《神经网络与深度学习》课程实验作业（一）

实验内容：深度学习基础

何尉宁 2021213599

要求：

机器学习：回归问题

> 皮马印第安人糖尿病数据集由年龄大于等于21 岁的皮马印第安女性的已有诊断信息组 成，包含若干医学预测变量和一个目标变量 Outcome，共九个字段。其中预测变量包括患者 的怀孕次数、BMI、胰岛素水平、年龄等。
> 请运用回归模型分析糖尿病数据集中自变量和因变量之间的关系，对某人是否患糖尿病 进行预测。

- 使用特征工程方法对数据集中的特征变量进行处理并在实验报告中作出说明
- 使用逻辑回归模型完成实验并绘制逻辑回归散点图 (Novelty: K-Fold & MICE)
- 使用train_test_split 函数对数据集进行拆分，并在拆分得到的测试集上测试模型准确率。我设置的`ratio = 0.8`
- 试分析多个特征值（自变量）与病情（因变量）的关系，依据与病情的关联性对自变量进行排序并进行可视化展示

多层感知机：分类问题

> CIFAR-10 数据集是深度学习中常用的数据集，其包含 60000 张 32×32 色图像，分为 10 个类，每类6000 张。有50000 张训练图片和 10000 张测试图片。
> 请基于该数据集搭建包含三个及以上的全连接层的多层感知机网络，以解决10 分类问 题。

- 输出网络结构
- 使用tensorboard对训练过程中的loss和accuracy 进行可视化展示
- 保存训练模型为模型文件，并使用训练好的模型对测试集中的图片进行预测，输出预测结果与预测概率
- 画出训练集和验证集的混淆矩阵
- 分析网络参数（例如网络深度、不同的激活函数、神经元数量等）对预测结果的影响
- 在损失函数为交叉熵的情况下，对比网络最后一层是否使用softmax 的性能差异并分析其产生的原因

---

1. 皮马印第安人糖尿病 (AvgAcc = 84.2%/K-Fold Best 93%)

1.1 特征工程

1.1.1 数据读取与观察

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

diabetes_data = pd.read_csv('../Assignment1_dataset/diabetes.csv')
diabetes_data.head()
# diabetes.shape
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

观察分析：

我们可以发现label存储在Outcome中，同时数据集的特征有“怀孕”，“葡萄糖”，“血压”，“皮肤厚度”，“胰岛素”，“BMI”，“糖尿病谱系功能”，“年龄”共8个方面。
通过CSV可以发现缺省值均为0，所以我们将其替换为NaN后再进行处理。

1.1.2 数据清洗与填充

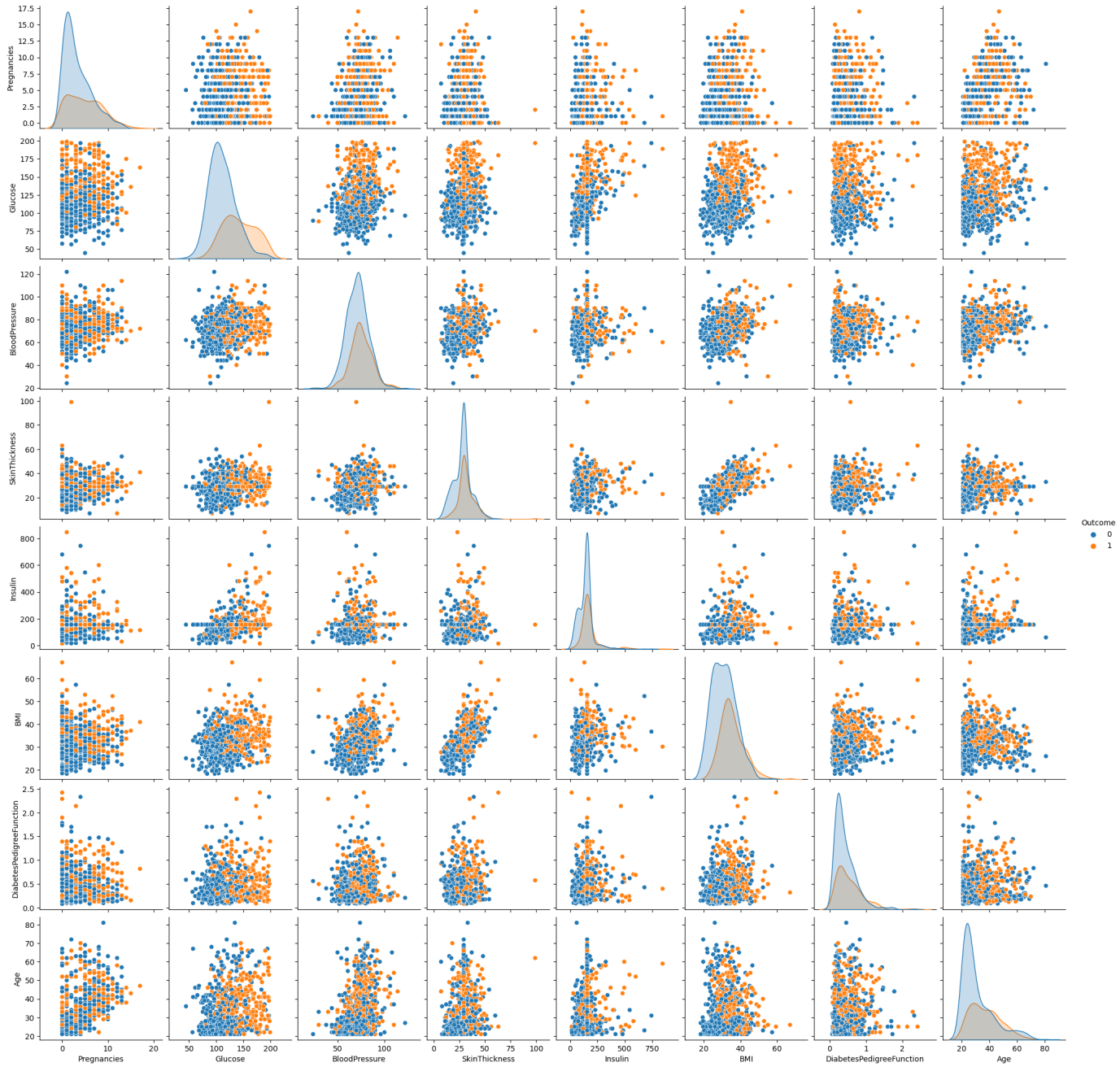
1. 填充：
   (1). 直接使用median/mean/mode填充缺值
   (2). 使用Outcome作为标签，映射至缺值中
2. 纠错：
   ~~有个800多胰岛素的姐们引起了我的注意，我本来打算去查一下表，看能不能规范一下异常数据的，结果发现看不懂。~~
   尝试删除离群值之后发现效果并不理想，因为有些极端的数据与患病概率是正相关的，所以删除后会减弱相关性。这个阈值的把控非常困难，很容易删除有用的数据。
   例如：
   那个800多胰岛素的姐们，一看就不正常
   那个60多BMI的姐们，一看就不正常

```
# 用平均值填充NaN
diabetes_data.fillna(diabetes_data.mean(), inplace = True)
diabetes_data.isnull().any()
```

