CSCM10: Specification

Andy Gray
445348

8/05/20

Swansea University
Prifysgol Abertawe

Abstract

Abstract here

1 Introduction

We have presented to us, a challenge to navigate an office space which contains red dots. The office space is in a U shape, which is requiring our solution to take an image from a starting location, decided where, within the image, the algorithm believes where the image is and then move to that location to collect the dot. Our algorithm, when we have reached the end of the corridor, needs to search the room to find the next dot manually by changing its angle. We will be using training data that has been provided to us to train a Convolutional Neural Network (CNN). As well as implementing a sliding window (SW), to extract the patches from the game worlds image cloud, to figure out where the algorithm believes the most likely place the dot is, then moving to that location.

Deep convolutional nets have brought about incredible breakthroughs in processing images, video, speech and audio [6]. Since its first implementation introduction in 1994 [8], but the idea was first proposed in 1989 by Yann LeCun [9]. He then went on to integrate Rumelhart et al.'s [11] backpropagation into his previous work [7], which then developed into what we know now as CNN's. CNN's are more straightforward to be trained due to the fewer connections and parameters [5].

We aimed to find out if changing the parameters to the CNN filter would have any effects on the speed and performance of the overall challenge. We wanted to figure out if changing the size of the mask provided any speed benefits that would outweigh the decrease in the model's accuracy. We will also be checking if increasing the number of steps within the SW, which would increased speed performance with extracting the patches from the image, does not affect performance in completing its task of detecting the potential location of the red dot in the image.

The results found that the 3x3 filter with a two-step SW was the quickest in all of the experiments. Showing that the trade-off between increasing the mask size and SW step size did not provide a justifiable trade-off, as not only was accuracy reduced with a 6x6 filter, the overall completion time of the program completing was longer. The quickest being 3x3 filter with 2 step SW taking 637.75 seconds and the longest being 6x6 filter with a two-step SW taking 1486.03 seconds.

2 Methodology

2.1 Motivation

The motivation for this report was, once we had created a working solution that collects all the balls, we wanted to experiment with parameters within the CNN and SW. Within the experiment, we are aiming to check if the speed of completing the challenge became faster with each increasing parameter, while still being able to complete the game. We aim to see what the best trade-off with regards to checking every part of the image and total completion time of the challenge.

2.2 Packages

We will be using the programming language Python 3 [10], as this allows us to use all the required additional packages needed. The additional packages we will be using are NumPy [13], OpenCV [2], Pptk, Matplotlib [4], SKImages [12], SKLearn [3] and Tensorflow version 2 [1].

2.3 Model Trained

For this experiment, we only used one model. The model was a convolutional neural network. We were going to make a performance comparison between an ANN and CNN, but we wanted to see what effect, especially the filter, has on the performance of the CNN.

The CNN we used was a sequential model. The first layer has a 2D Convolutional neural network with ten nodes, with a filter size, that changed depending on the set parameters required, with the input shape of the data, with the activation of 'relu' and a dropout of 0.2. The next layer used a 2D Max pool layer again with activation of 'relu' and a dropout of 0.2. The next layer, which is the dense layer first flattens, then dense it by 50 and uses a 'relu' activation. Finally, the output layer uses a dense layer of 2, as we have to labels to predict, positive or negative of the red dot, with a time activation of 'softmax'. We then compile the model using 'adam' and the 'categorical crossentropy' for loss function and accuracy for the metrics.

2.4 Hyperparameters

2.4.1 CNN

We set the CNN's filter to either 3x3 or 6x6 for this experiment. We also made sure the input data shape only focused on the dimensions 2,3 and 4, ignoring the first column as the required information for the patches were in the height, width and colour channels.

2.4.2 Sliding Window

The SW will scan the extracted image from the image cloud, this is due to the training of the CNN and with it is needing 51x51 images, but the provided image is of size 160x240. Therefore, by scanning the image in 51x51 sections, the patches can be fitted into the model. The SW will move by the number of steps that we request in the function call. We used the steps 2, 5 and 10 to see what impact these would have on the overall performance of the game.

2.5 Loading Data and Feature Extraction

We had the training data assigned to us from the start. It contained 308 positive images and 2600 negative images. Using CV's image read function, we loaded all the positive images into an array, along with assigning the labels, '1', into a new array within the same location. We then added the negative images to the array, assigning the negative label '0' as we did with the positive. These arrays created the features and the labels we required to train the CNN.

2.6 Patch Extraction and Sphere Detection

We used a provided function to acquire the required values of image, map1, map2, map3 and map4 from the points cloud. The image was a snapshot of what the game world was showing to the dimensions of 160x240, map1 was the information needed for the x-axis information, map2 was the y-axis information, map3 was the z-axis information while map4 was the depth information. However, because the image was too big for the CNN, we needed to use the SW on it to break it up into 51x51 patches. These patches we used to fit into the CNN, and the patch with the highest percentage probability position we selected. This position location we used to gain the x and y center points. With having the centre x and y location, we were able to map these values onto map1, map2 and map3 to extract the required information needed. Gaining the required information needed to move to most likely place the red dot will be.

