**CAPTCHA Recognition**

AN8008 (2020/1 Trimester 2) Group 1:

Gan Pou Ing (Matriculation Number: G2000701C)

Lee Hoi Ming (Matriculation Number: G2000903A)

Tran Thi Hong Van (Matriculation Number: G2003457L)

Vadlamudi Santosh Krishna (Matriculation Number: G2000466C)

William Lim (Matriculation Number: G2001522F)

# Executive Summary

The Completely Automated Public Turing test to tell Computers and Humans Apart (“CAPTCHA”) test was first invented in 1997 as a method to evaluate whether the user operating a computer application is human or not. This was to prevent malicious attacks, SPAM etc. One of the most widely-used CAPTCHA approaches is text-based CAPTCHA. It consists of alphanumeric characters written in distorted and uneven manner. In this project, the project group applies Convolution Neural Network (“CNN”) learnt during AN8008 to decipher characters in a set of text-based CAPTCHA images.

CNN is chosen for image classification because it can compress an image into far fewer dimensions without the model losing quality. During the training process, CNN is also identifying patterns in an image.

Configuration of the CNN models used in this project took inspiration from other successful CNNs used for image classification - more specifically RestNet and VGG16. A narrow and deep network design is chosen. To ensure that trained CNN generalizes well to unseen data, some regularization is preferred. To this end, drop out layers are introduced throughout the model.

Four models were tested. These models are evaluated on two metrics (i) ability to accurately predict each character in an image; and more importantly (ii) ability to accurately predict sets of 5-character CAPTCHAs. The results are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Model 1 –**  **64 features** | **Model 2 –**  **Model 1, but with more neurons in the dense layers** | **Model 3 –**  **Model 1, but with higher dropout deeper into the network** | **Model 4 –**  **Model 3 + 128 features convolution layers** |
| Individual Character Accuracy | 94.76% | 91.40% | 94.39% | 95.14% |
| Full CAPTCHA Accuracy | 82.24% | 67.28% | 79.43% | 79.43% |

The results point to an interesting trade-off. Training for more complex features and having more regularization improve the ability to predict individual characters for unseen images. However, it also reduces the correlation of error occurrence within a 5-character set. For the purpose of solving CAPTCHA, Model 1 (the base model) is actually preferred despite its lower character-level accuracy.

This project shows that entry-level CNN models can already achieve decent accuracy in solving text-based CAPTCHA. It means that text-based CAPTCHA is no longer fit-for-purpose in safeguarding mission-critical systems from SPAM attacks.

# Introduction and Objective

The Completely Automated Public Turing test to tell Computers and Humans Apart (“CAPTCHA”) test was first invented in 1997 as a method to evaluate whether the user operating a computer application is human or not. This was to prevent malicious attacks, SPAM etc.

One of the most widely-used CAPTCHA approaches is text-based CAPTCHA. It consists of alphanumeric characters written in distorted and uneven manner. The three main characteristics of a text-based CAPTCHA that made it difficult for the machines to read are:

* Invariant Recognition – ability to recognize large amount of variation in shapes of characters. Although humans can easily read different text sizes, it becomes difficult for machines.
* Segmentation – ability to separate one letter from the other. Characters are crowded together with no white spaces between them, making it harder for machines to tell apart characters.
* Context – ability to understand the entire code holistically. Sometimes few characters can make sense only when looking at the whole sequence of characters, which is something again difficult for the machines.

With growing computation abilities, machines were able to decipher many text-based CAPTCHA with time, giving rise to more complex versions of CAPTCHA like reCAPTCHA and other advanced tests.

In this project, the project group aims to apply Convolution Neural Network (“CNN”) learnt during AN8008 to decipher characters in a set of text-based CAPTCHA images.

# Literature Review

Many researchers have investigated into how to develop tests that can classify whether the user of a system is a human or not. Such tests are widely accepted to help artificial intelligence (“AI”) to develop further and solve advanced and relevant problems in real world.

