Optical Character Recognition (OCR) Project Documentation

Project Overview:

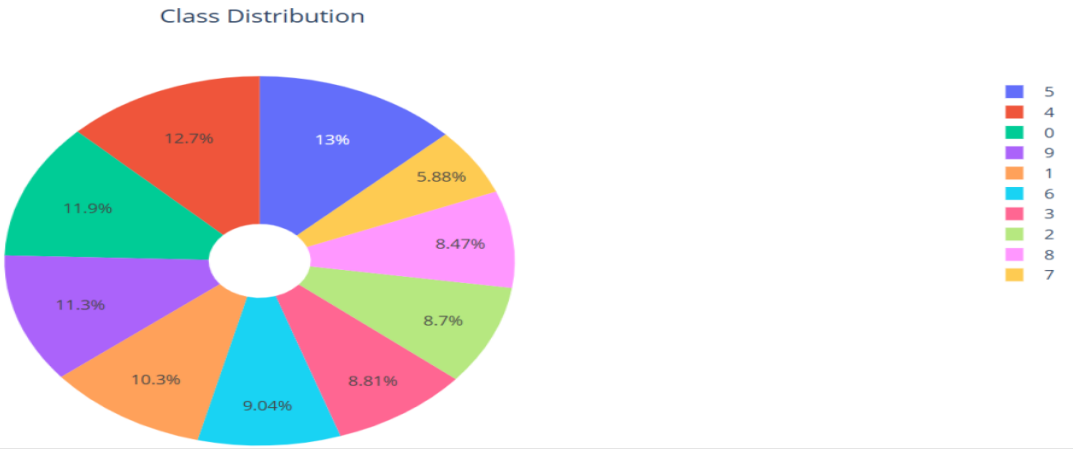
This project involves building an Optical Character Recognition (OCR) system to classify handwritten or printed digits. The OCR system utilizes a deep learning model to recognize and classify digits (0–9) based on a dataset of digital images. The solution has been designed to process input images, predict the digit, and provide the classification probability.

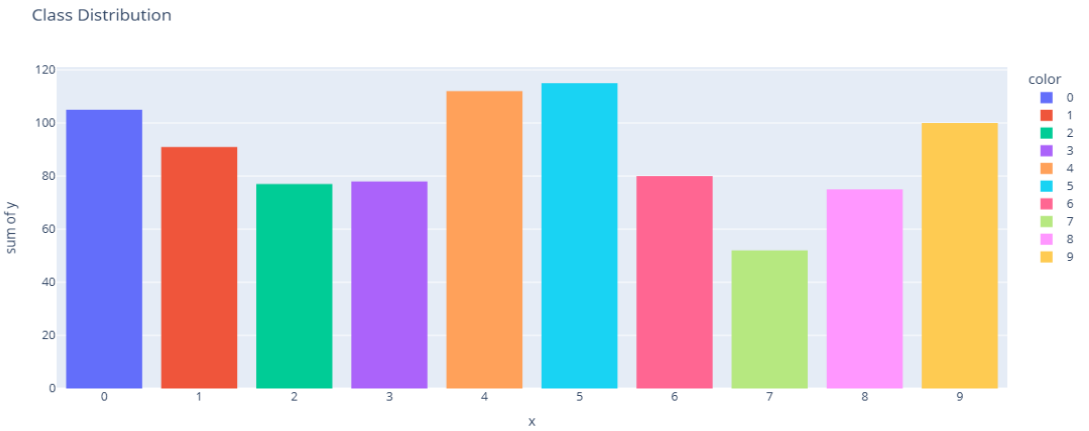
Dataset Description:

Dataset Overview

The dataset consists of 885 images, representing digits 0–9. Each image belongs to one of 10 classes, and the dataset is imbalanced, with the number of images per class varying significantly.

Class Distribution





Key Observations:

- The dataset is imbalanced, with class "7" having the least images (52) and class "5" having the most (115).
- The imbalance may affect model training and evaluation if not addressed properly.