1.1.3 可视化清洗后的数据

1. 以Outcome为标准，绘制热力图

```
import seaborn as sns

diabetes_data_raw = pd.read_csv('../Assignment1_dataset/diabetes.csv')
diabetes_data.to_csv('../Assignment1_dataset/diabetes_clean.csv', index=False)
diabetes_data_clean = pd.read_csv('../Assignment1_dataset/diabetes_clean.csv')
# 将outcome加回来
diabetes_data_clean['Outcome'] = diabetes_data_raw['Outcome']
# 用seaborn绘制散点图
sns.pairplot(diabetes_data_clean, hue = 'Outcome', diag_kind = 'kde')
```



1.1.4 特征查看与选取

该部分详细代码将在问题4中展示

特征之间的相关性：

很容易可以发现，这些症状都与是否患糖尿病正相关，其中否患病相关性最高的三项是葡萄糖，BMI和年龄。

同时，通过以葡萄糖为关键词，我们发现与之正相关性强的依旧是胰岛素，年龄和BMI

1.1.5 数据标准化

对已处理的数据进行标准化，使训练更精确

```
from sklearn.preprocessing import StandardScaler

# 标准化数据
scaler = StandardScaler()
diabetes_data_scaled = scaler.fit_transform(diabetes_data_clean.drop(['Outcome'], axis = 1))
diabetes_data_scaled = pd.DataFrame(diabetes_data_scaled, columns = diabetes_data_clean.columns[:-1])
# diabetes_data_scaled['Outcome'] = diabetes_data_clean['Outcome']

diabetes_data_scaled.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.865108	-0.033518	6.655021e-01	-3.345079e-16	0.166292	0.468492	1.425995
1	-0.844885	-1.206162	-0.529859	-1.746338e-02	-3.345079e-16	-0.852531	-0.365061	-0.190672
2	1.233880	2.015813	-0.695306	8.087936e-16	-3.345079e-16	-1.332833	0.604397	-0.105584
3	-0.844885	-1.074652	-0.529859	-7.004289e-01	-7.243887e-01	-0.634212	-0.920763	-1.041549
4	-1.141852	0.503458	-2.680669	6.655021e-01	1.465506e-01	1.548980	5.484909	-0.020496

1.1.6 总结

1. 这种较小型的数据集一般数据比较准确，所以错误数据并不太可能出现
2. 离群值在这种精确的数据集中反而能反应一些特征，比如这里BMI较大的患者，大概率是因为糖尿病导致的肥胖。
   在第一次的实验中，我用Z-Core删除了一定阈值外的数据，导致各特征与标签的相关性大大降低，背道而驰。

1.2 逻辑回归模型

1.2.1 构建预测模型

将数据集进行拆分，其中80%作为训练集，20%作为测试集。

```
# 构建模型
from sklearn.model_selection import train_test_split

source_x = diabetes_data_scaled
source_y = diabetes_data_clean['Outcome']
train_X, test_X, train_y, test_y = train_test_split(source_x, source_y, test_size = 0.2, random_state = 0)

# 训练集与测试集的大小
print("train data:  ", train_X.shape)
print("test data:   ", test_X.shape)
print("train label: ", train_y.shape)
print("test label:  ", test_y.shape)
```

