Lab report – TNM112

Deep learning for media technology, Lab 2

Emma Zettervall (emmze999)

Wednesday 11th December, 2024 (13:30)

Abstract

This lab focused on exploring Convolutional Neural Networks (CNNs) with a specific emphasis on model construction, regularization, and data augmentation. Key tasks included implementing CNN layers manually and testing various regularization strategies to improve model generalization and prevent overfitting. Additionally, data augmentation techniques were applied to enhance model robustness, with results showing that the choice of augmentation is highly dependent on the dataset. For example, the MNIST dataset required more constrained augmentations due to the sensitivity of digit orientation, while the PatchCamelyon dataset allowed for broader transformations. The lab reinforced the importance of understanding how different strategies affect model performance and emphasized the necessity of adapting these techniques based on the dataset and task at hand.

1 Introduction

This lab focuses on two main objectives. First, to understand the construction and implementation of Convolutional Neural Networks (CNNs), which are widely used in image processing and classification tasks [2]. Second, the lab emphasizes the importance of model generalization, which is the ability of a trained model to perform well on unseen data. Generalization is achieved through techniques like regularization and data augmentation, which help prevent overfitting and ensure robustness.

2 Background

The lab involved implementing a CNN with its core layers: convolutional, pooling, flatten, and dense layers. To improve generalization, regularization techniques such as dropout and weight decay were applied. Additionally, data augmentation techniques like cropping, rotation, and zooming were used to artificially expand the training dataset and enhance robustness. To evaluate the performance, the model was tested on two different datasets, the simple MNIST dataset and the more complex PatchCamelyon [1] dataset.

3 Method

The lab was done in a Jupyter notebook. Keras and python was used to build a CNN for classifying images.

3.1 Task 1

The task involved implementing functions in the script cnn.py for different layers of a CNN: activation, convolutional, pooling, flatten, and dense layers. Additionally, the activation function, the accuracy in the evaluation function and the total number of weights in the model was implemented.

3.1.1 Activation

The activation function was reused from Lab 1. It supports multiple activation types, including linear, ReLU, sigmoid, and softmax.

```
def activation(x, activation):
    if activation == 'linear':
        return x
    elif activation == 'relu':
        return np.maximum(0, x)
    elif activation == 'sigmoid':
        return 1 / (1 + np.exp(-x))
    elif activation == 'softmax':
        exps = np.exp(x - np.max(x))  # Stability trick
        return exps / np.sum(exps)
    else:
        raise Exception("Activation function is not valid",
    activation)
```

3.1.2 Convolutional layer

The convolutional layer applied 2D convolutions over the input feature maps. A nested loop iterated over input and output channels, where kernels were flipped before convolution, and the results were summed.

```
def conv2d_layer(h, W, b, act
):
    CI = h.shape[-1]
    CO = W.shape[-1]
    output = np.zeros((h.shape[0], h.shape[1], CO))
    for i in range(CO):
        for j in range(CI):
            kernel = W[:,:,j,i]
            kernel = np.flipud(np.fliplr(kernel))
            conv = signal.convolve2d(h[:,:,j], kernel, mode='same')
            if j == 0:
                conv_sum = conv
            else:
                conv_sum += conv
        output[:, :, i] = activation(conv_sum + b[i], act)
    return output
```

3.1.3 Pooling layer

A max pooling layer was implemented to reduce spatial dimensions by pooling over 2×2 windows using the `block_reduce` function.

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

    for i in range(h.shape[2]):
        ho[:,:,i] = skimage.measure.block_reduce(h[:,:,i], (sy, sx)
    , np.max)

    return ho
```

3.1.4 Flattening layer

The implemented flatten layer converted multi-dimensional input arrays into a 1D vector.

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

3.1.5 Dense layer

The dense layer performed matrix multiplication followed by adding a bias term and applying the activation function. Slight modifications were made to the code from Lab 1.

```
def dense_layer(h,   # Activations from previous layer
                W,   # Weight matrix
                b,   # Bias vector
                act  # Activation function
):
    h = h[:, np.newaxis]
    z = np.matmul(W, h) + b # Shape [M, 1]
    activated_output = activation(z, act)
    return activated_output[:,0]
```

3.1.6 Evaluation

The evaluation function was reused from Lab 1 but adapted to compute the loss with cross-entropy.

```
def evaluate(self):
    print('Model performance:')

    y_train_pred = self.feedforward(self.dataset.x_train)
    train_loss = np.mean((y_train_pred - self.dataset.y_train_oh)
    **2)
    train_acc = np.mean(np.argmax(y_train_pred, 1) == self.dataset.
    y_train)

    print("\tTrain loss:     %0.4f"%train_loss)
    print("\tTrain accuracy: %0.4f"%train_acc)

    y_test_pred = self.feedforward(self.dataset.x_test)
    test_loss = np.mean((y_test_pred - self.dataset.y_test_oh)**2)
```

```
    test_acc = np.mean(np.argmax(y_test_pred, 1) == self.dataset.
    y_test)

    print("\tTest loss:      %0.4f"%test_loss)
    print("\tTest accuracy:  %0.4f"%test_acc)
```

3.1.7 Setup model

The total number of weights in the model, including biases, was computed as follows:

```
        self.N = sum(w.size for w in self.W) + sum(b.size for b in
    self.b)
```

3.2 Verification of Layer Implementation

To verify the correctness of each layer, outputs were tested on a single input image and compared against the corresponding outputs from a Keras model. Each function was debugged until the following minimal losses were achieved:

- **Convolutional Layer**: 2.09375×10^{-5}
- **Pooling Layer**: 0
- **Flatten Layer**: 0
- **Dense Layer**: 1.14553×10^{-7}

An example of testing a specific layer is shown below:

```
test_layer = 'dense'
```

3.3 Task 2

To understand generalization better and how it can help with overfitting, the dataset MNIST was used and the network was extended with reguralization strategies.
The expanded code included:

Batch size and epochs were adjusted as bellow.