Gabriel Moy, Nathan Jones, Curt Harkless, and Randall Potter in their 2003 paper, “Distortion Estimation Techniques in Solving Visual CAPTCHAs”, argued that a program that can pass the CAPTCHA tests, can solve the much more complex problem of recognizing objects in scenes. This was proven to be true. Over the years advanced AI models have been developed that could not only beat traditional CAPTCHA tests but were also able to make great progress in object recognition. More advanced CAPTCHA tests were developed which, at times, are difficult even for humans to solve.

Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford in their 2004 paper, “CAPTCHA: Using Hard AI Problems for Security”, concluded that the fields of AI and Cryptography have a lot to contribute to each other – Cryptographers need to design keeping AI models in mind as machines are the way of the future, and AI researchers need to beat complex codes, like CAPTCHA, to make progress with object recognition in complex backgrounds.

Zahra Noury and Mahdi Rezaei in their 2020 paper, “Deep-CAPTCHA: a deep learning based CAPTCHA solver for vulnerability assessment”, tuned a CNN based model to reveal strengths and weaknesses of current CAPTCHA models and suggested ways to improve them. They have demonstrated how using Softmax layers, instead of Sigmoid layers helped improve the accuracy of image recognition significantly.

# Problem Statement

Some websites still use the traditional text-based CAPTCHA to filter bots from humans. In this project, the project group aims to apply Convolution Neural Network (“CNN”) learnt during AN8008 to decipher characters in a set of text-based CAPTCHA images. As the solution produced by the project group may not represent the most advanced image classification techniques available in the market today, a good classification accuracy will point to the need for companies that rely on text-based CAPTCHA to use more advanced approach if they want to prevent bots from accessing their systems.

# Data – Analysis, Visualization and Pre-processing

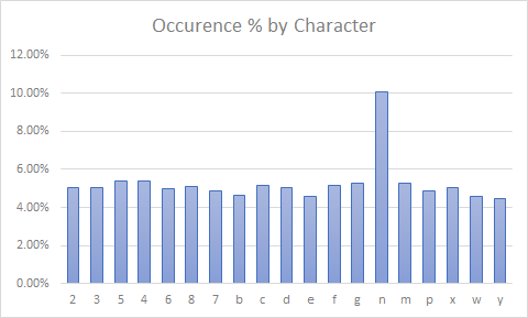
**Data source**

Dataset comes from Kaggle titled “CAPTCHA Images”. It is a collection of 1070 text-based CAPTCHA images with 200 X 50 pixels resolution. Most of the images are in .png format and a minority of them are in .jpg format. Each image consists of five alphanumeric characters with noise applied to them like blurs, lines, and distortions. This is a labelled dataset, with the file name of each image stating the five characters in the image.

****

**Descriptive statistics**

The following chart shows the frequency with which each alphanumeric character appeared in the dataset.



The following are noted:

* Character count are fairly evenly distributed, with the exception of “n”. Many instances of “n”s appearing next to each other, supposedly to induce machines to recognize them a “m”.
* Some characters that are hard for humans to decipher are excluded, e.g. “1” vs “l”, “0” vs “o”. This reduces full 36 alphanumeric character set to just 20 characters in the data.

**Data pre-processing – images**

Python’s Matplotlib library’s “image” module is used to read in the image files in the dataset. This module treats .png files and .jpg files separately. More specifically, the library returns values between 0 to 1 for .png files while it returns values between 0 to 255 for .jpg files. To standardize the input before training the model, all entries of .jpg files are divided by 255 to align to the magnitude of .png images. This treatment is chosen over grossing up the values from .png files because neural network tends to work better with small input values (e.g. [0,1], or values from a standard normal distribution).

The images are in RGBA format. Upon closer inspection, these images are intended to be greyscale (i.e. the values for the red, green and blue channels are the same). So, to speed up training, only values of the red channel are retained. This effectively turns the image back to a greyscale one.

In anticipation of the use of Kera’s Conv2D layer for the CNN model, the data has to be reshaped into a 4-dimension one representing [sample size, image height, image width, channel]. With 1070 greyscale images of 50 x 200 pixels, the input data frame dimension is set as [1070, 50, 200, 1].