```
train data:   (614, 8)
test data:    (154, 8)
train label:  (614,)
test label:   (154,)
```

1.2.2 训练模型

1. 选择Logistic Regression对数据进行训练
   同时选择评估模型对预测结果进行输出

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# 逻辑回归
lr = LogisticRegression()
lr.fit(train_X, train_y)
pred_y = lr.predict(test_X)

# 准确率
print("Accuracy: ", accuracy_score(test_y, pred_y))
# 混淆矩阵
confusion_matrix(test_y, pred_y)
```

```
Accuracy:  0.8116883116883117
array([[97, 10],
       [19, 28]], dtype=int64)
```

2. 修改模型超参数后再次训练
   加入L2正则化，同时增大迭代次数

```
lr = LogisticRegression(C = 0.1, max_iter = 1000)
lr.fit(train_X, train_y)
pred_y = lr.predict(test_X)
print("Accuracy: ", accuracy_score(test_y, pred_y))
```

```
Accuracy:  0.8051948051948052
```

1.2.3 模型改进 (K-Fold & MICE)

K折交叉验证

猜测：对于这种小型数据集，使用K折交叉验证是否能再提升一些性能

```
# 试了一下sklearn的函数，后面附上手写K-Fold
from sklearn.model_selection import cross_val_score

LR = LogisticRegression(C = 1, max_iter = 100)
scores = cross_val_score(LR, source_x, source_y, cv = 10)
# 打印每折的准确率
for i in range(len(scores)):
    print("Fold %d: %f" % (i + 1, scores[i]))
"""
def get_k_fold_data(k, i, x, y):
assert k > 1
fold_size = x.shape[0] // k # calculate the size of each fold
x_train, y_train = None, None
for j in range(k):
    idx = slice(j * fold_size, (j + 1) * fold_size) # slice the data into k folds
    x_part, y_part = x[idx, :], y[idx]
    if j == i:
        x_valid, y_valid = x_part, y_part
    elif x_train is None:
        x_train, y_train = x_part, y_part
    else:
        x_train = torch.cat((x_train, x_part), 0)
        y_train = torch.cat((y_train, y_part), 0)
return x_train, y_train, x_valid, y_valid # return the training data and validation data for each fold
"""
```

```
Fold 1: 0.753247
Fold 2: 0.766234
Fold 3: 0.792208
Fold 4: 0.688312
Fold 5: 0.779221
Fold 6: 0.792208
Fold 7: 0.779221
Fold 8: 0.792208
Fold 9: 0.710526
Fold 10: 0.842105
```

效果很好，第十折直接飙到 84.2% 了。
后来试了下20折，有几组能达到 90% 以上了，最高 93% ，确实分成小折之后有些参数表现较好，平均能力也有些提升。

MICE多元插补

猜测：使用链式回归的缺失值填充方式，可能会带来更多正相关的数据

```
# MICE填充缺失值
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

mice_imputer = IterativeImputer()
diabetes_data_mice = mice_imputer.fit_transform(diabetes_data_clean)
diabetes_data_mice = pd.DataFrame(diabetes_data_mice, columns = diabetes_data_clean.columns)

source_x = diabetes_data_mice.drop(['Outcome'], axis = 1)
source_y = diabetes_data_clean['Outcome']
train_X, test_X, train_y, test_y = train_test_split(source_x, source_y, test_size = 0.2, random_state = 0)

lr = LogisticRegression(C = 1, max_iter = 1000)
lr.fit(train_X, train_y)
pred_y = lr.predict(test_X)
print("Accuracy: ", accuracy_score(test_y, pred_y))
```

