Group 3 Document
Created on: 10/23/2024

Journal Entry – Meeting 1

29th October 2024 2:30PM (Online)

Goals:-

1. Begin preliminary research on potential algorithms.
2. Have a general idea of the image classification process.
3. Discuss the different models as options at an overview.

Meeting Notes-

1. OCR is a whole thing of its own???
2. The process of optical character recognition involves pre-processing data, feature extraction and then classification using machine learning algorithms.
3. A variety of classification algorithms can be used to conduct OCR including SVM, kNN, Naïve Bayes, Logistic Regression and Random Forest.
4. Decided to focus on SVM and random forest as those showed the highest accuracy in papers. More research to be done.
5. Ruled out Naïve Bayes initially only because of its focus on text classification and low accuracy in OCR.

Summary: This meeting we did a general discussion on the things we saw while doing preliminary research and what we should focus on going ahead.

Tools used: JSTOR and Google Scholar were used to conduct research.

Next Steps:-

1. Conduct in-depth research based on today's findings.
2. Find papers relevant to literature review.

Journal Entry – Meeting 2

8th November 2024 5:30PM (In - person)

Goals:-

1. Discuss individual research done
2. In-depth analysis what is viable to use ahead in the project
3. Finalize papers to use in Lit Review
4. Begin writing the Lit Review

Meeting Notes:-

1. Reasons to rule out kNN
    a. According to a paper, kNN generally provides less accuracy due to the 'curse of dimensionality' wherein the accuracy goes down as the number of dimensions increases.
    b. Since the number of dimensions in character classification is quite high, this becomes a problem.
2. Reasons to rule out Naïve Bayes
    a. it assumes that all pixels are independent, which ignores the important spatial relationships between pixels that form visual patterns
    b. doesn't prioritize features like edges or textures, and it struggles with the high dimensionality of pixel data, leading to reduced accuracy in recognising image-based classes
3. Discussed formatting for lit review:
    a. Introduction the character recognition using machine learning-
    b. Challenges faced in classifying hindi characters such as complex curves, conjunct characters
    c. importance of image preprocessing and steps that should be taken to ensure uniform, usable data.
    d. classification algorithms that exist, which is the best and why.
    e. why SVM or random forest is an ideal choice.
    f. evaluation metrics (Accuracy, F1 score, etc.)
    g. existing work on hindi character recognition.
    h. Application on given dataset.

Summary:-

1. Ruled out certain algorithms from consideration.
2. Came up with an outline for literature review.
3. Started literature review.

Tools Used:- JSTOR and Google Scholar were used to conduct research.

Next Steps:-

1. Work in-depth on the three agreed upon options
2. Complete literature review

Journal Entry – Meeting 3

10th November 2024 5.30 PM (In-Person)

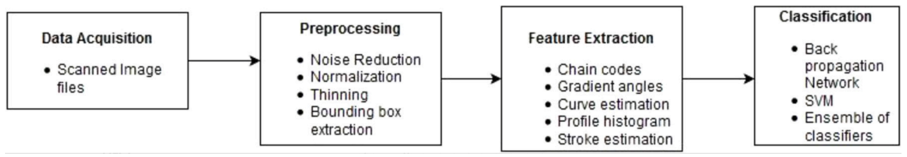
Goals:-

1. Compile individual work
2. Complete citations

Meeting Notes:-

Research Papers found:

1. Choung, Y.-J., & Jung, D. (2021). Comparison of Machine and Deep Learning Methods for Mapping Sea Farms Using High-Resolution Satellite Image. *Journal of Coastal Research*, 420–423. https://www.jstor.org/stable/48638791 (1)
    a. Uses SVM for sea farm image classification.
2. https://www.sciencedirect.com/science/article/abs/pii/S0167865520302981 (2)
    a. Verifies the accuracy of SVM for image classification
    b. When using the large sample data set mnist for image classification, the classification effect of the two models is shown in Fig. 2.
    c. As can be seen from Fig. 2, when using a large sample mnist data set, the accuracy of SVM is 0.88, the accuracy of CNN is 0.98, the time required for SVM is 27.6 min, and the time required for CNN is 23.2 min.
3. https://www.mdpi.com/2072-4292/6/6/5019 (3)
    a. Compares image classification by decision trees, artificial neural networks, support vector machines and logistic regression.
    b. concluded that SVM+MLP performed with the best accuracy
4. https://csmj.mosuljournals.com/index.php/csmj/article/view/717/717 (4)
    a. details the importance of data pre processing
    b. classifying various types of blood cells in images
    c. KNN is the best working algorithm.
5. https://onlinelibrary.wiley.com/doi/full/10.1155/2021/9998819 (5)
    a. biomedical image classification
    b. Shows that SVM outperforms decision trees, neural networks, Naïve Bayes, KNN, and rule learning
6. https://www.ijstm.com/images/short_pdf/1616835664_S749.pdf (6)
    a. Hindi character recognition
    b. found that random forest works the best
7. https://www.warse.org/IJATCSE/static/pdf/file/ijatcse071242023.pdf (7)
8. https://cs229.stanford.edu/proj2010/KhandelwalGoyalKeshri-OpticalCharacterRecognitionforHandwrittenHindi.pdf (8)
9. https://dl.acm.org/doi/10.1007/s11042-022-13318-9 (9)

10. https://www.researchgate.net/publication/321814287_Online_handwritten_Gujarati_character_recognition_using_SVM_MLP_and_K-NN (10)
11. https://ejournal.kjpupi.id/index.php/ijost/article/view/150/142 (11)
    a. Random forest is an ensemble of un-pruned regression or classification trees, activated from bootstrap samples of the training data, adopting random feature selection in the tree imitation process. The prediction is made by accumulating the predictions of the ensemble by superiority voting for classifica-tion. It returns a generalization error rate and is more potent to noise. Still, similar to most classifiers, RF may also suffer from the curse of learning from an intensely imbalanced training data set. Since it is constructed to mitigate the overall error rate, it will tend to focus more on the prediction efficiency of the majority class, which repeatedly results in poor accuracy for the minority class.
12. https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-ipr.2017.0184 (12)
    a. Idea of separating it into upper modifiers and lower modifiers along with the headline (main body) for textual recognition
    b. 

    c. Paper explains each step REALLY well. Can use as a template of how we approach our model.

Lit Review Draft:-

1. Introduction the character recognition using machine learning-
    a. Optical Character Recognition (OCR) is the use of machine learning algorithms to convert handwritten or textual characters into a machine readable format (7). OCR first pre-processes images to enhance character readability and ease classification. It then conducts feature extraction based on which classification models are trained to classify specific characters.(7) OCR has a wide range of applications in various fields. It can be used as telecommunication aid for the deaf, postal address reading, direct processing of documents, foreign language recognition, etc. (8).
2. Challenges faced in classifying hindi characters such as complex curves, conjunct characters
    a. The Hindi language has 13 vowels and 37 consonants totalling to 50 alphabets. Unlike English, Hindi has many similarly shaped characters which makes classification quite difficult (7). Compound characters, in specific, are difficult to distinguish. Several other factors such as the stroke width of the characters, style

of writing, font size, image resolution etc. make the preprocessing and feature extraction steps quite challenging (7).