```
epochs = 250
batch_size = 16
```

Kernel regularization with L2 weight decay was added to both the Conv2D layers and the final Dense layer.

```
kernel_regularizer=keras.regularizers.l2(0.0001)
```

Dropout layers were implemented after each Conv2D layer and the Dense layer to reduce overfitting. A dropout rate of 0.2 was used after the convolutional layers as shown bellow, and 0.5 after the dense layer.

```
conv1 = keras.layers.Dropout(0.2)(conv1)
```

Augmentation layers were applied after the input layer and included random cropping, rotation, and zooming:

```
[language=Python]
augmented = layers.RandomCrop(height=data.x_train.shape[1], width=
    data.x_train.shape[2])(x)
augmented = layers.RandomRotation(factor=0.1)(augmented)
augmented = layers.RandomZoom(0.1)(augmented)
```

3.4 Task 3

The PatchCamelyon dataset was used to further improve generalization. The code from Task 2 was reused with modifications, including the addition of augmentation layers and adjustments to parameters: the batch size was set to 128, the number of epochs was reduced to 10, and the weight decay was decreased to 0.00001. The final augmentation layers and their values were:

```
augmented = layers.RandomCrop(height=data.x_train.shape[1], width=
    data.x_train.shape[2])(x)
augmented = layers.RandomRotation(factor=0.5)(augmented)
augmented = layers.RandomZoom(0.5)(augmented)
augmented = layers.RandomFlip(mode='horizontal')(augmented)
augmented = layers.RandomFlip(mode='vertical')(augmented)
```

Since the dataset contains tissue samples from breast lymph nodes that could potentially be cancerous, the AUC_ROC score was calculated as part of the results. When a better result was achieved, the code for saving predictions to a CSV file, shown below, was commented out.

```
util.pred_test(model, data, name='your_submission.csv')
```

4 Results

The results from the tasks in lab 2 will be displayed below.

4.1 Task 1

The results for the implemented CNN and the Keras model were the same, as the weights and biases were obtained from the pre-trained Keras model.

Model	Loss	Accuracy
Keras	0.0627	97.94
Own implementation	0.0627	98

Table 1: The loss and accuracy from Task 1.

4.2 Task 2

Dropout, combined with minor augmentations, weight decay, a smaller batch size and a higher number of epochs improved the performance. Specifically, a batch size of 16 and 250 epochs yielded better results. The average accuracy, calculated over 5 different runs are presented in the table below.

Average loss	Average accuracy
0.5494	0.9018

Table 2: The loss and accuracy over five runs for Task 2.

4.3 Task 3

For the first dataset, MNIST, which contains handwritten digits, the orientation of the image is crusial for classification. Therefore, large rotations or flips cannot be used. However, for the PatchCamelyon dataset, which contains tissue images, rotations and flips can be applied because the direction of the image does not matter. The results from this task were uploaded to Kaggle [3] five times, with the highest achieved score being 0.74473. The corresponding run in the notebook produced the result shown below:

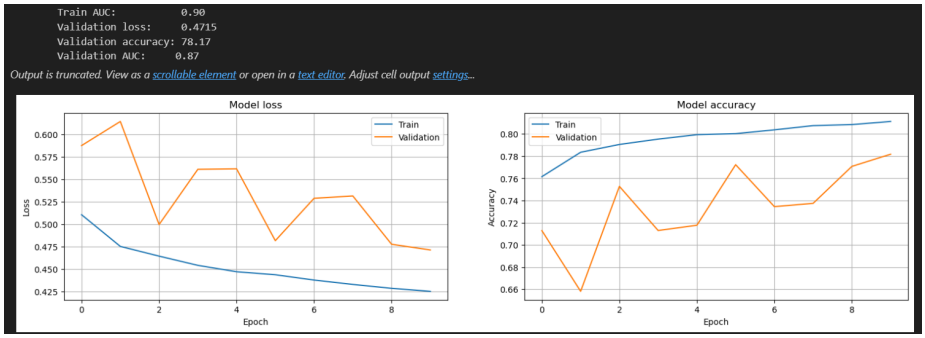

Figure 1: Results from task 3.

5 Conclusion

This lab provided valuable insights of Convolutional Neural Networks (CNNs), and the role of regularization. Based on the findings, the following conclusions can be drawn:

- The manual implementation of convolutional, pooling, flatten, and dense layers provided a deeper understanding of how a Convolutional Neural Network (CNN) processes input data.
- Regularization strategies such as dropout and weight decay were critical in improving model generalization. Dropout helped prevent overfitting by introducing noise during training, while L2 weight decay penalized large weights, ensuring smoother model performance.
- The use of augmentation techniques like cropping, flipping, and rotation improved the robustness of the CNN.
- This lab reinforced the importance of combining theory with practical implementation to understand deep learning concepts such as model construction, overfitting prevention, and generalization. It also highlighted

the value of testing models under different augmentation and regularization settings.

- This lab underscored that there is no universal approach to augmentation or regularization; these strategies must always be tailored to the specific dataset and task.

Use of generative AI

The abstract, background and conclusion was written with the help of ai, aswell as some rephrasing in method and result.

References

[1] J. Winkens T. Cohen M. Welling B. S. Veeling, J. Linmans. Rotation equivariant cnns for digital pathology, September 2018.

[2] TNM112. Lab 1 – the multilayer perceptron, 2023. Lab description – Deep learning for media technology.

[3] YifanD@LiU. Tnm112-lab2 (2024). `https://kaggle.com/competitions/tnm-112-lab-2-2024`, 2024. Kaggle.