CSCD94
Weakly Supervised Deep Learning for Object Localization
Project Report

University of Toronto Scarborough
Daerian Dilkumar, Venkat Korapaty

Supervisor: Francisco J. Estrada

April 2018

Contents

Section 1

Introduction

1.1 Overview

Our intent with this project as avid delvers into artificial intelligence was to take a shot at doing a form of statistical learning that is applicable in the current state of the industry. We were excited to get our feet wet with real data and solving a problem outside the setting of the classroom, that we would be able to tackle on our own.

1.2 Goals

Due to the fact that convolution neural networks have made great strides in the world of computer vision, we wanted to tackle a problem that would also allow us to keep up to date with the modern techniques employed in the field. Our idea was that by working with the current architectures that are being used in the sector, we would be able to get our feet wet and truly comprehend the nature of machine learning employed.

We can use deep neural networks to figure out whether certain objects are present in an image. However, we also wish to figure out where the object is located in the image. We will be using state-of-the-art Convolutional Neural Networks. The outputs of convolutional layer after pooling provide features that we can use to locate objects. These CNN layers are trained through repeated localizations done by the convolutions and max pooling layer, mapping to a set of feature maps. These feature maps are what the CNN layers decided the most important pieces of information (or rather, how to calculate them) are to tell us what objects are in the image. See figure 1.1. Our research project is based mainly on the paper, "Weakly Supervised Localization Using Deep Feature Maps"[1].

1.3 Architecture and Setup for CNN

A modified version of the AlexNet architecture[5] is what we will be using. It has convolutional layers C_1, C_2, ..., C_5, max pooling layer M_1, M_2, M_5, and fully connected layers F_6, F_7, F_8. The only main difference is that our fully connected layers are of size 2048 instead of 4096. This is mainly due to time/space constraints we deal with. As it so happened, we were working with limited memory and processing power, i.e 64 Gb of RAM, and six cores each running at 2.7 GHz of processing power. With these resources each attempt at training took roughly 32 to 33 hours. Attempting layers of double the size increases the time proportionally.

All our construction of the network and localization was done using Tensorflow. The weights of the network are learned using Adam Stochastic Gradient optimization[4]. We want to take advantage of optimizers that can use momentum and an adaptive learning rate. We use an initial learning rate of 0.001. As we have multiple potential classes for a single image, we use the sigmoid cross entropy loss function.