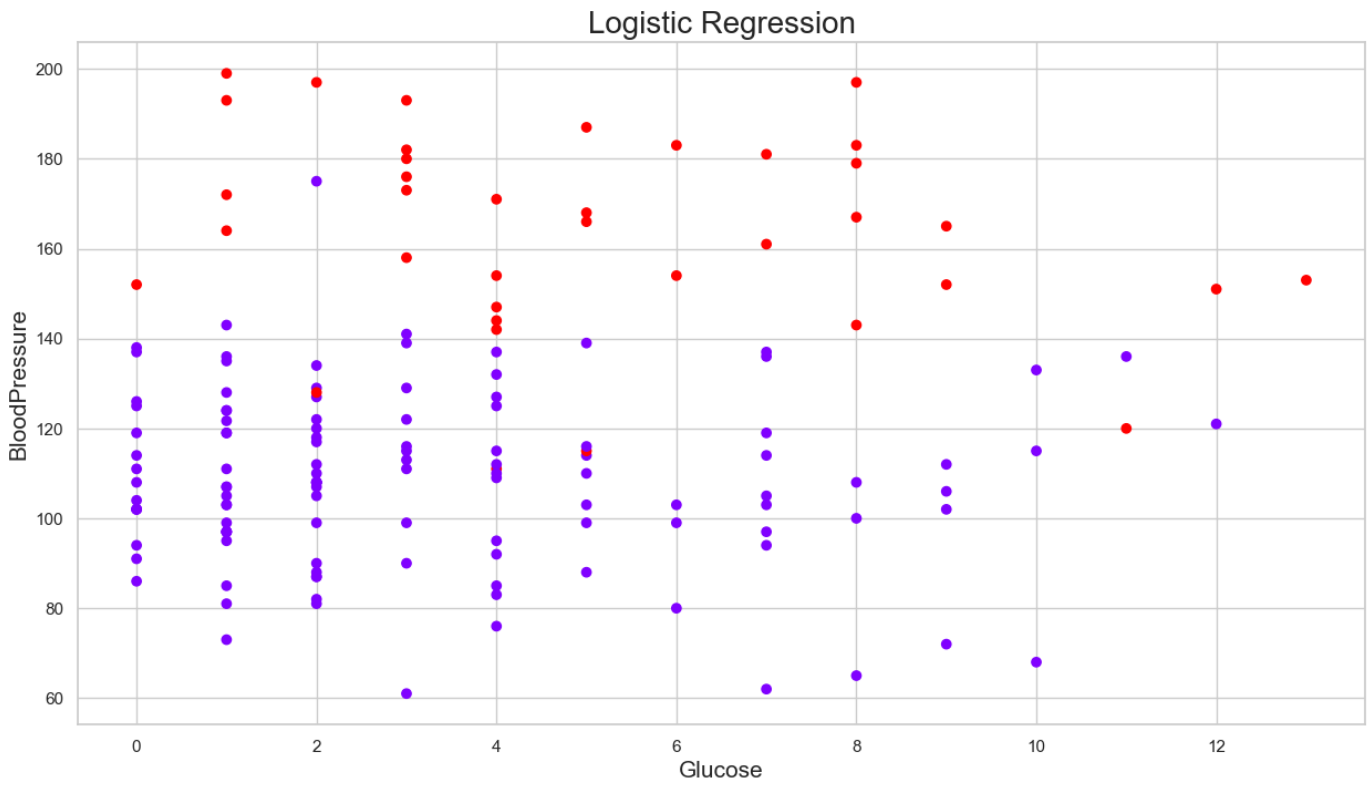
```
Accuracy:  0.8051948051948052
```

最后只能跑到 80.5% 左右，和直接用均值/中值填补差不多，也证实了我的想法，因为数据集的`Outcome`并不是五五开的，并且规模较小，所以填充后并不能为预测提供更多的有用信息。

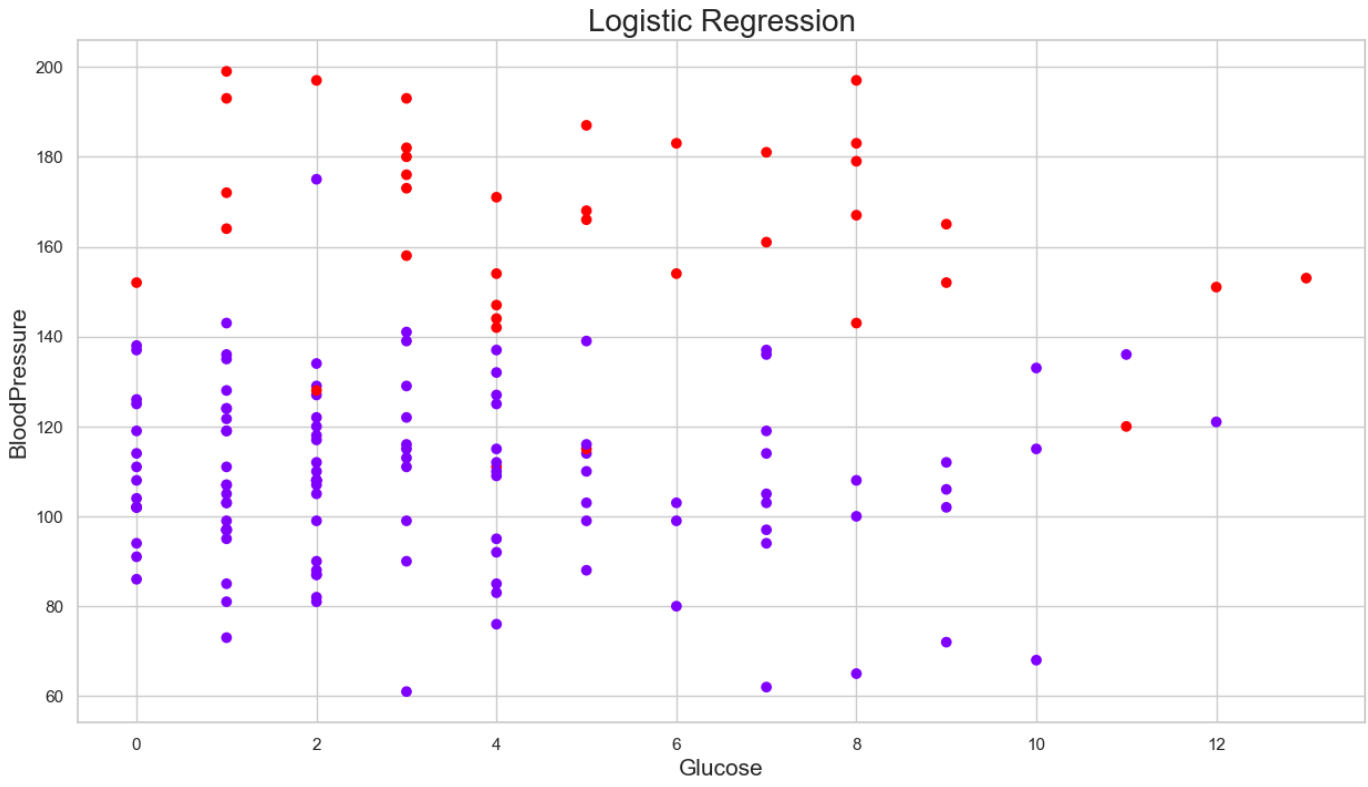
1.2.4 散点图绘制

1. 原始Logistic Regression

```
plt.figure(figsize=(15, 8))
plt.scatter(test_X.iloc[:, 0], test_X.iloc[:, 1], c = pred_y, cmap = 'rainbow')
plt.title('Logistic Regression', fontsize = 20)
plt.xlabel('Glucose', fontsize = 15)
plt.ylabel('BloodPressure', fontsize = 15)
plt.show()
```



