CO101A term project report

Step1: Load data

Load training file into data_list and read all images using PIL to load '.png' format and convert them to numpy array. The image shape should be (num_channels, H, W).

Make a dictionary, read from labels.txt, e. g. label_dict = {'0267.png': 4, ‘0267.png':5}

Iterate the data_list to add every image and label into train sets x and y correspondingly. And stack them to make a 3-D array into 4-D as (N, num_channels, H, W) and (N).

Step2: Define dataset class

For the convenience of reusing code, we write the dataset class and there are 3 parameters to pass into the class(x: image dataset, y: label dataset, transform: to resize, tensor and normalize image datas)

Use torch.utils.data.DataLoader to get a batch of iterable dataset for the main method to traverse.

Step3: Modify the training and testing functions

Write accuracy output in every epoch to result.txt.
Write predict labels of the testing_toy file output into predict_label.txt.

Step4: Train model and optimize it.

Split the train file into 2 files. The first one contains 1-660 images and labels as training, the second one contains 661-860 images and labels as test.
Final setting:
Applies 2 2D convolution over the input image and let the output channel to be 32, use padding kernel_size=5 and stride =1.
Add 2 max_pool with kernel_size=2 and 2 relu.
The output feature tensor from the _forward_features method will have shape (batch_size, n, n, channels). We can reshape it(flatten it) using view(x.size[0], -1)

Optimizer(ADAM vs SGD):
ADAM trains the neural network in less time.

Optimizer	State Memory [bytes]	# of Tunable Parameters	Strengths	Weaknesses
SGD	0	1	Often best generalization (after extensive training)	Prone to saddle points or local minima Sensitive to initialization and choice of the learning rate α
SGD with Momentum	$4n$	2	Accelerates in directions of steady descent Overcomes weaknesses of simple SGD	Sensitive to initialization of the learning rate α and momentum β
AdaGrad	$\sim 4n$	1	Works well on data with sparse features Automatically decays learning rate	Generalizes worse, converges to sharp minima Gradients may vanish due to aggressive scaling
RMSprop	$\sim 4n$	3	Works well on data with sparse features Built in Momentum	Generalizes worse, converges to sharp minima
Adam	$\sim 8n$	3	Works well on data with sparse features Good default settings Automatically decays learning rate α	Generalizes worse, converges to sharp minima Requires a lot of memory for the state
AdamW	$\sim 8n$	3	Improves on Adam in terms of generalization Broader basin of optimal hyperparameters	Requires a lot of memory for the state
LARS	$\sim 4n$	3	Works well on large batches (up to 32k) Counteracts vanishing and exploding gradients Built in Momentum	Computing norm of gradient for each layer can be inefficient

hyper parameters:
n_epochs: tried 50, 100, 200. After around 100 epochs, the trainset accuracy has reached to 1, so I think there is no meaning to continue fitting it.
batch_size_train = 128
batch_size_test = 1000
learning_rate = 0.001, for Adam, it is better to set small learning rate.

Step5: Save the model
Later we can load the save model and use the test function to calculate the accuracy and get the predict_label result.