$$J = -\frac{1}{K}\sum_{k=1}^{K}[p_k log(\hat{p}_k) + (1-p_k)log(1-\hat{p}_k)]$$

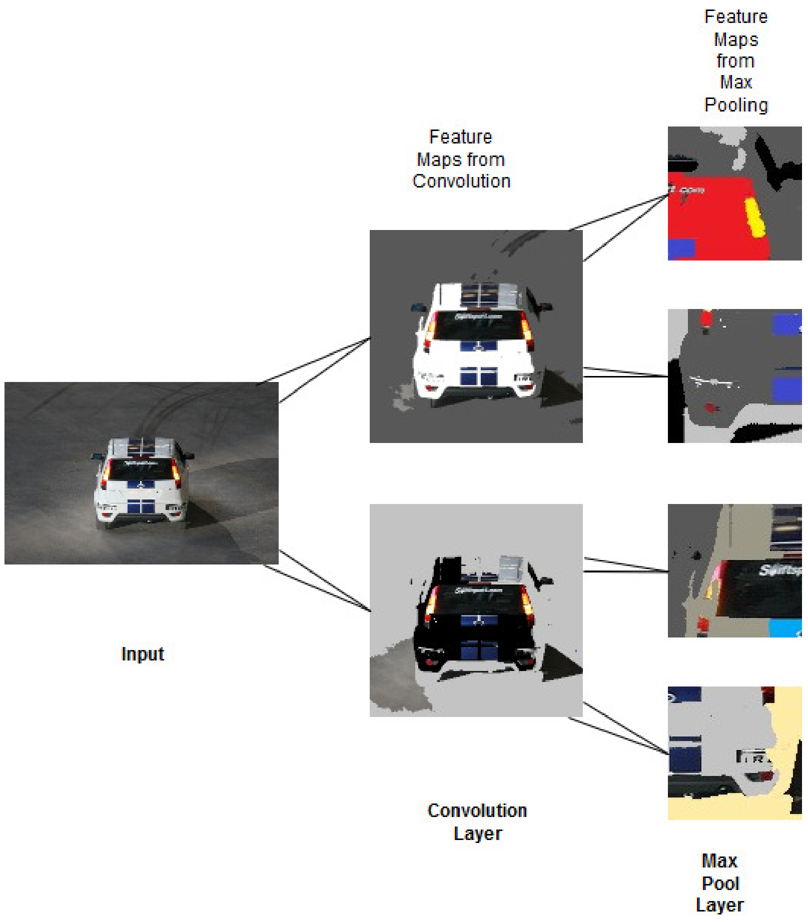

Figure 1.1: An example of Feature Maps from a Convolutional layer with Max Pooling

The layer trains to figure out what patterns are important to achieve our predictions. It creates features from the convolutional, and from there it applies max pooling to see which portions seem most important, and creates another set of useful, rich features.

To deal with overfitting, as part of the AlexNet architecture, we use the well known method of dropout[7] on our fully connected layers F_6 and F_7. We also used l1 regularization with a scale value of 0.0005. We also had issues with our logits giving the same output values with different batches. To deal with this, we used batch normalization[3] on layers C_3, C_4, F_6 and F_7.

During training, we would use a batch size of 100 and run for around 8 epochs. We used our cross validation as test data, and we would have to use batches of 250 for testing. Unfortunately, due to the limited time for the project, and the problems described in section 2, and our limited RAM causing us an inability to load all the test-data. we were limited to using the following changes:

1. The use of CV Data as Testing Data.
2. A run of 8 epochs.
3. Run the CV data in batches as well as the training data.

Section 2

Dataset

At first, we trained our neural network on the MNIST dataset for the purpose of making sure AlexNet was implemented properly. However, as our main purpose was to do localization, the dataset we used was the PASCAL VOC 2007[2] dataset which contains 20 possible classes, 2501 training, 2510 cross validation, and 4952 test images.

We face a lot of problems using this dataset due to the limited resources we had. The images in the PASCAL VOC dataset would be around 500x300 (or 300x500) to 500x500 in length and width, while also being of 3 channels (ie: RGB images.) We did not have the resources for that. We also have to make the input a fixed size so we can feed it into the network. The scripts we used to load all the JPEG images and labels provided to convert into NumPy arrays so we can use with Tensorflow are in the GitHub repository for this project.

2.1 Pre-processing the Data

Our initial idea after realizing we did not have sufficient resources was to pad the data so they're all 500x500 images, then do nearest neighbour interpolation[6]. This takes up about 500x500x3x2501x8 = 15006000000 bytes = 15.006Gb of space with a NumPy array for the training data alone before interpolation. We wanted to reduce the image size so they're all square images, 227x227.

Given our memory problems, we reached the conclusion that the best way to pre-processing was to load each image crop it by which ever dimension is smaller (i.e 400x500 becomes a 400x400 image, a 500x330 image becomes a 330x330 image). Then, blur the image using the Gaussian Blur functionality in the PIL python library with a standard deviation of 1. After blurring, we use nearest neighbour interpolation to reduce the image to a 227x227 image, which we then saved as a NumPy array. This approach is what we finally settled on.

Two things to note: 1) When we crop the data, we risk losing some important information but most images have their objects roughly around the centre so it should not have too poor of an effect. 2) We blur before nearest neighbour interpolation to prevent sharp segments in an image from blowing up parts of the image.

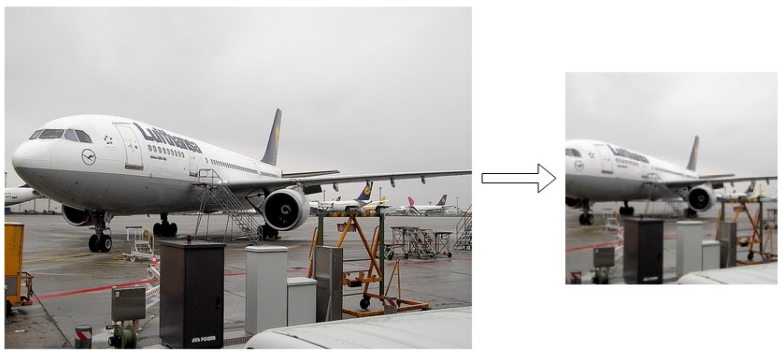

Figure 2.1: An image of class airplane from PASCAL VOC 2007 dataset

The image before and after it's pre-processed; we can see it loses some but not too much important information.

