Final

1.1 Literature Review

CAPTCHA is widely used in today's world to determine whether a user that is accessing a website is a human or a bot. It is based on the Turing Test that is made of hard AI that can be passed by humans but not by computers [1]. This test usually comes in the form of distorted letters and numbers that are easily read by humans but pose a challenge for computers or softwares to parse due to limitations in visualization of the image. Due to recent evolutions in the field of Computer Vision, there can be concerns of the effectiveness of CAPTCHA as a security test [2]. A study shows that when tested with the 800 artificial challenges, the overall success rate of its attacking algorithm is 18.0% which is significantly higher than the estimate of 0.0006% [3].

1.2 Dataset Description and Link

Our dataset has more than 113,000 5-character images that are both colorful and black and white. They intend to have distorted letters and numbers in different fonts and colors, with the introduction of noise, making it hard for a computer to parse. Some features include: the number of characters, the set of characters, image size, and the labels associated with each image.

Problem Definition

2.1 Problem

We are focusing on extracting words from 5 letter text-based CAPTCHA containing alphanumeric characters with distortion and noise. The primary steps to solve this problem is preprocessing, segmentation, and recognition of characters.

2.2 Motivation

Firstly, training a word recognizer from distorted images allows for creating more robust CV models. Secondly, automating CAPTCHA can enhance accessibility for users with visual impairments. Finally, by our attempts to break CAPTCHA, people can further develop robust alternatives staying ahead of malicious actors.

Methods

3.1 Data Preprocessing

In order to prepare our dataset of over 100,000 CAPTCHA images for the CNN model that we implemented (will go over this later), we processed and prepared the images for training in a model in order to improve the accuracy, efficiency, and speed up the training and improve the generalization. The main goal for us was to make sure the data was optimized for the CNN model and ensuring that the input features were consistent. The specific preprocessing methods that we implemented are:

- **Universal Encoding of Labels**
  - Removed the extension of .jpg from each image.
  - Took the strings and one-hot encoded them into 5x62 tensors with a 1 in the character position that represents the character.
  - This helps in easy calculations for the CNN model because it is easier to work with integers than it is with strings.
- **Resizing**
  - Ensured that all images in the dataset have the same size of 40 x 150 pixels so that they are all uniform before the model starts its training. This helps in generalization because the dataset will be divided into training and testing and therefore ensuring that all images are uniform becomes important for generalization and reducing overfitting. This also helps in keeping the aspect ratio and content the same across all images.
  - How we did it: `transforms.Resize((40, 150))`
- **Converting to a Tensor**
  - Converted it from a PIL image format to a PyTorch tensor. This becomes important for further processing because tensors are the primary data structure that allows images to be processed in batches and helps in gradient based learning. It also becomes easier for the model to do calculations and train on tensors.
  - How we did it: `transforms.ToTensor()`
- **Normalization:**

    - We did this to center the data around zero with unit variance. This could be done by subtracting the mean and dividing by the standard deviation. This is helpful because it allows for more stable gradient descent by reducing the variance and balancing pixel values. The model can more easily adapt itself to the characteristics of the images
    - How we did it: `transforms.Normalize(mean=mean, std=std)`
- **Grayscale Conversion:**
    - This is important to reduce the number of dimensions of the image from 3 (RGB channels) to just 1. This helps in speeding up the process of training by the model because it now only has to compute on one dimension rather than 3. It also helps in reducing any noise generated by multiple colors that could cause confusion for the model
    - How we did it: `transforms.Grayscale(num\_output\_channels=1)`

For more detailed implementation of the preprocessing methods:

3.2 ML Algorithms/Models

3.2.1 Convolutional Training Model

For the midterm, we implemented the Convolutional Training Model (CNN) for our problem statement. This is a supervised learning algorithm that we thought would be very useful and efficient for our problem and the images in our dataset.The reason why we chose this model is because we are essentially dealing with a computer vision problem where we want to extract the 5 letter CAPTCHA string that comes with different kinds of noise and distraction.We thought that the CNN is an appropriate model for the CAPTCHA dataset because of the following features:

- **Good at Local Feature Extraction:** The main goal is to extract those 5 alphanumeric characters that can be overlapping or curved. CNN has convolutional filters that allows it to recognize edges and corners. It can focus on gradient changes between the background and letters using unique filters
- **Spatial Invariance:** Images in the CAPTCHA dataset as can be seen in the link that we provided earlier, have random shifts and rotations in different directions which can confuse a model. However, as we learned in class, CNN's are robust to these transformations
- **Parallel Computation:** It can output the predictions associated with each character in the string simultaneously, thereby increasing the speed of training and testing while also

reducing memory usage.

- **Classification Problem:** It is very useful to create a confusion matrix that maps each image in the dataset to a finite set of characters, in this case, 62 characters (A-Za-z0-9). More of this implementation and the confusion matrix is mentioned below. \

Overview of CNN Implementation

After preprocessing the entire dataset, we divided it into a training and testing dataset with a 80-20 ratio. The architecture of the model is such that it outputs 5 different character predictions based on each character. There are 62 classes per character that helps in generating a confusion matrix grid that is 62 x 62. This matrix can be seen in the results section of the report.For training, on each epoch, the model then calculates the character and sequence accuracy. We used CrossEntropyLoss for the classification and Adam optimizer for gradients and hyperparameters.The confusion matrix is used to evaluate the character prediction accuracy on the testing set. We make the matrix by first collecting the true and predicted labels across the testing dataset and having the ground truth as the rows and the predicted labels as the columns. We used seaborn to visualize the confusion matrix. This matrix helps in:

- Analyzing the misclassifications for each character class
- Counting the correct predictions for individual characters (character level accuracy)
- Counting the correct predictions across all the 5 characters in the CAPTCHA image to see if it was correctly recognized (sequence level accuracy). This is the main goal of the project.

