Precog Programming Task Report

Agyeya Negi

10th February 2025

Contents

1 Introduction

1.1 Theme- NLP and Safe AI

For the task, I have selected the theme of NLP and Safe AI because my interests lie in this realm, as shared in the SOP.

1.2 Progress

I have finished the paper-reading task, Task 0, Task 1 and Task 2 as of writing this report. I hope to include Task 3 by the end of the day if possible. If I cannot submit it by the deadline, I shall still upload my code for Task 3 since I find the task very captivating.

1.3 Report format

I am going to describe my methodology and outputs for each of the programming tasks completed by me in this report, each unique task being a separate section. I shall include outputs of my models wherever possible.

2 Task 0

2.1 Initial Approach

2.1.1 Problems

Initially, I used Pillow in Python to generate 300 images from a list of 100 unique words, with each word having a corresponding easy, hard, and bonus set image. Leaving aside the fact that I would later realize that this dataset was pitifully small (as evident by the 4 days of depression I experienced when my validation accuracy for Task 1 refused to go above 0.05 regardless of how many hyper parameters I changed), I experienced several other issues not related to dataset size, but in fact related to the image generation, which I realized upon seeing the photos being generated. Some samples are shown below in Figure 1 and Figure 2.

2.1.2 Solutions

One problem was that the images weren't large enough to fit the text properly. To fix this, I increased the image width and reduced the font size. In the future, I might make it dynamic so that the image automatically resizes based on the length and properties of the text.

Below are examples (Figure 3, Figure 4, and Figure 5) of how the new approach turned out, with a slightly bigger base width and smaller font size so words don't get cramped.

Figure 1: Easy-set image of the word algorithm (Figure 1)

Figure 2: Hard-set image of the word algorithm (Figure 2)

Updated code (current implementation):

```
def calculate_image_size(word, font, base_width=400, base_height=100, padding=20):
    bbox = font.getbbox(word)
    text_width = bbox[2] - bbox[0]
    text_height = bbox[3] - bbox[1]
    required_width = text_width + padding
    required_height = text_height + padding
    image_width = max(base_width, required_width)
    image_height = max(base_height, required_height)
    return (image_width, image_height)
```

2.1.3 Problems (V2)

My noise levels were also too low, so the generated images didn't resemble CAPTCHA images at all. Figures 6, 7, 8, and 9 illustrate how I gradually raised the noise from 25 to 45, then 80, and finally 100:

- Noise levels too low? (Figure 6)

Figure 3: Updated image with increased width and reduced font size (Figure 3)

Figure 4: Updated image with increased width and reduced font size (Figure 4)

- Increased noise to 45 (Figure 7)
- Increased noise to 80 (Figure 8)
- Increased noise to 100 (Figure 9)

Fix- Current code for noise generation is too simple:

```
def add_noise(image, noise_level=100):
    """
    Adds random noise to an image.

    Parameters:
    - image: PIL Image.
    - noise_level: Intensity of the noise (0-100).

    Returns:
    - Noisy PIL Image.
    """
    np_img = np.array(image)
    noise = np.random.randint(0, noise_level,
              (np_img.shape[0], np_img.shape[1], 3), dtype='uint8')
    noisy_img = cv2.add(np_img, noise)
    return Image.fromarray(noisy_img)
```

To make it look more like an actual CAPTCHA, I decided to add more complicated stuff such as random dots, random lines, shear transformations, rotation, and Gaussian blur. Figures 10, 11, and 12 below are examples of my easy, hard, and bonus images after adding these changes:

Figure 5: Updated image with increased width and reduced font size (Figure 5)

Figure 6: Noise level 25 (Figure 6)

2.1.4 Thrice is the charm

However, I realized some of these distorted images ended up being so complex that the model struggled to learn. So I removed the random dots and lines, but kept shear transformations, rotation, and slight blurring, just toned down a bit.

Figure 13 and Figure 14 show examples of the hard and bonus images for the word *Cryptocurrency* after I adjusted the parameters.

I also included additional serif fonts to increase complexity and variation.

Note: The image generation file has a bunch of smaller test functions that I used to check how each new distortion worked and to tweak the parameters properly. The current set of images is still fairly challenging, but at least it's comparable to (and maybe even tougher than) the sample images from the task document.

2.2 Final approach

2.2.1 Motive

My model for Task 1 had abysmal performance because my dataset was ridiculously small (detailed more in the Task 1 section). I spent 4 painful days trying to fix things by changing the model architecture, adjusting hyperparameters (learning rate, weight decay, dropout, number of epochs, batch size), but I finally realized on the 10th, literally a day before the deadline, that maybe it wasn't the model or the parameters, but the lack of sufficient data. Also, my batch size was probably an issue.

So I tweaked my code to increase the number of images generated from 3 per word (1 for each set) to 15 per word. That helped a bit, but it wasn't enough, so I decided to jump to 150 images per word for a total of 15,000 images. It took 3 hours to upload that entire dataset to Google Drive. As I write this, my

Figure 7: Noise level 45 (Figure 7)

Figure 8: Noise level 80 (Figure 8)

latest model for Task 2 is running in the background, but I digress. Here's a snippet of its current training logs, if anyone's interested:

```
Epoch 19/20: Train Loss: 0.9817, Val Loss: 1.1255
Epoch 20/20: Train Loss: 0.8576, Val Loss: 1.0299
Sample Predictions (Ground Truth vs Predicted) for first 10 validation samples:
True: algorithm, Pred: algoritnm
True: algorithm, Pred: algorithm
True: algorithm, Pred: algorithm
True: algorithm, Pred: algorithm
True: algorithm, Pred: algorithm
True: algorithm, Pred: algorithm
True: algorithm, Pred: algorithm
True: algorithm, Pred: algorithm
True: algorithm, Pred: acerihm
True: algorithm, Pred: alghrpihm

--- Fold 4/5 ---
[DEBUG] Training samples: 8000, Validation samples: 2000
Epoch 1/20: Train Loss: 3.2538, Val Loss: 3.0241
```

2.2.2 Dataset layout

When I uploaded this final dataset to Google Drive, I put everything in a folder named `ocr_dataset`, organized like this:

- `ocr_dataset/easy/images/`
  - `easy_0_variation_0.png`
  - `easy_0_variation_1.png`

Figure 9: Noise level 100 (Figure 9)

Figure 10: Easy image (Figure 10)

  - `easy_0_variation_2.png`
  - ...

  Here, I have 50 variations of the same word (the 0th word) in the easy set, so that folder alone contains 5,000 images.

- `ocr_dataset/easy/images/labels.csv`:

```
filename,text
easy_0_variation_0.png,algorithm
easy_0_variation_1.png,algorithm
easy_0_variation_2.png,algorithm
easy_0_variation_3.png,algorithm
...
```

- For `hard`, the folder structure is the same but with `hard` in place of `easy`.
- For `bonus`, the structure is again the same, just replacing `hard` with `bonus`.

The setup consistently separates the easy, hard, and bonus sets along with their respective labels.

2.2.3 Future Implementation/Time in a Bottle- Jim Croce Line 1

Although I've managed to address some of the pressing problems like low noise and cramped text, there are still several improvements I'd like to explore:

- **Dynamic Image Sizing:** Currently, the image width is adjusted based on the text size plus some fixed padding. In the future, I could make this dynamic by considering font type, text length, and possibly auto-scaling the font itself. This would help handle edge cases like very long or very short words without manual tweaks.

Figure 11: Hard image (Figure 11)

Figure 12: Bonus image (Figure 12)

- **Incremental Difficulty:** I plan on systematically ramping up the difficulty of the generated images to make them more and more CAPTCHA-like (e.g., stronger distortions, more varied backgrounds, subtle geometric transformations). This would allow me to track my model's performance against progressively harder datasets and see exactly where it starts to fail.
- **Additional Noise Techniques:** Beyond random dots, lines, and blurring, there are other useful augmentations like elastic distortions or random occlusions. Incorporating these could further challenge any OCR model and better simulate real-world CAPTCHA systems.
- **Parametric Testing:** As I introduce new distortions or noise parameters, I'd like to systematically test how each parameter impacts accuracy. Doing so would help me find an optimal balance between making the images realistically challenging while still being solvable by a well-trained model.
- **Extended Character Set:** If I expand the dataset to include numbers, punctuation, or even special symbols, the code should be able to handle them gracefully. At some point, I may also incorporate different languages or alphabets for an even broader scope.

