Data Visualization

Project Title: Supervised Encoding Networks

Project Number: 15

Group Members:

Surname, First Name	Student ID	STAT 442	STAT 841	Your Dept. e.g. STAT, ECE, CS
Simmons, Rees	rsimmons	☑	☐	AMATH
Mall, Sunil	sk2mall	☑	☐	COMP. MATH
Wu, Mohan	mhwu	☑	☐	STAT
Wang, Peter	p59wang	☑	☐	CS & STAT

Your project falls into one of the following categories. Check the boxes which describe your project the best.

1. ☐ **Kaggle project.** Our project is a Kaggle competition.
   - This competition is active ☐ inactive ☐.
   - Our rank in the competition is ... .
   - The best Kaggle score in this competition is ... ..., and our score is ....
2. ☑ **New algorithm.** We developed a new algorithm and demonstrated (theoretically and/or empirically) why our technique is better (or worse) than other algorithms.
3. ☐ **Application.** We applied known algorithm(s) to some domain.
   - ☐ We applied the algorithm(s) to our own research problem.
   - ☐ We tried to reproduce results of someone else's paper.
   - ☐ We used an existing implementation of the algorithm(s).
   - ☐ We implemented the algorithm(s) ourself.

Our most significant contributions are (List at most three):

(a) .Developed a method for object detection

(b) .

(c) .

List the name of programming languages, tools, packages, and software that you have used in this project:

.

Keras, Tensorflow, Scikit-learn, Python, Numpy

Abstract

Object detection in images is a problem that has been researched for many years. Finding a method to perfect this task can be very beneficial to a wide variety of uses. Currently, a popular technique for object detection is the bounding box method. This technique looks for rectangular sections of an image that resemble the object of interest. An issue with this method is that it is difficult to calculate the bounding box for images outside of the training set. We propose a different method altogether; we use a convolutional neural network to take the original image as input, and output the object from the image. The method introduced in this paper can be used as a general preprocessing step in image classification tasks. In applying this method as a preprocessing step, more focus is placed on classifying the object of interest (the cropped image)-- this is beneficial as the classifier is not negatively impacted by noise in the image. We evaluate the performance of this method on a real dataset.

1.0 Introduction

The motivation for the creation of the techniques discussed in this paper stems from "The Nature Conservancy Fisheries Monitoring" Kaggle competition. This competition requires contestants to classify species of fish in very noisy photos. The data is noisy as the photos include more than just the fish themselves. The images are real photos of fisherman out in the sea catching fish, so there are various objects in the images such as people, boats and fish. To make matters worse, the photos are not taken professionally and are often blurry. This makes classifying the images based on what species of fish is in the image extremely hard. It is clear that we could help the classifier by preprocessing the dataset in getting rid of the noise. Our goal then became to crop out the fish from the images to feed to the classifier instead. To do this, we explore methods of object detection in the images. The original problem at hand is a classification task, but our approach introduces another one; we need to take the entire image as an input, and obtain the crop of the fish as an output. From this we decided to investigate the new problem further, in hopes of learning more about and perhaps, contributing to, the field of object detection.

Annotations for the bounding boxes of the object of interest, fish, in all the training images are provided. The bounding boxes are used to create crops of the fish in the images which serve as the desired output. Rather than looking for the coordinates of a bounding box in the images, we propose a different idea altogether: we want to construct the fish directly from the image. The idea is inspired from autoencoders (Baldi, 2012) where instead of trying to reconstruct the input images, we stop once we have constructed the cropped out fish. Throughout the course of this paper, when addressing network output as a 'crop', what is really meant is the generated fish image. We refer to the construction as a crop for convenience. To train the model, we feed the model with both the image and the cropped fish. The hope is that the model would pick up the features of the fish to construct out of sample images. We evaluate our model on the testing set to see how close our construction of the fish is compared to the fish in the photo. Since the pixels in the output image can take on a continuous range of values this could be considered a classic regression problem.

