INTRODUCTION TO COMPUTER VISION

COMPUTER VISION CURRENCY CONVERSION

December 15, 2018

Clayton Kramp and Katrina Steinman

Colorado School of Mines

INTRODUCTION

Optical Character Recognition is a challenging task, and there are many tools currently implemented that can interpret computer-generated text reasonably well. However, in order to interpret handwritten data, it is necessary to implement a neural network. We decided to approach this task not for words, but for numerical data. Specifically, we wanted to interpret currency values in an image. This report details the implementation and results for our currency conversion program. Our hope was to create an original program that is relevant at a worldwide scale. As the project progressed, we found there were three key aspects: region of interest localization, numerical classification, and symbolic classification. Region of interest localization enabled us to remove background noise by passing a specific region into the numerical and symbolic classification portions. Numerical classification enabled the currency conversion calculations. Symbolic classification enabled the identification of input and target currencies. Each of these portions was implemented via its own neural network. The expected input formats are described in the Data section, and the neural network implementations are described in the Methods section. The output of this program is the conversion projection back into the region of interest.

DATA

Localization Data

We have provided 32 training images to train our Faster R-CNN. The training images are photos of our expected input: black text on white background. We have varying images in different lighting, different backgrounds, centered and off-centered, and noise in background. This way, we can train our network on a variety of images. Provided the input data set, we manually declare the region of interest using `MATLAB`'s Image Labeler application and save a table containing filenames and the bounding box for each image.

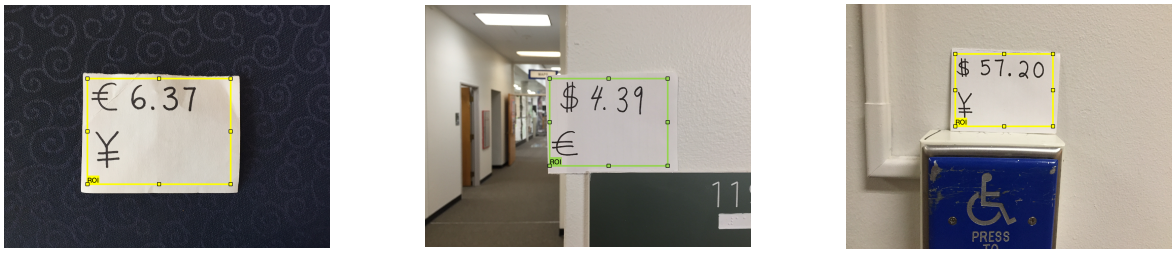

Figure 1: Sample input for localization

MNIST

We use the MNIST handwritten digits data set to train our digit classifying network [12]. It includes 60,000 training images and 10,000 test images. The digits are written in various thicknesses and styles to offer a vast variety. The MNIST dataset is often used as a benchmark for classification networks, and teams have proposed numerous networks that have achieve different results using different topologies. Each digit is a 28 x 28 image with white text on black background.

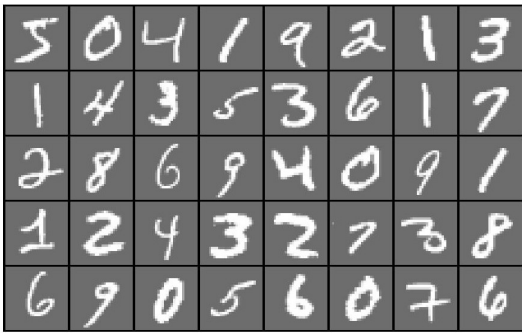

Figure 2: MNIST Sample

Symbolic Data

We have also created our own data set of handwritten images for a few currency symbols. In particular, we have included the Euro, US Dollar, and Yen symbols. This was done because a good data set for handwritten samples of these symbols was not found. We have included 60 training images per symbol for a total of 180 training images, and 15 testing images per

symbol. Various thicknesses of pen and style were used to fit a variety of samples.

