House number detection and identification

Introduction

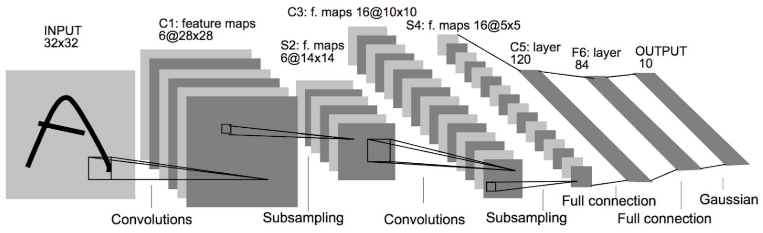
Character recognition has been an important task among researchers for a long time now. Various implementations of recognition of typed or even hand-written texts can be found nowadays in numerous OCR softwares and devices. But texts used on signs, buildings, billboards, etc. pose much greater challenge as they are located among numerous other objects, which could confuse even the latest state-of-the-art algorithm. Nevertheless number of applications for such text recognition increases every day: information gathering, robotics, automatic translators, augmented reality, etc.

In this project I'll pick on one of the subtypes of text recognition - House Numbers recognition, using Street View House Numbers database [1] and one of the most popular techniques to process images for recognition and classification purposes is Convolution Neural Networks (CNNs).

Previous work

First neural networks adapted for image classification used flattened image vectors as inputs where each pixel channel value is its own independent feature that is processed by several densely interconnected layers of neurons. Such approach works relatively well on well-prepared, nicely separated, centered characters, like a popular MNIST database [3]. But results usually drastically decrease when neural network receives rough input, where image might contain rotated, moved objects or parts of other objects.

When CNNs have been first introduced in [2] they immediately become popular in the field of image processing thanks to their adaptability to various tasks and data. Their primary advantage over dense neural networks is processing of the image not as a number of independent features but as a locally dependent ones. In its base CNNs use series of filters of predetermined size that slide over the input image and extract combined information of those image areas based on filter weights that are learned during CNN training.



Furthermore, new convolution layers can be added on top of each other allowing CNN to learn more complicated features based on more simple ones (e.g. first layer detects horizontal and vertical lines when second layer uses those lines to detect squares).

This principle isn't new as convolutions using various kernels have been used in Computer Vision for smoothing, edge/feature detection and other tasks for a long time now. The advantage of CNNs over those methods is that filter doesn't have to be predetermined and can change during training. As the result CNN learn on its own what features (filters) it needs for particular task and how to combine them for accurate classification.

Since first introduction of CNNs there's been many successful implementations of various architectures for wide variety of tasks. One of them is VGG16, CNN architecture that achieved excellent results on ImageNet competition purpose of which was to classify images of 8 different objects.

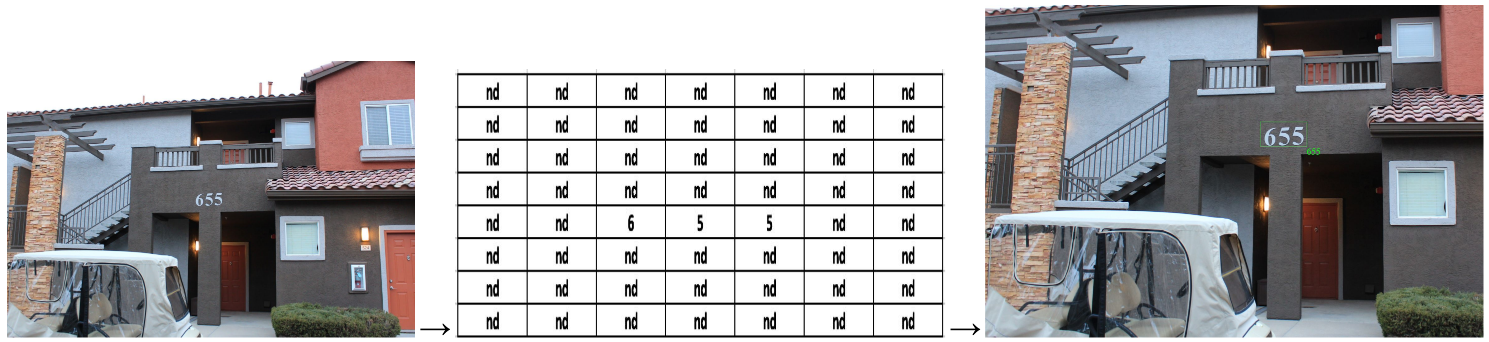
But in this project I'll mostly use my own CNN architecture to classify 11 types of objects on the input images (10 digits plus one separate class for non-digit images).