Image Characteristics

1. Format: Images are in standard formats such as .jpg
2. Dimensions: Images will be resized to a fixed size (e.g., 28x28 pixels) during preprocessing.
3. Color Channels: Images are expected to be grayscale; if in RGB, they are converted to grayscale for simplicity.

Model Architecture

The OCR classification system is built using a Convolutional Neural Network (CNN). CNNs are well-suited for image recognition tasks due to their ability to capture spatial patterns in images.

Model Details

1. **Input Layer:**
   - Accepts images of size 28x28 (after preprocessing).
2. **Convolutional Layers:**
   - Extract spatial features using multiple filters.
3. **Pooling Layers:**
   - Reduce dimensionality while retaining important features.
4. **Drop out Layers:**
   - Dropout reduces the reliance on specific neurons, preventing the network from over-memorizing the training data. This improves its ability to generalize to unseen data &prevent overfitting.
5. **Fully Connected Layers:**
   - Perform classification based on extracted features.
6. **Output Layer:**
   - Outputs probabilities for each of the 10 classes (digits 0–9).

Compilation

- **Loss Function:** Categorical Crossentropy (suitable for multi-class classification tasks).
- **Optimizer:** Adam optimizer for efficient training.
- **Metrics:** Accuracy and precision.

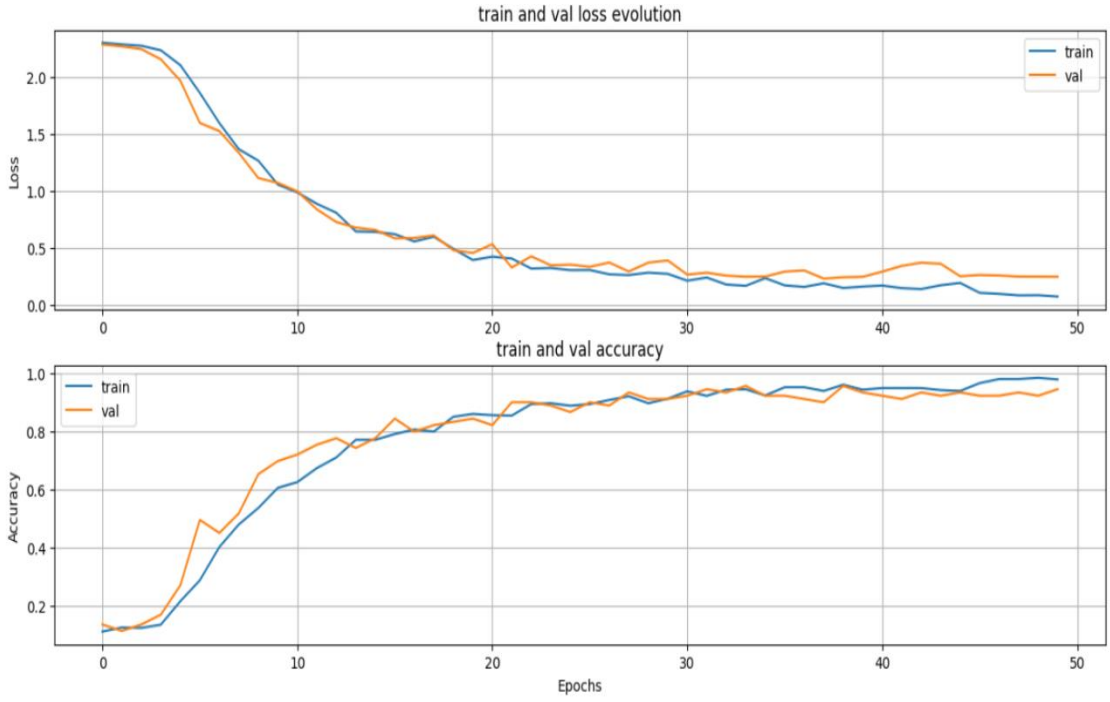
Training

- **Model_fit :** with Batch_sicze=32 ,epochs=50 , call back for early stopping .