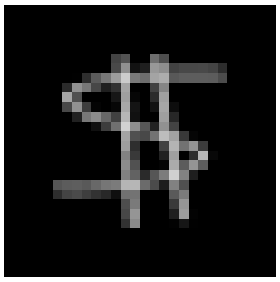

Figure 3: Dollar Sample

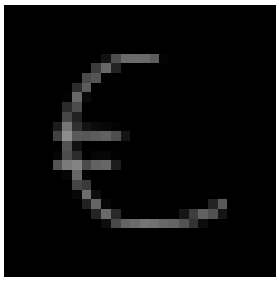

Figure 4: Euro Sample

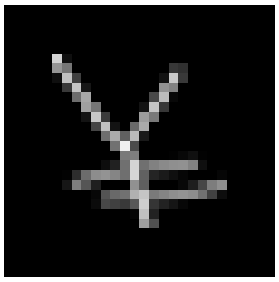

Figure 5: Yen Sample

Expected Input

For our input image, we expect black text on a white background. For the formatting, we expect on the first line to have the original currency followed by some numerical amount. Then, on the second line, we expect to have some currency to convert to, directly underneath the original currency. In this way, we have sufficient space to project our output onto the original image. Examine Figure [1] for an example input. As the network has been trained on a variety of inputs, the input is allowed to have noise in the background or be off-centered. The only restriction is to limit the noise inside the region of interest as much as possible, as they could lead to false detection of symbols or numbers.

METHODS

Faster R-CNN

Fast Region-based Convolutional Neural Networks use deep convolutional networks for object detection. Our decision to use a Faster R-CNN is due to its brilliance of conducting both region proposal generation and objection detection by the same convolutional network [3]. Faster R-CNNs are an extension on Fast R-CNNs. Faster R-CNNs improve the region proposal algorithm by implementing a region proposal mechanism using a CNN. In this way the

region proposal method is trained alongside the detection network [2]. `MATLAB`'s Computer Vision System Toolbox has the `trainFasterRCNNObjectDetector` function implemented such that the user can configure the dataset, CNN, and the training options. Then, we can train the object detector with the provided hyper-parameters and configurations. For our implementation we train a Faster R-CNN detector to find our region of interest.

Topology

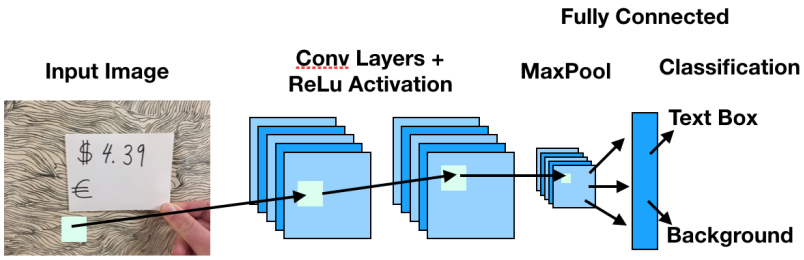

Figure 6: Topology of the Faster R-CNN

We have designed a deep CNN for our object detector. Deep CNN's are a powerful tool for object detection, as they are able to extract key features and localize features in an image [10]. Our input image is resized to be 200 x 300, which from our experimentation was the smallest size that preserved the detail of each character in our input image. Then we convolve by taking a small part of the image, and backpropagate to update the weights. We have set up 2 convolutional layers each followed by a Rectified Linear Unit (ReLU) activation layer. ReLU is regarded to be, in practice, the most effective activation function as it is simple, does not have the issues of sigmoid where sigmoid is computationally expensive, and does not saturate like sigmoid [1]. The convolved image is then passed through a maxpool layer that will downsample the image. A softmax layer is added to squash values from 0 to 1, and to have each node add to 1. Lastly, the network is passed through a classification layer. After training the detector, we can run the function `detect` with the trained detector and an image

as arguments. Then, this outputs the bounding box and the score of the confidence.

