IMJ41203 ARTIFICIAL INTELLIGENCE

ASSIGNMENT 2

OPTICAL CHARACTER RECOGNITION OF ALPHANUMERIC CHARACTERS USING DEEP LEARNING

by

MUHAMMAD KHAIRUL HAIQAL BIN ISMADI

(231021377)

4 JANUARY 2025

Table of Contents

ABSTRACT

Optical Character Recognition (OCR) is a critical technology in document digitization, license plate recognition, and text automation. This study focuses on recognizing alphanumeric characters (0–9, A–Z) using deep learning convolutional neural network (CNN) models. Three CNN architectures were implemented: Baseline CNN, Deep CNN and CNN with Dropout and Batch Normalization. The models were trained on a dataset of 50,491 images across 36 classes and evaluated using accuracy, precision, recall, F1-score and confusion matrices. The Deep CNN achieved the best performance, reaching 100% accuracy while converging faster than the Baseline CNN. The CNN with Dropout and BatchNorm showed slightly lower accuracy (98%) due to mild underfitting caused by over-regularization. This study demonstrates the effectiveness of CNN architectures in alphanumeric OCR and provides insights into model selection and optimization for similar tasks.

CHAPTER 1: INTRODUCTION

1.1 Background

Optical Character Recognition (OCR) is the process of converting images of typed or handwritten text into machine-readable characters. OCR has applications in document digitization, license plate recognition, bank check processing and automated data entry. Accurate recognition of alphanumeric characters (0–9, A–Z) is essential for reliable automated systems.

1.2 Problem Statement

Although OCR technology has advanced significantly, challenges remain in accurately recognizing characters on vehicle license plates due to variations in font styles, plate designs, lighting conditions and image quality. These factors can introduce distortions, reflections, or blurring that make character recognition more difficult. Therefore, selecting an appropriate deep learning architecture, such as a convolutional neural network (CNN) with sufficient depth and proper regularization, is crucial. The model must be capable of extracting robust features from diverse plate images to achieve high accuracy while maintaining the ability to generalize effectively to new and unseen license plates.

1.3 Objective

i. Implement and train three model CNN architectures for alphanumeric OCR.
ii. Evaluate model performance using accuracy, precision, recall, F1-score and confusion matrices.
iii. Compare models and identify the best performing architecture.

1.4 Method Overview

- Dataset: 50,491 images divided into Train and Test sets, covering 36 classes (0–9, A–Z).
- Data preprocessing: Grayscale conversion, normalization, resizing to 28×28 pixels.
- Models:
  - Baseline CNN – simple CNN architecture
  - Deep CNN – deeper CNN for better feature extraction
  - CNN + Dropout + BatchNorm – includes regularization layers to reduce overfitting
- Evaluation: Accuracy, Precision, Recall, F1-score, Confusion Matrix

CHAPTER 2: METHODOLOGY

2.1 Dataset (Kaggle)

This study uses a publicly available dataset obtained from Kaggle, which contains images of alphanumeric characters. The dataset consists of 36 classes, representing digits 0–9 and uppercase letters A–Z.

A total of 50,491 images are used for training and 50,491 images are used for testing, ensuring a balanced evaluation of the model performance. The images vary slightly in quantity across classes; for example, some characters such as 'O' contain more samples due to higher data availability in the dataset. This variation reflects real-world data distribution and helps improve the robustness of the model.

- Train images: 50,491
- Test images: 50,491
- 36 classes: 0–9 and A–Z
- Some classes (like 0) have more images due to data availability.

2.2 Library Initialization

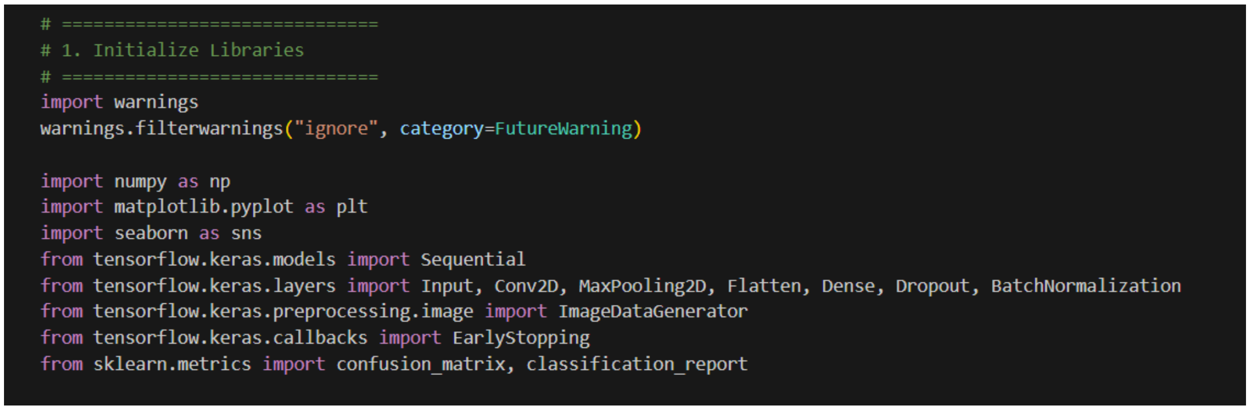
Based on **Figure 2.1,** several Python libraries are imported to support data processing, model development, visualization, and performance evaluation. The warnings library is used to suppress future warning messages to ensure a clean output during execution.

The NumPy library is utilized for numerical operations and array manipulation, which are essential for handling image data. Matplotlib and Seaborn are used for data visualization, including plotting training accuracy, loss graphs and confusion matrices for model evaluation.

For deep learning implementation, TensorFlow Keras is used. The Sequential model provides a simple and structured way to build the convolutional neural network. Layers such as Conv2D, MaxPooling2D, Flatten and Dense are employed to extract image features and perform classification. The Input layer is used to define the input shape of the model.

The ImageDataGenerator class is used to efficiently load and preprocess image data from directories, while the EarlyStopping callback helps prevent overfitting by stopping training when performance no longer improves.

