A Student Tailored Handwriting Recognition and Summarization Model

Brydon Herauf, Nathan Cameron, Meklit Alemu

April 7, 2024

Table of Contents

1. Project Summary

This project intends to deliver a handwriting recognition tool that is specifically designed for students who take notes during class. It will take an image of the handwritten notes as input, and run it through an optical character recognition (OCR) model to convert them to a digital form. Additionally, it will summarize the notes using natural language processing (NLP), with emphasis on highlighted segments. Ultimately, the goal of this project is to increase the usefulness of OCR technology for students, and allow those who still prefer to take notes on paper to succeed in an increasingly digital time.

2. Problems This Project Intends to Solve

In today's world, there is a massive benefit to storing information online. However, there are still many people, including students, who find it more enjoyable and less distracting to write out notes on paper. There are current state-of-the-art OCR models that allow students to convert their notes into a digital form. However, the accuracy of these models is not perfect, and they are not tailored towards students. Our project aims to increase the quality of output by using NLP on the converted notes, which should help correct OCR errors, and allow for summarization. Many students also like to highlight segments of their notes which are deemed important by their professor or themselves. Our model will include highlighting recognition, and place emphasis on these segments during the summarization step. Overall, this project will help solve the problem of students who prefer to take handwritten notes being at a disadvantage, by not having access to electronic tools. It will accomplish this by directly providing a digital summarization of notes, with emphasis on important sections.

3. Project Objectives

The primary objective of this project is to deliver a solution that takes a scanned page of handwritten notes, identifies highlighted segments, and produces a summary that accurately describes the content from them. We hope to accomplish this by experimenting with both state-of-the-art and custom-trained OCR models. The accuracy does not have to be perfect, we just require the results to be clear enough for an advanced natural language processing model to understand them. In addition to achieving this level of accuracy on the OCR front, we hope to successfully identify any part of the text that is highlighted, and use that to provide a more accurate summary. We hope to see this process in its full form on a page of notes written by one of us.

4. Approaches and Implementation Details

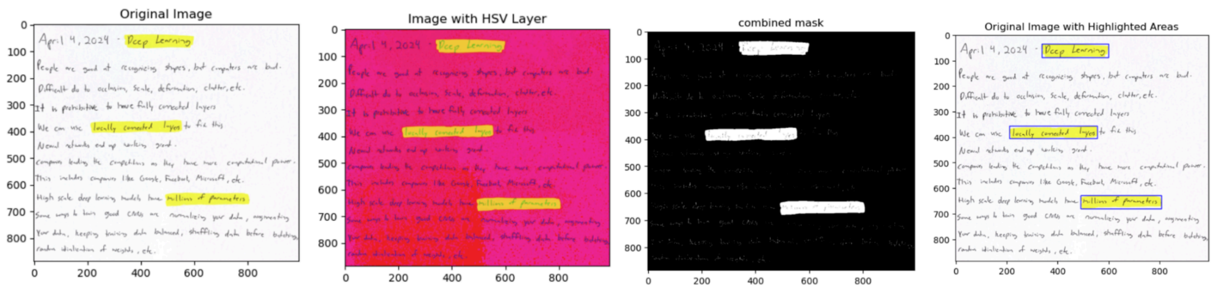
There are four parts of this project that are required to achieve our goals. The first is highlighted segmentation, where we separate sections of the notes that are highlighted, so we can emphasize them in the summarization step. Second is image preprocessing, which involves filtering and chopping up the original handwritten notes so that it works well with our OCR model. Next, we need an OCR model that can translate handwritten notes to a digital form with moderately high accuracy. Finally, we need access to a state-of-the-art NLP model, which will interpret the error-ridden digital notes, and summarize them accordingly.

4.1 Highlighted Segmentation - Worked on by Meklit

The first part of this process was separating highlighted segments from non-highlighted segments. Given an input of handwritten text with highlighted areas, the goal was to develop an algorithm to detect the highlighted areas with high accuracy. To accomplish this, a few different approaches were used.

