NYCU Pattern Recognition, Homework 4

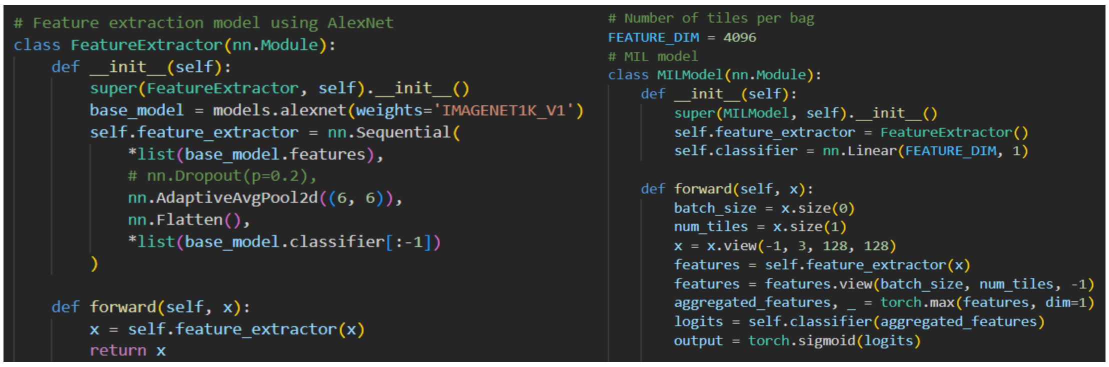
[312552056], [鄭璟翰]

Part. 1, Kaggle (70% [50% comes from the competition]):

(10%) Implementation Details

1. Model architecture

Use AlexNet pre-train model as feature extractor, and ignore last classifier layer, and use maxpooling to find aggregation feature. Finally, use a linear and sigmoid activation function to classify.



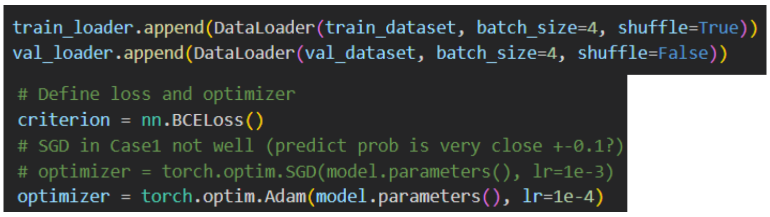
```
# Feature extraction model using AlexNet
class FeatureExtractor(nn.Module):
    def __init__(self):
        super(FeatureExtractor, self).__init__()
        base_model = models.alexnet(weights='IMAGENET1K_V1')
        self.feature_extractor = nn.Sequential(
            *list(base_model.features),
            # nn.Dropout(p=0.2),
            nn.AdaptiveAvgPool2d((6, 6)),
            nn.Flatten(),
            *list(base_model.classifier[:-1])
        )

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

```
# Number of tiles per bag
FEATURE_DIM = 4096
# MIL model
class MILModel(nn.Module):
    def __init__(self):
        super(MILModel, self).__init__()
        self.feature_extractor = FeatureExtractor()
        self.classifier = nn.Linear(FEATURE_DIM, 1)

    def forward(self, x):
        batch_size = x.size(0)
        num_tiles = x.size(1)
        x = x.view(-1, 3, 128, 128)
        features = self.feature_extractor(x)
        features = features.view(batch_size, num_tiles, -1)
        aggregated_features, _ = torch.max(features, dim=1)
        logits = self.classifier(aggregated_features)
        output = torch.sigmoid(logits)
```

2. hyperparameters
Batch Size:4
Loss: Binary Cross Entropy
Learning Rate: 0.0001
Optimizer: Adam



