Multi-Digit Detection and Classification using Convolutional Deep Neural Network

Senthil Kumaran - Computer Vision, Fall 2019, Georgia Tech

A Convolutional Deep Neural Network to utilized to correctly identify and detect a sequence of numbers in real time with high accuracy. The detection and classification follows some ideas described in paper by Ian J. Goodfellow et al, but uses a unique, a simplified approach to deal with the complexity of the problem. The accuracy reached by the project is not as high as described in the paper, however, for a limited training time and an extremely small dataset, the problem solving architectured described below consistently achieves 94.6% validation accuracy.

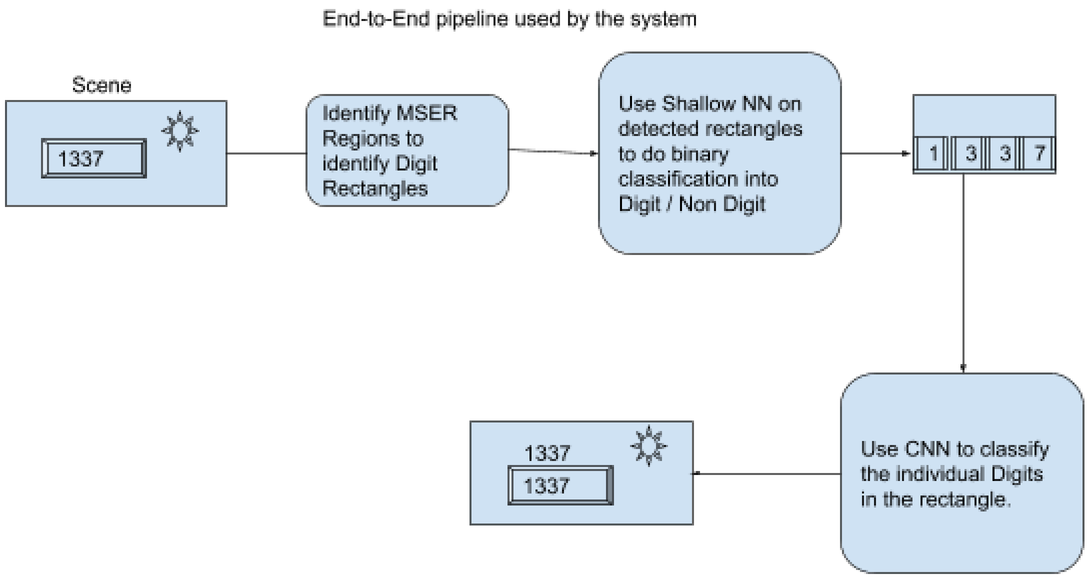
The detection of the sequence of digits is done by finding Maximally Stable Extremal Regions in a scenes, the numeric digits are in those regions are identified using a shallow CNN binary classifier. Once the digits are identified, a VGG16 based neural network is utilized to accurately categorize and label the digits. The system is invariant to the orientation, scale, and external characteristics of the image such a font, lighting and background.

The Convolution Based Deep Neural Network is built using a Keras high level library with tensorflow backend. Keras and Tensorflow provide high level tools to easily build a VGG16 based architecture and allows us to both train from scratch on a relevant dataset such as Google Street View House Numbers (SVHN) dataset, and also do a transfer learning on a pre-trained dataset like Imagenet.

I have implemented three models to do the classification of the digits in the deep neural networks. 1) A shallow neural network, 5 layers deep used for classification and quick run time. 2) A canonical VGG16 architecture trained from scratch and 3) transfer learning through VGG16 architecture pretrained on Imagenet dataset.

I ensured to have a high validation accuracy on all the models before these could be utilized in the pipeline for identification and classification tasks.

Since our goal is to build a real time classifier using neural networks, I designed the following computer vision pipeline to solve the end to end problem.


End-to-End pipeline used by the system
Scene
1337
Identify MSER Regions to identify Digit Rectangles
Use Shallow NN on detected rectangles to do binary classification into Digit / Non Digit
1 3 3 7
Use CNN to classify the individual Digits in the rectangle.
1337
1337

The figure describes the control flow between the modules of the pipeline. Each task is accomplished independently a software module and thus is system is easy to scale and debug for errors.