General approach

Facing task of house number recognition one can take one of the few approaches for this type of problem. First is a straightforward approach to classify image as a whole like an original VGG16 did. But this method is highly inefficient since CNN would have to be able to differentiate between 1000 classes (for up to three digit numbers) and would require enormous amounts of data, training time and big enough architecture.

Far more reasonable approach would be to find house number location on the image using either one of the Computer Vision methods (template matching, find all rectangular objects, etc.) or another Neural Network for this purpose (R-CNN [7]). After that one can cut out number out of the whole picture and using a sliding window over the cutout classify each area of the image as one of the 10 digits.

Another approach that was implemented for this project doesn't require preliminary step of number detection. Instead it uses sliding window over the image cutting image in smaller chunks (similarly how digits are recognised on the number image). Later each chunk is classified as one of the 11 classes: either one of the 10 images or non-image class object. Ideally most of the image parts this way would be classified as that extra non-digit class and only areas with number digits would return correct digit class. In that case recognizing the number would be as easy as to sort the digit areas from left to right and combine their class labels into number. In addition we get coordinates of the areas where recognised digits are and by combining them we can get the whole number area location.



nd	nd	nd	nd	nd	nd	nd
nd	nd	nd	nd	nd	nd	nd
nd	nd	nd	nd	nd	nd	nd
nd	nd	nd	nd	nd	nd	nd
nd	nd	6	5	5	nd	nd
nd	nd	nd	nd	nd	nd	nd
nd	nd	nd	nd	nd	nd	nd
nd	nd	nd	nd	nd	nd	nd

Data preprocessing

Despite the ability of CNN to adapt to many types of data some preprocessing steps would alway be required to help with classification and general network operation.

First of all, one step that is pretty much always required in neural networks is data normalization. This brings all the data to common ground and helps with learning network weights faster and more accurately. This is especially important in images where the same image taken in different lightning conditions can result in pixel values ending up in completely different data range despite representing the same scene.

Secondly all input images have been converted into grayscale leaving 1-channel images. This reduces amount of information 3 times which allows us to train network on much lesser dataset. This step especially makes sense for house numbers as they can be of any color, but color itself isn't needed for number recognition.

Next steps are data specific. For this project I used SVHN dataset with house number extracted from Google street view images [1]. This dataset includes images of various house numbers and each image has a metadata information with all digits location on the image.