Training

The training takes approximately 10 minutes, and involves 4 steps. The first 2 steps involve training the region proposal and detection network. The latter 2 steps combine the network to develop a single network for detection. In the first iteration of the training, the accuracy is 3%. This is expected as there are 32 training samples, and the probability of getting a correct result is $\frac{1}{32}$. However, over the course of training, the object detection improves significantly, and the training schedule can be observed as console output from our code.

Accuracy and Results

We run the detector on a test image, an image that was not used in the training set. We can then draw a rectangle over the proposed bounding box, and display the 'score' of the box. From this example, we can see that we have a very positive result. We can then crop the image using the bounding box to be processed by our character classifier later. We tested the performance of the object detector over 5 test images, and the results are as follows:

Image	Symbols fully within frame	No inclusion of externals
1	Yes	No
2	Yes	Yes
3	Yes	Yes
4	Yes	No
5	Yes	Yes

In all cases, we have that the symbols are fully within the frame. A few of the bounding boxes include some noise from slightly outside of the box. In most cases, the image pre-processing can handle the noise, so it is not a big issue. The object detector can successfully encapsulate all the symbols in a box, and minimizes the area as much as possible to reduce external noise.

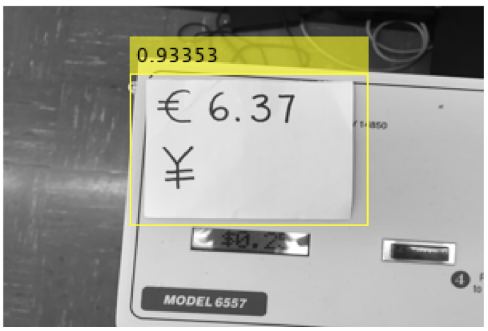

Figure 7: Example that includes symbols within frame but includes externals

Image Pre-processing

After we crop the image using the localization tool from the Faster R-CNN program, we must apply some pre-processing to be able to properly apply bounding boxes around each individual character. We run a series of processes to properly format each character.

Binarize and Invert Colors

We first binarize the image with the threshold 0.5 to make everything in the image black or white. Then, we must invert the colors in order to match the input from the MNIST library, as

all images in the MNIST library are white text on black background. This process allows us to remove noise from different lighting.

Removing Noise

We run `bwareaopen(I, 5)` in order to remove some noise. The function removes all bounding boxes of size ≤ 5 pixels, and this way we can remove any noise that is undesired. In our experimentation, we have found that this process is successful at removing any noise that is introduced inside of the region of interest.

Bounding Boxes

We run `MATLAB`'s `regionprops` function. After we have run clean up processes, we can finally run our bounding boxes function to compartmentalize each individual character.

Gaussian Filter and Resize

We run `imgaussfilt` to blur the input image slightly. Our input images from MNIST have 28 x 28 images and each character is drawn in gray. This means that a simple black and white threshold will not look similar to our testing inputs, so we smoothen edges to make our inputs similar to the testing inputs. This smoothening process resulted in far better recognition. Below is an example of the outcome of gaussian filtering:

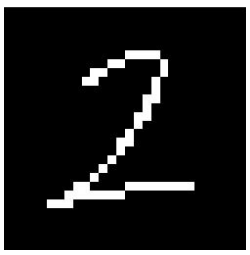

Figure 8: Original input image

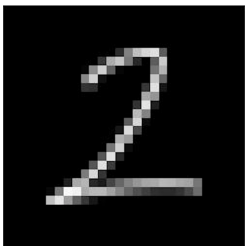

Figure 9: Post processing with Gaussian Filter

After finally running all the processes, we resize each input image to be 28 x 28. We also apply padding on each side, so that the digits are not in the frame edge-to-edge but instead

somewhere in the center. This process helps to match the input of the MNIST data, and makes the images look similar. Now, the input is ready to be passed through the neural network for classification.

