

# Machine Learning Engineer Nanodegree

---

## Capstone Project

---

Suhaib Hasan  
November 08th, 2018

## I. Definition

---

### Project Overview

Enterprise Content Management (ECM) is a technology which is used to capture, store, retrieve and manage content in electronic form. ECM tools allow storage and management of large-scale documents (millions) across range of sectors such as banking, auditing, insurance, pharmacy etc. A very critical component of ECM is capture of documents & metadata.

Capture is still a manual process in many organizations i.e. metadata entry is done by data entry operators. Goal of this project is to use deep learning for auto extraction of data for consumption by ECM tools such as IBM FileNet, OT DOCUMENTUM etc.

For this project we will be using OCR dataset provided in below link:

<https://supervise.ly/>

**Note:** User needs to first register on this link and then navigate to "import" → "Datasets Library" tab and then click on "anpr\_ocr" project

Another research on same topic can be found on below link:

<https://nicholastsmith.wordpress.com/2017/10/14/deep-learning-ocr-using-tensorflow-and-python/>

### Problem Statement

The Goal of this work is to automate data extraction process from electronic documents to be consumed by ECM tools. Manual data extraction has many disadvantages such as:

- 1) Precision error i.e. there are chances that incorrect manual entry is done which might result in data inconsistency.
- 2) Overhead Cost i.e. manual extraction requires lot of manual work to extract data from each document which in turn increases cost for organization.

Through automated data extraction process, we can achieve higher accuracy percentage, reduce manual work and operational cost.

I will be using CNN (Convolutional Neural Network) to resolve this problem as CNN's are best suitable to identify data from images. There are many pre-trained CNN models present but for this problem we will define our own CNN model which is discussed later in this report. There will be some preprocessing done on input data such as image resize, converting image to machine readable format etc. More details will follow later in the report.

## Metrics

This project will be evaluated on accuracy matrix i.e. how accurately our model is predicting characters and numbers in image. Accuracy in our case can be calculated using below formula:

**Accuracy = number of characters and numbers predicted correctly / total number of characters and numbers in image.**

Accuracy is an important metric of this project as our problem statement is to resolve inaccurate data entry issue during manual process. This can be better understood through example. Consider an insurance claim form containing all details including SSN. While entering data manually, data entry operator enters incorrect SSN details by mistake and it is saved in system. This will cause delay in claim process which might not be well received by customer. Further we will have to invest time and money to clean up data in system as well, hence organization will be at loss at both fronts. Thus, it is very important that data is entered correctly, thus to tackle this problem system should accurately read data present in electronic form.

## II. Analysis

---

### Data Exploration

The anpr\_ocr dataset contains more than 10K images for training system. These are colored images of size {"height": 34, "width": 152}, few examples are below:



This dataset also contains annotations for these images in json form, below is one example:

```
{"tags": ["train", "train"], "description": "A058BP45", "objects": [], "size": {"height": 34, "width": 152}}
```

This JSON file contains below fields:

- 1) Tags – this will define whether it's a training or testing image.
- 2) Description- this contains the characters and number of image
- 3) Size – it defines the size of image.

Maximum plate size is 8 in test data set. This calculation is done in "get\_counter" method, below is the result:

---

```
Max plate length in "anpr_ocr__train": 8
```

Same method "get\_counter" is used for finding out the most frequently occurring number and letters in test data set, below is the result for same:

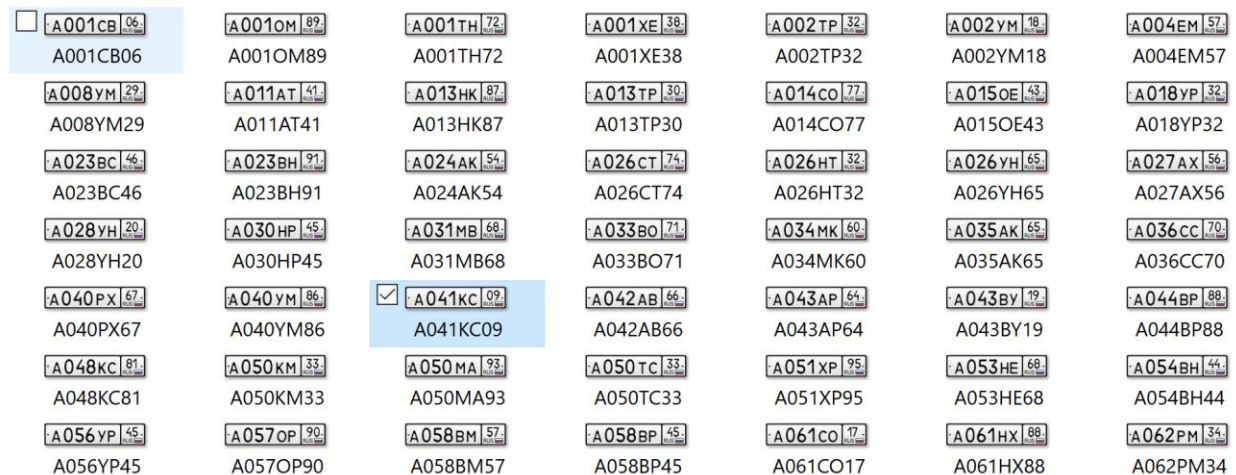
```
-----  
Letters: 0 1 2 3 4 5 6 7 8 9 A B C E H K M O P T X Y
```

**Note:** There are no mislabeled images in our dataset

## Exploratory Visualization

The input data set consists of two folders containing images and their corresponding labels.

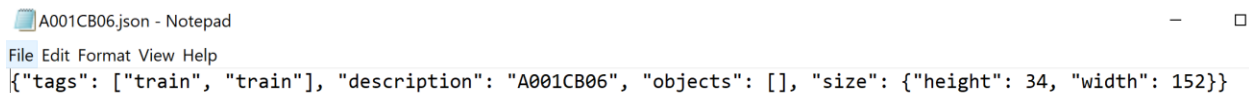
Below is the screen shot of images folder, it contains more than 10 k different images.



Other folder contains image labels in JSON format, below is the screen shot:

|  |               |                  |           |      |
|--|---------------|------------------|-----------|------|
|  | A001CB06.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A001OM89.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A001TH72.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A001XE38.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A002TP32.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A002YM18.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A004EM57.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A007MP60.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A008HM63.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A008OC32.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A008YM29.json | 10/24/2018 10:50 | JSON File | 1 KB |
|  | A011AT41.json | 10/24/2018 10:50 | JSON File | 1 KB |

An example JSON file is shown below:



```
A001CB06.json - Notepad
File Edit Format View Help
{"tags": ["train", "train"], "description": "A001CB06", "objects": [], "size": {"height": 34, "width": 152}}
```

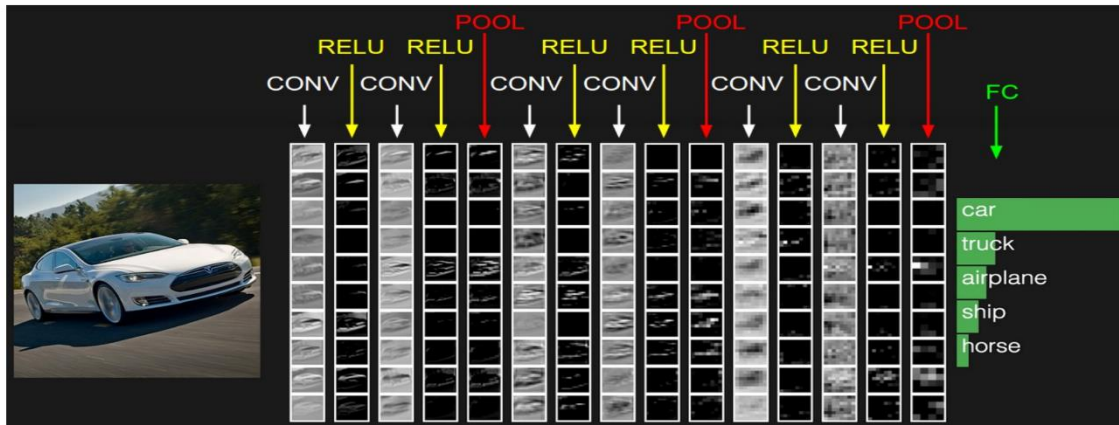
As it can be seen clearly from above JSON file it consists of three parameters which will help system in identifying type of image i.e. train, validate or testing, label of image and the image size.

