IMAGE DATA GENERATION and TRAINING:

1. Generated 50k images and trained the model but model was not predicting numbers and '@' (common in business cards) so generated more emails and numbers text and added to corpus. Discarded previous images.
2. Generated 50k images and trained the model which was failing to predict text which had all uppercase characters. Added a corpus with all uppercase words. Discarded previous images.

3. Generated 60,000 images using 57 different fonts, 145 coloured backgrounds and a large corpus of generated emails and randomly selected unique words from english vocabulary of nltk dataset. Added rotation, noise, underline, cut through and random crop ro the images.
    - 3.1. Generated 20000 one word and 5000 two word images using min:10 and max:22 font size and augmentations in text_renderer/configs/default.yaml.
    - 3.2. Generated 20000 one word and 5000 two word images using min:5 and max:12 font size and augmentations in text_renderer/configs/default.yaml.
    - 3.3. Trained a model on above data which was almost always predicting common words like 'and', 'his', etc for highly blurred data.
4. Generated 10000 one word images with length greater than 4 with extra noise. Trained model with combined data.
5. Concatenated canon data from public dataset in training data.

Examples of Initially Generated Images:

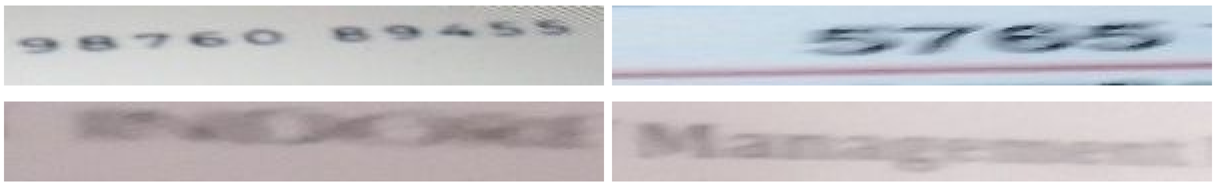
HEALTH
ALL
dffhc@tcm.dk
MIGHT
REASONS
would

Examples of generated extra noisy data:

catling
BRAINGE
most
STOORY
fitander

Real World Data

Took picture of 5 cards from different angles and adding shadows and shine to the picture. Cropped text regions keeping the text in different regions of the cropped image. The set have 161 images.
Examples:



Converting images to tfrecords

1. Used utility_scripts/create_annotate_data.py to make annotation files suitable to be used by aocr.
2. Used <aocr dataset> command to generate tfrecords.
3. Used utility_scripts/merge_tfrecords to merge different tfrecords

MODEL-DEFAULT:

1. Used default cnn of the library, embedding size 20 and 2 attention layers with 128 hidden units each.
2. Trained the model using batch size of 64 for 20 epochs on train.tfrecords.

3. Accuracy of model trained only on train.tfrecords

	reference	droid	blurry	canon	testing.tfrecords
Avg. percent of text match	50.50%	20.60%	23.53%	17.67%	98.40%
Full text match	24.91%	11.51%	4.73%	8.73%	91.80%

4. Trained the model for 10 epochs on a concatenated data of train, blurry and canon data.

Checkpoint no.	24490	24590	24690	24790	24890
reference	75.06%	74.95%	74.93%	75.09%	75.09%

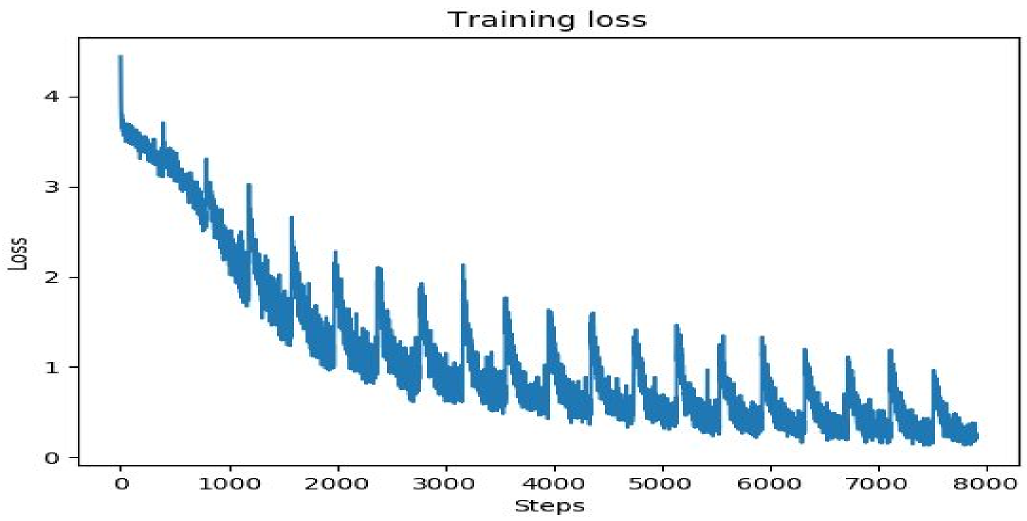
5. Selected checkpoint number 24790 and tested on real_data with accuracy of 11.09%. Fine-tuned the model for 5 epochs using canon data

