# Preprocessing

1. Load train data

Write two our own functions to load train file and test file. Then we try to use the function in Pytorch, Imagefolder. But finally we found it is not convenient to transform train set and test set separately. So we got back to our own function to load data.

1. Load class name with corresponding filaname

Apply the function supplied by teacher.

1. Transform data

We have tried flowlling method

* 1. Resize(224)
  2. Horizontalflip()
  3. Colorjitdier()
  4. Verticalflip()
  5. RandomRotation(60)
  6. Randomcrop()

But finally we found that only Resize(224) performed very good which can improve significantly. As for others, they cannot improve our performance even reduced the accuracy.

1. Split data to train set and validation set
   1. stractifiedShuffle, OR
   2. Random\_split

At the beginning, we apply the stractifiedshuffle in skitlearn. But wo finally apply the Random\_split in pytorch because it is more convenient to apply.

# Training model

## Model

### Torch.models (according to time)

1. GooGLeNet
2. ResNet18
3. ResNet50
4. MobileNet
5. ResNeXt
6. ResNet101
7. ResNet152
8. DenseNet
9. Wide\_ResNet50\_2
10. Wide\_ResNet101\_2

After preprocessing, we firstly choose to select the best model. We have tried all above models and we rank them as time going. Finally, we found that the Wide\_ResNet101\_2 performed best which can get the 85% for validation accuracy, so we choosed this model to train full train set.

## Optimizer

### type

1. SGD
2. Adam
3. Adamdelta

Although we apply SGD as optimizer by default, we still tried two other optimizers which performed very bad for performance.

### parameters

1. Learn rate
2. Momentum
3. Nesterov = True

After we fixed optimizer, we tried different learn rate and monmentum in SGD to train different models. And finally we found that the learn rate plays an important role for performance. We found that learn rate=0.005 is best for models based on Resnet (including Wide\_ResNet101\_2).

As for momentum and Nesterov, changing them even reduced our performance, so we abandoned them and fix momentum to 0.5 and fix Nesterov by default.

loss function = Cross entropy

batch\_size = 64

We have not tried to change loss function and batch\_size because we think the default is good enough and we do not have enough time to try for them.

Some points we learn from the project.

1. Never mix the train data and validation data. We once wrote wrong workflow which leads to mixture between train set and validation set which caused by the wrong application of Random\_split. We even get the best wrong validation accuracy (93%).
2. We learned how to apply the models in Pytorch and how to change the output classes.
3. Preprocessing is very important. At the beginning of the project, there is little wrong in preprocessing, so we get some bad results from every models. But luckily, we found it and advised it.

# Optimizer

Although we apply SGD as optimizer by default, we still tried two other optimizers which performed very bad for performance.

# Answer to first questions three：

We tried to apply grid search method at the beginning, but we find that the training time for a big neural network with a set of hyper-parameters is very time consuming. In general, we need to try three different kinds of hyper-parameters with three trials in each kind, so there will be 27 combinations. And we need to train 3 epochs to get the best result from each model. This means if we apply grid search method in one computer or Colab, it will take at least 40 hours to finish all combinations. When we run code in Colab, we find it will accidently beak off the execution. All in all, the grid search is difficult to apply for complex neural network.

Finally, we split the combinations of hyper-parameters to each group members to try. This will save much time and prevent the risk of breaking off.