We begin our paper in section 2.0 by providing the theory behind our proposed network, the Supervised Encoding Network (SEN). We look at the resulting output of our model when trained on the National Conservancy Fisheries Monitoring (NCFM) dataset in section 3.0. We conclude the paper with a summarization of our results, as well as the next steps, in sections 4.0 and 5.0 respectively.

2.0 Methodology

2.1 Data

As mentioned in the introduction, the main dataset of interest is the NCFM dataset. The target objects in the images are fish. There are 3,459 images in the data set, which is a relatively smaller dataset for image processing (when compared to other datasets such as CIFAR-10 or MNIST). Unfortunately, the images in the dataset are fairly noisy as some are taken in a shade of green and in various sizes. To simplify this problem, we scale all the original images to one size: 128 by 128 pixels, and convert all RGB images to grayscale. The original images are on average 512 by 487 pixels but the variance of the sizes is large. The target output images are also scaled down to 32 by 32. These output images are normalized such that value of the pixels are between 0 and 1. These decisions are made to account for computational constraints, so the network would train faster. When actually viewing the output, one must scale each pixel value by 255 so it can be viewable.

2.2 Network Architecture

As described in the introduction, the goal of SENs is to crop an image from the original image. Since this is a task surrounding images, we used convolutional layers throughout the network. This is due to the fact that they have been proven to be an industry standard for image related tasks, as noted from various achievements in the ImageNet challenge (VGG (OxfordNet), AlexNet, etc).

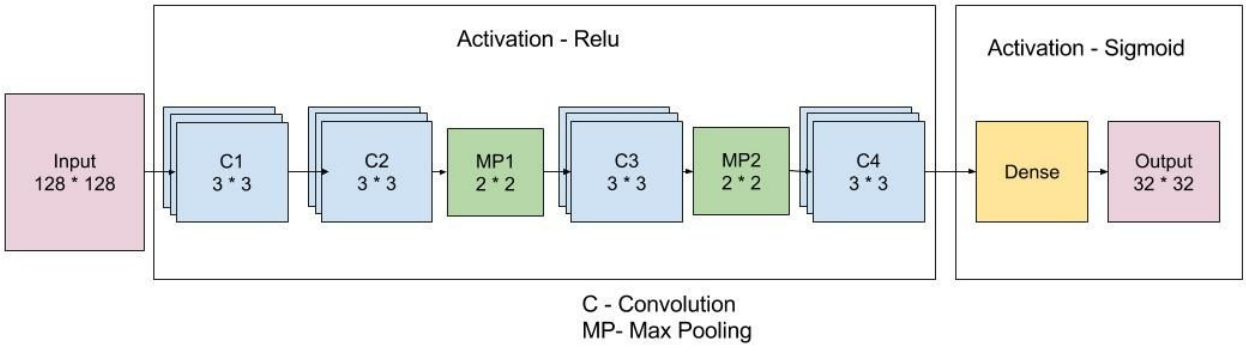

Figure 1: Supervised Encoding Network (SEN). All convolutional layers in this network use the ReLU activation. The first convolutional layer (C1) has 16 filters. The rest of the convolutional layers use 8 filters. Each time a layer's output is passed to a max-pooling layer, the output size becomes halved with respect to each dimension. The final fully-connected layer, the dense layer, uses the sigmoid activation function.

We construct this architecture because we are interested in outputting an image rather than classifying an image. Since the input and output of the network are both images it makes sense to only use convolutions and pooling in the intermediate layers. It is clear from the diagram and figure caption (Fig. 1) that the network is quite shallow. Since the dataset is small with eight classes, we decided to keep the network shallow to speed up training time, as deeper networks take more time to train.

ReLU is used as the activation function because it generally leads to faster training (Krizhevsky, Sutskever, & Hinton, 2012). The final layer is a dense layer with a sigmoid activation. The dense layer essentially combines all the activation maps from previous convolutional layers into one 32 by 32 image. The sigmoid activation at the end is very important to the network; it smooths the output (sigmoid is a smooth function compared to ReLU) and forces the pixel values to be between 0 and 1, which is the same as the input.

