Real-Time Multi-Digit Recognition and Localization

Ronny Restrepo

10 February 2016

Abstract

This paper will look at the problem of recognizing numbers from photos and video captured of the built up world. This is a significantly more dificult problem than Optical Character Recognition (OCR) of scanned documents due to the complexity of such imagery. A deep-neural network is trained to perform the end-to-end task of simultaneously recognizing the digits of a number as well as localizing the positions of these digits. The model is trained on the Street View House Numbers (SVHN) dataset, achieving an accuracy of aproximately 95.22% on recognition of entire digit sequences, and 98.63% accuracy on individual digits. This trained model is then used in an application that recognizes and localizes numbers of a video feed from a webcam that is capable of running in real-time.

1 The problem

Recognizing text and digits in scanned documents (Optical Character Recognition - *OCR*) is a well understood problem that has resulted in many commercial software packages for general use. Recognizing text and digits in natural scenes, e.g. on the sides of walls, billboards, or other objects is a much trickier task that remained unsolved until around 2011 (Netzer et al. 2011). OCR on scanned documents has the luxury of a perfectly well illuminated image with high contrast between the characters and the background, each of which is a flat color. It also has the luxury of having all digits perfectly alligned with a consitent size. Recognizing text in natural images, such as photos, however lacks all these luxuries. There is not always a high contrast between the pixels of the image that represent a character from those that represent the background. The regions representing the characters and the background will have texture and patterns, rather than being a flat color. Characters will not be prefectly alligned and uniformly sized. This might be because of stylistic choices made by those putting up the text or because of optical distoritions in the image. Such distortions may be because of the surface the characters are on, the angle the photo was taken from causing perspective transformations, or the camera itself causing lens distortion. Moreover, there will be many different lighting conditions, shadows, obstructions, and diferent levels of camera focus.

Traditionally the strategy for recognizing multiple text characters was to to separate the task into multiple systems. One system detects the individual digit positions. This information is then passed on to another system that looks at each of these positions one at a time to recognize the individual digits. Goodfellow et al. (2013) demonstrated that for tasks involving just a small number of characters, a single end-to-end deep-learning system could be created that recognizes *all* of the digits simultaneously.

This paper will take the same approach as Goodfellow et al. (2013), and extend it to simultaneously recognize not just the digits, but also the bounding boxes of those digits. This will be used in a real-time recognition and localization application that makes use of a webcam video feed.

2 The Data

The data used for training the model is the Street View House Numbers (SVHN) format 1 dataset (Netzer et al. 2011), which can be found on this webpage http://ufldl.stanford.edu/housenumbers. This data consists of 3 separate datasets that are composed of images containing house numbers taken from Google Street View, along with labels specifying the digits for each number, and the bounding box positions for each digit. The three datasets are:

- **Training dataset:** containing 33,402 images of what are considered to be dificult images to train on.
- **Extra dataset:** containing 202,353 images that are to be used as aditional (but simpler) images to train on.
- **Test dataset:** containing 13,068 images for evaluating the final trained model.

A sample of the types of images in the data can be seen in figure 1. The house numbers range from 1 to 6 digits in length in the quantities shown in figure 2. As can be seen, there is a very uneven distribution of samples for each digit length, with several orders of magnitude difference between some of them. It is clear that there is just not enough six digit samples for them to be of any use for training. There is a small quantity of five digit numbers, but they might be of some use.

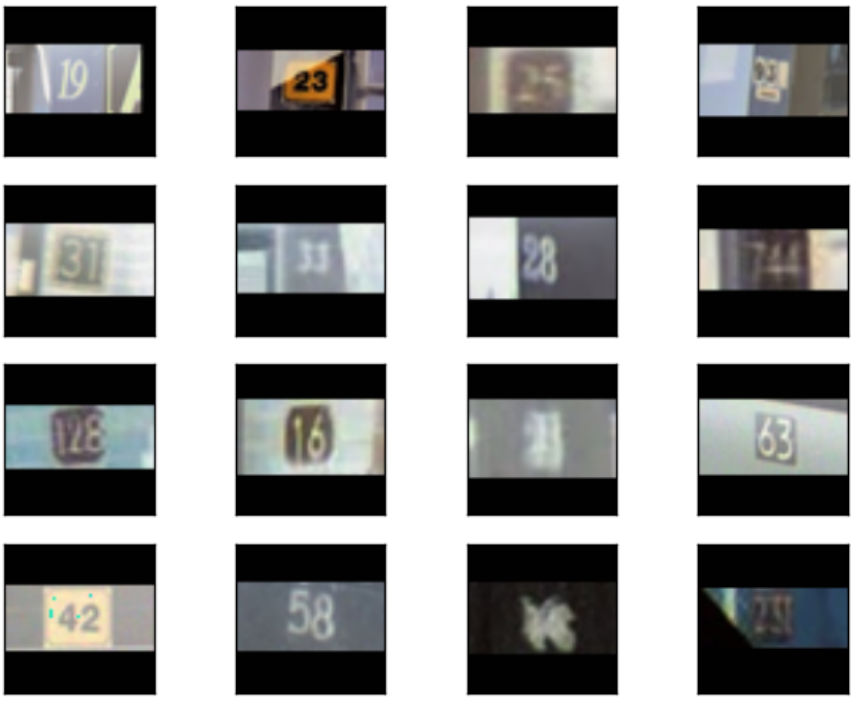


Figure 1: A small sample of images from the training dataset.