2. K-Fold



1.3 数据集拆分

ration定为0.8, 准确率为84.2%

```
source_x = diabetes_data_scaled
source_y = diabetes_data_clean['Outcome']
train_X, test_X, train_y, test_y = train_test_split(source_x, source_y, test_size = 0.2, random_state = 0)
```

1.4 特征值与病因分析

1.4.1 相关性查看

将糖尿病人的几项特征进行排序，观察正相关程度，将其排序后进行后续处理。
这里针对简单线性数据，我直接使用的pearson相关性，也可以使用泛化性更好的spearman方法

```
correspond = diabetes_data_clean.corr()
correspond['Outcome'].sort_values(ascending = False)
```

```
Outcome                   1.000000
Glucose                   0.492928
BMI                       0.311924
Age                       0.238356
Pregnancies               0.221898
SkinThickness             0.215299
Insulin                   0.214411
DiabetesPedigreeFunction  0.173844
BloodPressure             0.166074
Name: Outcome, dtype: float64
```

再查看葡萄糖和其他症状的相关性

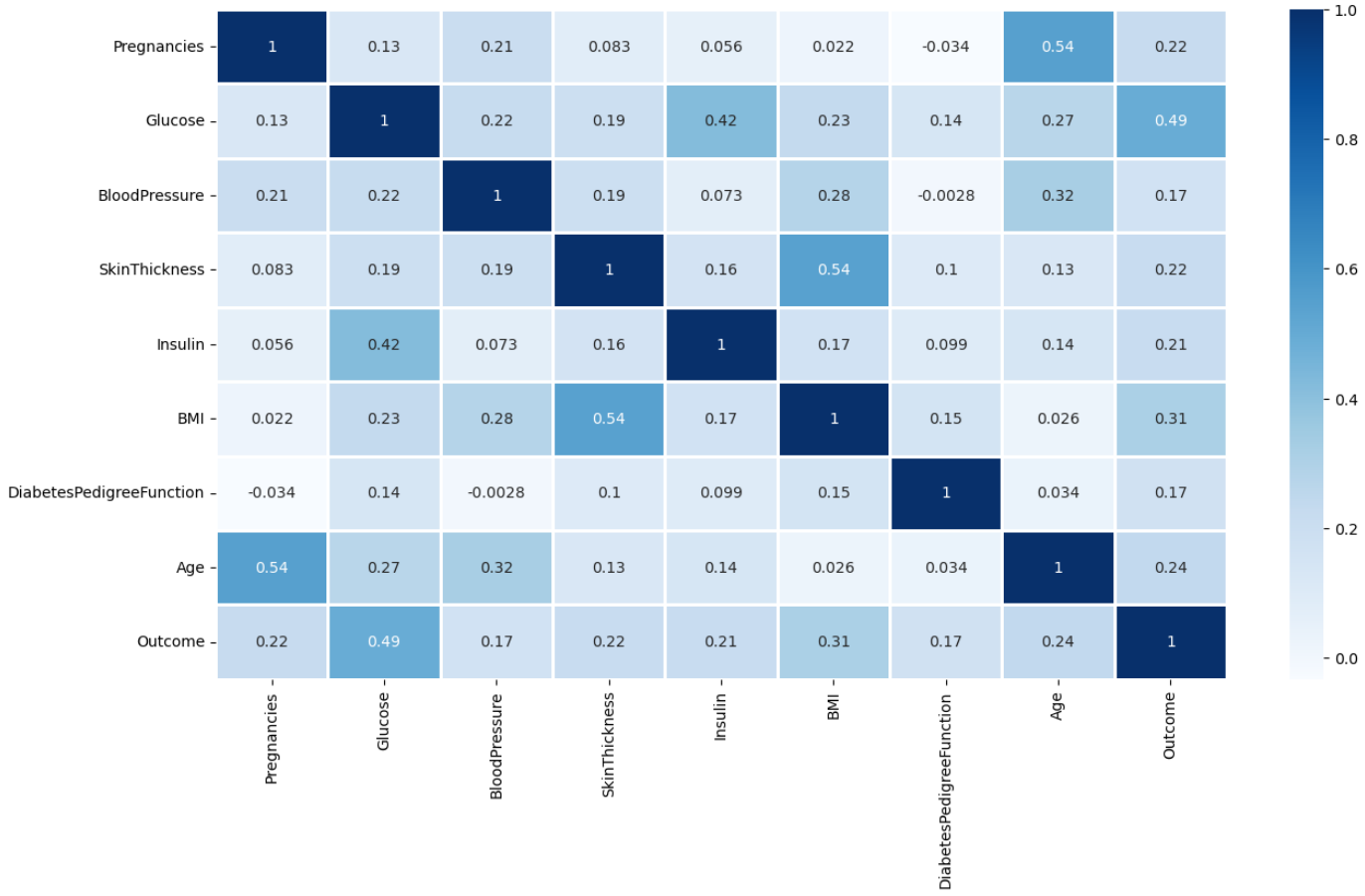
```
correspond['Glucose'].sort_values(ascending = False)
```

```
Glucose                   1.000000
Outcome                   0.492928
Insulin                   0.420157
Age                       0.266534
BMI                       0.230941
BloodPressure             0.218367
SkinThickness             0.192991
DiabetesPedigreeFunction  0.137060
Pregnancies               0.127911
Name: Glucose, dtype: float64
```