1. MSER Based Region Identifier. MSER provides stable regions based on thresholds. Since digit signs are normally placed on contrasting background, the MSER algorithm is an excellent choice to detect the digit rectangles.
2. A shallow neural network with customized weights provided a high degree of validation accuracy to classify a given figure as digit or non digit. The shallow networks makes the training tie of the system extremely fast.
3. A standard, trained VGG16 net is used for classifying the 10 digits of the figures. This was implemented using Keras and has a 94.6 validation accuracy.
4. Finally, the software framework and modular architecture ties all the components together and provides scalable and customizable system for doing a multi digit recognition both on an image and video.

Binary Classifier

Layer (Type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1	(MaxPooling2 (None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
dense_1 (Dense)	(None, 2)	130
Total params:	**122,050**	
Trainable params:		
Non-trainable params:	**0**	

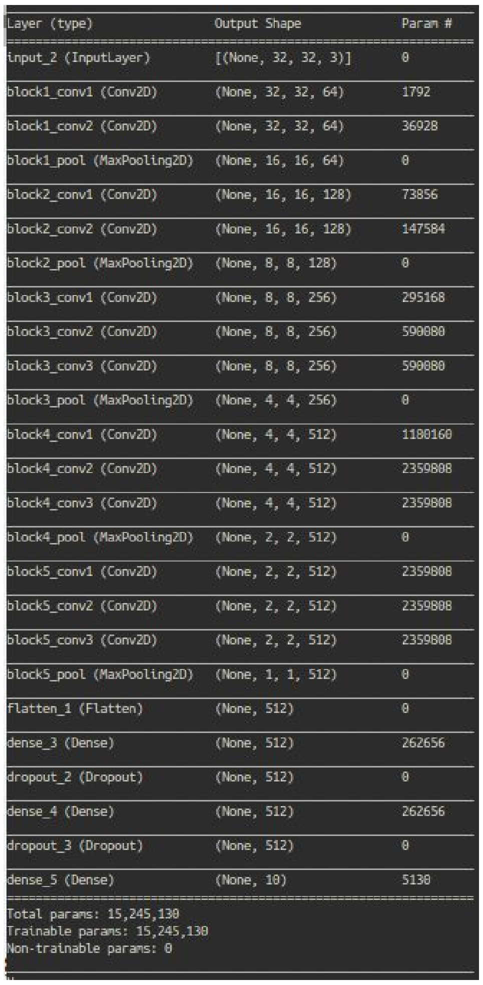
Neural Network for Digit / Non-Digit Binary Classification Sequential Model with only 3 Conv Layers

Shallow Neural Network

Layer (Type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1	(MaxPooling2 (None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
dense_1 (Dense)	**(None, 10)**	130
Total params:	**122,050**	
Trainable params:	**122050**	
Non-trainable params:	**0**	

Model 3 Conv Layers. The only difference from the binary classifier is the last layer has 10 classifiers instead of 2.

VGG16



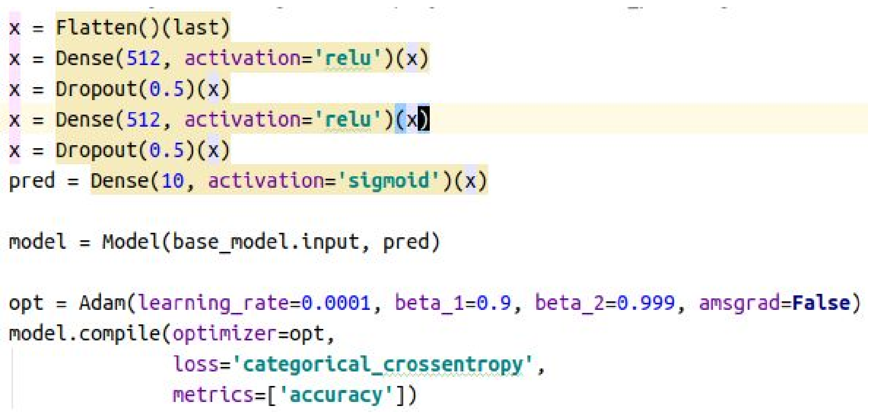
```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5130
=================================================================
Total params: 15,245,130
Trainable params: 15,245,130
Non-trainable params: 0
_________________________________________________________________
```

The other two architectures are standard VGG16 architecture, the only customization done in the fully connected layers and parameters tuned for learning rate.
Since SVHN dataset has an input shape of (32, 32, 3), and 512 fully connected layer with dropout of 0.5 performed better over the traditional 4096 fully connected layer of VGG16.