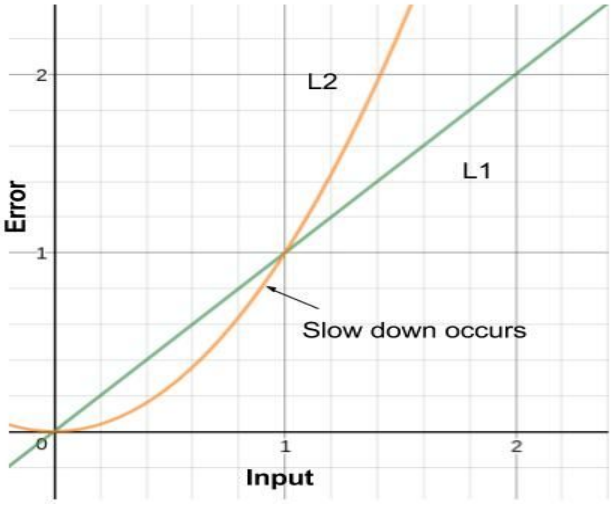

Figure 2: L1/L2 Loss function comparison. L2 loss function

L1 loss is used to compare the similarity of the output to the target image. We started with L2 loss but the output from the network is noisy (grainy). There are a few possible explanations. If we look at the L2 loss as a least squares estimation problem, then the assumption on the data is that the per-pixel variance is a gaussian distribution. When gaussian error is added to any image the output looks noisy. This could account for why L2 loss remains to be noisy even as the network becomes well trained. The second issue with L2 loss is that it causes a massive slowdown in learning rate. As mentioned before, the output and training data's pixel values is normalized to [0, 1], so when L2 is applied, it measures the per-pixel difference between two numbers between 0 and 1 which becomes very small when they are squared. This means that the L2 loss is very small and thus training is slow. L1 loss does not have this issue as it's derivative is constant and thus no slowdown occurs. Additionally, L1 enforces a "sparse" representation which in images more contrast and sharper edges.

The next step is to reduce overfitting on the training set. When we first implemented the model without any regularization, the model overfit the training set very quickly. To do this, we

introduce regularization in our architecture. First, we add L1 weight regularization on each of the convolutional layers. The results is a lot better on the testing set with regularization than without. We further explore regularization in other ways. We add a dropout layer after two convolutional layers. After fine tuning the parameters of the dropout layer (dropout value of 0.2), the model outputs more accurate results.

To increase our training speed, we also include batch normalization layers. Batch normalization allows the model to have a faster learning rate without the risk of the model not converging properly (Ioffe, & Szegedy, 2015). This will increase computational efficiencies and reduce runtime. Batch normalization works by shifting the mean to 0 and the variance to 1 in between layers, which is very similar to the standard normalization we are familiar with. The difference is that we have normalize the data after each convolutional layer within the batch normalization layer.