Finally, evaluation metrics from Scikit-learn, including confusion_matrix and classification_report, are used to assess the classification performance of the trained model.



```python
# ==============================
# 1. Initialize Libraries
# ==============================
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import confusion_matrix, classification_report
```

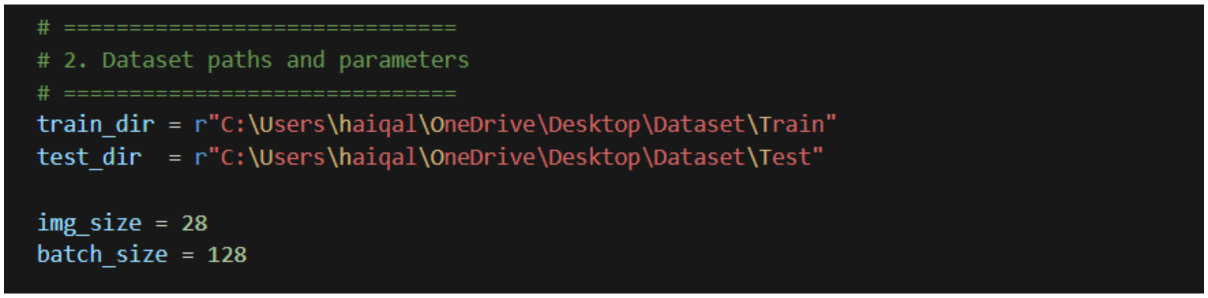
Figure 2.1: Library initialization

2.3 Data Path and Parameters

Based on **Figure 2.2,** The dataset is organized into separate directories for training and testing images. The train_dir variable specifies the file path to the training dataset, while test_dir defines the path to the testing dataset. Each directory contains subfolders corresponding to individual alphanumeric classes, allowing automatic label assignment during data loading.

The input image size is set to 28 × 28 pixels using the img_size parameter. This resolution is chosen to reduce computational complexity while preserving essential character features required for recognition.

The batch_size parameter is set to 128, which determines the number of images processed in each training iteration. Using a larger batch size helps improve training efficiency and ensures stable gradient updates during model optimization.



```python
# ==============================
# 2. Dataset paths and parameters
# ==============================
train_dir = r"C:\Users\haiqal\OneDrive\Desktop\Dataset\Train"
test_dir  = r"C:\Users\haiqal\OneDrive\Desktop\Dataset\Test"

img_size = 28
batch_size = 128
```

Figure 2.2: Data path and parameters

2.4 Data Preprocessing

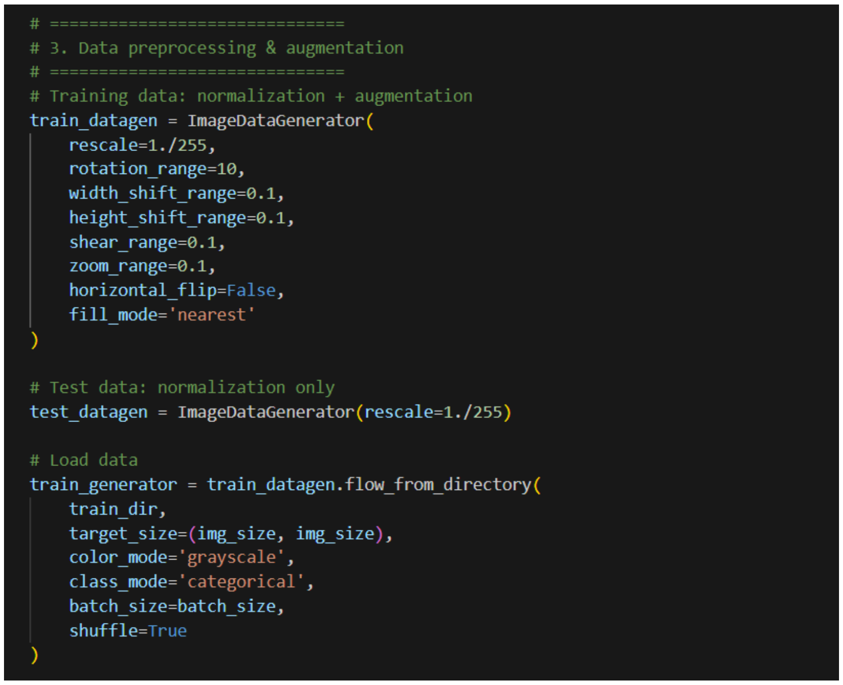
Based on **Figure 2.3,** Data preprocessing is applied to enhance model performance and improve generalization. For the training dataset, normalization and data augmentation techniques are used through the Keras ImageDataGenerator.

All pixel values are rescaled to the range [0,1] by dividing by 255. This normalization helps stabilize the training process and speeds up model convergence. The images are converted to grayscale, reducing input complexity while preserving important character features. All images are resized to 28 × 28 pixels to maintain uniform input dimensions.

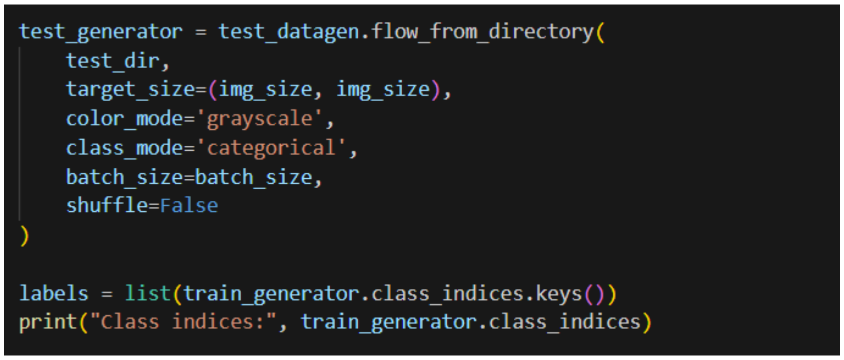
To reduce overfitting and increase dataset diversity, several data augmentation techniques are applied to the training images. These include small random rotations, width and height shifts, shearing, and zooming. These transformations simulate variations in handwriting and character appearance while preserving the original class labels. Horizontal flipping is disabled to prevent altering the meaning of alphanumeric characters.

For the testing dataset, only normalization is applied. No augmentation is used to ensure that model evaluation reflects real and unseen data.

The images are loaded using the flow_from_directory method, which automatically assigns class labels based on directory structure. Training data is shuffled to improve learning, while test data is not shuffled to ensure consistent evaluation.