Section 3

Main Strategy

Our over-all strategy was a to create and train a Convolutional Neural Network using a modified version of the AlexNet architecture, in addition to our pre-processed dataset, PASCAL VOC 2007. Then, use this trained net to help us perform Localization.

We use two main approaches for localization: a greedy strategy and a slightly smarter one that uses beam search. While greedy is generally faster, it doesn't always provide the best result. So, we also adapt the beam search strategy as presented in [1].

AlexNet's final max pooling layer produces 28x28x256 feature maps, which we use to create a localization candidates. With the output of the final pooling layer, we break it into 4 candidates each with a row or column missing as follows: one with the top row, the bottom row, left column, and right column missing. Each of those are then zero-padded so it can be fit back into the fully connected layers. We choose to pad instead of reshape, as reshaping may introduce interpolative artifacts. Interpolative artifacts warp the image, and introduce "fake" information from the "guessing" that is done to stretch the cropped image back into shape. Zero-padding meanwhile simply adds zeros to the proper sides of the image until it is successfully back to it's required size. The zeroes add no new information and preserve the cut while simultaneously allowing the image to be of appropriate size to one again be sent into the net.

With each of our 4 new candidates, we run those through the fully connected layers, and look at the probability of each one. From there, we use higher probabilities to see which are better candidates to continue from. For the plain greedy approach, we see which candidate gave us the highest probability of having the class we are looking for. From there, take that section of the image, pad it to feed into our network, and repeat this process.

The beam search is a generalization of the greedy approach. As proposed in the original paper[1], we first have a beam size of two. So, after we take our candidate from the final max pooling layer, we take the candidates with the two highest probabilities. From there, we run the greedy approach on both candidates in hopes that we find two objects of the same class in the image. Generalizing this to find all occurrences of a class is a tough problem.

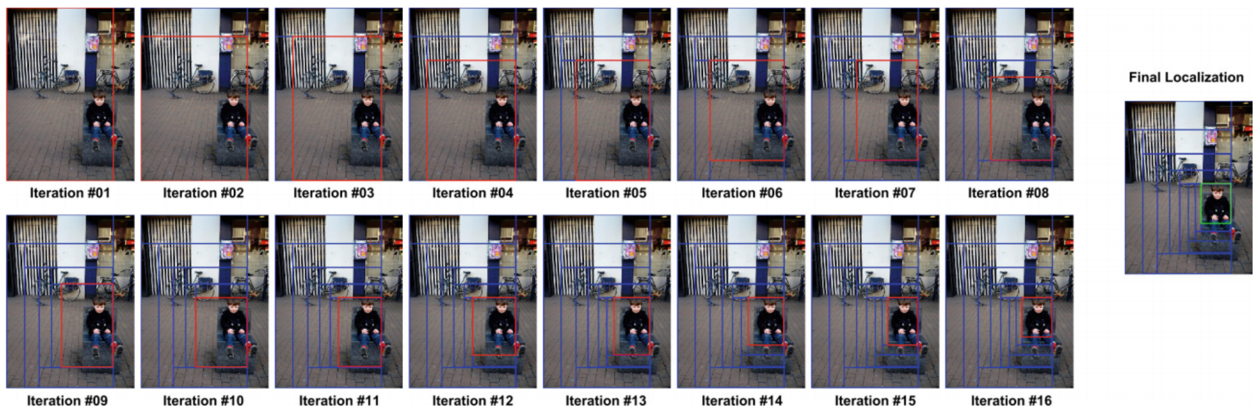

Figure 3.1: Diagram from [1]

The overall localization visualized in an example, for the class person.