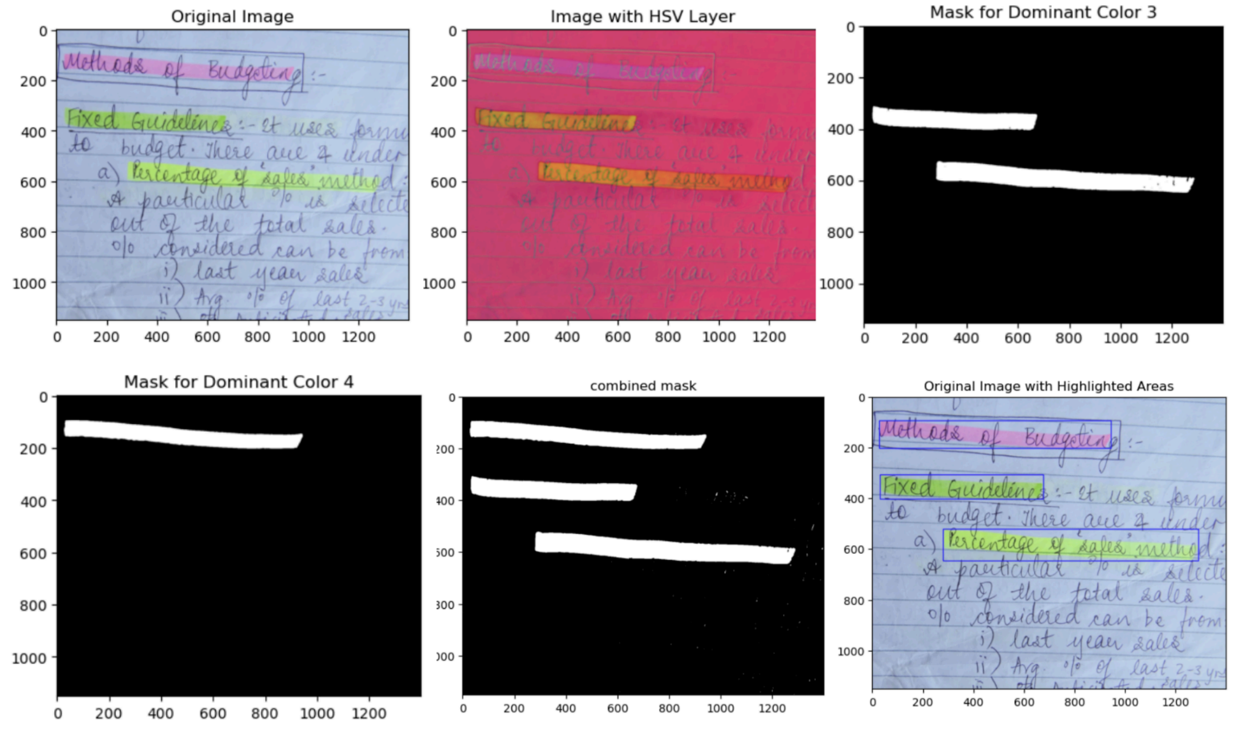
4.1.1 Hardcoded HSV Color Boundaries and OpenCV

Our initial attempt used binarization of the handwritten text image and hardcoded HSV color boundaries to identify highlighted areas. Bounding boxes were drawn around these areas using OpenCV and displayed for visualization. The image processing steps included grayscale conversion, Otsu's thresholding for noise reduction and contrast enhancement, color segmentation through conversion from RGB to HSV and creation of masks, and morphological transformations for cleaner boundary delineation. The resulting output featured bounding contour boxes around highlighted regions. The main drawback was the requirement and complexity of hardcoding lower and upper boundaries for desired HSV ranges.