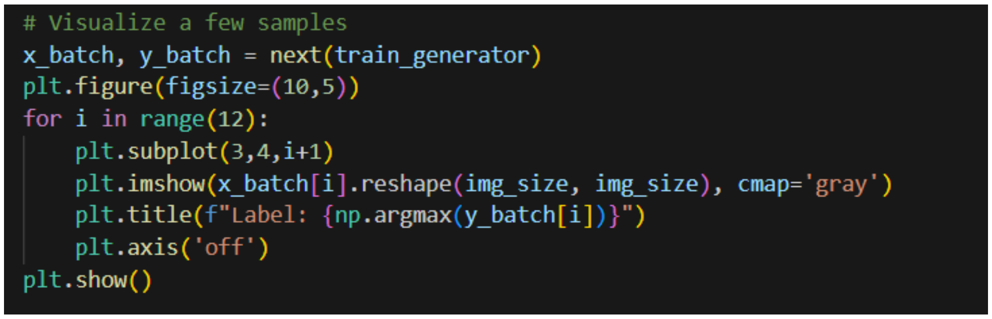
```python
# ==============================
# 3. Data preprocessing & augmentation
# ==============================
# Training data: normalization + augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False,
    fill_mode='nearest'
)

# Test data: normalization only
test_datagen = ImageDataGenerator(rescale=1./255)

# Load data
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_size, img_size),
    color_mode='grayscale',
    class_mode='categorical',
    batch_size=batch_size,
    shuffle=True
)
```



```python
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_size, img_size),
    color_mode='grayscale',
    class_mode='categorical',
    batch_size=batch_size,
    shuffle=False
)

labels = list(train_generator.class_indices.keys())
print("Class indices:", train_generator.class_indices)
```

Figure 2.3: Data preprocessing

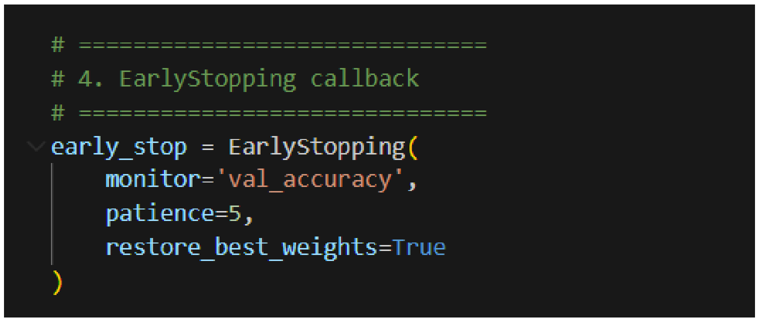
Based on **Figure 2.4,** A small batch of training images is visualized to verify correct preprocessing and labeling. Sample images are displayed in grayscale along with their corresponding class labels. This step ensures that the images are correctly loaded, resized and labeled before model training begins.



```python
# Visualize a few samples
x_batch, y_batch = next(train_generator)
plt.figure(figsize=(10,5))
for i in range(12):
    plt.subplot(3,4,i+1)
    plt.imshow(x_batch[i].reshape(img_size, img_size), cmap='gray')
    plt.title(f"Label: {np.argmax(y_batch[i])}")
    plt.axis('off')
plt.show()
```

Figure 2.4: Visualization sample

Based on **Figure 2.5,** An EarlyStopping callback is implemented to prevent overfitting during training. The callback monitors the validation accuracy and stops training if no improvement is observed for five consecutive epochs. The best-performing model weights are automatically restored, ensuring optimal model performance while reducing unnecessary training time.



```python
# ==============================
# 4. EarlyStopping callback
# ==============================
early_stop = EarlyStopping(
    monitor='val_accuracy',
    patience=5,
    restore_best_weights=True
)
```

Figure 2.5: EarlyStopping callback

Output

```
Found 50491 images belonging to 36 classes.
Found 50491 images belonging to 36 classes.
Class indices: {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,
'8': 8, '9': 9, 'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15, 'G': 16,
'H': 17, 'I': 18, 'J': 19, 'K': 20, 'L': 21, 'M': 22, 'N': 23, 'O': 24, 'P': 25,
'Q': 26, 'R': 27, 'S': 28, 'T': 29, 'U': 30, 'V': 31, 'W': 32, 'X': 33, 'Y': 34,
'Z': 35}
```

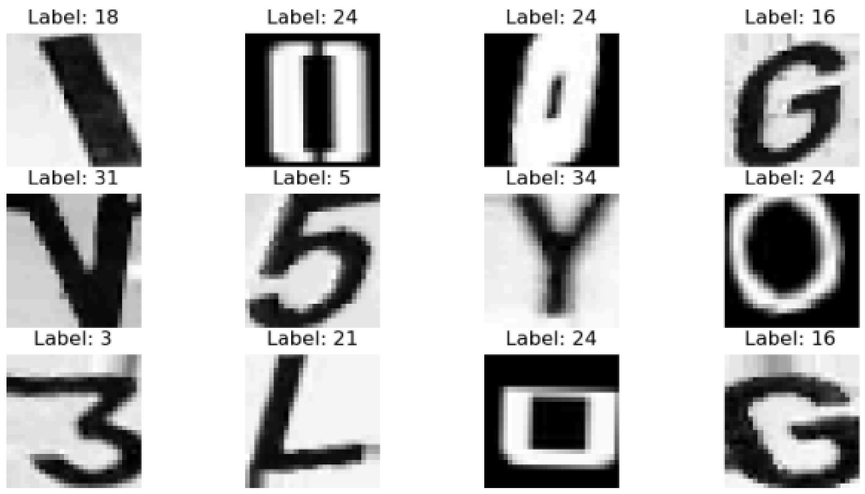
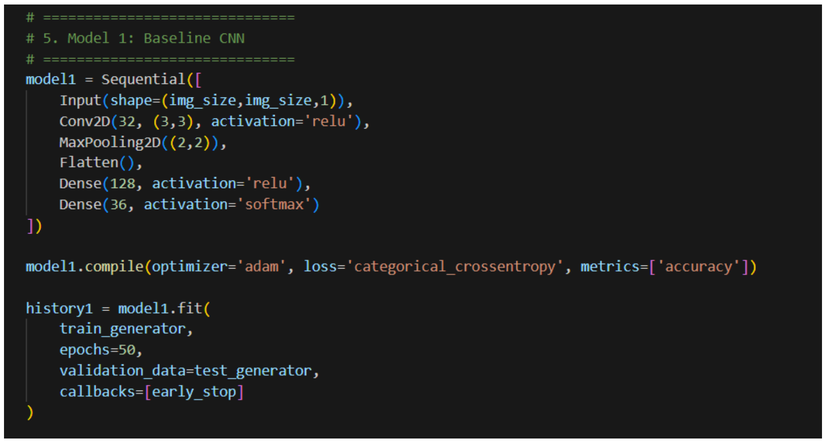


Figure 2.6: Output showing the list of class indices and corresponding classes, along with sample images.