3.0 Results

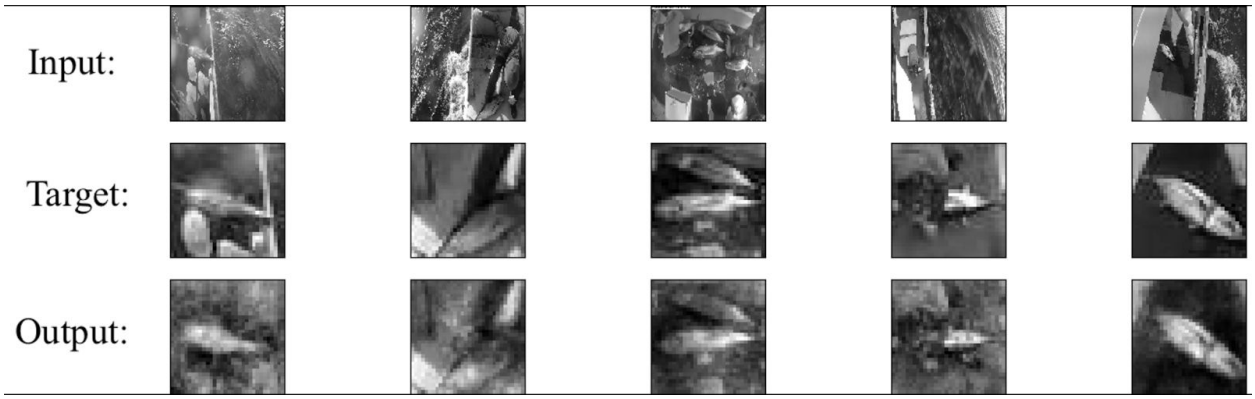

Figure 4: Training results. The first row is comprised of the input images, which include background objects such as the boat, humans, tables, and much more noise. The middle row (Target) consists of the actual crops of the fish from the images. The results can be found on the bottom row (Output). Our algorithm learns to extract the fish from the training images.

As can be seen from Fig. 4, the network was able to identify and generate the crop of the fish from the training set. Since the network is repeatedly given the image for which it is supposed to crop the fish, as well as the actual crop of the fish, this is to be expected. The network was trained for 100 epochs with a batch-size of 50. To evaluate the network's performance, one must view its output from the test set. As can be seen in Fig. 5, the network was able to successfully identify and construct the crop of the fish from the original image. The produced image crops are surprisingly clear given the few number of epochs. The images for which results are displayed in Fig. 4 & Fig. 5 are chosen at random.

Figure 5: Test results. After training our network on the training images, we passed it the test set of images. The first row is comprised of the original images (Input). The middle row (Target) consists of the actual crop of the fish from the image. The results can be found on the bottom row (Output).

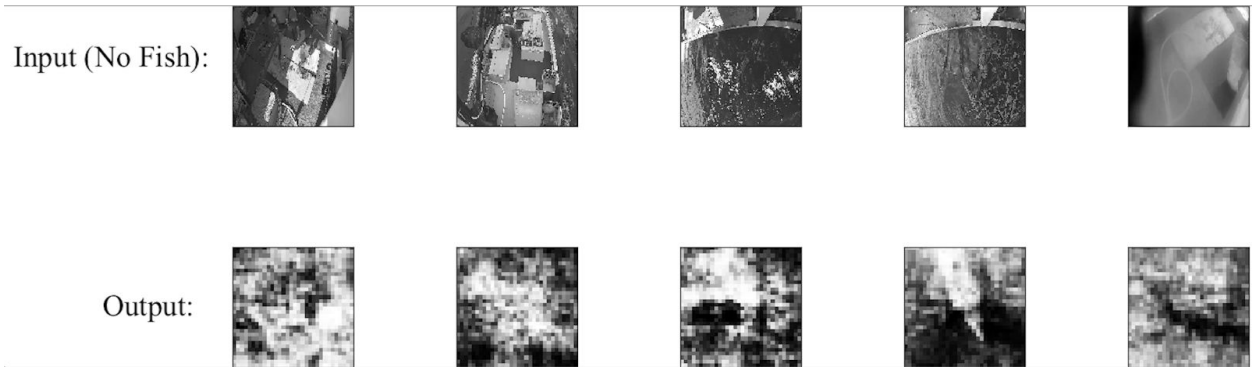

Figure 6: Viewing network output when passed images without fish.

The SEN was trained with only positive example (images with fish) so it was important to see what would happen in the negative cases. Since these images do not have fish it would not make sense if the output has any fish-like features. This prediction is confirmed by the results from Fig 6., as the output has little structure. It is worth noting that these images are jagged in comparison to the generated output in Fig.4 & Fig. 5.

