Clab-2 Report

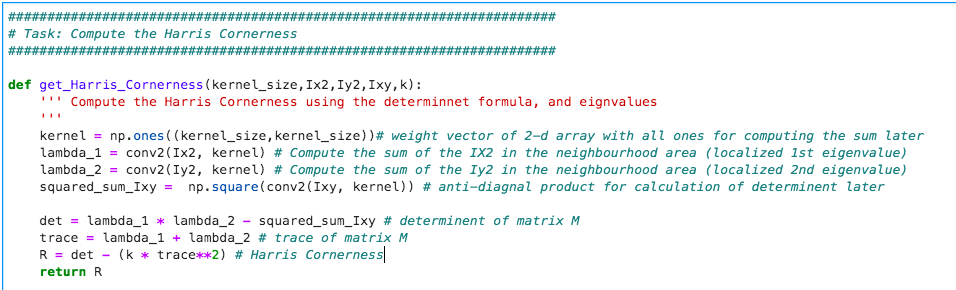
ENGN6528

Han Zhang

U6541559

29/04/2022

**Task 1 Harris Corner Detector. (5 marks)**



图形用户界面, 文本, 应用程序

描述已自动生成

Fig 1. Screenshot of the code of missing part.



Fig 2. Comment on block 5

The comment of the solution (missing part) is shown in the previous part of screenshot and is also available in the script.

Computational result of Harris corner detection of ‘Harris\_1.jpg’. Computational result of Harris corner detection of ‘Harris\_2.jpg’

图形用户界面

中度可信度描述已自动生成

Computational result of Harris corner detection of ‘Harris\_3.jpg’. Computational result of Harris corner detection of ‘Harris\_4.jpg’

建筑旁的房子

低可信度描述已自动生成

Built-in result of Harris corner detection of ‘Harris\_1.jpg’. Built-in result of Harris corner detection of ‘Harris\_2.jpg’

图形用户界面, 应用程序

描述已自动生成

Built-in result of Harris corner detection of ‘Harris\_3.jpg’. Built-in result of Harris corner detection of ‘Harris\_4.jpg’.

图形用户界面, 应用程序

描述已自动生成

Compared with built-in result, my result might identify a bit less corners. For example. In the enlarged pictures of the picture 2 shown below. There are a bit more nested corners counted in the logo of the clothes. While my result only counts one instance of the corner in the neighborhood area. If several corners nested in very closed local area, all of the others will be omitted. I believe it is related to the window (kernel) size when calculating the two eigenvalues of the M matrix. In my calculation, I set the kernel to 5, which corresponding a 5 \* 5 localize window to calculate the eigenvalue, trace, and also determinant. It must be a factor that affect the strongness of harris cornerness in a local area.

模糊的娃娃

中度可信度描述已自动生成娃娃放在一起

中度可信度描述已自动生成

Fig 3 enlarged pic 2 computational result (left), built-in result (right)

Also, the value of empirical constant k will affect the identification. K ranges from 0.01 to 0.1. The bigger k is, the smaller the R value. I chose k equals to 0.05. It can be further tuned to match the standard result.

**Task 2 - Deep Learning Classification (10 Marks)**

**normalize the data to the range between (-1, 1).文本

描述已自动生成**

**randomly flip the image left and right.**

**图形用户界面, 文本, 应用程序

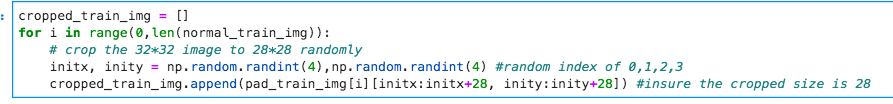
描述已自动生成**

**zero-pad 4 pixels on each side of the input image**

**文本, Word

描述已自动生成**

**randomly crop 28x28 as input**

****

**Build a CNN with the following architecture:**

图形用户界面, 文本, 应用程序

描述已自动生成

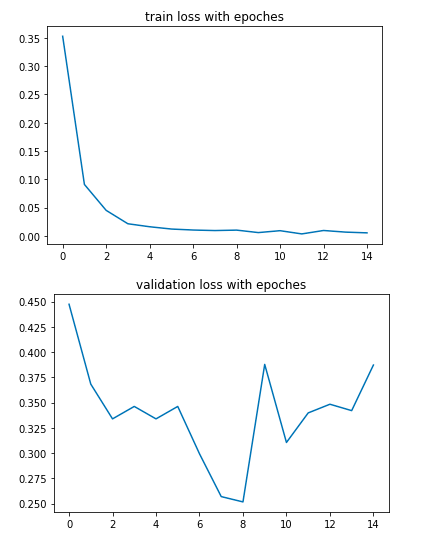
**Set up cross-entropy loss, Adam optimizer, with 1e-3 learning rate and betas=(0.9, 0.999)**

图片包含 图表

描述已自动生成

**Train your model. Draw the following plots:**

图表, 折线图

描述已自动生成

In practice, I used batch size of 128 in training, which can fasten the training speed and thus much less iterations are required. The model can reach relatively good performance in less than 15 epochs. I have also tried training without setting mini batch. The performance of loss and accuracy is similar but requiring much more iterations and time.