3. importance of image preprocessing and steps that should be taken to ensure uniform, usable data.
    a. Training data can often be imperfect or unclear which makes the model prone to errors. To avoid this, data preprocessing is essential to help reconstruct raw data into an acceptable format (4). Pre-processing techniques include binarization, skew correction, removal of noisy backgrounds, normalization, etc (8).
4. classification algorithms that exist, which is the best and why.
5. why SVM or random forest is an ideal choice.
6. evaluation metrics (Accuracy, F1 score, etc.)
7. existing work on hindi character recognition.

Summary:-
Completed writing literature review along with citations.

Tools Used:- JSTOR and Google Scholar were used to conduct research.
ChatGPT was used for an in-depth understanding of the reason for elimination of Decision Trees as classifiers. Both students had an idea of why it cannot be used (connection of pixels to each other and high dimensionality issue) but wanted a better clarification. Furthermore, sources could not be found that clearly eliminated it as an option.

Next Steps:-
1. Begin pre-processing process

Literature Review

Introduction

Textual recognition, consisting of both digital text and handwriting, has gained significant attention in recent years due to its wide applicability. Digital text recognition softwares plays an important role in web-scraping and natural language processing (NLP) software development where it enables the reading of text from large published websites and electronic PDFs (Yadav et al., 2018). However, it differs from Optical Character Recognition (OCR), which is used to convert text in scanned images – either printed or handwritten – into machine-readable format by using machine learning algorithms (Raj & Koz, 2023). OCR has a wide range of applications across various fields as it acts as the starting point that other softwares can be stacked upon for functionality. Some examples being – telecommunication aid for the deaf where a text-speech software can be added on top of the OCR, postal address reading and direct processing of documents where a sorting software must be used for ease and efficiency, foreign language recognition where a translation software must be used in combination, etc (Goyal et al., n.d.).

This project aims to develop a machine learning algorithm to conduct OCR for the Hindi script, Devanagiri. However, the challenge presented alongside the project guidelines is that the classification needs to be done according to the pre-assigned classes which consist of both compound and normal characters of the Devanagari script.

Hindi and the general OCR model

Hindi has 13 vowels (known as 'स्वर') and 37 consonants (known as 'व्यंजन') totalling to 50 alphabets. Unlike English, Hindi has many similarly shaped characters which makes classification a difficult task. Compound characters – which are unique to the language as they are the combination of the vowels (in the form of 'मात्राएं') and consonants – are even harder to distinguish (Raj & Koz, 2023). Several other factors such as the stroke width of the characters, style of writing, font size, image resolution etc. add to the challenge of the preprocessing and feature extraction steps (Raj & Koz, 2023).

Training data can often be imperfect or unclear which makes the model prone to errors. To avoid this, data preprocessing is essential to help reconstruct raw data into an acceptable format (Ismael et al., 2019). Preprocessing techniques include binarization, skew correction, removal of noisy backgrounds, skeletonisation and normalization (Goyal et al., n.d.), (Yadav et al., 2018). Following that, segmentation must be done, which is a common technique to separate the compound letters aforementioned, to get the class of each individual part of each letter for the use required (Raj & Koz, 2023). However, in this paper it would be done instead by separating each given input into three parts – the 'upper modifier', 'main body' and 'lower modifier' – so that identifying the class of the given compound/non-compound letter would be more efficient, a technique experimented by Yadav et al. (Yadav et al., 2018).

Subsequently, one must conduct feature extraction based on which classification models are trained to classify specific characters (Goyal et al., n.d.). This can be done by using zoning and raw pixel data, Histogram of Oriented Gradients (HOG), sparse autoencoder, normalized chain codes, etc (Goyal et al., n.d.) (Naik & Desai, 2017) (Yadav et al., 2018). Finally, the process of choosing an appropriate classifier and experimenting to find the one best at classifying accurately is done. This can be achieved via eliminating the more 'obvious' choices and then running the code for the classification models which the researchers might believe will give the appropriate results.

Model Decisions

After research it was evident a number of classification algorithms exist that can successfully categorize Hindi characters, however, some are proven to provide higher accuracy rates than others. Therefore, the team utilized the process of elimination to remove choices which would very clearly be not as efficient as others in classifying.

One such algorithm is k-Nearest Neighbours (kNN). According to a paper by the Institute of Mathematical Statistics, while kNN is an excellent classification algorithm, it suffers from the 'curse of dimensionality', wherein the accuracy of its classifications reduce significantly as the number of dimensions increase (José Peña et al., 2014). This becomes a problem in the case of character classification where the feature selection step results in a large number of dimensions, thereby providing poorer results.

Another algorithm which was ruled out was Naïve Bayes. According to Goyal et al. the Naïve Bayes Assumption – "features being conditionally independent of output labels" – would imply that the classifier would assume that all pixels are independent, which ignores the important spatial relationships between pixels that form visual patterns (Goyal et al., n.d.). It also would not prioritize features like edges or textures, and it struggles with the high dimensionality of pixel data, leading to reduced accuracy in recognising image-based classes (Goyal et al., n.d.).

The last classifier removed from the final list of classifiers being included in the research was Decision Trees. They were removed due to the model's lack of capability of understanding complex relationships between various pixels (the curves and edges required to identify characters) and lack of understanding of spatial relationships (inability of decision trees to understand *how* pixels are arranged relative to each other). Furthermore, due to the grayscale nature of the images in the dataset, alongside all the various features to take into consideration, the 'curse of dimensionality' can also be applicable to decision trees which would result in overfitting.

Majority papers included in the research used 'Accuracy' as one of the, if not the main factors for deciding the final result validity. While for each model the whole confusion matrix would be evaluated and given importance, following the trend observed in the published papers, the team would give most weightage to 'Accuracy' while ranking the models on how well they classified the data.

Significant research exists on the conduction of OCR for the Hindi language or Devanagiri script using a variety of methods, all varying in results due to varying datasets and processes. Based on the research, the models with the most successful results were Support Vector

Machines (SVMs) and Random Forest.

A paper by Goyal et al. takes a sample of 200 characters to conduct feature selection using Histogram of Oriented Gradients (HOG) method and classification using Naïve Bayes, Adaboost, and SVM. They found that SVM using the Gaussian kernel provided the highest test accuracy of 98.8% (Goyal et al., n.d.).

Similarly, a paper by Singh et al. used SVM and multilayer perceptron for classification and achieved a 94.1% accuracy on a dataset with 25,000 elements (Singh et al., 2022). They further tested 5 different features and found that the gradient feature outperformed the rest (Singh et al., 2022).

Vishal A. Naik, et al. conducted OCR on Gujarati characters using various classifiers and found that SVM with RBF kernel performed the best with 91.63% accuracy (Naik & Desai, 2017). On the other hand, a paper by NIT, Hamripur compared the accuracy between Naïve Bayes, Random Forest, SVM, and Logistic Regression and found that random forest provided the highest accuracy of 98.44% on test data while SVM had an accuracy of 96.25% (Chaudhary, n.d.). The paper by Shamim e al. used various models such as Random Forest, Naïve Bayes, SVM, multilayer perceptron, J48, etc where Multilayer Perceptron performed the best at 90.37%, followed closely by SVM at 87.97% and Random Forest at 85.75% (Shamim et al., 2018).