**Data pre-processing – data labels**

As discussed earlier, names of the image files contain the characters shown in the images. So, the file names are first read into a list. The nature of the problem is essentially a classification exercise. For that, Keras requires the data to take the format of “one-hot” representation. More specifically, image labels are re-shaped as follows:

* Each character can take one of the 36 values – 10 digits plus 26 lowercase alphabets.
* There are 1070 samples in the data set, each containing 5 characters.
* It is expected that the entire image will be read into the training network at once, and produce an output of a sequence of five predicted characters to match against the actual 5-character set.

This result in an output data frame dimension of [5, 1070, 36]

**Split between training, validation and testing data**

To ensure the robustness of the model trained in predicting unseen images, the group adopted the “train-validate-test” approach to segregate available samples. As the image file list is sorted in alphabetical order, to ensure that the training data has a good mix of different alphanumeric characters, the samples are first randomly reshuffled. Then, the samples are split as follows:

* 80% of the samples (i.e. 856 samples) are used for training.
* 10% of the samples (i.e. 107) are used for validation. Only when a model shows good accuracy on the validation set will the model be applied to the testing data.
* 10% of the samples (i.e. 107) are used for testing.

# Model Configuration and Evaluation

The project group chose to use CNN to model text-based CAPTCHA images because images inherently have high dimensionality. Each pixel can be considered as one feature. It is computationally expensive to train such a high dimensional network. CNN is effective for image classification because it can compress an image into far fewer dimensions without the model losing quality. During the training process, CNN is also identifying patterns in an image.

CNN model was coded using Python’s TensorFlow and Keras libraries.

A total of four model configurations were tested. Here is a summary of the model specifications. Model 1 represents the base configuration. For subsequent models, changes from one model to the next is highlighted in bold.

| **Model** | **Model 1** | **Model 2** | **Model 3** | **Model 4** |
| --- | --- | --- | --- | --- |
| Convolution 1 | 16 3X3 filters | 16 3X3 filters | 16 3X3 filters | 16 3X3 filters |
| Convolution 2 | 16 3X3 filters | 16 3X3 filters | 16 3X3 filters | 16 3X3 filters |
| MaxPool 1 | Size 2, Stride 2 | Size 2, Stride 2 | Size 2, Stride 2 | Size 2, Stride 2 |
| Dropout 1 | Dropout – 0.2 | Dropout – 0.2 | Dropout – 0.2 | Dropout – 0.2 |
| Convolution 3 | 32 3X3 filters | 32 3X3 filters | 32 3X3 filters | 32 3X3 filters |
| Convolution 4 | 32 3X3 filters | 32 3X3 filters | 32 3X3 filters | 32 3X3 filters |
| MaxPool 2 | Size 2, Stride 2 | Size 2, Stride 2 | Size 2, Stride 2 | Size 2, Stride 2 |
| Dropout 2 | Dropout – 0.2 | Dropout – 0.2 | **Dropout – 0.3** | Dropout – 0.3 |
| Convolution 5 | 64 3X3 filters | 64 3X3 filters | 64 3X3 filters | 64 3X3 filters |
| Convolution 6 | 64 3X3 filters | 64 3X3 filters | 64 3X3 filters | 64 3X3 filters |
| MaxPool 3 | Size 2, Stride 2 | Size 2, Stride 2 | Size 2, Stride 2 | Size 2, Stride 2 |
| Dropout 3 | Dropout – 0.2 | Dropout – 0.2 | **Dropout – 0.4** | Dropout – 0.4 |
| Convolution 7 | - | - | - | **128 3X3 filters** |
| Convolution 8 | - | - | - | **128 3X3 filters** |
| MaxPool 4 | - | - | - | **Size 2, Stride 2** |
| Dropout 4 | - | - | - | **Dropout – 0.5** |
| Flatten | Yes | Yes | Yes | Yes |
| Fully connected layers  (5 parallel stacks) | Dense 64 Dropout – 0.2  Dense 36 | **Dense 128** Dropout – 0.2  Dense 36 | **Dense 64** **Dropout – 0.4**  Dense 36 | **Dense 128** **Dropout – 0.5**  Dense 36 |
| Batch Size | 32 | 32 | 32 | 32 |
| Epoch | 100 | 100 | 100 | **200** |

