Machine Learning Report

Xinran TANG 20126518

Task 1

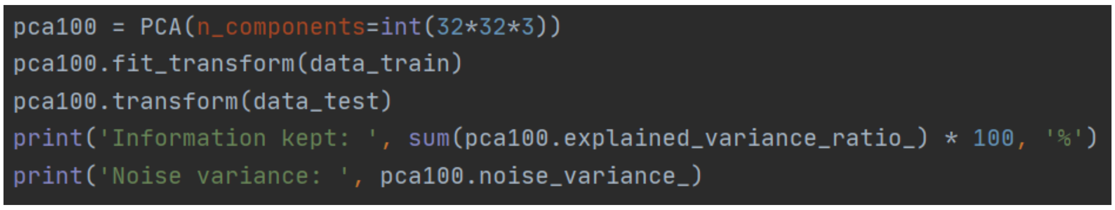
Apply PCA to reduce the original input features into new feature vectors with different amount of information kept, e.g. 10%, 30%, 50%, 70%, 100%.

For ***"PCA(n_components = numberA)"***, ***numberA*** refers to the features of the input image. ***n_components***: The number of features retained, so all information is kept when the total number of features = 32 * 32 * 3. The degree of information kept is directly proportional to the value of ***n_components***. When ***n_components*** is larger, the percentage of information kept is higher. Therefore, decrease the value of ***n_components*** can decrease the percentage of information kept.

- **information kept 100%**

  Information kept: 100.00000000000011 %

  Noise variance: 0.0

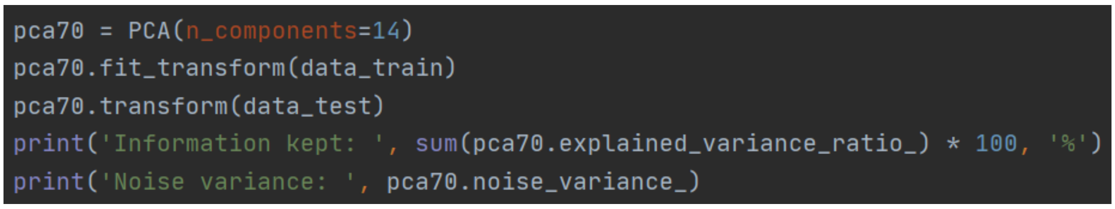


```
pca100 = PCA(n_components=int(32*32*3))
pca100.fit_transform(data_train)
pca100.transform(data_test)
print('Information kept: ', sum(pca100.explained_variance_ratio_) * 100, '%')
print('Noise variance: ', pca100.noise_variance_)
```

- **information kept 70%**

  Information kept: 69.93227189264823 %

  Noise variance: 0.01872152451551815

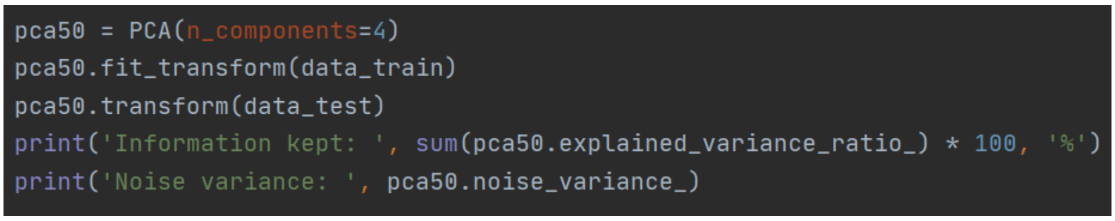


```
pca70 = PCA(n_components=14)
pca70.fit_transform(data_train)
pca70.transform(data_test)
print('Information kept: ', sum(pca70.explained_variance_ratio_) * 100, '%')
print('Noise variance: ', pca70.noise_variance_)
```

- **information kept 50%**

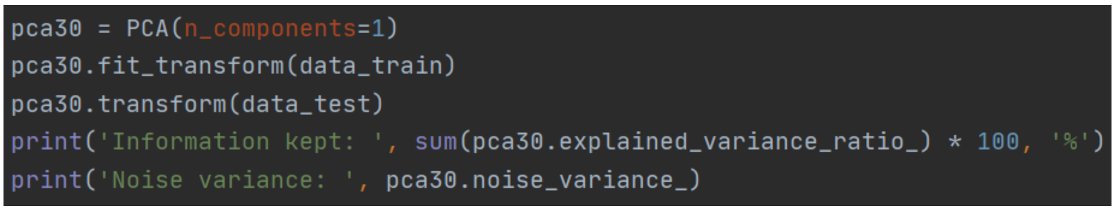
  Information kept: 50.700646387370405 %

  Noise variance: 0.030595950344108568



```
pca50 = PCA(n_components=4)
pca50.fit_transform(data_train)
pca50.transform(data_test)
print('Information kept: ', sum(pca50.explained_variance_ratio_) * 100, '%')
print('Noise variance: ', pca50.noise_variance_)
```