```
train_loader.append(DataLoader(train_dataset, batch_size=4, shuffle=True))
val_loader.append(DataLoader(val_dataset, batch_size=4, shuffle=False))
```

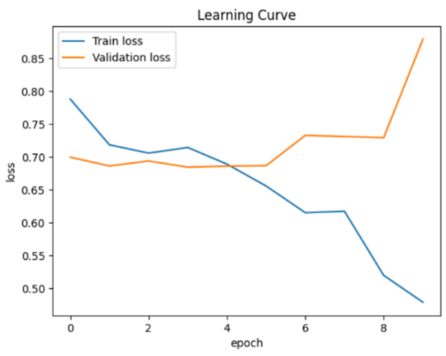
```
# Define loss and optimizer
criterion = nn.BCELoss()
# SGD in Case1 not well (predict prob is very close +-0.1?)
# optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
```

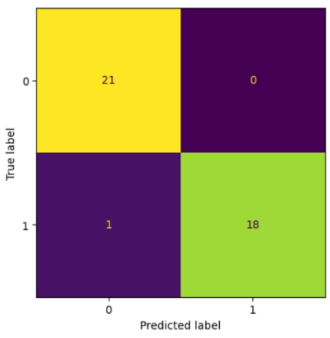
3. Training strategy

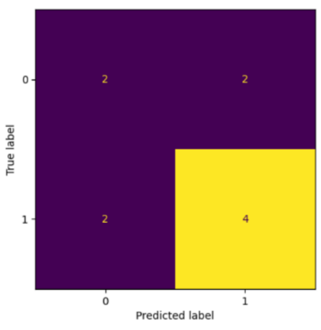
Because the training data is very large (2 * 150 bags * 256 * 128 * 128 * 3), loading all the data at once would exceed the memory limit on Kaggle. Therefore, I chose to load only 50 bags at a time and convert the data into a data loader format, training each data loader sequentially. Initially, I attempted to use ResNet50 as the feature base model, but the model weight size was too large, causing Out of memory error on Kaggle. Consequently, I find the smaller AlexNet as the feature base model. Lastly, I tried using Resnet18 as the base model for training. Although the accuracy on the training data improved significantly, the accuracy on the validation data did not change accordingly. (see resnet18-train.ipynb and resnet18-interface.ipynb in other_test_code)

(10%) Experimental Results

1. Learning curve, Evaluation metrics of training data and validation data.

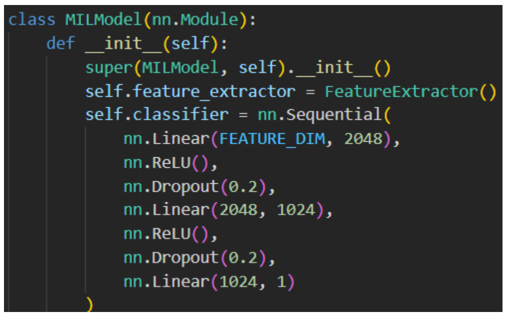






2. Ablation Study

I attempted to add several layers of activation functions and dropout to the final classification layer in MIL model to see if the results would improve. However, compared to the original structure with only one layer, the prediction results on both the training data and validation data did not improve. This indicates that the additional layers did not yield better performance.



