OCR Improvement and Experimentation Report

MUSAB BIN JAMIL – 29409

HUSSAIN DIWAN – 29410

1. Introduction

This report outlines our exploration and development process for Optical Character Recognition (OCR) technology. The project is divided into two phases:

1. Developing and training a deep learning model for OCR, starting with PyTesseract as a baseline and training our own OCR model.
2. Enhancing OCR performance by integrating Azure OCR API and using GPT-2 from Hugging Face for generative post-processing.

We also document our journey, including iterations, miss-steps and lessons learned, providing insights into the challenges and successes encountered throughout.

---

2. Phase 1: Training a Deep Learning Model for OCR

2.1. PyTesseract as Baseline

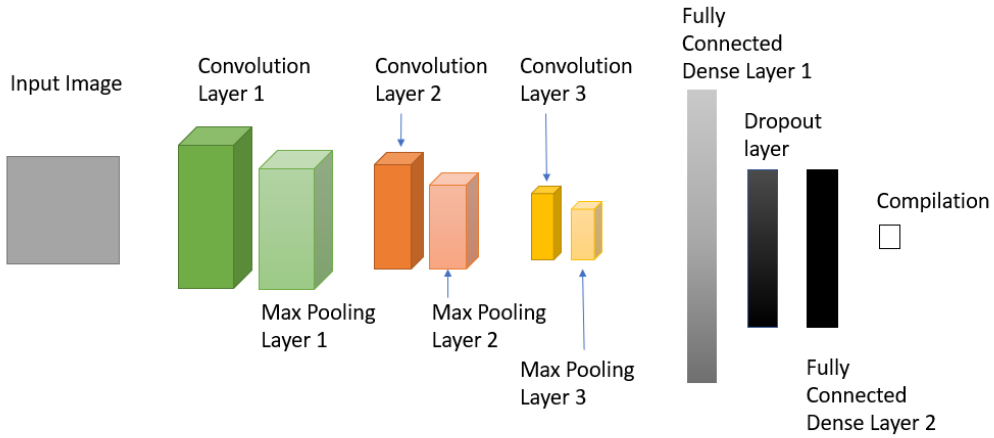
We began by using PyTesseract, a Python wrapper for Google's Tesseract OCR, as a baseline for OCR tasks. This provided a reference point for accuracy and performance. Our initial goal was to match the performance of PyTesseract with our own model in order to gain deeper understanding on OCR models.

2.2. Building a Custom Deep Learning Model

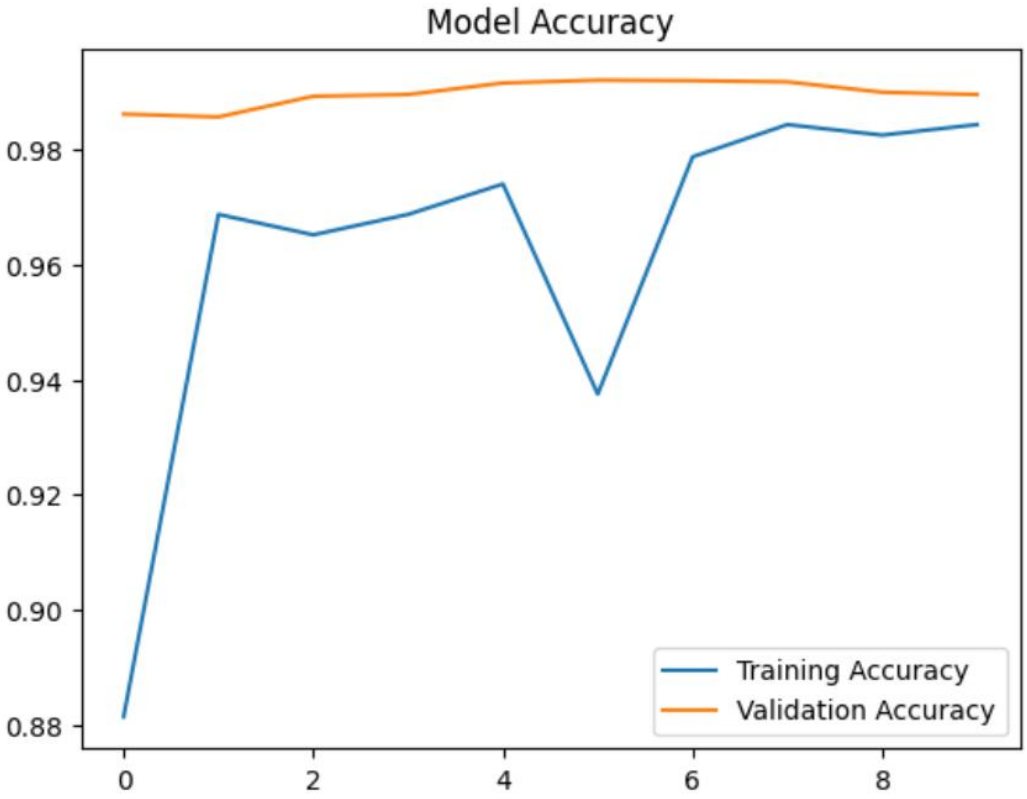
To develop said understanding of PyTesseract, we trained a deep learning model tailored to our specific OCR needs:

- **Dataset Preparation:** We used MNIST's database which consisted of a diverse dataset containing handwritten, printed, and multi-lingual text. It also consisted of an easily accessible python implementation.

- **Model Architecture:** We developed a Convolutional Neural Network (CNN) model for our image classification task, particularly targeting grayscale images such as handwritten digits from the MNIST dataset. The model consists of three convolutional layers with increasing filter sizes (32, 64, 128) and ReLU activation functions, which extract hierarchical features from the input images. Each convolutional layer is followed by a max-pooling layer to reduce spatial dimensions and computational complexity while retaining dominant features. The feature maps are flattened and passed to a fully connected dense layer with 128 neurons and a ReLU activation function, which processes the high-level features. To mitigate overfitting, we incorporated a dropout layer that randomly drops 50% of the connections during training. The final dense layer, with 10 neurons and a softmax activation function, outputs probabilities for each class. The model was compiled using the Adam optimizer and categorical cross-entropy loss function, with accuracy as the evaluation metric.

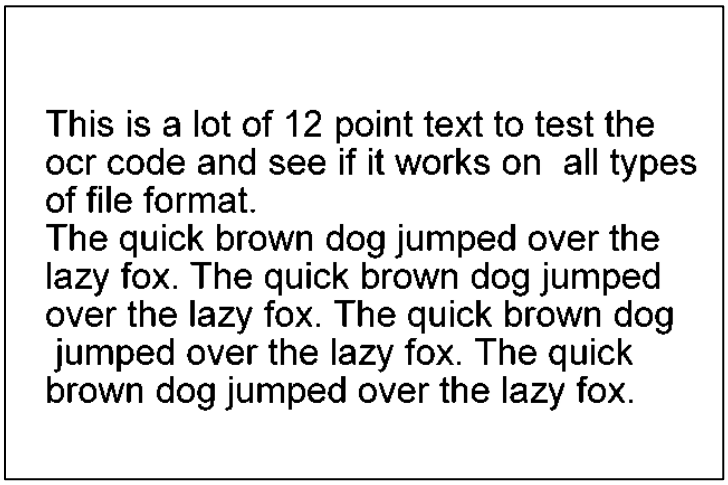


- **Training:** To enhance model generalization and reduce overfitting, we implemented data augmentation using ImageDataGenerator with transformations such as random rotations (up to 10 degrees) and horizontal/vertical shifts (up to 10% of the image dimensions). The augmented data was fed into the model during training in batches of 64 samples. The model was trained for 10 epochs, with validation data used to monitor performance after each epoch. Training progress was visualized through accuracy plots, showing both training and validation accuracy trends. This approach improved the model's ability to generalize by exposing it to a wider variety of data variations.



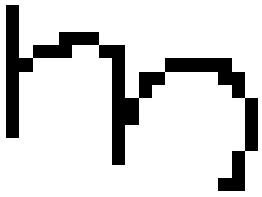
- **Challenges:**
  - Difficulty in recognizing sequences.

  - False classification of test image. We used this image to test model's tendency for OCR,



    But the model classified it as "8".

- While attempting single character recognition, on this image,



The model classified it as “0”.

- **Miss-steps:**
  - After reevaluating our choices, we recognized the issue with our initial training on only handwritten digits ranging from 0-9 as provided by MNIST
  - We tried synthetically generating sequences of characters using python’s pillow library, when we should’ve gone for EMNIST, or some other freely available dataset.

Despite these challenges, the model showed a 99% accuracy during training.