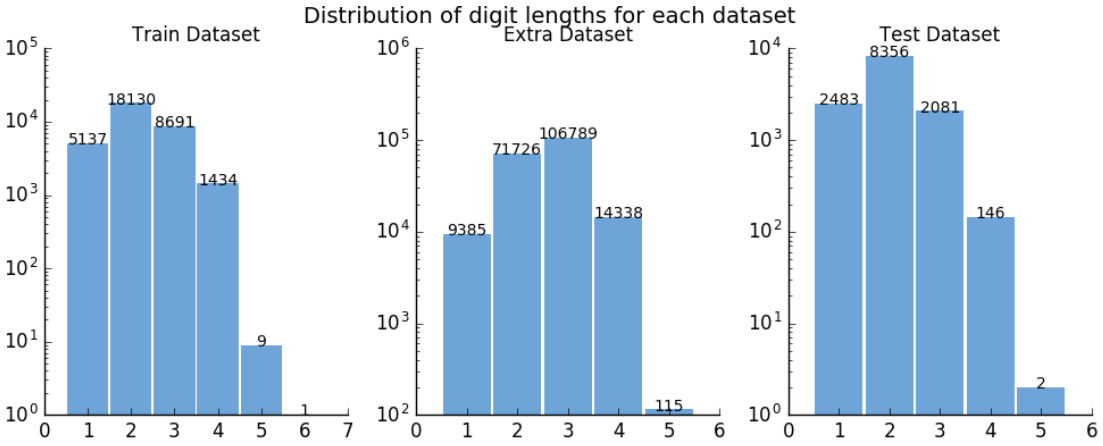


Figure 2: Distributions of digit lengths displayed in log scale.

3 Data Processing

The SVHN dataset contains images of many different sizes and aspect ratios. In order to make use of them in a neural network they had to be turned into a consistent size. The SVHN data contains bounding box information for each individual digit in the number. This information was used to calculate the bounding box for the entire number. This bounding box was then used to create crops of the images cetered in the region containing the number. As per Goodfellow et al. (2013), rather than cropping precisely to the region specified by the bounding box, this rectangle was expanded by 30%, and that expanded rectangle was used as the region for cropping the image. The cropped image was then resized and reshaped into 64×64 square images (without preserving aspect ratio). This process is illustrated figure 3.

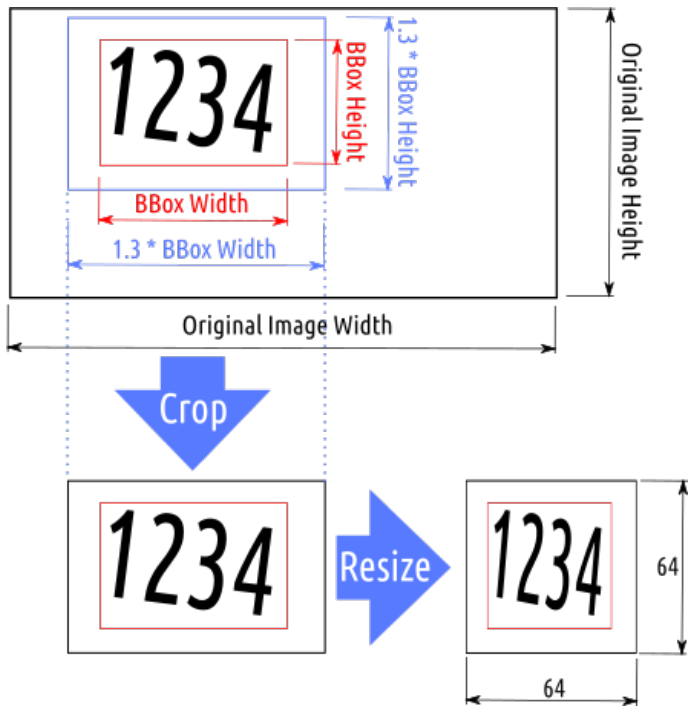


Figure 3: The process of cropping and resizing the images using bounding box information.

The bounding box information was also updated to account for the crop and resize. Aditionally, the bounding box coordinates were converted from absolute pixel position values to a ratio of the width and height of the image, such that all values would be within the range $[0, 1]$. Bounding boxes for null digits were placed outside of the limits of the image, at coordinates [0.4, 1.3, 0.6, 1.5], as is shown in figure 4.

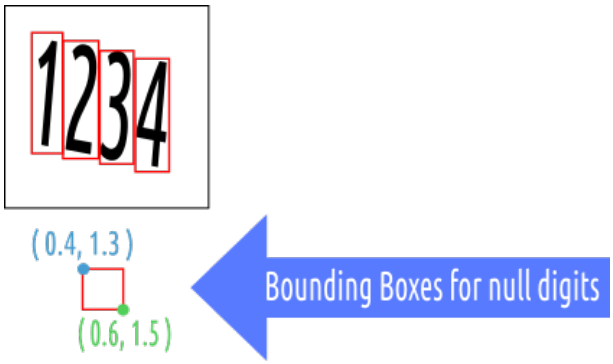


Figure 4: Bounding box labels for null digits.

The images were converted to greyscale since color was not considered to be an important feature in the classification of digits. There is for example no reason to think that a greater proportion of blue makes the digit 2 more likely than the digit 5. Having color might cause the model to concentrate on the wrong features.