2.7 Identifying new Camera Locations ans Scene update

The location of the patch with the highest percentage value is determined, along with its position in the array. The x and y values, to find the middle point of each patch, of the map1, map2 and map3 values. We use these to extract the position coordinates from the point cloud. We are therefore moving to that location. We then call a function that updates the scene. This function checks if the location is within the threshold of the location of the dot. If the dot is in that location, the points cloud is updated, and the dot gets removed. We keep repeating the process, each time we find the red dot, but we are making sure we are getting the updated image from the points cloud at the new current position each time. However, if we do not locate the dot, the position is reset to the previous location. While the angle's Y-axis gets changed by

a radian value of -0.75, this process of rotating will keep happening until the correct location of the red dot is established, returning to the usual process but at the new changed angle and position.

3 Results

	CCN Taining Time	2 Step Sliding Window	5 Step Sliding Window	10 Step Sliding Window
3x3 Filter	6.91 sec 7.11 sec 6.50 sec	708.19 sec 860.16 sec 637.75 sec	716.51 sec 861.87 sec 641.66 sec	1060.81 sec 881.87 sec 856.81 sec
6x6 Filter	9.85 sec 11.65 sec 9.35 sec	1486.03 sec 1035.58 sec 757.05 sec	767.54 sec 909.84 sec 753.31 sec	783.68 sec 881.36 sec 955.60 sec

Both the 3x3 and 6x6 CNN's had 100% accuracy while training, which fig **??** shows. The results of the CNN with a filter of 3x3 trained quicker than the CNN with a filter of 6x6. These outcomes were the same in all three trials. All the trials found all of the red dots, but the quickest was the 3x3 2 step SW's third trail, with a time of 637.75 seconds while the 6x6 2 step window was the slowest at 1486.03 seconds. All of the 3x3 with a 2 step window was generally the quickest from their runs. However, for the 6x6 filter with a 5 step SW, results were quicker than the other 6x6 results, but still not as fast as the 3x3 2 step SW. On average, we found the 6x6 filter took about 3 seconds longer to train the model.

4 Discussion

We can see from the results that the most effective method was to use a 3x3 filter within the CNN and SW that is stepping a step of 2. Even though the SW stepping two steps was the slowest, taking 0.01 seconds while the others took 0.00 seconds, the overall experiments completed the quickest. We believe this is due to the CNN training with higher accuracy results, and with the SW moving only two steps, was able to break the image into more patches allowing more chance of the patches matching up. Therefore, even though the SW took longer, with the more patches to be able to predict with, it meant that the program was less likely to make wrong predictions and having to keep searching around the room. It was, therefore, making the entire process a lot quicker. However, there were occasions, with a 6x6 filter that had either a 5 or 10 step SW, that the program kept circling, searching for the red dot until it got found. Sometimes after taking more than two full rotations till it found the dot. Showing that the dot was present within the first circulation, but it failed to predict the correct coordinates. This outcome could be due to the SW patches having part of the red dot equally in each patch, but not enough of it the gain the right centre coordinates to move to that location.

The program was able to complete all the required task of collecting all the dots. However, there were some results that we did not initially expect. In regards to the 6x6 filter, we expected that the accuracy of the predictions would be lower than the 3x3. However, we did not expect the overall time to take longer than the 3x3 filter. We acknowledge that more calculations get done in each filter, but by covering more of the overall image, we expected it to be of a similar time or possibly quicker. We believe more exploring into these parameters is needed by the author.

We can see that in every run, for each setting, the program was able to find all eleven dots within the map. However, we feel that there are other ways that we could achieve this, which could potentially improve the complexity and run time of the algorithm and speed up the overall process.

5 Conclusion

Overall, the CNN with the 3x3 filter and using a 2 step SW performed the best. In both terms of being the fastest to train and being the quickest to complete collecting

all the dots. Therefore, showing that the trade-off of steps and accuracy is not worth pursuing within this case. However, the 6x6 filter worked best with a 5 step SW. We believe that more exploring into the effect the CNN's dense layers and the parameters need to be experimented on, to see if increasing the dense layers will see a benefit in the increase filter size and over performance time, compared to the 3x3. Interestingly when the 6x6 filter increased its SW steps, the program finished quicker than when it did with a 2 step SW.

We believe more research needs carrying out into the trade-off between training time and the step of SW. Enabling to see if performance, by increase layers and density of CNN. While it would take the training time longer, but would it speed up the overall process? Especially when looking at the bigger filter and the higher SW steps, to see what the correlation between these two factors are.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[3] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013), pp. 108–122.

[4] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in science & engineering 9*, 3 (2007), 90–95.

[5] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[6] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[7] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation 1*, 4 (1989), 541–551.

[8] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[9] LeCun, Y., et al. Generalization and network design strategies. *Connectionism in perspective 19* (1989), 143–155.

[10] Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation, Vienna, Austria, 2020.

[11] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[12] Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. scikit-image: image processing in python. *PeerJ 2* (2014), e453.

[13] Walt, S. v. d., Colbert, S. C., and Varoquaux, G. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering 13*, 2 (2011), 22–30.