**Model 1 – Base Model**

Configuration of the base CNN model took inspiration from other successful CNNs used for image classification - more specifically RestNet and VGG16. The following features are adopted in the base model of this project:

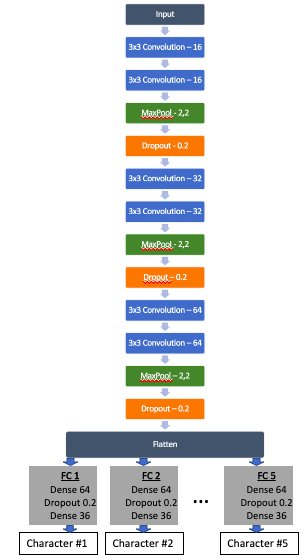
* A narrow and deep network design is chosen
* Filter size of 3x3 is used
* Number of filters doubles in successive (groups of) layers (e.g. 64 - 128 - 256 - 512). This is logical as earlier layers identifies basic patterns, and later layers identifies more complex patterns using a permutation of earlier basic patterns.
* The “same” padding approach is used so that pixels at the corners and edges are used as frequently as pixels in the centre.
* These networks seem to follow the structure of a convolution – convolution – pooling stack.
* All layers, except for the final softmax layer, uses Relu.

In addition, good practice of machine learning is observed:

* To ensure that trained CNN generalizes well to unseen data, some regularization is preferred. To this end, drop out layers are introduced throughout the model.
* Since the output is categorical, “categorical cross-entropy” loss function is used.
* Optimizer “Adam” is chosen for its dynamic learning rate.

Unlike usual image classification problems which has a 1:1 mapping between input and output, this CAPTCHA problem takes in one input image and provides five classification outputs for each of the characters in the image. To cater for this, once the convolution layers have identified the patterns in the overall image set, the flattened interim output is fed into five separate stacks of fully-connected layers. Each stack is responsible for learning one position in the image (i.e. first stack is responsible for the left-most character, the last stack for the right-most character).

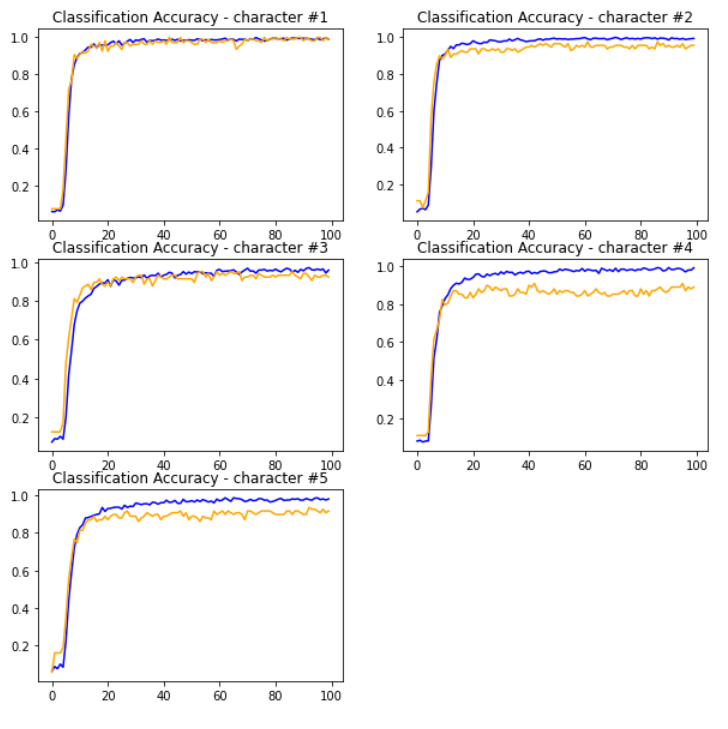
This leads to the following base configuration:



As the images are small and involve simple characters in greyscale, the base model starts with convolution layer with 16 features, growing eventually to 64 features. Fully-connected layers start with 64 neurons to correspond to the 64 features coming out from the convolution layers. Output layers correspond to each of the 36 alphanumeric characters.

