Classification and Detection with Convolutional Neural Networks

Erica McEvoy
emcevoy6@gatech.edu

Introduction

One of the fundamental problems in Computer Vision research is that of Object Detection, whose task is to locate various objects in an image, draw bounding boxes over those objects, and classify them according to their object type. Object Detection algorithms are generally of two types: single-stage or multi-stage. In single-stage models, the localization and classification components are computed as a whole, while multi-stage models separate the localization and classification components into stages of a broader pipeline. In this paper, we discuss the results of creating and applying a multi-stage Object Detection model to localize and classify digits appearing on houses and buildings while utilizing Convolutional Neural Networks.

Related Work

Recent developments in multi-stage models are defined by the family of R-CNN detection techniques (short for "region-based convolutional neural networks"), whose methods include the original R-CNN [1], Fast R-CNN [2], and Faster R-CNN [3]. The R-CNN involves the following stages: A Selective Search algorithm is used to propose locations for bounding boxes; these regions are then warped; the warped regions have features that are extracted via a CNN; bounding box predictions and object classifications are made with linear regressors and linear SVMs. The Fast R-CNN and Faster R-CNN are variational improvements of this architecture that allow for a faster computed test time for each image.

Single stage detectors take a different approach, and are the most recent developments in Object Detection. They skip the region proposal stage altogether, and instead run the detection directly over a very large number of potential locations. The SSD [4] and family of YOLO [5] [6] architectures (short for "single shot detector" and "you only look once", respectively) are the most widely recognized one-stage detectors today. In general, single-stage detectors tend to be less accurate than their multi-stage counterparts, but are significantly faster to compute.

Method

The multi-stage method used for object detection in this project can be explained in two phases. The first phase involves applying the MSER algorithm [7] to generate bounding boxes around detected Regions of Interest (ROI). The MSER algorithm was tuned to detect and include ROIs containing individual digits within an image. These ROIs were then sent to a

CNN-based classification model to classify the detected digit. While multiple pipelines/algorithms were experimented with during image pre-processing, the following pipeline appeared to produce the best overall results for both images and video: (1) Read in the image with OpenCV and convert to greyscale; (2) MSER reads in the greyscale image and produces bounding boxes around multiple Regions of Interest (ROIs); (3) The ROIs were extracted from the color image and were reshaped to 32x32x3 dimension; (4) Those cutouts were fed into the trained custom CNN classifier (described below) and generated a class prediction per ROI; (5) Plotted the bounding box and corresponding class prediction onto the original color image.

For the classification task, three models were created, trained, evaluated, and ultimately the best performing model was used as the classifier in the Object Detection pipeline. All of these models were trained on numerical digits contained in the SVHN dataset (Format 2, specifically). Each model was built to perform a multi-class classification task on a total of 10 classes (each class representing a digit between 0 and 9). * (Footnote: A model was built using 11 classes to account for non-digit objects, and was ultimately not chosen as high performing, see Experimentation section).

In the first model, Transfer Learning was applied to the dataset using a PyTorch implementation of the VGG16 network utilizing its pre-trained weights. The weights in the convolution layers, as well as the first two fully-connected layers, were kept frozen; while the last convolutional layer was replaced with two fully-connected layers (with ReLU activation in the first), a 0.1 percent dropout rate applied between them (for slight regularization), and an output size of 10 classes at the last layer. The beginning layers were kept frozen in order to preserve the general, low-level features that were originally learned from the entire ImageNet dataset the network was originally trained on. Modification of the last layers allows the model to learn the more specific features of the SVHN dataset, and thus be “fine-tuned” to this particular task.

In the second model, a PyTorch implementation of the VGG16 network was trained from scratch (no usage of pre-trained weights). To accommodate our particular classification task, the same modification was made to the last convolutional layer as described above (in the first model), with all model parameters made available for training (i.e., no model weights were frozen).

In the third model, a custom CNN network was built in the following manner: a simplified version of the VGG16 network, consisting of 4 convolution layers and 2 fully-connected layers. Batch normalization, max pooling, and ReLU activation were applied to each convolution layer, while the last fully-connected hidden layer had ReLU activation and a dropout rate of 0.1 percent. The final layer contained 10 classes to accommodate our specific classification task.

