PR Final

Captcha Hacker

311551170 林琨堯

- Environment details:

  python=3.7

  torch=1.13.1+cu117, install with the follow command

  pip install torch==1.13.1+cu117 torchvision==0.14.1+cu117 torchaudio==0.13.1 --extra-index-url https://download.pytorch.org/whl/cu117

  file relative address:

- Implementation details

  ➢ **Model architecture**

  I use DenseNet201 as the train model of task1, task2, and use efficientnet_v2_l as the train model of task3.

  ➢ **Hyper-parameters**

	Task 1	Task 2	Task 3
Training epochs	50	150	300
Batch size	50	50	25
Learning rate	0.001	0.001	0.01

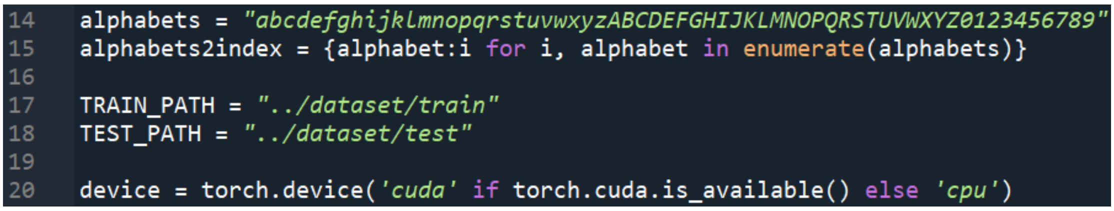
  ➢ **Loss function:** MultiLabelSoftMarginLoss(), optimizes a multi-label one-versus-all loss based on max-entropy. It is a general loss function used for captcha estimation.

  ➢ **Optimizer:** Adam()

  ➢ **Dataset:** In task2, the label of a sample named task2/esc… show ‘04’ in Kaggle, but it shows ‘4’ in csv. Since I can’t save the change if I change the cell format, so I set the label of this image as ‘o4’.

  ➢ **Code: train.py**

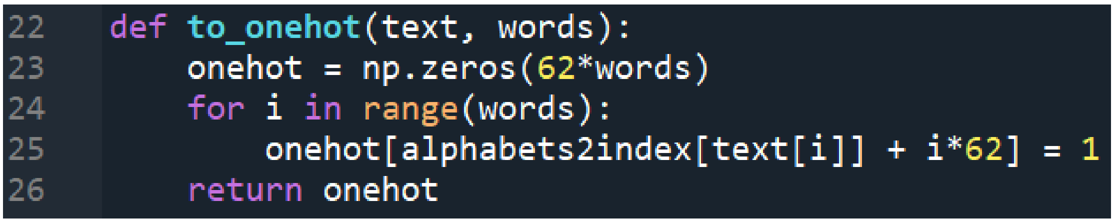
  Variable:



```
alphabets = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
alphabets2index = {alphabet:i for i, alphabet in enumerate(alphabets)}

TRAIN_PATH = "../dataset/train"
TEST_PATH = "../dataset/test"

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

  to_onehot:

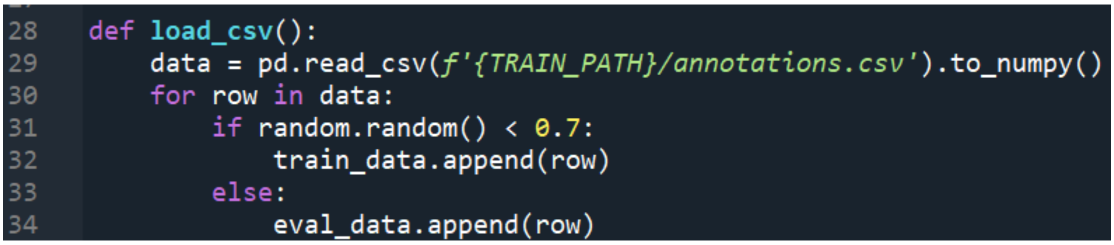
  According to task number, create a n-length vector named onehot, n = {62*1, 62*2, 62*4}



```
def to_onehot(text, words):
    onehot = np.zeros(62*words)
    for i in range(words):
        onehot[alphabets2index[text[i]] + i*62] = 1
    return onehot
```

load_csv:

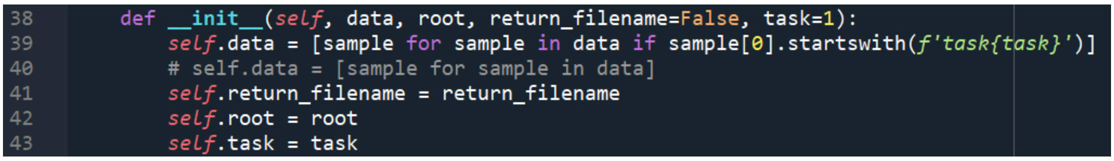
Read data in submission.csv, and then random select 70% as train data, 30% as evaluation data