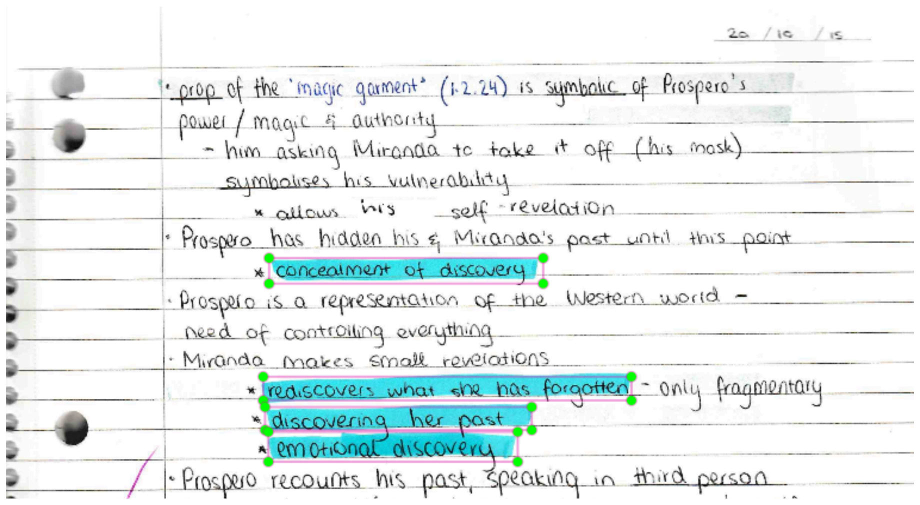
4.1.2 K-Means Clustering for Color Identification

To avoid the requirement of hardcoding HSV ranges, we leveraged K-means clustering to pinpoint dominant colors within an image. Image pixels are clustered based on color similarity, which allows us to identify dominant colors in the image. Through a defined set of criteria, including brightness and darkness thresholds, the algorithm filters out irrelevant colors such as black or white, focusing solely on colors that may potentially represent highlighted text. For each relevant color identified, the algorithm generates a mask based on its HSV (Hue, Saturation, Value) values. This isolates regions of the image that match the criteria for highlighted text. If multiple relevant colors are identified, their respective masks are combined to create a unified mask that encompasses all highlighted areas. The remaining drawback is that text written in the same color as the highlighter will be identified as a highlighted area.


Original Image
Image with HSV Layer
Mask for Dominant Color 3
Mask for Dominant Color 4
combined mask
Original Image with Highlighted Areas

4.1.3 Using CNNs to Differentiate Between Text and Regions

Our third and final method attempted to use an R-CNN Segment Detection Algorithm to solve the issue of highlighter colored text being identified as a highlighted region. It works by proposing potential regions of interest within an image, which are then subjected to a convolutional neural network (CNN) analysis to determine their relevance. This approach enables precise localization and extraction of highlighted regions, contributing to enhanced accuracy in identifying relevant information.



20 / 10 / 15

• prop of the 'magic garment' (1.2.24) is symbolic of Prospero's
power / magic & authority
- him asking Miranda to take it off (his mask)
symbolises his vulnerability
* allows his self-revelation
• Prospero has hidden his & Miranda's past until this point
* concealment of discovery
• Prospero is a representation of the Western world -
need of controlling everything
• Miranda makes small revelations
* rediscovers what she has forgotten - only fragmentary
* discovering her past
* emotional discovery
• Prospero recounts his past, speaking in third person

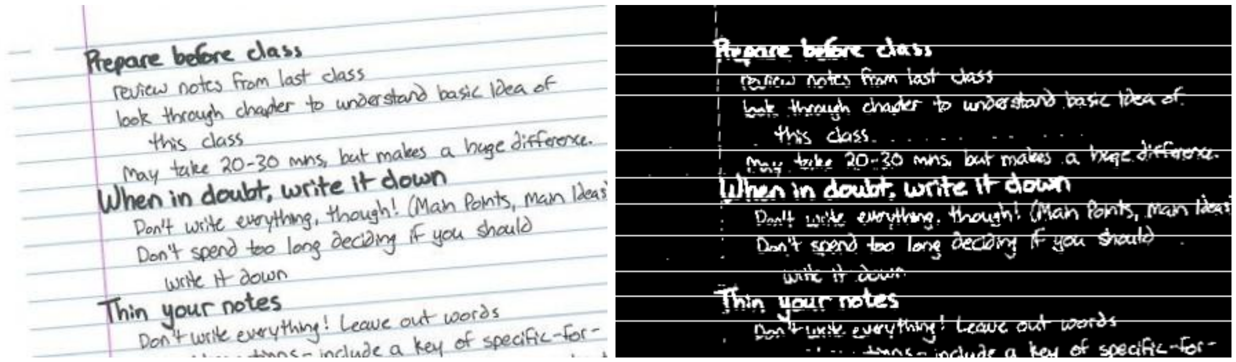
The following table shows the accuracy of each approach, measured against a test set containing ten samples.