Performance on training and validation data

Accuracies on both training and validation data increase quickly and stabilize around the epoch #10. Validation accuracy on the first character performed best. Validation accuracy on the fourth character performed worst, and has greater concern of over-fitting. Overall, character-level prediction accuracy exceeds 90%, sufficiently well to apply to testing data.



Performance on testing data

Two performance metrics are used to evaluate accuracy on testing data:

* Individual character accuracy: Proportion of accurately-predicted characters.
* Full CAPTCHA accuracy: Proportion of accurately-predicted 5-character set. (i.e. all five characters in the 5-character set must be predicted correctly). Given the use case of CAPTCHA prediction, this is the metric that really matters.

Assume that a 90% character-level accuracy is achieved. If character-level prediction error occurs randomly, one would expect a full CAPTCHA accuracy of 59% (= 90% ^5). This is the lower bound of full CAPTCHA accuracy. In contrast, if a wrong prediction in, say, the first character occurs only when all other characters in the set are also predicted wrongly, full CAPTCHA accuracy will be 90%. This is the upper bound of full CAPTCHA accuracy. By comparing the pair of performance metrics, one can infer the propensity for prediction errors to correlate.

For the base model, its performance on the testing data is as follows:

* Individual character accuracy: 94.8%
* Full CAPTCHA accuracy: 82.2% (vs an implied lower bound of 76.4%). This indicates a mild correlation between prediction errors within a 5-character set.

The spread of prediction errors is consistent with the observation on the training/ validation data. All of the first characters were predicted correctly. Accuracy on the last character also did well, likely to be due to it being next to the clearly-identifiable right boundary of the image. Classification of characters in the middle is more challenging for the model, possibly because the boundaries with adjacent character is less clear.

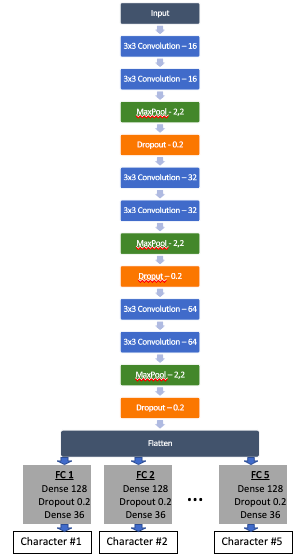
|  |  |  |
| --- | --- | --- |
| **Position of prediction error** | **Count** | **Share of Error** |
| 1st character | 0 | 0.0% |
| 2nd character | 8 | 28.6% |
| 3rd character | 7 | 25.0% |
| 4th character | 9 | 32.1% |
| 5th character | 4 | 14.3% |

Inability to distinguish between n vs m is the main issue with the model. Other than this, there is no other systemic issues with the model.

|  |  |  |  |
| --- | --- | --- | --- |
| **Expected** | **Predicted** | **Count** | **Share of Error** |
| n | m | 3 | 10.7% |
| m | n | 3 | 10.7% |
| 3 | b | 2 | 7.1% |