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout_3 (Dropout)	(None, 13, 13, 32)	0
conv2d_4 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_4 (Dropout)	(None, 5, 5, 64)	0
conv2d_5 (Conv2D)	(None, 3, 3, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 128)	0
dropout_5 (Dropout)	(None, 1, 1, 128)	0
flatten_1 (Flatten)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8,256
dense_5 (Dense)	(None, 128)	8,320
dense_6 (Dense)	(None, 64)	8,256
dense_7 (Dense)	(None, 10)	650

Total params: 118,154 (461.54 KB)
Trainable params: 118,154 (461.54 KB)
Non-trainable params: 0 (0.00 B)



Evaluation Metrics

To assess the performance of the OCR system, the following metrics are used:

1. **Accuracy:**
   - Measures the proportion of correctly classified digits.
2. **Precision, Recall, and F1-Score:**
   - Evaluate the model's performance for each digit class, especially important for imbalanced datasets.
3. **Confusion Matrix:**
   - Provides a detailed breakdown of true positives, false positives, true negatives, and false negatives for each class.
4. **ROC Curve and AUC for multiclass**:

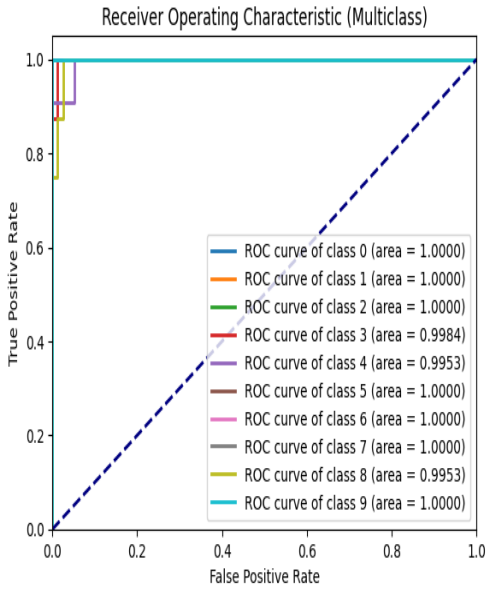
```
_, acc = loaded_model.evaluate(x_test,
                               y_test,
                               verbose=0)
print("Accuracy_Testing: %.1f%%n" % (100.0 * acc))
```

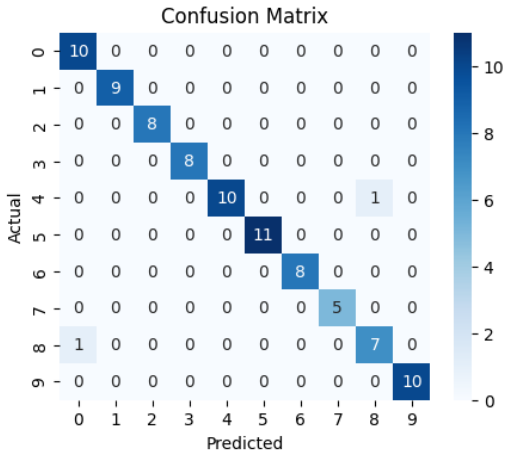
```
Accuracy_Testing: 97.7%n
```

```
_, acc = loaded_model.evaluate(x_train,
                               y_train,
                               verbose=0)
print("Accuracy_Training: %.1f%%n" % (100.0 * acc))
```

```
Accuracy_Training: 99.6%n
```

	precision	recall	f1-score	support
0	0.9091	1.0000	0.9524	10
1	1.0000	1.0000	1.0000	9
2	1.0000	1.0000	1.0000	8
3	1.0000	1.0000	1.0000	8
4	1.0000	0.9091	0.9524	11
5	1.0000	1.0000	1.0000	11
6	1.0000	1.0000	1.0000	8
7	1.0000	1.0000	1.0000	5
8	0.8750	0.8750	0.8750	8
9	1.0000	1.0000	1.0000	10
accuracy			0.9773	88
macro avg	0.9784	0.9784	0.9780	88
weighted avg	0.9783	0.9773	0.9773	88