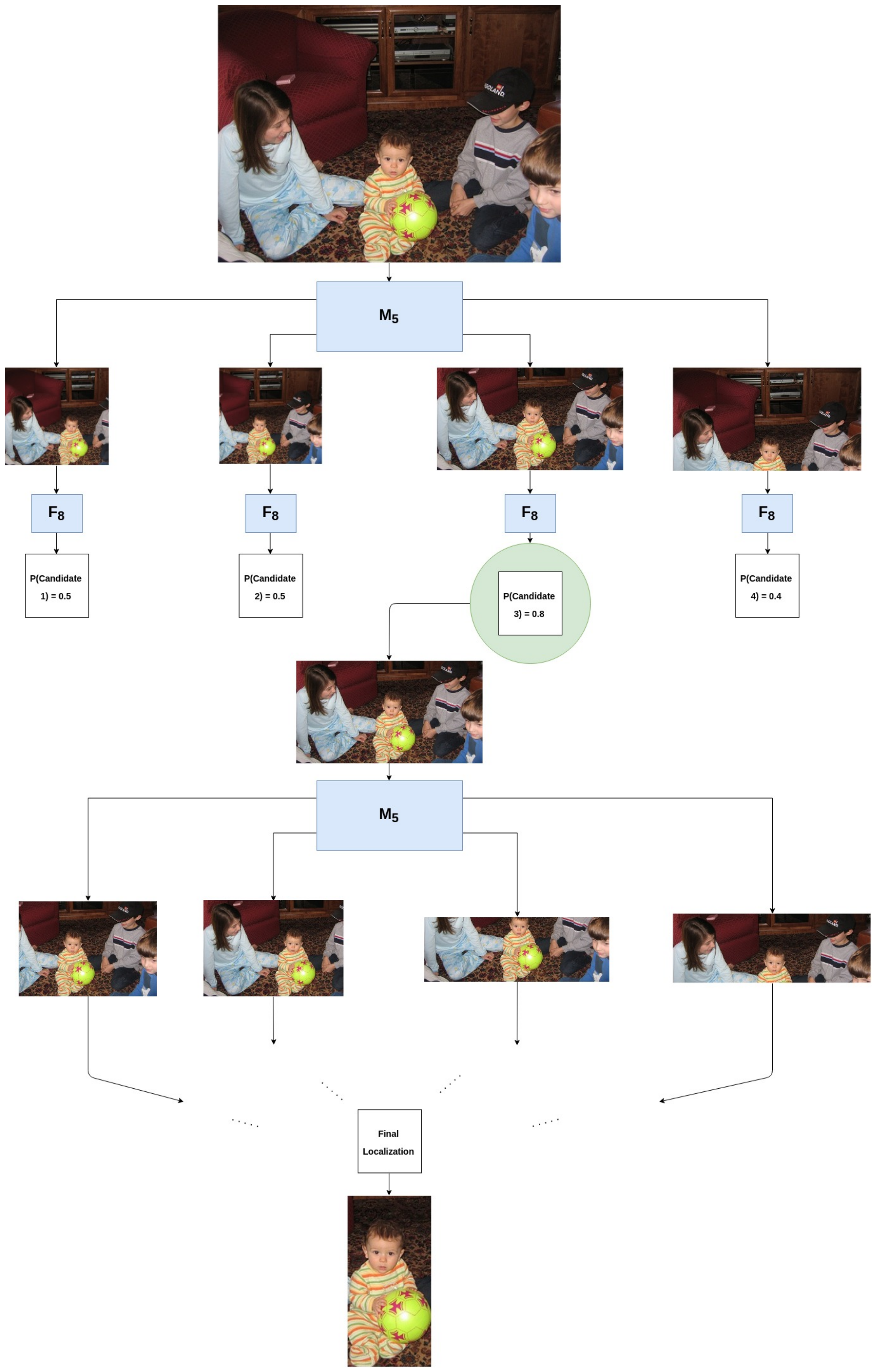

Figure 3.2: The intuition behind the greedy localization with many possible targets

This is the desired, but not actual, result of using the greedy approach we wished to achieve. As described, we take an image, use the output from M_5, which is 28x28x256 feature maps. We then make 4 candidates as described, and continue executing the greedy algorithm.

Section 4

Localization

Localization of images is a common occurrence in computer vision problems. Localization is the process by which an algorithm, given an image and an object to search for, finds the object in the area in the image containing the object. This process will "zoom-in" or "focus" on the object in question, thereby finding the "locale" of the object - or localizes the image. For the purposes of our project, we approach localization by using a heuristic searching algorithm called beam search. Beam search is a variant of the best-first search algorithm, except instead of ordering the solutions and trying them all, we choose only the top few.