The data was restricted to only samples containing 5 or less digits. Since the number of five digit numbers in the training data was quite small it was quite important to make sure that there were no mislabelled samples. As it turns out, there were two incorrectly labelled samples (which constitutes 22% of the samples labelled as 5 digit numbers). The two incorrect labels can be as can be seen in figure 5, which shows all nine samples from the training dataset that were originally labelled as having 5 digits.

These incorrect labels were manually changed:

- **03163:** changed to 31637
- **22827:** changed to 2827

The original digit labels contained lists of varying lengths, depending on the number of digits in that sample. The values ranged from 1-10, with a 10 representing the number zero. These labels were converted to a consistently sized array with 5 digit positions. Each digit postition was modified to take on a value between 0-9 to represent the corresponding digit. An aditional *null* value was introduced (represented by a '10') for digit positions that contain no digit. The following table illustrates how numbers are represented using this method.

Number	Representation in array
3	[10, 10, 10, 10, 3]
507	[10, 10, 5, 0, 7]
10768	[1, 0, 7, 6, 8]

Throughout this paper, digit positions will be refered to in the following way: digit 1 will refer to the digit on the left most position of the digits array and digit 5 will refer to the right most digit of the digits array.

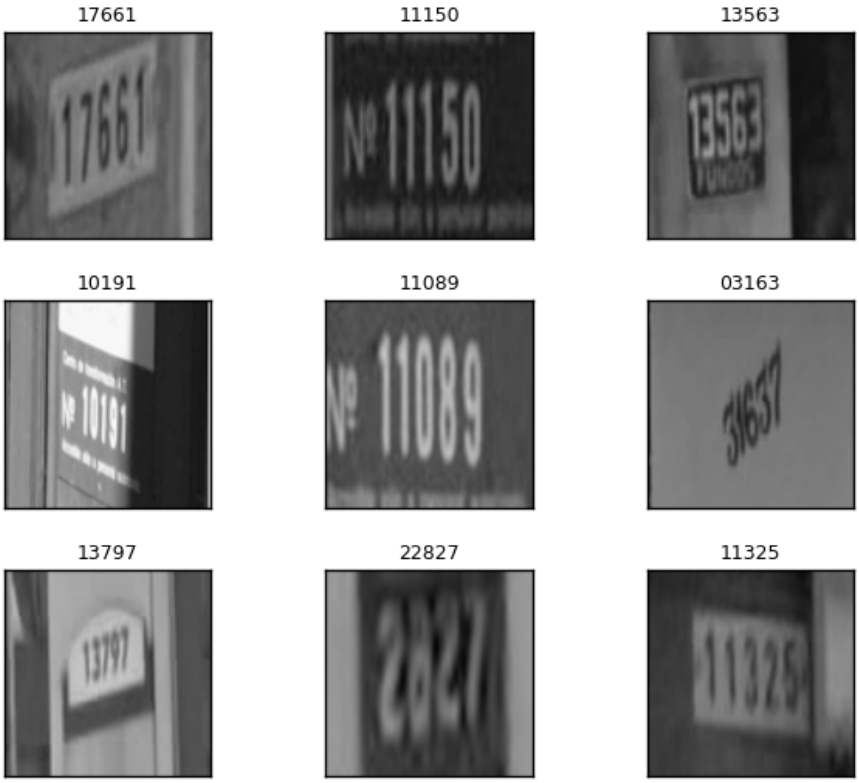


Figure 5: All samples labelled as a five digit number in the train dataset

The SVHN "training" and "extra" datasets were merged into one large dataset of 235,754 samples to be used for training the models. Due to the great imbalance of digit lengths, especially for five digit numbers, the data was increased so that there would be at least 5000 samples of images for each digit length. This was done by randomly duplicating other samples of data from under-represented digit lengths. There was no need to perform any image transformations on those duplicates to make them appear as different images since this would be done dynamically for every batch of images during the training phase, as will be discussed in the data augmentation section. There is certainly the danger that for five digit numbers, the model would be overfit to the specific numbers in the training data as a result of this duplication process. But this is perhaps preferable to the posibility of five digit numbers being ignored altogether by the model.

Taking a look at the data that will be used for training, we can see that there is going to be some weaknesses in any model that we train. The plot in figure 6 shows the distribution of digits for each of the five digit pisitions. We can see that it is going to struggle to learn the first digit for 5 digit numbers, but especially for any number that does not begin with a 1,2 or 3. It will completely fail to learn to recognize any 5 digit numbers starting with 8 or 9. For the second position it is going to struggle most learning to recognize 0,8 and 9. For all other digit positions, there is a sufficiently large number of samples for each digit.

The test dataset was processed in a similar way for cropping of the images, processing the bounding boxes, and digit labels, but there was no duplication of any of the under-represented samples.