```
def load_csv():
    data = pd.read_csv(f'{TRAIN_PATH}/annotations.csv').to_numpy()
    for row in data:
        if random.random() < 0.7:
            train_data.append(row)
        else:
            eval_data.append(row)
```

class: TaskDataset:

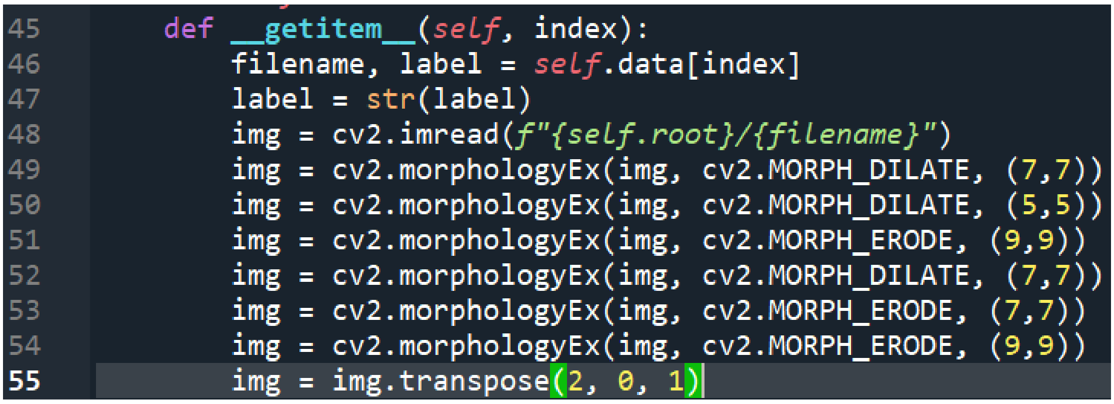
For each task, I trained one model respectively. So, I create their own data.



```
    def __init__(self, data, root, return_filename=False, task=1):
        self.data = [sample for sample in data if sample[0].startswith(f'task{task}')]
        # self.data = [sample for sample in data]
        self.return_filename = return_filename
        self.root = root
        self.task = task
```

Image preprocessing:

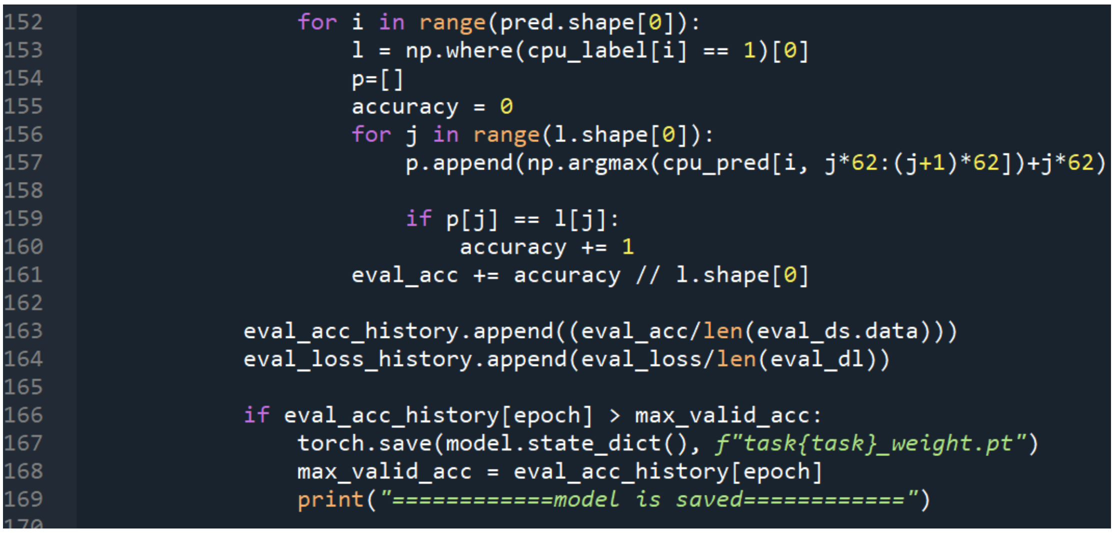
I use OpenCV to load images, and erase the background noise by dilation method to dilate the light part, then make the words bold by erosion method. Finally, return images and one-hot label to train and evaluate, return images and their name to test.