4.1 Greedy: A Specific Case of Beam Search

The greedy approach is a case of beam search where we only take the best or optimal case for the search in each iteration. This approach is meant to minimize the amount of resources required for an execution. In particular it is focused on using the minimum RAM space, and only considers one possible candidate. However the issue with this approach is it might not always pick the best option in the overall run. This algorithm chooses the best option at the time of each iteration. However this can mean we leave behind a lot of information. Our algorithm uses a single beam when moving forward from the first iteration. Included below is an example of how this follows a single-minded approach. Our greedy approach follows the following steps:

1. Start with the original image.
2. Feed-Forward the image through to the fifth convolutional layer.
3. Create four prospective sub-images by slicing the far top, bottom, left, right row or column off the original image respectively. Therefore we have 4 sub-images, each differs from the original by one row or column but not both.
4. We pass these four images through the network to the final fully connected layer, and receive predictions for each of these "cut" images containing the object.
5. Using these probabilities, we take the image most likely to be the object, and "toss" the rest of them.
6. Using the chosen image as the new "original image," we repeat these steps as many times as we have iterations.

For a visual example of this process, please refer to figure 3.2. The green circle represents the image with the highest probability, M5 is the fifth convolutional layer, and F8 is the final fully-connected layer. This example asks the greedy algorithm to find where the humans are given the starting picture. The example shows how the greedy search accurately finds a human. However it ignores much of the information, and fails to recognize there are many humans, not just one. As a result, the result of the greedy search isn't a good representation of "where the humans are" in the picture.

Therefore we can quickly see how this approach might not be the best for accurate localization of some images. That said, for cases where the object is singular or close together, this approach works much faster and more efficiently than other approaches. An easy way to see this is with figure 4.1, which shows how the algorithm quickly finds the boy within the picture.

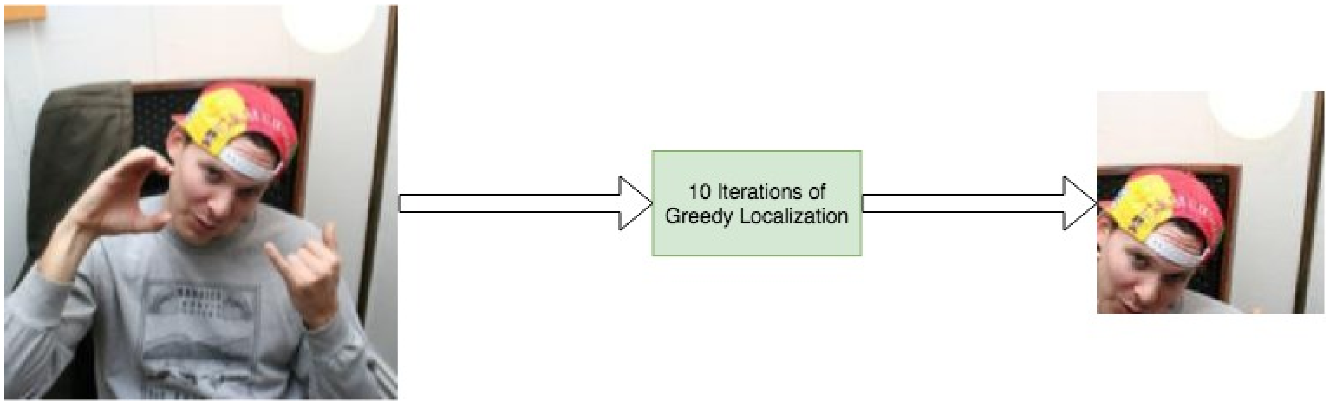

Figure 4.1: Output of our greedy localization for class person

The result of running our trained network with the greedy strategy with a test image.