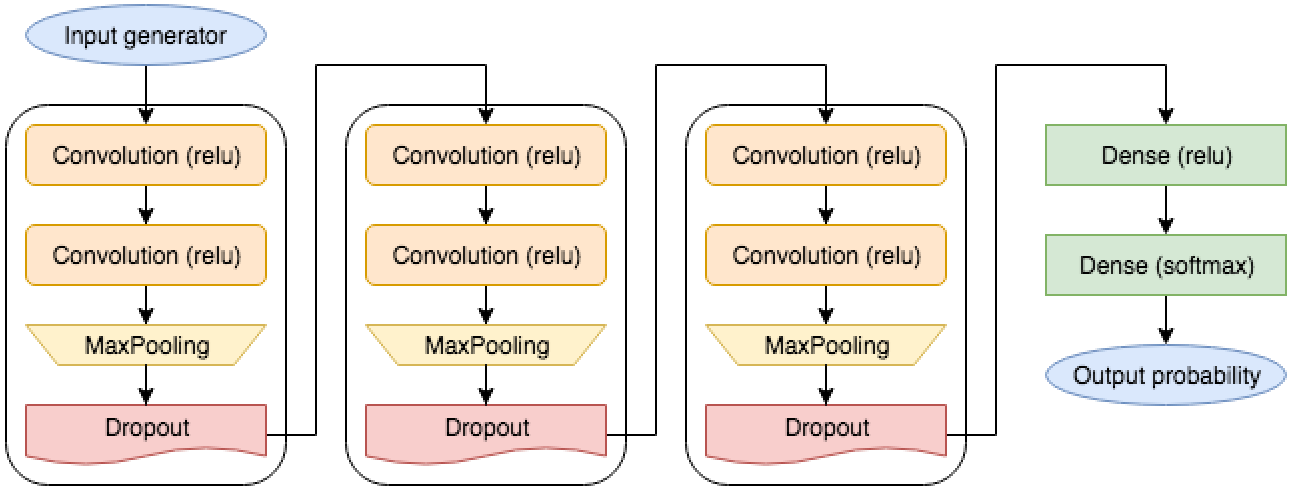
This dataset also has cropped and resized images, so that images include only numbers and nothing else. But for my approach I had to also extract examples of non-digit images for proper classification. They could be taken from any image dataset, but I decided to try to extract image regions that don't have digits on them from SVHN dataset itself. It limits all the variability of non-digit samples that can be found on street view image as the areas around the house numbers are usually monotone house walls, but this is one of the assumptions of this project that has to be taken in consideration during recognition stage. Of cause all the cutouts of the digits from the same images have been saved with appropriate labels and used for training.

Last step in data preparation was to resize all cropped images (digits and non-digits) samples to the same size, as CNN always expects certain shape of the input. Since digits usually have elongated form I resized all the images to 32x24x1 (1 stands for one grayscale channel) shape to not stretch the digits too much.

It's worth mentioning that I didn't use any noise reduction preprocessing since as I mentioned earlier CNN learns convolution filters on its own and those filters themselves can have noise reduction incorporated in them as the result of learning, just the same way as gaussian kernel can be combined with some edge detector into one kernel by simple taking their dot product.

Model

My model consists of Input generator, three blocks of two convolution layers, one max-pooling layer and a dropout, and ending with two densely connected layers of neurons. All convolution layers and one dense layer use ReLU activation function which converts neurons linear input into non-linear output allowing neural network as a whole to learn more complex fitting functions. Unlike many other activation functions ReLU doesn't suffer from the vanishing gradient problem which helps during backpropagation stage decreasing training time. Last dense layer however uses softmax activation. It's special characteristic is that all output features of such layer are in range of [0, 1] and their sum is equal to one. This makes this function popular in cases when one needs to find probability for all possible output classes. Class with highest probability in the output is usually considered best matching class.



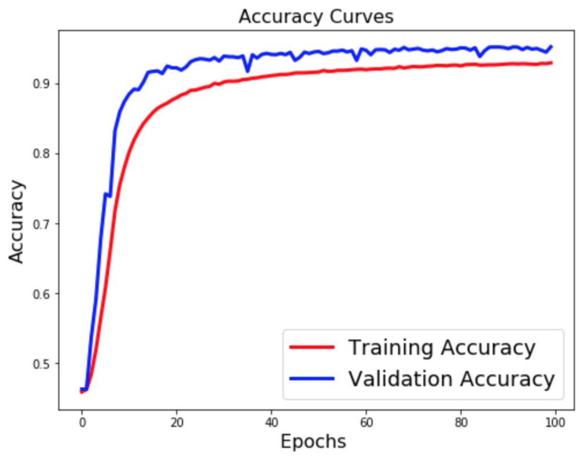
Instead of using just images as inputs for the network I used Input generator which plays an important task for this project during network training step. Its purpose is to modify inputs in various ways during training stage generating each time slightly different input even if input image is the same. Not only it helps to prevent overfitting and forces network to be more adaptable to various inputs but it also generates images that are present in real life but not in the dataset. That includes tilted images. This way neural network should be able to classify digits even if camera has been tilted to the side. Other image modifications like noise and movements also can be included. I didn't however enable flip transformation as it would make no sense for the numbers which are rarely can be seen on the building flipped.