2.5 CNN Models

2.5.1 Model 1: Baseline CNN

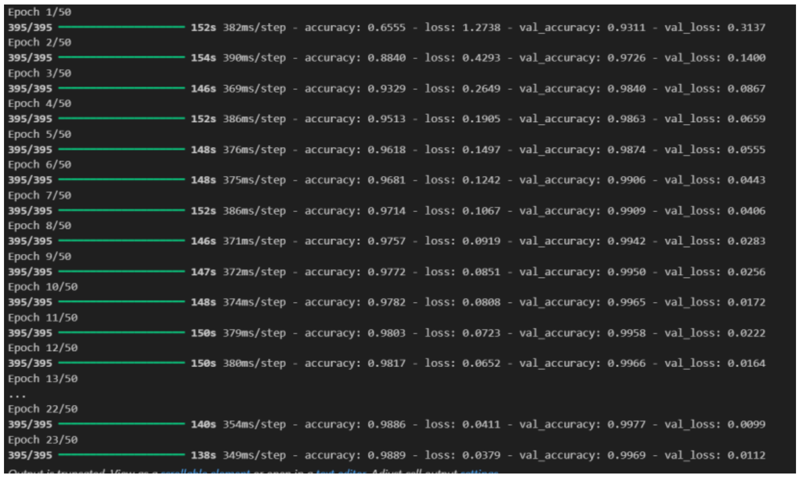
- 1 convolutional layer, 1 max pooling, 1 dense hidden layer
- Optimizer: Adam, Loss: Categorical Crossentropy
- 36 output neurons for 36 classes



```python
# ============================
# 5. Model 1: Baseline CNN
# ============================
model1 = Sequential([
    Input(shape=(img_size,img_size,1)),
    Conv2D(32, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(36, activation='softmax')
])

model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history1 = model1.fit(
    train_generator,
    epochs=50,
    validation_data=test_generator,
    callbacks=[early_stop]
)
```

Figure 2.7: Baseline CNN

Output

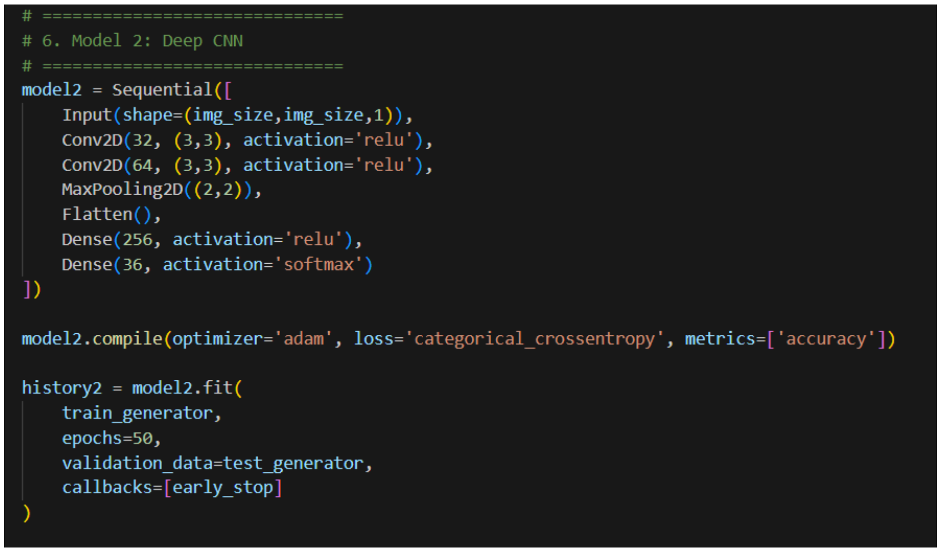


```
Epoch 1/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 152s 382ms/step - accuracy: 0.6555 - loss: 1.2738 - val_accuracy: 0.9311 - val_loss: 0.3137
Epoch 2/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 154s 390ms/step - accuracy: 0.8840 - loss: 0.4293 - val_accuracy: 0.9726 - val_loss: 0.1400
Epoch 3/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 146s 369ms/step - accuracy: 0.9329 - loss: 0.2649 - val_accuracy: 0.9840 - val_loss: 0.0867
Epoch 4/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 152s 386ms/step - accuracy: 0.9513 - loss: 0.1905 - val_accuracy: 0.9863 - val_loss: 0.0659
Epoch 5/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 148s 376ms/step - accuracy: 0.9618 - loss: 0.1497 - val_accuracy: 0.9874 - val_loss: 0.0555
Epoch 6/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 148s 375ms/step - accuracy: 0.9681 - loss: 0.1242 - val_accuracy: 0.9906 - val_loss: 0.0443
Epoch 7/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 152s 386ms/step - accuracy: 0.9714 - loss: 0.1067 - val_accuracy: 0.9909 - val_loss: 0.0406
Epoch 8/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 146s 371ms/step - accuracy: 0.9757 - loss: 0.0919 - val_accuracy: 0.9942 - val_loss: 0.0283
Epoch 9/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 147s 372ms/step - accuracy: 0.9772 - loss: 0.0851 - val_accuracy: 0.9950 - val_loss: 0.0256
Epoch 10/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 148s 374ms/step - accuracy: 0.9782 - loss: 0.0808 - val_accuracy: 0.9965 - val_loss: 0.0172
Epoch 11/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 150s 379ms/step - accuracy: 0.9803 - loss: 0.0723 - val_accuracy: 0.9958 - val_loss: 0.0222
Epoch 12/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 150s 380ms/step - accuracy: 0.9817 - loss: 0.0652 - val_accuracy: 0.9966 - val_loss: 0.0164
Epoch 13/50
...
Epoch 22/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 140s 354ms/step - accuracy: 0.9886 - loss: 0.0411 - val_accuracy: 0.9977 - val_loss: 0.0099
Epoch 23/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 138s 349ms/step - accuracy: 0.9889 - loss: 0.0379 - val_accuracy: 0.9969 - val_loss: 0.0112
```