- **information kept 30%**
  Information kept: 29.076629890565098 %
  Noise variance: 0.043973154857423 64



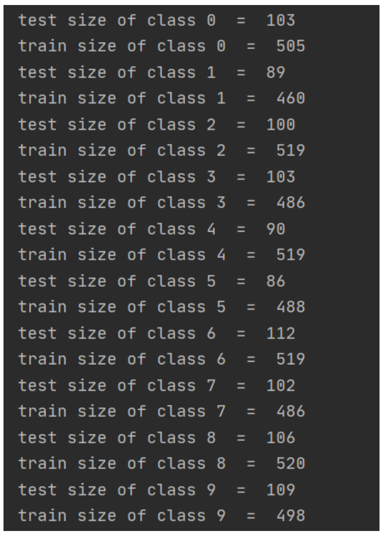
```
pca30 = PCA(n_components=1)
pca30.fit_transform(data_train)
pca30.transform(data_test)
print('Information kept: ', sum(pca30.explained_variance_ratio_) * 100, '%')
print('Noise variance: ', pca30.noise_variance_)
```

When the feature number is minimum (=1), the information kept is 29.076629890565098%, so the minimum information kept is 29.076629890565098%. When the number of features is 0, the percentage of information kept is 0.

Apart from that, if using ***"PCA(0.5)"*** , the information kept will around 50%.

Task 2

The first 5000 pictures of total training set were set as the training set because the time taken by training 50000 images is too long. For the test set, similarly, the first 1000 pictures of 10,000 pictures were used for testing. Through running, it can be concluded that the number of each class is basically the same in the first 5000 pictures of training set and the first 1000 pictures of test set. Therefore, using the first 5000 and the first 1000 directly can meet the requirements. The Following image shows the number of images with different class in training set and test set:



```
test size of class 0  =  103
train size of class 0  =  505
test size of class 1  =  89
train size of class 1  =  460
test size of class 2  =  100
train size of class 2  =  519
test size of class 3  =  103
train size of class 3  =  486
test size of class 4  =  90
train size of class 4  =  519
test size of class 5  =  86
train size of class 5  =  488
test size of class 6  =  112
train size of class 6  =  519
test size of class 7  =  102
train size of class 7  =  486
test size of class 8  =  106
train size of class 8  =  520
test size of class 9  =  109
train size of class 9  =  498
```

Task 2.1

Task 2.1 aims at using 10-fold validation for images with different information kept records, training 4500 images in the training set at a time and performing validation for the remaining 500 images.

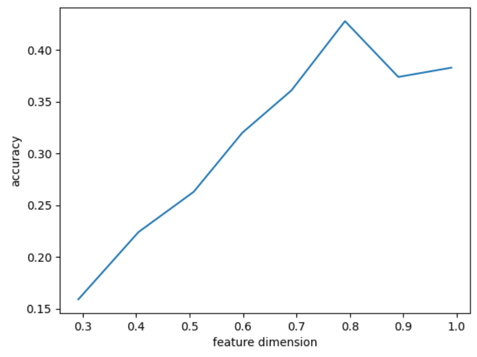
The default ***"MLPClassifier()"*** and ***"cross_val_score()"*** functions in sklearn were used to train the first 5000 images in the training set and predict the label of the first 1000 images in the test set. 10-fold validation is performed for each of the different information kept values. The values of the 10 validations were stored into a list and the mean values were stored into another list. The results are shown below:

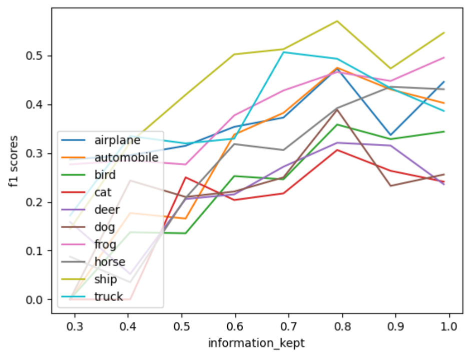
```
k =  29
information kept:  0.29194361665830787  score =  [0.17  0.164 0.176 0.188 0.18  0.174 0.168 0.182 0.172 0.172]
information kept:  0.29194361665830787  mean score =  0.17459999999999998
k =  39
information kept:  0.4043950481018702  score =  [0.2   0.248 0.192 0.238 0.23  0.246 0.202 0.216 0.212 0.194]
information kept:  0.4043950481018702  mean score =  0.2178
k =  49
information kept:  0.5078907532446614  score =  [0.266 0.296 0.298 0.276 0.296 0.29  0.268 0.246 0.244 0.284]
information kept:  0.5078907532446614  mean score =  0.27640000000000003
k =  59
information kept:  0.59863908423181  score =  [0.352 0.338 0.356 0.362 0.32  0.37  0.334 0.34  0.29  0.34 ]
information kept:  0.59863908423181  mean score =  0.34019999999999995
k =  69
information kept:  0.6906810505336544  score =  [0.352 0.366 0.37  0.37  0.398 0.398 0.392 0.386 0.354 0.346]
information kept:  0.6906810505336544  mean score =  0.37320000000000003
k =  79
information kept:  0.7909319893252035  score =  [0.42  0.398 0.402 0.422 0.402 0.394 0.44  0.432 0.4   0.426]
information kept:  0.7909319893252035  mean score =  0.4136
k =  89
information kept:  0.8907215485844828  score =  [0.39  0.368 0.392 0.416 0.392 0.41  0.354 0.374 0.344 0.416]
information kept:  0.8907215485844828  mean score =  0.3856
k =  99
information kept:  0.9900147615535209  score =  [0.364 0.376 0.328 0.368 0.346 0.37  0.346 0.346 0.274 0.374]
information kept:  0.9900147615535209  mean score =  0.3492
```

