Lab report – TNM112

Deep learning for media technology, Lab 2

Oliver Lundin (olilu316)

Lab partner: Filip Bägén (filma379)

Wednesday 6th December, 2023 (12:28)

Abstract

The report documents the second lab of TNM112, focusing on convolutional neural networks (CNNs) for image processing tasks. Implementation of CNN components and regularization strategies to enhance generalization on datasets like MNIST and PatchCamelyon. The lab demonstrates the effectiveness of CNNs in computer vision applications and the importance of proper model regularization.

1 Introduction

This report describes the work done in the second lab of the course Deep Learning for Media Technology (TNM112)1. The lab is performed to learn about the convolutional neural network (CNN), a powerful deep learning model for processing structured data with spatial correlations, such as images2. The lab is important because CNNs are widely used in various applications of computer vision, such as image classification, object detection, face recognition, and more. The specific purpose of this lab is to:

Implement the building blocks of a CNN, such as convolutional, pooling, flattening, and dense layers, and understand how they work together to extract and classify image features34. Apply different regularization strategies, such as augmentation, dropout, weight decay, and batch normalization, to improve the generalization performance of a CNN on a small dataset of handwritten digits (MNIST). Explore different ways to design and optimize a CNN for a more challenging task of tumor classification on a medical imaging dataset (PatchCamelyon).

2 Background

CNN's consist of layers that extract features from the input data and perform classification or regression tasks. CNNs have been widely used for various applications in computer vision, such as image recognition, segmentation, and generation. Regularization is a technique that aims to reduce the risk of overfitting, which occurs when a model learns too much from the training data and fails to generalize well to new data.

3 Method

3.1 Task 1

To implement our own CNN we need to define the convulotion layer in python. The shape of our output from this layer should be the same as the width and height from the activations of the previous layer. The input and output channels should as be the corresponding width and height of the previous channels. To perform convulotion with 2d arrays we can use the convolve2d function from the scipy module.

```python
def conv2d_layer(h, W, b, act):
    CI = h.shape[2]
    CO = W.shape[3]

    output = np.zeros((h.shape[0], h.shape[1], CO))

    for i in range(CO):
        conv_sum = np.zeros_like(output[:, :, i])

        for j in range(CI):
            kernel = W[:, :, j, i]
            flipped_kernel = np.flipud(np.fliplr(kernel))
            conv_sum += signal.convolve2d(
                h[:, :, j], flipped_kernel, mode='same')

        output[:, :, i] = activation(conv_sum + b[i], act)

    return output
```

The function above performs a 2D convolution operation between the input activations (h) and a set of convolutional kernels (W). It does this by taking each kernel for each output channel, convolving it with the corresponding input channel, summing these convolutions, adding the bias, and applying an activation function to generate the output activations for each channel. This process is repeated for each output channel, generating the final output tensor with convolved and activated features across all specified channels.

After the convolution, 2D max pooling is performed

```python
def pool2d_layer(h):
    sy, sx = h.shape[0] // 2, h.shape[1] // 2

    pool_size = (2, 2)

    ho = np.zeros((sy, sx, h.shape[2]))

    for channel in range(h.shape[2]):
        ho[:, :, channel] = measure.block_reduce(
            h[:, :, channel], block_size=pool_size, func=np.max)

    return ho
```

This function divides the input tensor into non-overlapping rectangular regions. The size of these regions is determined by a specified pooling window or kernel size (2,2). For each region in each channel, the maximum value within that region is chosen as the representative value for that region. The output tensor has reduced spatial dimensions (height and width) depending on the pooling window size.

```python
def flatten_layer(h):
    return h.flatten()

def dense_layer(h, W, b, act):

    h = h[:, np.newaxis]
    z = np.matmul(W, h)+b
    a = np.maximum(0.0, z)
    a = activation(z, act)
    return a[:, 0]
```

The first of these functions flattens the input tensor to a vector output. The dense layer function implements a fully connected layer.
We then compared our own model with the keras model using a provided code block.