For more detailed view of the CNN implementation

3.2.2 K-Means

For the final, we implemented K-Means for our problem statement. This is an unsupervised learning algorithm that we thought would be very useful and efficient for our problem and the images in our dataset.The reason why we chose this model is because K-Means is very useful for data clustering, or grouping unlabeled points together into clusters. Our CAPTCHA dataset, that has over 100,000 images has a lot of diverse patterns and requires segmentation of data based on the 5 characters that are in each image. We thought that K-Means was an efficient model for our dataset because of the following features:

- **Clustering of Data:** It can create meaningful clusters based on similarity of text (in this case). It can help in segregation of the characters from the background. This becomes useful because our CAPTCHA dataset contains a lot of noise, background distortion in the form of lines and overlapping letters.

- **Unsupervised Nature of the Model:** It does not require labels, which was the case for CNN. This means that it does not need perfect preprocessing to get the most accurate results. This makes the preprocessing step simpler.
- **Feature Extraction:** This is one of the most important uses of K-Means for our dataset because it is able to identify the characters from the image which can potentially be used by future downstreamed supervised learning algorithms to get more accuracy and increase the efficiency of the results.
- **Dimensionality Reduction:** Since the results are groups of clusters that are most closely related to the input image (in this case, 5 clusters), it helps in reducing the complexity of the dataset. This, as a result, can help in making inferences faster and more efficiently because the dimension has been reduced to 5 clusters while keeping the core structure of the dataset the same.

Overview of K-Means Implementation

The first step done before implementing the K-Means algorithm was flattening the images in the dataset into vectors of length 6000. Each vector represents a single image in the dataset. The dataset can be accessed through the code (refer to the link at the end of this section).After flattening the images, clustering is performed on those flattened images using Scikit-learn's K-Means function. This will specify 62 clusters because of the 62 characters that can be present in the images (A-z a-z 0-9). Therefore, the number of clusters were chosen to align with the number of classes.

There were two approaches taken for this algorithm:

- Labeling each cluster based on the majority class of its members
- Pairing each class with its largest clusterThe reason for the second approach was to ensure representation of all the classes in the cluster. This is because the first approach resulted in several clusters sharing the same label. This left some of the classes unrepresented.For inference, the input images are flattened, and the five cluster centers closest to the input vector in the feature space are identified. The labels of these cluster centers are predicted to be present in the image. This approach does not aim to determine the exact permutation of the CAPTCHA sequence.

For more detailed view of the K-Means implementation

3.2.3 Transformer-based Model

Another method we explored was a Transformer-based model. Transformers are generally great in sequence based tasks. For this problem, we are given an input image, and we need to output a fixed length sequence. CNN processes the image character by character, whereas

Transformer runs inference on the whole image at once. However, in our case, the Transformer model did not perform as well as the CNN model. Despite our expectations, the CNN-only model outperformed the Transformer model in terms of both accuracy and generalization.

Some of the other reasons for choosing a Transformer are -

- **Attention Mechanism** - The self-attention mechanism could allow the model to focus on different parts of the image when predicting each character, potentially handling overlapping or distorted characters better.
- **Robustness to Noise and Distortions** - The dynamic weighting of features via attention could help the model to be more robust to the various types of noise and distortions.

Overview of Transfer Learning Implementation

The images were converted to grayscale, resized to a consistent size, and normalized to match the input size expectations of the model. The model architecture consists of an Encoder and a Decoder.

- **CNN Encoder** - Similar to the CNN model, we used a convolutional neural network to extract features from the input images. This encoder produced a sequence of feature vectors intended for the decoder.
- **Attention Mechanism** - Implemented an attention mechanism to focus on relevant parts of the encoded features during decoding.
- **GRU Decoder** - Instead of a Transformer decoder, we used a Gated Recurrent Unit (GRU) to generate the output character sequence.

The training process uses the Loss Function - CrossEntropyLoss for multi-class classification at each character position. We used the Adam optimizer with an initial learning rate of 0.001. During inference, beam search decoding was employed to improve the quality of the predicted sequences.

Despite the theoretical advantages of the Transformer model, it did not perform as well as our initial CNN model. The CNN model achieved higher accuracy in both character-level and sequence-level predictions. The Transformer model struggled to generalize and often produced incorrect sequences during testing.

Some possible reasons it failed to give accurate results -

- **Insufficient Data for Transformers** - Transformers generally require large amounts of data to train effectively due to their complexity and large number of parameters. Our dataset may not have been large enough to fully leverage the capabilities of the Transformer model.

- **Model Complexity** - The added complexity of the Transformer model might have been unnecessary for the CAPTCHA recognition task. The CNN model was possibly sufficient to capture the necessary features without the need for an advanced decoder.
- **Sequence Length** - CAPTCHA sequences are relatively short (typically 5 characters). The Transformer's strength in handling long-range dependencies might not have provided a significant advantage over simpler models in this context.

Any of these reasons could have caused the inaccurate model, causing frequent misclassifications with certain characters with similar visual features (e.g., 'O' and '0', 'I' and 'l').