```
    def __getitem__(self, index):
        filename, label = self.data[index]
        label = str(label)
        img = cv2.imread(f"{self.root}/{filename}")
        img = cv2.morphologyEx(img, cv2.MORPH_DILATE, (7,7))
        img = cv2.morphologyEx(img, cv2.MORPH_DILATE, (5,5))
        img = cv2.morphologyEx(img, cv2.MORPH_ERODE, (9,9))
        img = cv2.morphologyEx(img, cv2.MORPH_DILATE, (7,7))
        img = cv2.morphologyEx(img, cv2.MORPH_ERODE, (7,7))
        img = cv2.morphologyEx(img, cv2.MORPH_ERODE, (9,9))
        img = img.transpose(2, 0, 1)
```

train:

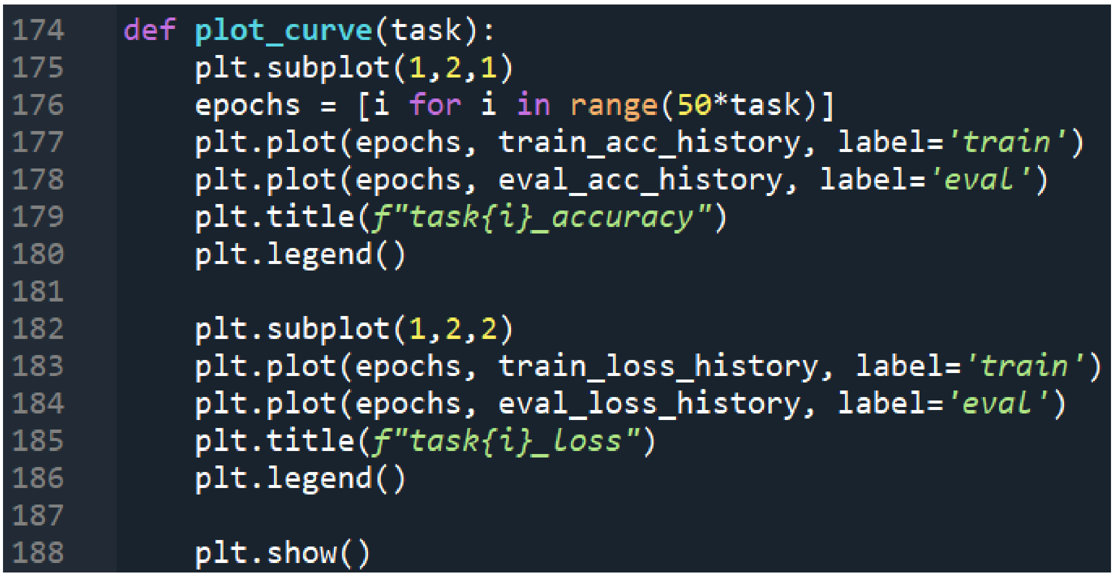
The prediction of each image will find the most possible class in every 62 classes. After calculate the evaluation accuracy, the parameters of the model with the highest evaluation accuracy will be saved. If the prediction accuracy is not good, they can be load for next training.



```
                for i in range(pred.shape[0]):
                    l = np.where(cpu_label[i] == 1)[0]
                    p=[]
                    accuracy = 0
                    for j in range(l.shape[0]):
                        p.append(np.argmax(cpu_pred[i, j*62:(j+1)*62])+j*62)

                        if p[j] == l[j]:
                            accuracy += 1
                    eval_acc += accuracy // l.shape[0]

            eval_acc_history.append((eval_acc/len(eval_ds.data)))
            eval_loss_history.append(eval_loss/len(eval_dl))

            if eval_acc_history[epoch] > max_valid_acc:
                torch.save(model.state_dict(), f"task{task}_weight.pt")
                max_valid_acc = eval_acc_history[epoch]
                print("===========model is saved===========")
```

Plot_curve:

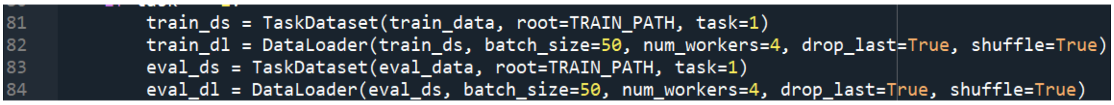
Plot the accuracy and loss curve of training and evaluation



```
def plot_curve(task):
    plt.subplot(1,2,1)
    epochs = [i for i in range(50*task)]
    plt.plot(epochs, train_acc_history, label='train')
    plt.plot(epochs, eval_acc_history, label='eval')
    plt.title(f"task{i}_accuracy")
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(epochs, train_loss_history, label='train')
    plt.plot(epochs, eval_loss_history, label='eval')
    plt.title(f"task{i}_loss")
    plt.legend()

    plt.show()
```

Dataset and Dataloader:

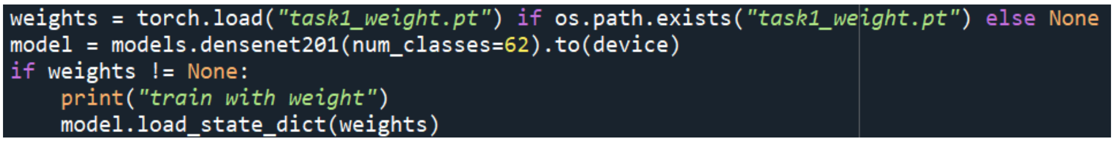
I drop data which is less than batch size, so that, I do not need to deal with them.