4.2 Beam search

Now, we take a look at the smarter approach we mentioned earlier. Beam Search works in a way that is quite similar to the greedy algorithm with one major difference - we have multiple possible candidates to pursue. Beam search is still a heuristic approach, in fact, we choose a number K, and only pursue candidates who fall within the top K probabilities when compared to their batch-mates. In fact, beam search is generally used for help in sorting through searching through trees made for language processing and text-finding, since those libraries tend to be grandiose, and take up large amounts of memory. However, in this case, the constant processing of the image through the network along with it's constant splitting into four, also eats up RAM quickly. Therefore, we chose to use 2 beams in our beam search. A beam search is done in the following way:

1. Start with the original image, and a selected number of beams - we have chosen 2.
2. Feed-Forward the image through to the fifth convolutional layer.
3. Create four prospective sub-images by slicing the far top, bottom, left, right row or column off the original image respectively. Therefore we have 4 sub-images, each differs from the original by one row or column but not both.
4. We pass these four images through the network to the final fully connected layer, and receive predictions for each of these "cut" images containing the object.
5. Using these probabilities, we take the image most likely to be the object, store it, then select the second most-likely and store it, and so on, as many times as we have "beams." and toss the rest. NOTE this is where the Beam search differs from the greedy approach.
6. For each of the chosen images in step five, we pad them to be the size of the original image. We then refer to each of these candidates as "beams" as each will now be their own "original image" in our search.
7. For each beam in step 6, we repeat the five steps of the greedy algorithm, treating each beam as a unique greedy that has one less iteration than normal (since the first split into beams is in it self an iteration.)
8. After each beam has returned to us it's localization, we put it through our algorithm to find the appropriate area of first image passed in during step 1, and save these images.

An example of a result of such a beam search is Figure 4.2

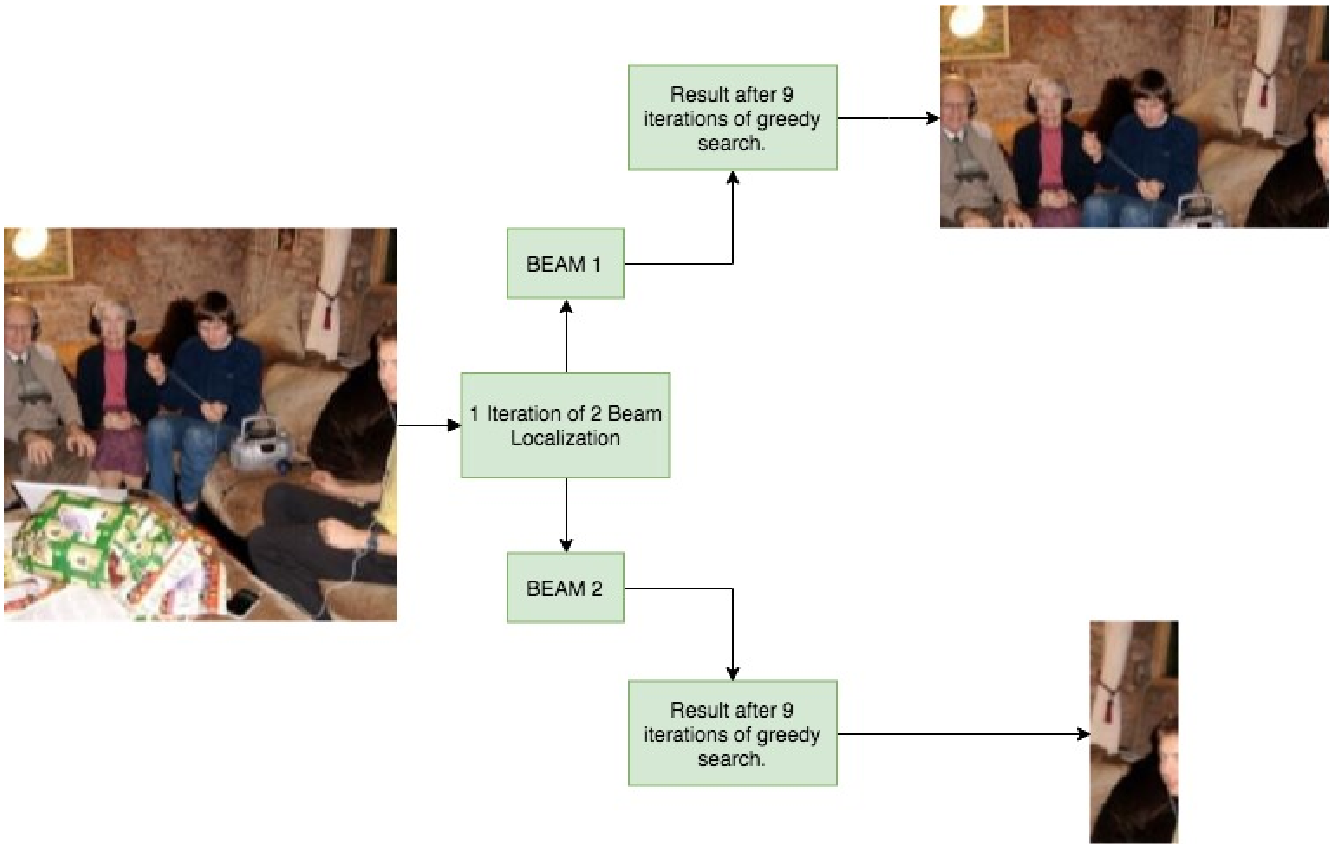
Result after 9 iterations of greedy search.
BEAM 1
1 Iteration of 2 Beam Localization
BEAM 2
Result after 9 iterations of greedy search.