For more detailed view of the Transformer-Based Learning implementation

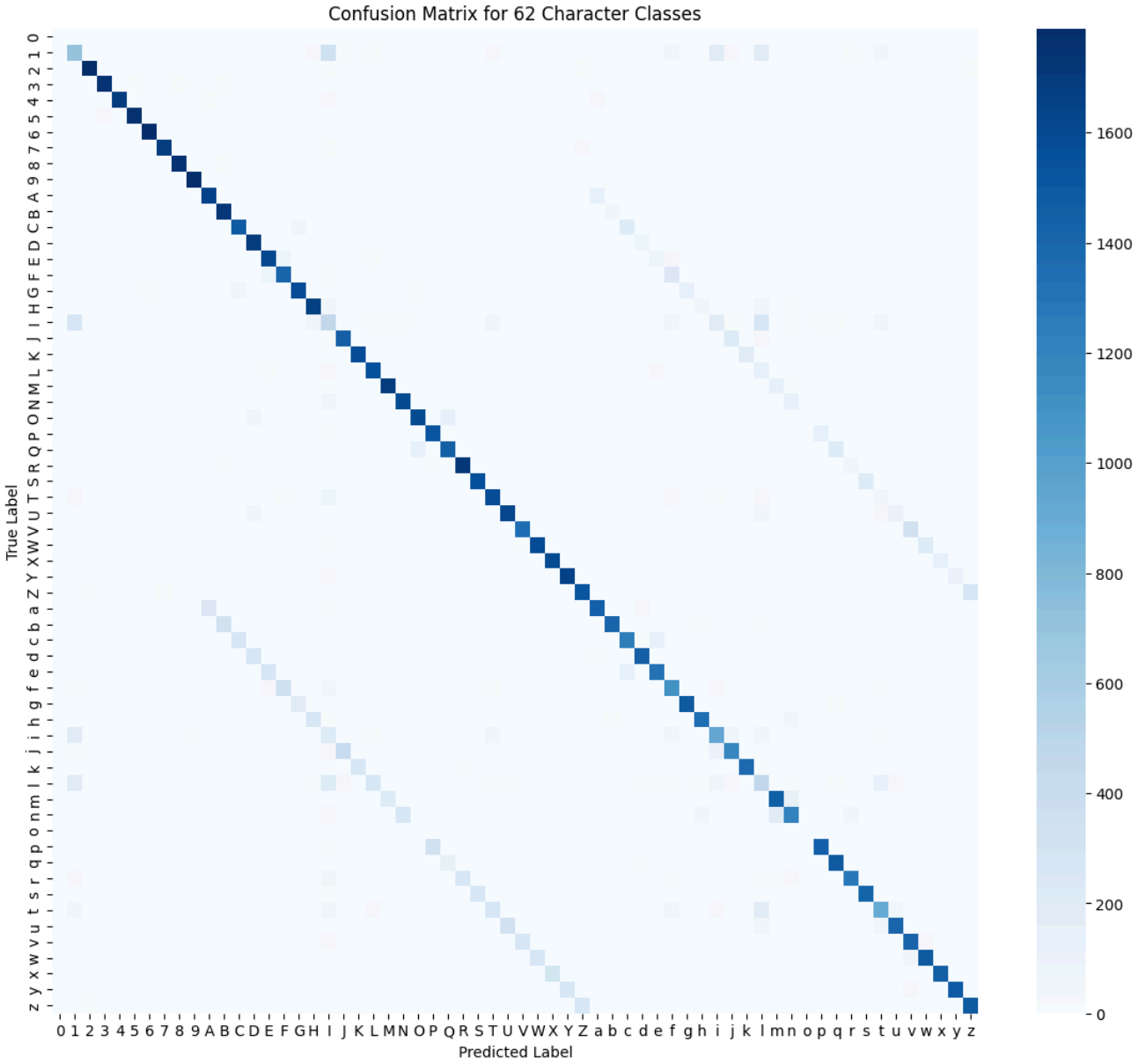
Results and Discussion

4.1 Convolutional Training Model

4.1.1 Numerical Analysis and Quantitative Metrics

- **Character Level Accuracy** The character level accuracy in the training dataset came out to be 70.39 % and in the test dataset 77.15 %.
- **Sequence Level Accuracy** The Sequence Level Accuracy in the training dataset came out to be 20.78 % and in the test dataset 35.62%. This jump in accuracy felt abnormal and is attributed to the dropout regularization layers which are deactivated during inference. The dropout layer deactivates some neurons in the network, reducing the complexity of the model. Running the test dataset with the model in model.train() mode while encompassing the entire method in torch.no_grad()so that the model's weight does not change, we obtain sequence level accuracy of 10.90 %. We present all evaluation result with model being in model.eval()as that is the true inference model.
- **Character Level Precision** We calculate weighted precision across the 62 classes, and get 77.18% for the evaluation set.
- **Character Level Recall** The weighted recall across the 62 classes is 77.16% for the evaluation set. The low recall can also be attributed to the class imbalance.