Hardcoded HSV Ranges	0.93
K-Means Clustering	0.75
R-CNN Segmentation	0.62

If we compare the different Highlighted Segment Detection methods, Hardcoded HSV Ranges was the most accurate since the algorithm was given the exact HSV for each input image. The only inaccuracy in the K-Means Clustering method was identification of line borders or markings made using the dominant/highlighter colors. The R-CNN Segmentation used an underfit model, so accuracy was suboptimal. Our machine learning approaches were for experimentation and hoped to improve the versatility of the system. However, due to the lower accuracy, we will move forward with the hardcoded approach.

4.2 Image Preprocessing - Worked on by Nathan

An important part of machine learning is manipulating data so that new samples are in the same format that the model is trained on. To conform with our training data, a full page of handwritten notes had to be split into single lines, and converted to black letters on a white background. To do this we used OpenCV. We found where each line of text was on the page by looking for white space in between lines and using this as the upper and lower bounds. This worked really well for typed text, but for handwritten text characters such as ‘g’, ‘y’ and ‘p’ extend below, and can interfere with white space detection. Our implementation of this preprocessing is somewhat fragile. It suffers with crooked, close together, and lighter text. This is one section of the project we would put extra work into if we had more time, but as long as the notes are appropriately spaced, straight, and dark, the algorithm works fine. After the separate lines are identified, they can be saved as images, and used by the OCR model.



Prepare before class
review notes from last class
look through chapter to understand basic idea of
this class
May take 20-30 mins, but makes a huge difference.
When in doubt, write it down
Don't write everything, though! (Main Points, Main Ideas
Don't spend too long deciding if you should
write it down
Thin your notes
Don't write everything! Leave out words
things - include a key of specific-for-

The image above shows an attempted straightening and splitting. Due to spacing constraints and text color differences, the results are not perfect.

4.3 OCR Model - Worked on by Brydon

In the field of machine learning, OCR has many applications. This includes extracting typed text from images, which can be done with state-of-the-art models with very high accuracy. However, the problem we are looking at is handwritten text recognition, which is far more complicated. Due to differences between writing styles, and inconsistencies within one's own, this problem is far from solved. State-of-the-art models still struggle with messy handwriting. Luckily, our project does not require perfect accuracy, only a translation similar enough for an NLP model to understand. For this part of the project, we tried a few different things.

4.3.1 Tesseract

Early research on this project pointed heavily towards using the open source "Tesseract" OCR engine. What we failed to understand at the time, is that Tesseract has been trained to extract typed text, in particular fonts, from images; not handwriting. Meklit ran a few handwritten samples through the Tesseract engine, and received extremely poor results. Approximately 5-10% of the original words were accurately detected. Further research explained that it is possible to train Tesseract on new fonts, including handwriting. It is possible that after this training process, it would have given us the results we were after. However, the process was going to require a lot of work, so we pursued other options.

4.3.2 GPT-4

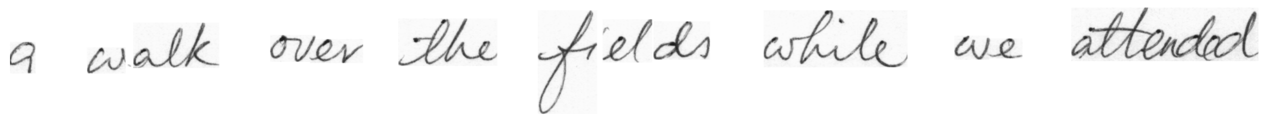
Second, we tried using GPT-4. Nathan ran a few examples through it, but found it to be very restrictive, inconsistent, and not very accurate when it did work. For these reasons, we abandoned this approach immediately.