## Algorithms and Techniques

The classifier is Convolutional Neural Network, which is standard algorithm for problems involving image processing. A Convolutional Neural Network is like a Neural Network i.e. they are made up of neurons and have learnable weights and biases. The difference being that a CNN can take input as an image which is primary requirement of our problem statement. CNN consist of four main layers:

- 1) Convolutional Layer – it will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- 2) Pooling Layer – it will perform a down sampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12]
- 3) Fully Connected Layer - layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.
- 4) Activation Layer – it will apply an elementwise activation function, such as the  $\max(0, x)$  thresholding at zero (Relu, Softmax etc.). This leaves the size of the volume unchanged ([32x32x12]).

An example demonstration on how CNN processes images is depicted in below screen shot (Source: <http://cs231n.github.io/convolutional-networks/>):



The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full [web-based demo](#) is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.

Usually it takes a large data set for training purpose, anpr\_ocr dataset is having more than 10k images which should be enough for it. Following parameters can be tuned to improve performance or learning of CNN:

- Training Parameters:
  - Conv\_filters – number of CNN filters.
  - Kernel\_size – size of kernel window.
  - Batch\_size- number of images to take at once during single training step
  - Rnn\_size- size of RNN.
  - Epochs- training frequency
  - Momentum (previous learning step while calculating next one)
  - Weight Decay (prevents model to be dominated by few heavy weight neurons.)
- Neural Network architecture
  - Number of layers
  - Layer Types (Convolutional, max pooling)
  - Activation function (relu, SoftMax etc)
  - GRU layers
  - Dense layer size

## Benchmark

Benchmark model for this project can be the extraction capability of existing OCR ECM solutions such as Captiva, IBM DataCap. These are full-fledged system and do more than extraction of data. In practical these exiting solutions have few issues such as slowness, data extraction accuracy etc. For this project we will be using deep learning to improve extraction process i.e. good accuracy in reading data.

IBM DataCap and Captiva are licensed products, so it is not possible to provide performance of these systems with this dataset. However, I tried it with a simple one-layer CNN, below is the result:

Predicted: A006CA30

True: A065CA30

Input img



Predicted: A1HB61  
True: A128HB61



As evident from above two images accuracy is near about 60 %. This can be considered as baseline score for dataset.

### III. Methodology

---

#### Data Preprocessing

The preprocessing of data is done in "build\_data" function, it consists of following steps:

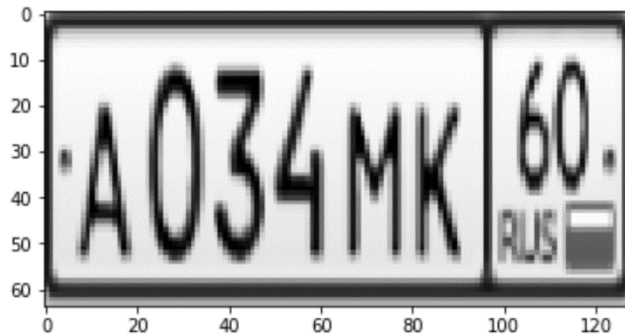
- 1) Read image from the file path and convert BGR image to grayscale.
- 2) Resize this image to standard height and width provided as input to maintain uniformity.
- 3) Convert this image into float format and then divide each pixel in image by 255.
- 4) Assign the above image representation to numpy array created at the start of function.