During training of all three CNN models, a Categorical Cross-Entropy loss was chosen for the loss function of the network, which is an appropriate measure of distinguishing between probability distributions that allows for computing probabilities (outputs) across 10 classes for each image.

Because the training set size is so large (73K images), Mini-Batch Gradient Descent was utilized for the optimization technique, which ensured the fastest amount of learning to occur during training. By definition, Stochastic Gradient Descent processes each training image one at a time before updating the model weights, so the computational speed gains resulting from efficient vectorization of the network computations are mostly lost, and would thus not be an appropriate choice. Similarly, the choice of Batch Gradient Descent would be inappropriate in

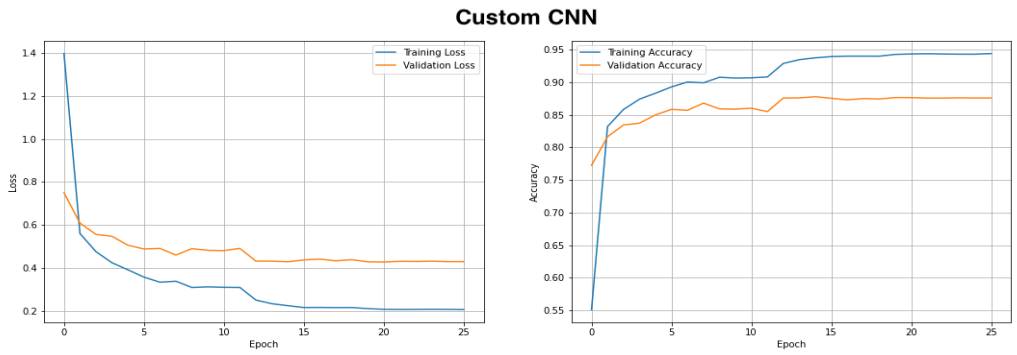
this case because of the lengthy time requirements needed to process through the entire training set for a single model update. For each of the three models trained, the mini-batch size was chosen so that all of the batched training data could fit within the memory size of the GPU. The VGG16 models allowed a maximum mini-batch size of 256, while the custom CNN implementation allowed a maximum mini-batch size of 64, so these were the values used in the training of the three models (the choices of sizes expressed as powers of 2 is due to the way computer memory is laid out and accessed).

The learning rate is defined as the value of the parameter that controls how quickly the gradients are allowed to change at each iteration. When the learning rate is too large, convergence to a minimum of the loss function is compromised. When the learning rate is too small, convergence can become too slow. Thus, the learning rate is a crucial hyperparameter to tune for optimal training within the network. Several training runs were performed for the custom CNN model, where the initial learning rate was tuned to occur between 0.003 and 0.0003, and learning rate decay was utilized with a decay factor of 10 (if the validation set accuracy ceased decreasing over 3 epochs). The outcome of the training runs where the learning rate was > 0.0003 had resulted in poor classification accuracy (specifically, accuracies around 20% were observed). As a result, the learning rate of 0.0003 was chosen for all three models.

During training of each of the three models, an Early Stopping technique was utilized. In Early Stopping, training is stopped when the validation set error stops decreasing. This indicates that the network was performing at its best at that precise iteration since the validation set error has reached a minimum. A Patience parameter of 5 epochs was used to monitor if early stopping was indicated during training.

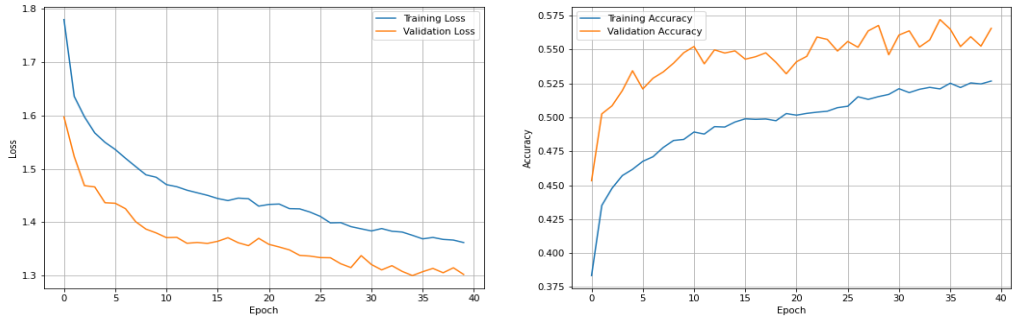
Experiments

After training of each model was performed, the training and validation accuracy and loss were plotted, see below. The VGG16 model trained from scratch performs the worst with a