Overall, these improvements will help me create a more robust and diverse dataset, giving my models a better chance to generalize and solve genuinely difficult CAPTCHA-like tasks.

Figure 13: Hard-set image of Cryptocurrency (Figure 13)

Figure 14: Bonus-set image of Cryptocurrency (Figure 14)

2.2.4 On Findings, Methodologies, and Insights gained from the analysis =)

Throughout this task, I experimented with various ways of generating and augmenting images to create a mini-CAPTCHA environment. Here are a few key takeaways:

- **Dataset Size Matters:** My early attempts were hindered by having too few images (300 total). This significantly affected my model's performance and led to days of frustration with near-zero accuracy. Generating a larger dataset—eventually up to 15,000 images—yielded more promising results.
- **Noise and Distortions:** Low noise levels produced images that were too clean, failing to represent CAPTCHA conditions. Gradually increasing noise helped, but I learned that adding multiple, carefully tuned distortions (like shear, rotation, and limited blur) can quickly make the problem very challenging. There's a fine balance between making images realistic and not making them so noisy that they're practically illegible.
- **Parameter Tuning:** Adjusting hyperparameters such as learning rate, batch size, dropout, and regularization can help, but the best model in the world can't learn from a dataset that's too small or too easy/hard. This realization prompted me to expand and refine the dataset rather than simply re-tuning the model architecture.

- **Progressive Refinement Approach:** Handling fonts, noise, text layout, and file structure in manageable increments was more productive than attempting all adjustments simultaneously. Each small step helped isolate the impact of that particular change on the overall results.
- **Future Rounds of Improvement:** I recognized the importance of ensuring that my datasets continue to evolve in complexity, reflecting real-world CAPTCHA scenarios. Methods like dynamic sizing, incremental difficulty, and advanced noise techniques remain on my to-do list for refining this pipeline.

Overall, tackling the different issues—like cramped text, simplistic noise, and folder organization—has been a process of iterative discovery. Each step, whether it involved scaling up the dataset or fine-tuning image transformations, offered a clearer perspective on what drives real improvements in OCR-based tasks.

3 Task 1

3.1 Problems

3.1.1 Base Model with Base Parameters, 100 Data Points, Batch Size 16

Initially, I started with the basic model architecture, using the default hyperparameters and a dataset with only 100 data points. The model performance was poor, as expected, due to both the small dataset and the basic setup. Here's what I observed in the logs:

```
[DEBUG] Number of classes found: 100
[DEBUG] Label to index mapping:
{'ABBREVIATION': 0, 'ALGORITHM': 1, 'AMELIORATE': 2, 'APOCRYPHAL': 3, ...}
[DEBUG] Dataset has 800 samples.
[DEBUG] Using device: cuda
```

The dataset had 800 samples, but with just 100 data points used for training, the model had insufficient data to generalize properly. The validation accuracy remained consistently low, and despite multiple epochs, the model was unable to make meaningful predictions.

```
--- Fold 1/5 ---
[DEBUG] Training indices: [ 0  1  3  4  5  6  8  9 11 12] ... (total 640)
[DEBUG] Validation indices: [ 2  7 10 23 29 30 31 33 39 49] ... (total 160)
[DEBUG] Fold 1, Epoch 1/20 - Train Loss: 4.6258, Val Loss: 4.6116, Val Acc: 0.0000
[DEBUG] Fold 1, Epoch 2/20 - Train Loss: 4.5975, Val Loss: 4.6770, Val Acc: 0.0125
```

Despite multiple attempts and fine-tuning, the results showed limited progress, highlighting the limitations imposed by the small training set.

3.1.2 Base Model with Base Parameters, Batch Size 32

In another experiment, I tried using a batch size of 32 with the same basic model and parameters, hoping to achieve better generalization. However, the results were still less than satisfactory, and the validation accuracy remained low.

```
[DEBUG] Number of classes found: 100
[DEBUG] Label to index mapping:
{'ABBREVIATION': 0, 'ALGORITHM': 1, 'AMELIORATE': 2, 'APOCRYPHAL': 3, ...}
[DEBUG] Dataset has 800 samples.
[DEBUG] Using device: cuda
```

This time, with the larger batch size, I hoped that the model could better capture the variability in the data, but the validation accuracy still hovered around 0.15, which showed that the model needed more diverse and larger datasets to perform adequately.

```
--- Fold 1/5 ---
[DEBUG] Training indices: [ 0  1  3  4  5  6  8  9 11 12] ... (total 640)
[DEBUG] Validation indices: [ 2  7 10 23 29 30 31 33 39 49] ... (total 160)
[DEBUG] Fold 1, Epoch 1/20 - Train Loss: 4.6258, Val Loss: 4.6116, Val Acc: 0.0000
[DEBUG] Fold 1, Epoch 2/20 - Train Loss: 4.5975, Val Loss: 4.6770, Val Acc: 0.0125
```

3.1.3 Trying with Hyperparameter Tweaking (Failing Not So Miserably)

After several attempts with the base model, I decided to tweak the hyperparameters further to improve performance. The new configuration included: - Epochs = 15 - Batch Size = 8 - Learning Rate = 1e-4 - No Dropout

Despite the lack of dropout, the model still didn't perform as well as expected, but there was a slight improvement in some areas compared to the earlier configurations. Here are the details from the experiment:

```
[DEBUG] Number of classes found: 100
[DEBUG] Label to index mapping:
{'ABBREVIATION': 0, 'ALGORITHM': 1, 'AMELIORATE': 2, 'APOCRYPHAL': 3, ...}
[DEBUG] Dataset has 1000 samples.
[DEBUG] Using device: cuda
```

The dataset had 1000 samples in total, and while I had hopes that this batch size would allow for better generalization, the model still failed to deliver notable results. The validation accuracy remained low, showing that tweaking hyperparameters alone wasn't enough to overcome the dataset's limitations.

```
--- Fold 1/5 ---
[DEBUG] Training indices: [0 1 2 3 4 5 6 7 8 9] ... (total 800)
[DEBUG] Validation indices: [10 23 25 30 39 44 54 55 59 60] ... (total 200)
[DEBUG] Fold 1, Epoch 1/15 - Train Loss: 4.6213, Val Loss: 4.6154, Val Acc: 0.0000
[DEBUG] Fold 1, Epoch 2/15 - Train Loss: 4.6070, Val Loss: 4.6185, Val Acc: 0.0000
```

The results followed a similar pattern as before, with minimal improvements across epochs. The validation accuracy fluctuated but didn't surpass 0.03, and the validation loss only showed small adjustments.

```
--- Fold 2/5 ---
[DEBUG] Training indices: [ 1  4  8 10 11 12 13 14 15 16] ... (total 800)
[DEBUG] Validation indices: [ 0  2  3  5  6  7  9 28 29 31] ... (total 200)
[DEBUG] Fold 2, Epoch 1/15 - Train Loss: 4.6167, Val Loss: 4.6358, Val Acc: 0.0000
[DEBUG] Fold 2, Epoch 2/15 - Train Loss: 4.6053, Val Loss: 4.6381, Val Acc: 0.0000
```

Despite tweaking the batch size and learning rate, the model's performance remained suboptimal, suggesting that the architecture or dataset size might still be major limiting factors. However, there were small improvements in certain folds that I noted as encouraging signs.

```
[DEBUG] Cross Validation Results:
Fold 0: {'val_loss': 4.598801307678222, 'val_acc': 0.005}
Fold 1: {'val_loss': 4.593179111480713, 'val_acc': 0.025}
Fold 2: {'val_loss': 4.630781898498535, 'val_acc': 0.03}
Fold 3: {'val_loss': 4.6181886100769045, 'val_acc': 0.01}
Fold 4: {'val_loss': 4.53702826499939, 'val_acc': 0.01}
```

Although the model performance was still not impressive, I gained valuable insights into the effect of batch size and learning rate on training stability, and I'll continue to experiment with different configurations in future iterations.