**Train a good model.**

To obtain a better result than the model using original setting, it is crucial to identify the main problem of the previous model. It is obvious to observe from the previous 4 plots, the empirical loss (training loss) is shrinking from epoch 0 to 8, and almost keeps unchanged from epoch 8. The leaning becomes hard. Similarly, the validation accuracy is increasing in the initial training period but also oscillated from epoch 8. It means the model is a bit over-training and the performance of the eventual model is even worse than the performance of the intermediate model (at epoch 6-8). The main problem of the model is overfitting.

To reduce overfitting, there are some measures.

* Early stopping

Stop the training as long as reach the satisfiable performance. In practice, I trained 10 epochs first and decided whether to continue from the previous model based on the performance. The performance curves were plotted with the combination of several periods.

* Regularization

Use L1 or L2 norm embedded in the layers of the model.

* Dropout

Add dropout layer between layers.

* Use different sets of parameters and tune the best model

Try combination of batch size, learning rate, optimizer, decay rate, etc. And find the best one.

* Batch normalization

Add batch normalization layer between layers.

Meanwhile, to enhance generalization, more convolution layers are added. The batch size was still set to 128 and thus the learning process was still very fast. Much less computation time and iteration steps are required. The best combination of setting is ‘bacth\_size =128’, ‘lr = 1e-3’,’regulairzer =L2’, ‘batch normalization = false’, ‘dropout=0.1’,‘optimizer = adam’, and ‘convolution layers added’ = True.

The training loss has already closed to 0 but the validation loss is still unstable. The training accuracy has already closed to 100% but the maximum validation accuracy is around 94%.

Therefore, the overfitting problem is reduced bit but still exists. The good thing is that the derivative of training loss seems to be not 0, motivating the model being learnable. While in the original setting, the derivative of training loss vanishes quickly.

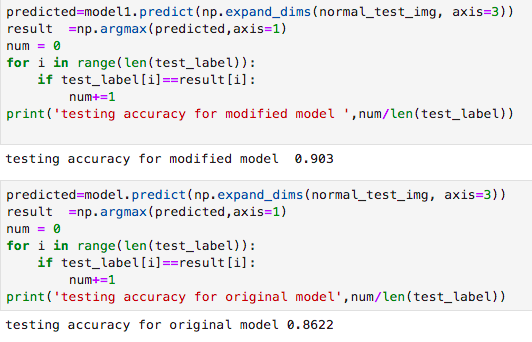
图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

When visually comparing the validation accuracy, it is hard to tell any improvement of the modified model. However, if all the performance is listed, it is easily to see the improvement on testing set. Although the training accuracy and validation accuracy become worse, the testing accuracy has a large improvement. That is, the original model overfits to the training sample and give false indication of the good generalization but the modified one is much more reasonable to indicate the true performance. The tuned model is much better in generalization and thus has a good testing score.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training accuracy | Testing accuracy | Validation accuracy |
| Original model | 0.9984 | 0.862 | 0.938 |
| Modified model | 0.9935 | 0.903 | 0.935 |



**Compare and discuss.**

Compared with the benchmark result, my model has a relative lower performance. The lowest benchmark is no less than 92% and with several models even high up to 98%. I can see that the models that have high performance up to 95% are actually the ensemble of several models. It is acceptable that the ensembled model have higher performance than vanilla CNN. Compared with some simple models such as Nearest neighbor, KNN, SVM, my result is worse.

Despite the model itself, I believe one important factor is the size of given training samples. The training sample size for my model is 59000 while they have 70,000 samples. The sufficient training sample can play a key role in reducing the overfitting thus motivating a good model.

Compared to the simple CNN model, which is most similar to mine. Its score is 94% which is slightly better than mine. The difference is that the images are not processed using data augmentation techniques. It is possible that involving random noise to a set of training samples that is not very large will lead to worse performance.

Compared to ResNet18 model, it is much more superior than mine with over 98% accuracy. The reason is that the Resnet topology has a much better generalization ability than basic CNN model.

In the Resnet, the input from previous layer is added directly to the output of the other layer, and the network is actually leaning the residual. The residual learning can deal with the problem of vanishing gradient when there are many layers in model. In the later stage of the training, the learning will not be stuck, it can still keep improving and finally generate a better model. While my model is hard to learn any more in the end, and thus has a worse performance than ResNet18 model.