NYCU Introduction to Machine Learning, Homework 4

[111550151], [徐嘉亨]

Part. 1, Kaggle (70% [50% comes from the competition]):

(10%) Implementation Details

Model architecture: I use the resnet-50 model pretrained on the ImageNet.

Data pre-processing:
I have tried two methods to do data pre-processing. First, I only use the transform in the figure below, then the best accuracy I reached in the public leaderboard is 59.137.

```
train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

test_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

However, I tried the second method which has more data augmentation. I make each training image to become 6 images, which are original, original with left rotate 15 degrees, original with right rotate 15 degrees, original with horizontal flip, flip with left rotate 15 degrees, flip with right rotate 15 degrees. And in the testing phase, I also do the same thing, and I averaged the predicted probability of 6 augmented testing images to be that test image's prediction. And it has better performance than method1. The best accuracy I reached in the public leaderboard is 61.691. But its training and testing time is almost 6 times of method 1.

```python
if aug_type == 0:  # Original image
    aug = transforms.Compose([transforms.RandomResizedCrop(224),
                              transforms.ToTensor(),
                              transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
elif aug_type == 1:  # Rotate left 15 degrees
    aug = transforms.Compose([transforms.RandomResizedCrop(224),
                              transforms.RandomRotation(degrees=(-15, -15)),
                              transforms.ToTensor(),
                              transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
elif aug_type == 2:  # Rotate right 15 degrees
    aug = transforms.Compose([transforms.RandomResizedCrop(224),
                              transforms.RandomRotation(degrees=(15, 15)),
                              transforms.ToTensor(),
                              transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
elif aug_type == 3:  # Horizontal Flip
    aug = transforms.Compose([transforms.RandomResizedCrop(224),
                              transforms.RandomHorizontalFlip(p=1),
                              transforms.ToTensor(),
                              transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
elif aug_type == 4:  # Horizontal Flip + Rotate left 15 degrees
    aug = transforms.Compose([transforms.RandomResizedCrop(224),
                              transforms.RandomHorizontalFlip(p=1),
                              transforms.RandomRotation(degrees=(-15, -15)),
                              transforms.ToTensor(),
                              transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
elif aug_type == 5:  # Horizontal Flip + Rotate right 15 degrees
    aug = transforms.Compose([transforms.RandomResizedCrop(224),
                              transforms.RandomHorizontalFlip(p=1),
                              transforms.RandomRotation(degrees=(15, 15)),
                              transforms.ToTensor(),
                              transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
```

```python
tta_transforms = [
    transforms.Compose([transforms.Resize(256),
                        transforms.CenterCrop(224),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]),
    transforms.Compose([transforms.Resize(256),
                        transforms.CenterCrop(224),
                        transforms.RandomRotation(degrees=(-15, -15)),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]),
    transforms.Compose([transforms.Resize(256),
                        transforms.CenterCrop(224),
                        transforms.RandomRotation(degrees=(15, 15)),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]),
    transforms.Compose([transforms.Resize(256),
                        transforms.CenterCrop(224),
                        transforms.RandomHorizontalFlip(p=1),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]),
    transforms.Compose([transforms.Resize(256),
                        transforms.CenterCrop(224),
                        transforms.RandomHorizontalFlip(p=1),
                        transforms.RandomRotation(degrees=(-15, -15)),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])]),
    transforms.Compose([transforms.Resize(256),
                        transforms.CenterCrop(224),
                        transforms.RandomHorizontalFlip(p=1),
                        transforms.RandomRotation(degrees=(15, 15)),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
]
```

Bagging: And I tried bagging on those two methods to compare the performance, I trained 41 models based on method 1. And I only trained 4 models based on method2(due to the constraint). I will leave the comparison to the next part.

About finetune: I set the new fully connected layer of the resnet-50 to be output 7 class and added the LogSoftmax(for training, faster convergence than normal Softmax. But in the testing, I modified it to be normal Softmax due to the probability output I needed.) to the output. For the learning rate of pretrained weights(without fully connected layers) is 1e-4, and the learning rate of new fully connected layers is 1e-3. And the total parameters and trainable parameters are 23522375.

