

**National Tsing Hua University**  
**Fall 2023 11210IPT 553000**  
**Deep Learning in Biomedical Optical Imaging**  
**Homework 3**

**WISARUD YONGBANJERD<sup>1</sup>**

*Student ID:109003888*

**1. Task A: Reduce Overfitting**

Since, there is an overfitting problem occurred from the lab 4 exercise, I am thinking of using drop out and L2 regularization to reduce overfitting problem as well as increasing the test accuracy. I will first use dropout and later use L2 regularization.

*1.1 Introducing constructor parameters “drop\_prob” with a default value of 0.0.*  
Which shown on Fig. 1.

```
def __init__(self, dropout_prob=0.0):
```

Fig. 1. Creating constructor parameters drop\_prob.

*1.2 Padding Change to ensure that we are getting the same spatial dimensions for the output.*

Which shown on Fig. 2.

```
self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)  
self.conv2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)  
self.conv3 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
```

Fig. 2. Changing value of padding.

*1.3 Adding drop out layer as well as incorporate the dropout layer between fc1 and fc2 in “Forward” pass*

Which shown on Fig. 3.

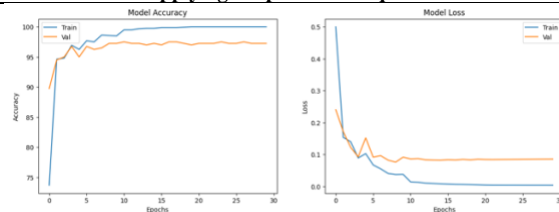
```
self.dropout = nn.Dropout(dropout_prob)  
  
x = F.relu(self.fc1(x))  
x = self.dropout(x)  
return self.fc2(x)
```

Fig. 3. Adding drop out layer and incorporate the dropout layer between fc1 and fc2 in “Forward” pass.

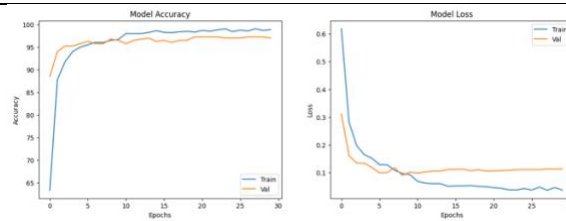
When initialize the model, I specify dropout\_prob = 0.5 and here is the result as shown on Table 1.

**Table 1. Showing result before and after applying drop out technique.**

Before Applying Drop Out



After Applying Drop Out



The training accuracy doesn't reach as high as it did without dropout, which means it's not fitting the training data as tightly. This is good and expected behavior when dropout is used. While the gap between the training and validation metrics is narrower in the dropout model, suggesting improved generalization. The test accuracy has improved by 5% from 72% to 77%, indicating better generalization to unseen data. While the model still exhibits a high training accuracy, the gap between training and validation metrics has reduced, indicating less overfitting. The smoother curves in the validation loss and reduced gap between training and validation loss also suggest that the model generalizes better and the increase in test accuracy supports the notion that dropout helped in reducing overfitting.

The overfitting issue has been mitigated to some extent with the use of dropout. The dropout has clearly helped in this case, making the model generalize better to the test set.

*1.4 When applying L2 Regularization by using ConvModel, I do not have to specify the dropout\_prob I only have to regularization in the optimizer.*

Which shown on Fig. 4.

```

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=0.01)

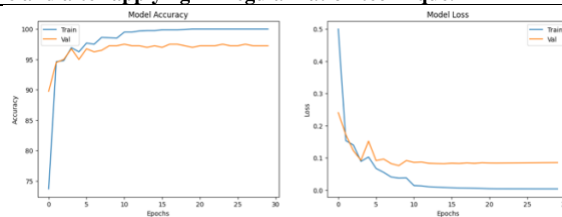
```

Fig. 4. Adding weight decay for L2 regularization

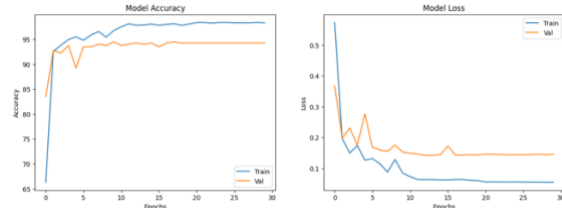
After running the model here is the result as shown on Table 2

**Table 2. Showing result before and after applying L2 regularization technique.**

Before Applying  
L2 Regularization



After Applying  
L2 Regularization



There's indeed a larger gap between training and validation in the L2 regularized model for both accuracy and loss. This suggests that the L2 regularization might have introduced some underfitting or didn't address the overfitting adequately. The validation loss in the L2 regularized model is more unstable with those distinct peaks. The increase in test accuracy from 72% to 78% is promising, but the plots do indicate that the model might still have some generalization issues.