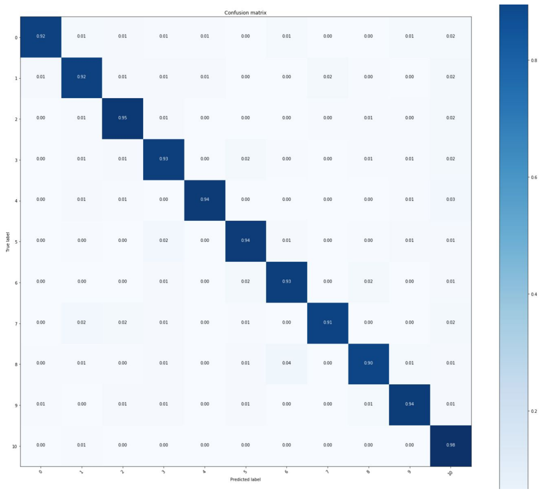
Other layers that haven't been discussed yet are MaxPooling layers Dropout. MaxPooling is the method of subsampling outputs of convolution layers. This technique helps reducing amount of information while keeping most locally dominant information. And finally Dropout [4] is a regularization layer that randomly nullifies previous layer output features before passing them to the next layers. This happens only during training stage. Intuitive idea behind this method is that it forces network not to rely too much on the same features again and again and to try to incorporate other features in its calculations. This usually leads to more robust classification as the result. For this project dropout rate was set at 25%.

Training

Model was trained on 108925 training samples and validated on 27232 samples. These were all cropped digit and non-digit images from SVHN dataset. Training stage minimized categorical cross-entropy loss function, that shows how different output probability is from real value for that input image where only correct class had probability of one while others' probability is set to zero. In the end after 100 iterations I was able to achieve accuracy of 95% on validation set.

Accuracy curve also shows no indication of overfitting as both training and validation curves rise and converge together (though validation accuracy was always higher). And confusion matrix also showed promising results as all classes were predicted as correctly in at least 90% cases and non-digit class having accuracy of 98%.



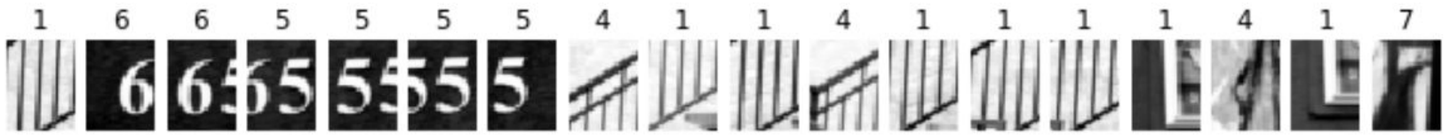


Post-processing

Unfortunately in real-world application outside the SVHN dataset I had much more non-digit cuts being misclassified as one of the digit class than it was during the training. And the reason behind that is as it was mentioned earlier that SVHN dataset doesn't have enough non-digit images even after cropping out areas next to the numbers. And those non-digit class samples are mostly images of a walls while the generic image can have much greater object variety.

One of the things I implemented to tackle this problem was to filter returned results based on labels (remove all non-digit predictions as they aren't actually needed) and then based on digit class probability. I ended up throwing away all samples where digit probability was less than 98%.

But even after that I still had many false positives in my output.



Usually those were the objects that came from the real world and seemed like a digit for a network that was previously trained mostly on the pictures of the walls, e.g. fence looked like 1, or part of golf-cart seat that looked like fancy 7 (last sample in the image above).