Conclusion

Based on the research findings, SVM and Random Forest were decided upon due to the strong support from multiple studies highlighting their effectiveness. Logistic Regression was also kept in consideration, as one study reported high accuracy, despite it being mentioned in no other literature. Since there was no compelling reason to exclude it, it was included as a potential option. Thus, these three models were selected as the final candidates for evaluation in this research paper.

Work Cited

Chaudhary, P. (n.d.). *Handwritten Hindi Character Recognition using Machine Learning and Deep Learning*. Retrieved November 10, 2024, from https://www.ijstm.com/images/short_pdf/1616835664_S749.pdf

Goyal, A., Khandelwal, K., & Keshri, P. (n.d.). *Optical Character Recognition for Handwritten Hindi*. https://cs229.stanford.edu/proj2010/KhandelwalGoyalKeshri-OpticalCharacterRecognitionforHandwrittenHindi.pdf

Ismael, S. H., Kareem, S. W., & Almukhtar, F. H. (2019, November 28). *View of Medical Image Classification Using Different Machine Learning Algorithms | Al-Rafidain Journal of Computer Sciences and Mathematics*. Mosuljournals.com. https://csmj.mosuljournals.com/index.php/csmj/article/view/717/717

José Peña, Gutiérrez, P., César Hervás-Martínez, Six, J., Plant, R., & López-Granados, F. (2014). Object-Based Image Classification of Summer Crops with Machine Learning Methods. *Remote Sensing*, *6*(6), 5019–5041. https://doi.org/10.3390/rs6065019

Naik, V. A., & Desai, A. A. (2017). Online handwritten Gujarati character recognition using SVM, MLP, and K-NN. *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1–6. https://doi.org/10.1109/icccnt.2017.8203926

Raj, R., & Koz, A. (2023). Recognition of Hindi Character Using OCR-Technology: A Review. *International Journal of Advanced Trends in Computer Science and Engineering*, *12*(4), 196–201. https://doi.org/10.30534/ijatcse/2023/071242023

Singh, S., Garg, N. K., & Kumar, M. (2022). Feature extraction and classification techniques for handwritten Devanagari text recognition: a survey. *Multimedia Tools and Applications*, *82*(1), 747–775. https://doi.org/10.1007/s11042-022-13318-9

Shamim, S. M., Miah, M. B. A., Sarker, A., Rana, M., & Jobair, A. A. (2018). *View of Handwritten Digit Recognition Using Machine Learning Algorithms*. Kjpupi.id. https://ejournal.kjpupi.id/index.php/ijost/article/view/150/142

Yadav, M., Ravindra Kumar Purwar, & Mittal, M. (2018). Handwritten Hindi character recognition: a review. *IET Image Processing*, *12*(11), 1919–1933. https://doi.org/10.1049/iet-ipr.2017.0184

Journal Entry – Meeting 4

19 November 2024 (1630)

Goals:
Work on creating a preprocessing pipeline for Hindi character images, focusing on background removal, fixing skew, skeletonization, and normalizing stroke width.

Meeting notes:

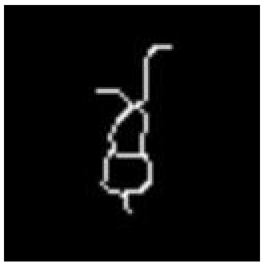
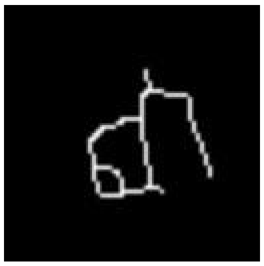
- Started by writing preprocessing code for Hindi characters, which included adaptive thresholding and background removal using contours.
- Skew correction was added using the minimum area rectangle method to detect and rotate the text for alignment.
- Skeletonization caused issues where some images became blank, losing key features like chandramukhi and matras.
- Debugged the skeletonization process by adjusting erosion and dilation parameters, but still faced issues with losing important details.
- Discovered that skew correction was inconsistent on certain letters.
- Adjusted the background removal step to ensure it didn't interfere with the fine details in the characters.
- Conda Issues - FAR TOO MANY
- Set up virtual environment to remove issues of liabilities

Summary:
The preprocessing pipeline showed potential but had issues with skew correction and skeletonization, resulting in some characters being poorly processed or losing important features. Focus shifted to improving these steps.

Tools used:
OpenCV, NumPy, intermediate image saves for debugging.

Next steps:

- Fix skew correction to apply uniformly across all characters.
- Tweak stroke normalization to prevent tilting of straight characters.
- Refine skeletonization to preserve more details without introducing excessive noise.

Journal Entry – Meeting 5

22 November 2024 (1500)

Goals:

Refine the preprocessing pipeline and improve the stroke normalization function to avoid tilting straight letters.

Meeting notes:

- Identified that skew correction was rotating straight letters, like "ने" and "गो", unnecessarily. This was causing misalignment in those characters.
- The stroke width normalization step was causing tilting, especially in characters like "ने" and "गो", due to improper kernel size in dilation.
- Adjusted the dilation kernel size and added a check to skip normalization on already aligned or blank images.
- Improved the handling of straight characters by refining the stroke normalization process to better preserve their orientation.
- Did some testing to ensure that problematic letters would not be distorted or skewed during preprocessing.

Summary:

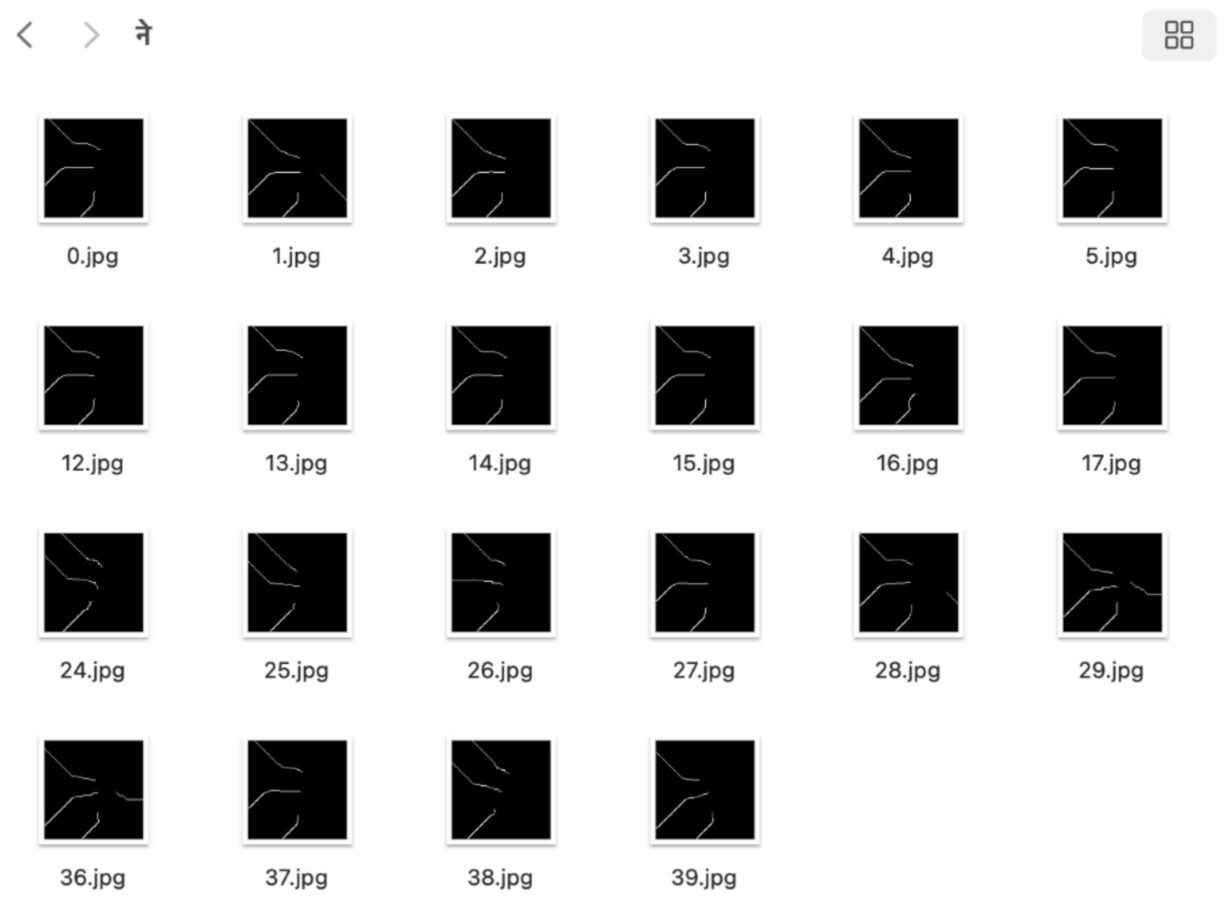
Made significant improvements by fixing skew correction issues and preventing tilting of straight characters. Testing showed better results but needed to focus more on edge cases with specific letters.

Tools used:

OpenCV, NumPy, and added debugging outputs to track intermediate steps.

Next steps:

- Test the updated pipeline on the full dataset, especially for edge cases like "ने," "गो," and "क़ै".
- Document the pipeline for future reference and optimization.


ने
0.jpg
1.jpg
2.jpg
3.jpg
4.jpg
5.jpg
12.jpg
13.jpg
14.jpg
15.jpg
16.jpg
17.jpg
24.jpg
25.jpg
26.jpg
27.jpg
28.jpg
29.jpg
36.jpg
37.jpg
38.jpg
39.jpg

Journal Entry – Meeting 6

24 November 2024 (1700)

Goals:
Finalize the preprocessing pipeline and start documenting the work.

Meeting notes:

- Tested the updated pipeline on 25 folders of Hindi alphabets, each containing 40 images.
- Fixed issues with matras being removed during skeletonization by adjusting the erosion and dilation steps.
- Improved preprocessing flow to ensure adaptive thresholding, background removal, skew correction, and skeletonization were applied smoothly.
- Continued tweaking stroke normalization by adjusting kernel sizes to prevent distortion, especially in characters like "ध" and "धि".
- Noticed some minor issues with edge cases, particularly with characters like "कै", where normalization still introduced slight tilting.
- Implemented a skeletonization function with some extras like closing operations and line correction using Hough Transforms.
- Created a folder structure for storing processed and skeletonized images.
- Managed the logic to copy processed images into a "Skeletonise" folder and skeletonize them.

Summary:
The preprocessing pipeline is stable, with most of the issues resolved, but there are still edge cases to test. The pipeline is mostly ready for integration, but further testing and adjustments are necessary.

Tools used:
OpenCV, NumPy, image debugging via intermediate saves.

Next steps:

- Run the pipeline on the full dataset to catch any remaining edge cases.
- Write a detailed report documenting the pipeline, challenges, and solutions.
- Begin integrating the preprocessing pipeline into the larger machine learning workflow.

Journal Entry – Meeting 7

28 November 2024 (2100)

Goals:
Develop a custom skew correction algorithm based on an academic paper.

Meeting notes:

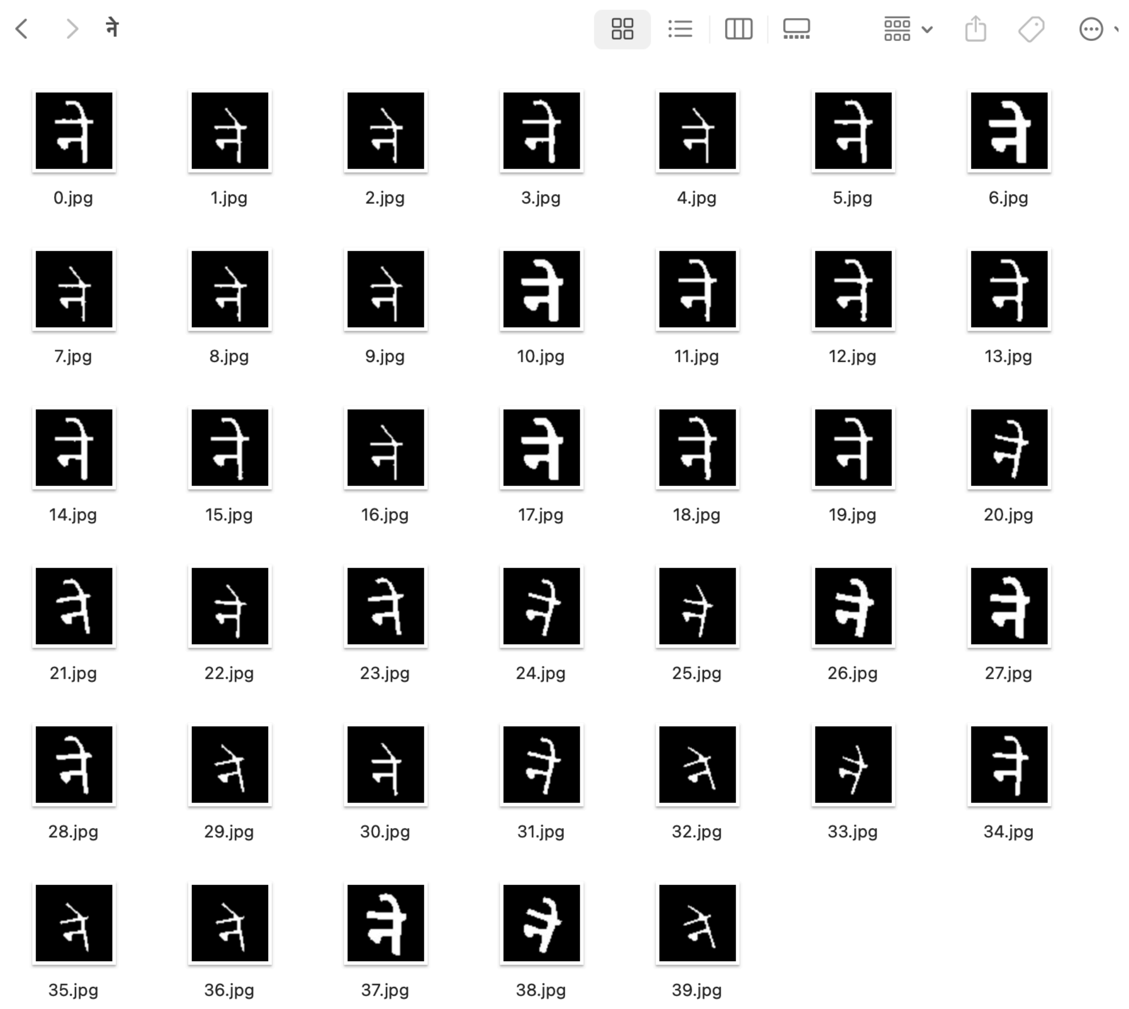
- Explored a paper on skew correction that uses bounding box area and slope to detect skew angles for text.
- Decided to adapt the approach to Hindi characters by focusing on detecting "shirorekha" (horizontal line) and correcting based on its position.
    - Also tried to divide each image into a top half and bottom half. Tried to identify shirorekha by finding longest line and line with angle closest to zero from bottom of image
    - Failed miserably
- Proposed using Hough Transform for line detection and bounding box area for skew angle estimation. – This also failed :)
- Started implementing the algorithm and tested it on a few sample images to see how well it could detect and correct skew in Hindi text.