MNIST CNN

Classification networks take in a labelled data set for training, and attempt to apply labels to a separate set of images during the testing phase. The convolutional neural network we developed to recognize handwritten numbers is a classification network, and trains on the MNIST database. Though this network is shallow, it still achieves very high levels of accuracy [5].

Topology

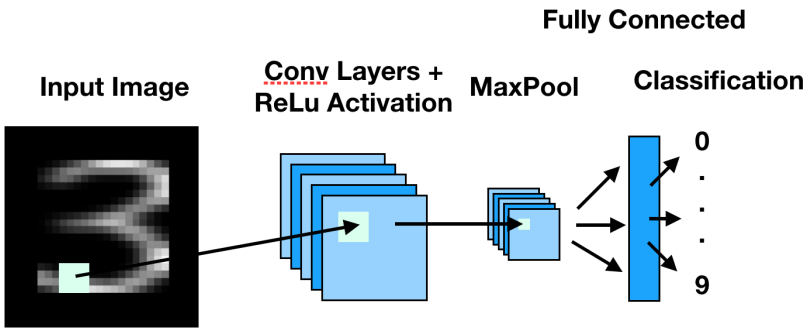

Figure 10: Topology of the MNIST CNN

We have designed a shallow CNN for number classification. The input image is resized to be 28 x 28, since this is what the network expects, and the resulting image is what we convolve on. The network consists of a convolutional layer, ReLU activation layer, max pool layer, dropout layer, fully connected layer, and softmax loss layer. This is a classic topology for image classification, and is a fundamental basis for convolutional neural networks [6]. The details of the hyperparameters can be found in our code. The ReLU layer was included

because the non-linear activation function will allow for non-linear mappings from inputs to outputs during the learning process. The dropout layer was included because it improved accuracy. This is due to the fact that dropout layers add regularization, so the network avoids overfitting [7] [9].

Training

The training takes approximately 1-2 minutes for 1 epoch, which achieves approximately 97% accuracy. We decided to extend the training to 10 epochs to achieve higher accuracy, but it should be noted that this takes significantly longer. From Figure 11, we can see that the training accuracy starts very low. This occurs because the initial labels of 0-9 are essentially random, and since there are 10 digits, we see approximately 10% accuracy. However, after this point the training accuracy increases very rapidly, and we see that the training reaches a plateau around 98%. We can also see that our loss function decreases accordingly, which is an expected result and shows that our network is properly training.

In Figure 12, we can see filters that this neural network develops. These are a visualization of the features that the network has identified.

