CSCM77: Computer Vision and Deep Learning Coursework

Andy Gray
445348

27/05/20

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, requiring our solution to take an image from a starting location, decided where within the image the algorithm believes is where the image is, move to that location and then collect the dot. Our algorithm, when the algorithm reaches the end of the corridor, needs to search the room to find the next dot manually by changing its angle. We will achieve this by using training data that has been provided to us, to train a Convolutional Neural Network. As well as a sliding window to extract the patches from the game worlds image cloud, to figure out where the algorithm believes the most likely place the dot is placed, then moving to that location.

[Explain why a CNN is being used, maybe over a ANN.]

We aimed to find out if changing the parameters to the CNN mask, would making the size of the mask provide speed benefits that would outweigh the decrease in the model accuracy. We will also be checking if the increase in steps for the sliding window increased performance but does not affect performance in completing its task of detecting the potential location of the red dot in the image.

The initial results were: Increasing the results were: CNN 3x3/6x6 SW 2/5/10

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 sliding window. These changes were aiming to check if the speed of the process became faster with each increasing parameter, while still being able to complete the game. To see what the best trade-off with regards to checking every part of the image and total completing time of the game.

2.2 Packages

We will be using the programming language Python 3 [4], as this allows us to use all the required packages needed to analyse the dataset. Numpy, opencv, pptk, matplotlib, skimages (dont think I need this), sklearn (train split), tensorflow version 2 [1].

With aiding in loading in the dataset, we will use the library Pandas library [3]. We will be using Matplotlib's [2] package library for visualising our data, to allow us to be able to get insights and spot possible trends.

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 10 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, and this time activation of 'softmax'. We then compile the model using 'adam' 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 sliding window will scan the extracted image from the image cloud, this is due to the training and CNN needing 51x51 images, but the provided image is of size 160x260. Therefore, by scanning the image in 51x51 sections, the patches can be fitted into the model. The sliding window 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 700 positive images and 1800 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 values for image, map1, map2, map3 and map4. The image was a snapshot of what the game world was showing to the dimensions of 160x260, 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 sliding window 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 extracted. This position location we used to gain the x and y center points. With having the center 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, are used to extract the position coordinates from the point cloud. We are therefore moving to that location. A function that updates the scene gets called, which then 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. Therefore, repeating the process, but getting the image from the points cloud from the current position. However, if we do not locate the dot, the position is reset to the previous location, and the angle is changed on its Y-axis by a radian value pf -0.75. This process of rotating will keep happening until the correct location of the red dot has 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

We can see from the results that the CNN that had a filter of 3x3 and trained quicker than the CNN with a filter of 6x6, in all three trials. All the trials found all of the red dots, but the quickest was the 3x3 2 step sliding window'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 window, these across the board were quicker than the other 6x6 results, but still not as fast as the 3x3 step. On average, we found the 6x6 filter took about 3 seconds longer to train the model.

4 Discussion

note: maybe insteead of finding the highest % patch, using the predicted label might have worked as well. Increasing dense layers node number-¿ will incrase training time but should make predictions better, therefore potentially speeding upu the overall game.

5 Conclusion

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

[3] McKinney, W. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (2010), S. van der Walt and J. Millman, Eds., pp. 51 – 56.

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