Figure 4.2: Output of beam search localization with beam size of 2 for class person

The result of running our trained network with the beam search strategy with a test image (after 10 full iterations). The first beam was closing in on a group of people on the couch, the second beam closed in on a face on the right side of the image.

Section 5

Conclusion

5.1 Summary

When the neural net was working properly, we were getting accuracies of around 90-92%, but with poor recall and precision. If we had more time, we would be able to fix this. We originally had more poor accuracy, but the use of regularization fixed this.

Along the way, as we planned and built the network, we decided to perform a series of experiments, and found that there was a large difference in probabilities when candidates are created from the first pooling layer versus the final pooling layer. While both approaches will eventually find a suitable candidate, should it exist, we noticed that the first layer alone took much much longer to reach an answer, and due to the size of the image, all beams would tend to converge to a singular image, making beam-search mostly useless. The first layer variant would show much larger differences in probabilities as opposed to the fifth layer, and took longer to converge to a low-variance set of candidates.It was precisely due to this that no matter how many beams were used, the beam-search would mimic the greedy algorithm, with all beams concluding with almost exactly the same image. Thankfully however, using the final layer instead caused much larger discrepancies in each cut, and as a result each beam would find a different sub-set of the original image to focus on, while maintaining close probabilities with one another.

This approach has proved to run in excellent time, and come up with brilliant localization results. In conclusion we have found that with our hyper-parameters and strategy, the results turned out to be the best combination of accuracy and speed.

5.2 Comments

We expected to learn the theory behind convolutional neural networks, how to implement them, and how to perform localization. We successfully learned and implemented CNNs in TensorFlow. Since we used Python, we also were able to take the time to learn different libraries that were necessary, such as NumPy, PIL and SciPy to name a few. It was not just learning libraries, but we taught ourselves to learn more efficiently (no pun intended) when exposing ourselves to solutions that required tools we haven't used before. We learned to find resources we hadn't thought existed, and communicate with others to get resources we desperately needed. We also learned how to look for hints and create creative approaches to solutions.

There are quite a few things we for sure will take away and use in the future from this course. The ideas of CNNs, the use of TensorFlow and properly batching using a dynamic size dataset, using different Python libraries where appropriate, to name a few.

We also gained valuable experience in managing our time and ensuring development was going forward. In addition we learned to communicate with our supervisor well and learned about the value of resources, and gaining them in environments where they are scarce. As a result we learned (no pun intended, again) a few guidelines on choosing hyperparameters and minimizing our resource consumption - most importantly, our time.

References

Literature

[1] Archith John Bency et al. "Weakly Supervised Localization Using Deep Feature Maps". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 714–731 (cit. on pp. 2, 5).

[2] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html (cit. on p. 4).

[3] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167 (cit. on p. 3).

[4] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (cit. on p. 2).

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (cit. on p. 2).

[6] Olivier Rukundo and Hanqiang Cao. "Nearest Neighbor Value Interpolation". In: *CoRR* abs/1211.1768 (2012). arXiv: 1211.1768. URL: http://arxiv.org/abs/1211.1768 (cit. on p. 4).

[7] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 3).