Project 1

Deep Learning in Computer Vision

Authors

Daniel Juhász Vigild - s161749

Lau Johansson - s164512

Frederik Kromann Hansen - s161800

June 11, 2020

1 Hotdog / not hotdog

In the following, we will build and train a Convolutional Neural Network to predict whether a given image contains a hotdog or a "not hotdog". The dataset contains approx. 4,000 images with 52% used for training and a more or less balanced dataset.

Vanilla

To be able to classify hotdog / not hotdogs a Convolutional Neural Network has been implemented. The vanilla architecture consists of two convolutional layers and a fully connected linear layer. All convolutional layers uses a kernel size of 3 with 1 stride. To preserve the images' size throughout the network the padding size is 1. The first layer has 3 input features as the images are RGB represented and outputs 6 features. The second layer takes 6 input features and outputs 6 features. Before passing the data to the fully connected network, maxpooling is applied with kernel size 2. This results in a reduction of image size to 64x64 instead of 128x128. The fully connected network has a hidden layer with 512 neurons including dropout (0.5) for regularization. Both SGD and Adam have been tested as optimizers. The test result of the vanilla model was approximately 72.6 % for both optimizers. However, the SGD optimizer needed almost twice as many epochs as Adam, and therefore we chose to move on with Adam.

Batch normalization

To let the network converge faster and be more stable, batch normalization has been implemented after each convolutional layer. I.e. when data has been activated at each of the convolutional layers the output are normalized. The batch normalization improved the test performance to 75.9%.

Data augmentation

Two different operations of data augmentation have been performed. First all original images have been horizontally flipped with a probability of 50%, rotated by -30% to 30%, and center cropped, to enlarge the (not) hotdogs. Then the same operations have been performed, but all with a probability of 50%. Finally, all these three data sets have then been combined. Thus, the data set contains three times as much data as originally with up to three duplicate images. Using data augmentation did

not change the test performance (still 75.9%), but it reduced the gap between the training and test performance suggesting that we get rid of some of the overfitting, creating more room for improvement than when using only the original data set.

Increasing the number of convolutional layers

Having a training performance of 86.5 % indicated that even more complexity could be added to the network. The complexity of the model has been changed in a number of ways. For example: Introduction of another convolutional layer, one more hidden layer in the fully connected network, and eight more convolutional layers. None of the models increased the test performance but the more complex the model, the poorer the training accuracy. However, when increasing the number of features in the convolutional layers to 64 and 128 respectively test accuracy increased to 79.0%.

ResNet

Using 10 convolutional layers, the model can be subject to vanishing gradient problem. Therefore, we add residual blocks to the model as this will add the residuals of a preceding layer to the next, and thereby enlarging gradients very close to 0 and thus letting the signal run through. The ResNet model did not improve the performance of the previous model.

Network accuracy: Which are classified wrong?

Below is illustrated a selection of images and their corresponding classifications predicted by the model. Some of the misclassifications seem to make sense, for instance if the model has learned that there are often straight lines in images with hot dogs or that the colors brown and red are well represented in hotdog pictures.

Figure 1: Images with their corresponding predictions.

Conclusion

The Convolutional Neural Network performs well in classifying the images in two categories, hotdog and not hotdog. After testing multiple different architectures and hyperparameter settings, the best implementation in this project showed a test accuracy of 79.0% which is a significantly better performance than blind guessing, which would yield approximately 50% given the balanced dataset. This was achieved with a simple design with two convolutional layers with maxpooling and a single hidden fully connected linear layer with dropout. The performance of all tested extensions can be assessed in the appendix Figure 5.

2 O, where the party at?

To be able to detect and read house numbers we build a convolutional neural network. The dataset used is a combination of Street View House Numbers (SVHN) and Pytorch's CIFAR10 as a new class (negative samples), which represents 'no digit'. The test set consists of 26,000 images from SVHN and 10,000 from CIFAR10. The training set consist of 73,000 SVHN images and 50,000 CIFAR10. As a consequence of this balance the model will be better at predicting digits relative to background in a real life example, but this is not considered an issue in this project. During training, the size of the images are 32x32. The model that is build in this project aims at being able to output 11 different convolutional outputs depending on the image size. Thus, when passing full house numbers (of larger size than 32x32) the output represents a sliding window predicting the house numbers.