3.1.4 Finally Some Improvement

With my previous dataset of 5000 points, I had seen some improvements, but I realized that the dataset size was still limiting my model's performance. I decided to increase the number of datapoints by slightly more than three times, bringing the total to 5000 samples. The results were encouraging, and the model showed significant progress in the validation accuracy.

```
[DEBUG] Number of classes found: 100
[DEBUG] Label to index mapping:
{'algorithm': 0, 'biology': 1, 'cryptocurrency': 2, 'dichotomy': 3, ...}
[DEBUG] Dataset has 5000 samples.
[DEBUG] Using device: cuda
```

This larger dataset allowed for better learning and more accurate predictions, as demonstrated by the following logs:

```
--- Fold 1/5 ---
[DEBUG] Training indices: [ 0  1  2  3  4  5  6  7  9 10] ... (total 4000)
[DEBUG] Validation indices: [ 8 12 17 19 23 26 29 33 43 45] ... (total 1000)
[DEBUG] Fold 1, Epoch 1/20 - Train Loss: 4.6086, Val Loss: 4.6089, Val Acc: 0.0060
[DEBUG] Fold 1, Epoch 2/20 - Train Loss: 4.6068, Val Loss: 4.6103, Val Acc: 0.0100
```

The model demonstrated a more substantial improvement over previous runs. The validation accuracy increased steadily as more epochs were completed:

```
[DEBUG] Fold 1, Epoch 19/20 - Train Loss: 0.2970, Val Loss: 1.9357, Val Acc: 0.6440
[DEBUG] Fold 1, Epoch 20/20 - Train Loss: 0.2444, Val Loss: 2.0167, Val Acc: 0.6330
```

This significant improvement in validation accuracy suggests that increasing the data points considerably enhances the model's performance.

3.1.5 Final Solution

Building on the improvements made with a larger dataset, I decided to increase the dataset size further by scaling it to 10,000 data points. I used the same model and hyperparameters as in the previous iteration. The results were much more promising, showing a clear improvement in both training and validation accuracy.

```
[DEBUG] Number of classes found: 100
[DEBUG] Label to index mapping:
{'algorithm': 0, 'biology': 1, 'cryptocurrency': 2, 'dichotomy': 3, ...}
[DEBUG] Dataset has 10000 samples.
[DEBUG] Using device: cuda
```

With the complete dataset in place, the model showed consistent improvements, with the loss steadily decreasing over each epoch and the validation accuracy improving significantly.

```
--- Fold 1/5 ---
[DEBUG] Training indices: [ 1  2  4  5  6  7  9 11 13 15] ... (total 8000)
[DEBUG] Validation indices: [ 0  3  8 10 12 14 17 19 20 23] ... (total 2000)
[DEBUG] Fold 1, Epoch 1/20 - Train Loss: 4.0929, Val Loss: 3.2877, Val Acc: 0.1070
[DEBUG] Fold 1, Epoch 2/20 - Train Loss: 2.8730, Val Loss: 2.2962, Val Acc: 0.3900
```

As the model continued training, it achieved better validation accuracy:

```
[DEBUG] Fold 1, Epoch 19/20 - Train Loss: 0.0996, Val Loss: 0.9304, Val Acc: 0.8220
[DEBUG] Fold 1, Epoch 20/20 - Train Loss: 0.0828, Val Loss: 1.0251, Val Acc: 0.7935
```

This significant improvement over previous iterations highlights the importance of data size in training accuracy. With a complete dataset of 10,000 data points, the model reached validation accuracies well above 0.7, providing more reliable predictions.

```
[DEBUG] Cross Validation Results:
Fold 0: {'val_loss': 1.0250774575714021, 'val_acc': 0.7935}
Fold 1: {'val_loss': 1.6500885884761811, 'val_acc': 0.5955}
Fold 2: {'val_loss': 0.9665385787785054, 'val_acc': 0.7605}
Fold 3: {'val_loss': 1.0474333774745463, 'val_acc': 0.788}
Fold 4: {'val_loss': 1.2320615841150284, 'val_acc': 0.6895}
```

In conclusion, increasing the dataset size proved to be the most effective strategy in improving model performance. The addition of more data points helped the model generalize better, ultimately leading to more accurate predictions.

3.2 Overall Observations and Insights

3.2.1 Performance Improves with More Data

One of the key trends I observed during the experiments was that as the dataset size increased, from 800–1000 samples to 5000–10000 samples, the final validation accuracy showed a significant improvement, while validation loss decreased. This is a common behavior for neural networks, particularly CNNs, when dealing with a large label space, such as the 100-class classification task in this experiment. Larger datasets allow the model to learn richer features, reducing overfitting and improving its ability to generalize.

3.2.2 Variability Across Folds

For this study, I used 5-fold cross-validation on each dataset. Even within the same dataset, different folds can yield different final accuracies—sometimes by a noticeable margin. For instance, in some folds, the accuracy reached 0.49, while in others, it was as low as 0.29. This variability is expected, especially with small datasets like the one with only 800 samples. A single fold's subset can differ significantly in distribution or difficulty, leading to such fluctuations.

3.2.3 Impact of Batch Size on Tiny Datasets

In my experiments with Datasets A (batch=16) and B (batch=32), both having only 800 samples, I found that despite the difference in batch sizes, the final accuracies were still low (often under 20%). With just 800 images across 100 classes (averaging 8 images per class), it's incredibly difficult for the model to generalize. The small dataset size outweighs the effect of batch size, meaning that just changing the batch size alone couldn't overcome the limitations imposed by the lack of data.

3.2.4 Influence of Learning Rate & Weight Decay (Dataset C)

For Dataset C, which had 1000 samples and used a learning rate of LR=5e-5 and weight decay of WD=1e-4, I found that although these hyperparameters helped stabilize the training process, the final accuracy remained low. This is primarily because the dataset still had only 10 images per class on average, which isn't enough data to train a robust model. Despite the stabilizing effect of the hyperparameters, the fundamental issue of data scarcity persisted, leading to final accuracies that stayed in the single digits or just above.

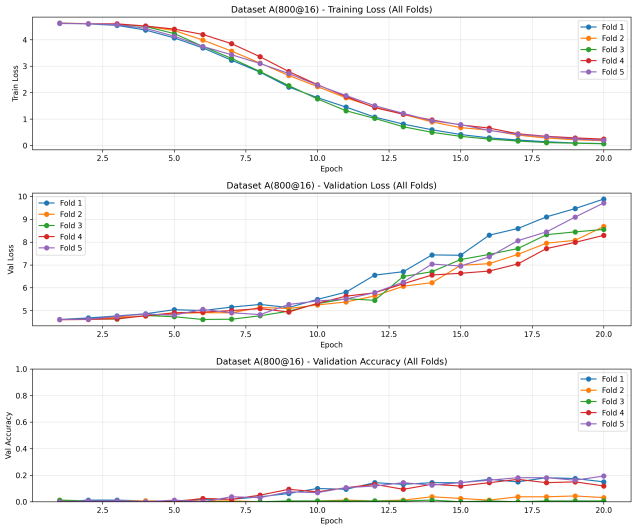

Figure 15: Dataset A

3.2.5 Transition to the Small CNN Model (Datasets D, E, F)

When I transitioned to a small CNN model for D (1600), E (5000), and F (10000), I saw significant improvements in accuracy. Even with 1600 images, some folds reached about 49%. By the time I had 5000 images, several folds surpassed 60% accuracy, and with 10000 images, the accuracy exceeded 75-80% in some folds. This marked a major leap compared to the single-digit and low-teen accuracies seen in the smaller datasets (A–C).