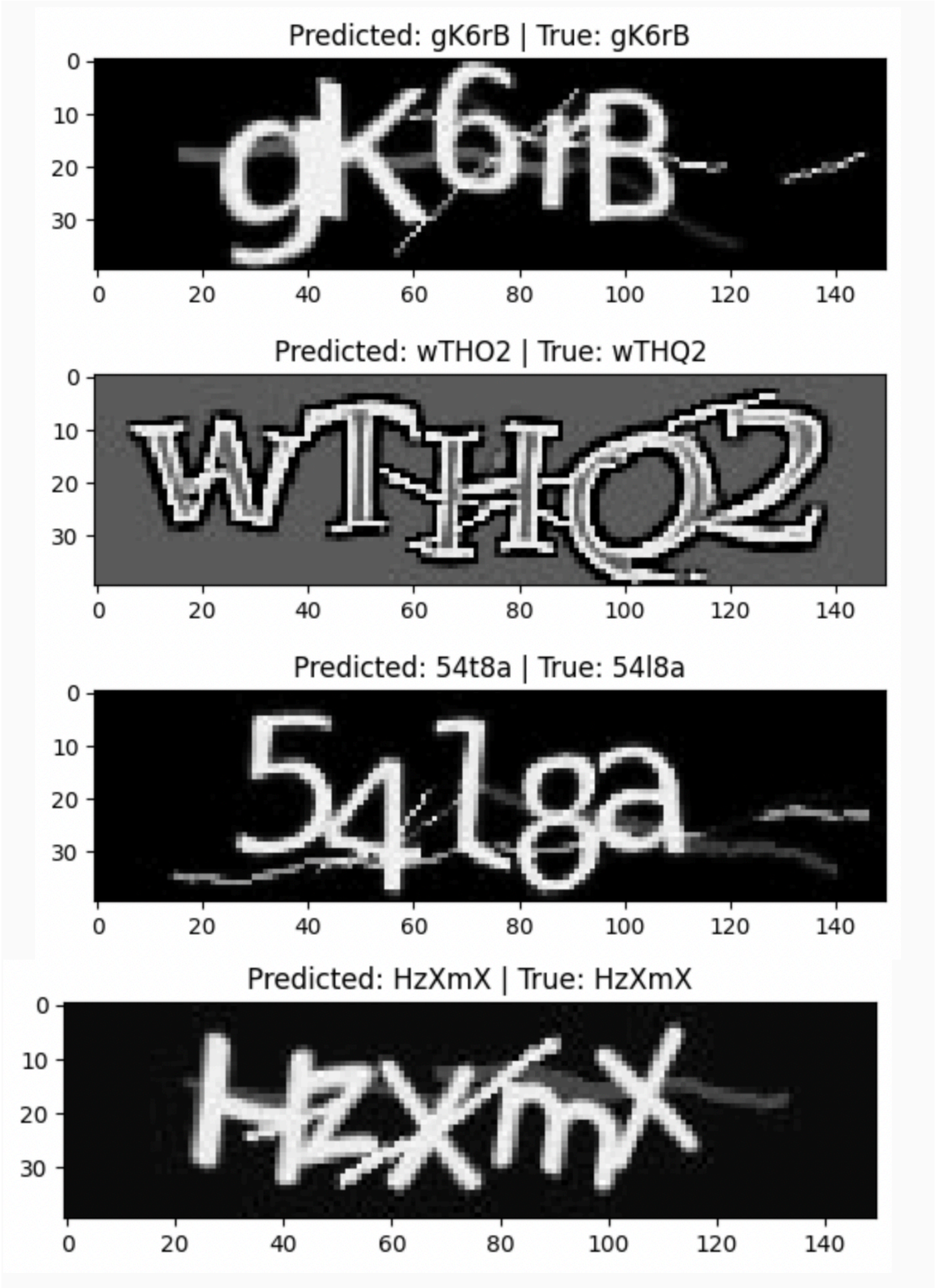
- **Confusion Matrix**



The main diagonal being strong represents the high precision and accuracy of the model. The two diagonal patterns on either side of the main diagonal map the small letters to capital letters representing that in the cases where the model incorrectly predicts a letter it mostly predicts the incorrect case of the same letter which is a very interesting finding.We do not calculate sequence level precision and accuracy because of the high number of classes and low data points.

4.1.2 Visualization

Following are some images with their true and predicted labels shown in the image. Notice that the image shown is after the preprocessing steps on a RGB image.



The two places where the model predicts incorrectly is O <-> Q and t <-> l. Not only are these

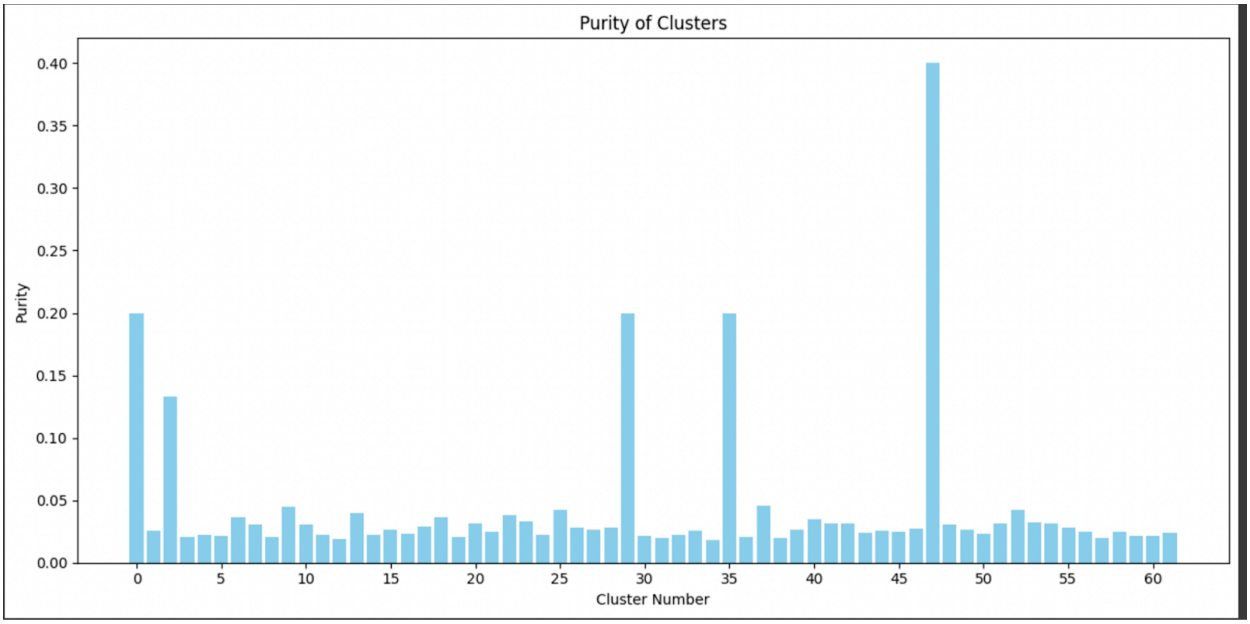
letters visually similar, but the confusion matrix also supports these classes and are often predicted as the other class!