Architecture and training

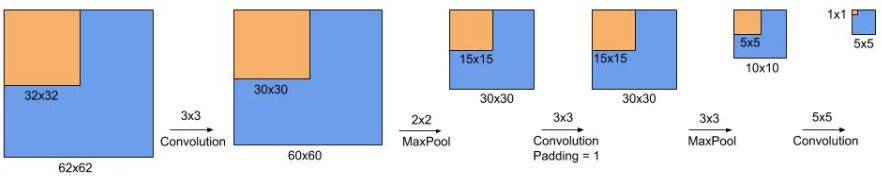

Figure 2: CNN architecture

In the following, the architecture of the model and processing of the training is

explained and illustrated in Figure 2. The first layer of the CNN is a convolutional layer with kernel size 3. The image is thereby reduced to 30x30. A subsequent maxpool (2x2) layer reduces the image to 15x15. Another convolutional layer with kernel size 3 padded with 1 is applied to let the network learn before the next maxpooling layers. Then a maxpool (3x3) layer reduces the image to 5x5. Having the suitable data size (5x5) it is passed on to a convolutional layer with kernel size 5 and finally softmaxed. Between all the layers ReLU acitivation has been applied to ensure non-linearity. The performance of the model, when classifying single digits is 84.9% in training and 80.6% in testing. Full house number images can be passed to the model no matter the size - though the image is padded if its dimensions are smaller than 32. In the following example, a image with a size of 62x62 is passed (note that it does not have to be quadratic). The shape of sliding window output is then a 6x6 matrix containing the predicted number for each of the windows. Below, the model tries to classify a house number 120.

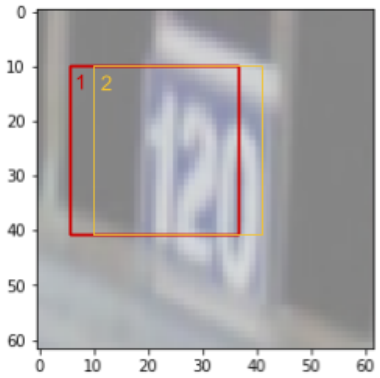

Figure 3: Image of house number 120

As in the first step of non-maximum suppression, the Softmax values of the predictions has been thresholded by a value of 0.9. It means, that the network has to be sure that the number detected at a given location in the picture should be with at least 90% probability. If it is lower than 90%, the area is labelled as background (no digit). The following matrix represents the classified digit/background for each of the windows.

```
tensor([[[10, 10, 10, 10, 10, 10],
         [10, 10, 10, 10, 10, 10],
         [10,  1,  2, 10, 10, 10],
         [10, 10, 10, 10, 10, 10],
         [10, 10, 10, 10, 10, 10],
         [10, 10, 10, 10, 10, 10]]], device='cuda:0')
```

Figure 4: Predicted classes for each sliding windows

As seen in Figure 4 (the matrix) the network has been able to detect the digits '1' and '2', with more than 90% probability, but not detect the '0'. In Figure 6 in the appendix the predicted digits without thresholds are shown and the corresponding probabilities in Figure 7. Looking at some of the highest probabilities, the network classifies e.g. a '7' in the top right corner of the house number sign with a probability of 84% and a '9' in the right bottom corner of the sign with a probability of 89%. The reason could be the that the structure of a '0' and a '9' are very similar.

Further work: Bounding box and IoU

Each of the pixels in the 6x6 representation (from Figure 4) represents a 32x32 bounding box in the input image. By backtracking the output pixels with the corresponding pixels in the input, we could find the bounding boxes. To get clean bounding boxes of the classified digits, cleaning of the boxes should be done by non-maximum suppression. First, using a probability threshold (as we already have done). Then, use Intersection over Union to clean out boxes and not be constrained by rectangular boxes.