4.3.3 Deep Learning

Next, we found a github repository with a deep learning handwritten sentence recognition model. (https://github.com/pythonlessons/mltu/tree/main/Tutorials/04_sentence_recognition).

Through a series of training and parameter experimentation, this ended up being the backbone of the OCR implementation in our final solution. The model was built using tensorflow, with a fairly complicated architecture. It consists of nine convolutional layers, followed by two bidirectional LSTM layers. This allows the model to extract high level features from input, and store dependencies required to fully identify characters. Finally, the output is passed through a softmax activation function. Additionally, a dropout rate of 0.2 is used to prevent overfitting.

The data we used to train this model was the sentences portion of the IAM Handwriting Database (https://fki.tic.heia-fr.ch/databases/download-the-iam-handwriting-database). We used approximately 12,000 single line labeled samples in the training set, about 1,500 samples in the validation set, and only 160 samples in the test set. The test set is so small, because we used that same set with our Google Cloud Vision model, which required API calls for each iteration. To improve training time, we limited each epoch to 500 steps of 4 samples. This means that each epoch included 2000 of the 12,000 training samples. Below is an example from the training set.


a walk over the fields while we attended

During the training process, we experimented with model complexity by varying the number of nodes in each layer. We tried three levels of complexity. These levels included around 34,000, 340,000, and 2,500,000 total trainable parameters. Each of these models used a learning rate of 0.001. These models were evaluated based on character error rate (CER), and word error rate (WER). A CER of 1 indicates that every predicted character was wrong. A CER of 0 indicates perfect accuracy.

	CER After 20 Epochs	WER After 20 Epochs	CER After 50 Epochs	WER After 50 Epochs
34,000	0.9974	1.0000	N/A	N/A
340,000	0.9262	1.0000	0.8028	1.0000
2,500,000	0.3189	0.6858	0.1483	0.4094

All error rates in the table above were calculated against the validation set. As you can see, the results between the models varied greatly. The 34,000 parameter model appeared to be too simple, and hardly showed any improvement after 20 epochs, so we abandoned it. The 340,000 parameter model did better, and the 2,500,000 parameter model did impressively well. It is important to mention that we did see slightly altered results between training sessions. I believe this is due to randomness in weight initialization, and favorability of gradient descent steps. To get a conclusive answer on how many trainable parameters are necessary, we would need to perform more and longer training sessions. However, our results suggest that the more complex model has higher potential.

Next, we experimented with learning rate. This hyperparameter dictates the size of each gradient descent step. Using the best performing model from the complexity testing (2,500,000 trainable parameters), we experimented with learning rates of 0.1, 0.01, and 0.001. Learning rates of 0.1 and 0.01 proved to be too high, and the model did not improve at all over 20 epochs in either case. We determined that a learning rate of 0.001 was the sweet spot, with relatively fast convergence, and impressive results, as seen in complexity testing.

4.3.4 Google Cloud Vision API

In addition to having a working deep learning model, we wanted to experiment with something state-of-the-art. For this we chose Google's Cloud Vision API. Setting this up required creating a project, installing gcloud, and writing a simple script to make API calls. It is hard to say whether this is the best model out there right now, as there is a lot of competition and we did not have time to try everything. Regardless, Cloud Vision comes pretrained, and offers a solid out of the box solution.

4.3.5 Comparing Models

After experimenting with all the methods above, we have three valid models. First is the 2,500,000 parameter, 50 epoch deep learning model that we trained. Second is Google Cloud Vision. Third is a pre-trained deep learning model that came with the github repository we used for our training. It has the same architecture and parameter count as our model, but was trained on a smaller portion of the IAM Handwriting Database, and with far more epochs. It was able to achieve a CER of 0.0035 and WER of 0.0171 against this subset.

To fairly assess these models, our test set consists of 160 samples from the IAM Handwriting Database, that none of the models have seen before. The table below shows the average CER and WER from each model across this set.