---

3. Classroom Discussion: Transition to Azure OCR API

Following class discussions, we were assigned the task to improve upon Micrsoft’s OCR model using generative AI. So, we decided to explore the Azure OCR API.

3.1. Integration and Evaluation

- **Integration:** Implemented the Azure OCR API to handle image-to-text conversions.
- **Evaluation:** Observed substantial improvements in:
  - Handling non-standard fonts and complex layouts.
  - Speed and reliability of text extraction.

3.2. Postprocessing

- We focused on post-processing outputs to fix minor inaccuracies.

- This resulted in using the GPT-2 model for improving OCR results.

---

4. Phase 2: GPT-2 for Generative Post-Processing

4.1. GPT-2

We used Hugging Face's implementation of GPT-2 to enhance the Azure OCR outputs by:

- Correcting grammar and punctuation errors.
- Generating contextual improvements for partially recognized text.

4.2. Workflow

- **Perform OCR with Azure Computer Vision**:
  - Load the image.
  - Send the image to Azure's OCR service to extract text data.
  - Wait for and retrieve the OCR results.
- **Extract Text from OCR Results**:
  - Parse the OCR results to extract readable text.
- **Enhance Text with a Hugging Face Model**:
  - Use a pre-trained Hugging Face model (GPT-2) to improve the text by filling in gaps or refining it.
- **Save Enhanced Text to a Color-Coded Word Document**:
  - Compare the original OCR text with the enhanced text.
  - Create a Word document where:
    - Matching words are highlighted in green.
    - Newly added words are highlighted in blue.
- **Test the Workflow**:
  - Provide an image (we used test_ocr.jpg) to process through the pipeline, generating a final color-coded Word document.

4.3. Results and Challenges

- **Improvements:**
  - Improved text readability and accuracy.
  - Fixed errors in fragmented sentences.
- **Challenges:**

- Elongating the text recognized by the OCR model,

OCR Result:	GPT-2 Result
Periodicals postage paid at Champlain and at additional	Periodicals Postage paid at Champlain and at additional cost to the United States. The following is a list of the

- **Miss-steps:**
  - We tried using the openai API to preprocess our text, but dropped the idea due to the complexity of openai's documentation and their migrations.
  - For OCR we tried using a pyautogui bot to perform OCR using Microsoft's Power Toys Text Extractor feature. This method quickly fell apart when collab wasn't able to establish a connection to the client machine for pyautogui.
  - We also tried T5 for text generation coupled with the pyautogui bot but since that didn't work, we left the idea entirely.
  - While uploading code to GitHub

---

5. Learning outcomes and Miss-steps

5.1. Learning outcomes

- Initially working on an OCR model, we identified the complexity of the task and garnered a new appreciation of the readily available OCR tools like Google Lens and Microsoft's Power Toys. We also developed an understanding for text classification models using CNNs and how its components work in a practical setup.
- After developing a pipeline integrating Azure OCR with GPT-2 text generation we were able to build a fundamental understanding on creating AI enhanced tools all the while polishing our python skills to utilize these tools.
- Most of all we were able to explore and gain deeper insight on how OCR and Text generation models work, and put to test the knowledge we gained on these models throughout the course.

5.2. Miss-steps

- The PyTesseract model's frequent substitution of numbers with special characters.
- Early GPT-2 outputs generating overly formal or irrelevant text.
- Instances where Azure OCR completely misinterpreted handwritten notes, leading to comical results.

6. Final Results and Comparison

6.1. Performance Metrics

Approach	Accuracy	Readability	Speed
PyTesseract	Not Implemented	Not Implemented	Not Implemented
Custom Model	99%	Low	Fast
Azure OCR	90%	Very High	Fast
GPT-2 Enhanced OCR	95%	Excellent	Medium

6.2. Key Takeaways

- GPT-2 enhanced OCR provided the best results but required substantial computational resources.
- Azure OCR API offered a balanced combination of performance and efficiency.
- Training a custom model remains valuable for specialized OCR tasks.

---

7. Conclusion

Our journey through OCR development highlighted the strengths and limitations of various approaches. Starting with PyTesseract as a baseline, we explored the potential of custom deep learning models, integrated the robust Azure OCR API, and leveraged GPT-2 to refine outputs. Each step contributed valuable insights, and our documented miss-steps underscore the iterative nature of innovation.

This project demonstrates the power of combining traditional and generative AI techniques to solve complex problems like OCR.