I had to make another assumption that there's only one number on the whole image and picked digit with highest probability along with all the digits close to it.

As you can see on the image above, another problem was duplicated predictions. This is a common problem in computer vision when sliding window doesn't move far enough ending up capturing the same object twice. But it can be easily countered by non maximum suppression algorithm that picks samples with highest probability and removes (suppresses) other samples with the same class which were too close to it.

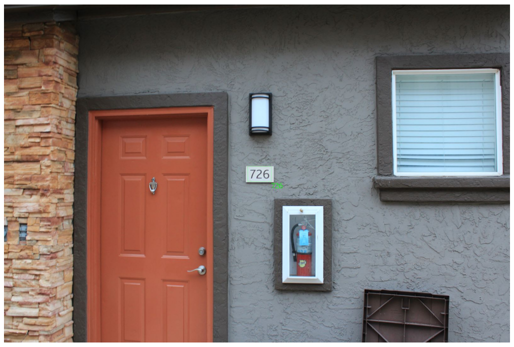
Finally to predict numbers of different sizes on different images I implemented image pyramid. Scaling original image to predetermined sizes and passing each rescaled image through the prediction pipeline. Number with highest probability among all the images was considered as a correct one. It's worth mentioning that even though non maximum suppression algorithm can work on samples of different size I decided to process each image in the pyramid separately as digits on house numbers are usually of the same size anyway.

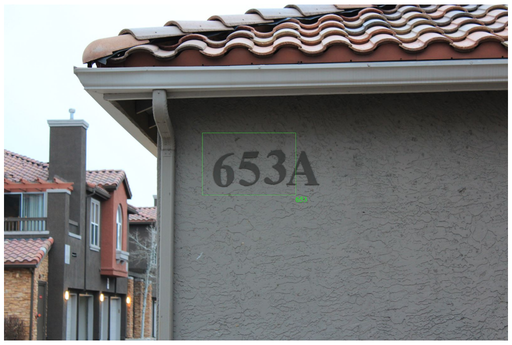
Results

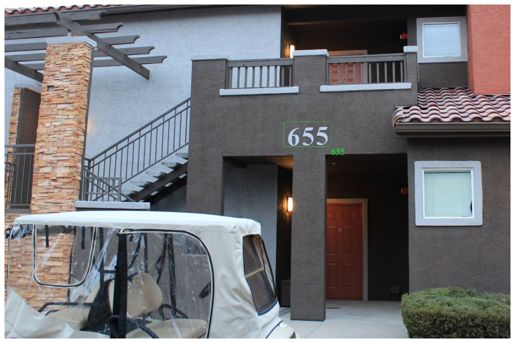
In the end, my pipeline with its preprocessing steps, scanning using sliding window, CNN and all post-processing steps was able to detect and identify with high probability numbers on various images with different lightning conditions, number styles (font, color, background, etc.), relative number position and orientation.

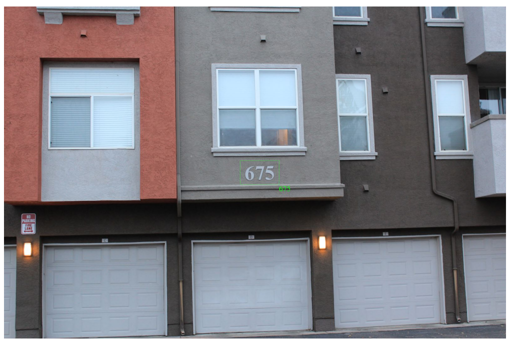
Not all of the images could be identified successfully, those that didn't usually had some elements or objects on the image that to CNN resembled digits but wasn't one. I believe that this can be fixed by adding more non-digit samples from other datasets (like original Google street view images) and retraining CNN using those samples.