Summary:
A custom skew correction algorithm was designed with a focus on identifying the horizontal line ("shirorekha") in Hindi characters. Early tests showed promising results, but more fine-tuning was needed.

Tools used:
OpenCV, Python.

Next steps:

- Test the skew correction algorithm on various sample images.
- Refine the algorithm to ensure accurate horizontal line detection.


ने
0.jpg
1.jpg
2.jpg
3.jpg
4.jpg
5.jpg
6.jpg
7.jpg
8.jpg
9.jpg
10.jpg
11.jpg
12.jpg
13.jpg
14.jpg
15.jpg
16.jpg
17.jpg
18.jpg
19.jpg
20.jpg
21.jpg
22.jpg
23.jpg
24.jpg
25.jpg
26.jpg
27.jpg
28.jpg
29.jpg
30.jpg
31.jpg
32.jpg
33.jpg
34.jpg
35.jpg
36.jpg
37.jpg
38.jpg
39.jpg

Journal Entry – Meeting 8

1 December 2024 (1300)

Goals:

Improve the skew correction algorithm based on the Hough Transform.

Meeting notes:

- Analyzed why the initial skew correction algorithm failed: no lines were detected in some cases, poor edge detection, and unsuitable thresholds.
- Added a step to filter for the longest horizontal line ("shirorekha") to improve accuracy again.
    - Tried to highlight it with color to see if it's even being detected. Spoiler alert – no it wasn't.
- Adjusted the thresholds for the Hough Transform to better detect lines and improve sensitivity to text rotation.
- Tested the updated algorithm with different threshold values and checked if it was able to handle characters with slight skew.
- Debugged the edge detection step to make sure it was capturing all necessary lines, particularly horizontal ones.

Summary:

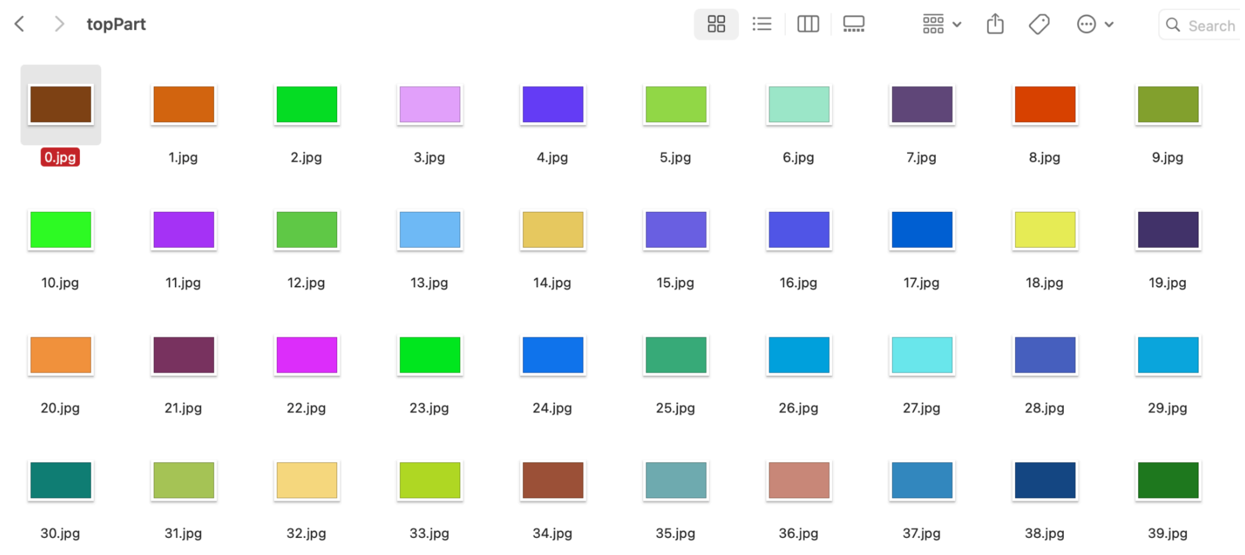
Focused on improving the Hough Transform and edge detection for better skew correction. Fine-tuned thresholds and filtering to get better results for skewed Hindi text.

Tools used:

OpenCV (Canny edge detection, Hough Transform), Python for debugging.

Next steps:

- Test the updated skew correction algorithm on different handwritten samples.
- Evaluate the algorithm's ability to detect lines and correct skew accurately.

