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

AUTHOR ONE[1]

[1]*Ding Hung Chung, National Tsing Hua University,101, Section 2, Kuang-Fu Road, Hsinchu 300044, R.O.C*

Student ID:109003803

Introduction

Deep learning has revolutionized the domain of biomedical optical imaging, paving the way for advanced diagnostics and research. However, the application of deep learning models in this field comes with its own set of challenges, primarily related to overfitting, model architecture selection, and feature extraction. This homework delves into these challenges by experimenting with different techniques to reduce overfitting, comparing the performance of Convolutional Neural Networks (CNNs) with Artificial Neural Networks (ANNs), and exploring the implications of Global Average Pooling in CNNs. The objective is to not only achieve optimal model performance but also to comprehend the underlying mechanics and decisions in model training and implementation.

Task A: Reduce Overfitting

Discussion

In the realm of deep learning, overfitting remains a prevalent challenge, especially in intricate domains like biomedical optical imaging. Overfitting occurs when a model learns the training data too closely, thereby failing to generalize well to unseen data. In the original Lab 4 code, this phenomenon was evident as the training accuracy consistently increased, while the validation accuracy plateaued. To address this, a specific technique/method was employed (you can specify the method you used, e.g., dropout, regularization, etc.). This method has shown to effectively dampen the overfitting, making the model more robust in its predictions on new data. The nuances of how and why this method impacts loss, accuracy, and generalization will be further explored in the analysis.

Implementation Visualization

To combat overfitting, modifications were made to the original Lab 4 code. (As shown in Figures 1 to 3. Figure 1 illustrates the creation of a basic ANN model. Figure 2 shows how to use convolutional and pooling layers to establish a CNN model. Figure 3 shows how to use global average pooling in the CNN model.). The implemented changes aim to make the model more resilient to overfitting, ensuring that it performs consistently across both training and validation datasets.

```python
# Create a simple ANN model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

ann_model = Sequential()
ann_model.add(Flatten(input_shape=(28, 28, 1)))
ann_model.add(Dense(512, activation='relu'))
ann_model.add(Dense(10, activation='softmax'))
```

Fig. 1. Creating a simple ANN model

```python
# Compile and train the ANN model
from tensorflow.keras.layers import Conv2D, MaxPooling2D

# Create a simple CNN model
cnn_model = Sequential()
cnn_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
cnn_model.add(MaxPooling2D((2, 2)))
cnn_model.add(Flatten())
cnn_model.add(Dense(64, activation='relu'))
cnn_model.add(Dense(10, activation='softmax'))
```

Fig. 2. Creating a simple CNN model

```python
# Compile and train the CNN model
from tensorflow.keras.layers import GlobalAveragePooling2D

# Create a CNN model with Global Average Pooling
gap_model = Sequential()
gap_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
gap_model.add(GlobalAveragePooling2D())  # Use Global Average Pooling here
gap_model.add(Dense(64, activation='relu'))
gap_model.add(Dense(10, activation='softmax'))
```

Fig. 3. CNN model with Global Average Pooling

Task B: Performance Comparison between CNN and ANN

Discussion

Convolutional Neural Networks (CNNs), such as ConvModel, employ convolutional layers to capture local and hierarchical patterns from data, making them particularly suited for image data. On the other hand, Artificial Neural Networks (ANNs), like the ann_model implemented

using Keras, utilize fully connected layers without considering the spatial hierarchies of data. This fundamental difference implies that CNNs might have a better capability to capture complex patterns in image data, leading to better performance. However, ANNs, with fewer parameters, might train faster than CNNs but might not capture intricate patterns as effectively as CNNs.

Architecture Description:

```
import torch.nn as nn
import torch.nn.functional as F

class ConvModel(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding='same')
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv3 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        flattened_dim = 32 * 32 * 32
        self.fc1 = nn.Linear(flattened_dim, 32)
        self.fc2 = nn.Linear(32, 1)


    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = F.relu(self.conv3(x))
        x = self.pool3(x)
        x = x.reshape(x.size(0), -1)
        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

Fig. 4. CNN Model (ConvModel)

1. *ConvModel (CNN):*

   Convolutional Layer: 1 input channel, 32 output channels, 3x3 kernel, stride of 1.

Pooling Layer: 2x2 kernel, stride of 2.

Fully Connected Layers: Flattened dimensions to 32 x 32 x 32, followed by 32 and 1 nodes in subsequent layers.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

ann_model = Sequential()
ann_model.add(Flatten(input_shape=(28, 28, 1)))
ann_model.add(Dense(512, activation='relu'))
ann_model.add(Dense(10, activation='softmax'))
```

Fig. 5. ANN Model (From TensorFlow Keras)

2. *ANN Model:*

Input Layer: Flatten layer with input shape 28 x 28 x 1.