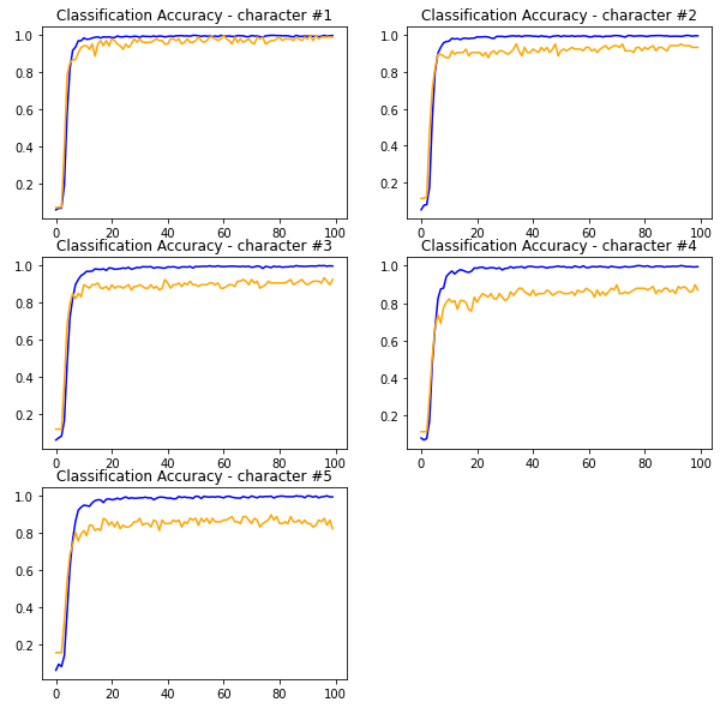
**Model 2 – Increase neurons to fully-connected layers**

In Model 1, the model exits the convolution/pooling stage with 64 features on a reduced image size of 7 x 25. This flattens to 11200 parameters. Instead of connecting that to a 64-neuron dense layer for each predicted character, Model 2 tests if having more neurons in the dense layer improves model performance. A 128-neuron is used in Model 2.



Performance on training and validation data

While prediction accuracy on the first character remains good, there is more problems of over-fitting on the other characters. Overall character-level accuracy is still close to 90%.



Performance on testing data

For Model 2, its performance on the testing data is as follows:

* Individual character accuracy: 91.4%
* Full CAPTCHA accuracy: 67.3% (vs an implied lower bound of 63.8%). This indicates prediction errors within a 5-character set is nearly random.

The spread of prediction errors is consistent with the observation on the training/ validation data. All of the first characters were predicted correctly. Accuracy on the third and fourth characters worsened significantly.

|  |  |  |
| --- | --- | --- |
| **Position of prediction error** | **Count** | **Share of Error** |
| 1st character | 0 | 0.0% |
| 2nd character | 7 | 15.2% |
| 3rd character | 13 | 28.3% |
| 4th character | 19 | 41.3% |
| 5th character | 7 | 15.2% |

The pairing of “n vs m” continues to create problem. Misclassifying 6 as e, and 3 as 7, emerged as a problem in Model 2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Expected** | **Predicted** | **Count** | **Share of Error** |
| m | n | 9 | 19.6% |
| 3 | 7 | 4 | 8.7% |
| 6 | e | 4 | 8.7% |
| n | m | 3 | 6.5% |

Due to the relatively poor performance, subsequent model tuning will not build upon Model 2.

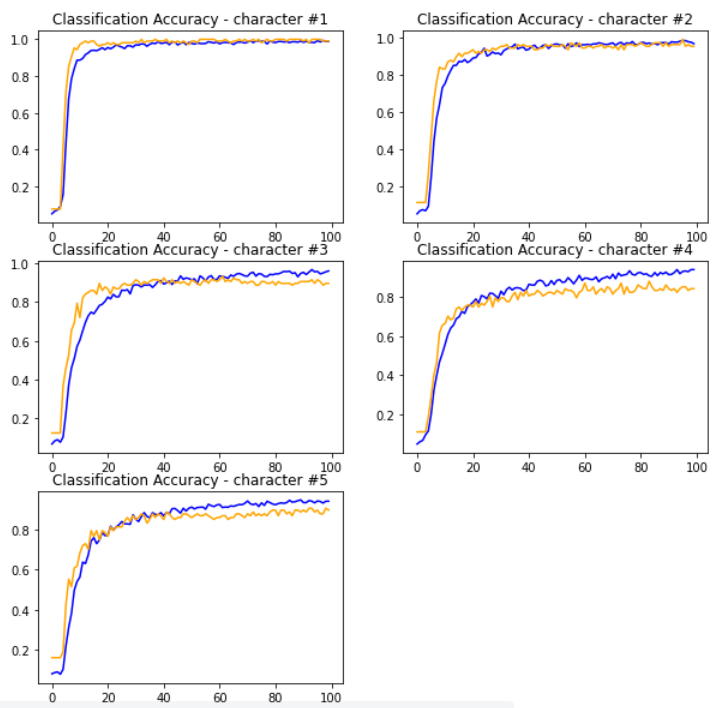
**Model 3 – Increase dropout rate deeper into the network**