1.4.2 相关性可视化

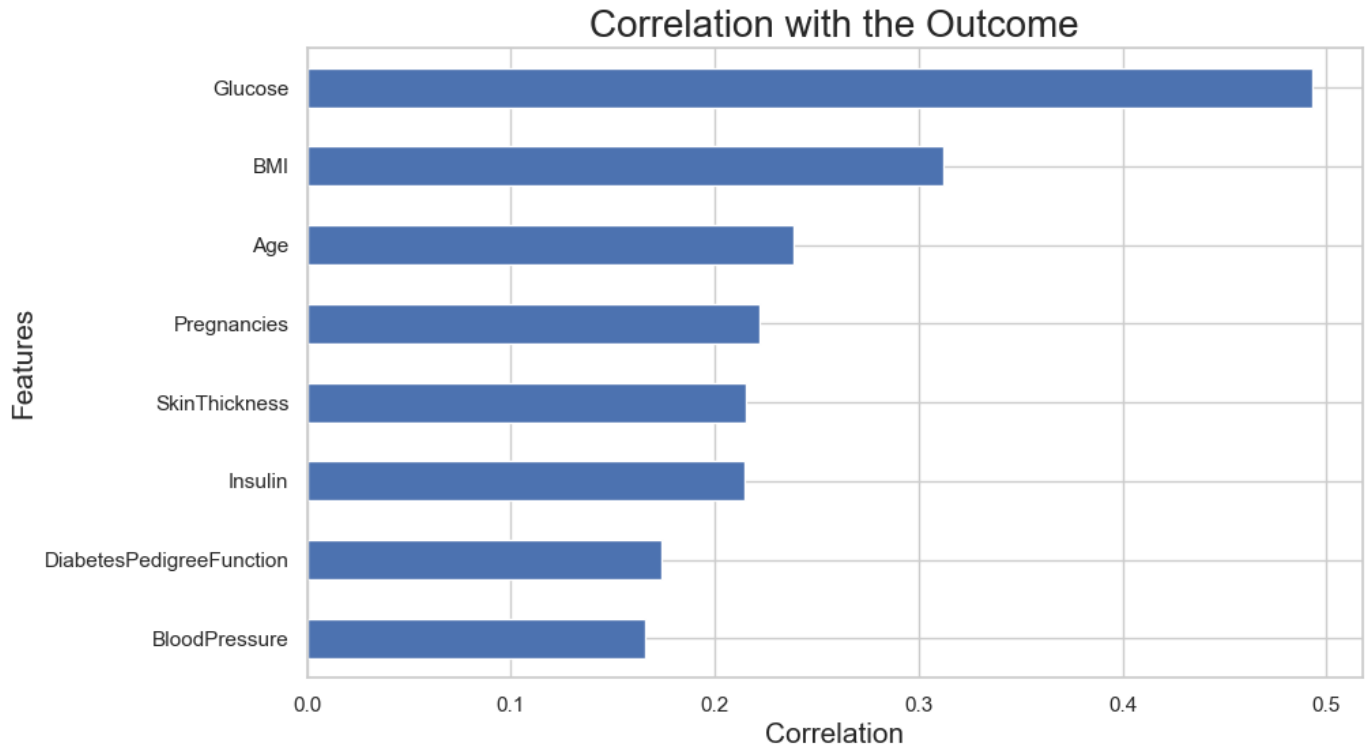
1. 热力图

```
plt.figure(figsize=(15, 8))
sns.heatmap(correspond, vmax = 1, square = False, annot=True, linewidths = 1, cmap='Blues')
```