4.2 K-Means

The complete K-Means evaluation can be found here

4.2.1 Numerical Analysis and Quantitative Metrics

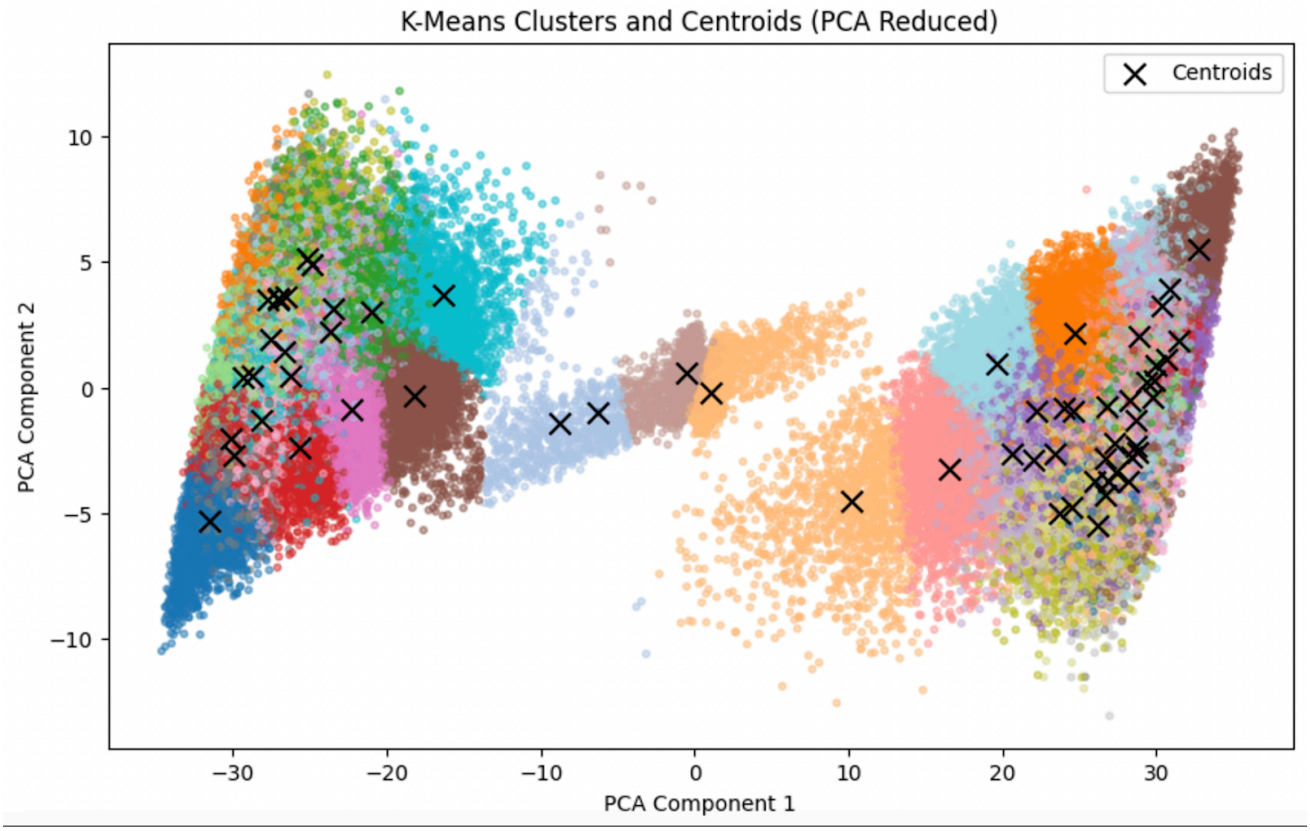
- **Silhouette Coefficient:**
  - From doing the numerical analysis on the trained dataset, we got that silhouette coefficient to be **-0.0657**.
  - Since this value is very close to -1, we can conclude that the clusters outputted from the K-Means clustering algorithm is not distinct or well-defined enough to the point that is was correctly able to place a character in its respective cluster (out of the 62 characters)
- **Calinski-Harabasz Index:**
  - After the evaluation, we got the value as **4320.3340** which is high enough to tell us that there were some clustering done, however from the other numerical analysis done, we know that the accuracy is quite low and therefore, even though there were clusters formed, it might not have led to the most accurate assignment of classes to the clusters.
  - Relatively, for a dataset of ~130,000 images, the index value should be higher, so the clustering algorithm could be made more efficient with better preprocessing.
- **Purity Score**
  - Using the visualization of a bar graph, we plotted the purity score of each cluster below



- We know that the purity score ranges from 0 to 1 with 1 indicating that the cluster has all the correct labels assigned to it. However, from this graph, we can see that each cluster has a purity score below a 0.4 with many ranging between 0.01 to 0.05. This shows that the clusters are definitely assigned the correct labels showing poor clustering on the dataset.