	Average CER	Average WER
Our DL Model	0.1384	0.3672
Pre-Trained DL Model	0.1486	0.4145
Google Cloud Vision	0.1579	0.3848

As you can see, all of the models performed pretty consistently against the test set. The pre-trained deep learning model was trained to extremely high accuracy against a subset of the IAM Handwriting Database, but performed worse than our model in this test. Despite being given a fraction of the epochs, the larger training set resulted in far better generalization to unseen samples. Google Cloud Vision had underwhelming performance. After some manual inspection of the results, it seems that Cloud Vision gave mostly reasonable responses, but lost a lot of accuracy due to having access to more character options, and simple things like whitespace.

4.4 NLP - Worked on by Nathan

The final stage of our project was using natural language processing on the OCR results to attempt to correct character errors, and summarize the notes. For this, we looked into the different models that were available from OpenAI. We looked into the time it took to process the request and the quality of the output for each model. We looked at all the API models that were available with GPT-3.5 and GPT-4. We found that 'gpt-3.5-turbo-16k' was giving us the most consistent results based on our prompts. It was fairly quick with a time of around 2-3 seconds, compared some GPT-4 models taking up to 15 seconds with less reliable outputs.

Implementation involved obtaining a key, and making API calls through a python script. Once this was set up, all we needed was the prompts. For error correction, we used:

"The following text is the result of a student's handwritten notes run through an OCR model. There are likely a lot of character errors. Rewrite the notes as they are, fixing character errors with your best guess as to what the original notes said. Do not include any extra phrases, simply output the rewritten notes." + (String containing full OCR response)

For summarization, we used:

"The following text is the result of a student's handwritten notes run through an OCR model. There are likely a lot of character errors." + (String containing full OCR response)

"The following text is a list of segments from the original notes that were deemed important and highlighted." + (String containing only highlighted phrases OCR response)

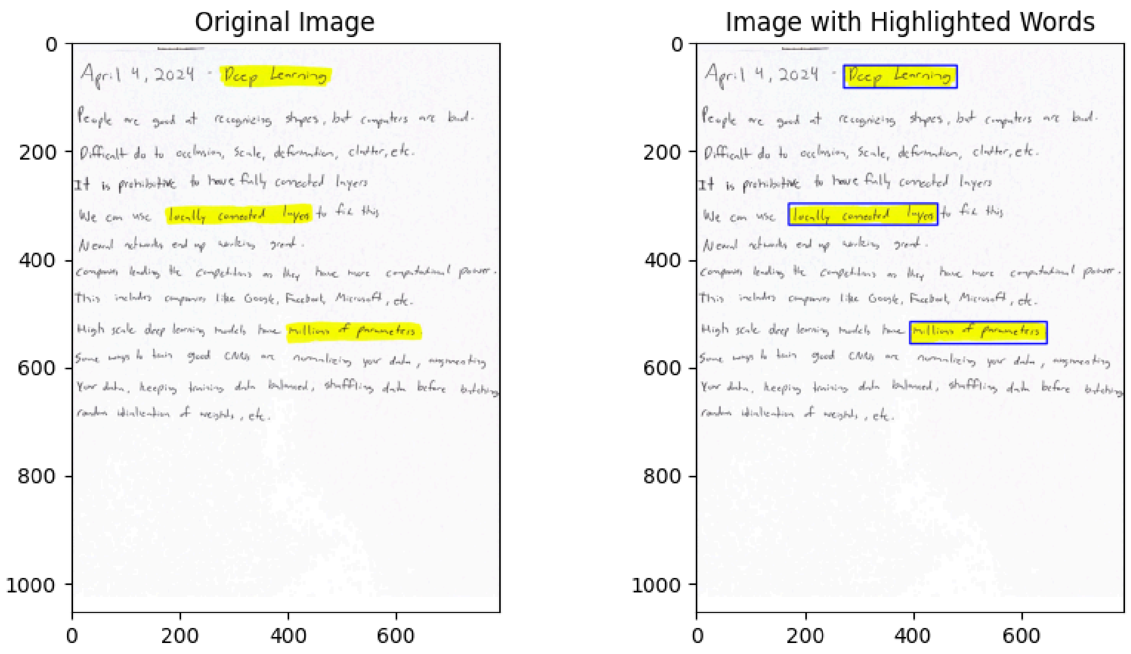
"Doing your best to look through the character errors, summarize these notes in a short way that would cater to a student looking to review them for studying purposes later. Place emphasis on the highlighted segments. Try your best to identify all highlighted segments. If you cannot interpret what it might say as it relates to the full notes, ignore it. Do not quote the uncorrected text. Do not include any extra phrases, simply output the summarized notes. Ensure the summarization does not exceed the length of the full notes."