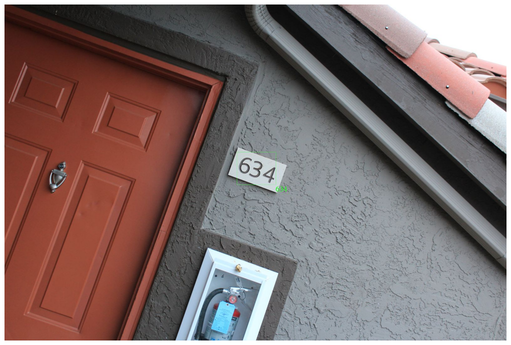
But nevertheless this model showed good enough results to be applied to videos like the one at the link below. However significant performance improvements should be made to apply it to live-stream inputs, currently each image/frame can be processed in 3-4 seconds at 1280x720 resolution.











Link to the video with number detection: https://youtu.be/r2tf0E-BPXs

Comparison with other models

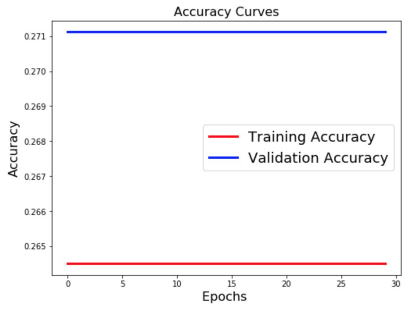
I also tried to adapt VGG16 network architecture [5], which consists mostly of 5 blocks with various number of convolution and a maxpooling layers in each block. So it’s pretty similar to my model except mine has less number of layers, and also incorporates dropout and input generator.

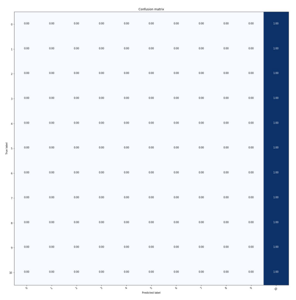
The biggest problem of adapting VGG16 is that it accepts images of 224x224x3 shape (3 color channels). So I had to create new colored dataset and significantly enlarge all the samples. Also last output layer had to be replaced as in this project there are 11 classes instead of original 8 from ImageNet.

Unfortunately such a big network requires a lot of training samples, while at the same time resized sample images significantly increased size of the dataset, making it difficult to process on standard PC.

I tried initiating model in two ways: with random weights as any new network would have been and with pre-trained weights from ImageNet training. Last one should have shown much faster training speed and results from the start thanks to the filters that it had learned from other images and that can usually be reused for other image applications [6]. Unfortunately in both cases instead of learning how to classify digits network decided to classify all samples as one of the random classes. Usually it was classifying everything as non-digit class since there were more samples of those than any other class (digits).

I think that this problem can be fixed by using bigger dataset by resizing images right before the input and not in advance to minimize required memory. Also perhaps some regularization techniques or other optimizers can be used to force network to actually train on new samples.





References

[1] SVHN dataset, http://ufldl.stanford.edu/housenumbers/
[2] Object Recognition with Gradient Learning, Yoshua Bengio, http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf
[3] MNIST database, http://yann.lecun.com/exdb/mnist/
[4] Dropout: A Simple Way to Prevent Neural Networks from Overfitting, N. Srivastava, https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
[5] Very Deep Convolutional Networks for Large-Scale Image Recognition, K. Simonyan, A. Zisserman, https://arxiv.org/abs/1409.1556
[6] A Gentle Introduction to Transfer Learning for Deep Learning, J. Brownlee, https://machinelearningmastery.com/transfer-learning-for-deep-learning/
[7] Recurrent Convolutional Neural Network for Object Recognition, M. Liang, X. Hu, https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Liang_Recurrent_Convolutional_Neural_2015_CVPR_paper.pdf
[8] Activation functions, https://en.wikipedia.org/wiki/Activation_function
[9] Vanishing gradient, https://en.wikipedia.org/wiki/Vanishing_gradient_problem
[10] Cross entropy, https://en.wikipedia.org/wiki/Cross_entropy