Report of Q3
Short Video Classification

50015627 JIANG Zhuoyang

1. **Question Description:**

Short video applications are becoming more and more popular among the young. In reality, internet companies generally use automatic classification algorithms to process large amounts of short video uploaded by users. We are asked to implement a short video classification algorithm, which contains four parts:

- Load Data Especially Video Data
- Select methods and build models
- Model Training and Evaluation
- Use Model to Do Classification On Test set

2. Load Data Especially Video Data

I designed a dataloader to load a dataset for weather recognition tasks. It first converts images into a format acceptable to neural networks, while assigning corresponding category labels to each image. And set the batch size to provide data in batches during training, while shuffling the data before each epoch to increase the randomness of model training.

(1) Load the Training Data Pair (Video Filename, Tag)

	video	tag
0	873879927.mp4	0
1	872438072.mp4	0
2	796902701_219_229.mp4	1
3	829923135_39_49.mp4	1
4	859022208.mp4	0
...	...	...
2058	304035088.mp4	10
2059	303734394.mp4	10
2060	892673978.mp4	0
2061	613605247.mp4	9
2062	305624414.mp4	6

2063 rows × 2 columns

Figure.1 Data Pair (Video Filename, Tag)

(2) Load the Video with Specific Shape：

Each MP4 Video file is loaded as a image-frame-batch with image-size = (64, 64, 3) and frame-num = 20.

```
path = 'Data_Q3//train_video//'
x = []
y = []

img_size = (64, 64, 3)
frames = 20

# load_ MP4 function
def load_mp4(file_path, img_size, frames):
    cap = cv2.VideoCapture(file_path)
    video_frames = []
    for _ in range(frames):
        ret, frame = cap.read()
        if ret:
            frame = cv2.resize(frame, (img_size[0], img_size[1]))
            frame = np.transpose(frame, (2, 0, 1))
            video_frames.append(frame)
        else:
            break
    cap.release()
    return video_frames
```

Figure.2 Data Pair (Video Filename, Tag)

(3) Use multithreading to load video data:

Video data processing is inefficient, so we use multithreading to improve the efficiency of importing and processing video files:

```
# Use multithreading to load video data
def process_video(filename):
    if filename.endswith(".mp4"):
        v = load_mp4(path + filename, img_size, frames)
        tag = train_tag['tag'][train_tag['video'] == filename].values[0]
        return v, tag

# Loading video data with multiple threads
with ThreadPoolExecutor() as executor:
    future_to_file = {executor.submit(process_video, filename): filename for filename in os.listdir(path)}

    completed_count = 0
    for future in tqdm(as_completed(future_to_file), total=len(future_to_file)):
        filename = future_to_file[future]
        try:
            result = future.result()
            if result is not None:
                x.append(result[0])
                y.append(result[1])
            completed_count += 1
#             print(f"Processed {completed_count}/{len(future_to_file)} videos")
        except Exception as e:
            print(f"Error processing {filename}: {e}")

#         if len(x) == size:
#             break

print("Video processing completed.")
```

```
100%|██████████| 2063/2063 [00:38<00:00, 53.01it/s]
Video processing completed.
```

Figure.3 Multithreading Video Loading

3. Select methods and build models

(1) Training Sample's Form:

```
label:  9
shape:  (20, 3, 64, 64)
```

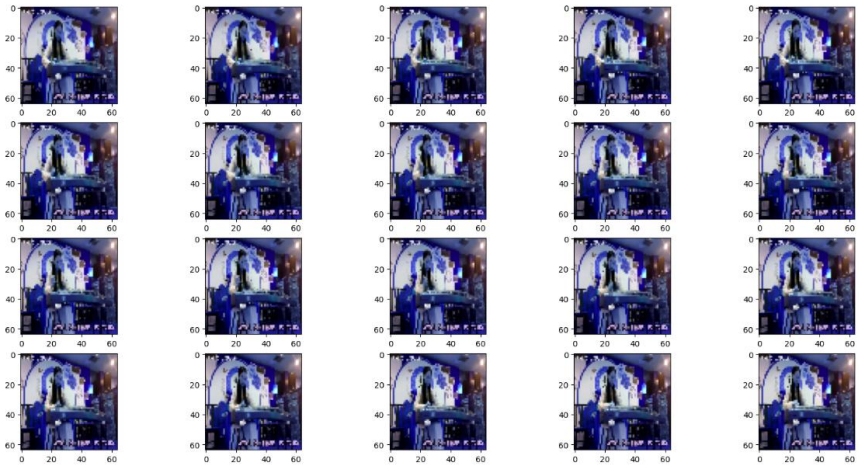


Figure.4 Training Sample's Form

(2) Model-1: 3D Convolution Deep Model

Due to the nature of working with video data, my initial approach involved a model based on a 3D Convolutional Neural Network. I built this model structure using the PyTorch framework, as shown in the diagram.

```
def forward(self, x):
    # The shape of x: (batch_size, channels, numframes, height, width)
    x = x.permute(0, 2, 1, 3, 4)
    x = self.relu(self.conv1(x))
```