4.2.2 Visualization

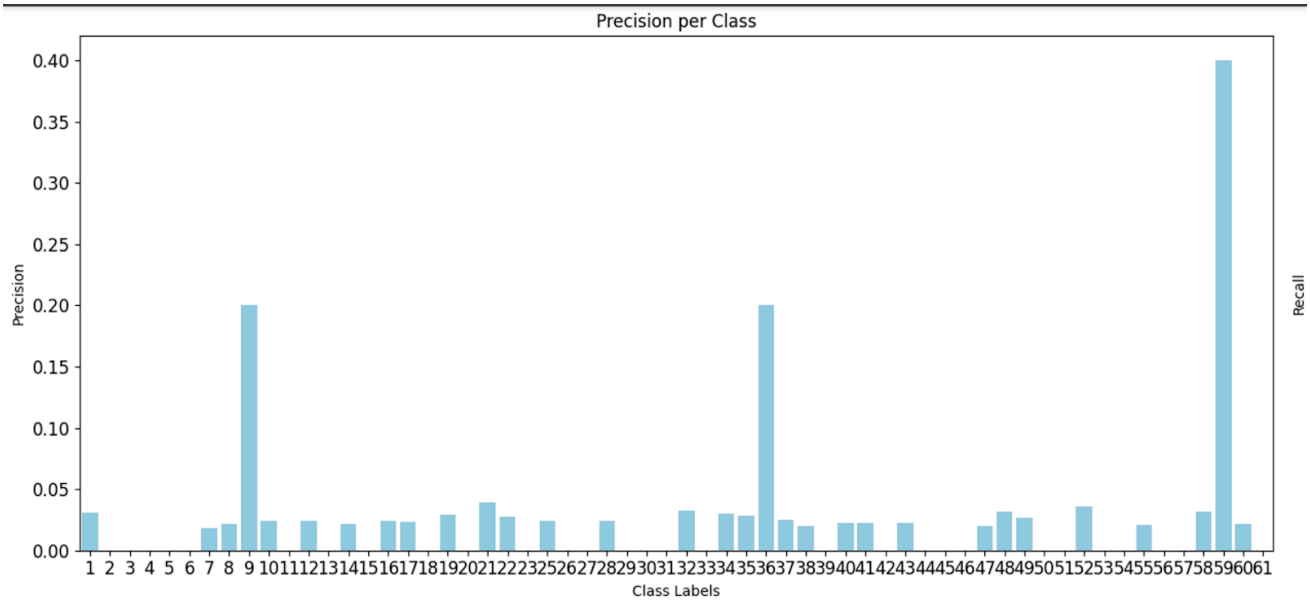
- **K-Means Clusters and Centroids**



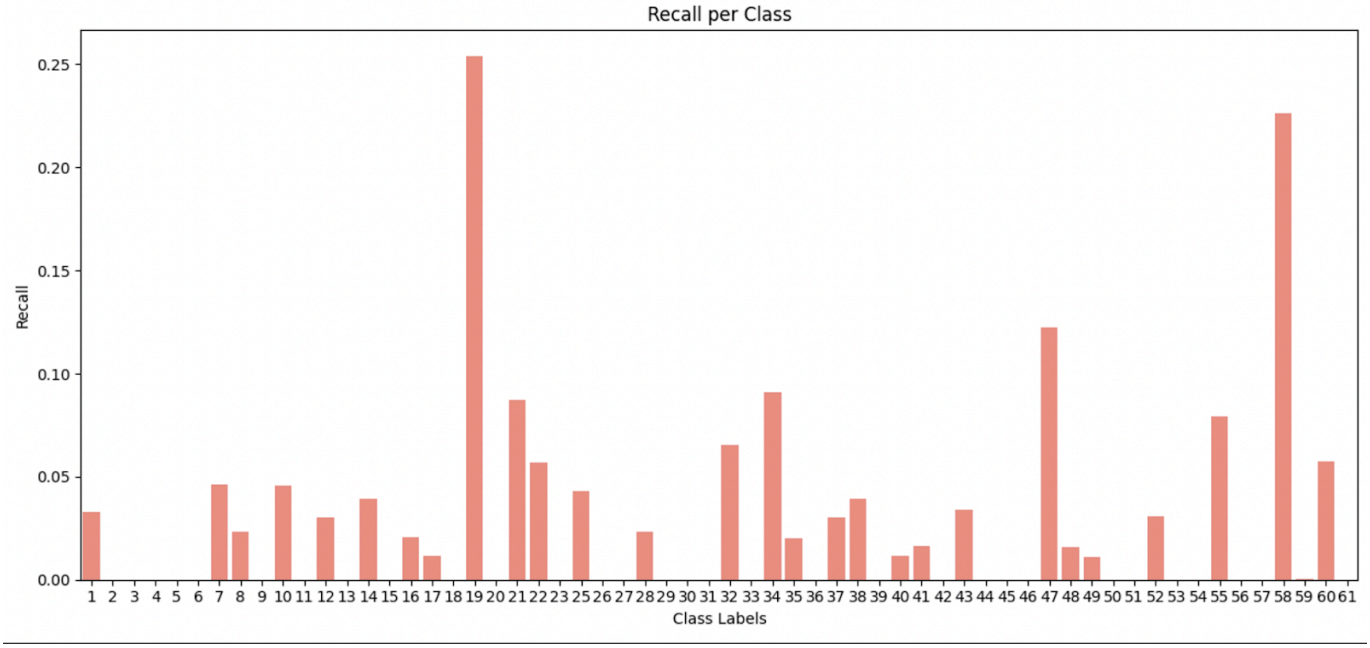
We first used PCA to reduce the dimensions to make the computation simpler. From this

plot, as seen above, we can conclude a few things. The K-Means clustering algorithm definitely outputs distinct 62 clusters based on the 62 different classes possible in the CAPTCHA image dataset. Additionally, we also see the centroids almost at the center of each cluster which shows that K-Means converged to a decently reasonable solution. However, if we look closer, then we can see that the clusters overlap quite a lot. This goes in hand with the other analysis we have done so far, indicating that there are misclassifications in terms of assigning the data points to the respective clusters. The clusters are also not spherical in shape which may be because the images in the dataset are too complex for the K-Means model to make predictions on. There are random noises in terms of lines in the background. These lines overlap with the characters which could have confused the model, thereby making incorrect predictions.