Task 2.2

Task 2.2 aims at training sklearn's default ***“MLPClassifier()”*** model with 5000 images of the training set, predict the first 1000 images of the test set, and draw the f1 curve for each class in the test set and the accuracy curve for the entire test set.

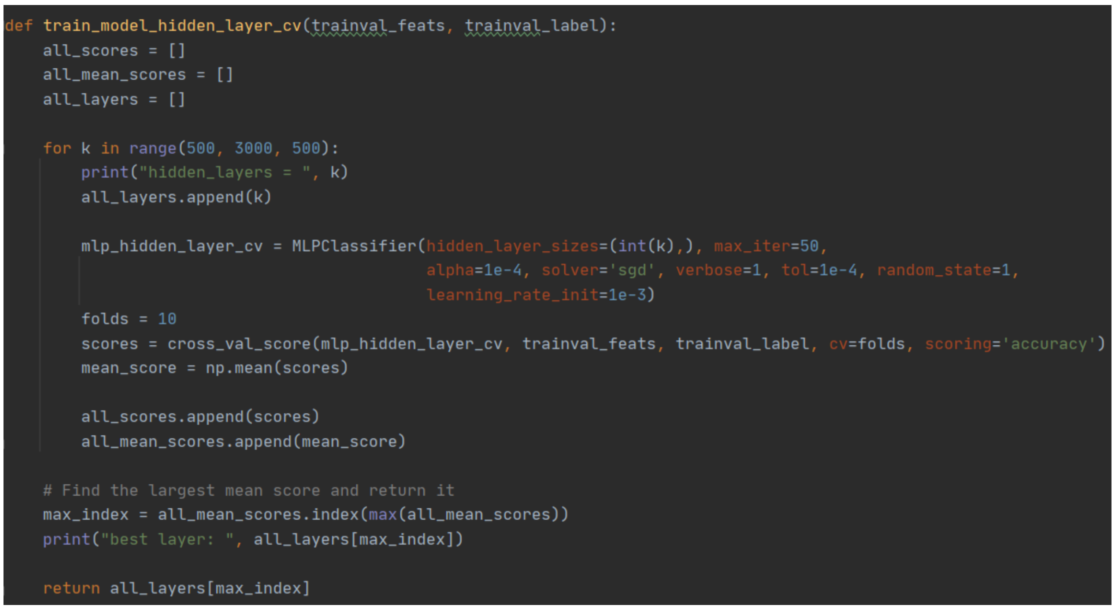
Use ***“f1_score (te_label, label_pred, average=None)”*** to calculate the f1 of each class. Use “***mlp.score(data_test_input,Labels_test_input)”*** to calculate the accuracy of the entire test set. The results are shown in the figures below:



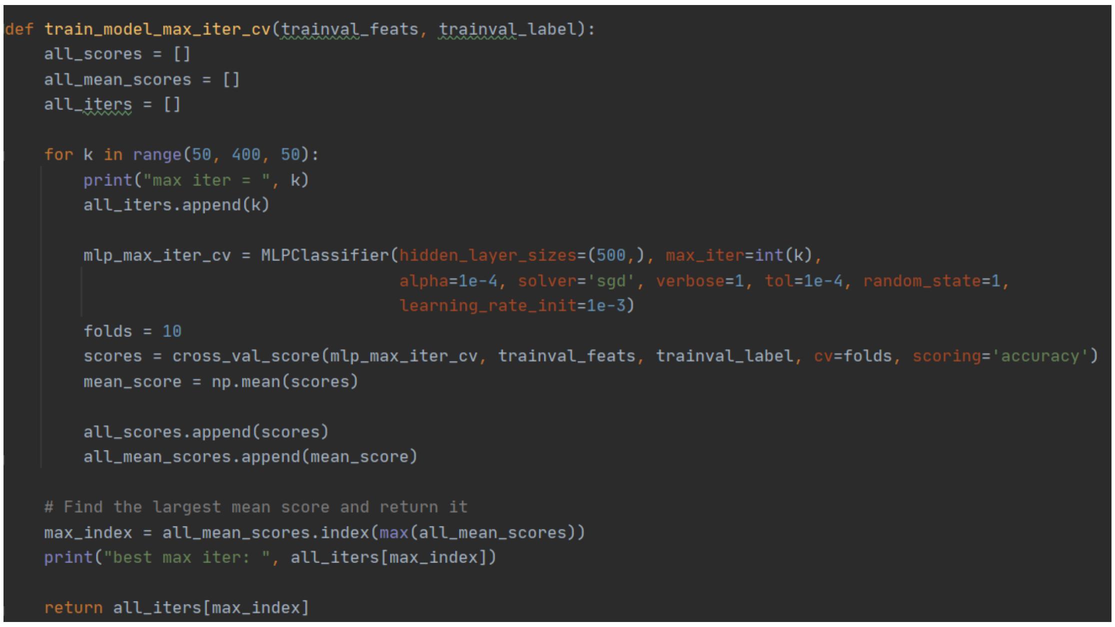


Task 2.3

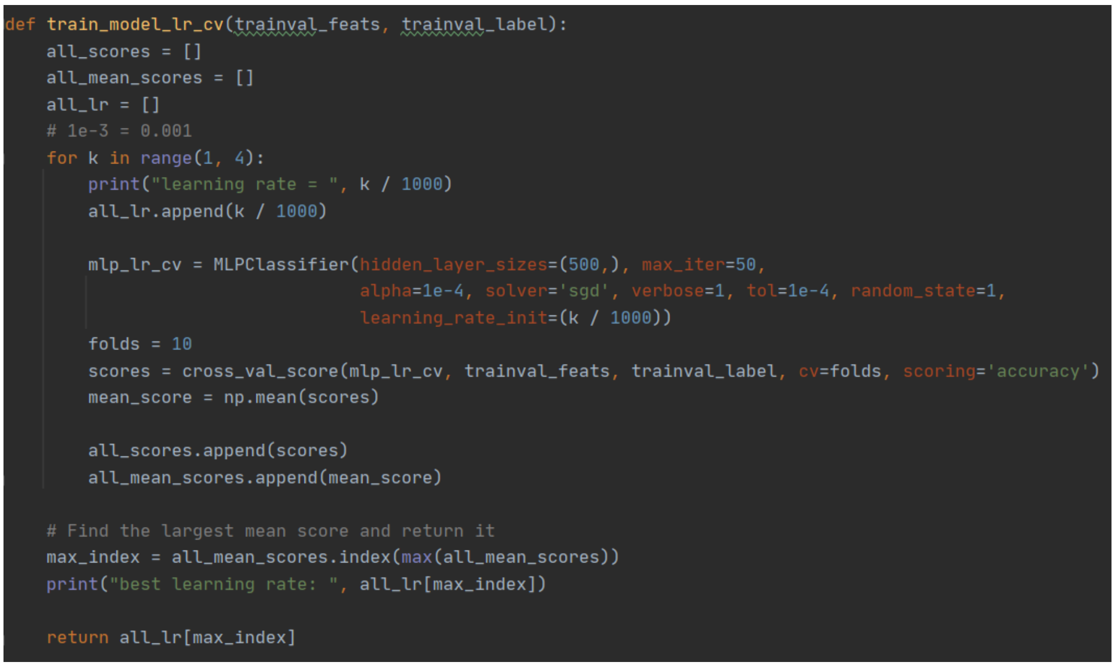
Task 2.3 aims at adjusting the hyperparameters of MLP in sklearn by 10-folds using 5000 images in the training set. Perform validation for the remaining 500 images from the training set. hidden layer size, max iteration number and learning rate are hyperparameters adjusted in task 2.3. The idea is: first set a set of initial values, and then adjust the range of one parameter each time through the method of controlling variables to find the value of this parameter when the model accuracy is highest in the case of the other two parameters unchanged. The initial values for these three hyperparameters are: hidden layer size = 500, max iter = 50, learning rate = 0.001. The change range of hidden layer size is 500, 1000, 1500, 2000, 2500; The change range of the max iteration number is 50, 100, 150, 200, 250, 300, 350; The change range of the learning rate is 0.001, 0.002, 0.003. Print out and return the parameter with the max score. The codes used are shown below:



```python
def train_model_hidden_layer_cv(trainval_feats, trainval_label):
    all_scores = []
    all_mean_scores = []
    all_layers = []

    for k in range(500, 3000, 500):
        print("hidden_layers = ", k)
        all_layers.append(k)

        mlp_hidden_layer_cv = MLPClassifier(hidden_layer_sizes=(int(k),), max_iter=50,
                                            alpha=1e-4, solver='sgd', verbose=1, tol=1e-4, random_state=1,
                                            learning_rate_init=1e-3)
        folds = 10
        scores = cross_val_score(mlp_hidden_layer_cv, trainval_feats, trainval_label, cv=folds, scoring='accuracy')
        mean_score = np.mean(scores)

        all_scores.append(scores)
        all_mean_scores.append(mean_score)

    # Find the largest mean score and return it
    max_index = all_mean_scores.index(max(all_mean_scores))
    print("best layer: ", all_layers[max_index])

    return all_layers[max_index]
```