Conclusion

The Convolutional Neural Network performs well in classifying single digits house numbers with a testing accuracy of 80.6%. Passing full house numbers to the model, it detects and classify house numbers well with the use of probability threshold. We did not manage to implement actual bounding boxes, but the model output do provide the necessary information to further enable us to implement boundary boxes.

Appendix

Diary for project 1.1

For this project, we collaborated on making a vanilla convolutional network. Thereafter, we diverged such that Daniel, worked on implementing data augmentation, while Lau and Frederik both experimented independently with model complexity and hyper parameter settings. Then we converged again, to discuss our findings regarding model performance and worked together on combining our best performing code snippets. Finally, we implemented ResNet while screen sharing.

Diary for project 1.2

First, we made a regular CNN based on the previous project and exercises. Then we discussed the related theory and how to best implement it. Pen and paper was used and shared on webcam to better understand the concepts. Occasionally, we diverged to research on how to code the model in practise, how to load the full size images, etc. Despite extensive research, unfortunately, we did not figure out how to include the bounding boxes in the model. However, the concept is clear. Due to the challenging tasks in this project part and when not having made any assignments about bounding boxes during the week - we found it necessary that one of us screen shared and the two others assisted verbally.

Performance results

Description	Performance train (%)	Performance test (%)
Two convlayers, dropout in FC (Adam)	78,9	72,6
Two convlayers, dropout in FC (SGD)	75,8	72,6
Two convlayers, batchnorm, dropout in FC (adam)	92,1	75,9
Two convlayers, batchnorm, dropout in FC + aug (adam)	86,5	75,9
Three convlayers, batchnorm, dropout in FC + aug (adam)	88,1	75,4
Three convlayers + dropout i last, batchnorm, dropout in FC + aug (adam)	70,6	70,1
Three convlayers + dropout i last, batchnorm, dropout in FC two layer + aug (adam)	76,6	70,8
Ten convlayers + dropout i last, batchnorm, dropout in FC two layer + aug (adam)	71,7	71,6
Ten convlayers + dropout i last, batchnorm, dropout in FC two layer 5000 neurons + aug (adam)	71,8	71,4
Resnet 3 lag	70,8	72,2
resnet 10 lag dropout og batch maxpool	68,8	62,1
resnet 10 lag NO dropout og batch maxpool	87,6	70,1
resnet 18 lag NO dropout og batch maxpool	81,4	64,9
Two convlayers, batchnorm, dropout in FC (adam) (increase features to 64 and 128)	98,0	78,1
Two convlayers, batchnorm, dropout in FC + aug (adam) (increase features to 64 and 128)	**89,3**	**79,0**

Figure 5: Performance results

Predicted digits without threshold

```
tensor([[[ 1,  4,  5, 10, 10, 10],
         [ 1,  6,  3,  3, 10, 10],
         [ 1,  1,  2,  5,  7,  7],
         [ 4,  1,  2,  4,  1,  7],
         [ 1,  4,  4,  9,  1,  0],
         [ 3, 10,  5,  9,  1,  4]]], device='cuda:0')
```

Figure 6: Predicted classes for each sliding windows without threshold

```
tensor([[[0.5352, 0.3409, 0.4754, 0.9779, 0.9763, 0.9809],
         [0.6439, 0.6159, 0.2883, 0.4474, 0.6321, 0.9661],
         [0.3916, 0.9700, 0.9101, 0.4612, 0.8424, 0.3943],
         [0.5441, 0.7463, 0.7162, 0.4853, 0.4617, 0.4201],
         [0.2878, 0.6249, 0.2884, 0.8856, 0.6683, 0.3470],
         [0.4560, 0.8633, 0.6109, 0.2580, 0.4680, 0.6677]]], device='cuda:0',
       grad_fn=<MaxBackward0>)
```

Figure 7: Probabilities for predicted classes for each sliding windows