6. Final test accuracy (24960)

	Average of percentage of text match	Full text match
reference	77.58%	59.87%
droid	46.39%	33.77%
e63	14.24%	4.71%
Unseen set from generated data (new-test)	85.61%	58.30%
real_data	13.21%	0.62%

MODEL-CUSTOM:

1. Added an extra convolutional layer with relu activation at the top of default cnn architecture. Used Embedding size 20 and 2 attention lstm layers with 128 hidden units each.
2. Trained the model using batch size of 64 for 5 epochs.



Accuracy of different model checkpoints on reference data

Checkpoint no.	8200	8300	8400	8500	8600
reference	72.86%	72.65%	71.26%	71.65%	71.22%

Selected checkpoint no. 8200 as final model.

Final model accuracy

	Average of percentage of text match	Full text match
reference	72.86%	48.55%
droid	47.44%	32.26%
e63	14.28%	3.07%
Unseen set from generated data (new-test)	85.80%	57.64%
real_data	10.6%	0%

API:

1. Exported the modes to SavedModel format.
2. Created REST API in flask to use the model.
3. Created page where user can upload multiple images and get the following json response

```
        {
                "response" :[
                        {
                                "image_file_name" : "   ",
                                "predicted text" : "   ",
                                "probability" : "   "
                        },
                        {},
                ...]
        }
```

ONLINE APP

1. Exported model checkpoint 24960 to savedmodel format and used it in the app.
2. Made changes to app.py and added necessary files to deploy to heroku.
3. Deployed at https://ocr-abhishar-sinha.herokuapp.com/

CHANGES TO text_renderer:

1. Added different fonts from font-squirrel.com (open-source) to generate images.
2. Added coloured background images and added more font colours in config/default.yaml
3. Added emails and numbers corpus, uppercase and lowercase words corpus to data/corpus.

CHANGES TO attention-aocr/aocr:

1. Modified train method of Model class in model/model.py to save training loss at every step to a file and save the loss vs steps plot as png.
2. Modified test method of Model class in model/model.py to save predicted vs actual text for every incorrect prediction. Used it to see if there is a pattern in incorrect predictions.
3. Created CNN_cust class in model/cnn_custom_two.py containing custom cnn architecture.
4. Added variable self.custom_cnn in Model in model/model.py. If true use CNN_cust as cnn layer else use CNN (default).
5. Change CHARMAP of DataGen class in util/datagen.py to include larger alphabet.
6. Changed default values of FORCE_UPPERCASE=False (To allow lowercase too) and TARGET_VOCAB_SIZE = 84 (After counting size of input data alphabet using utility_scripts/get_alphabet.py)

Project Structure

```
├── attention-ocr    --->    library to train and test ocr model
├── checkpoints    --->    contains model checkpoints at different training steps
├── datasets    --->    contains training and test data
├── exported-model    --->    contains models exported in SavedModel format
├── history   --->   contains loss vs steps data and plot, and model config for every train run
├── static
    ├── index.html    --->    webpage where user can upload images to get json response
├── test_logs    --->    contains predicted vs actual texts for each test run
├── text_renderer    --->    library to generate text images
└── utility_scripts
    ├── create_annotate_data.py    --->    makes annotation file generated by text-renderer
suitable for use by attention-ocr
    ├── crop_backgrounds.py    --->    splits an image into 6x6 grid and saves each part
    ├── gen_email_mob.py    --->    generates fake emails and numbers
    ├── get_alphabet.py    --->    gives the alphabet set by traversing through labels
    ├── merge_tfrecords.py    --->    merges multiple tfrecords files into a single file
    ├── only_english.py    --->    removes non-english and non-numeric labels from annotations
    └── testing_inception.ipynb    --->    used to get index of mixed5 layer from layers list of
keras inceptionV3 model
├── app.py    --->    flask web app that lets users upload text images and returns json response
├── daily_logs.md    --->    daily log of completed tasks
├── instructions.txt    --->    instructions to run app
├── make_predictions.py   --->   generates predictions in <img_name> <predicted_text> format
├── my_run.sh    --->    runs make_predictions.py
├── requirements.txt    --->    requirements to run the web app
├── train_model.sh    --->    used to train model
```