```
model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1)

fc_features = model.fc.in_features
model.fc = nn.Sequential(nn.Linear(fc_features, 7),
                         nn.LogSoftmax(dim=1))
```

```
params = [
    {"params": model.fc.parameters(), "lr": 1e-3},
    {"params": [p for name, p in model.named_parameters() if not name.startswith("fc")], "lr": 1e-4},
]

optimizer = optim.Adam(params)
```

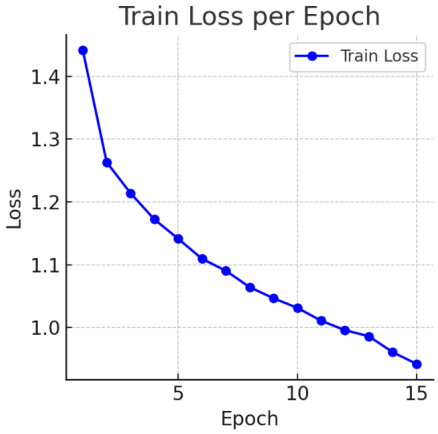
```
Total parameters: 23522375
Trainable parameters: 23522375
```

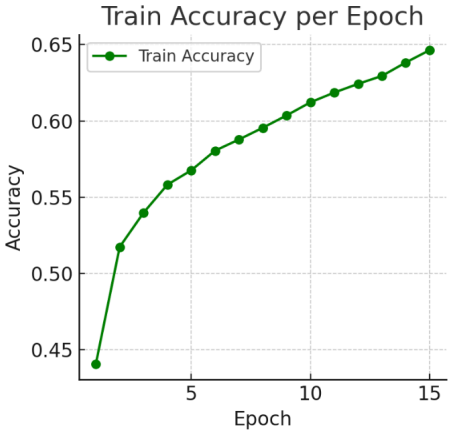
Model backbone (e.g., VGG16, VGG19, Custom, etc)	Resnet-50
Number of model parameters	23522375
Trainable parameters	23522375
Batch size	32
Learning rate for pretrained weights	1e-4
Learning rate for new fully connected layers	1e-3
Epoch	15
Loss function	CrossEntropyLoss

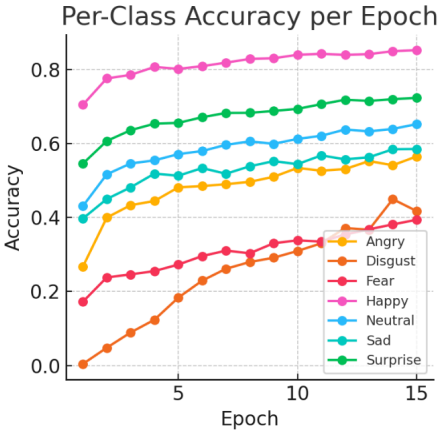
Optimizer	Adam

(10%) Experimental Results

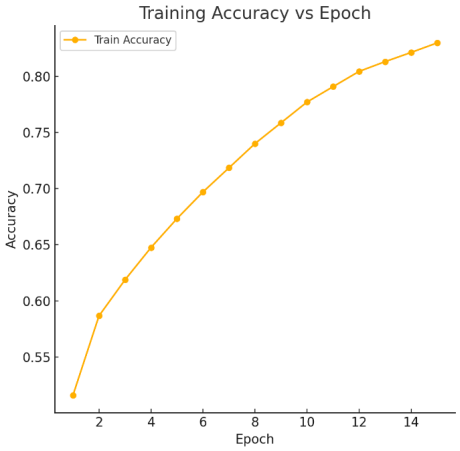
Some learning curves: I only show one classifier of bagging for each method

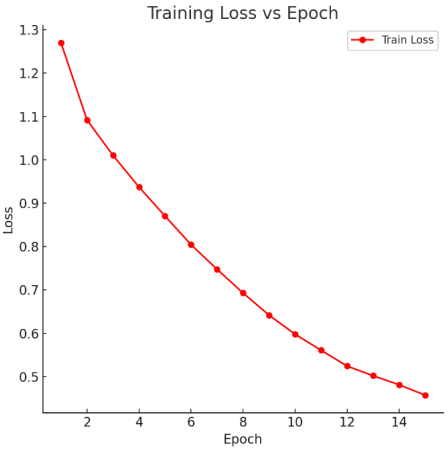


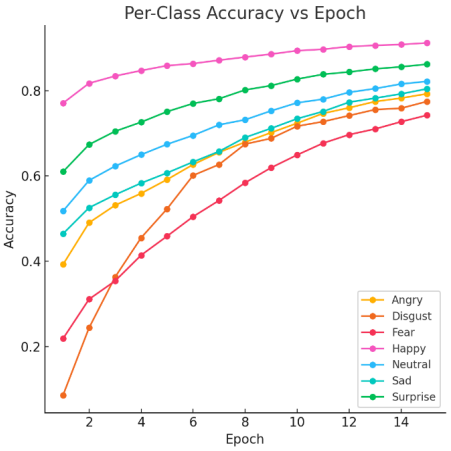




Method 1







Method 2

Evaluation metrics with ablation study:
For method1 with bagging, I found that using 26 classifiers can reach the highest accuracy. I think the reason is that the other 15 classifiers perform badly on the test images, so the dropout of them can improve the accuracy.
For method2 with bagging, due to the constraint, I only train 4 models and test it(**so the total parameters in the inference phase is 94089500, which is less than 100M**).
And I also include some ablation study on method1.
The classifiers in bagging I used are all trained on the same condition with the best classifier of each method.

Method	Public Accuracy
Method1 but all learning rate = 1e-4	55.788
Method1 with weighted CE loss	57.888
Method1 with epoch = 10	56.072
Method1 with epoch = 20	58.853
Method1 with pretrained weight lr = 3e-5	58.967
Method1 with histogram equalization	56.867
Method1 with label smoothing	58.683
Method1 without LogSoftmax	56.753
Best classifier of method1	**59.137**
Bagging for method1(11 classifiers)	59.364
Bagging for method1(21 classifiers)	59.704
Bagging for method1(31 classifiers)	60.385
Bagging for method1(41 classifiers)	60.669
Bagging for method1(26 classifiers)	60.896
Best classifier of method2	**61.691**
Bagging for method2(4 classifiers)	**63.450**

My kaggle best public submission(bagging for method2): 63.450(with **94089500** parameters < 100M)

Part. 2, Questions (30%):

1. (10%) Explain the support vector in SVM and the slack variable in Soft-margin SVM. Please provide a precise and concise answer. (each in two sentences)
   Support vectors are the data points which lie on the maximum margin hyperplane in feature space. They will be used to predict the label or value of the test point.
   Slack variables are introduced in Soft-margin SVM to allow some data points to violate the margin for better generalization on non-linearly separable data. Their values indicate the degree of misclassification.

2. (10%) In training an SVM, how do the parameter C and the hyperparameters of the kernel function (e.g., γ for the RBF kernel) affect the model's performance? Please explain their roles and describe how to choose these parameters to achieve good performance.