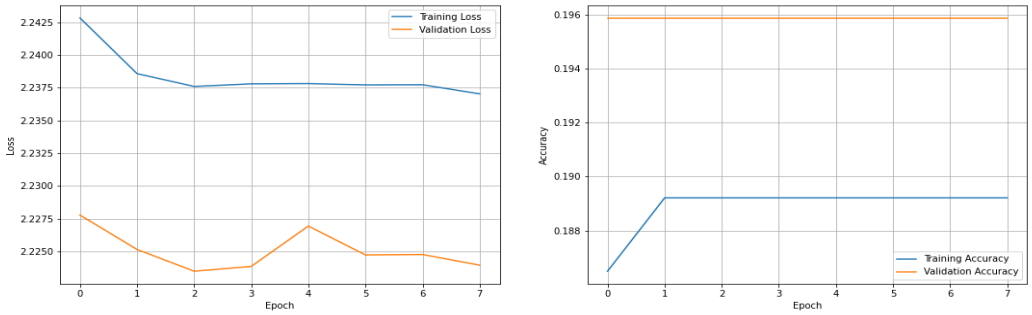


test set accuracy of only ~20%, where it predicts all test set digits as class 1, most likely due to the relatively low number of training images. The VGG16 model using pre-trained weights performs well with a test set accuracy of 57%. The precision ranged from 47-58%, and the recall ranged from 41-80%. The Custom CNN model performed the best with a test set accuracy of 88%. The precision for this model ranged from 80-91%, and the recall ranged from 78-93%.

VGG16 PreTrained (Transfer Learning)



VGG16 from scratch



Below are Classification Report results which summarizes the test set performance and errors associated with the Custom CNN and PreTrained VGG16 models. The poor performance of the VGG16 model trained from scratch was also reflected in its classification report, so it is omitted from this Report.

VGG16 PreTrained

	precision	recall	f1-score	support
0	0.68	0.58	0.62	1744
1	0.61	0.80	0.69	5099
2	0.51	0.76	0.61	4149
3	0.47	0.41	0.44	2882
4	0.58	0.47	0.52	2523
5	0.57	0.41	0.48	2384
6	0.62	0.42	0.50	1977
7	0.63	0.49	0.55	2019
8	0.60	0.55	0.57	1660
9	0.59	0.35	0.44	1595
accuracy			0.57	26032
macro avg	0.59	0.52	0.54	26032
weighted avg	0.58	0.57	0.56	26032

Custom CNN

	precision	recall	f1-score	support
0	0.90	0.87	0.88	1744
1	0.91	0.93	0.92	5099
2	0.91	0.90	0.90	4149
3	0.85	0.83	0.84	2882
4	0.89	0.89	0.89	2523
5	0.86	0.89	0.87	2384
6	0.84	0.82	0.83	1977
7	0.88	0.89	0.89	2019
8	0.83	0.78	0.81	1660
9	0.80	0.84	0.82	1595
accuracy			0.88	26032
macro avg	0.87	0.86	0.87	26032
weighted avg	0.88	0.88	0.88	26032

(a) Results

The results using the final pipeline applied to five images of houses are shown above. The results of this pipeline when applied to a video of a housing digit surface can be found at the following link:

https://gatech.box.com/s/mt2d64313hoxel2awgwejimhtde448b2

In the Image set and the Video, the bounding boxes were nearly always localized around the individual digits. However, these digits were not always classified correctly. Additionally, some false positive detections tend to occur to some degree in each of the images and video.

(b) Analysis

Multiple images show true positive detections of digits on the houses. The scale, location, pose, or orientation of the digits in the images does not appear to be an issue as digits at different scales, locations, pose, and orientations were classified correctly. The font of the digits in the house images are also different and can be classified to some degree as correct. One of the house images has the digits in shadow with a lower lighting condition than the rest, and the pipeline correctly classified several of the digits in the image.