With increasing depth of the neural network, each feature represents a more complex pattern in the image. Over-fitting is more likely to occur if the weights given to these complex patterns mimics the training data too closely. More regularization is preferred to cater for unseen images. One way to achieve that is to increase the dropout rate as one goes deeper into CNN. In Model 3, dropout rate increases from 0.2 at the start of the network to 0.4 at the deepest part of the network.



Performance on training and validation data

As intended, much of the over-fitting seen in Model 1 is cured by increasing drop out rate at the deep part of the network. Overall accuracy continues to stand above 90%.



Performance on testing data

For Model 3, its performance on the testing data is as follows:

* Individual character accuracy: 94.4%
* Full CAPTCHA accuracy: 79.4% (vs an implied lower bound of 75.0%). This indicates prediction errors within a 5-character set is nearly random.

The spread of prediction errors is consistent with the observation on the training/ validation data. All of the first characters were predicted correctly; and performance on the second character has also improved. Accuracy on the third and last characters worsened compared to Model 1.

|  |  |  |
| --- | --- | --- |
| **Position of prediction error** | **Count** | **Share of Error** |
| 1st character | 0 | 0.0% |
| 2nd character | 4 | 15.2% |
| 3rd character | 11 | 28.3% |
| 4th character | 9 | 41.3% |
| 5th character | 6 | 15.2% |

Inability to distinguish between n vs m is the main issue with the model. Other than this, there is no other systemic issues with the model.

|  |  |  |  |
| --- | --- | --- | --- |
| **Expected** | **Predicted** | **Count** | **Share of Error** |
| m | n | 3 | 10.0% |
| n | m | 2 | 6.7% |
| 3 | c | 2 | 6.7% |
| B | f | 2 | 6.7% |

Since the accuracy of Model 3 is comparable to Model 1, and is able to address over-fitting better, subsequent model tuning builds on Model 3.

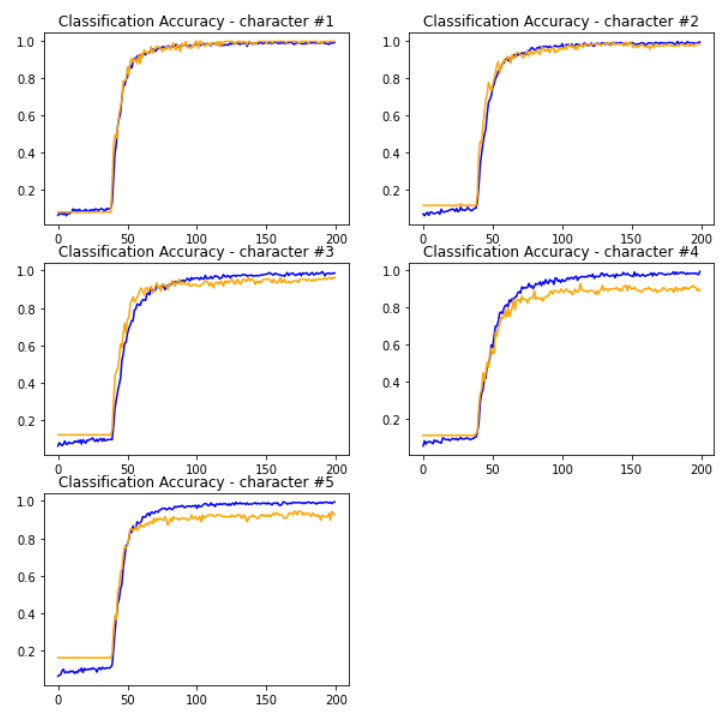
**Model 4 – Add more convolution layers**

In VGG16, convolution layers increase from 64 feature at the start to 512 features at the deep end – an 8-fold increase. In Model 4, more convolution layers are added to Model 3 to achieve the same 8-fold increase in feature count. Drop out and neurons in the fully connected layers are increased accordingly. Epoch was also increased to 200 in case that more parameters require a longer training routine.