   For C, it controls the trade-off between achieving a low error on the training data and minimizing model complexity (or maximizing the margin). If C is high, it will penalize misclassified points heavily and encourage the model to fit the training data tightly, leading to a smaller margin. Risk: Overfitting, as the model may capture noise in the data. If C is low, it will allow some misclassification of training points and encourage a larger margin and simpler decision boundary. Risk: Underfitting, as the model may fail to capture important patterns.
   For γ, it controls the influence of a single training example. It determines the scale of the kernel function. If γ is high, the influence of each support vector is confined to a smaller region and leads to a more complex model that can capture fine-grained patterns. Risk: Overfitting, as the model may become too sensitive to small variations. If γ is low, the influence of support vectors extends over a larger region and leads to a smoother decision boundary. Risk: Underfitting, as the model may miss important details in the data. For choosing the proper value of C and γ, 1. Grid Search with Cross-Validation: Perform a grid search over a range of C and γ values and use k-fold cross-validation to evaluate the performance for each combination. Then choose the combination with the best validation performance. 2. Start with Heuristics: for γ, choose γ = 1 / (number of features). For C, choose C = 1 or similar mid-range value. 3. Consider Dataset Characteristics: If the data is noisy, use a smaller C to allow some misclassification. If the data has complex patterns, consider a higher γ to allow more flexibility, but ensure γ isn't too high.

3. (10%) SVM is often more accurate than Logistic Regression. Please compare SVM and Logistic Regression in handling outliers.

SVM: SVM constructs a decision boundary by maximizing the margin between the classes and focusing on support vectors (the data points closest to the boundary). So it is less affected by outliers because they are not near the margin. Outliers far from the decision boundary do not influence the placement of the boundary. By adjusting the cost parameter C, SVM can control the trade-off between maximizing the margin and minimizing classification errors. A small C(soft margin) increases the model's tolerance to outliers by allowing some misclassifications.
Logistic Regression: Logistic Regression finds a linear decision boundary that maximizes the likelihood of the data given the model. Outliers can heavily influence the model because Logistic Regression uses all data points to compute the cost function. A single extreme outlier may disproportionately affect the gradient, leading to a suboptimal boundary. But Logistic Regression utilizes the logistic sigmoid function which will transfer the input value to a real number in [0, 1]. That means it can reduce the outliers' effect by reducing the difference between normal points and the outliers. But this isn't enough, we can consider some regularization (e.g., L1 or L2) or Huber loss and other robust loss functions to reduce the impact of outliers.