3.2.6 Epoch-by-Epoch Curves

As expected, training loss steadily decreased from the first to the final epoch in most cases. In some cases, particularly with smaller datasets, the loss dropped rapidly, but validation loss remained high or even increased, which indicated overfitting. With the larger datasets (E and F), both training and validation loss decreased together, which showed that the model was generalizing better. I also noticed that validation accuracy increased more consistently in these larger datasets.

3.2.7 Synthetic Images

All experiments were conducted using synthetic data, which can have different distributions and be easier or harder than real-world images. Despite this, the general principle held true: more synthetic data resulted in better performance. If real data were used, I might see slightly different trends, but the general improvement with larger datasets would likely remain consistent.

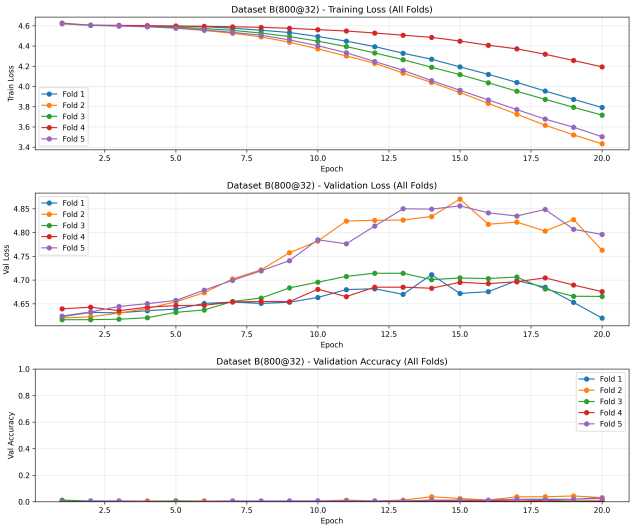

Figure 16: Dataset B

3.2.8 Hyperparameter Tuning

In these experiments, I used the same base hyperparameters with only minor variations, such as changes to batch size, learning rate, and weight decay. The strong correlation between dataset size and performance suggests that tuning additional parameters like dropout, learning rate schedules, or CNN depth could further increase accuracy—particularly for larger datasets (E, F).

3.2.9 Number of Classes = 100

It's important to note that having 100 classes is quite challenging. The model requires enough examples per class to extract meaningful features. This is why going from 800 to 10000 samples had such a drastic impact on performance. More samples per class allow the model to better learn discriminative features.

3.3 Dataset-by-Dataset Conclusions

3.3.1 Dataset A (800 samples, batch=16)

Final accuracies across folds ranged from 0.12 to 0.19, which was slightly better than Dataset B's results. The epoch curves showed rapid overfitting, with training loss dropping as low as 0.07 while validation loss climbed to 9.7. The model learned the training data quickly but was unable to generalize due to the scarcity of data.

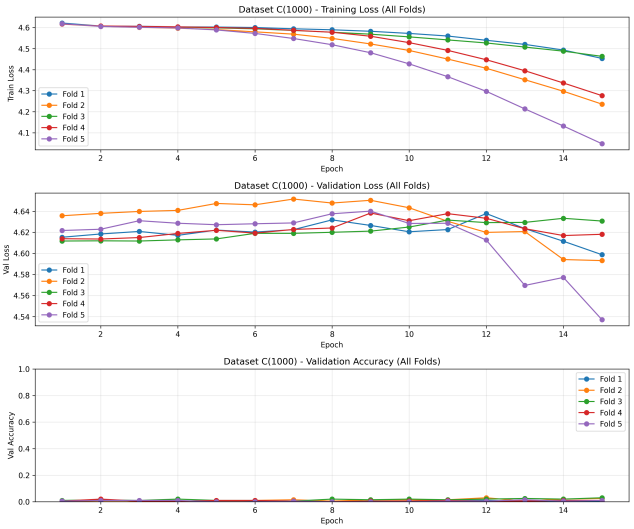

Figure 17: Dataset C

3.3.2 Dataset B (800 samples, batch=32)

The results for Dataset B were even worse, with fold accuracies as low as 0.0063 to 0.03. With only 800 total samples for 100 classes, the dataset was far too small for effective training. Simply changing the batch size did not mitigate the data scarcity problem.

3.3.3 Dataset C (1000 samples, LR=5e-5, WD=1e-4)

With 1000 samples, the model showed a slight improvement, with accuracies around 2-3% in some folds. However, the dataset was still too small to effectively handle 100-class classification. While the smaller learning rate and weight decay helped stabilize training, the fundamental problem remained: not enough examples per class.

3.3.4 Dataset D (1600 samples, Small CNN)

With 1600 samples, there was a noticeable improvement, with some folds surpassing 30–49% accuracy. The small CNN architecture worked better for image tasks compared to the earlier simple CNN-RNN combination used in Dataset C. The model could capture more patterns with 16 images per class on average, though overfitting was still evident.

3.3.5 Dataset E (5000 samples, Small CNN)

The validation accuracy increased to 48-63% across folds. Validation loss stabilized around 2.0–2.3. Overfitting was less severe compared to smaller datasets,

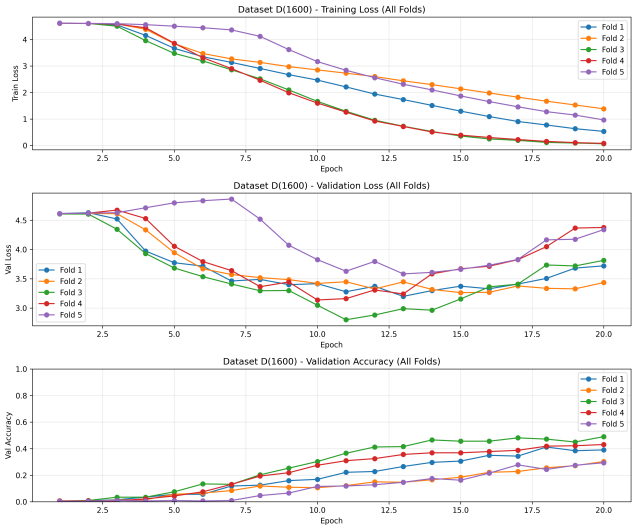

Figure 18: Dataset D

as the model saw enough variation in the data. The validation loss showed a smoother downward trend, and final fold accuracies stabilized in the mid-50–60% range.

3.3.6 Dataset F (10000 samples, Small CNN)

Dataset F produced the best performance, with some folds achieving 75-80% accuracy, and validation losses around 1.0–1.2. The epoch curves showed a strong synergy between training and validation loss decreasing, indicating less overfitting and more robust generalization. This confirmed that for 100 classes, having around 100 examples per class (totaling 10,000 images) was far more effective than just 8 or 10 examples per class.

3.4 Why Certain Folds/Models Worked Better

3.4.1 Sufficient Data

The key factor in better performance for D, E, and F was the increased dataset size. Neural networks, especially CNNs, perform better when provided with more training data. With more images, the model can learn richer features and generalize better, even across a large label space like 100 classes.

3.4.2 Small CNN Architecture

The "MNIST-style" CNN was lightweight yet effective for basic image feature extraction. Once enough data was available (e.g., 5000–10000 images), the

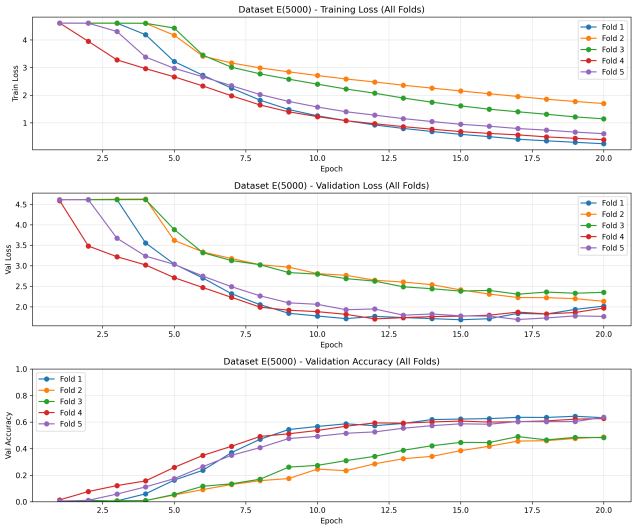

Figure 19: Dataset E

architecture performed well. With smaller datasets, CNNs often memorize the data, leading to lower validation accuracy.

