AIML 425 T2/2022 - ASSIGNMENT 3 - PROBLEM 2

Corvin Idler - ID 300598312

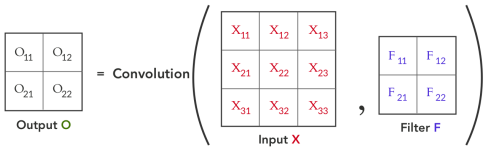
1. INTRODUCTION

This report describes the results of using a deep convolutional neural network (CNN) to perform object recognition on basic geometric shapes (triangle, circle, square) randomly located in binary 28*28 pixel images.

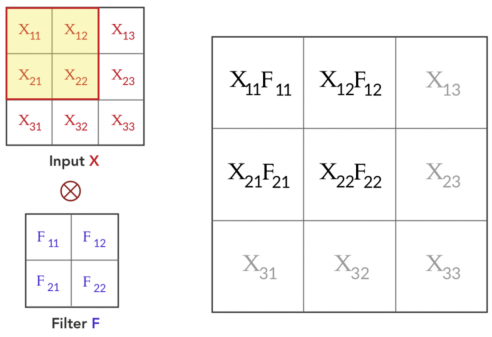
2. THEORY / CONCEPT

2.1. General concept

The main idea behind convolutional neural networks (CNNs) is to train a set of filter masks to detect features relevant to a particular task (e.g. classification of geometric shapes). As



(a) Convolution schematics



(b) Convolution example

Fig. 1: Schematics of the convolution working principle . Source: https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c

can be seen in Figure 1 any given pixel in the output of the convolution (here $O_{1,1}$) is dependent on as many pixels in the input image as the kernel size of the filter dictates (here 2x2). The pixels of the input image are multiplied with the corresponding weights of the filter mask and the result is summed up and stored as a pixel in the output image. The weights ($F_{i,j}$) of the filter mask is what needs to be learnt during the training process. In traditional computer vision these filter weights would have been defined ex-ante to e.g. perform edge detection or other specific tasks. In the machine learning paradigm one follows a data driven approach to find the best combination of filters and the best weights for each filter for a given task and data set. While the convolutions and convolutional layers might be the work horses or name givers of the CNNs, there are other components and layers usually present in a network architecture for object classification. The convolutional stages or layers are usualy followed by a so called pooling layer which performs a non-linear down-sampling (maxpooling) to reduce the spatial dimensions of the feature maps produced by the convolutional layer. This step helps to reduce the amount of free parameters in the network and therefore reduces training time and might aid with stability and prevention of over-fitting. Before the pooling operation is performed, one usually applies an activation function (like ReLu) to each pixel to allow for non-linearities. It is quite common to construct neural networks with many layers of convolution and pooling (deep networks) to distill more complex features than just edges or corners. That could be features relating to more complex geometric primitives or to texture etc. Towards the end of these pipelines of convolution, Relu and Pooling, when feature maps become sufficiently small in terms of spacial dimensions, they are vectorized or "flattened" into a one-dimensional vector and fed into one or more fully-connected network layers. The last step in a object classification pipeline is a softmax function to perform the one hot encoding for the object class.

3. RESULTS OF EXPERIMENTS / CONCLUSION

The jupyter-notebook underpinning this report can be found on GitHub: `https://colab.research.google.com/github/econdatatech/AIML425/blob/main/AIML425_Assignement_3.ipynb`.

3.1. Implementation details

I used the PIL python package to generate 28*28 single channel black and white picture of three basic geometric shapes.

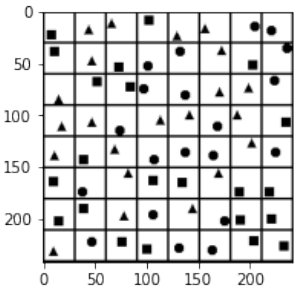


Fig. 2: Mosaic of a batch of 64 generated images

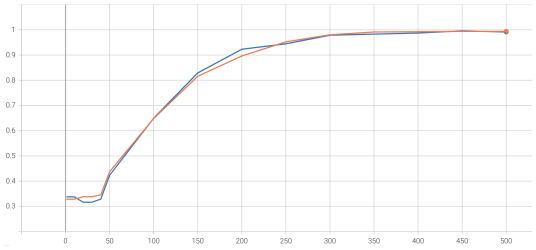
The shapes themselves were one circle, one triangle or one square, all placed in the upper left corner of the image canvas and all the shapes have maximum dimensions of 9*9 pixels, leaving 19 pixels in each direction for geometric translation to guarantee that over 1000 images can be created by placing the geometric shapes at different locations on the image canvas without creating duplicates. I used 70% of the images for the training set (819 images), 20% for the validation set (234 images) and 10% for the test set (117 images). Image 2 shows a mosaic of 64 of the generated images of the test set. The network architecture for performing the classification is as follows (c.f. figure 1): A convolutional layer with six channels (2x2 filters and stride 1), passed through a ReLu and followed by a 2x2 MaxPool layer. The results feed into a second convolutional layer with 6 channels (2x2 filters and stride 1), again passed through a ReLu and fed into a second MaxPool layer (2x2). A drop out layer follows to prevent overfitting. The results are then flattened and passed into a fully connected layer of 20 neurons + ReLu, followed by another drop out layer and then a layer of 10 neurons + ReLu. From there the results are passed into a fully connected layer with 3 neuron + SoftMax. As mini-batch size I chose 64 images and for the number of epochs I selected 500 as loss and accuracy were tapering off after that point. The number of free parameters are 3,107.

