (type:depth-idx)	Output Shape	Param #
ImprovedCNN	[32, 5]	--
├─Conv2d: 1-1	[32, 16, 128, 128]	448
├─BatchNorm2d: 1-2	[32, 16, 128, 128]	32
├─MaxPool2d: 1-3	[32, 16, 64, 64]	--
├─Conv2d: 1-4	[32, 32, 64, 64]	4,640
├─BatchNorm2d: 1-5	[32, 32, 64, 64]	64
├─MaxPool2d: 1-6	[32, 32, 32, 32]	--
├─Conv2d: 1-7	[32, 64, 32, 32]	18,496
├─BatchNorm2d: 1-8	[32, 64, 32, 32]	128
├─MaxPool2d: 1-9	[32, 64, 16, 16]	--
├─Linear: 1-10	[32, 256]	4,194,560
├─Dropout: 1-11	[32, 256]	--
├─Linear: 1-12	[32, 5]	1,285
Total params: 4,219,653		
Trainable params: 4,219,653		
Non-trainable params: 0		
Total mult-adds (Units.GIGABYTES): 1.58		
Input size (MB): 6.29		
Forward/backward pass size (MB): 234.95		
Params size (MB): 16.88		
Estimated Total Size (MB): 258.12		

Figure 11: Improved CNN architecture summary