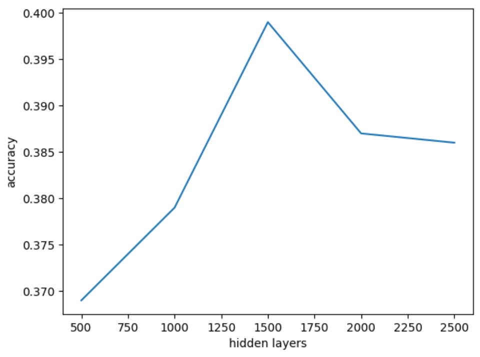
```python
def train_model_max_iter_cv(trainval_feats, trainval_label):
    all_scores = []
    all_mean_scores = []
    all_iters = []

    for k in range(50, 400, 50):
        print("max iter = ", k)
        all_iters.append(k)

        mlp_max_iter_cv = MLPClassifier(hidden_layer_sizes=(500,), max_iter=int(k),
                                        alpha=1e-4, solver='sgd', verbose=1, tol=1e-4, random_state=1,
                                        learning_rate_init=1e-3)
        folds = 10
        scores = cross_val_score(mlp_max_iter_cv, trainval_feats, trainval_label, cv=folds, scoring='accuracy')
        mean_score = np.mean(scores)

        all_scores.append(scores)
        all_mean_scores.append(mean_score)

    # Find the largest mean score and return it
    max_index = all_mean_scores.index(max(all_mean_scores))
    print("best max iter: ", all_iters[max_index])

    return all_iters[max_index]
```

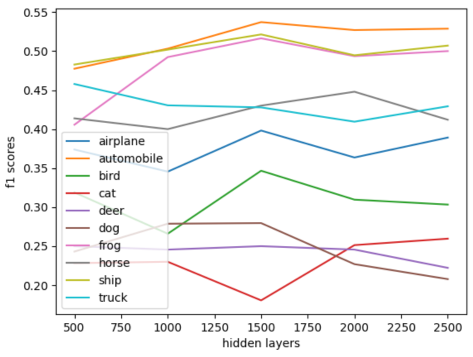


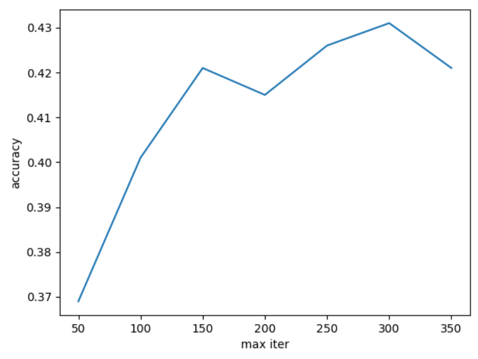
```python
def train_model_lr_cv(trainval_feats, trainval_label):
    all_scores = []
    all_mean_scores = []
    all_lr = []
    # 1e-3 = 0.001
    for k in range(1, 4):
        print("learning rate = ", k / 1000)
        all_lr.append(k / 1000)

        mlp_lr_cv = MLPClassifier(hidden_layer_sizes=(500,), max_iter=50,
                                  alpha=1e-4, solver='sgd', verbose=1, tol=1e-4, random_state=1,
                                  learning_rate_init=(k / 1000))
        folds = 10
        scores = cross_val_score(mlp_lr_cv, trainval_feats, trainval_label, cv=folds, scoring='accuracy')
        mean_score = np.mean(scores)

        all_scores.append(scores)
        all_mean_scores.append(mean_score)

    # Find the largest mean score and return it
    max_index = all_mean_scores.index(max(all_mean_scores))
    print("best learning rate: ", all_lr[max_index])

    return all_lr[max_index]
```

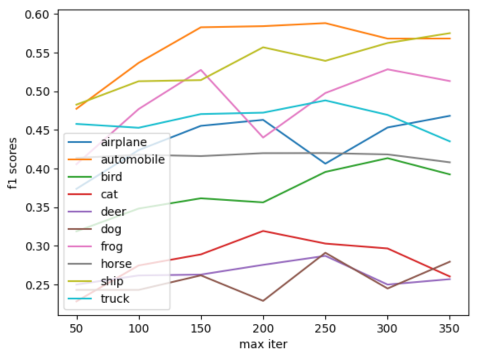
Task 2.4

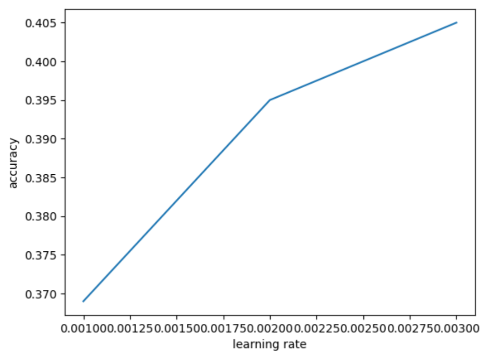
Task 2.4 aims at training the MLP model with 5000 images of the training set, make predictions for the first 1000 images of the test set, and draw the f1 curves of each class in the test set and the accuracy curves of the entire test set for different parameters. The initial values and change range of the hyperparameters used Task 2.4 are same with Task 2.3. The results are shown below:

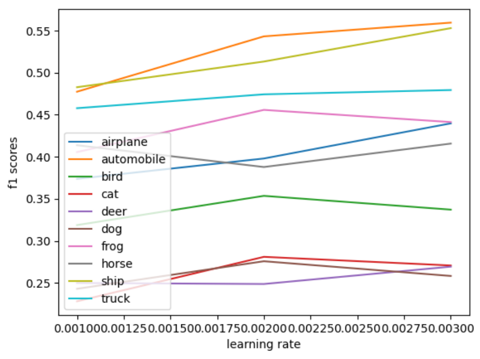




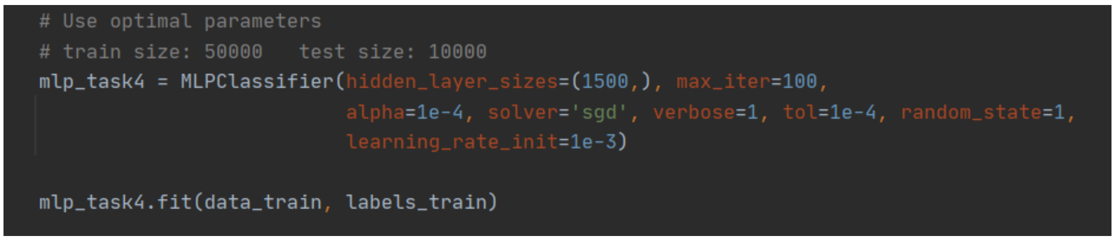




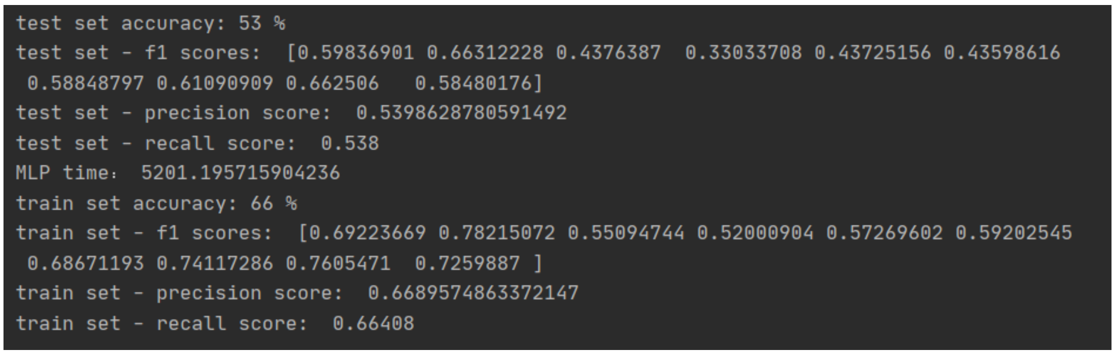




The result trends for Task 2.4 are similar to those for Task 2.3, and the range of optimal hyperparameters derived from the work of 2.3 and 2.4 will be used in Task 4. The optimal MLP got from previous results is shown below:



```
   # Use optimal parameters
   # train size: 50000    test size: 10000
   mlp_task4 = MLPClassifier(hidden_layer_sizes=(1500,), max_iter=100,
                             alpha=1e-4, solver='sgd', verbose=1, tol=1e-4, random_state=1,
                             learning_rate_init=1e-3)

   mlp_task4.fit(data_train, labels_train)
```



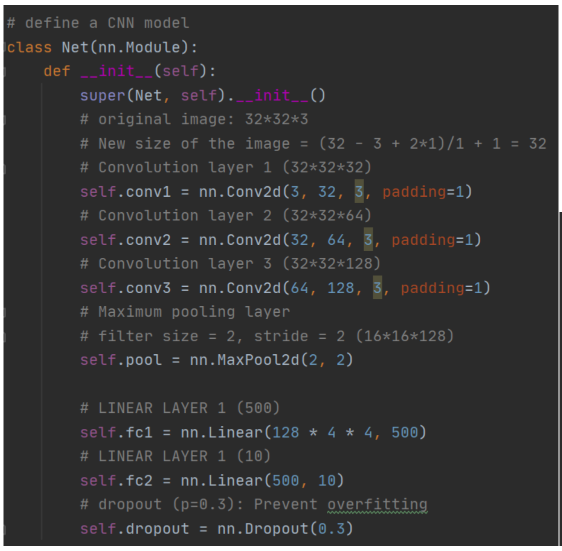
```
test set accuracy: 53 %
test set - f1 scores:  [0.59836901 0.66312228 0.4376387  0.33033708 0.43725156 0.43598616
 0.58848797 0.61090909 0.662506   0.58480176]
test set - precision score:  0.5398628780591492
test set - recall score:  0.538
MLP time:  5201.195715904236
train set accuracy: 66 %
train set - f1 scores:  [0.69223669 0.78215072 0.55094744 0.52000904 0.57269602 0.59202545
 0.68671193 0.74117286 0.7605471  0.7259887 ]
train set - precision score:  0.6689574863372147
train set - recall score:  0.66408
```