```
        train_ds = TaskDataset(train_data, root=TRAIN_PATH, task=1)
        train_dl = DataLoader(train_ds, batch_size=50, num_workers=4, drop_last=True, shuffle=True)
        eval_ds = TaskDataset(eval_data, root=TRAIN_PATH, task=1)
        eval_dl = DataLoader(eval_ds, batch_size=50, num_workers=4, drop_last=True, shuffle=True)
```

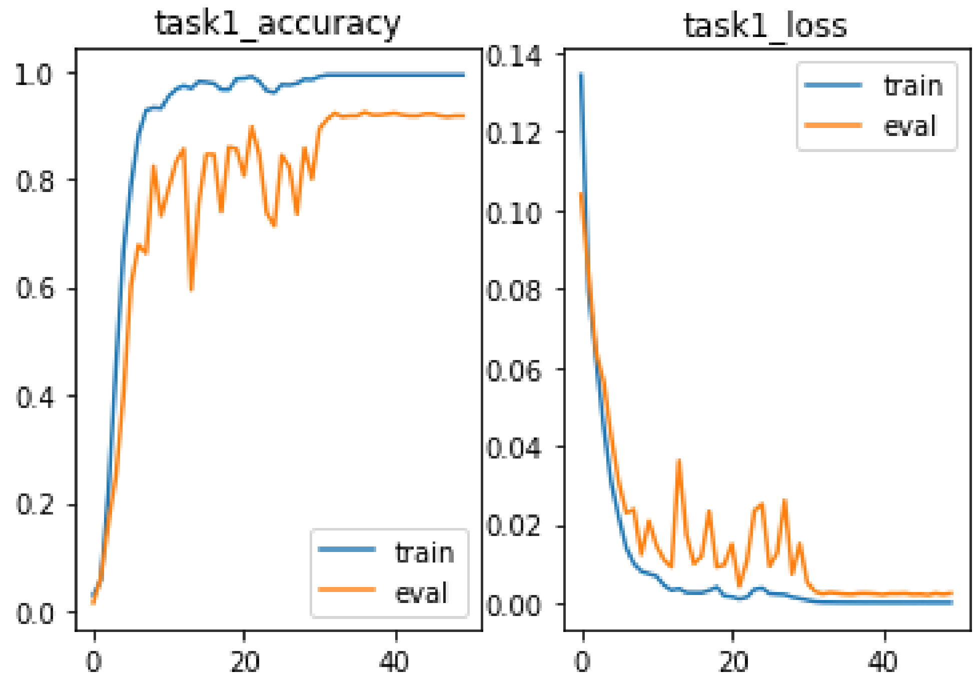
Model:

I choose densenet201 for all task at first, but it occurred overfitting on task3, so I choose efficient-net after try different pretrained model.

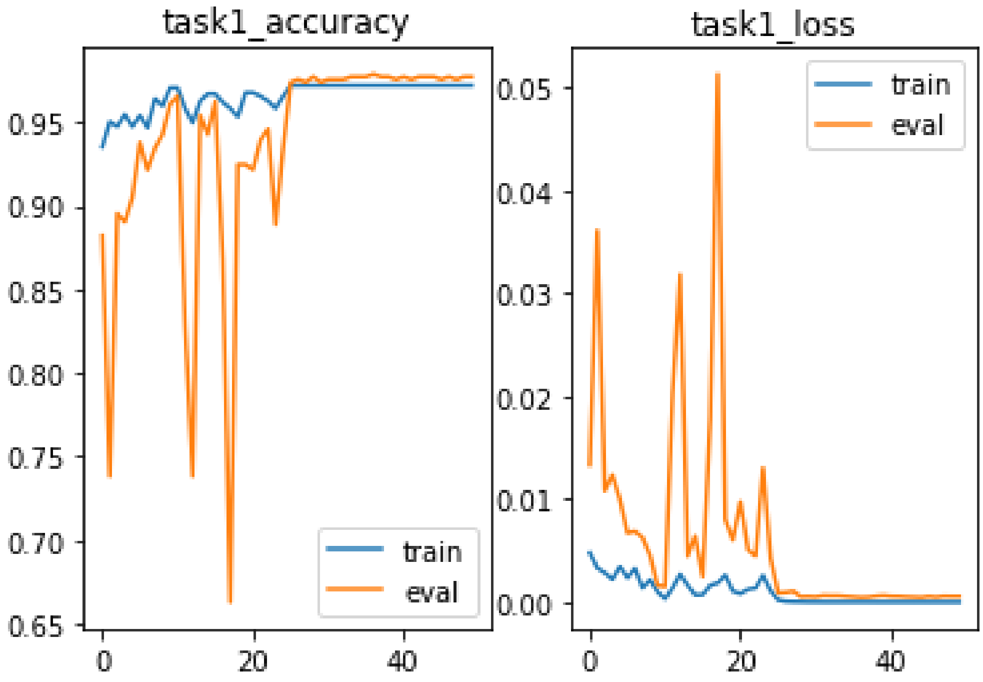