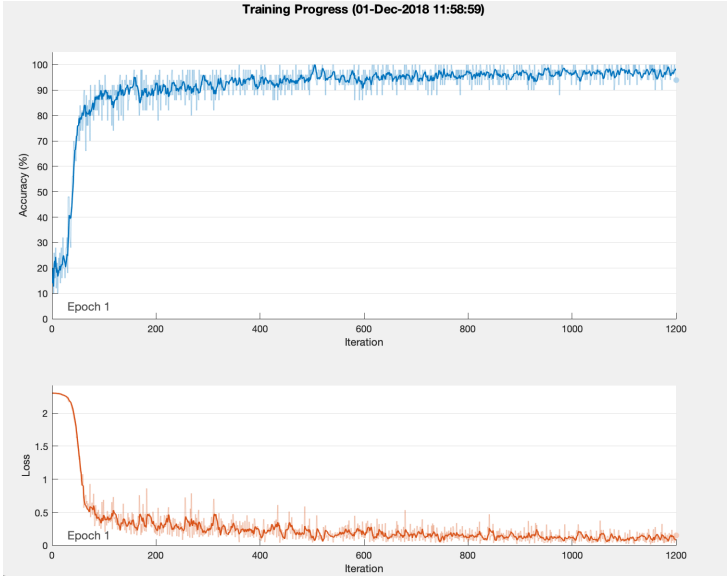

Figure 11: MNIST Training Progress

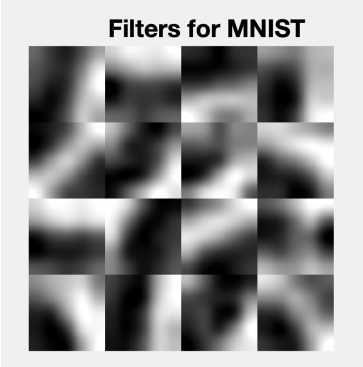

Figure 12: Filters for Numerical Classification

Accuracy

With the topology described above, we were able to get an accuracy level of 98.6% for the 10,000 testing images in the MNIST database. This was over the course of 10 epochs, using a batch size of 50.

Symbol Classification CNN

Similar to the previous network, this CNN is used for classification. This network is trained to recognize handwritten currency symbols from our original database. As before, we found that a shallow network was sufficient to achieve very high levels of accuracy.

Topology

We have designed a shallow CNN for symbol classification. As in the previous network, the input image is resized to be 28 x 28. The topologies of the number classification and symbol classification networks are the same. This maintains high accuracy because both networks perform in an extremely similar fashion. They both receive the same input image and expect the same formatting.

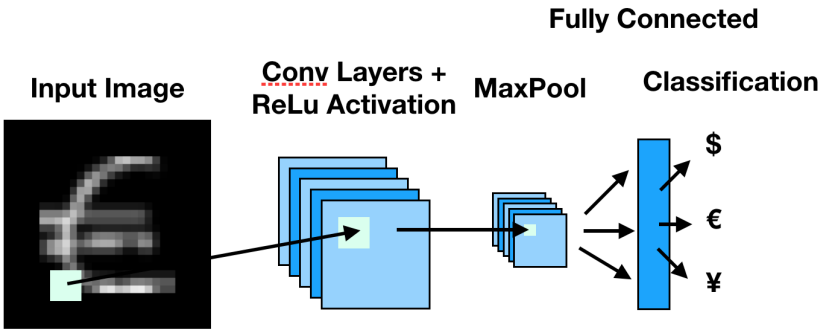

Figure 13: Topology of the Symbol Classification CNN

Training

The training takes approximately 15 seconds for 10 epochs, which achieves very high accuracy. In Figure 15, we can see that the accuracy will not increase significantly after this point. As before, the training accuracy starts very low because the initial symbol labels are essentially random. The training accuracy increases very rapidly after this point.

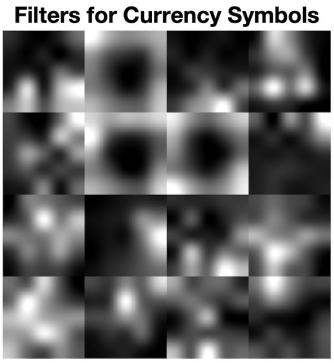

Figure 14: Filters for Currency Symbol Classification

Figure 14 displays the filters that the neural network develops. These are the features that the network has determined were key features in an input image. It is very interesting and fascinating to see what the statistical procedures of learning has led to.