- **Classification Report**

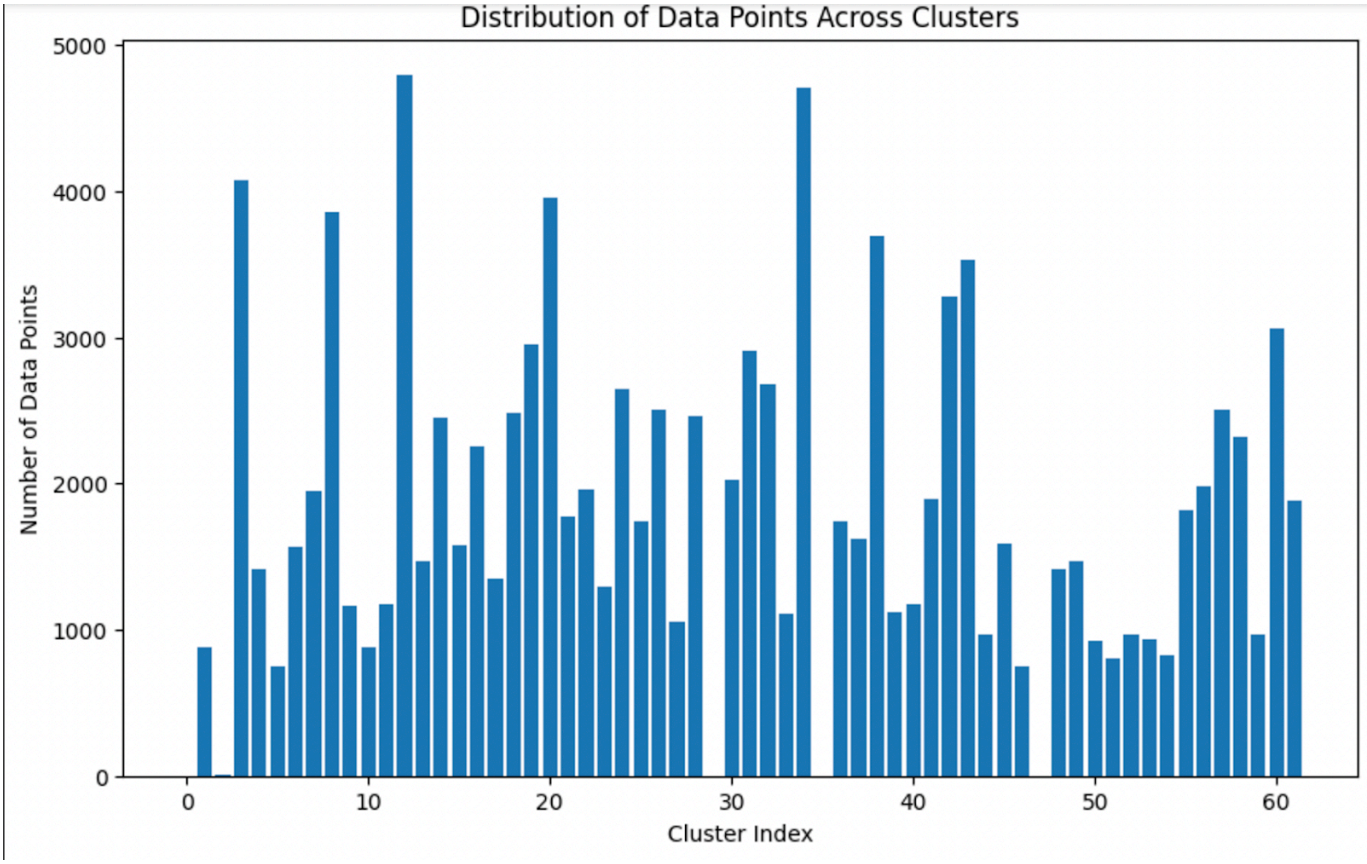


From the precision plot above, we can see that the precision from each of the 62 clusters is quite low except 3 of them. However, we know that having precision less than 0.5 is not a good indicator of confidence. All the clusters have a precision < 0.5. Therefore, while K-Means did effectively create clusters, it did not do so great in making sure that each character was matched with the right cluster number.



A lot of the recall value (from the plot above) from most of the class labels show that the value is closer to 0. This shows that the correct characters being assigned to the respective cluster is being missed by the model and it leads to a high false positive rate. If we compare this with the precision graph, we can see that it performs a lot worse showing the tradeoff between precision vs recall.

- **Data Point Distribution Amongst the Clusters**



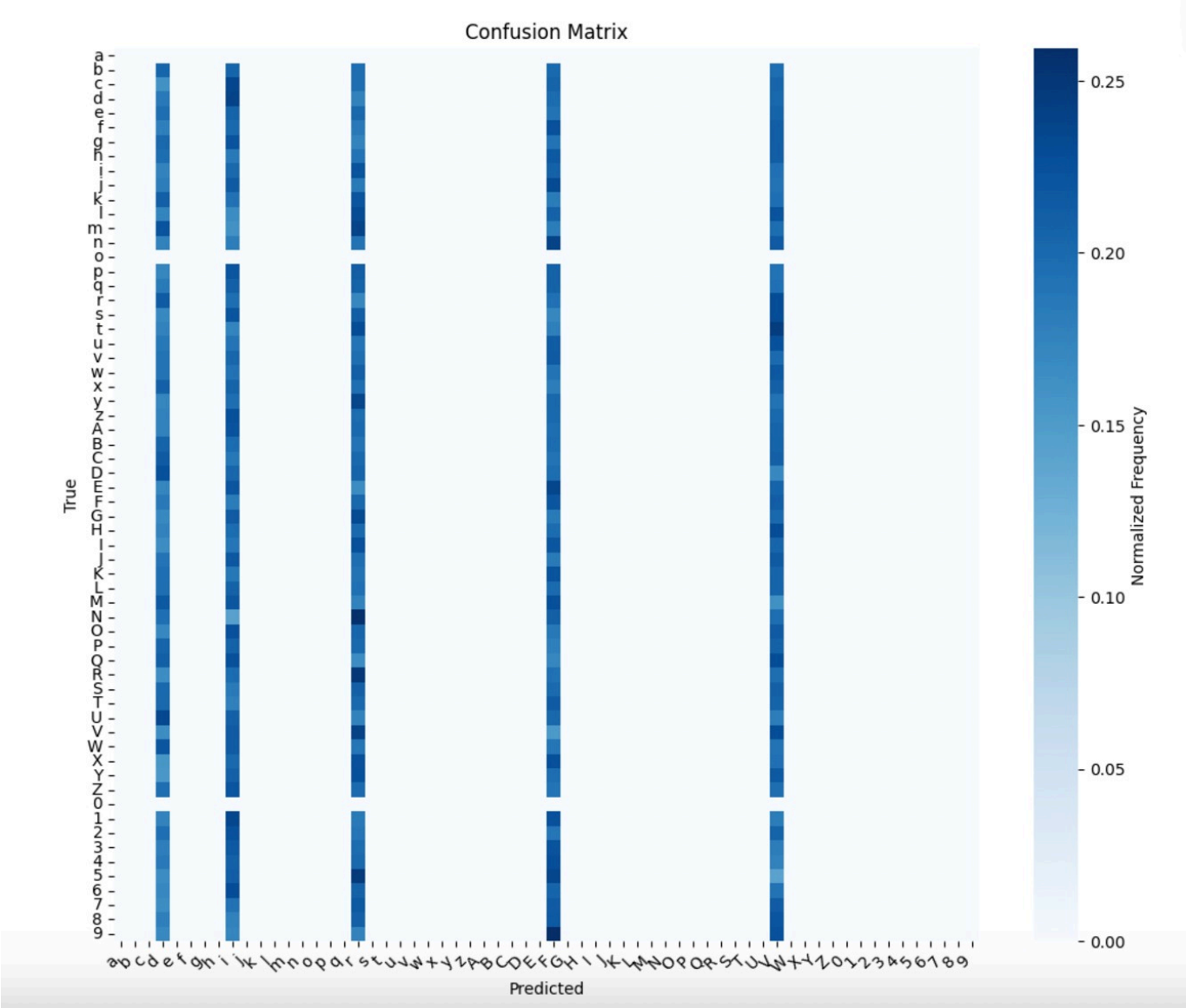
To conclude, from the plot above, we can see that the bars are quite imbalanced and close together, showing that certain clusters are not well-defined and the classes are misclassified. Furthermore, we can also see that there are a few clusters that have no points at all which definitely tells us that the K-Means did not converge to the most optimal solution and can be made more efficient. This could be due to noise or background interference that wasn't removed during preprocessing or that the CAPTCHA image dataset was just too complicated for the K-Means algorithm to form meaningful clusters. Thus, because of all these points mentioned above, we can see that K-Means performed quite poorly.