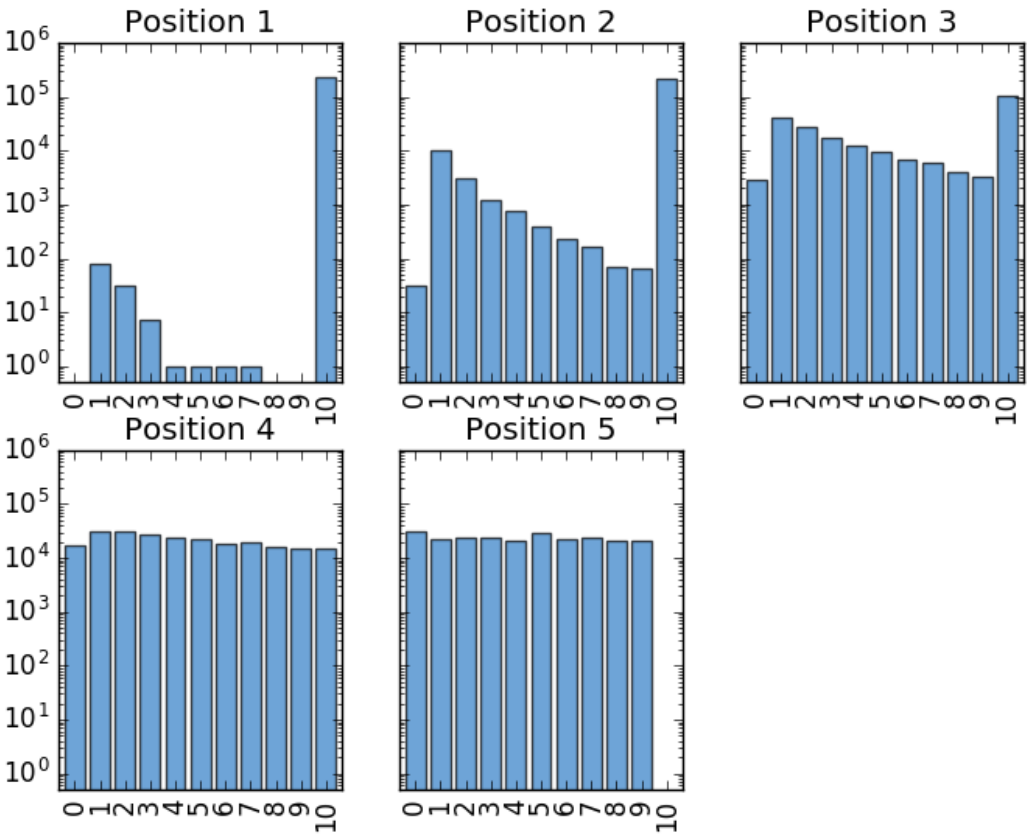


Figure 6: Distribution of digits at each digit position

3.1 Data Augmentation

Data augmentation is the process of making random variations of the data in order to simulate a larger dataset. For the current task of digit classification of images, this is quite simple. There are many trasformations which can be performed that do not change the associated label. Examples of such transformations include changes in brightness, contrast, noise, blur, etc. Since the task involved here is to also predict the position of the digit bounding boxes, any spatial deformation would also have an effect on the labels. But the changes to the labels are quite predictable. So transformations like cropping and shifting in space can be done, as long as the corresponging bounding box data is also updated in the appropriate manner.

Data augmentation was performed dynamically on each batch of data during the training process. The following transformations were made on each image in the batch:

- **Crop:**
  - For each axis of the image, a random value within the range $[54, 64]$ was taken. These values specify the size of the crop to be taken from the 64x64 images.
  - for each axis of the image, a random value was chosen as the spatial offset of the crop, based on how much room for movement there is along each axis, based on the selected crop size.
  - The output of the crop was resized to a 54x54 image (without preserving aspect ratio).

– The labels for the bounding boxes were updated to account for this shift.

- **Brightness:** random amount of brightness and contrast to simuate different lighting conditions.
- **Blur:** random amount of gaussian blur to simulate differences in camera focus.
- **Noise:** random amount of gaussian noise to simulate sensor noise in cameras.

4 Architecture

Goodfellow et al. (2013) came up with a model that correctly identified the entire digit sequence of numbers from the SVHN dataset with 96.03% accuracy. This architecture is used as inspiration for the models that will be tested in this paper, all of which are slight variants of each other.

Each model tested used the overall architecture illustrated in figure 7. Images were fed into a trunk, which consisted of multiple modules that will be discussed further below. The output of the trunk then gets branched out into five separate softmax classifiers (one for each digit), plus another branch that performs regression on 24 values. These 24 values represent the coordinates for all the bounding boxes of each individual digit ($5 \times 4 = 20$) plus the bounding box for then entire number.