By the nature of GPT, the results are somewhat varied, however these prompts yield acceptable responses consistently, so long as the input strings are interpretable.

5. Results

Overall, we are very satisfied with the results we were able to achieve in this project. Our models are not perfect, but perform quite well on carefully captured handwritten pages. For highlighted segmentation, we were able to achieve very high accuracy using a hardcoded color approach. Additionally, we achieved results that suggest a more dynamic machine learning method is possible with further refinement. Our image preprocessing works as intended for specific types of inputs. The OCR model we trained is able to compete with state-of-the-art products for single line inputs found in the IAM Handwriting Database. Finally, we successfully integrated NLP, and achieved a solution that is able to convert and summarize handwritten notes, with emphasis on highlighted segments, as we set out to do. To demonstrate its capabilities, and paint a clear picture of the full project, here is a full example on a page written by us.

Original Image and Highlighted Segmentation:



April 4, 2024 - Deep Learning

People are good at recognizing shapes, but computers are bad.

Difficult do to occlusion, Scale, deformation, clutter, etc.

It is prohibitive to have fully connected layers

We can use locally connected layers to fix this

Neural networks end up working great.

Companies leading the competitions as they have more computational power.

This includes companies like Google, Facebook, Microsoft, etc.

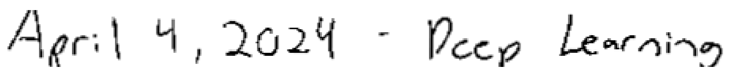
High scale deep learning models have millions of parameters.

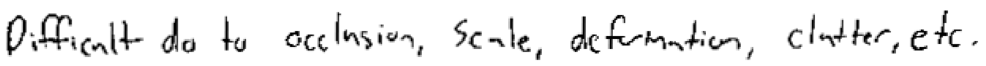
Some ways to train good CNNs are normalizing your data, augmenting

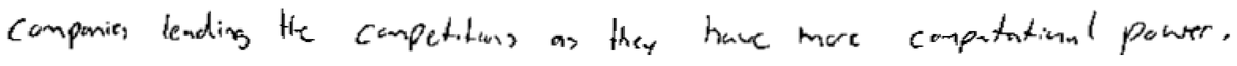
Your data, keeping training data balanced, shuffling data before batching

random initialization of weights, etc.

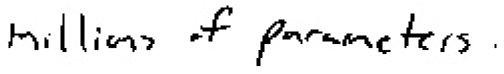
A few examples of single line files generated:


April 4, 2024 - Deep Learning


Difficult do to occlusion, Scale, deformation, clutter, etc.


Companies leading the competitions as they have more computational power.

A example of a highlighted segment file generated:


millions of parameters.

Results of full text OCR using our 50 epoch model.

Aeril 4 , 200l- Reep learing

People are god at recogierng sthpes , but componters are band .

pffielt do to oculasion , Seat , deferntior , clter,ecte.

If is protibntive to thave fily coneoted layers

We can use loually cometed layes to fil this

Wenat natuohey end uo woreing gest .

compaies lending the cumpeters as the have mhore copestatinl pour .

this inctites cempanrs the Gest , raebed , trceset , ct . "

TseMA, RR , exe .

Migh sele dep lerning moded thave rillion, of parenecters .

sore was o tain god censs are nomnlieing yor aste , asmentin "

ter dite , heping trnining datk bortaned , strfting disth before bottina

Results of highlighted segment OCR:

nillios it paraneters .

