- **GitHub：https://github.com/dodo0517cc/Visual-DL**

- **Reference：**

  Focal Loss: https://github.com/clcarwin/focal_loss_pytorch

  Focal loss is a loss function proposed in the article “Focal Loss for Dense Object Detection” to decay simple samples. The aim of focal loss is to perform down-weighting for inliers (easy examples), because it hopes to train hard examples as much as possible during the training process and ignore those easy examples. It is an improvement of the standard Cross Entropy Loss. The function is as below.

$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$

  Setting $\gamma > 0$ reduces the relative loss for well-classified samples, putting more focus on hard, misclassified examples.

- **Brief introduction：** There are only 3000 images for training. After counting,

  I found that it’s only 15 images for each class. Obviously, it’s a small-scaled dataset. As a result, I met overfitting problems. In order to deal with the problems, I did some data augmentation, and added Dropout layer in my fully connected layer of my transfer learning model—Densenet161. Also, I added weight decay for L2 regularization in my optimizer—SGD.
  For the training process, I did 5-fold cross validation to valid my training results.

- **Methodology：**

✓ **Data pre-process：**

Transformation for Training（done by Albumentation package）:

1. Random Resized Crop images to 299*299 pixels
2. RGB Shift—set the ranges for changing values for the red channel, green channel and blue channel as 15. Also, set the probability as 0.5.
3. Horizontal Flip—set the probability as 0.5
4. Shift Scale Rotate—set rotation range as 20, probability as 0.5
5. Normalization

Transformation for Validation（done by Albumentation package）:

1. Resized images to 375*375pixels
2. Center Crop to 299*299 pixels
3. Normalization

✓ **Model architecture**：Transfer learning—Densenet161

1. Use the pretrained weights, but unfreeze the layers.
2. Change the original fully connected layer—
   Add a Fully Connected Layer with 1024 output channels
   Apply batch normalization
   Choose ReLU as the activation function
   Add Dropout layer for 0.3 probability to reduce overfitting
   Finally add a Fully Connected Layer with 200 output channels, because there are 200 classes

✓ **Hyperparameters**：

Learning Rate—0.005

Batch Size—64

Epochs—80

Loss—Focal Loss (gamma=4)

Optimizer—SGD (momentum = 0.9, weight_decay = 1e-5)

- **Summary**

To reproduce the submission file:

Step1: Install albumentations package

Step2: Load in the training images and correspond them with their labels

Step3: Split the data for 5-fold validation

Step4: Create Dataset and DataLoader for both training sets and validation sets, also do some augmentation.

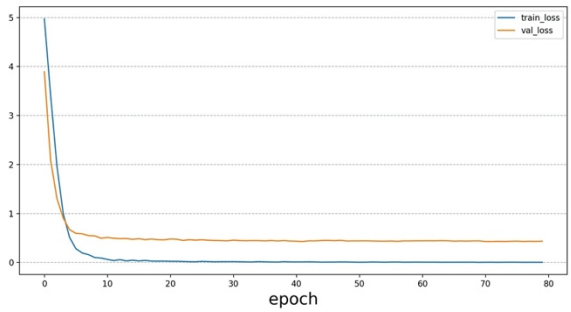
Step5: Train the model for 80 epochs per fold with Densenet model and SGD optimizer, also, set the loss function to focal loss. Save the weights only if the validation accuracy raises. After training, we can get the loss curve and the accuracy curve as below.

Step6: Load in the testing images and the txt file for detailed classes.

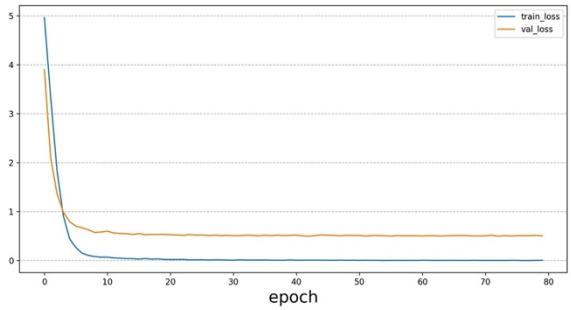
Step7: Create Dataset and DataLoader for testing set. Then, load the weights that is saved previously.(The weights file is too big to upload, so I provide a google drive link for fold0:https://drive.google.com/drive/folders/1FaYXQT4MRJA8A38fXFtqcGu4u3LwpCfb?usp=sharing) The homework is done now by saving the answer file.