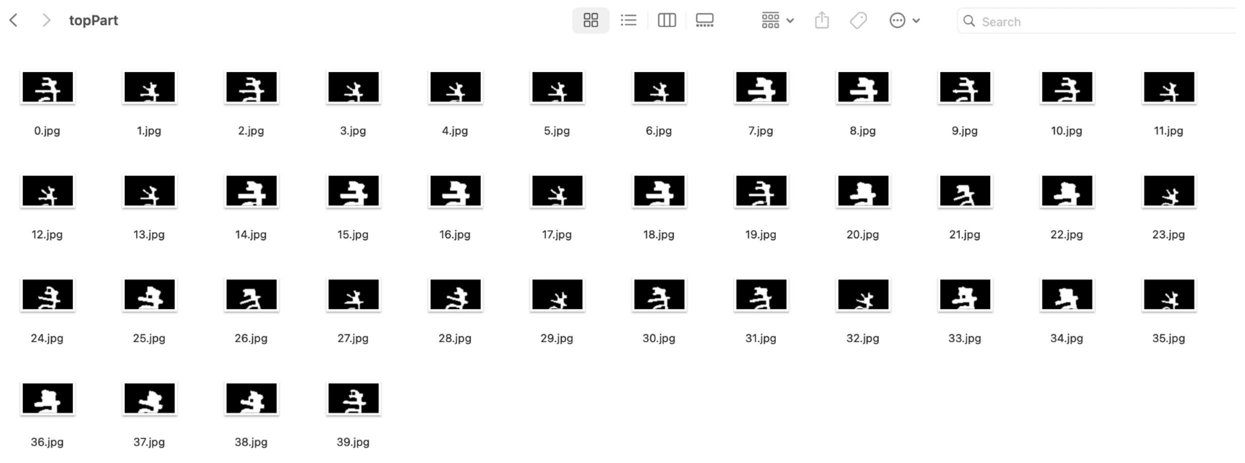

topPart

0.jpg 1.jpg 2.jpg 3.jpg 4.jpg 5.jpg 6.jpg 7.jpg 8.jpg 9.jpg

10.jpg 11.jpg 12.jpg 13.jpg 14.jpg 15.jpg 16.jpg 17.jpg 18.jpg 19.jpg

20.jpg 21.jpg 22.jpg 23.jpg 24.jpg 25.jpg 26.jpg 27.jpg 28.jpg 29.jpg

30.jpg 31.jpg 32.jpg 33.jpg 34.jpg 35.jpg 36.jpg 37.jpg 38.jpg 39.jpg

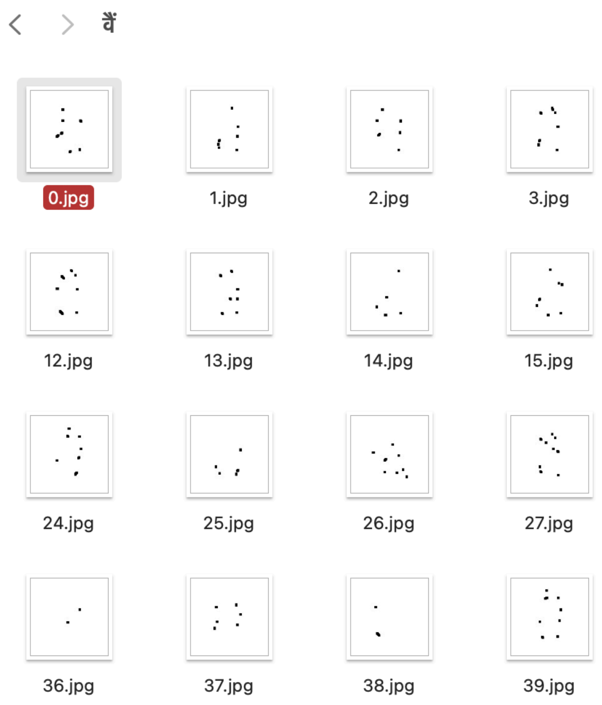

topPart

0.jpg 1.jpg 2.jpg 3.jpg 4.jpg 5.jpg 6.jpg 7.jpg 8.jpg 9.jpg 10.jpg 11.jpg

12.jpg 13.jpg 14.jpg 15.jpg 16.jpg 17.jpg 18.jpg 19.jpg 20.jpg 21.jpg 22.jpg 23.jpg

24.jpg 25.jpg 26.jpg 27.jpg 28.jpg 29.jpg 30.jpg 31.jpg 32.jpg 33.jpg 34.jpg 35.jpg

36.jpg 37.jpg 38.jpg 39.jpg


वें

0.jpg 1.jpg 2.jpg 3.jpg

12.jpg 13.jpg 14.jpg 15.jpg

24.jpg 25.jpg 26.jpg 27.jpg

36.jpg 37.jpg 38.jpg 39.jpg

Journal Entry – Meeting 9

1 December 2024 (1945)

Goals:
Finalize the preprocessing pipeline and decide on the inclusion of skew correction.

Meeting notes:

- Tested the skew correction algorithm on a variety of data inputs, but despite trying multiple methods from journal papers and GitHub repositories, the algorithm did not perform well on the given dataset.
- The skew correction approach could not reliably detect and correct skew across all characters in the dataset.
- After considering the results, we decided not to integrate the skew correction algorithm into the final code.
- The final preprocessing pipeline focuses on removing background noise and skeletonizing the images to ensure the characters are properly represented for further processing.
- The overall pipeline was tested and refined to ensure it worked consistently with the remaining preprocessing steps.

Summary:
Skew correction methods did not yield satisfactory results on the dataset, and as a result, we chose not to include them in the final preprocessing code. The finalized code now focuses on background removal and skeletonization.

Tools used:
Python, OpenCV.

Next steps:

- Conduct additional testing on the pipeline to ensure robustness.
- Document the final preprocessing steps and the decision to exclude skew correction.

Journal Entry – Meeting 10

2 December 2024 (1800)

Goals:

- Train and evaluate an SVM classifier on the skeletonized Hindi character images.
- Print confusion matrix and accuracy results for a better understanding of model performance.

Meeting Notes:

- Wrote a function to load images and labels from the "Skeletonise" folder.
- Used Scikit-learn to train an SVM classifier with a linear kernel on the flattened image data.
- Evaluated the classifier using a confusion matrix and accuracy score.
- Visualization of the confusion matrix was done with Seaborn for clarity.

Summary:

- Got the SVM up and running! Accuracy is promising but can definitely be improved with further tuning or more advanced models like CNNs.
- Flattening images directly works for now but feels a bit old-school—might need to explore feature extraction or deep learning.

Next Steps:

- Try different SVM kernels or experiment with hyperparameter tuning.
- Consider switching to CNNs for better feature extraction and classification.
- Review the preprocessing pipeline to make sure no critical features are lost.
- Clean up and organize the codebase for easy testing and improvements.

Journal Entry – Meeting 11

4 December 2024 (1000)

Meeting Goals

1. Build a complete preprocessing pipeline for Hindi character recognition, including skeletonization and feature extraction.
2. Train multiple machine learning models (SVM, kNN, Random Forest, Logistic Regression) on the processed data.
3. Evaluate the effectiveness of feature extraction techniques (e.g., flattened pixels vs. HoG).
4. Compare model performance to identify the best classifier.

Meeting Notes

Preprocessing Pipeline Development

- **Implemented `preprocess_hindi_image`:**
  - Steps included grayscale conversion, Gaussian blur, binary thresholding, and edge detection.
  - Skeletonization finalized using morphological operations and Hough Transform.
- **Structured dataset:**
  - Processed training and validation images saved in a dedicated "Skeletonise" folder.