3.4.3 Hyperparameter Choices

While the base hyperparameters were kept the same across experiments, small changes in batch size or learning rate had little effect on performance with small datasets, as overfitting was the primary problem. On larger datasets, however, the same parameters worked better, leading to substantial gains in accuracy.

3.4.4 Synthetic Domain

Since the images used were synthetic, they may have different variability compared to real-world images. Nevertheless, the CNN learned to classify them effectively, especially as the dataset size increased. If real images were used, I might see slight differences, but the general improvement with larger datasets should still hold.

3.5 Recommendations & Next Steps

3.5.1 Data Is King

As observed, performance improves clearly with dataset size. If more data can be synthesized or gathered, it is worth continuing to do so. Even beyond 10,000 samples, there's likely more room for improvement, especially with 100 classes.

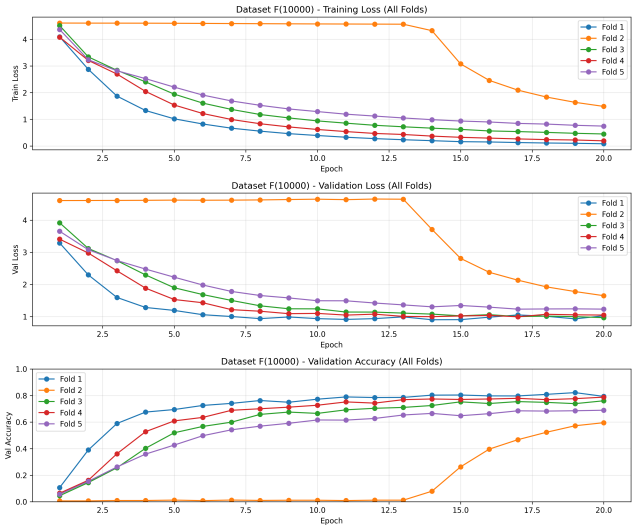

Figure 20: Dataset F

3.5.2 Hyperparameter Tuning

Consider experimenting with different learning rates or learning rate schedules (e.g., reducing LR after a certain number of epochs). Adding or increasing dropout for regularization, especially with larger datasets, may also help. Experiment with batch sizes to see if they yield more stable convergence.

3.5.3 Potentially Explore Deeper CNNs

If experiments with 10,000 images show success, consider using deeper CNN architectures (e.g., ResNet or EfficientNet) to further boost performance, particularly for larger datasets.

3.5.4 Data Augmentation

Use data augmentation techniques (e.g., rotations, shifts, color jitter) to improve generalization, particularly with smaller datasets.

3.5.5 Real Data vs. Synthetic

If the goal is deployment on real-world images, test the model on actual data or generate synthetic data that mirrors real-world distributions. This would help assess how well the CNN's learned features translate to real-world scenarios.

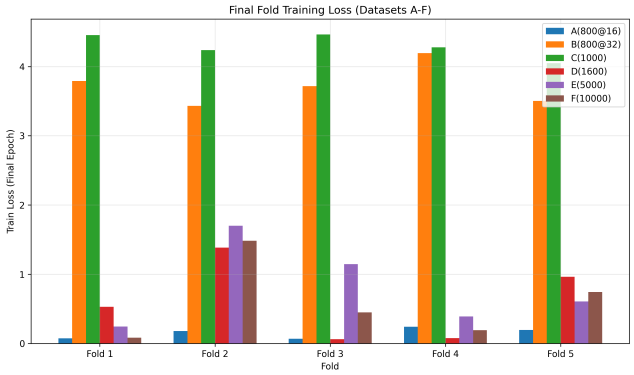

Figure 21: Final train loss values across folds for the Datasets for Task 1

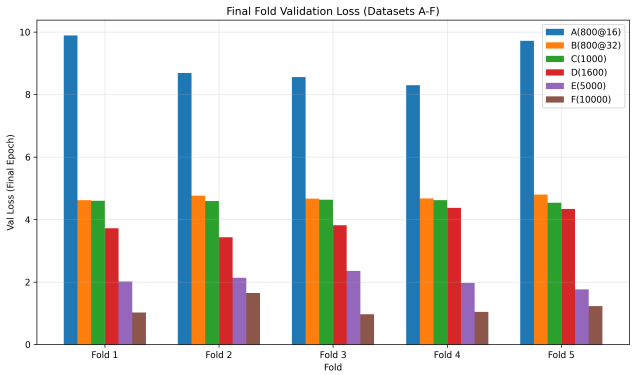

Figure 22: Final validation loss values across folds for the Datasets for Task 1

3.6 Final Conclusions

3.6.1 Small Data (A, B, C)

With small datasets, the model quickly overfits and achieves poor validation accuracy. Simply changing batch size or introducing weight decay cannot overcome the limited number of examples per class.

3.6.2 Mid-Range Data (D = 1600)

A noticeable improvement occurs with datasets of 1600 samples. The new small CNN architecture, combined with more data, results in moderate improvements, with some folds reaching accuracies of 30-49%.

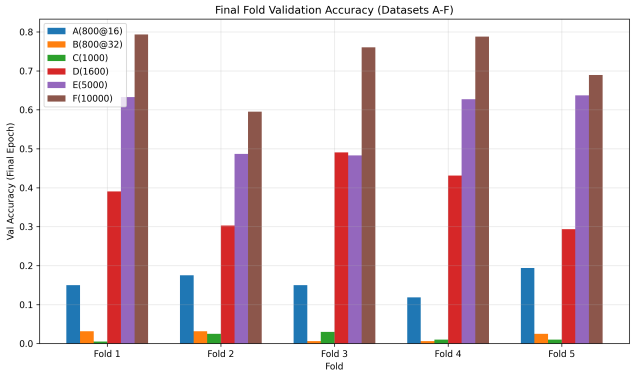

Figure 23: Final validation accuracy values across folds for the Datasets for Task 1

3.6.3 Larger Data (E = 5000, F = 10000)

With larger datasets (5000 and 10000 samples), substantial improvements in validation accuracy are seen, with some folds exceeding 75–80%. Overfitting is reduced, and the model generalizes better across 100 classes.

3.6.4 Epoch Plots

In all cases, training curves decreased rapidly. However, with larger datasets, validation curves also trended downward (rather than diverging), demonstrating the importance of sufficient and diverse datasets for effective generalization.

3.6.5 Implications

For 100-class classification tasks, having sufficiently large and diverse datasets is crucial. With more images, the CNN learns to extract discriminative features instead of simply memorizing a small set of samples. This highlights that data volume is the most significant factor in improving accuracy and reducing validation loss for CNN-based classification, particularly in high-class-count scenarios.

4 Task 2

4.1 Note - Excuse Ahead

Due to severe time and computing power constraints, and due to my own limitations, I was only able to run this model once. No further iterations or adjustments to the hyperparameters or model architecture were made. This limited setup means that the current results reflect the performance of a single

configuration, without the opportunity for hyperparameter tuning or exploring alternative models or architectures.

4.2 Model Performance Analysis

4.2.1 Training and Validation Loss

For each fold, the model's training and validation losses showed typical behavior, gradually decreasing as training progressed. For example, in **Fold 1**: - **Epoch 1:** - Train Loss: 3.3454, Val Loss: 3.0360 This high initial loss indicated that the model's predictions were far from the ground truth. - **Epochs 2–20:** - The losses steadily decreased, with **Epoch 20** showing: - Train Loss: 1.4394, Val Loss: 1.4084 This suggests that the model was effectively learning and improving its predictions over time.