Task 3

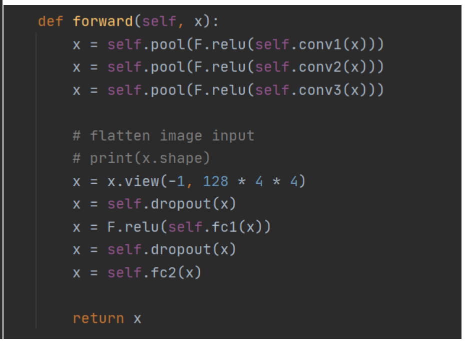
- **CNN design**

  When solving a more complex problem, either increase depth or increase width is needed, and the cost of increasing width is often much higher than that of depth. Therefore, in Task 3, I increased the depth of the network to improve the performance of designed CNN.

  By reading the paper "*Very Deep convolutional Networks for Large-scale Image Recognition*", learned that the VGG network adopts several consecutive 3x3 kernels to replace the larger one in AlexNet. Cascade two 3x3 kernels can replace a 5x5 kernels, which can increase the depth of the model by using small filters. The core idea of VGG network is to use 3x3 kernel and 2x2 Max pooling to realize a CNN. Therefore, the idea of VGG network is used for guiding the design of the CNN in Task 3: use multiple 3*3 kernels and 2x2 Max poolings to increase the depth of the network and finally improve the performance. The CNN used in Task 3 is shown below:



```python
# define a CNN model
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # original image: 32*32*3
        # New size of the image = (32 - 3 + 2*1)/1 + 1 = 32
        # Convolution layer 1 (32*32*32)
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        # Convolution layer 2 (32*32*64)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        # Convolution layer 3 (32*32*128)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        # Maximum pooling layer
        # filter size = 2, stride = 2 (16*16*128)
        self.pool = nn.MaxPool2d(2, 2)

        # LINEAR LAYER 1 (500)
        self.fc1 = nn.Linear(128 * 4 * 4, 500)
        # LINEAR LAYER 1 (10)
        self.fc2 = nn.Linear(500, 10)
        # dropout (p=0.3): Prevent overfitting
        self.dropout = nn.Dropout(0.3)
```



```python
def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = self.pool(F.relu(self.conv3(x)))

    # flatten image input
    # print(x.shape)
    x = x.view(-1, 128 * 4 * 4)
    x = self.dropout(x)
    x = F.relu(self.fc1(x))
    x = self.dropout(x)
    x = self.fc2(x)

    return x
```

- **data set deign**

  The data set used in task 3 are **training set**, **validation set** and **test set**. Training loss and Validation loss were calculated in 20 epochs by splitting the 50000 images training set into train set and validate set. Training Loss refers to the fitting ability of the model on the training set, while Validation Loss refers to the fitting ability (generalization ability) of the model on the unknown data. The true effectiveness of the model is measured by Validation Loss. The stored model "model_50000.pt" is updated to a better model when validation Loss is smaller than the smallest value stored previously. We can also judge the fit degree of the model by comparing these two loss values. The stored model will be loaded before testing.

- **batch size design**

  In the stochastic gradient descent SGD algorithm, a Batch of samples are randomly selected from the training data each time, and the number of samples is Batch Size. Many experiments have proved that under the condition of constant learning rate, the larger the Batch Size is, the worse the model convergence effect will be. Therefore, when learning rate = 0.01, the effect is the best when batch size = 16 after several tests.

- **epoch design**

  The epoch is set to 50 at the beginning. After many tests, it shows that when epoch is larger than 30, the network is overfitting to a certain extent. Therefore, epoch = 20 is used in Task 3.

- **training speed**

  Use GPU to accelerate the network training (cuda)