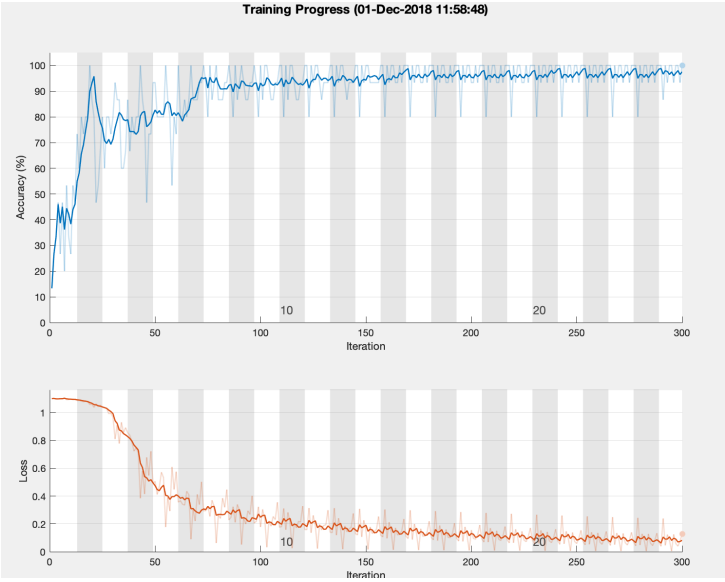

Figure 15: Symbols Training Progress

Accuracy

With the topology described above, we were able to get an accuracy level of 97.7% for the 45 testing images in our original database. This was over the course of 25 epochs, using a batch size of 15.

OUTPUT

The cropped image from the Faster R-CNN is passed to our symbol/digit CNN. This network examines the left two symbols to know which currency to convert to which. Then we loop through each of the bounding boxes and convert them to digits. We convert the currency, and project it back to the image. Figure 16 shows that we successfully implemented our project, and that our program successfully takes in an image, classifies it, and projects the outcome to the image.

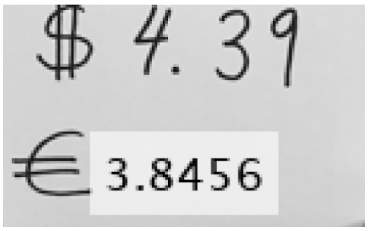

Figure 16: Final Output of Currency Conversion

RESULTS

Our MNIST Network achieves 98.6% accuracy with the 10,000 test images, and our symbolic network accuracy achieves 97.7% accuracy with 45 test images. The Faster R-CNN successfully creates an appropriate bounding box encapsulating all the symbols in our image, and on average only includes up to 10% of external noise. Of the 5 test images, when no external noises are added, we have perfect detection. When external noises are added, we construct bounding boxes around random external noise, and since we are working with a closed world classifier, noise becomes classified as some digit. Overall, we see that our symbol and digit classifier performs extremely well, and with improved symbol detection, we can improve our results.

DISCUSSION

Analysis

From our results and outputs, we have found that the performance of this program is highly dependent on the output from the Faster R-CNN step. As previously discussed, there are two metrics to describe the output from this step: inclusion of external noise and full enclosure of input data. If there is outside noise included in the region of interest, then there will be incorrect bounding boxes used in the numerical and symbol recognition steps. This results in incorrect output. If not all input symbols or numbers are included, then the output will

also be incorrect.

If Faster R-CNN produces perfect output, meaning that all desired characters are included in a region of interest with minimal noise, then the results and calculations returned are perfect. We have also found that the concept of “perfect" is relatively tolerant to imperfect inputs. For example, small amounts of noise will be eradicated via preprocessing of the input image. Additionally, variance in lighting does not significantly affect output accuracy.

These observations explain the results that we see for each given test image.

Technical Challenges

We encountered several challenges while developing the project. Learning and implementing the various neural networks and creating our own data sets accordingly was a major challenge. Also, after parsing in the inputs, it was challenging to structure the input to look similar to the trained data. If the input does not look like the trained data then the input is not useful.

Further Improvements

In the future, we would like to improve the practicality of this system. If an image was taken at a store, there would likely be large amounts of background noise. This could include labels, non-price related numbers, and bar codes. Currently, our program is not equipped to handle inputs with these features. Additionally, there may be font differences across different labels, which would require training on data sets that are not handwritten.

Outside of this, we would like to extend our implementation to handle more than the three currency symbols in our symbol database. This would make the program applicable on a worldwide scale.

We would also like to combine the detection and classification into one network such that the network will search for individual symbols and then classify them directly.

Finally, we would like to remove the need for the specific format our program expects.