Performance on training and validation data

The results are similar to Model 3 with a mild improvement in the performance on the third and last characters.



Performance on testing data

For Model 4, its performance on the testing data is as follows:

* Individual character accuracy: 95.1%
* Full CAPTCHA accuracy: 79.4% (vs an implied lower bound of 77.8%). This indicates prediction errors within a 5-character set is nearly random.

Compared to Models 1 and 3, prediction errors in Model 4 are more concentrated in the third and fourth characters. Significant improvement is seen for the prediction accuracy of the last character.

|  |  |  |
| --- | --- | --- |
| **Position of prediction error** | **Count** | **Share of Error** |
| 1st character | 1 | 3.8% |
| 2nd character | 3 | 11.5% |
| 3rd character | 10 | 38.5% |
| 4th character | 11 | 42.3% |
| 5th character | 1 | 3.8% |

Prediction error relate to the pairing of “n vs m” is also far more concentrated in Model 4 compared to the other models tested. Misclassifying “m” as “n” is more frequent than the reverse, possibly due to there being more samples of “n” than “m” in the data. Other than this, there is no other systemic issues with the model.

|  |  |  |  |
| --- | --- | --- | --- |
| **Expected** | **Predicted** | **Count** | **Share of Error** |
| m | n | 9 | 34.6% |
| n | m | 2 | 7.8% |
| 3 | 7 | 2 | 7.8% |

# Model Comparison

To recap, the four models tested are:

* Model 1 – Base model
* Model 2 – From Model 1, increase number of neurons in the fully-connected layer
* Model 3 – From Model 1, increase dropout rate for layers deeper into the network since the more "advanced" features should be more generalized
* Model 4 – From Model 3, (i) add one more stack of layers with 128 features before flattening, (ii) change the first dense layer to 128 neurons to match, (iii) adjust dropout rate accordingly, (iv) increase to 200 epochs.

Below table is a summary of performance of these models:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Model 1** | **Model 2** | **Model 3** | **Model 4** |
| Individual Character Accuracy | 94.76% | 91.40% | 94.39% | 95.14% |
| Full CAPTCHA Accuracy | 82.24% | 67.28% | 79.43% | 79.43% |

The results point to an interesting trade-off. Training for more complex features and having more regularization improve the ability to predict individual characters for unseen images. However, it also reduces the correlation of error occurrence within a 5-character set. For the purpose of solving CAPTCHA, it is preferred that errors concentrate on a few 5-character sets rather than spread over many sets. So, Model 1 (the base model) is actually preferred for this use case despite its lower character-level accuracy.

# Business Implications and Recommendation

This project shows that entry-level CNN models can already achieve decent accuracy in solving text-based CAPTCHA. More advanced techniques are expected to perform even better. It means that text-based CAPTCHA is no longer fit-for-purpose in safeguarding mission-critical systems from SPAM attacks. Anecdotal evidence supports this view as most Google CAPTCHA seen nowadays are of the sort that ask users to identify items (e.g. traffic light) from small and noisy images.

# Future Study

The group may conduct future studies along the following directions:

* Improving accuracy of existing dataset. This may include (i) using image augmentation techniques such as slightly rotating or sheering training images to enrich data set and improve generalization; or (ii) instead of CNN, use techniques such as LSTM which recognizes all characters are serially correlated because elements designed to interfere with predictions cuts across all characters in the image.
* Develop new models to cater for more complex CAPTCHAs. Existing model deals with texts that forms a 5-character set. An improved version of the model may take images with variable length text. It will start by counting the number of characters and pinpointing the location of each in the image before deciphering for the characters. This approach can also be extended to recognizing number plates on vehicles. Alternatively, a new model may be trained to cope with text set that contains both uppercase and lowercase characters.

# Appendix – Presentation Video Link

https://youtu.be/fvHm4S0foT8