```
weights = torch.load("task1_weight.pt") if os.path.exists("task1_weight.pt") else None
model = models.densenet201(num_classes=62).to(device)
if weights != None:
    print("train with weight")
    model.load_state_dict(weights)
```

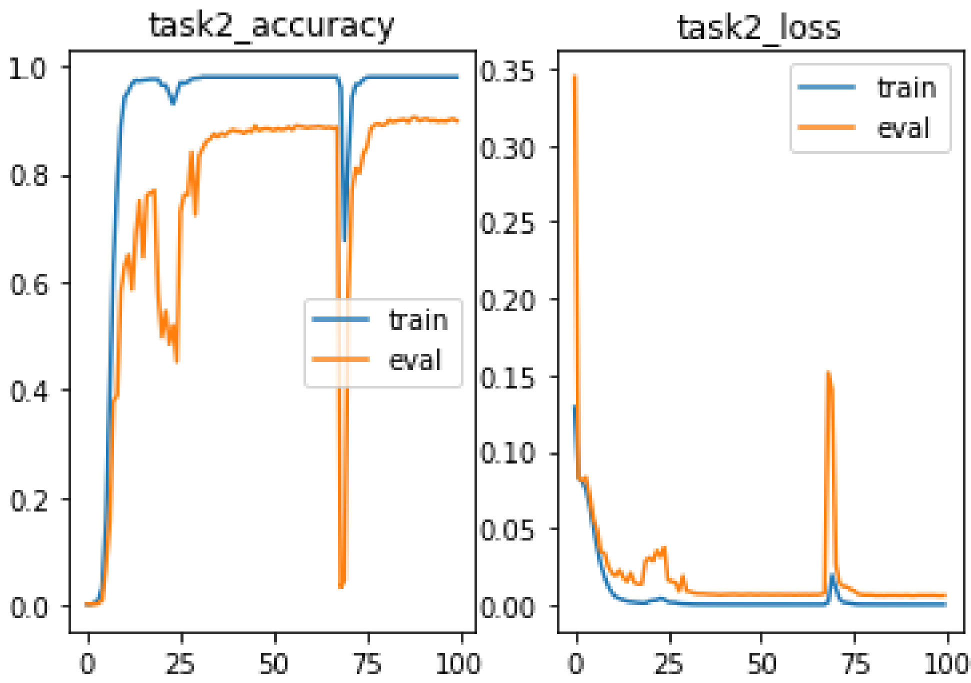
- **Experiment result and analysis**



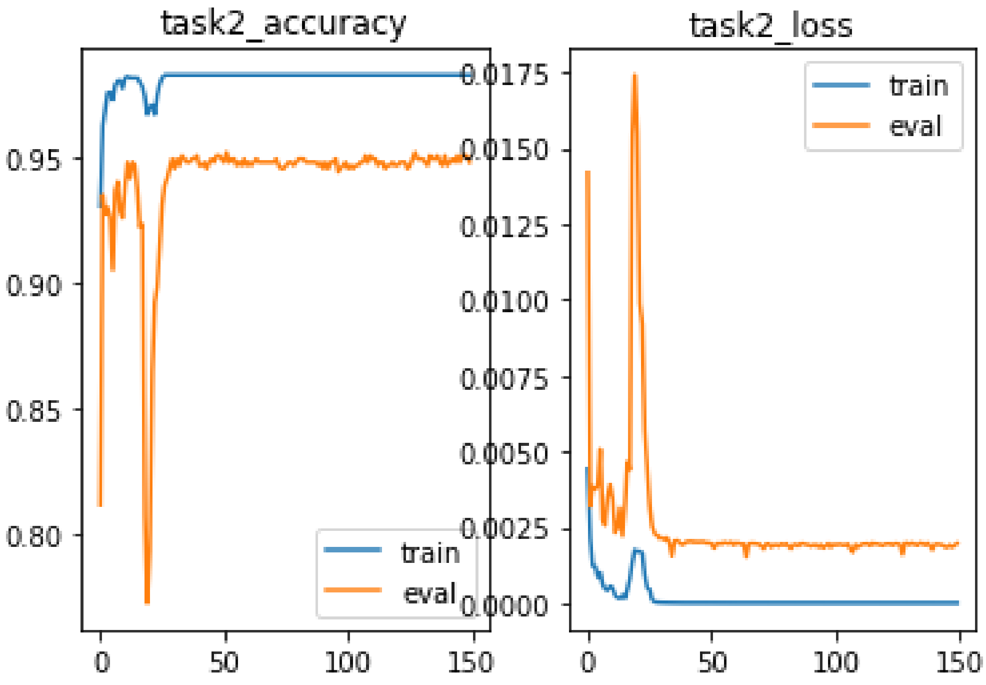
Max eval acc = 0.92