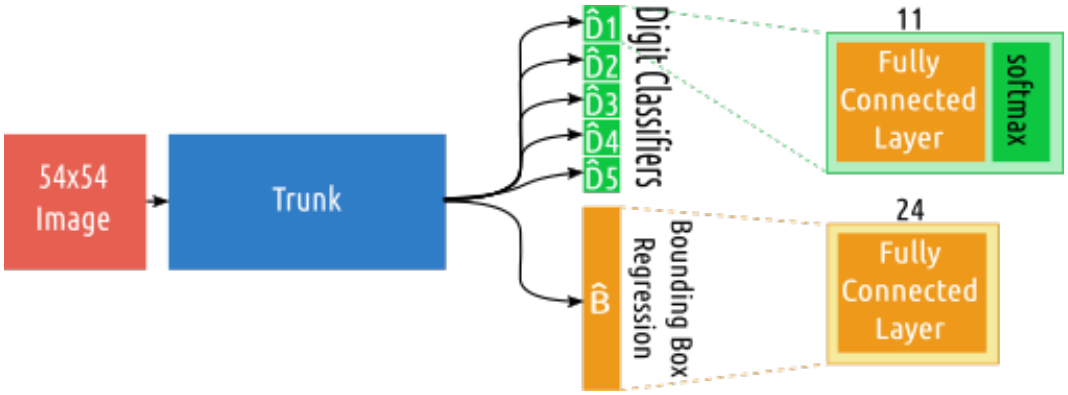


Figure 7: Image of overall architecture

The trunk is composed of a stack of two types of modules, which will be refered to as a *conv battery* and an *FC battery*. These two modules are illustrated in figure 8. The conv battery starts off with a convolutional layer (Fukushima 1988; LeCun 1989; LeCun et al. 1998), with a fixed stride **s** of 1 along each axis. The kernel size **k** and the number of output filter layers **n** can vary. Batch normalization (Ioffe and Szegedy 2015) is then applied, followed by dropout (Srivastava et al. 2014). Batch normalization can be thought of as a form of regularization, which makes the need for dropout less necessary. As such a small dropout rate of 0.1 is used. This is then followed by a max pooling operation (Zhou and Chellappa 1988), where a the kernel size **k** and the stride **s** can vary from layer to layer. Finally, this is fed through to a Leaky Rectified Linear Unit (Leaky ReLU)(Maas, Hannun, and Ng 2013), with a leakiness rate of 0.01. No bias value is used for the convolutional layer since the learned offset in batch normalization fulfills the same role. The *FC battery* is very similar, except that it starts with a fully connected layer, and does not contain a max pooling operation.

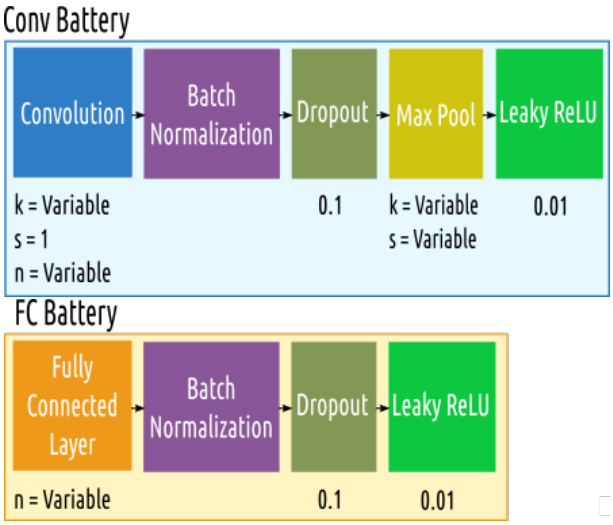


Figure 8: Components of **Conv Battery** and **FC Battery** modules

The specific arrangement of these two modules within the trunk are where the various models differ. Figure 9 illustrates how they are arranged.

All the weights of the network are initialized to He et al. (2015) weights.

5 Training and Evaluation

Since the models perform two tasks simultaneously, digit classification as well as regression on bounding box coordinates, there are two different kinds of loss functions that need to be calculated. The loss for the bounding box task is calculated using the root mean square error (where the mean is defined as the average over not just all samples in the batch, but all 24 of the bounding box coordinates as well). The formula used is as follows:

$$J_{bbox} = \sqrt{\frac{(\hat{B} - B)^2}{24N}}$$

Where:

- $\hat{B}$ = the predicted bounding box coordinates
- B = the correct bounding box coordinates
- N = the number of samples in the batch

The loss for the digit classification task is calculated using the negative log likelihood of each of the separate digit branches, averaged over the samples in the batch as well as the 5 separate digit predictions. This is shown in the following formula.