In conclusion, while L2 regularization improved the test set performance, it might not have solved the overfitting problem entirely, and in some aspects, the generalization could have been compromised. It might be worth exploring other techniques or combinations, such as dropout, different architectures, or data augmentation, to further improve the model's robustness.

## 2. Task B: Performance Comparison between CNN and ANN

### 2.1 Feature Extraction Capabilities.

#### 2.1.1 Artificial Neural Networks (ANN):

ANNs consist of fully connected layers. This means every neuron in one layer is connected to every neuron in the next layer. The ANNs don't have any inherent feature extraction mechanism, meaning they don't have specialized layers that recognize patterns or features in input data. Due to this, they might require a larger number of parameters compared to CNNs for the same task, especially for image data.

#### 2.1.2 Convolutional Neural Networks (CNN):

CNNs are specifically designed for processing data that have a grid-like topology, e.g., image data. They use convolutional layers that can automatically and adaptively learn spatial hierarchies of features from input images. This allows them to capture local patterns like edges, corners, and textures. Pooling layers in CNNs help in reducing the spatial dimensions, thus summarizing the features, and making the network less sensitive to the location of the feature in the input.

Overall, CNNs can reduce the number of parameters needed, leading to a more compact and efficient model for image data.

### 2.2 Training Speed.

#### 2.2.1 Artificial Neural Networks (ANN):

ANNs might be faster to train on smaller datasets or simpler tasks due to fewer parameters and operations. Moreover, they might struggle or overfit on large image datasets because they don't have the capability to understand the spatial hierarchies of images.

#### 2.2.2 Convolutional Neural Networks (CNN):

The training of CNN might be slower initially, especially if there are many convolutional layers and parameters. However, CNNs are typically more efficient on image datasets, converging faster to a better solution because they can understand the spatial structure of the data.

From Table 3 showing the time for each of the model take for training.

**Table 3. Showing the time for each of the models take for training.**

Model	Time Taken
ANN	0.37 seconds
CNN	2.38 seconds

### 2.3 Model Performance.

#### 2.3.1 Artificial Neural Networks (ANN):

Likely to underperform on image datasets compared to CNNs, especially with large datasets or complex images. May work adequately on simpler tasks or datasets where the spatial hierarchy is not vital.

### 2.3.2 Convolutional Neural Networks (CNN):

Generally, outperform ANNs in image classification tasks. Able to handle more intricate patterns and variations in image data. Less likely to overfit than ANNs because of the shared weights in convolutional layers.

From Table 4 showing each of the models' performance on test data.

**Table 4. Showing each of the models' performance on test data.**

Model	Test Accuracy
ANN	73%
CNN	77.25%

## 2.4 Architecture Description.

### 2.4.1 Artificial Neural Networks (ANN) ('LinearModel')

Here is the breakdown of the ANN architecture description:

Input Layer: The images are flattened making the input size 256 x 256.  
Hidden Layer 1: Fully connected layer with 32 neurons and ReLU activation.  
Hidden Layer 2: Fully connected layer with 32 neurons and ReLU activation.  
Hidden Layer 3: Fully connected layer with 32 neurons and ReLU activation.  
Output Layer: Fully connected layer with 1 neuron (binary classification).

Which can be seen in Fig. 5.

LinearModel Architecture:

Layer (type)	Output Shape	Param #
Flatten-1	[-1, 65536]	0
Linear-2	[-1, 32]	2,097,184
Linear-3	[-1, 32]	1,056
Linear-4	[-1, 32]	1,056
Linear-5	[-1, 1]	33

Fig. 5. Showing Linear Model Architecture (ANN)

### 2.4.2 Convolutional Neural Networks (CNN):

Here is the breakdown of the CNN architecture description:

Convolutional Layer 1: 1 input channel, 32 output channels, 3x3 kernel, stride of 1.  
Followed by ReLU activation and 2x2 MaxPooling.  
Convolutional Layer 2: 32 input channels, 32 output channels, 3x3 kernel, stride of 1.  
Followed by ReLU activation and 2x2 MaxPooling.  
Convolutional Layer 3: 32 input channels, 32 output channels, 3x3 kernel, stride of 1.  
Followed by ReLU activation and 2x2 MaxPooling.  
Flattening: The output from the last pooling layer is flattened to feed into the fully connected layers.  
Fully Connected Layer 1: 32x32x32 input neurons, 32 output neurons, and ReLU activation.  
Output Layer: Fully connected layer with 1 neuron (binary classification).