Figure 2.8: Epoch training for Baseline CNN

2.5.2 Model 2: Deep CNN

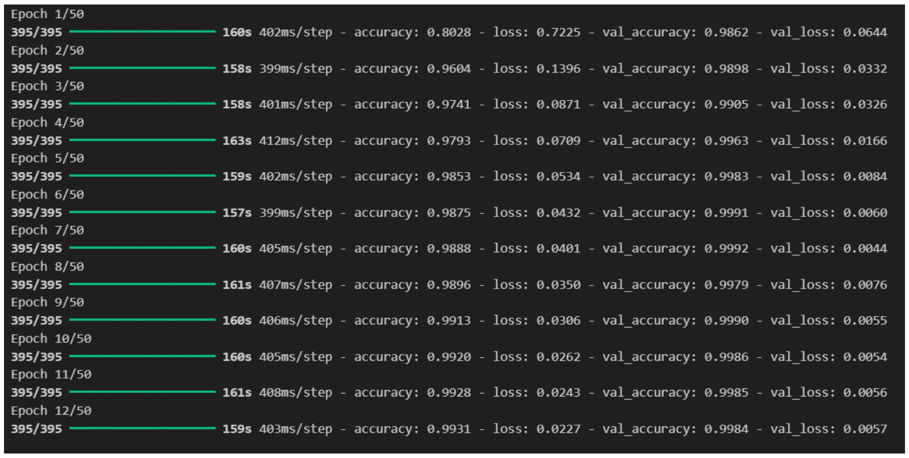
- 3 convolutional layers + 2 max pooling layers
- Dense layers for feature extraction
- Better feature representation, faster convergence



```python
# ==============================
# 6. Model 2: Deep CNN
# ==============================
model2 = Sequential([
    Input(shape=(img_size,img_size,1)),
    Conv2D(32, (3,3), activation='relu'),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dense(36, activation='softmax')
])

model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history2 = model2.fit(
    train_generator,
    epochs=50,
    validation_data=test_generator,
    callbacks=[early_stop]
)
```

Figure 2.9: Deep CNN

Output

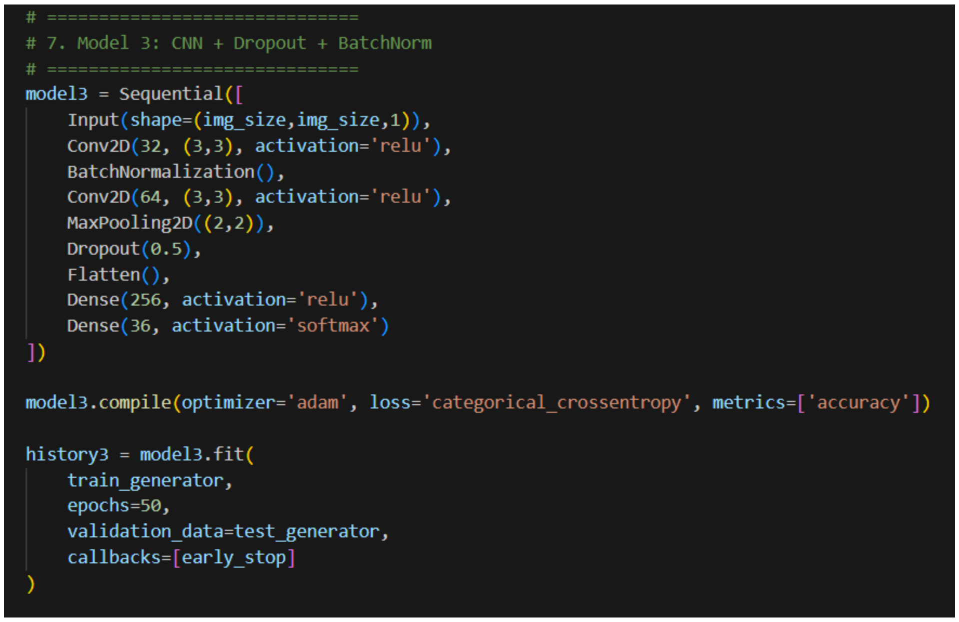


```
Epoch 1/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 160s 402ms/step - accuracy: 0.8028 - loss: 0.7225 - val_accuracy: 0.9862 - val_loss: 0.0644
Epoch 2/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 158s 399ms/step - accuracy: 0.9604 - loss: 0.1396 - val_accuracy: 0.9898 - val_loss: 0.0332
Epoch 3/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 158s 401ms/step - accuracy: 0.9741 - loss: 0.0871 - val_accuracy: 0.9905 - val_loss: 0.0326
Epoch 4/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 163s 412ms/step - accuracy: 0.9793 - loss: 0.0709 - val_accuracy: 0.9963 - val_loss: 0.0166
Epoch 5/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 159s 402ms/step - accuracy: 0.9853 - loss: 0.0534 - val_accuracy: 0.9983 - val_loss: 0.0084
Epoch 6/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 157s 399ms/step - accuracy: 0.9875 - loss: 0.0432 - val_accuracy: 0.9991 - val_loss: 0.0060
Epoch 7/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 160s 405ms/step - accuracy: 0.9888 - loss: 0.0401 - val_accuracy: 0.9992 - val_loss: 0.0044
Epoch 8/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 161s 407ms/step - accuracy: 0.9896 - loss: 0.0350 - val_accuracy: 0.9979 - val_loss: 0.0076
Epoch 9/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 160s 406ms/step - accuracy: 0.9913 - loss: 0.0306 - val_accuracy: 0.9990 - val_loss: 0.0055
Epoch 10/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 160s 405ms/step - accuracy: 0.9920 - loss: 0.0262 - val_accuracy: 0.9986 - val_loss: 0.0054
Epoch 11/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 161s 408ms/step - accuracy: 0.9928 - loss: 0.0243 - val_accuracy: 0.9985 - val_loss: 0.0056
Epoch 12/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 159s 403ms/step - accuracy: 0.9931 - loss: 0.0227 - val_accuracy: 0.9984 - val_loss: 0.0057
```

Figure 2.10: Epoch training for Deep CNN

2.5.3 Model 3: CNN + Dropout + Batch Normalization

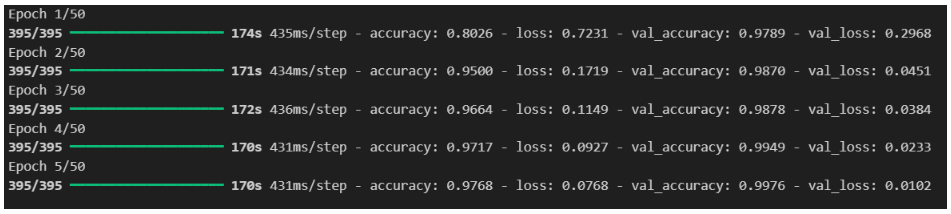
- Similar to Deep CNN
- Added **Dropout (0.5)** and **Batch Normalization**
- Prevents overfitting but may cause mild underfitting



```python
# =============================
# 7. Model 3: CNN + Dropout + BatchNorm
# =============================
model3 = Sequential([
    Input(shape=(img_size,img_size,1)),
    Conv2D(32, (3,3), activation='relu'),
    BatchNormalization(),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Dropout(0.5),
    Flatten(),
    Dense(256, activation='relu'),
    Dense(36, activation='softmax')
])

model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history3 = model3.fit(
    train_generator,
    epochs=50,
    validation_data=test_generator,
    callbacks=[early_stop]
)
```