Data Preprocessing

- For Binary Classifier, SVHN Dataset training, test, and extra labels are set 0 to denote DIGITS CIFAR10 Dataset is added as negative data and digits set 1 to denote NON-DIGITS. The network was trained to identified DIGITs from Non Digits.
- For all SVHN data the Images are multiplied by 1.0/255 to set their values between 0 to 1. This gives higher for conv neural work.
- The labels were set to categorical and the loss function was chosen to the categorical cross entropy as it fits the data set properly

Implementation



```
x = Flatten()(last)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
pred = Dense(10, activation='sigmoid')(x)

model = Model(base_model.input, pred)

opt = Adam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

The model was fit to the data iteratively.

The checkpoints were used during training so that entire training was done done over multiple attempts.

Categorical Cross Entropy

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

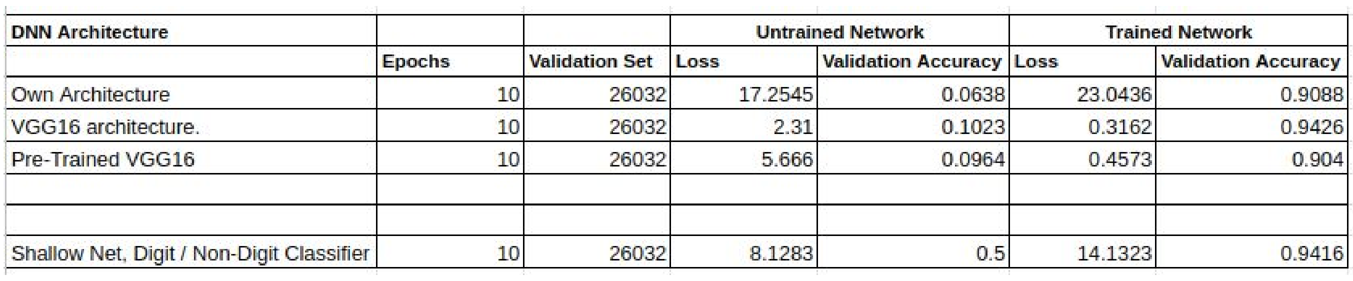
This spreads the loss across the labels. Yi. The Yi chosen is a one-hot-encoding, that thus categorical_cross_entropy was the proper choice for the loss function.

For lazy loading of data, a subclass of **Keras.utill.Sequence** was designed that could be customized for batched data processing, and loading, only a small set of data was held in memory at any given point in time, and this helped me to develop the model on 4 core computer, 16 GB RAM with 1 NVIDIA GPU GTX 106.

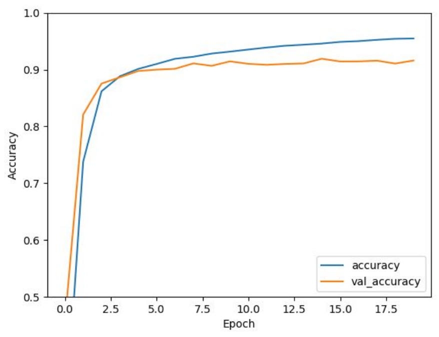
I tried with both Stochastic Gradient Descent (SGD) and Adam Classifier. The Adam Classifier with the learning rate of 0.0001 provided higher validation accuracy.

The checkpoint metric was **val_accuracy** over **max,** only the model that provided high validation accuracy was chosen for snapshot.

Performance Statistics



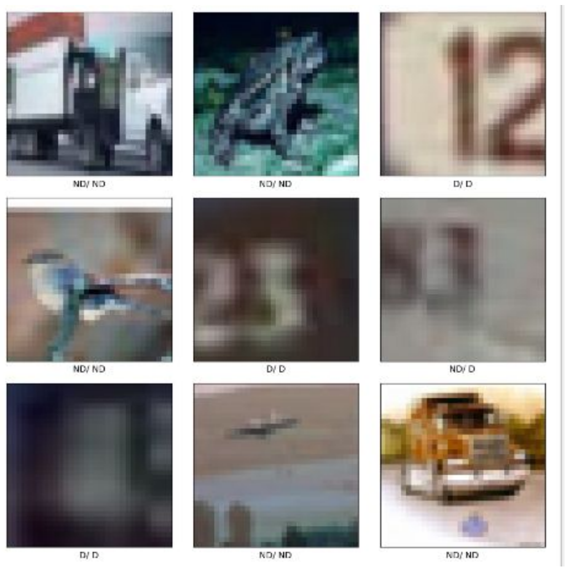
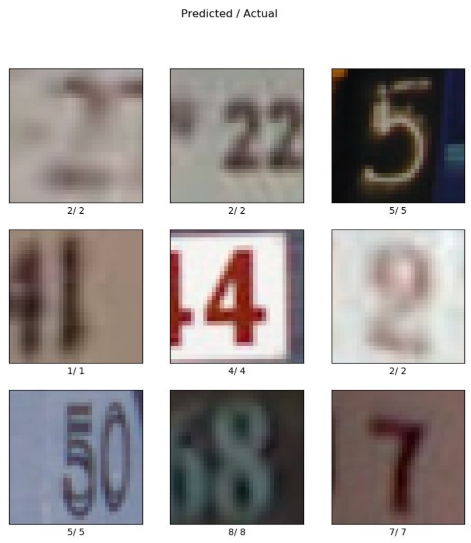
DNN Architecture			Untrained Network		Trained Network	
	Epochs	Validation Set	Loss	Validation Accuracy	Loss	Validation Accuracy
Own Architecture	10	26032	17.2545	0.0638	23.0436	0.9088
VGG16 architecture.	10	26032	2.31	0.1023	0.3162	0.9426
Pre-Trained VGG16	10	26032	5.666	0.0964	0.4573	0.904
Shallow Net, Digit / Non-Digit Classifier	10	26032	8.1283	0.5	14.1323	0.9416



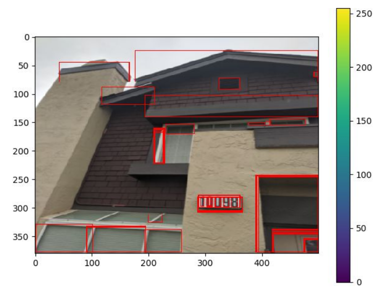