Which can be seen in Fig. 6.

ConvModel Architecture:		
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 256, 256]	320
MaxPool2d-2	[-1, 32, 128, 128]	0
Conv2d-3	[-1, 32, 128, 128]	9,248
MaxPool2d-4	[-1, 32, 64, 64]	0
Conv2d-5	[-1, 32, 64, 64]	9,248
MaxPool2d-6	[-1, 32, 32, 32]	0
Linear-7	[-1, 32]	1,048,608
Linear-8	[-1, 1]	33

Fig. 6. Showing ConvModel Architecture (CNN)

### 3. Global Average Pooling in CNN

#### 3.1 Explanation of Global Average Pooling (GAP):

Global Average Pooling (GAP) is a technique used in convolutional neural networks (CNN) which replaces the flattening operation traditionally seen prior to the fully connected layers. Here's an elucidation:

##### 3.1.1 Role of GAP:

Instead of flattening the entire feature map and then feeding it into the fully connected layers (Dense layers), GAP computes the average value of each feature map. This means that if you have  $C$  feature maps, after applying GAP, you will get a tensor of shape  $(C, )$ .

##### 3.1.2 Eliminating Manual Dimension Calculations:

With GAP, you no longer need to manually calculate the number of features when transitioning from a convolutional layer to a fully connected layer. This is because GAP condenses the spatial dimensions (height and width) of the feature maps into a single value per feature map. Therefore, regardless of the spatial dimensions of your feature maps, after GAP, the tensor will always have a shape of  $(C, )$ .

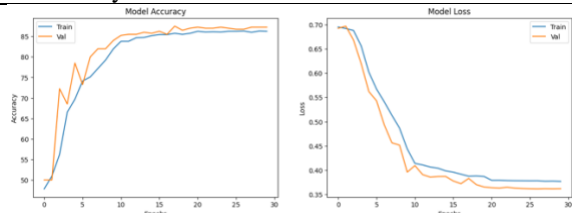
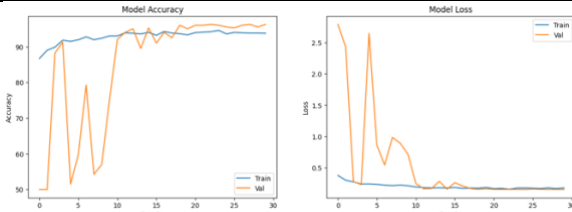
#### 3.2 Increase Performance:

Using GAP can result in a decrease in performance because you're discarding a lot of spatial information when averaging. However, there are strategies that I will use to elevate the performance:

1. I'll increase the number of filters in the convolution layers to extract more features.
2. I'll introduce dropout for regularization.
3. I'll introduce a batch normalization layer after each convolutional layer to accelerate learning.

Here are the results shown in table 5.

**Table 5. Comparing result before and after increasing performance in GAP**

Model Accuracy and Model Loss		
Before Increasing the Performance		
		
Training Time and Test Accuracy		
	Training Time	Test Accuracy
Before Increasing the Performance	2.33 seconds	74%
After Increasing the Performance	9.43 seconds	79.25%

From before increasing the performance, it's evident that in the model accuracy both training and validation accuracy rise as the number of epochs increase. There's a sudden spike in validation accuracy around epoch 4, but then it stabilizes. The training accuracy consistently improves, showcasing that the model is learning, while the validation accuracy plateaus around 80%. While the training loss is decreasing steadily, which is a good sign. The validation loss also decreases.

After modified several strategies, in model accuracy, the validation accuracy has fluctuations but generally stays above the "before" curve, hovering close to 90% in the later epochs. While in model loss, the validation loss has significant fluctuations, especially in the initial epochs, but settles down in the later epochs. Despite the fluctuations, the overall loss values are lower compared to the "before" results.

The modified ConvGAP() model shows improved performance in terms of test accuracy. The loss plot indicates that the model might be experiencing a bit of instability, especially in the initial epochs. This could be due to the batch normalization and the increased number of filters, which might make the optimization landscape more complex. Overall, the modifications have resulted in a better-performing model, even though the training time has increased.

## References

1. Pytorch Contributors. (2023). MODELS AND PRE-TRAINED WEIGHTS. <https://pytorch.org/vision/main/models.html#table-of-all-available-quantized-classification-weights>
2. Ng, A. (n.d.). Deep Learning Specialization. Coursera. Retrieved [October 26, 2023], from <https://www.coursera.org/specializations/deep-learning>