$$J_{digits} = -\frac{1}{5N} \sum_{i=1}^{5} log(\hat{D}_i^{(y)})$$

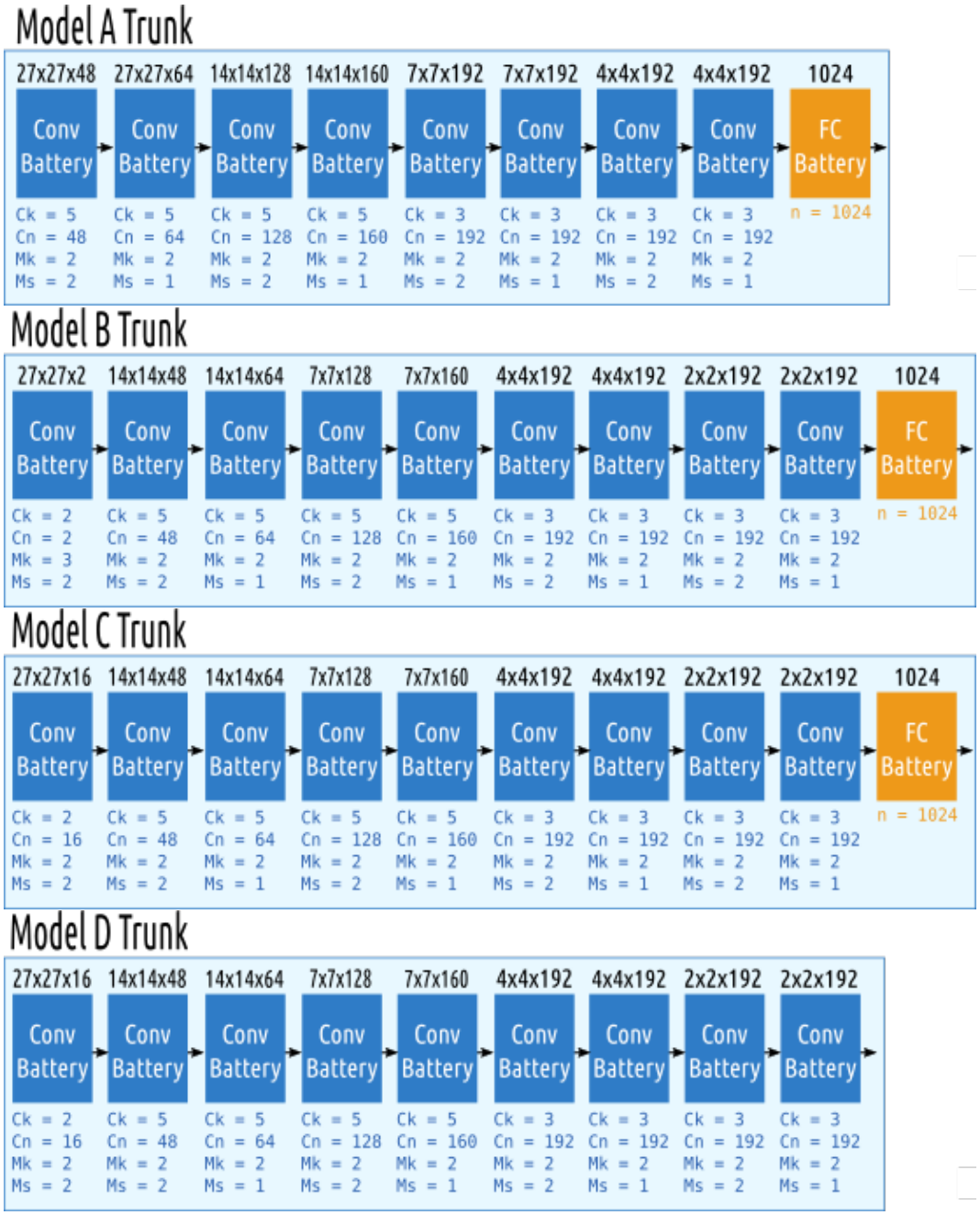


Figure 9: Composition of modules within the trunk of different models. The numbers above each module indicate the dimensions of the output tensor (ignoring batch size). The values underneath each module are explained in this table:

Key	What it represents
Ck	Kernel size for colvolution operation (square kernel)
Cn	Number of output filters for convolution operation
Mk	Kernel size for max pool operation (square kernel)
Ms	Stride for max pool operation (same along each 2D axis)
n	Number of output nodes (width) of fully connected layer

where:

- $\hat{D}_i^{(y)}$ = the predicted probability of the correct label for digit i.
- N = the number of samples in the batch

These two losses are then summed up to create a total loss that is used for training the model.

$$J_{total} = J_{bbox} + J_{digits}$$

This loss is then fed into an Adam optimizer (Kingma and Ba 2014) with the following parameter settings:

- $\beta_1 = 0.9$
- $\beta_2 = 0.999$

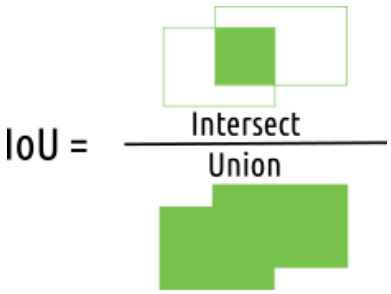
The learning rate was decayed according to the following schedule.

- 10 epochs @ learning rate of 0.001
- 5 epochs @ learning rate of 0.00075
- 5 epochs @ learning rate of 0.0005
- 5 epochs @ learning rate of 0.00025
- 5 epochs @ learning rate of 0.00001

This schedule was chosen on the basis that during preliminary tests, the the rate at which a model made improvements was flattening out after just a few epochs.

At the end of each epoch, several evaluation metrics were taken.

- **Per Digit Accuracy (PDA):** The proportion of individual digits that were correctly predicted.
- **Whole Number Accuracy (WNA):**The proportion of samples where every single digit was correctly predicted for the whole number, including the prediction that certain digit positions contain no number. This is the metric that will be used to compare the performace of different models.
- **Intersect of Union (IoU):** IoU is used to evaluate how good the predicted bounding boxes are compared to the bounding box labels. It is a measure of the ratio of how much the two boxes overlap each other, and is calculated using the following formula.



The values of the weights are saved as a checkpoint file at the end of each epoch. A second checkpoint file - called the *max checkpoint* - is saved if the current epoch is the best performing epoch. This is done in case the model becomes worse after subsequent training steps, and allows us to make use of the best version of the model as the final model.

The model was implemented using Tensorflow v0.12 (Abadi et al. 2015).

6 Results

The following table shows the performance of each model at the point when the *max checkpoint* was taken.

Model	IoU	PDA	WNA
A	0.819	99.238	97.070
B	0.832	99.004	95.996
C	0.829	99.102	96.484
D	0.834	99.062	96.289

We can see that the best performing model (model A) correctly predicts 97.07% of the entire numbers from the validation data. It also has the highest accuracy on individual digits with 99.24%. But it does happen to have the worst performance on the bounding box task. However, it is still a very good score and is only marginally worse than the other models. It is preferable to have a model that makes better predictions on numbers than it is to have a model that makes slightly neater bounding boxes. As such, it is model A that will be used for all subsequent discussions for the remainder of the paper.