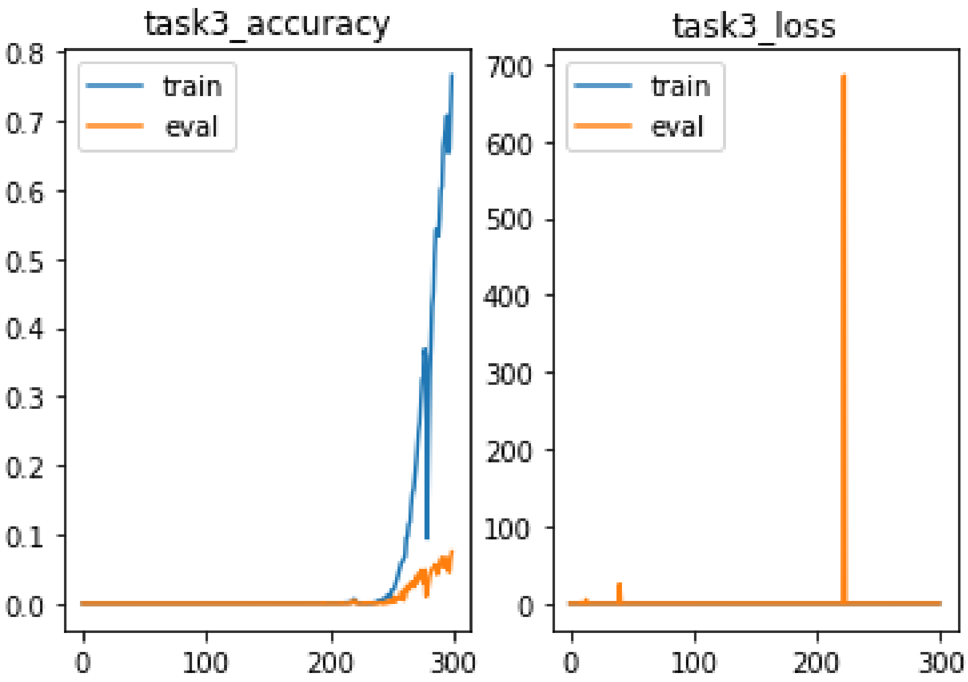


Max eval acc = 0.98

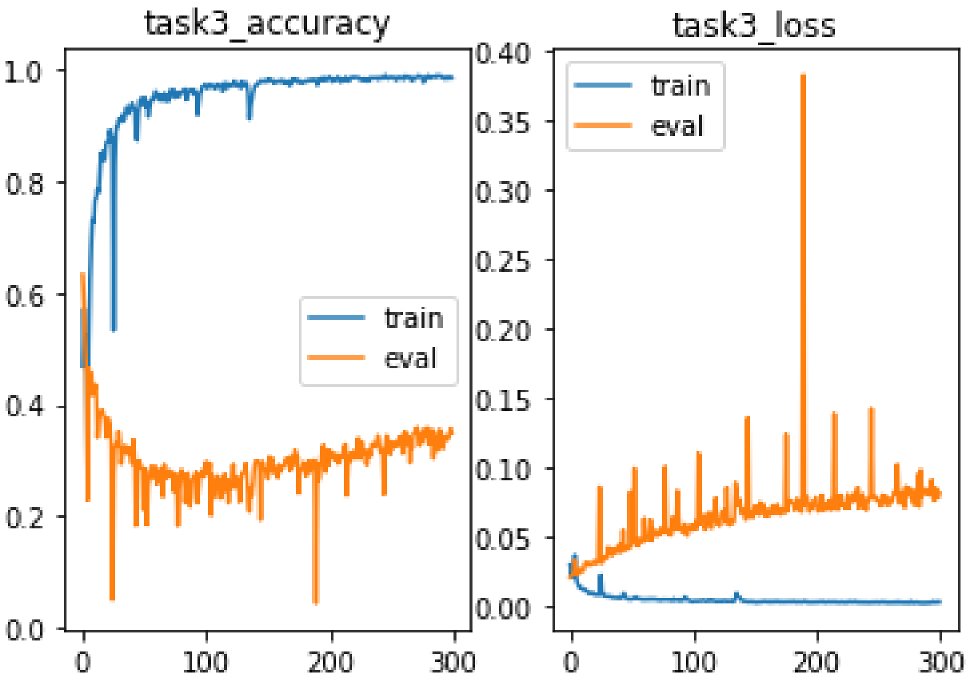


Max eval acc = 0.90

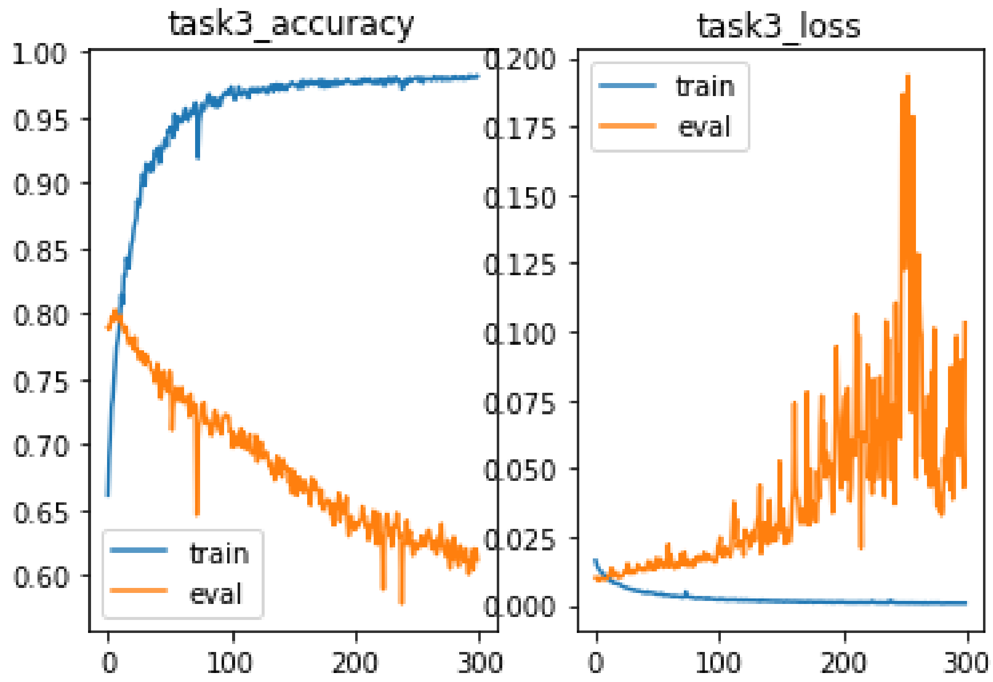




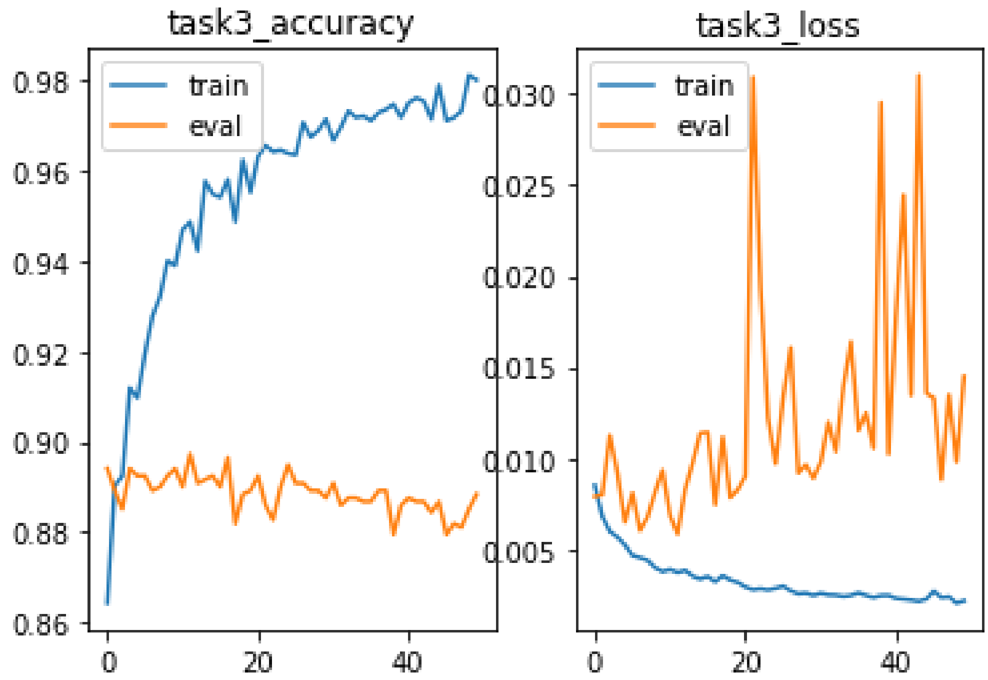
Max eval acc = 0.07



Max eval acc = 0.64



Max eval acc = 0.80, but seams overfitting

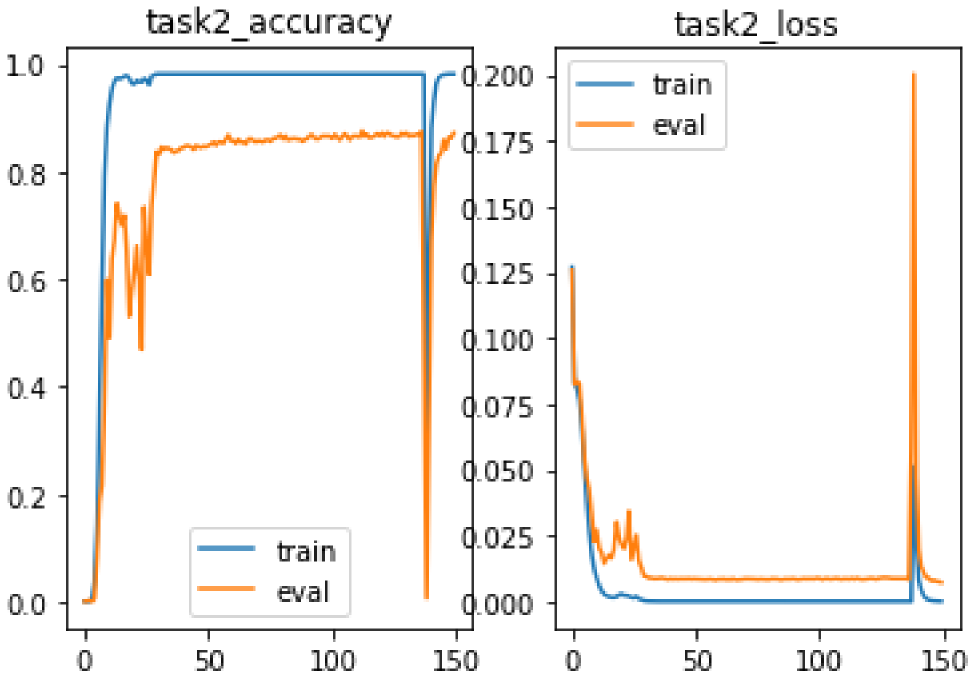


Use the latest weight of the last training to train more 50 epochs, it seems more normal compare with last one.