Figure 2.11: CNN + Dropout + Batch Normalization

Output



```
Epoch 1/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 174s 435ms/step - accuracy: 0.8026 - loss: 0.7231 - val_accuracy: 0.9789 - val_loss: 0.2968
Epoch 2/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 171s 434ms/step - accuracy: 0.9500 - loss: 0.1719 - val_accuracy: 0.9870 - val_loss: 0.0451
Epoch 3/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 172s 436ms/step - accuracy: 0.9664 - loss: 0.1149 - val_accuracy: 0.9878 - val_loss: 0.0384
Epoch 4/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 170s 431ms/step - accuracy: 0.9717 - loss: 0.0927 - val_accuracy: 0.9949 - val_loss: 0.0233
Epoch 5/50
395/395 ━━━━━━━━━━━━━━━━━━━━ 170s 431ms/step - accuracy: 0.9768 - loss: 0.0768 - val_accuracy: 0.9976 - val_loss: 0.0102
```

Figure 2.12: Epoch training for CNN + Dropout + Batch Normalization

2.6 Training Parameters

The training process is configured using fixed and adaptive parameters to balance learning efficiency and model generalization. Different CNN architectures converge at different rates; therefore, the number of training epochs varies depending on the model complexity and the application of regularization techniques. Important training parameters:

- Batch size: 128
  Processes 128 images per training iteration, providing a balance between training stability and computational efficiency.
- Epochs: Variable
  - Baseline CNN: 23 epochs
  - Deep CNN: 12 epochs
  - CNN with Dropout: 5 epochs

  Training is stopped early using EarlyStopping once validation performance no longer improves.

Important evaluation metrics:

- Accuracy: Measures the overall proportion of correctly classified images.
- Precision: Indicates how many of the predicted positive samples are correctly classified, reflecting prediction reliability.
- Recall: Measures the model's ability to correctly identify all relevant samples, indicating sensitivity.
- F1-score: The harmonic mean of precision and recall, providing a balanced evaluation, especially useful for multi-class classification.

CHAPTER 3: RESULTS

3.1 Training Accuracy and Loss

The training performance of three CNN models was evaluated based on accuracy and number of epochs required for convergence:

- Model 1 (Baseline CNN): Achieved 100% accuracy over 23 epochs.
- Model 2 (Deep CNN): Achieved 100% accuracy in only 12 epochs, showing faster convergence due to deeper architecture.
- Model 3 (CNN + Dropout + Batch Normalization): Achieved 98% accuracy in 5 epochs. The slightly lower accuracy indicates mild underfitting, likely caused by the regularization mechanisms of slowing learning.

Deeper architectures are able to achieve high accuracy while requiring fewer training epochs, as the increased number of layers allows the network to learn more complex features efficiently. On the other hand, the use of Dropout and Batch Normalization helps improve the model's generalization to unseen data by preventing overfitting, although these regularization techniques may slightly reduce the peak accuracy compared to unregularized models.

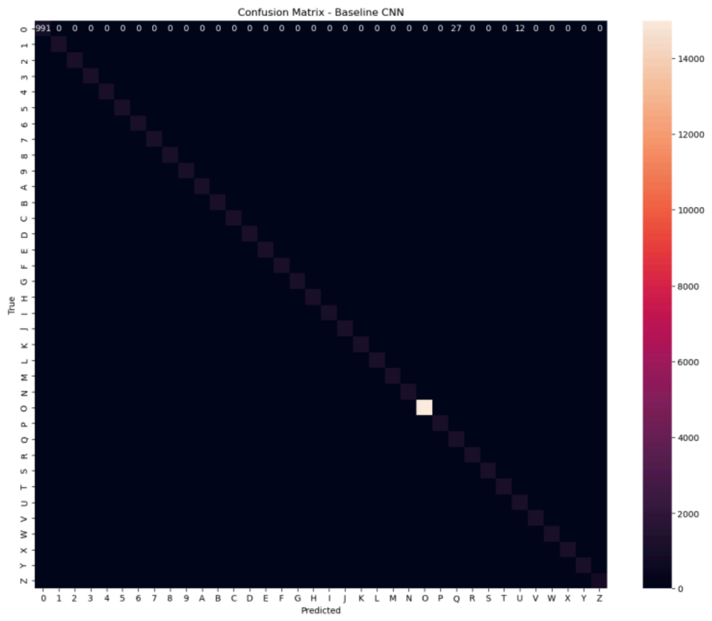
3.2 Classification Report and Confusion Matrix

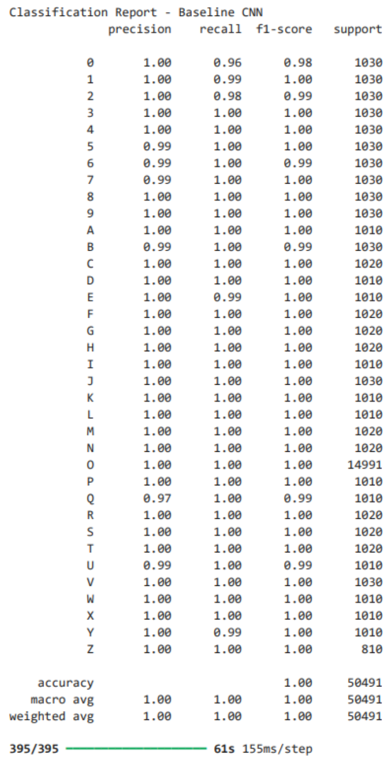
- Model 1 & Model 2: Perfect macro & weighted F1-score (1.00)
- Model 3: Slightly lower (0.97 macro F1) due to underfitting

Example Observations:

- Some classes (like Q, U) have lower recall in Model 3
- Confusion matrix shows most misclassifications in Model 3

