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

Name: 吳欣鴻

Student ID: 112066523

A. Specifics of the dataset

1. *Load Dataset & View Data Shapes*

```
x_train = np.transpose(np.load('report_train.npy'), (0, 3, 1, 2))
x_val = np.transpose(np.load('report_val.npy'), (0, 3, 1, 2))
```

Load data from the files report_train.npy and report_val.npy.

Use NumPy to transpose the axes of the data to ensure the correct order in PyTorch.

```
print(f'Shape of x_train: {x_train.shape}')
print(f'Shape of x_val: {x_val.shape}')
```

Print the shapes of the input data x_train and x_val.

2. *Create Labels*

```
num_classes = 6

# Create labels
y_train = np.concatenate([np.full(425, i) for i in range(num_classes)])
y_val = np.concatenate([np.full(100, i) for i in range(num_classes)])
```

Create arrays of labels corresponding to each class.

```
x_train = torch.from_numpy(x_train).float()
y_train = torch.from_numpy(y_train).long()

x_val = torch.from_numpy(x_val).float()
y_val = torch.from_numpy(y_val).long()
```

Convert labels to PyTorch tensors.

3. *Create Datasets and Data Loaders*

```
train_dataset = TensorDataset(x_train, y_train)
val_dataset = TensorDataset(x_val, y_val)
```

Use PyTorch's TensorDataset to create training and validation datasets.

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

Use DataLoader to create the corresponding data loaders.

4. *Plot Data Samples*

```
class_names = ['Tumor', 'Stroma', 'Complex', 'Lympho', 'Debris', 'Mucosa']

num_classes = 6
samples_per_class = 3

fig, axes = plt.subplots(samples_per_class, num_classes, figsize=(12, 6))

plt.subplots_adjust(left=0.05, right=0.95, bottom=0.05, top=0.95, wspace=0.1, hspace=0.1)

for class_idx in range(num_classes):
    indices_of_class = np.where(y_train == class_idx)[0]
    random_indices = random.sample(list(indices_of_class), samples_per_class)

    for i in range(samples_per_class):
        ax = axes[i, class_idx]
        img = x_train[random_indices[i]].numpy().transpose((1, 2, 0))
        img = img / img.max()
        ax.imshow(img)
        ax.axis('off')

        if i == 0:
            ax.set_title(f'{class_idx} {class_names[class_idx]}', pad=3)
```

Use Matplotlib to plot sample images from each class in the training dataset.

These operations ensure that the data is successfully loaded, normalized, and prepared for training and validation in a deep learning model. Additionally, the plotted sample images help visualize the features of the data.

Results:

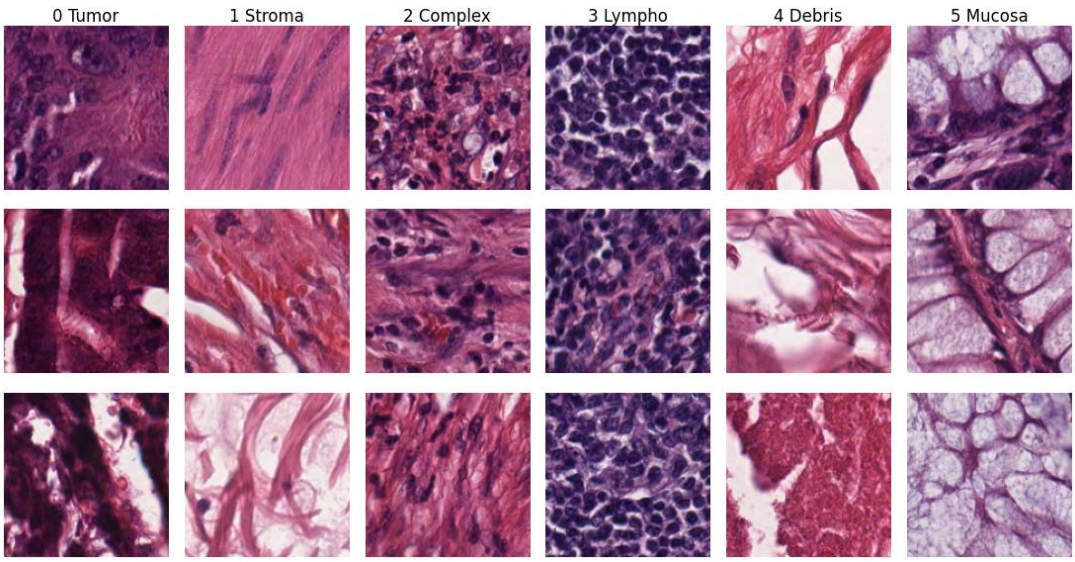


Fig. Based on the two settings of "num_classes = 6" and "samples_per_class = 3" (equivalent to a 3*6 matrix), this schematic diagram of a total of 6 categories of cell slices is obtained.