Layer type	Output Shape	# of Parameters
myConv2D-1	[64, 6, 27, 27]	24
MaxPool2d-2	[64, 6, 13, 13]	0
myConv2D-3	[64, 6, 9, 9]	900
MaxPool2d-4	[64, 6, 4, 4]	0
Dropout-5	[64, 6, 4, 4]	0
Linear-6	[64, 20]	1,940
Linear-7	[64, 10]	210
Dropout-8	[64, 10]	0
Linear-9	[64, 3]	33

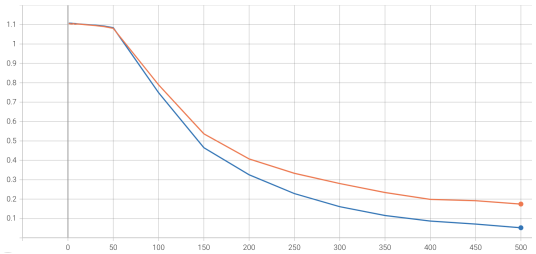
Table 1: Architecture of neural network used for experiments

3.2. Results and discussion

In the result visualisations in figure 3 we can see that the network achieves 100% accuracy on the training as well as validation set. We can also see that for quite a while the loss of the training and the validation set go ”hand in hand” but at some point the loss of the validation set is actually dropping below the loss of the training data set. This can be explained with the drop out layers that are present in the training phase but switched off for the evaluation on the validation set. So the network that performs predictions on the training data set is always a bit handicapped in comparison to the network that evaluates the validation set. The prediction accuracy on the test set was as well 100% as can be seen from confusion matrix in Figure 4. I decided to experiment with random rota-

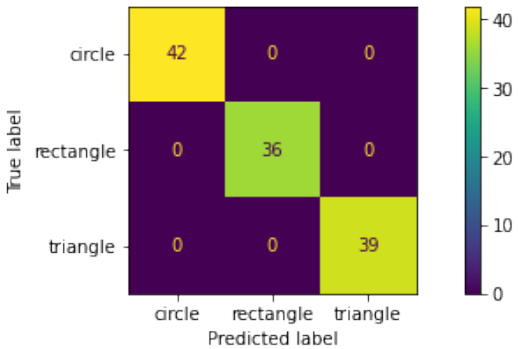


(a) Accuracy



(b) Loss

Fig. 3: Results with translation only (training + validation)

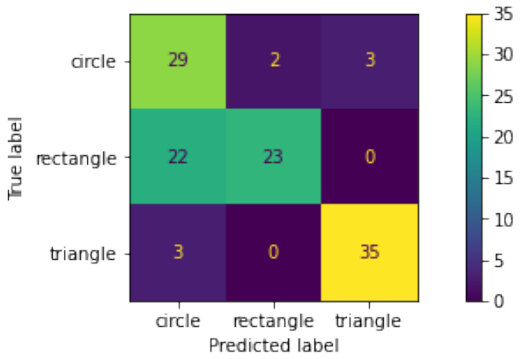


(a) Accuracy

Fig. 4: Results with translation only (test set)

tions of between -15 to + 15 degrees on the test set and the

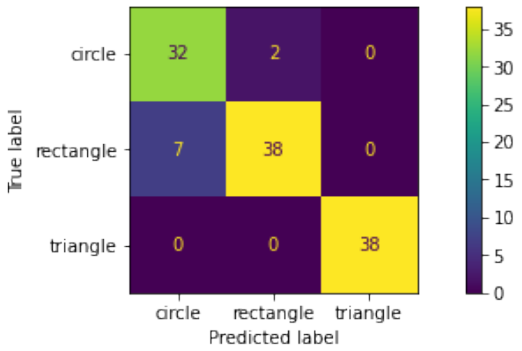
accuracy dropped to 0.8291. From the confusion matrix in Figure 5 we can see that most of the accuracy drop originates from many squares being mis-classified as circles. Once the



(a) Accuracy

Fig. 5: Results with translation and rotation (test set)

random rotation is introduced as well in the training data set, the network is quite good at dealing with it. The accuracy on the test set (incl. rotation) jumps back up to 0.9316 and as we can see in Figure 6 far less confusions between squares and circles occur. This shows that the network is powerful enough to deal with geometric transformations other than just pure translations as long as the there are sufficient training examples.



(a) Accuracy

Fig. 6: Results with translation and rotation (test set) after training with rotation