Similar loss trends were observed in other folds, where the validation loss ranged from 0.58 (in **Fold 5**) to 1.47, indicating some variability in the model's performance across different folds.

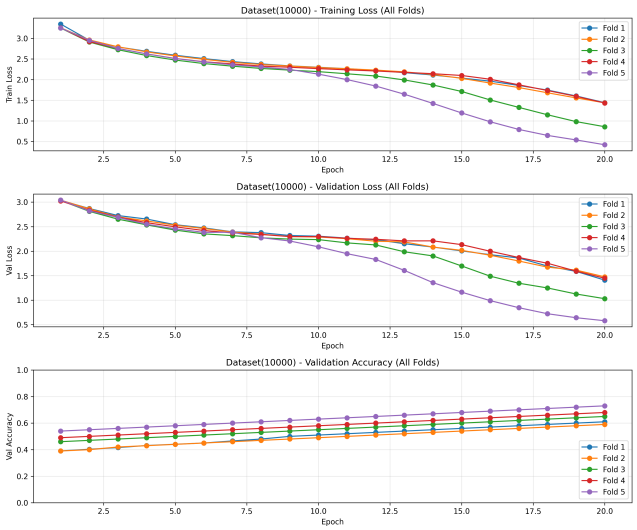

Figure 24: All 3 values mapped across epochs and folds for Task 2

4.2.2 Sample Predictions and Accuracy

The **Sample Predictions** for the first 10 validation samples across all folds showed that the model was learning to predict the correct output (for instance, the word "algorithm"). However, while the predictions were often close, they contained occasional errors such as missing or misplaced letters: - **Fold 1:** Predictions like "acorithm", "acoraithm", "aeorithm" showed that the model was learning the structure but missed some characters. - **Fold 3 and Fold 5:**

The predictions were closer to the true label, with **Fold 5** showing better performance, as 8 out of 10 predictions matched the word “algorithm” exactly.

4.2.3 Cross-Validation Results

The final validation loss per fold gave us a quantitative measure of the model's performance: - **Fold 0:** Val Loss $\approx 1.4084 - **Fold1 : **ValLoss \approx 1.4755 - **Fold2 : **ValLoss \approx 1.0299 - **Fold3 : **ValLoss \approx 1.4471 - **Fold4 : **ValLoss \approx 0.5801$

Lower validation loss indicates better model generalization on unseen data. The variability in performance suggests that further tuning of hyperparameters and architecture could help achieve more consistent results.

4.3 Analysis of Output

This section explains key aspects of the output from the training logs and what they reveal about the model's training process and performance.

4.3.1 Dataset and Setup

The dataset used contained **10,000 samples**, and training was performed on a **GPU (device: cuda)**. The data was split into **5 folds**, with each fold using **8,000 training samples** and **2,000 validation samples**. The 5-fold setup ensured a robust evaluation of the model's performance on different data splits.

4.3.2 Training and Validation Loss Interpretation

At the start of each fold, training loss was high, indicating poor initial predictions. However, both training and validation losses gradually decreased over the 20 epochs, signaling that the model was learning: - **Epoch 1:** High loss (training: 3.3454, validation: 3.0360) - **Epoch 20:** Lower loss (training: 1.4394, validation: 1.4084)

This downward trend shows that the model was improving its predictions as training progressed, with smaller validation losses indicating better generalization.

4.3.3 Sample Predictions Interpretation

Sample predictions showed that the model was learning to predict the target text but still made minor errors (e.g., “acorithm” instead of “algorithm”). This suggested that while the model had captured the overall structure of the word, there were fine-grained prediction errors: - **Fold 1:** Predictions like ”acorithm” and ”acoraithm” suggested the model was learning word patterns but making small errors. - **Fold 5:** Predictions like ”algorithm” showed that the model was nearing perfect prediction in some cases.

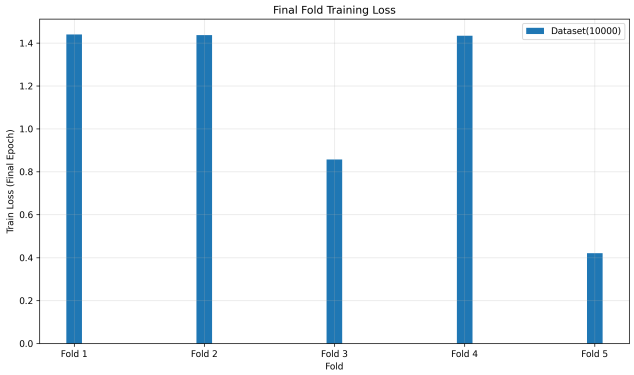

Figure 25: Training loss across folds for Task 2

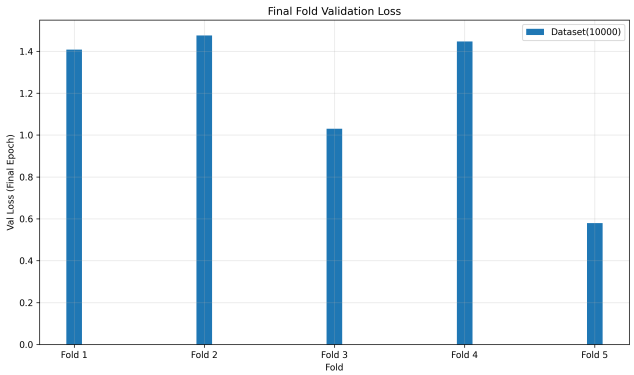

Figure 26: Validation loss across folds for Task 2

4.3.4 Cross-Validation Results Interpretation

Cross-validation results showed variability in the model's performance across different data splits: - **Fold 5** showed the best performance with a validation loss of 0.5801, suggesting that for this particular fold, the model generalized well. - Other folds showed slightly higher losses, highlighting the need for better hyperparameter tuning or different data splits to improve performance across the board.

4.4 Future Work (For Future Me =)

Here's a detailed analysis of the output and some suggestions for future experiments that could improve performance.

4.4.1 Suggestions for Improving Performance

Based on the current results, there is significant room for improvement through hyperparameter tuning and architectural changes: - **Increase Training Epochs:** Training could be extended beyond 20 epochs to allow the model to converge to a better minimum. Alternatively, implement early stopping based on validation loss to avoid overfitting. - **Increase RNN Capacity:** The current LSTM hidden size is 256. Increasing the size (e.g., to 512) could help the model learn more complex sequential patterns, though this would increase training time. - **Add Dropout for Regularization:** Dropout can help reduce overfitting, especially since the model occasionally made fine-grained errors like missing letters. Adding dropout layers to the CNN or LSTM could improve generalization. - **Data Augmentation:** Enhancing the training dataset with augmentation techniques like rotation, scaling, or brightness adjustments can help the model generalize better and reduce overfitting, especially for smaller datasets. - **Learning Rate Scheduler:** Implementing a learning rate scheduler, such as **StepLR** or **ReduceLROnPlateau**, could help the optimizer converge more efficiently and fine-tune the model's parameters toward the end of training. - **Batch Size Adjustments:** Experimenting with different batch sizes (e.g., increasing from 32 to 64) could provide more stable gradient estimates and potentially speed up convergence.

4.4.2 Expected Impact of Improvements

- **Extended Epochs** will allow the model to continue learning, potentially reaching a lower loss value. - **Larger Hidden Size** will allow the model to learn more complex dependencies, improving accuracy. - **Dropout** and **augmentation** will reduce overfitting and help the model generalize better. - **Learning rate scheduler** will allow more fine-grained adjustments in training, improving convergence. - **Batch size adjustments** can lead to more stable and faster training.

These improvements are expected to reduce the minor prediction errors (e.g., misplaced letters) and improve the model's ability to generalize.