✓ **Loss curve：**

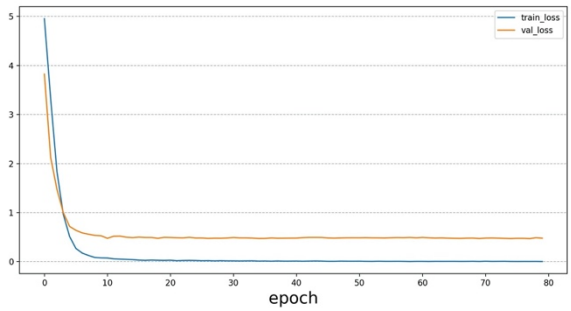
Fold 0



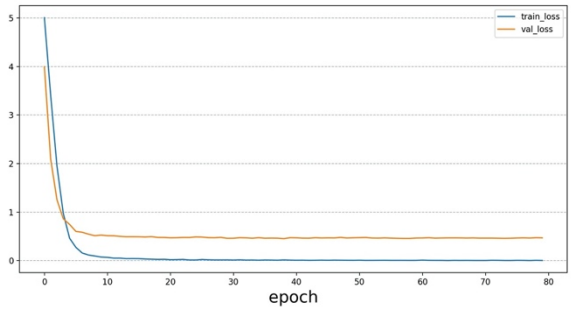
Fold 1



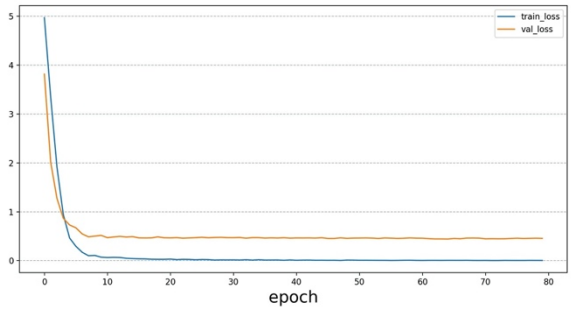
Fold 2



Fold 3

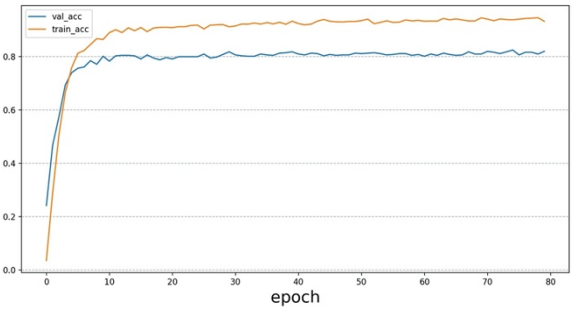


Fold 4

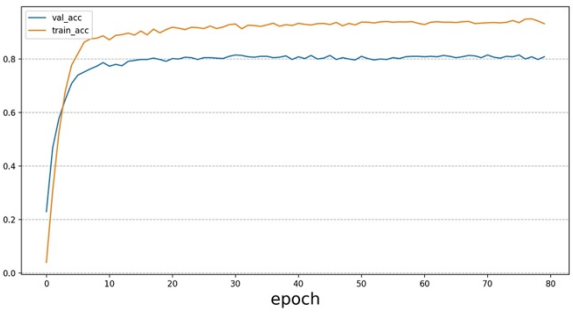


✓ Accuracy curve

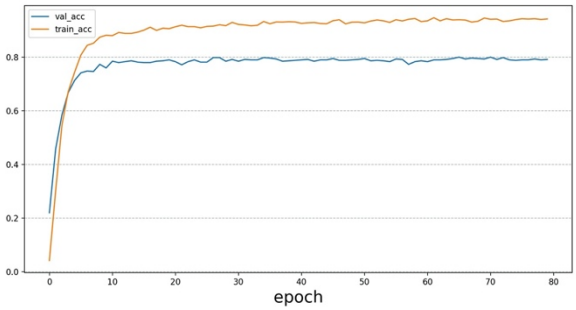
Fold 0



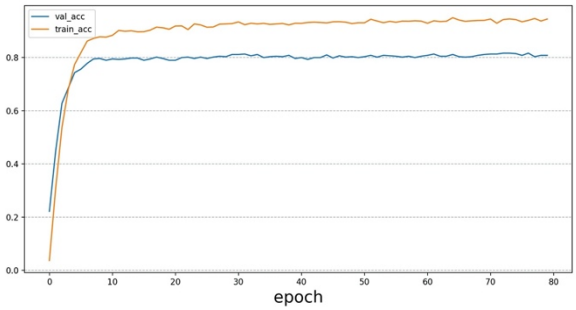
Fold 1



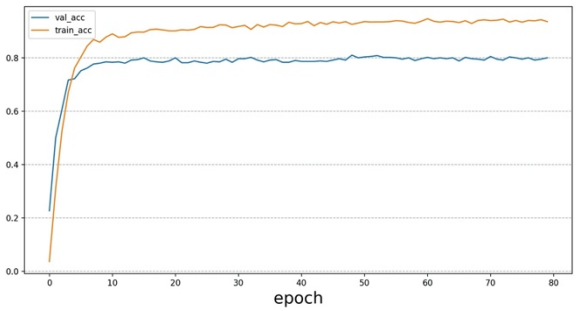
Fold 2


val_acc
train_acc
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
60
70
80
epoch

Fold 3


val_acc
train_acc
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
60
70
80
epoch

Fold 4


val_acc
train_acc
0.0
0.2
0.4
0.6
0.8
0
10
20
30
40
50
60
70
80
epoch