3.2 Task 2

For this task we were given a CNN model that was slightly overfitting and not performing good enough on the validation accuracy. We implemented some optimization strategies such as: dropout, batch normalization and data augmentation [1]. Both dropout and batch normalization can be easily implemented using the keras.layers module. For the data augmentation we used the keras.preprocessing module.

```python
datagen = keras.preprocessing.image.ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1
)
```

3.3 Task 3

The last task was to implement our own network to classify images using the PatchCamelyon dataset. We used the model from the previous task but with slightly modified parameters [1], such as the augmentation in which we added a random contrast operation to the images.

```python
data_augmentation = keras.Sequential([
    keras.layers.RandomFlip('horizontal'),
    keras.layers.RandomRotation(0.5),
    preprocessing.RandomContrast(factor=0.5),
])
```

4 Results

4.1 Task 1

Below are the result of our own implemented CNN.

When comparing our own model to the keras the absolute sum of both models should be the same, which seems to be the case to an extent of a couple of decimals. The absolute difference should also be close to zero which we also achieved.

Train Loss	Train Acc	Test Loss	Test Acc
0.2239	0.93	0.2147	0.93

Table 1: Our own CNN accuracy and loss.

Abs sum keras	Abs sum our	Abs difference
683.078	683.0779463015497	5.125999450683594e-05

Table 2: The comparison between our model and keras.

4.2 Task 2

For task 2 an accuracy of 90 was achieved:

Train Loss	Train Acc	Test Loss	Test Acc
0.7944	100.00	1.0524	91.00

Table 3: Accuracy for Task 2.

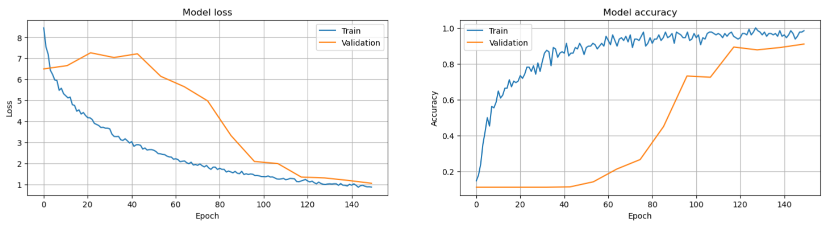

Figure 1: Results from Task 2.

4.3 Task 3

For task 3 we achieved a validation accuracy of 78.29, this was around the highest accuracy we could achieve with our model.

Train Loss	Train Acc	Validation Loss	Validation Acc
0.4499	80.80	0.4793	78.29

Table 4: Accuracy for Task 3.

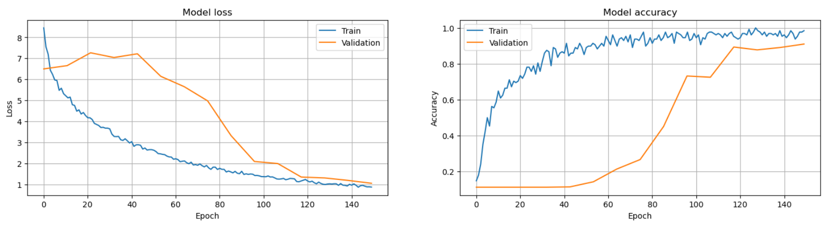

Figure 2: Results from Task 3.

5 Conclusion

The implementation of a custom CNN demonstrated the effectiveness of convolutional, pooling, flattening, and dense layers in feature extraction and classification1. The comparison with a Keras model validated the accuracy of the custom model to a high degree of similarity. The custom CNN achieved an 93% accuracy on both training and test datasets. The absolute sum and difference between the custom model and the Keras model were nearly identical, indicating a successful replication of the CNN's functionality.

For task 2 the model was initially overfitting and still might be doing so altough it is converging in the right direction. We found that the dropout and where it was implemented was an important factor for how much the model was fitting onto the training data. We also found that the other optimization strategies helped improve accuracy such as batch normalization and data augmentation. Especially for task 3 where we could not only change the transformations of the images but also the contrast.

Use of generative AI

I have used generative ai extensively throughout this lab report.

References

[1] TNM112. Lab 2 – convolutional neural networks, 2023. Lab description – Deep learning for media technology.