4.3 Transfer Learning

4.3.1 Numerical Analysis and Quantitative Metrics

- **Character-level Accuracy:** The Character-level accuracy on the testing set was 1.68%. This is extremely low compared to our standards from the Convolutional Neural Network. This can be attributed to the fact that Transformers are not well-suited for image classification problems. Transformers require very large datasets because they do not have the same ability to form inductive biases and detect local features. This low accuracy can be attributed to having a smaller dataset and the model failed to generalize the data.
- **Sequence-level Accuracy:** The Sequence-level accuracy on the testing set was 0.00%. Again, this is an extremely low accuracy. We believe that this is due to the small dataset size for Transformers and failure to generalize the dataset.

4.3.2 Visualization



- **Confusion Matrix:** As can be seen in the above confusion matrix, there are only five characters that are predicted by the model: e, j, s, G, W. As mentioned in the Numerical analysis, the difficulty that transformers have with local feature detection and inductive biases causes the model to predict the same characters for the true labels. Also, transformers require large datasets with a lot of diversity. Thus, this failure of the model to generalize and overfit to predicting specific characters every time may be a result of our training dataset being too small or not diverse enough. Incorporating a much larger dataset with the Transformer method and changing the problem from one of image classification to a numerical model may be a much better approach.
- **Why the Transformer Method Fails:** Due to the fact that Transformer based methods cannot calculate inductive biases, detect local features and require large, diverse datasets to function, the Transformer approach did not perform to the standards of the CNN.

4.4 Comparison of the 3 Models

Our project utilized three distinct models: CNN, K-Means, and a Transformer-based model, each tailored to the CAPTCHA recognition task.

CNN:

- **Strengths**
  - Achieved a character-level accuracy of 70.39% on the training set and 77.15% on the test set
  - Effective in recognizing local features, such as edges and corners, which are crucial for parsing distorted characters
  - Demonstrated spatial invariance, performing well despite rotations and shifts in the CAPTCHA images
- **Weaknesses**
  - The sequence-level accuracy (20.78% on the training set and 35.62% on the test set) was much lower than the character-level accuracy, suggesting room for improvement in recognizing entire sequences
  - Required extensive preprocessing for optimal performance

K-Means

- **Strengths**
  - As an unsupervised model, it effectively segmented data into clusters
  - Demonstrated basic clustering capability, as shown by the formation of 62 clusters corresponding to possible characters
- **Weaknesses**
  - The silhouette coefficient (-0.0657) and purity scores (<0.4) indicate poor clustering accuracy and lack of distinct character representations
  - High overlap between clusters and misclassification rates limited its utility for precise CAPTCHA recognition

Transformer-based model

- **Strengths**
  - Incorporated advanced attention mechanisms theoretically capable of handling overlapping and noisy characters
  - Showcased potential robustness to noise due to dynamic weighting features
- **Weaknesses**
  - Delivered extremely low character-level accuracy (1.68%) and sequence-level accuracy (0.00%)
  - Struggled to generalize due to the limited dataset size, highlighting Transformers' dependence on large-scale data

The CNN model appears to be the most effective approach, balancing accuracy and computational efficiency. While K-Means provided useful insights for segmentation, its clustering performance was insufficient for reliable recognition. The Transformer-based model, despite its theoretical advantages, was not suited for our dataset and task.

4.5 Conclusion and Next Steps

We investigated the effectiveness of three machine learning models - CNN, K-Means, and a Transformer-based model - for recognizing text-based CAPTCHA images. Among these, the CNN model achieved the best results, achieving high character-level accuracy and effectively handling challenges such as distortion and noise. Its ability to extract local features and adapt to spatial invariance made it particularly well-suited for the task. However, the model's sequence-level accuracy was comparatively lower, highlighting an area for improvement in recognizing entire CAPTCHA strings.

In contrast, the K-Means model provided limited success in clustering characters accurately. While it showed potential for unsupervised segmentation, its performance was hindered by overlapping clusters and low purity scores, likely due to the complexity and noise in the dataset. Similarly, the Transformer-based model underperformed, achieving minimal accuracy due to its reliance on large datasets and the added complexity of its architecture, which proved unnecessary for the relatively small and straightforward sequences in this task.

Moving forward, there are several opportunities to enhance performance. Further refinement of the CNN architecture, such as exploring deeper or more complex configurations, could improve sequence-level accuracy. Incorporating transfer learning using pre-trained models may also boost recognition capabilities, particularly for challenging sequences. Additionally, unsupervised segmentation techniques could be employed to preprocess the data more effectively, and augmenting the dataset with synthetic samples might enhance generalization. Exploring hybrid approaches that combine CNNs for feature extraction with simpler sequence models could also address current limitations. By building on these findings, future efforts can advance the development of robust CAPTCHA recognition systems that balance accuracy, efficiency, and adaptability.

References

[1] X. Xu, L. Liu, and B. Li, "A survey of CAPTCHA technologies to distinguish between human and computer," Neurocomputing, vol. 408, pp. 292–307, Sep. 2020, [Online] Available: doi: https://doi.org/10.1016/j.neucom.2019.08.109. Accessed: Oct. 03, 2024

[2] Roshanbin. N and Miller. J, "A SURVEY AND ANALYSIS OF CURRENT CAPTCHA APPROACHES," Journal of Web Engineering, pp. 001-040, 2021. [Online]. Available: https://journals.riverpublishers.com/index.php/JWE/article/view/4173. Accessed: Oct. 03, 2024

[3] Q. Li, “A computer vision attack on the ARTiFACIAL CAPTCHA,” Multimedia Tools and Applications, vol. 74, no. 13, pp. 4583–4597, Dec. 2013. [Online]. Available: doi: https://doi.org/10.1007/s11042-013-1823-z. Accessed: Oct. 03, 2024

Gantt Chart

Link: Gantt Chart

Contribution Table

Name	Final Contributions
Sanskriti Gupta	K-Means Results and Analysis, Github README
Vidushi Maheshwari	K-Means Model Coding, Github Pages
Deeksha Punchithiraya	Transfer Learning Model Coding
Preity Chavan	Comparison, Conclusion, Next Steps
Henry Raman	Transfer Learning Results and Analysis