The false positive detections tend to show non-discrimination between similar digits. For example, the classifier predicts an 8 when the ground truth is a 3 or a 5, or it predicts a 7 when the digit is really a 2 or a 1. These misclassifications could be due to the different fonts of the digits in the training set vs. those seen on the house images, as they appear similar in that font style. There are also false ROI detections from the MSER algorithm that are then classified by the CNN model. For example, several windows or panes of glass in a door are localized as ROIs and then classified. In addition, there are several times in the images where multiple bounding boxes appear over the same digit and the classifier predicts different digits for these boxes.

Conclusions

State-of-the-art (SOTA) methods such as YOLO [5] [6] could localize the digits and classify them with greater accuracy and lower false positive detections. This method is scale invariant and location invariant. This technique is known to have difficulty detecting objects in low-light conditions [8].

The most recent SOTA method for Object Detection is the YOLOR algorithm [9]. It is a unique method in that it encodes both implicit and explicit knowledge at the same time. Using the COCO dataset, it pre-trains over the various related tasks to approximate the implicit knowledge so as to learn a general representation. It then optimizes for specific tasks (representing the explicit knowledge), and both of these representations are used for inference. YOLOR has shown to perform much faster than the YOLO family of algorithms by 88%, and has demonstrated improvements in most areas of Object Detection tasks compared to the YOLO family.

There are many avenues for improvements to the algorithm used in this project. One area of improvement would be to replace the use of the MSER algorithm for localizing the ROIs that are used to define the bounding box locations. This technique could be replaced with the use of an Image Pyramid in combination with the Sliding Window technique to generate sections of the image for an ROI detection. The ROI detection could be done by threading these image sections into a Binary Classifier ("digit" and "non-digit" classes) to identify them as likely containing a digit. These ROIs could then be passed through to the classifier to classify which specific digit the ROI contains. This technique could potentially reduce the number of false positive detections made when compared to using MSER, but would also come at a much greater computational expense.

Another idea is to use Non-Maximum Suppression to assist in reducing the number of overlapping bounding boxes detected around each digit. This technique was explored for a simple binary classification toy model, but was not able to be extended to apply to the multiclass classification problem at hand (e.g.,multiple bounding boxes centered around a single digit could have multiple class predictions, so an averaging/smoothing technique would be needed to derive a single class prediction). With more time, this approach would be feasible.

Finally, training a digit classification model with an additional "None Class" could be used instead, which would potentially reduce the number of false positive detections in the image. This method was experimented with, and it was observed that a large number of false negative detections were produced (e.g., actual digits were frequently classified as "None" class). Because of this, this method wasn't chosen for the final pipeline. A possible fix to this problem would be to pay closer attention to how the "None" class samples are generated for the training. Ensuring that the "None" class training samples are similar to actual non-digit features encountered in the dataset (e.g., brick, siding, windows, grass, etc.) would be prudent and may help reduce the false negative detections generated.

References and Citations

[1] Girshick, et. al., “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”, 2014. http://arxiv.org/abs/1311.2524

[2] Girshick, R. “Fast R-CNN”, 2015. http://arxiv.org/abs/1504.08083

[3] Shaoqing, et. al., “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, 2016. http://arxiv.org/abs/1506.01497

[4] Liu, et. al. “SSD: Single Shot MultiBox Detector”, 2016. http://arxiv.org/abs/1512.02325

[5] Redmon, et. al. “You Only Look Once: Unified, Real-Time Object Detection”. 2016. http://arxiv.org/abs/1506.02640

[6] Redmon and Farhadi. “YOLOv3: An Incremental Improvement”. 2018. http://arxiv.org/abs/1804.02767

[7] Matas, J., et. al. “Robust Wide Baseline Stereo from Maximally Stable Extremal Regions”. BMVC 2002.

[8] Chen, et. al. “Learning to See in the Dark”. CVPR 2018. https://arxig.org/abs/1805.01934

[9] Wang., et. al. “You Only Learn One Representation: Unified Network for Multiple Tasks”. 2021. https://arxiv.org/abs/2105.04206