Currently, it expects one line with a currency symbol and numerical value, followed by a separate line with a secondary currency symbol. In the future, the expected format should not need the secondary currency symbol, since these do not occur on existing labels and price tags. Instead, the user should be able to manually select the currency they wish to convert to. Making these improvements could make these systems applicable in real-world scenarios.

RELATED WORKS

Though there are patents that have been filed for tools similar to the one we have created, there is no literature published that describes currency conversion via optical character recognition. The following related works have executed similar tasks or techniques.

DropConnect is a technique to regularize a neural network. Instead of a simple dropout layer which sets a subset of activations to zero, DropConnect sets a subset of the weights to zero. Thus, instead of output being dropped, a connection is dropped which leads to prevention of co-adaptation of units [4]. This network proposed by NYU achieves 99.79% accuracy on the MNIST library. We found that our network performed sufficiently well without this addition.

Tesseract is an optical character recognition software that is sponsored by Google. It is recognized as the most accurate character recognition software and is capable of recognizing 116 languages, and also capable of parsing entire paragraphs [8]. However, Tessaract is very poor at recognizing characters in images with significant noise, external features, or rotations. Tessaract was a good starting point for this project, but it only works for computer-generated text. Therefore, this was not applicable for training with handwritten data sets.

Optical Character Recognition using Matlab discusses an OCR implementation with handwritten alphanumeric characters. Though this is similar to what we wanted to achieve, it performs very poorly on inputs even when there is no noise, and it does not identify specific

regions of interest [11]. Therefore, our work has extended beyond the feature extraction described in this paper, though our work does not include alphabetic characters.

CONCLUSION

This project demonstrates our knowledge in various convolutional neural networks. We successfully implemented two classification CNNs with very high accuracy, and one deep Faster R-CNN object detector, a state-of-the-art object detection network. We implemented each network from scratch and configured layers, epochs, learning rates, and options to specifically cater to each dataset's needs. We also developed two original data sets, processed them, and applied them to our network in order to develop networks to work for our problem. Finally, we were able to gather all the networks to work as one and developed a handwritten currency converter. Overall, this project has developed our understanding of neural networks and their applications in computer vision topics.

LINK TO VIDEO

`https://www.youtube.com/watch?v=paVdY1oUDe8`

Bibliography

[1] Peter Sadowski Pierre Baldi Forest Agostinelli, Matthew Hoffman. Learning activation functions to improve deep neural networks. *ICLR*, 2014.

[2] Ross Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[3] K. He, G. Gkioxari, P. DollÃąr, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Oct 2017.

[4] Sixin Zhang Yann LeCun Rob Fergus Li Wan, Matthew Zeiler. Regularization of neural networks using dropconnect. In *Dept. of Computer Science, Courant Institute of Mathematical Science, New York University*.

[5] Mark D McDonnell, Migel D Tissera, Tony Vladusich, Andre Van Schaik, and Jonathan Tapson. Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the âĂŸextreme learning machineâĂŹalgorithm. *PloS one*, 10(8):e0134254, 2015.

[6] John Murphy. An overview of convolutional neural network architectures for deep learning. *Microway, Inc.*, 2016.

[7] Alex Krizhevsky Ilya Sutskever Ruslan Salakhutdinov Nitish Srivastava, Geoffrey Hinton. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 2014.

[8] Chirag Patel, Atul Patel, and Dharmendra Patel. Optical character recognition by open source ocr tool tesseract: A case study. *International Journal of Computer Applications*, 55(10), 2012.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[10] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2553–2561. Curran Associates, Inc., 2013.

[11] Sandeep Tiwari, Shivangi Mishra, Priyank Bhatia, and Praveen Km Yadav. Optical character recognition using matlab. *International Journal of Advanced Research in Electronics and Communication Engineering*, 2(5):pp–579, 2013.

[12] Christopher J.C. Burges Yann LeCun, Corinna Cortes. The mnist database. *Courant Institute, NYU, Google Labs, New York, Microsoft Research, Redmond.*