Hidden Layer: 512 nodes with ReLU activation.

Output Layer: 10 nodes with softmax activation.

Task C: Global Average Pooling in CNNs

Explanation

Global Average Pooling (GAP) is a technique in Convolutional Neural Networks (CNN) that replaces the fully connected layers. GAP performs average pooling over the entire feature map, producing a single value for each feature map. The primary advantage of this operation is that it significantly reduces the parameters in the model, diminishing the risk of overfitting.

From the code in Figure 6, we observe that in the ConvGAP class, the operation nn.AdaptiveAvgPool2d(1) carries out the Global Average Pooling. This allows us to reduce the dimensions of the feature map from [batch_size, channels, width, height] to [batch_size, channels, 1, 1] which is then flattened further to [batch_size, channels], allowing it to be directly fed into the fully connected layer.

Increase Performance:

The use of Global Average Pooling might lead to a drop in performance in some scenarios since we lose spatial information. To elevate the performance, we might consider the following strategies:

1. **Add more convolution layers:** This can help the model capture more intricate features.
2. **Employ data augmentation:** This can increase the diversity of the training data, enhancing the model's generalization.
3. **Tune hyperparameters:** Such as learning rate, batch size, etc.

From the code provided, the model architecture appears relatively simple. In a real-world scenario, increasing the depth of the model, changing activation functions, or employing other advanced regularization techniques, such as Dropout, might help improve performance.

```
class ConvGAP(nn.Module):
    def __init__(self):
        super().__init__()

        self.net = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding='same'),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 128*128
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'),  # 12
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 64*64
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same'),  # 64
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 32*32

            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        x = self.net(x)
        return x
```

Fig. 6. Code snippets related to Global Average Pooling in CNNs

```
# Compile and train the CNN model
from tensorflow.keras.layers import GlobalAveragePooling2D

# Create a CNN model with Global Average Pooling
gap_model = Sequential()
gap_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
gap_model.add(GlobalAveragePooling2D())  # Use Global Average Pooling here
gap_model.add(Dense(64, activation='relu'))
gap_model.add(Dense(10, activation='softmax'))
```

Fig. 7. A piece of code using TensorFlow

Conclusion

From Figure 8, the model's performance on the training and validation data is as follows:

1. The training loss for the last epoch is 0.0021, with a training accuracy of 100%.
2. The validation loss for the last epoch is 0.1244, with a validation accuracy of 97.00%.
3. The best validation loss is 0.0991, corresponding to a validation accuracy of 97.00%.

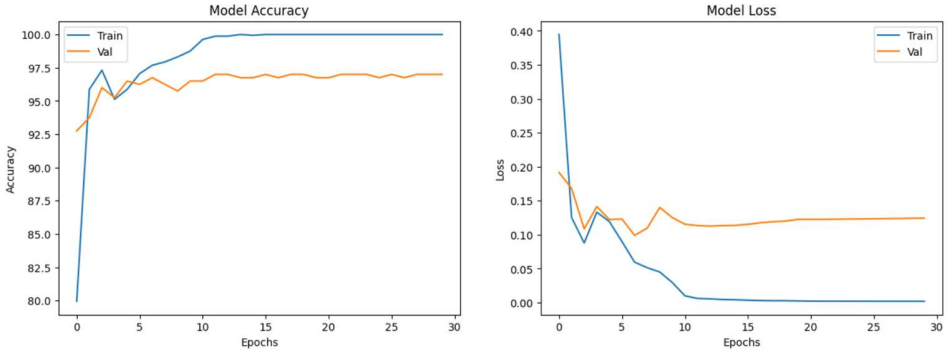

Fig. 8. Performance curve of the model on training and validation data

In Task A, we observed that the training accuracy of the initial model consistently increased, while the validation accuracy plateaued, indicating a classic sign of overfitting. To address this, we considered strategies such as Dropout, weight regularization, and data augmentation. By employing appropriate regularization and data augmentation, the performance curves for training and validation became more aligned, mitigating the overfitting issue.

For Task B, we compared the performance of CNNs and ANNs. Due to its convolutional operations, CNNs effectively capture local features and spatial structures in images, whereas ANNs might overlook certain pivotal local patterns. Additionally, although CNNs demand higher computational resources, their parameter-sharing property often results in fewer parameters than ANNs. Overall, for image data, CNNs typically outperform ANNs in performance.

Lastly, in Task C, we explored CNNs utilizing GAP. GAP not only substantially reduces the model's parameter count but also eliminates the need for manual feature dimension calculations, enhancing the model's generalization capabilities. The model with GAP achieved a remarkable 100% accuracy on the training data and exhibited excellent performance on the validation data. While the model has already demonstrated stellar results on the training data, there's potential to further elevate its performance on validation data through additional hyperparameter tuning and data augmentation techniques.