- **Compare with other method (use task2 as example)**

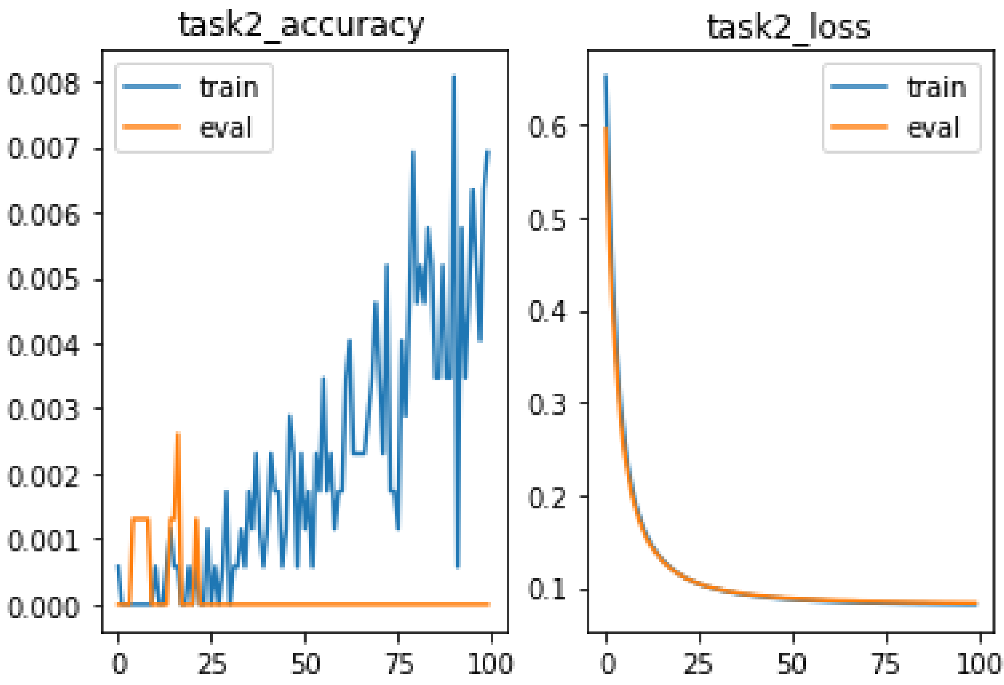
  In this part, I use task2 to try other methods because I consider task1 is too simple because of the only one word and task3 is too complex in contrast.

1. Without image preprocessing



Max eval acc = 0.86 until 100 epochs, and 0.90 with image preprocessing.

2. Using SGD as optimizer



Lower than 0.003

- Others

  The training accuracy of task 3 is very unstable compare with other two tasks. It may occur underfitting or overfitting when I train model with same pretrained weight.

  There are some words which features are very similar, like 1 and l(lower L) and I(capital i), W and w, etc. To separate them is very difficult.