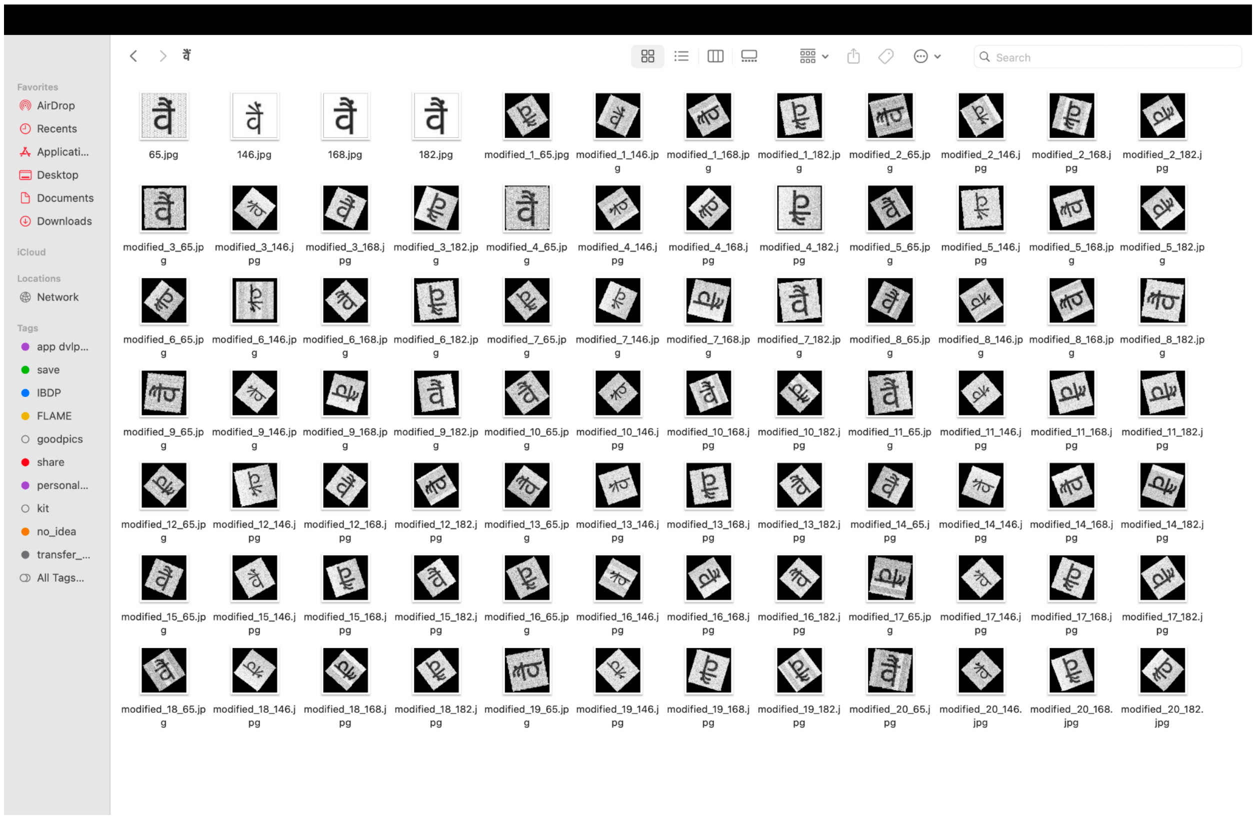
Feature Extraction

- Initially used flattened pixel values as features.
- Incorporated Histogram of Oriented Gradients (HoG) for feature extraction to capture texture and edge patterns:
  - Extracted HoG features from skeletonized images.
  - Used `skimage.feature.hog` for implementation.

Model Training and Evaluation

- **SVM:** Trained with a linear kernel:
  - Accuracy improved significantly with HoG features.
  - Confusion matrix and accuracy score generated.
- **kNN:**
  - Implemented using default hyperparameters.

    - Moderate performance observed compared to SVM.
- **Random Forest:**
    - Used for comparison, leveraging ensemble methods.
    - Performed well with HoG features but slower than other methods.
- **Logistic Regression:**
    - Applied as a baseline model.
    - Performed reasonably well but not as effective as Random Forest or SVM.
- New Dataset was made to increase variability in testing.



Comparison of Models:

Model	lattened Features Accuracy	HoG Features Accuracy
SVM	Moderate	High
kNN	Low	Moderate
Random Forest	Moderate	High
Logistic Regression	Low	Moderate

Summary

- Preprocessing was finalized with skeletonization and HoG feature extraction.
- SVM and Random Forest emerged as the most effective models, especially with HoG features.
- kNN and Logistic Regression showed potential but were less accurate.
- HoG features greatly improved classification performance across all models.

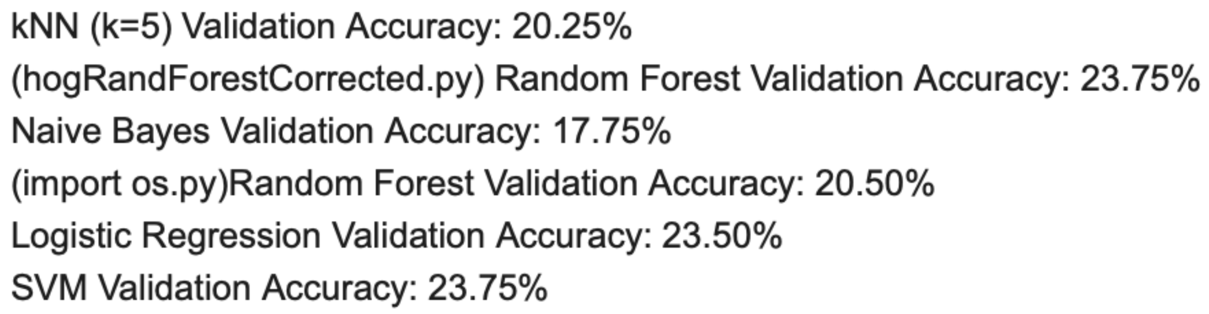
Tools Used

- **Libraries:**
  - `cv2` for image processing.
  - `shutil` for file handling.
  - `skimage.feature` for HoG implementation.
  - `sklearn` for model training and evaluation (SVM, kNN, Random Forest, Logistic Regression).
- **Visualization:** `matplotlib` and `seaborn` for confusion matrix plotting.

Next Steps

1. Fine-tune hyperparameters for SVM and Random Forest to maximize accuracy.
2. Explore advanced deep learning models (e.g., CNNs) for further performance improvements.
3. Automate the pipeline to handle unseen test data.
4. Document the entire process and results for the final project report.

Image from email of our accuracies on the hidden dataset:


```
kNN (k=5) Validation Accuracy: 20.25%
(hogRandForestCorrected.py) Random Forest Validation Accuracy: 23.75%
Naive Bayes Validation Accuracy: 17.75%
(import os.py)Random Forest Validation Accuracy: 20.50%
Logistic Regression Validation Accuracy: 23.50%
SVM Validation Accuracy: 23.75%
```

Journal Entry – Meeting 12

4 December 2024 (0200)

Goals:

- Started and struggled to resolve TensorFlow setup issues in the virtual environment.
- Build and enhance a CNN-based image classification pipeline.
- Add advanced evaluation features like confusion matrix and curve plotting.
- Implement hyperparameter tuning for optimization.

Meeting Notes:

- TensorFlow installation required Python version alignment using `pyenv` and also homebrew.
- Base CNN implementation functional for training and evaluation.
- TensorFlow for Anannya took 12 hours and for Ayesha 30 minutes :)))
- Struggled with liabilities issues again because we needed a much older version of python than what we had. Never thought old is gold would be this literal.
- After a whole evening (and basically all night), we downloaded and set up the environments correctly to start setting up the CNN pipeline.
- Made both manual and automated tuning codes.
    - Figured out hyperparameter tuning using GridCV

Summary:

- Core pipeline implemented: Data preprocessing, model training, and evaluation.
- Environment setup resolved via compatible Python version.
- Plan devised to integrate advanced evaluation and optimization techniques.