```python
class MILModel(nn.Module):
    def __init__(self):
        super(MILModel, self).__init__()
        self.feature_extractor = FeatureExtractor()
        self.classifier = nn.Sequential(
            nn.Linear(FEATURE_DIM, 2048),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(1024, 1)
        )
```

(see other_code_test/alexnet_ablation_test.ipynb for details)

Part. 2, Questions (30%):

1. (10%) Why Sigmoid or Tanh is not preferred to be used as the activation function in the hidden layer of the neural network? Please answer in detail.
   Both Sigmoid and Tanh functions saturate for large positive and large negative inputs, meaning their gradients approach zero in these regions. When the input to these functions is very high or very low, the gradient becomes extremely small, and it would cause Vanishing Gradient Problem. In neural network, gradients are propagated backward through the network to update the weights during backpropagation. Another reason is that both Sigmoid and Tanh involve exponential calculations, which are computationally expensive compared to simpler functions like ReLU. If we use Sigmoid or Tanh in hidden layer, it may consume too much time.

2. (10%) What is overfitting? Please provide at least three techniques with explanations to overcome this issue.
   Overfitting occurs when, as training progresses, the model's performance improves on the training data but does not improve, or even worsens, on the validation data. The first method to overcome overfitting is Early Stopping. This involves stopping the training as soon as it is detected that the validation loss is no longer decreasing, thereby preventing the model from overly fitting the training data and ignoring the validation data. The second method is to include dropout terms in the model. During each training session, this randomly drops certain units, ensuring that the data used for each session varies. This prevents the model from overly conforming to specific data. The third method involves adding a regularization term to the model weights, which prevents the weights from becoming too large. When weights are excessively large, it is likely overfitting, as such weights indicate the model is overly trying to match the data's performance.

3. (15%) Given a valid kernel $k_1(x, x')$, prove that the following proposed functions are or are not valid kernels. If one is not a valid kernel, give an example of $k(x, x')$ that the corresponding K is not positive semidefinite and show its eigenvalues.

a. $k(x,x') = k_1(x,x') + ||x||^2$

Let $k_2(x,x') = ||x||^2$, and Gram Matrix of $k_2(x,x')$is $\begin{bmatrix} ||x_1||^2 & 0 \\ 0 & ||x_2||^2 \end{bmatrix}$, which eigenvalues is always positive. Thus, $k_2(x,x') = ||x||^2$ is a valid kernel.

By (6.17), $k(x,x') = k_1(x,x') + ||\mathrm{x}||^2 = k_1(x,x') + k_2(x,x')$ is also a valid kernel. Therefore, $k(x,x') = k_1(x,x') + ||\mathrm{x}||^2$ is a valid kernel.

b. $k(x,x') = k_1(x,x') - 1$

Let Gram Matrix of $k_1(x,x')$ to be $K_1 = \begin{bmatrix} k_1(x_1,x_1) & k_1(x_2,x_1) \\ k_1(x_2,x_1) & k_1(x_2,x_2) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, which is a PSD Matrix. Thus, Gram Matrix of $k(x,x')$ can be written as $K_1 = \begin{bmatrix} k_1(x_1,x_1) - 1 & k_1(x_2,x_1) - 1 \\ k_1(x_2,x_1) - 1 & k_1(x_2,x_2) - 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$, which eigenvalues are ± 1, is not a PSD Matrix. So $k(x,x') = k_1(x,x') - 1$ is not a valid kernel.

c. $k(x,x') = k_1(x,x') + \exp(x^T x')$

By (6.20), let $A = I$, $k_2(x,x') = x^T A x' = x^T I x' = x^T x'$ is a valid kernel.

By (6.16), $k_3(x,x') = \exp(x^T x') = \exp(k_2(x,x'))$ is a valid kernel.

By (6.17), $k(x,x') = k_1(x,x') + \exp(x^T x') = k_1(x,x') + k_3(x,x')$ is a valid kernel. Therefore, $k(x,x') = k_1(x,x') + \exp(x^T x')$ is a valid kernel.

d. $k(x,x') = \exp(k_1(x,x')) - 1$

By Taylor series, $\exp(k_1(x,x')) = 1 + k_1(x,x') + \frac{k_1(x,x')^2}{2!} + \frac{k_1(x,x')^3}{3!} + \ldots$

Then $k(x,x') = \exp(k_1(x,x')) - 1 = 1 + k_1(x,x') + \frac{k_1(x,x')^2}{2!} + \frac{k_1(x,x')^3}{3!} + \ldots - 1 = k_1(x,x') + \frac{k_1(x,x')^2}{2!} + \frac{k_1(x,x')^3}{3!} + \ldots$

By (6.15), $k_1(x,x')^2, k_1(x,x')^3, k_1(x,x')^4, \ldots$ are valid kernels.

By (6.13), $\frac{k_1(x,x')^2}{2!}, \frac{k_1(x,x')^3}{3!}, \ldots$ are valid kernels. (divide a constant)

By (6.17), $k(x,x') = k_1(x,x') + \frac{k_1(x,x')^2}{2!} + \frac{k_1(x,x')^3}{3!} + \ldots$ is a valid kernel.

So, $k(x,x') = \exp(k_1(x,x')) - 1$is a valid kernel.