Taking a look at the training curves for model A (figure 10), we can see that it reaches its best performance on validation data on the 25th epoch (using zero indexing for epochs). The amount of learning flattens out at about 15 epochs, which coincides with the transition from 0.00075 to 0.0005 learning rate. It is possible that this model could have reached higher performance if it had been allowed to train for longer on the 0.001 and 0.00075 learning rates.

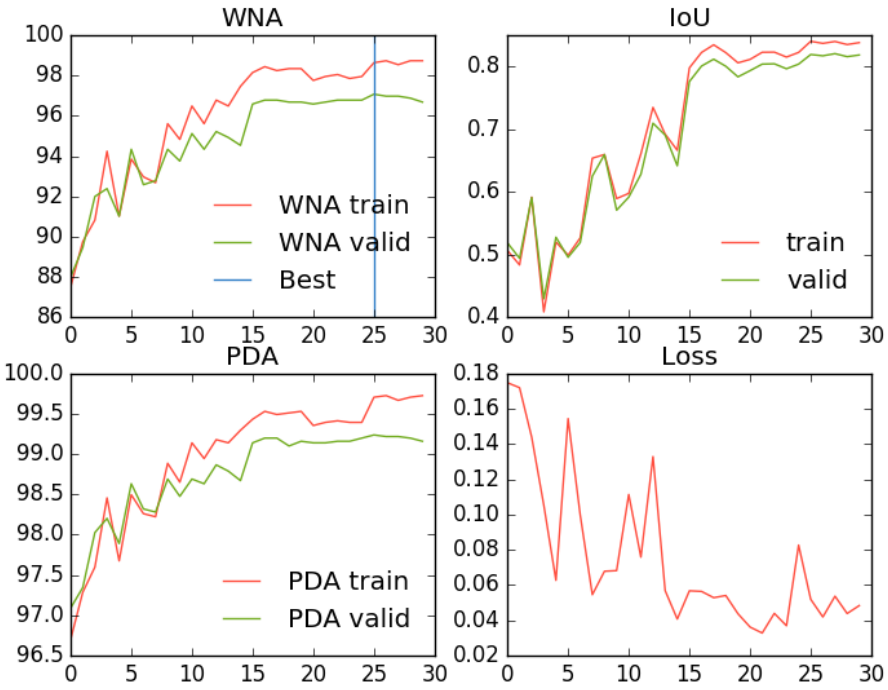


Figure 10: Learning curves for model A evaluated at each epoch of training

Evaluating this model on the test data gives the following results.

Metric	Score
IoU	0.828
PDA	98.633
WNA	95.215

We can see that it correctly predicts 95.22% of the entire numbers from the test set. This certainly falls short of the benchmark set by Goodfellow et al. (2013), but it is never the less quite a good result.

The image in figure 11 shows a sample of predictions made by the model on the test data. We can see that most of the samples are doing a very good job at both predicting the digits and the bounding boxes. There is of course one sample on the lower left hand corner which performs very poorly on the bounding boxes task. Figure 12 takes a closer look at the samples with the lowest IoU scores. We can see that these low scores can be attributed to several different factors:

1. **Incorrectly labelled data:** There are a few samples where the ground-truth bounding box labels are actually incorrect to begin with. For example, the ground-truth labels for the two center images in the top row are missing a digit (and corresponding bounding box data). The trained model on the other hand correctly predicts that there is a digit in that position.
2. **Poorly labelled data:** There are samples where the ground-truth bounding box labels are much bigger than they really should be, capturing more than just the actual digit. In some of these cases the predicted bounding box produced by the model gives a much tighter fit around the actual digit. This unfortunately results in a lower IoU score.
3. **Vertical stacking:** The trained model has great dificulty with images where the

Figure 11: Sample of predictions on the test dataset. The predicted bounding boxes are in red, and ground-truth bounding boxes are in green. The first line of text above each image is the IoU score for the predicted bounding boxes. The second line contains the predicted number, along with the ground-truth label in brackets.

numbers are vertically alligned, as in the first image of the first row. This could be due to a lack of training samples with such allignment of digits.

4. **Legitimately dificult samples:** There are some images where it is very dificult to read the number even for humans, either because it is too blury, or because it is dificult to determine if something is a real digit or just a shaddow or other obstruction.

5. **Round frame:** There are two images on the far right column that are clearly a 7, but have a sharp round frame around them. In one case it is interpreting this round frame as a zero, so it is possible that such round frames with numbers inside them also confuse the bounding box branch of the model.

6. **Genuinely bad predictions:** There are several images which seem prefectly clear, but which never the less produce bad predictions. For example, there are three images of digit numbers ending in a 5 that for some reason produce very bad bounding boxes.

Figure 13 shows the samples that it has dificulty on the digit classification task. The reasons for why the model fails on this task for these samples fall into the same sort of categories as mentioned above for the bounding boxes task.