B. Training the model & Evaluating

1. *Training the Neural Network*

(1) Model Architecture:

The ResNet50 model is loaded from torchvision's models. The final fully connected layer ("fc") is modified to output 6 classes. The model's parameters are frozen, except for the final layer.

(2) Training Loop:

The script defines a training loop using cross-entropy loss and the Adam optimizer. Training is performed for 30 epochs.

Set epochs = 30；learning rate to 10^{-4}

```
epochs = 30  optimizer = optim.Adam(model.parameters(), lr=1e-4)
```

(3) Performance Tracking:

Train and validation losses, as well as accuracies, are tracked during training.

(4) Visualization:

At the end of training, the script generates a plot showing the training and validation accuracy and loss over epochs.

Analysis:

(1) The training loop iterates over the dataset for 30 epochs, updating the model parameters, and adjusting the learning rate using cosine annealing.
(2) The model's performance is evaluated on both the training and validation sets.
(3) The script saves the model with the best validation accuracy as 'model_classification.pth'.

Results:

Train accuracy: 88%；Validation accuracy: 85.83%

Train loss: 0.3764；Validation loss: 0.4322

2. *Evaluating Your Trained Model*

This part is used to evaluate the performance of the model on test data.

(1) Test Data Preprocessing:

Load the test data ("report_test.npy") and transpose it to match the model's input format. Create PyTorch tensors and convert them to floating-point data type.

(2) Data Loading and Model Loading:

Combine the test data and labels into a dataset (“TensorDataset”). Create a DataLoader to load data in batches (“test_loader”). Load previously trained model weights.

(3) Model Evaluation:

Set the model to evaluation mode. Use “test_loader” to load test data and make predictions with the model. Calculate test accuracy.

Analysis:

(1) This code evaluates the model's performance on test data and prints the test accuracy.

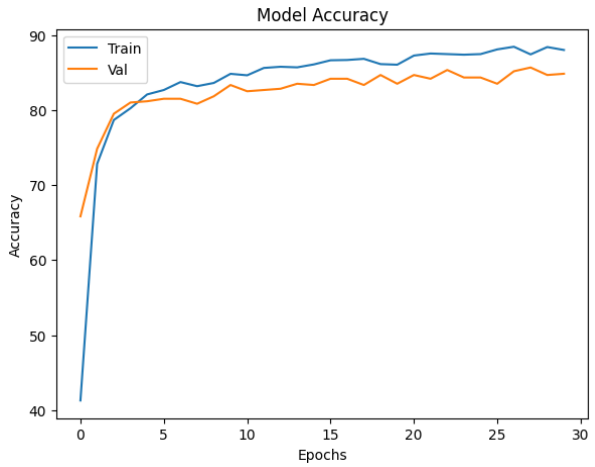
(2) Note that during model evaluation, the model is in eval mode, which means it doesn't compute gradients, helping to save memory and speed up the process.

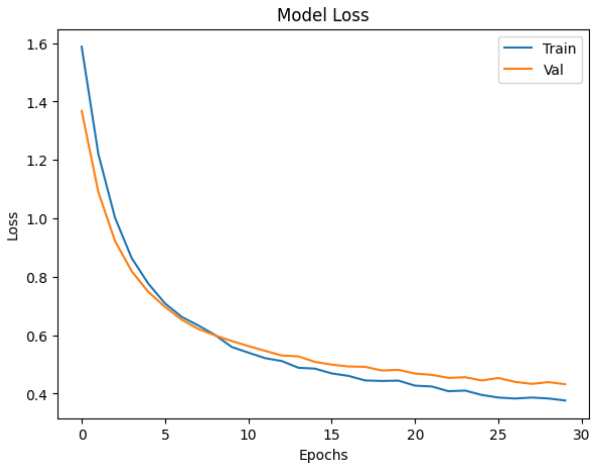
(3) Test accuracy is a performance metric indicating how well the model performs on new, unseen data.

Results:

Test accuracy is 85.67%

C. Expected model performance





Test accuracy is 85.66666666666667%