Cocally comected layes

Pcep learning

Results of rewriting using NLP:

April 4, 2001 - Deep Learning

People are good at recognizing shapes, but computers are bad.

Difficult due to occlusion, noise, deformation, clutter, etc.

If it's prohibitive to have fully connected layers

We can use locally connected layers to fill this

Want natural hierarchy to working best.

Companies leading the computers as they have more computational power.

This includes companies like Google, Baidu, Facebook, etc.

Might select deep learning models that have millions of parameters.
Some ways to obtain good results are normalizing your input, augmenting it, shuffling data before training.

Results of summarization with emphasis on highlighted segments using NLP:

- People are good at recognizing shapes, but computers are bad
- Possible reasons: occlusion, noise, deformation, etc.
- It's expensive to have fully connected layers
- Solution: locally connected layers
- Deep learning models can have billions of parameters
- Training data should be preprocessed and normalized
- Emphasis on highlighted segments:
    - Locally connected layers
    - Deep learning and billions of parameters

6. Conclusion

This project proved to be a challenging, yet rewarding undertaking. Handwritten text recognition is a difficult problem in the world of machine learning, that not even tech leaders have fully solved. Despite that, we were able to achieve what we set out to do, by providing a handwritten notes summarizer, with emphasis on highlighted segments. The results depend heavily on the quality and specific format of the input image. If we had more time with this project, we would spend it improving the quality of each part. That would involve continuing our machine learning experimentation for highlighted segmentation, improving our line splitting algorithm, training our OCR model for longer, and refining our prompts. We also believe that allowing users to train the OCR model on their specific handwriting could lead to vast improvements in that regard. In conclusion, while there are areas for further enhancement, our efforts have yielded a final product that we can be proud of, in such a challenging domain.

References

Some of these are academic, others are not.

AlKendi, W., Gechter, F., Heyberger, L., & Guyeux, C. (2024). Advancements and Challenges in Handwritten Text Recognition: A Comprehensive Survey. *Journal of Imaging*, *10*(1), 18. https://doi.org/10.3390/jimaging10010018

Alex, (2016, January 24). *Split text lines in scanned document*. Stack Overflow. https://stackoverflow.com/questions/34981144/split-text-lines-in-scanned-document

Cook, C. (2023, March 14). *Extract highlighted text from a book using Python*. DEV Community. https://dev.to/zirkelc/extract-highlighted-text-from-a-book-using-python-e15

Dougherty, E. (2018). *Mathematical Morphology in Image Processing*. CRC Press.

Handwritten sentence recognition with TensorFlow. PyLessons. (n.d.). https://pylessons.com/handwritten-sentence-recognition

IAM handwriting database. Research Group on Computer Vision and Artificial Intelligence - Computer Vision and Artificial Intelligence. (2019). https://fki.tic.heia-fr.ch/databases/iam-handwriting-database

Jackdaw(2016, July 1). Using tesseract for handwriting recognition. Stack Overflow. https://stackoverflow.com/questions/39556443/using-tesseract-for-handwriting-recognition

Ma, Q., & Huang, X. (2022). Research on recognizing required items based on opencv and machine learning. *SHS Web of Conferences*, *140*, 01016. https://doi.org/10.1051/shsconf/202214001016

Matcha, A. C. N. (2024, March 26). Handwriting recognition with ML (an in-depth guide). Nanonets Intelligent Automation, and Business Process AI Blog. https://nanonets.com/blog/handwritten-character-recognition/#datasets

Pythonlessons. (n.d.). Pythonlessons/MLTU: Machine Learning Training Utilities (for tensorflow and pytorch). GitHub. https://github.com/pythonlessons/mltu

Rakshit, S., Basu, S., & Ikeda, H. (2010, March 30). *Recognition of Handwritten Textual Annotations using Tesseract Open Source OCR Engine for information Just In Time (iJIT)*. ArXiv.org. https://doi.org/10.48550/arXiv.1003.5893

Smith, R. (2007). An Overview of the Tesseract OCR Engine. *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2*. https://doi.org/10.1109/icdar.2007.4376991