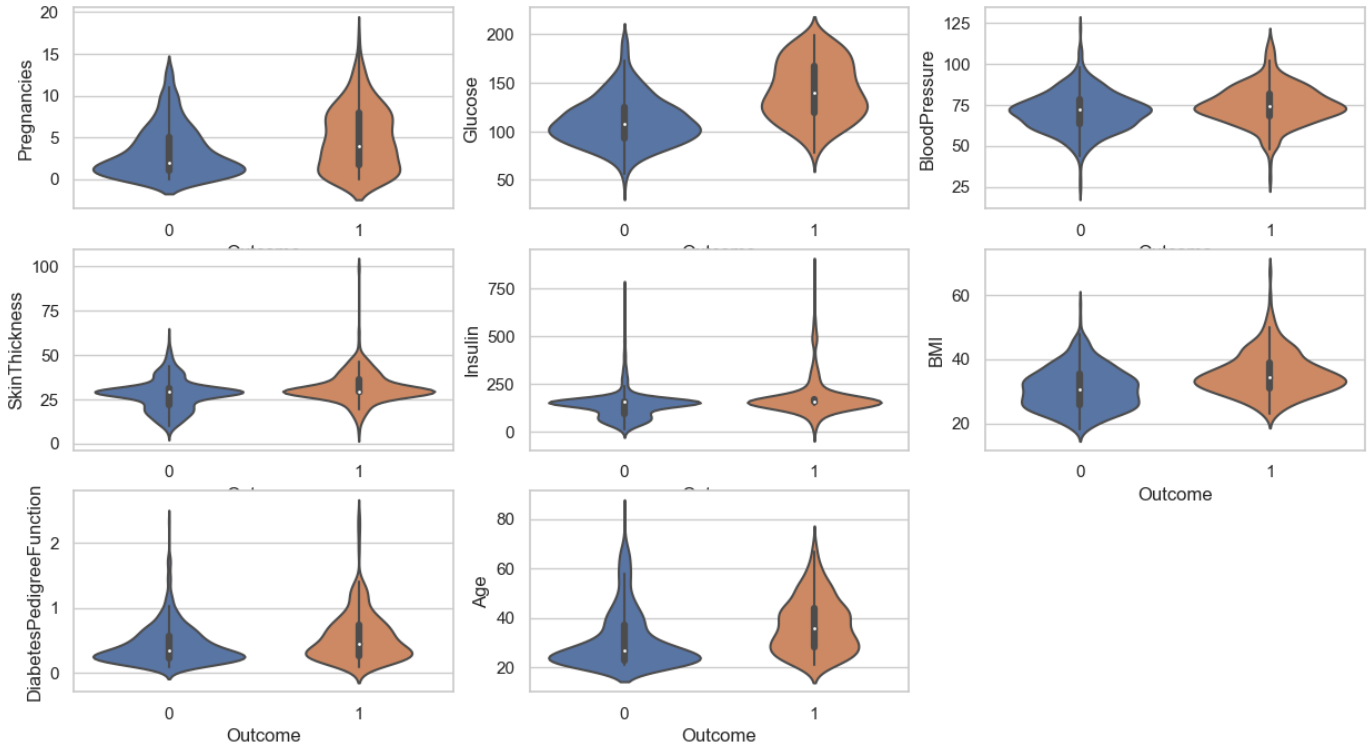
2. 柱状图

```
%config InlineBackend.figure_format = 'png'
import matplotlib

sns.set(font = 'Arial', style="whitegrid", color_codes=True)
matplotlib.rcParams['axes.unicode_minus'] = False

correlations = diabetes_data_clean.corrwith(diabetes_data_clean['Outcome']).sort_values(ascending = True)
plt.figure(figsize=(10, 6))
correlations.drop('Outcome').plot.barh()
plt.title('Correlation with the Outcome', fontsize = 20)
plt.xlabel('Correlation', fontsize = 15)
plt.ylabel('Features', fontsize = 15)
plt.show()
```



3. 琴状图

```
plt.figure(figsize=(15, 8))

for column_index in range(diabetes_data_clean.shape[1]):
    if column_index == 8:
        break
    plt.subplot(3, 3, column_index + 1)
    sns.violinplot(x = 'Outcome', y = diabetes_data_clean.iloc[:, column_index], data = diabetes_data_clean)
```



2. CIFAR-10 (Acc = 59.6%)

2.1 网络结构

利用Pytorch框架搭建 MLP，input_size, hidden_size, num_classes分别为网络输入/隐藏层/输出层神经元个数

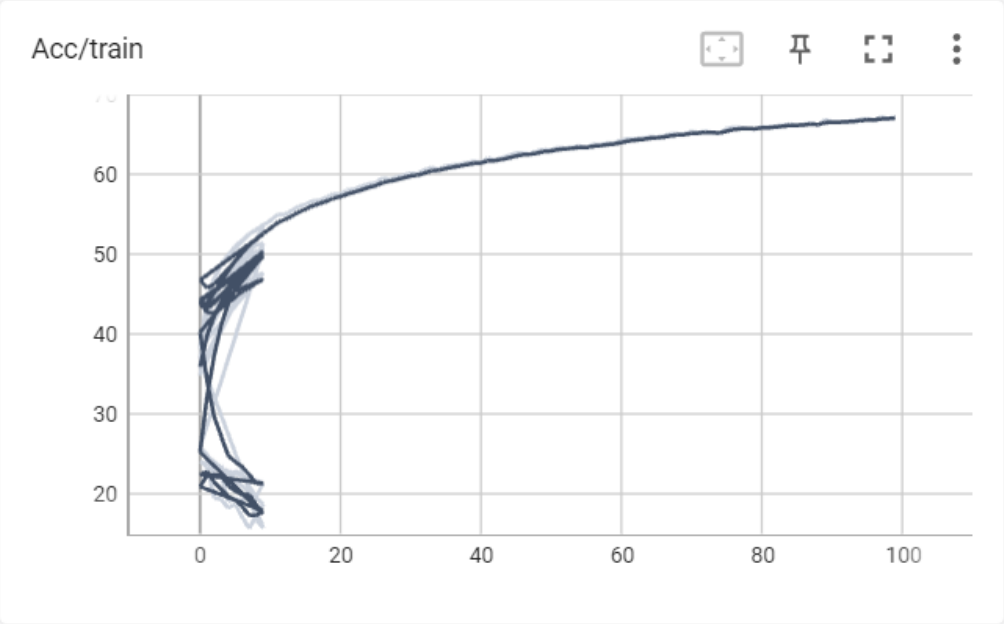
```
class MLP(torch.nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(MLP, self).__init__()

        h1, h2, h3 = hidden_size
        self.linear1 = torch.nn.Linear(input_size, h1) # 输入层到第一层隐藏层的线性变换
        self.relu1 = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(h1, h2) # 第一层隐藏层到第二层隐藏层的线性变换
        self.relu2 = torch.nn.ReLU()
        self.linear3 = torch.nn.Linear(h2, h3) # 第二层隐藏层到第三层隐藏层的线性变换
        self.relu3 = torch.nn.ReLU()
        self.linear4 = torch.nn.Linear(h2, num_classes) # 第四层隐藏层到输出层的线性变换
        self.dropout = torch.nn.Dropout(0.25)
        self.softmax = torch.nn.Softmax(dim=1)

    def forward(self, x):
        out = self.linear1(x)
        out = self.relu1(out)
        # out = self.dropout(out)
        out = self.linear2(out)
        out = self.relu2(out)
        # out = self.dropout(out)
        out = self.linear3(out)
        out = self.relu3(out)
        out = self.dropout(out)
        out = self.linear4(out)
        # out = self.softmax(out)
        return out
```