Below is an example image and important features printed after image preprocessing is done:



Text generator output (data which will be fed into the neural network):

1) the\_input (image)



2) the\_labels (plate number): A034MK60 is encoded as [10, 0, 3, 4, 16, 15, 6, 0]

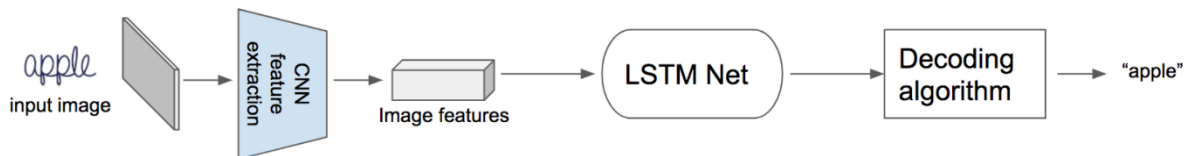
3) input\_length (width of image that is fed to the loss function):  $30 == 128 / 4 - 2$

4) label\_length (length of plate number): 8

Here image is converted into grey scale format and the encoded array is printed as well. Model will take these details for training purpose.

## Implementation

Below is the high-level implementation diagram:



1) Image is fed into CNN to extract features.

2) These features are applied to Recurrent Neural network.

3) Finally, there is a decoding algorithm takes LSTM output from each step and produce final text.

Below diagram depicts our CNN architecture:



|                             |                  |       |                            |
|-----------------------------|------------------|-------|----------------------------|
| concatenate_1 (Concatenate) | (None, 32, 1024) | 0     | gru2[0][0]<br>gru2_b[0][0] |
| dense2 (Dense)              | (None, 32, 23)   | 23575 | concatenate_1[0][0]        |
| softmax (Activation)        | (None, 32, 23)   | 0     | dense2[0][0]               |
| =====                       |                  |       |                            |
| Total params: 4,857,319     |                  |       |                            |
| Trainable params: 4,857,319 |                  |       |                            |
| Non-trainable params: 0     |                  |       |                            |

This is a bit complex neural network and it took a long time for training. However once trained it produced excellent result.

Initially I started with a single CNN which is usually a test to see how our model is working. The initial result achieved were not up to the mark as expected and accuracy was hovering around 40%. Later I realized that my problem statement is a bit different from the usual image identify task, it required system to identify each letter and number separately.

Hence, I researched on this topic and came across concept of GRU or LSTM which I was not aware of earlier. Below is the article which explains this concept very nicely.

<https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>

After my research was done, I decided to put one more CNN layer and add a GRU unit. This did improve results quite a bit, but I was looking for more improvement in model. In between I also played with learning rate, decay and momentum rate which did not impacted much. Finally, I ended up writing a decoding algorithm which would combine our LSTM output to give a text output which was the desired result. Initially I was training system with 5k images, later did it with complete dataset which contains more then 10k Images. These all factors combined did give the desired result.

## Refinement

As mentioned in proposal document target of this project implementation was to achieve 65 % accuracy i.e. system should read 65% of characters and numbers correctly

from given input image. During the initial phase of implementation system was able to achieve this target. Below steps were taken to improve accuracy:

- 1) Momentum – this is an important factor as it tells system about the previous step learning so that it can build on next step accordingly. Tweaking this value helped in increasing accuracy.
- 2) Weight Decay – this factor helps system to decrease effect of certain heavy weight neurons. This is important factor in preventing over fitting of data which improves accuracy.
- 3) Number of filters – These filters form the core of CNN architecture. Increasing its count increases system complexity but it helps in recognizing image better. There should be a balance maintained so that system is not too complex, and image is also recognized. This also helped in increasing accuracy.
- 4) Kernel Size – Usually kernel size is taken as (2,2), however we experimented by increasing size to (3,3). It also helped in increasing accuracy slightly.
- 5) Activation function – most common activation function used in CNN is relu, we tried using softmax. Though it did not have a significant impact on accuracy, but it was worth giving a shot.