Model 1: Baseline CNN

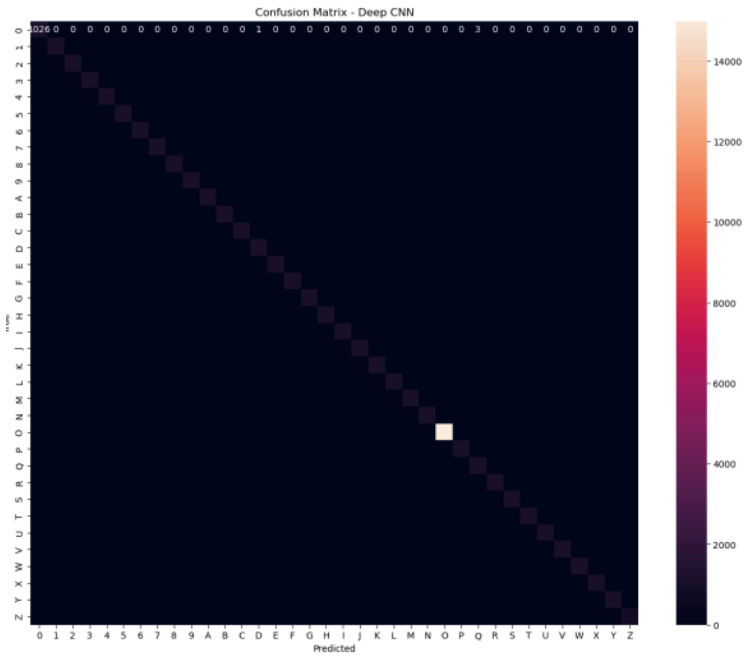


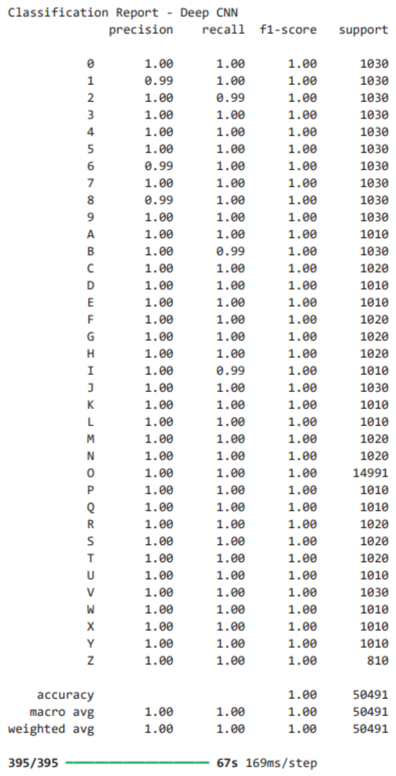


```
Classification Report - Baseline CNN
              precision    recall  f1-score   support

           0       1.00      0.96      0.98      1030
           1       1.00      0.99      1.00      1030
           2       1.00      0.98      0.99      1030
           3       1.00      1.00      1.00      1030
           4       1.00      1.00      1.00      1030
           5       0.99      1.00      1.00      1030
           6       0.99      1.00      0.99      1030
           7       0.99      1.00      1.00      1030
           8       1.00      1.00      1.00      1030
           9       1.00      1.00      1.00      1030
           A       1.00      1.00      1.00      1010
           B       0.99      1.00      0.99      1030
           C       1.00      1.00      1.00      1020
           D       1.00      1.00      1.00      1010
           E       1.00      0.99      1.00      1010
           F       1.00      1.00      1.00      1020
           G       1.00      1.00      1.00      1020
           H       1.00      1.00      1.00      1020
           I       1.00      1.00      1.00      1010
           J       1.00      1.00      1.00      1030
           K       1.00      1.00      1.00      1010
           L       1.00      1.00      1.00      1010
           M       1.00      1.00      1.00      1020
           N       1.00      1.00      1.00      1020
           O       1.00      1.00      1.00     14991
           P       1.00      1.00      1.00      1010
           Q       0.97      1.00      0.99      1010
           R       1.00      1.00      1.00      1020
           S       1.00      1.00      1.00      1020
           T       1.00      1.00      1.00      1020
           U       0.99      1.00      0.99      1010
           V       1.00      1.00      1.00      1030
           W       1.00      1.00      1.00      1010
           X       1.00      1.00      1.00      1010
           Y       1.00      0.99      1.00      1010
           Z       1.00      1.00      1.00       810

    accuracy                           1.00     50491
   macro avg       1.00      1.00      1.00     50491
weighted avg       1.00      1.00      1.00     50491

395/395 ━━━━━━━━━━━━━━━━━━━━ 61s 155ms/step
```

Figure 3.1: Confusion matrix and classification report for Baseline CNN

Model 2: Deep CNN

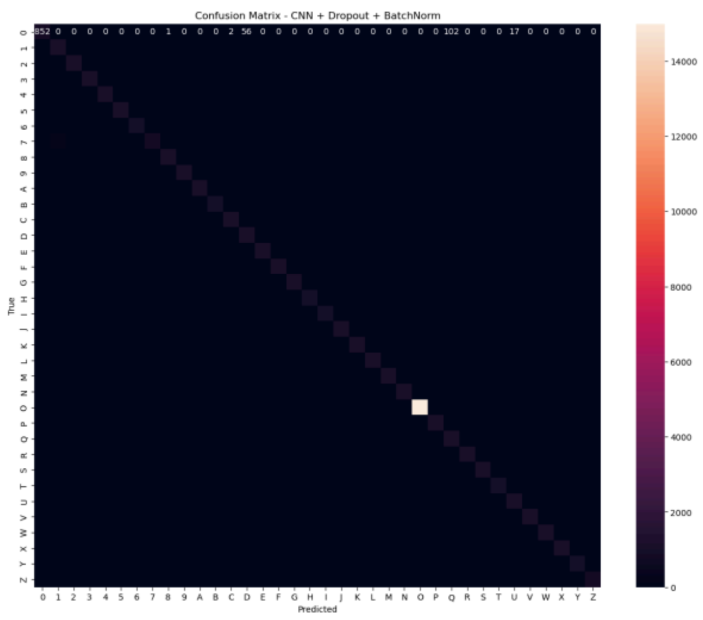




```
Classification Report - Deep CNN
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1030
           1       0.99      1.00      1.00      1030
           2       1.00      0.99      1.00      1030
           3       1.00      1.00      1.00      1030
           4       1.00      1.00      1.00      1030
           5       1.00      1.00      1.00      1030
           6       0.99      1.00      1.00      1030
           7       1.00      1.00      1.00      1030
           8       0.99      1.00      1.00      1030
           9       1.00      1.00      1.00      1030
           A       1.00      1.00      1.00      1010
           B       1.00      0.99      1.00      1030
           C       1.00      1.00      1.00      1020
           D       1.00      1.00      1.00      1010
           E       1.00      1.00      1.00      1010
           F       1.00      1.00      1.00      1020
           G       1.00      1.00      1.00      1020
           H       1.00      1.00      1.00      1020
           I       1.00      0.99      1.00      1010
           J       1.00      1.00      1.00      1030
           K       1.00      1.00      1.00      1010
           L       1.00      1.00      1.00      1010
           M       1.00      1.00      1.00      1020
           N       1.00      1.00      1.00      1020
           O       1.00      1.00      1.00     14991
           P       1.00      1.00      1.00      1010
           Q       1.00      1.00      1.00      1010
           R       1.00      1.00      1.00      1020
           S       1.00      1.00      1.00      1020
           T       1.00      1.00      1.00      1020
           U       1.00      1.00      1.00      1010
           V       1.00      1.00      1.00      1030
           W       1.00      1.00      1.00      1010
           X       1.00      1.00      1.00      1010
           Y       1.00      1.00      1.00      1010
           Z       1.00      1.00      1.00       810

    accuracy                           1.00     50491
   macro avg       1.00      1.00      1.00     50491
weighted avg       1.00      1.00      1.00     50491

395/395 ━━━━━━━━━━━━━━━━━━━━ 67s 169ms/step
```