- **result**

```
CUDA IS AVAILABLE!
Files already downloaded and verified
Files already downloaded and verified
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=2048, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=10, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
)
Epoch:1   Training loss:1.9757603382110596       Validation loss:1.6378971216201783
Validation loss decreased (inf --> 1.6378971216201783). Saving model ...
Epoch:2   Training loss:1.5312318350076675       Validation loss:1.3984691304206849
Validation loss decreased (1.6378971216201783 --> 1.3984691304206849). Saving model ...
Epoch:3   Training loss:1.3558801407337189       Validation loss:1.2427470332145691
Validation loss decreased (1.3984691304206849 --> 1.2427470332145691). Saving model ...
Epoch:4   Training loss:1.2253588579893113       Validation loss:1.112976249551773
Validation loss decreased (1.2427470332145691 --> 1.112976249551773). Saving model ...
Epoch:5   Training loss:1.1108729843854903       Validation loss:1.0500603909015656
Validation loss decreased (1.112976249551773 --> 1.0500603909015656). Saving model ...
Epoch:6   Training loss:1.0183199293255807       Validation loss:0.9620086884975433
Validation loss decreased (1.0500603909015656 --> 0.9620086884975433). Saving model ...
Epoch:7   Training loss:0.9397999723792076       Validation loss:0.8918657217502594
Validation loss decreased (0.9620086884975433 --> 0.8918657217502594). Saving model ...
Epoch:8   Training loss:0.8720764249563218       Validation loss:0.833623397397995
Validation loss decreased (0.8918657217502594 --> 0.833623397397995). Saving model ...
Epoch:9   Training loss:0.8099288375735283       Validation loss:0.810740816116333
Epoch:10 Training loss:0.7565706873834133       Validation loss:0.7794363332986831
Validation loss decreased (0.810740816116333 --> 0.7794363332986831). Saving model ...
Epoch:11 Training loss:0.7152093853741884       Validation loss:0.7667701857566833
Validation loss decreased (0.7794363332986831 --> 0.7667701857566833). Saving model ...
Epoch:12 Training loss:0.6716862097740174       Validation loss:0.75675211353302
Validation loss decreased (0.7667701857566833 --> 0.75675211353302). Saving model ...
Epoch:13 Training loss:0.632555461037159        Validation loss:0.7005856438636779
Validation loss decreased (0.75675211353302 --> 0.7005856438636779). Saving model ...
Epoch:14 Training loss:0.5963397412419319       Validation loss:0.7098636044979095
Epoch:15 Training loss:0.5603394621193409       Validation loss:0.6840003116607666
Validation loss decreased (0.7005856438636779 --> 0.6840003116607666). Saving model ...
Epoch:16 Training loss:0.5245122542828321       Validation loss:0.7078416915655136
Epoch:17 Training loss:0.4968831776678562       Validation loss:0.6736108600616455
Validation loss decreased (0.6840003116607666 --> 0.6736108600616455). Saving model ...
Epoch:18 Training loss:0.46634120652526617      Validation loss:0.6660065638780593
Validation loss decreased (0.6736108600616455 --> 0.6660065638780593). Saving model ...
Epoch:19 Training loss:0.4316590280711651       Validation loss:0.655623676609993
Validation loss decreased (0.6660065638780593 --> 0.655623676609993). Saving model ...
Epoch:20 Training loss:0.40588696829900145      Validation loss:0.6711872287511825
test set accuracy: 89 %
test set - f1 scores:  [0.92344689 0.9593234  0.84799383 0.80887372 0.85947349 0.8463395 0.92252682 0.92144374 0.95759017 0.94816587]
test set - precision score:  0.9025841975988345
test set - recall score:  0.8997890747181974
CNN time:   2200.7563548088074
train set accuracy: 92 %
train set - f1 scores:  [0.95105566 0.98079764 0.89603214 0.86168521 0.90083868 0.89380531 0.94087923 0.95286885 0.97367114 0.97502498]
train set - precision score:  0.9342982232476679
train set - recall score:  0.9333183811648643
```

Task 4

- Train MLP and CNN with 50000 images and test with 10000 images, the performances comparison is shown below:
  - **accuracy, precision score, recall score – test set**

	MLP	**CNN**
accuracy	53%	89%
precision score	0.54	0.90
recall score	0.54	0.90

  - **accuracy, precision score, recall score – train set**

	MLP	**CNN**
accuracy	66%	92%
precision score	0.67	0.93
recall score	0.66	0.93

  - **f1 scores – test set**

	MLP	**CNN**
airplane	0.60	0.92
automobile	0.66	0.96
bird	0.44	0.85
cat	0.33	0.81
deer	0.44	0.86
dog	0.44	0.85
frog	0.59	0.92
horse	0.61	0.92
ship	0.66	0.96
truck	0.58	0.95

  - **f1 scores – train set**

	MLP	**CNN**
airplane	0.69	0.95
automobile	0.78	0.98
bird	0.55	0.90
cat	0.52	0.86
deer	0.57	0.90
dog	0.59	0.89
frog	0.69	0.94
horse	0.74	0.95
ship	0.76	0.97
truck	0.73	0.98

- **Time – training + testing**

MLP	**CNN**
5201.195715904236	2200.7563548088074

- **level of overfitting**

  The previous results show that the overfitting level of MLP is bigger than the CNN.

- **better approach**

  In my opinion, I think CNN is a better approach. Firstly, the training time needed for MLP is much longer than the CNN if they got the similar performance. Secondly, MLP uses fully connected layer and CNN uses locally connected layer. CNN can solve the lost pixels or 2D spatial information between pixels in the input of MLP vector, which help CNN get better accuracy easier than MLP.