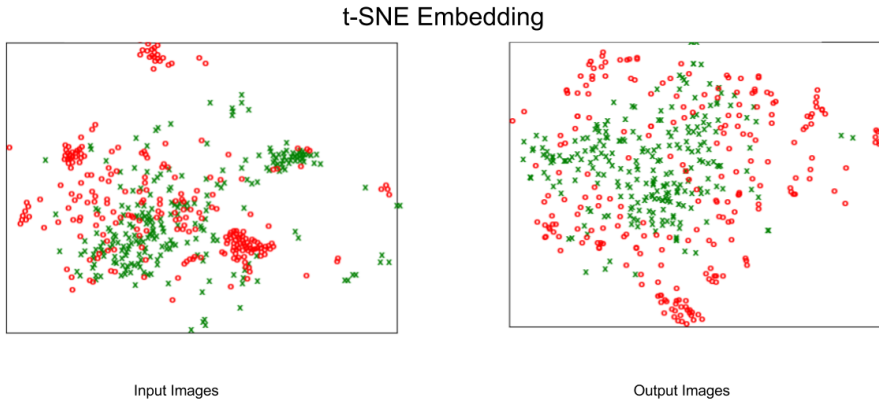

Figure 7: t-SNE clustering of our fish vs no fish images. The first graph is the lower dimensional embedding of the input images before going through the network. The second graph is the lower dimensional embedding of the output of the network (the crops). The green represents the fish class and the red represents no-fish class. These images are generated from the test data.

The output from Fig 7. shows that the learned features are meaningful. This can be see by observing the differences in the two plots. The clusters in the first plot are clearly indistinguishable. The second plot however has two clear clusters. For the most part, crops containing no-fish cluster around the crops containing fish. The direct objective of the model was not to distinguish fish vs no-fish clusters, but it was expected and confirmed that the output could be separable.

4.0 Conclusions

We have shown that framing object detection as a regression task is a plausible and surprisingly successful strategy. The SEN architecture is quite extensible since the only main requirement is that it maps an image to a reconstruction of the region of interest in the image. We have seen the network reconstruct the fish in the training set, and it generalized well to the test set. Input containing no fish led to output that was very unstructured and noisy. This noisy output could be leveraged to determine whether or not a fish is present in the photo, assuming we did not know much about the input. The results obtained in this paper were achieved with a shallow network (when compared to VGG16 or AlexNet) that was trained on a CPU for 100 epochs, with a batch-size of 50. We expect results to improve with a deeper model, but we must be careful not to overfit the data when training a deeper network.

5.0 Future Steps

Though the model showed promise there were noticeable improvements that could be made. When investigating the intermediate convolutional layers of the SEN it was clear that many of the convolution filters were capturing simple features like edges. Only later in the model did the convolutions represent more complex structures like fins, tails, etc. Using layers of VGG-16 would most likely increase the accuracy of the network since the beginning layers of VGG-16 are pre-trained to features like edges and surfaces. As well, fish were one of the classes in ImageNet so there are mostly likely a few filters that capture fish features.

The greatest weakness of the SEN is relying on L1 as the image comparison function. It managed to train well, however there are better possible ways of computing image similarity. We propose adding a discriminator to the end of the SEN which is trained to recognize images of fish, thus making it an adversarial SEN (ASEN). The goal of the ASEN is to find the target in the image in order to 'trick' the discriminator. This goal is differs from a traditional GAN because it is not just learning a low-dimensional embedding for the data. In tricking the discriminator, higher quality fish reconstructions would be created.

The ASEN would have an interesting advantage over the SEN when it comes to training data, which can be explained via an example. Imagine trying to train an ASEN to find bananas in images. Suppose bounding boxes for the bananas in the images are not given. If we had a pretrained banana-image discriminator, then the ASEN would theoretically 'learn' to find the bananas well enough to 'trick' the discriminator. However, there would be issues to be worked on when it comes to the possibility of overtraining images without bananas in them.

References and Notes

Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105 (2012). URL http://papers.nips.cc/paper/4824-imagenet.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015). URL http://arxiv.org/abs/1502.03167.

Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. Journal of Machine Learning Research (2011) . URL http://proceedings.mlr.press/v27/baldi12a/baldi12a.pdf