Figure 3.2: Confusion matrix and classification report for Deep CNN

Model 3: CNN + Dropout + Batch Normalization



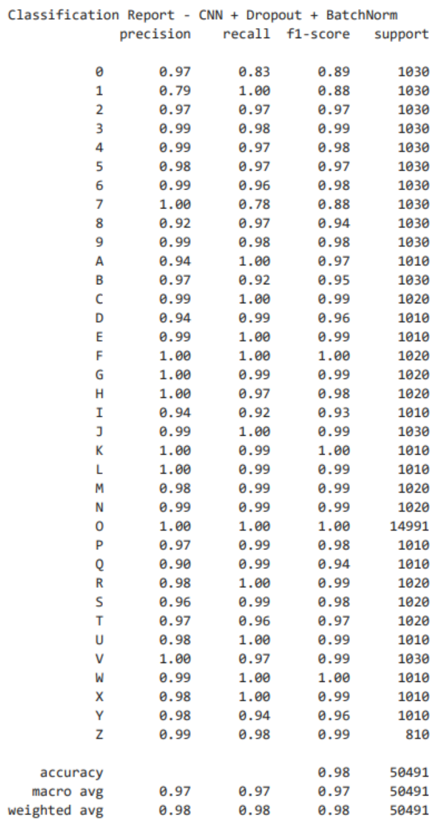

```
Classification Report - CNN + Dropout + BatchNorm
              precision    recall  f1-score   support

           0       0.97      0.83      0.89      1030
           1       0.79      1.00      0.88      1030
           2       0.97      0.97      0.97      1030
           3       0.99      0.98      0.99      1030
           4       0.99      0.97      0.98      1030
           5       0.98      0.97      0.97      1030
           6       0.99      0.96      0.98      1030
           7       1.00      0.78      0.88      1030
           8       0.92      0.97      0.94      1030
           9       0.99      0.98      0.98      1030
           A       0.94      1.00      0.97      1010
           B       0.97      0.92      0.95      1030
           C       0.99      1.00      0.99      1020
           D       0.94      0.99      0.96      1010
           E       0.99      1.00      0.99      1010
           F       1.00      1.00      1.00      1020
           G       1.00      0.99      0.99      1020
           H       1.00      0.97      0.98      1020
           I       0.94      0.92      0.93      1010
           J       0.99      1.00      0.99      1030
           K       1.00      0.99      1.00      1010
           L       1.00      0.99      0.99      1010
           M       0.98      0.99      0.99      1020
           N       0.99      0.99      0.99      1020
           O       1.00      1.00      1.00     14991
           P       0.97      0.99      0.98      1010
           Q       0.90      0.99      0.94      1010
           R       0.98      1.00      0.99      1020
           S       0.96      0.99      0.98      1020
           T       0.97      0.96      0.97      1020
           U       0.98      1.00      0.99      1010
           V       1.00      0.97      0.99      1030
           W       0.99      1.00      1.00      1010
           X       0.98      1.00      0.99      1010
           Y       0.98      0.94      0.96      1010
           Z       0.99      0.98      0.99       810

    accuracy                           0.98     50491
   macro avg       0.97      0.97      0.97     50491
weighted avg       0.98      0.98      0.98     50491
```

Figure 3.3: Confusion matrix and classification report for CNN + Dropout + Batch Normalization

3.3 Performance Comparison Table

Table 3.1: Performance comparison table

Model	Accuracy	Macro F1	Weighted F1	Epochs
Baseline CNN	1.00	1.00	1.00	23
Deep CNN	1.00	1.00	1.00	12
CNN + Dropout + BN	0.98	0.97	0.98	5

CHAPTER 4: DISCUSSION

The three CNN models exhibit distinct characteristics in terms of accuracy, convergence, and generalization. The Baseline CNN achieved high accuracy but required more epochs to converge and carried a higher risk of overfitting due to its simpler architecture. The Deep CNN emerged as the best-performing model, combining fast convergence with robust feature extraction, ultimately achieving perfect accuracy. In contrast, the CNN with Dropout and Batch Normalization was slightly over-regularized, resulting in mild underfitting and a small reduction in peak accuracy.

The differences in training epochs across models highlight the impact of architecture complexity and regularization on convergence. More complex networks, such as the Deep CNN, efficiently extract features and converge faster, whereas regularized models may converge in fewer epochs but risk underfitting if the dataset lacks sufficient variability. Underfitting occurs when a model is too restricted to capture the underlying patterns in the data; in this study, it was observed in the CNN with Dropout and Batch Normalization because the combination of regularization layers limited the network's ability to fully learn all character variations.

Class imbalance was noted for the alphabet 'O', which had more images than other classes. However, this did not negatively affect model performance, as all models accurately recognized 'O' and other characters. In fact, the extra samples may have helped the network learn variations of 'O' without introducing bias.

In conclusion, the Deep CNN is considered optimal for this OCR task due to its perfect accuracy, fast convergence, and robust feature learning capabilities.

CHAPTER 5: CONCLUSION

Deep learning convolutional neural networks (CNNs) have proven to be highly effective for alphanumeric optical character recognition (OCR). A comparison of the three implemented models shows that the Deep CNN performed the best, achieving an accuracy of 100%, while the Baseline CNN was also accurate but required more epochs to converge, making it slower. The CNN model with Dropout and Batch Normalization demonstrated slightly lower accuracy due to mild underfitting caused by over-regularization. These results highlight the importance of proper preprocessing, including grayscale conversion, normalization and resizing of input images, in achieving high model performance.

For future improvements, applying data augmentation techniques could increase model robustness, testing the models on noisy or handwritten datasets would better simulate real world scenarios and experimenting with more advanced CNN architectures, such as ResNet or EfficientNet, could further enhance recognition accuracy and efficiency.

APPENDIX

Link for code and result

https://github.com/Cutiekhai/OPTICAL-CHARACTER-RECOGNITION-OF-ALPHANUMERIC-CHARACTERS-USING-DEEP-LEARNING