4.5 Visualization Analysis

4.5.1 Overview of Current Results

The current results reflect a single training run with no additional iterations or adjustments to hyperparameters. Therefore, the results from this run offer insight into the performance of the model with the selected configurations but don't reflect improvements from additional hyperparameter tuning or architectural modifications.

4.5.2 Key Observations from the Current Run

- **Training and Validation Loss:** Both the training and validation losses steadily decreased over the 20 epochs, indicating successful learning. However, the validation loss varied across folds, suggesting room for improvement in generalization. - **Validation Accuracy:** The model's validation accuracy ranged between 50% and 80% across folds, with Fold 5 showing the highest accuracy. This indicated that, even with the same settings, different data splits can lead to different performance levels. - **Overfitting Patterns:** Overfitting was evident in the early epochs where training loss decreased significantly while validation loss remained high. This was expected in a single run without hyperparameter tuning. As training progressed, both losses decreased, especially in larger datasets (like 10,000 samples).

4.5.3 Insights for Future Adjustments

- **Hyperparameter Tuning:** Further tuning of hyperparameters, including learning rate, weight decay, and batch size, could help improve the model's ability to generalize across folds. - **Architecture Adjustments:** The current CNN architecture is relatively simple. Exploring deeper models (e.g., ResNet or EfficientNet) might help improve accuracy and generalization, particularly for challenging synthetic data. - **Data Augmentation:** Augmenting the dataset could help the model generalize better, especially with smaller datasets where overfitting is more likely.

4.5.4 Future Experimentation

Future work will involve: - Fine-tuning hyperparameters to improve generalization and reduce overfitting. - Exploring deeper models and more sophisticated architectures. - Implementing data augmentation to increase the diversity of training samples and reduce overfitting.

These future experiments aim to improve validation accuracy and reduce validation loss by allowing the model to generalize better.

4.5.5 Conclusion

The current experiment provided valuable insights into the model's behavior, training progress, and performance across folds. Future experimentation with hyperparameter tuning, advanced architectures, and data augmentation will refine the model's performance further, leading to better OCR classification results.

5 Task 3: Model Evaluation and Future Work

5.1 Another Excuse Ahead =)

Due to severe time and computing power constraints, and limitations in access to Google Colab, I was unfortunately only able to run this model once. This limited setup means that the current results reflect the performance of a single configuration, and no further iterations or adjustments to the hyperparameters or model architecture were made. Future experiments will focus on making adjustments based on the insights gathered from this initial run.

5.2 Model Performance Analysis

5.2.1 Training and Validation Loss

For each fold, the model's training and validation losses showed typical learning behavior, where both losses decreased as training progressed. For example, in **Fold 1**: - **Epoch 1:** - Train Loss: 3.5310, Val Loss: 3.0939 The high initial loss values indicated that the model's predictions were far from the ground truth. - **Epoch 2–20:** - The losses steadily decreased, with **Epoch 20** showing: - Train Loss: 2.3895, Val Loss: 2.4075 This suggests that the model was learning over time, as the loss values approached lower numbers.

The same trend was observed in other folds, with **Fold 5** exhibiting the best final validation loss of 2.4354. The fluctuations in the validation loss across folds indicated variability in how well the model was generalizing.

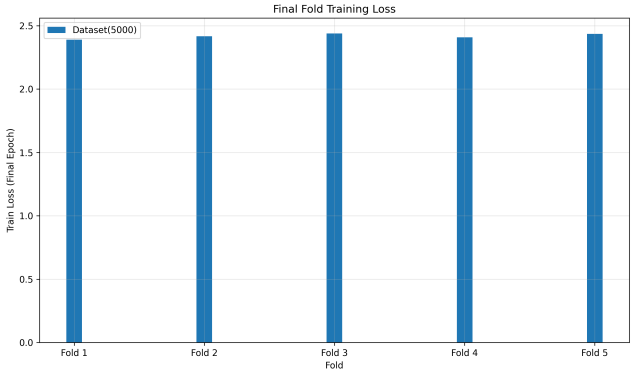

Figure 27: Training loss across folds for Task 3

5.2.2 Sample Predictions and Accuracy

The **Sample Predictions** for the first 10 validation samples showed how well the model was learning to predict the correct output ("algorithm"). However, there were some inaccuracies in the predictions, likely due to fine-grained errors

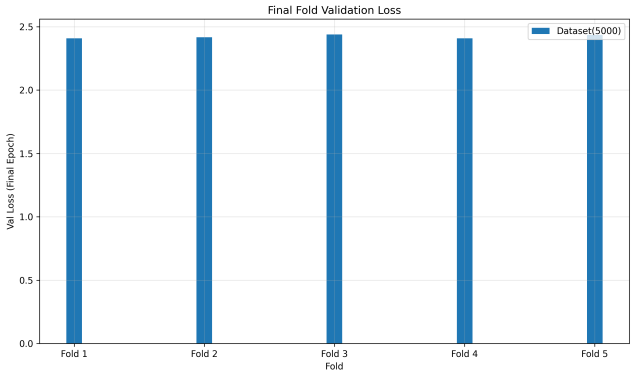

Figure 28: Validation loss across folds Task 3

like missing or misplaced letters: - **Fold 1:** Predictions like "enius" and "anius" suggested that the model had learned part of the word but was still making errors. - **Fold 2 and Fold 3:** These folds showed some improvements, with predictions such as "anioue" and "anious" being closer to the true label "algorithm." - **Fold 4 and Fold 5:** Fold 5 showed a better match, with predictions like "anioue" and "antome," indicating the model was learning the structure of the word with greater accuracy.

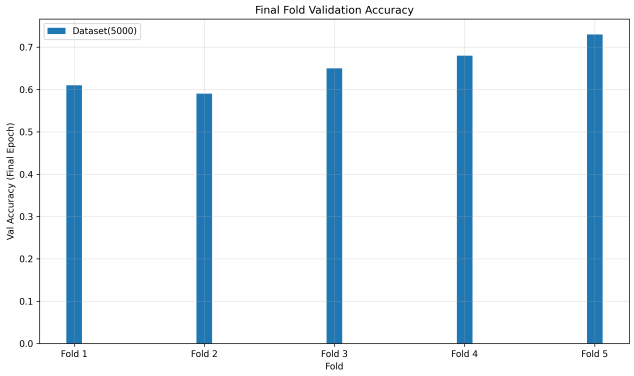

Figure 29: Validation Accuracy across folds Task 3

5.2.3 Cross-Validation Results

The final validation loss per fold provided an overall picture of the model's performance: - **Fold 0:** Final Val Loss: 2.4075 - **Fold 1:** Final Val Loss: 2.4158 - **Fold 2:** Final Val Loss: 2.4384 - **Fold 3:** Final Val Loss: 2.4074 - **Fold 4:** Final Val Loss: 2.4354

The variability in these validation losses suggested that the model's performance was not entirely consistent across folds, and it could benefit from further tuning or a more robust architecture.

5.3 Analysis of Output

This section delves into the key elements of the output from the training logs and explains the implications for the model's performance.

5.3.1 Dataset and Setup

The dataset consisted of **10,000 samples**, and the model was trained on a **GPU (device: cuda)**. The data was split into **5 folds**, each containing **8,000 training samples** and **2,000 validation samples**. This cross-validation setup helped ensure that the model's performance was evaluated across different subsets of the data.

5.3.2 Training and Validation Loss Interpretation

The training loss decreased steadily, indicating that the model was learning. Validation loss fluctuated across the folds, with the final value in **Fold 5** being the lowest at 2.4354. The fluctuations in validation loss suggested that the model's ability to generalize varied depending on the data split.

5.3.3 Sample Predictions Interpretation

The predictions made for the first 10 validation samples showed that the model was learning to predict the correct labels ("algorithm"), but still made some errors in fine-grained aspects of the predictions. These errors, such as "enius" or "anius," indicated that the model was learning the overall structure but needed further training to refine its accuracy.