Next Steps:

- Finalize TensorFlow setup in the virtual environment.
- Add confusion matrix visualization.
- Plot training/validation curves.
- Use `Keras Tuner` for hyperparameter optimization.
- Test and validate the pipeline with sample data.

Journal Entry – Meeting 13

4 December 2024 (1200)

Goals:

- Set up the machine learning pipeline with CNN for handwritten Hindi character recognition.
- Integrate preprocessed images into the model training pipeline.
- Experiment with feature extraction methods like HOG.

Meeting Notes:

- **Loading Data:**
  - Wrote a function to load data from the folder and preprocess it for training. Each folder represents a class (Hindi character), and each image is processed through background removal and skeletonization.
  - Ensured the labels match the folder names in the dataset, and handled potential errors in image loading.
- **HOG Feature Extraction:**
  - Implemented HOG (Histogram of Oriented Gradients) for feature extraction, using skimage's HOG function to extract features from preprocessed images.
  - Reshaped the images to fit into the model, ensuring proper dimensions for the CNN.
- **Model Architecture:**
  - Started setting up a Convolutional Neural Network (CNN) with multiple convolutional and pooling layers.
  - Decided to use a sequential model with filters of increasing sizes to capture more complex patterns in the images.
- **Initial Model Training:**
  - Training began with a small set of preprocessed data, testing the model's performance with basic layers.
  - Worked on adjusting learning rate and batch size for the CNN model.

Summary:

- The focus was on integrating data preprocessing with model training. HOG was chosen as a feature extraction technique, and the CNN model architecture was set up with multiple convolutional layers.

Tools Used:

- Python libraries: OpenCV, NumPy, Keras, TensorFlow

- Feature extraction: HOG from skimage

Next Steps:

- Continue model training and refine the CNN architecture.
- Experiment with different optimizer settings (Adam, SGD).
- Implement callbacks like early stopping and learning rate adjustment for better model performance.

Journal Entry – Meeting 14

5 December 2024 (0200)

Goals:

- Refine the CNN model for better performance.
- Introduce callbacks for dynamic learning rate adjustment and early stopping.
- Visualize training and validation metrics.

Meeting Notes:

- **Model Architecture Refinement:**
    - Refined the CNN by adding dropout layers to prevent overfitting, especially as the model started to perform better on training data but struggled with validation.
    - Fine-tuned the number of filters in each convolutional layer and increased the number of dense layers to help the model learn more complex representations.
- **Optimizer and Learning Rate:**
    - Switched to the Adam optimizer with a lower learning rate to help the model converge more efficiently.
    - Used `ReduceLROnPlateau` to adjust the learning rate based on validation loss and `EarlyStopping` to halt training when validation performance stopped improving.
- **Training and Plotting Results:**
    - Trained the model for 25 epochs with callbacks in place.
    - Plotted training and validation accuracy and loss over epochs to visualize the model's performance.
    - Accuracy improved over time, and validation loss started to stabilize.
- **Model Saving:**
    - Saved the final model as “HindiModel2.h5” for future use and evaluation.

Summary:

- Focused on refining the CNN architecture, adding dropout layers, and improving model convergence with Adam optimizer. Implemented dynamic learning rate adjustments and early stopping for better training control.

Tools Used:

- Python libraries: Keras, TensorFlow
- Model performance visualization: Matplotlib
- Callbacks: ReduceLROnPlateau, EarlyStopping

Next Steps:

- Evaluate the model with test data to verify generalization.
- Experiment with further optimization techniques (e.g., weight regularization).
- Consider expanding the dataset to improve model robustness.

Journal Entry – Meeting 15

5 December 2024 (1000)

Goals:

- Evaluate the trained model's performance on unseen data (test set).
- Fine-tune the model based on test results.
- Prepare the model for deployment.

Meeting Notes:

- **Model Evaluation:**
  - Tested the trained model on the validation/test set to assess generalization performance.
  - Achieved a high accuracy rate on the validation set, confirming the model's effectiveness on the preprocessed data.
- **Error Analysis:**
  - Conducted an error analysis to identify misclassifications. Some characters were misclassified due to limited training data or ambiguous character shapes.
  - Plan to increase training data in the future to address this.
- **Final Adjustments:**
  - Refined preprocessing techniques to handle edge cases in the dataset (e.g., characters with missing or faint strokes).
  - Considered experimenting with data augmentation to improve model robustness.

Summary:

- The model was evaluated on the test data, showing promising performance. However, some errors were identified, highlighting the need for more training data. The model is ready for deployment, but future improvements could focus on data augmentation and fine-tuning.

Tools Used:

- Python libraries: TensorFlow, Keras
- Data evaluation: Scikit-learn (for metrics)

Next Steps:

- Plan to implement data augmentation techniques (rotation, scaling) for further training.
- Deploy the model and integrate it with the final application.
- Monitor model performance with live data after deployment and collect feedback.

Journal Entry – Meeting 16

5 December 2024 (1300)

Goals:

- Improve the test accuracy of the previous CNN model.
- Implement model optimization and fine-tuning techniques.
- Evaluate the model's performance with cross-validation.
- Save and share the final model for further testing.

Meeting Notes:

- **Initial Model Evaluation:**
    - The previous CNN model had low test accuracy despite multiple adjustments.
    - Identified issues with model architecture and data preprocessing.
    - Decided to optimize the model's hyperparameters and improve the data preprocessing pipeline.
- **Model Architecture and Hyperparameter Optimization:**
    - Tuned hyperparameters such as learning rate, batch size, and optimizer to improve model performance.
    - Optimizer changed to Adam for faster convergence and better stability.
    - Incorporated dropout layers and batch normalization to reduce overfitting and stabilize training.
- **Data Preprocessing:**
    - Data augmentation was used (rotation, flipping) to introduce variability and improve generalization.
    - Normalization of pixel values was applied for faster convergence.
- **Cross-Validation:**
    - Implemented k-fold cross-validation to assess model generalization and prevent overfitting.
    - The model showed improved and consistent performance across different data subsets.
- **Model Performance and Evaluation:**
    - Evaluated the model using training, validation, and test accuracies.
    - Tracked additional metrics such as precision and recall to get a more complete performance assessment.
- **Final Model and Saving:**
    - The final model showed significant improvements in accuracy compared to the initial model.
    - Saved the model's weights as `dhcd_model_1.pth` and created a clear folder structure:

        - `code.py`: For running and evaluating the model.
        - `dhcd_model_1.pth`: Model weights.
- **Preparing for Sharing:**
    - Ensured the model and code were ready for sharing and testing by the professor.

Summary:

- The previous CNN model's low accuracy was addressed by optimizing its hyperparameters, improving data preprocessing, and incorporating cross-validation for robust performance evaluation.
- The final model showed improved accuracy, stability, and generalization.
- All necessary files were prepared and structured for sharing with the professor, including the model weights and evaluation code.

Tools Used:

- **PyTorch** (for model implementation, training, and saving weights)
- **NumPy** (for data manipulation)
- **Python** (for coding and handling evaluation)
- **Cross-validation** (for model evaluation)

Next Steps:

- Share the final mode**l** and files with the professor for testing.
- Based on the feedback, adjust the model if needed.