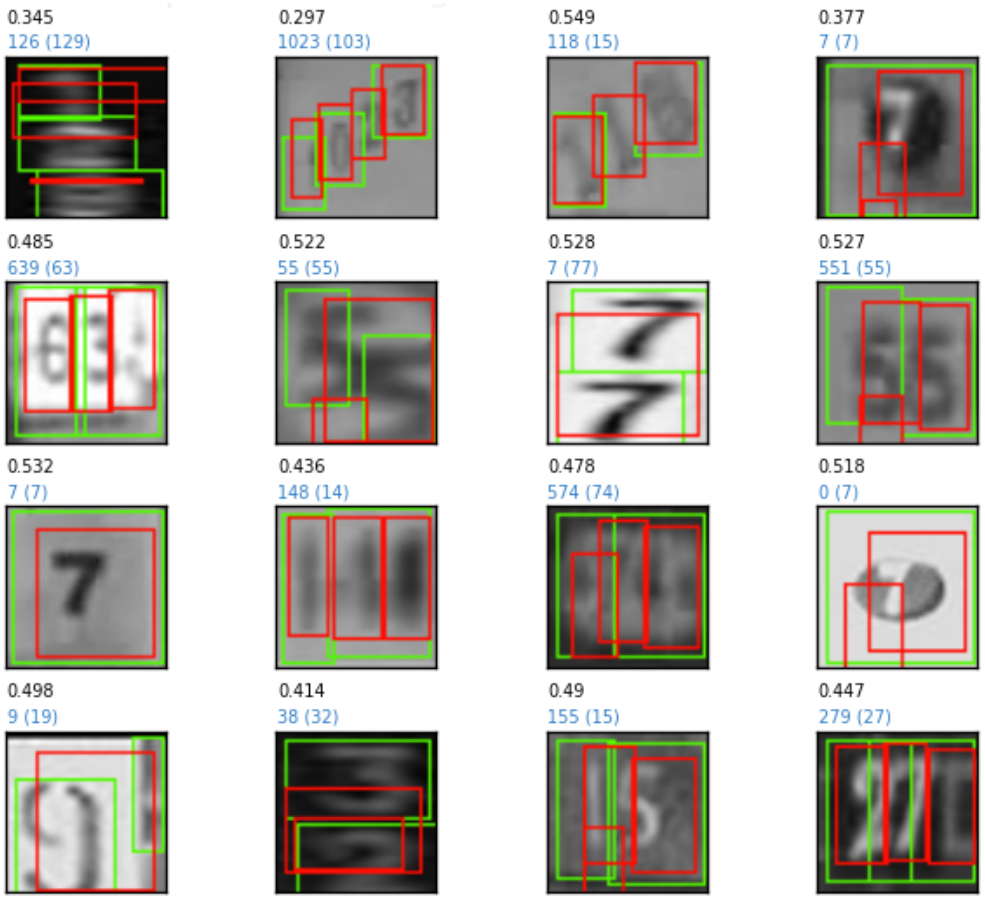


Figure 12: Sample of predicted bounding boxes with an $IoU < 0.55$ on test dataset. These are the images the trained model had most dificulty with

7 Real-time multi-digit recognition app

The trained model was used in a desktop application that makes use of a webcam to perform multi-digit recognition plus localization of the digits when there is a number in the scene. It is capable of real-time performance on a quad-core 2.20GHz laptop, running entirely on CPU (no GPU processing was used for this app). A demo of this app can be seen in this youtube video https://youtu.be/lL5HrBRdnZg.

The video shows the app performing quite well on printed images of photos of house numbers found on various photo sharing websites, but also on hand-drawn numbers. However it does have a few limitations. Firstly, since the model was trained on images that *always* contained a number, the model *always* tries to predict some number in the scene, even when there is no number. It tries to find patterns where there are none.

Secondly, since the numbers in the training data were all close up, the model only performs well when the numbers take up a large portion of the input image. If the numbers are a bit further away, it makes completely incorrect predictions, it does not recognize the different digits as separate digits.

Thirdly, since the training data contained all numbers where all the digits were upright, it makes very poor predictions if the numbers coming in are rotated by too much of an angle. This could be improved by performing an aditional data augmentation step of randomly rotating the images. however this was avoided in the current project due to the

Figure 13: Sample of incorrectly predicted numbers on test dataset

aditional complexity of re-calculating the bounding box information to account for these transformations.

Finally, as predicted in the data processing section, the model struggles the most with 5 digit numbers. If presented with a five digit number, it either does not acknowledge the existence of the first digit, or tries to label it as a 1 or 2. This is due to the lack of five digit numbers in the dataset, and in particular, five digit numbers that start with anything other than a 1,2 or 3. It also struggles demonstrates unecertainty for digits in the second digit position, quite notably when presented with a 9, where it constantly alternates between predicting it as a 9,4,1,8 or 6.

References

Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. URL: http://tensorflow.org/.

Fukushima, Kunihiko (1988). "Neocognitron: A hierarchical neural network capable of visual pattern recognition". In: *Neural networks* 1.2, pp. 119–130.

Goodfellow, Ian J. et al. (2013). "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". In: *arXiv:1312.6082 [cs]*. arXiv: 1312.6082. URL: http://arxiv.org/abs/1312.6082.

He, Kaiming et al. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *arXiv:1502.01852 [cs]*. arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]*. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]*. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

LeCun, Yann (1989). "Generalization and network design strategies". In: *Connectionism in perspective*, pp. 143–155.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng (2013). "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. ICML*. Vol. 30.

Netzer, Yuval et al. (2011). "Reading digits in natural images with unsupervised feature learning". In:

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Zhou, Y. T. and R. Chellappa (1988). "Computation of optical flow using a neural network". In: *IEEE International Conference on Neural Networks*. Vol. 1998, pp. 71–78.