Implementation Details:

Project Workflow

1. **Data Preprocessing:**
   - Normalize pixel values to range [0, 1].
   - Resize images to 28x28 pixels.
   - Augment the dataset to address imbalance.
2. **Model Training:**
   - Split the dataset into training, validation, and test sets.
   - Train the CNN using the training data.
   - Validate the model on the validation set to tune hyperparameters.
3. **Model Evaluation:**
   - Evaluate the model on the test set using accuracy, precision, recall, and F1-score.
4. **Deployment:**
   - Integrate the trained model into a Django web application for digit classification.

Key Libraries and Tools

- **TensorFlow/Keras:** For building and training the CNN.
- **OpenCV:** For image processing.
- **Django:** For web application development.
- **Bootstrap:** For frontend styling.

Web Application:

The OCR system is deployed using a Django-based web application that provides an intuitive interface for users to upload images and view predictions.

Frontend Details

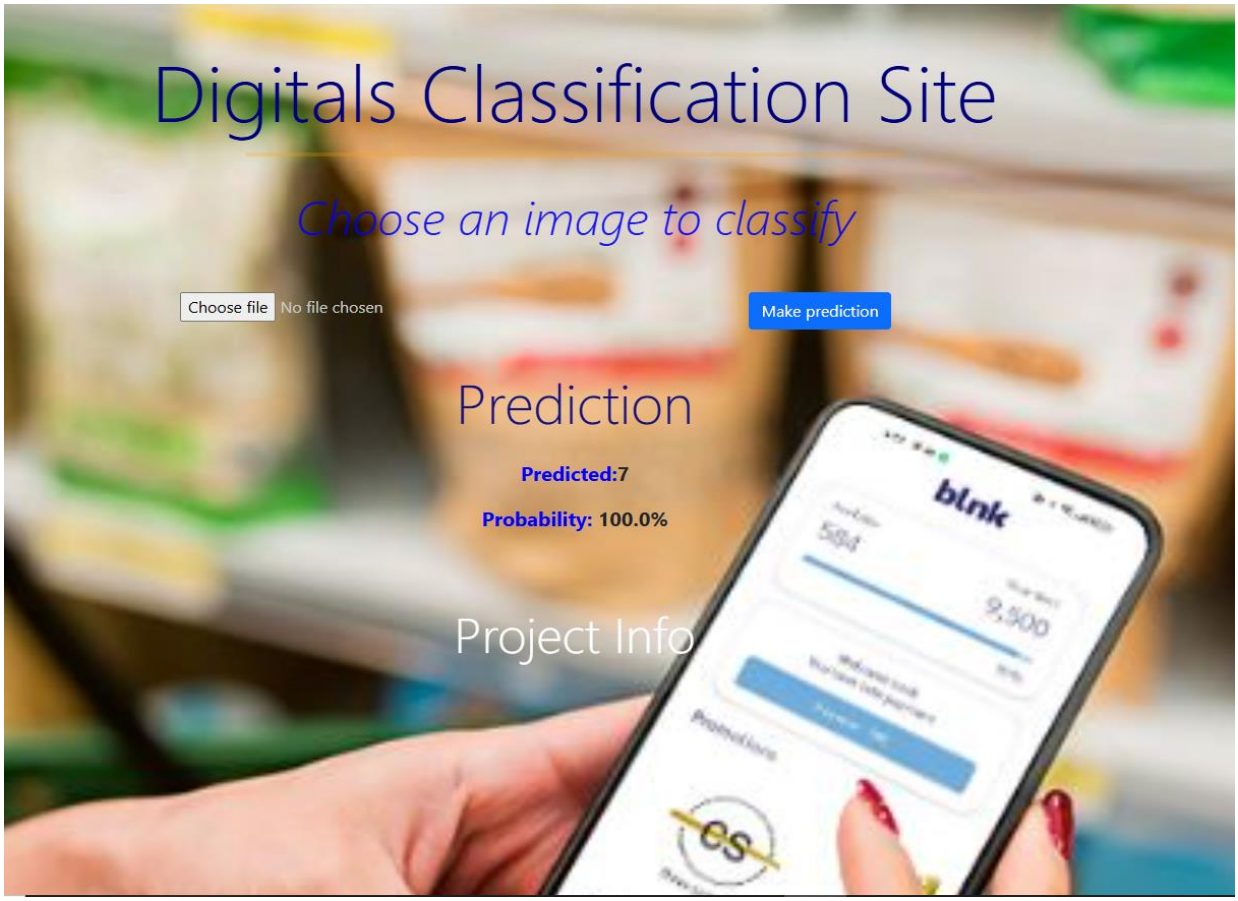
- **HTML/CSS:** Used to design the user interface.
- **Bootstrap:** Integrated for responsive and visually appealing styling.