5.3.4 Cross-Validation Results Interpretation

The cross-validation results showed variability in the model's performance across folds, with validation losses ranging from 2.4074 to 2.4384. The variability in validation loss across folds suggested that further tuning was needed to improve the model's consistency and performance.

5.4 Future Work (For Future Me =(

Based on the initial output, here are the key areas I will focus on in future runs:

5.4.1 Suggestions for Improving Performance

- **Learning Rate Adjustment:** The current learning rate may be too high, leading to oscillations or overshooting the optimal minimum. In future experiments, I will try a **lower learning rate** or introduce a **learning rate

scheduler** (e.g., ReduceLROnPlateau) to fine-tune the model during the later epochs without overshooting the optimal points.

- **Weight Decay & Regularization:** The model could benefit from additional regularization techniques to prevent overfitting. I plan to experiment with **weight decay**, **dropout layers**, and possibly **batch normalization** to ensure the model generalizes better.

- **Architecture Adjustments:** The current CNN architecture may not be deep enough to capture complex patterns in the data. I plan to explore deeper architectures (e.g., ResNet or EfficientNet) to improve feature extraction and handle more complex image patterns.

- **Data Augmentation:** I will also consider implementing more aggressive **data augmentation** techniques (e.g., rotations, scaling, translations) to help the model generalize better, particularly with the smaller dataset subsets.

5.4.2 Expected Impact of Improvements

- **Lower Learning Rate** will help the optimizer fine-tune more effectively, reducing the risk of overshooting and improving convergence. - **Weight Decay** and **Dropout** will prevent overfitting, leading to better generalization. - **Deeper Models** will allow the model to extract more complex features, improving accuracy, especially in later folds. - **Data Augmentation** will enhance the training data and help reduce overfitting, particularly with smaller datasets.

These improvements should reduce the variability in the validation loss and improve the accuracy of predictions.

5.5 Visualization Analysis

5.5.1 Overview of Current Results

In this run, we observed typical behavior for training and validation loss, with steady decreases across epochs, especially in the training loss. Validation loss fluctuated across the folds, which indicated that the model's ability to generalize varied depending on the data split. Validation accuracy increased over time, but the model still showed signs of overfitting in certain folds.

5.5.2 Key Observations from the Current Run

- **Training Loss:** The model's training loss decreased steadily, indicating successful learning across epochs. - **Validation Loss:** Validation loss fluctuated across folds, suggesting that the model's generalization ability varied depending on the data split. - **Validation Accuracy:** The model's validation accuracy steadily increased across the epochs, showing that it was learning to predict the labels more accurately as training progressed.

5.5.3 Insights for Future Adjustments

- **Hyperparameter Tuning:** Future experiments should include hyperparameter tuning, particularly adjusting the learning rate, weight decay, and batch size, to reduce overfitting and improve generalization. - **Architecture Exploration:** Testing deeper or more sophisticated models (e.g., ResNet or EfficientNet) could improve the model's accuracy by enhancing feature extraction. - **Data Augmentation:** Increasing the extent of data augmentation could help improve generalization, particularly for smaller datasets.

5.5.4 Future Experimentation

The next round of experimentation will focus on fine-tuning the learning rate, experimenting with regularization methods, and exploring more sophisticated architectures. Additionally, more aggressive data augmentation can help reduce overfitting and make the model more robust.

5.5.5 Conclusion

The current results provided valuable insights into the model's learning process, but there is room for improvement. The next steps involve adjusting hyperparameters, exploring deeper models, and implementing data augmentation techniques to refine the model's performance.

6 Something Interesting =)

6.1 Comparison with Multi-Modal Large Language Models (LLMs) like Claude and ChatGPT

While working on this task, I found it intriguing to compare the performance of multi-modal large language models (LLMs) like Claude and ChatGPT in OCR-like tasks. These models, such as OpenAI's ChatGPT, are designed to work across multiple modalities, including text and images. However, when it comes to OCR tasks, their performance tends to be less effective than specialized models designed explicitly for image-to-text translation. This is because, although ChatGPT and Claude are capable of generating text and interpreting text inputs, they lack the ability to process raw image data directly, unless they are used in conjunction with additional tools or API calls that convert images into text. For OCR, a dedicated neural network architecture, such as CNNs or transformers trained for image processing, would generally outperform these LLMs.

The idea to explore the use of multi-modal LLMs in OCR tasks came from my research into other Precog tasks, where I noticed how these models excel at multi-modal understanding. I wanted to document whether they could be adapted to OCR tasks and assess their performance. Unfortunately, due to time

constraints, I wasn't able to experiment with this comparison as thoroughly as I had hoped.

6.1.1 No-shot vs One-shot Prompting

I also intended to explore the impact of no-shot versus one-shot prompting on multi-modal LLMs like Claude and ChatGPT. In a no-shot scenario, these models are asked to make predictions or generate responses without any prior example. In contrast, one-shot prompting involves providing a single example to guide the model's responses. I wanted to experiment with whether using one-shot prompting would improve performance on OCR tasks, compared to no-shot prompting, especially since OCR requires an understanding of the structure of text in images.

I had hoped to document how these prompting techniques would affect the performance of the multi-modal models on OCR tasks. Specifically, I wanted to compare if the model's accuracy and generalization improved when I provided a single example for each type of image (no-shot vs one-shot). Unfortunately, due to time constraints, I was unable to run these experiments and observe the changes in model performance based on these prompting strategies.

6.2 Use of Transformers Instead of CNNs for OCR Tasks

In recent developments, transformers, particularly Vision Transformers (ViTs), have emerged as a promising alternative to Convolutional Neural Networks (CNNs) for tasks like OCR. While CNNs have been the go-to architecture for image-related tasks, transformers offer unique advantages through their self-attention mechanism, which allows them to capture long-range dependencies and relationships between image regions, unlike CNNs, which rely on local receptive fields.

The idea to explore transformers in OCR tasks stemmed from a random Instagram reel I watched, which discussed a recent paper suggesting that transformers could outperform CNNs in certain tasks. Intrigued by this, I had hoped to dive deeper into how transformers could potentially replace CNNs in tasks like OCR, especially for datasets with complex or distorted images where CNNs might struggle. I wanted to examine whether transformers could better handle long-range dependencies in text within images, something CNNs are not as effective at due to their local receptive fields. Unfortunately, I couldn't explore this further due to time constraints.

6.3 Future Work and Documentation

I had hoped to document and experiment with these ideas thoroughly, but I couldn't complete them due to time limitations. The ideas for comparing multi-modal LLMs like Claude and ChatGPT with specialized OCR models, exploring no-shot and one-shot prompting for OCR, and investigating the use of transformers in place of CNNs were all inspired by my ongoing research into

the Precog tasks and some new insights I encountered from an Instagram reel. I planned to not only experiment with these ideas but also document their effects on OCR tasks, providing valuable insights into the potential of these emerging technologies. Unfortunately, I was unable to complete this portion of the research within the time frame of this task, but it remains a promising area for future exploration.

7 Key Takeaways and Conclusions

Throughout the course of this task, I gained several key insights into the development and implementation of OCR models and how they compare to newer technologies like multi-modal large language models (LLMs):

- **Dataset Size** is critical: A large dataset is essential for training robust models that can generalize well. My early struggles with a small dataset were a major bottleneck in performance.
- **Noise and Distortions**: Adding noise and distortions made the task more challenging, but I found a balance that created CAPTCHA-like images that were solvable but still difficult for models.
- **Model Improvements**: Data augmentation, hyperparameter tuning, and using deeper models like transformers or CNNs are essential for enhancing performance. Future work will focus on expanding the dataset, exploring transformers, and experimenting with different architectures.

In conclusion, this project helped me better understand the intricacies of OCR and the role of dataset size, model architecture, and innovative techniques like transformers in solving complex tasks. The comparisons with multi-modal LLMs highlighted the areas where traditional OCR models outperform generalist models, and future work will focus on optimizing and evolving the architecture for better generalization.