Figure.5 Input Organization of 3D Convolution Deep Model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv3d-1       [-1, 16, 20, 64, 64]           1,312
              ReLU-2       [-1, 16, 20, 64, 64]               0
         MaxPool3d-3       [-1, 16, 10, 32, 32]               0
            Conv3d-4       [-1, 32, 10, 32, 32]          13,856
              ReLU-5       [-1, 32, 10, 32, 32]               0
         MaxPool3d-6        [-1, 32, 5, 16, 16]               0
            Conv3d-7        [-1, 64, 5, 16, 16]          55,360
              ReLU-8        [-1, 64, 5, 16, 16]               0
         MaxPool3d-9          [-1, 64, 2, 8, 8]               0
           Linear-10                 [-1, 128]       1,048,704
             ReLU-11                 [-1, 128]               0
           Linear-12                  [-1, 15]           1,935
================================================================
Total params: 1,121,167
Trainable params: 1,121,167
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.94
Forward/backward pass size (MB): 27.88
Params size (MB): 4.28
Estimated Total Size (MB): 33.09
----------------------------------------------------------------
```

Figure.6 3D Convolution Deep Model

(3) Model-2: 2D Convolution Deep Model

Based on observations of the data format, I noticed minimal frame-to-frame variations in the short video data. Hence, I considered flattening all frames into 2D and then constructing a model using a 2D Convolutional Neural Network. I built this model structure using the PyTorch framework, as depicted in the diagram.

```python
def forward(self, x):
    batch_size, num_frames, channels, height, width = x.size()
    x = x.view(batch_size * num_frames, channels, height, width)  # Flatten the video frames first
    x = self.relu(self.conv1(x))
```

Figure.7 Input Organization of 2D Convolution Deep Model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 64, 64]             448
              ReLU-2         [-1, 16, 64, 64]               0
         MaxPool2d-3         [-1, 16, 32, 32]               0
            Conv2d-4         [-1, 32, 32, 32]           4,640
              ReLU-5         [-1, 32, 32, 32]               0
         MaxPool2d-6         [-1, 32, 16, 16]               0
            Conv2d-7         [-1, 64, 16, 16]          18,496
              ReLU-8         [-1, 64, 16, 16]               0
         MaxPool2d-9           [-1, 64, 8, 8]               0
           Linear-10                 [-1, 128]         524,416
             ReLU-11                 [-1, 128]               0
           Linear-12                  [-1, 15]           1,935
================================================================
Total params: 549,935
Trainable params: 549,935
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.94
Forward/backward pass size (MB): 1.97
Params size (MB): 2.10
Estimated Total Size (MB): 5.01
----------------------------------------------------------------
```

Figure.8 2D Convolution Deep Model

4. Model Training and Evaluation

(1) Train 3D Convolution Deep Model:

① Training Configuration:

```python
# Define lists to store training and validation losses
output_model_path = "Q4_model_3dConv.bin"

train_losses = []

train_accuracies = []
val_accuracies = []

# Defining loss functions and optimizers
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 32
```

Figure.9 3D Convolution Deep Model's Training Configuration

② **Training Processing:**

```
    print(f"Epoch {epoch + 1}/{epochs} - Train Loss: {train_loss} - Train Acc: {train_accuracy:.2f}% - Val Acc: {val_accuracy:.2f}%")
    # Save the model if needed
    torch.save(model.state_dict(), output_model_path)
```

```
Epoch 27/32 - Train Loss: 0.007350565043947581 - Train Acc: 99.76% - Val Acc: 44.55%
Epoch 28/32 - Train Loss: 0.005616455410527131 - Train Acc: 99.76% - Val Acc: 44.31%
Epoch 29/32 - Train Loss: 0.007090178141712482 - Train Acc: 99.76% - Val Acc: 44.31%
Epoch 30/32 - Train Loss: 0.007704861721966092 - Train Acc: 99.70% - Val Acc: 44.07%
Epoch 31/32 - Train Loss: 0.005406787479721778 - Train Acc: 99.82% - Val Acc: 43.34%
Epoch 32/32 - Train Loss: 0.005743895891580582 - Train Acc: 99.88% - Val Acc: 44.31%
```

Figure.10 3D Convolution Deep Model's Training Process

③ **Training Result Evaluation:**

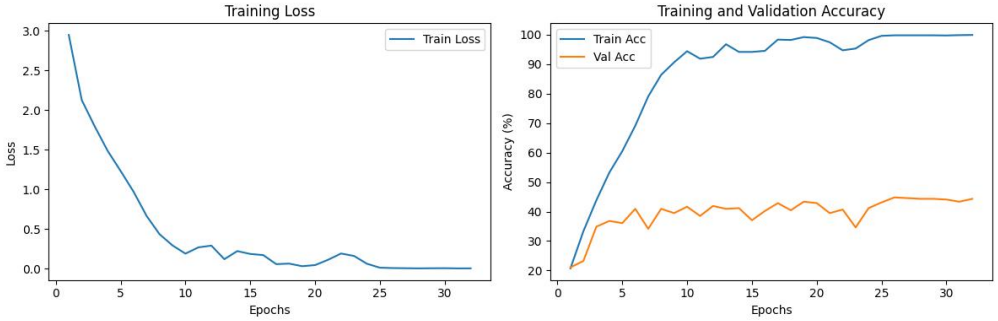


Figure.11 Training Result of 3D Convolution Deep Model

(2) Train 2D Convolution Deep Model:

① **Training Configuration:**

```
# Define lists to store training and validation losses
output_model_path = "Q4_model_2dConv.bin"

train_losses = []

train_accuracies = []
val_accuracies = []

# Defining loss functions and optimizers
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 32

```

Figure.12 2D Convolution Deep Model's Training Configuration

② **Training Processing:**

```
    print(f"Epoch {epoch + 1}/{epochs} - Train Loss: {train_loss} - Train Acc: {train_accuracy:.2f}% - Val Acc: {val_accuracy:.2f}%")
    # Save the model if needed
    torch.save(model.state_dict(), output_model_path)
```

```
Epoch 27/32 - Train Loss: 0.013233616085472022 - Train Acc: 99.70% - Val Acc: 48.91%
Epoch 28/32 - Train Loss: 0.008163489600147063 - Train Acc: 99.76% - Val Acc: 48.67%
Epoch 29/32 - Train Loss: 0.009183248898886212 - Train Acc: 99.70% - Val Acc: 48.43%
Epoch 30/32 - Train Loss: 0.007225286210025563 - Train Acc: 99.76% - Val Acc: 48.43%
Epoch 31/32 - Train Loss: 0.007138503651479875 - Train Acc: 99.76% - Val Acc: 48.67%
Epoch 32/32 - Train Loss: 0.007188787579239558 - Train Acc: 99.76% - Val Acc: 48.91%
```

Figure.13 2D Convolution Deep Model'sTraining Process

③ **Training Result Evaluation:**

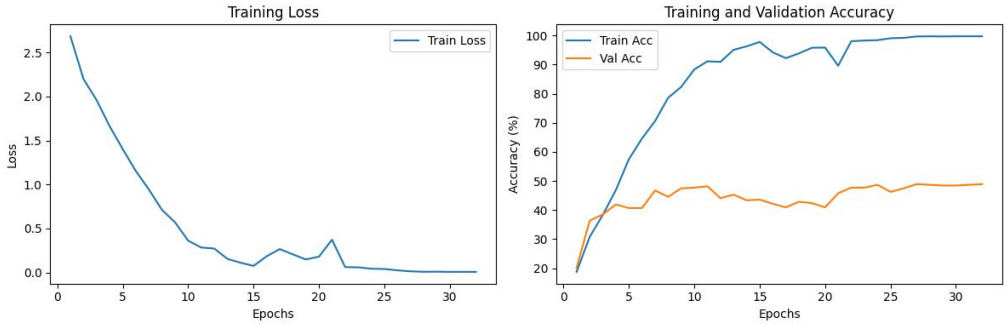
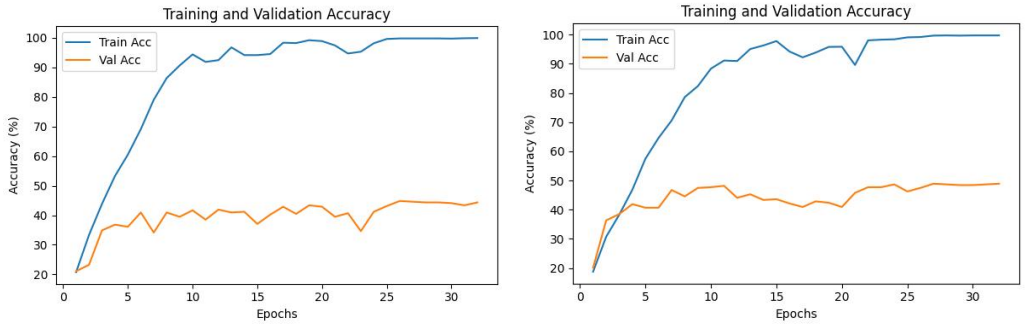


Figure.14 Training Result of 3D Convolution Deep Model

5. Use Model to Do Classification On Test set

I compared the performance of two models on the validation set, and 2D Convolution Deep Model emerged as the superior one. Hence, I ultimately chose to use 2D Convolution Deep Model for classifying the videos in the test set.



3D Convolution Deep Model **2D Convolution Deep Model**

Figure.15 Comparison of Training Evaluation Result

I used 2D Convolution Deep Model for classifying the videos in the test set. The result on the test set shows as below:

	file_name	label
0	304037982.mp4	13
1	303746162.mp4	6
2	303942687.mp4	0
3	30018754.mp4	2
4	304078846.mp4	2
...	...	...
557	897551456.mp4	12
558	898969484_32_42.mp4	4
559	899183470.mp4	9
560	899865823.mp4	3
561	899731707_220_230.mp4	1

562 rows × 2 columns

Figure.16 Classification Result on test set