Tweaking all these parameters had a significant effect on accuracy and we were able to attain accuracy up to 90%.

## IV. Results

---

### Model Evaluation and Validation

During evaluation, a validation set was used to evaluate model.

The final architecture and hyperparameters were chosen as they performed best among all the tried parameters. Below is the list of parameters:

- The shape of filters of the convolutional network is  $3 * 3$
- The convolutional network learns 16 filters.
- Dense layer size is 32.
- RNN size is 512.
- Weight decay rate is  $1e-6$

- Momentum is set to .9
- Activation function is softmax.

This model was tested on a validation dataset containing more than 500 different images where it performed well i.e. it correctly read text present in image. Proposal document target was to achieve 65% accuracy which was achieved by model.

This is a robust model as it reads individual characters and predicts the best symbol (character or number). Finally, a decoding algorithm combines all these characters and numbers, removes any extra spaces to give complete text of image.

Small changes in input space should not affect our output to large extent. To validate this statement, I performed following steps:

- 1) Carefully looking at training dataset json, we have two tags "train" and "val" as shown below:

```
{ "tags": ["val", "train"], "description": "P334EA73", "objects": [], "size": {"height": 34, "width": 152}}
```

```
{ "tags": ["train", "train"], "description": "P343YM70", "objects": [], "size": {"height": 34, "width": 152}}
```

The "val" tag data serve as the validation set.

- 2) I manually changed the "val" tag to "train" tag and vice versa for few images (It's similar to k-fold technique, here I am doing it manually due to nature of our dataset).
- 3) Repeated the process in point 2 couple of times and tested the output which was inline to the output shared in conclusion.

Hence until there is a major change in dataset our results should not differ much.

## Justification

I tested solution using image dataset containing more than 500 images. This model was able to read text from images accurately which was the primary goal of this project i.e. to improve accuracy.

Other benchmark solutions mentioned in above section are full fledged products having one of the capabilities as OCR. I have personally worked on these products and faced issue of data accuracy even after system was trained multiple times with quality documents. This caused a lot of manual intervention and the whole idea of automating extraction process to improve accuracy and reduce cost went into jeopardy. Through

this project I aim to increase accuracy of extracted data so that organization is confident of removing manual processes. I have initially set up benchmark of 65% accuracy, however through hyper parameters tuning I was able to achieve 90 % accuracy of data.

With all above said, in actual production system there are number of factors which effect accuracy of extracted data. Few of them are quality of scanned documents, training data set etc.

## V. Conclusion

---

### Free-Form Visualization

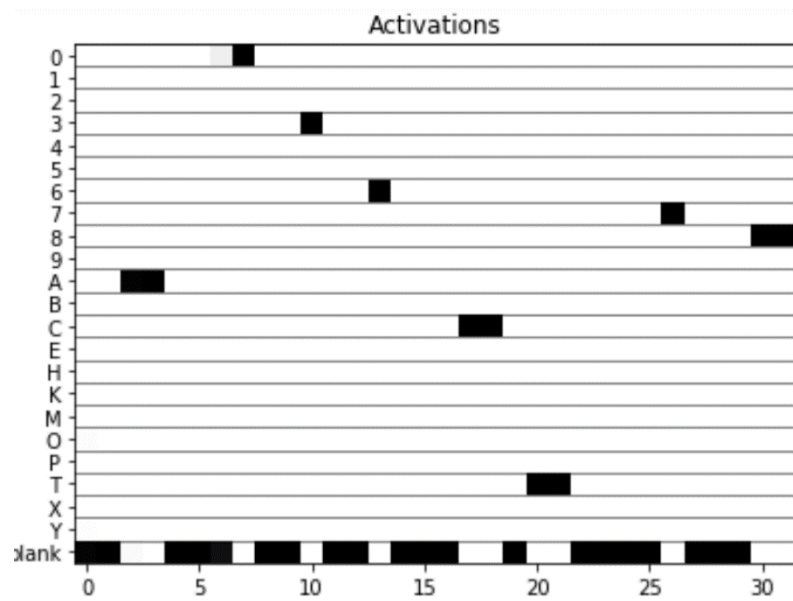
Below are few example texts predicted by our model:

---

Predicted: A036CT78  
True: A036CT78

Input img

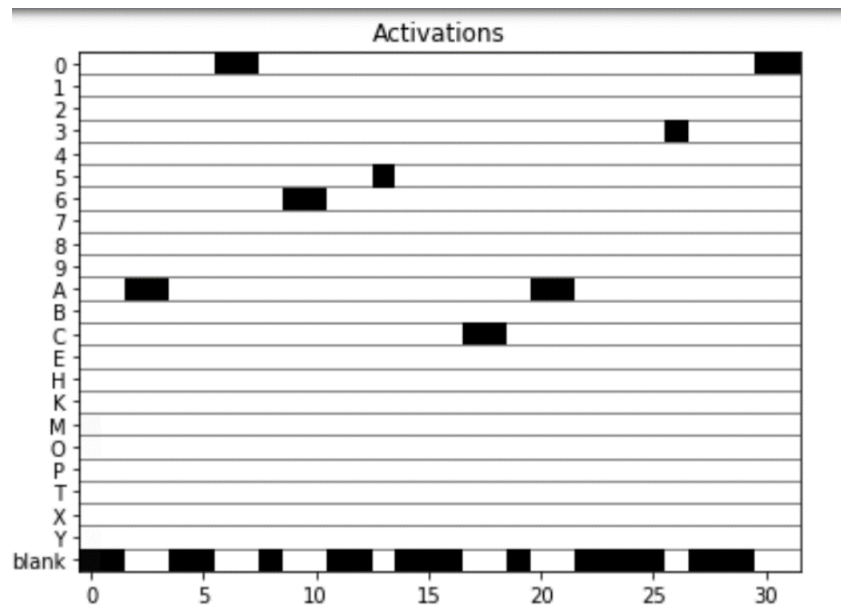




Predicted: A065CA30

True: A065CA30





As evident from above figures system has identified all the letters and numbers in text correctly which was our core problem statement i.e. extracted data accuracy should be high.

## Reflection

The process used for this project can be summarized using following steps:

- Initial problem statement is formulated, and relevant public data set was found.
- Preprocessing checks were done on this dataset i.e. enough data set is present in training and validating our network, testing data set is present.
- Benchmark products were identified.
- Image data is preprocessed as described in above mentioned section.
- Build CNN architecture and train it.
- Model was tested using the testing data set to check accuracy of extracted data.

I found step 4 & 5 as most difficult step due to following reasons:

- 1) Preprocessing data – this is very important and challenging aspect as we need to identify correct technique to preprocess data so that our CNN network can be trained in correct parameter which directly impacts accuracy.



- 2) Build CNN architecture – there are no pre-set benchmarks for a CNN which will give good performance. There were lot of architectures and hyperparameters tried to arrive at conclusion. Though it is a bit cumbersome task, but ultimately it is this final architecture which helped in achieving our desired result.

## **Improvement**

There are few improvement areas as discussed below:

- 1) Accuracy should be checked when our input image is not clear i.e. how accurate our results are with blur image.
- 2) There are few mobile applications which are extracting data from pre-filled forms for further processing, for ex – there are applications which extract data from W2 form to file tax return but there performance is extremely poor i.e. user ends up entering most of the data manually. Can this model be extended to a mobile app which can extract data accurately there as well?