Backend Details

- **Django Framework:** Handles the business logic and integrates the trained CNN model for predictions.
- **Endpoints:**
  - **/upload:** Allows users to upload an image.
  - **/predict:** Processes the uploaded image and returns the prediction.

Example User Flow:

1. **User uploads an image of a handwritten or printed digit.**
2. **The image is processed by the backend, and the trained model predicts the digit.**
3. **The result is displayed on the webpage, showing:**
    - Predicted Digit
    - Probability



Back to prediction page

TensorFlow with Django

In the digital transformation process,companies often deal with large volumes of documents that require data entry, which is both time-consuming and error-prone. The goal is to automate this process by This OCR digits classification project leverages a Convolutional Neural Network (CNN) with TensorFlow to recognize printed digits with high accuracy.With a training accuracy of 99.6% and validation accuracy of 97.7%,The dataset consists of 885 digital images across 10 classes 0,1,2,3,4,5,6,7,8,9 ,trained and tested for robust prediction. The application processes input images and provides real-time predictions with probabilities through a user-friendly interface.

This is simple demo about Digital classification using Artificial Intelligence.

Project Instructions

- Install Project Requirements: pip install -r requirements.txt
- Run python manage.py migrate
- Run python manage.py collectstatic
- Run the application: python manage.py runserver
- And finally, copy and paste this link in your browser http://127.0.0.1:8000/

Clone or download project from my repo

You can Try my application with follow these steps:

1. First select any image of printed digits by press on choose file Button
2. Then press on Make Prediction Button
3. Then result of prediction will display as you seen in image
4. Then you can press on Project Info to learn more about my app
5. this will transfer you into TensorFlow With Django page (
   in this page you can follow these instructions and access my repo from link repo in the bottom
6. last you can return to home page by press on Back to prediction page

Challenges and Solutions:

1. **Class Imbalance:**
   - Solution: Applied data augmentation and class weighting during model training.
2. **Ambiguity Between Digits:**
   - Example: Digits "6" and "9" may be visually similar.
   - Solution: Enhanced model robustness through augmentation and deeper architectures.
3. **Deployment Issues:**
   - Solution: Integrated the TensorFlow model into Django seamlessly using best practices.

Future Improvements:

1. **Expand Dataset:**
   - Include more images to improve generalization.
2. **Advanced Architectures:**
   - Experiment with state-of-the-art models like ResNet or EfficientNet.
3. **Additional Features:**
   - Add multilingual OCR support for characters in other languages.

Conclusion:

This OCR project demonstrates the effective use of deep learning to classify handwritten or printed digits. By addressing challenges such as class imbalance and integrating the model into a web application, the project offers a practical solution for digit recognition tasks.