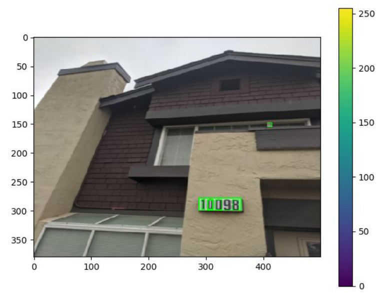
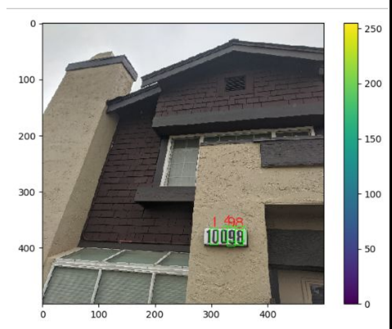
The VGG16 classifier of 94% accuracy was trained and used for Digit Classification. This was combined with the Shallow Net Digit / Non-Digit Classifier which also had a 94.16% accuracy.

All the models reach the validation accuracy of close 0.9 to 0.95 over 10 epochs. The figure on the right is the personal architecture reaching upto 0.9

Validation Results

 ND/ ND ND/ ND D/ D ND/ ND D/ D ND/ D D/ D ND/ ND ND/ ND	 Predicted / Actual 2/ 2 2/ 2 5/ 5 1/ 1 4/ 4 2/ 2 5/ 5 8/ 8 7/ 7
The neural network has properly identified picture containing Digit from a picture that does not contain Digit. There is one negative identification shown in the data set where network classified as ND (Non Digit) when it was Digit.	VGG16 Classifier for the Digits. All the test values in the validation set were properly identified by the network.

Result of the pipeline

MSER output	**Digits identified by binary classifier**	**NMS done with Digits written above the identified digits.**
		

Improvements

Accuracy of 94% is still not high. LongFellow et al have accuracy near 98% for multi-digit classification by a metric of accuracy defined more toward real world analysis.

A near 99% validation accuracy for Digits vs Non Digits can help identify all the digits in the picture. Here in the above results, my model identifies digits but fails to classify 0s due to overlap with nearby digit.

My negative dataset of CIFAR10 was not representative enough or the non digits. Higher quality data for negative data, derived from the SVHN dataset itself will have provided higher accuracy.

My tuning of MSER is not accurate for all the real world images. The digit recognition could itself be a Deep neural network that will be provide very high validation accuracy.

Video Presentation

https://youtu.be/fqUHKYGHvtE

References and citations

1. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks by Ian J. Goodfellow et al.
2. https://keras.io/utils/ Keras Documentation
3. https://www.tensorflow.org/ Tensorflow Documentation
4. Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan, Andrew Zisserman https://arxiv.org/abs/1409.1556
5. The Street View House Number Dataset http://ufldl.stanford.edu/housenumbers/
6. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html Keras Tutorials
7. https://en.wikipedia.org/wiki/Rectifier_(neural_networks)