2.2 TensorBoard可视化展示

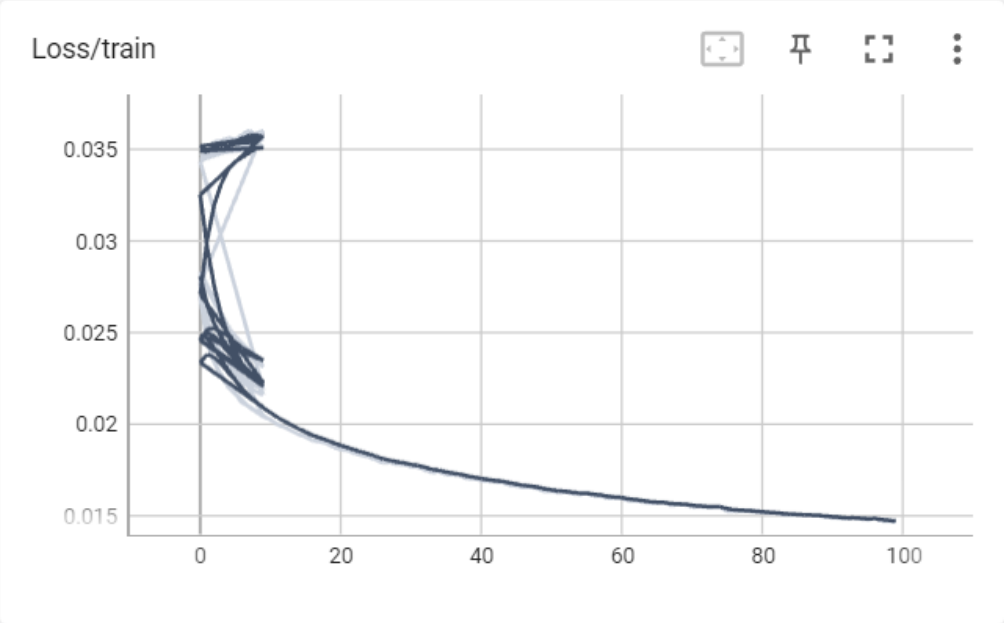
嫌麻烦就把它嵌入了
前面有几十个step显示有问题，没用太明白

```
from tensorboard import notebook
notebook.display(port=6006, height=1000)
%tensorboard --logdir ./runs/MLP
```

Acc



Loss



2.3 保存训练模型并训练

2.3.1 保存模型

```
input_size = IMAGE_SIZE * IMAGE_SIZE * COLOR_CHANNELS
hidden_size = [512, 256, 128]
num_classes = 10

model = MLP(input_size, hidden_size, num_classes)
if cuda:
    model = model.cuda()
train(model)

# Save Model
torch.save(model.state_dict(), './model/MLP.pth')
```

```
Epoch: 1 Loss: 0.027643 Acc: 36.910000 Time: 23.314
Epoch: 2 Loss: 0.025291 Acc: 42.934000 Time: 24.255
Epoch: 3 Loss: 0.024285 Acc: 45.066000 Time: 25.225
Epoch: 4 Loss: 0.023631 Acc: 46.622000 Time: 25.337
Epoch: 5 Loss: 0.023122 Acc: 47.740000 Time: 25.483
Finished Training
```

2.3.2 使用保存的模型测试

```
# 使用保存的模型进行预测
input_size = IMAGE_SIZE * IMAGE_SIZE * COLOR_CHANNELS
hidden_size = [512, 256, 128]
num_classes = 10
model = MLP(input_size, hidden_size, num_classes)
model.load_state_dict(torch.load('./model/MLP.pth'))
if cuda:
    model = model.cuda()
test(model, test_loader)

for X in test_loader:
    inputs, labels = X
    if cuda:
        inputs, labels = inputs.cuda(), labels.cuda()
    inputs = inputs.view(inputs.size(0), -1)
    outputs = model(inputs)
    _, predicted = torch.max(outputs, 1)
    # 输出图片
    print('预测结果: ')
    print(predicted)
    print('真实结果: ')
    print(labels)
    break
```

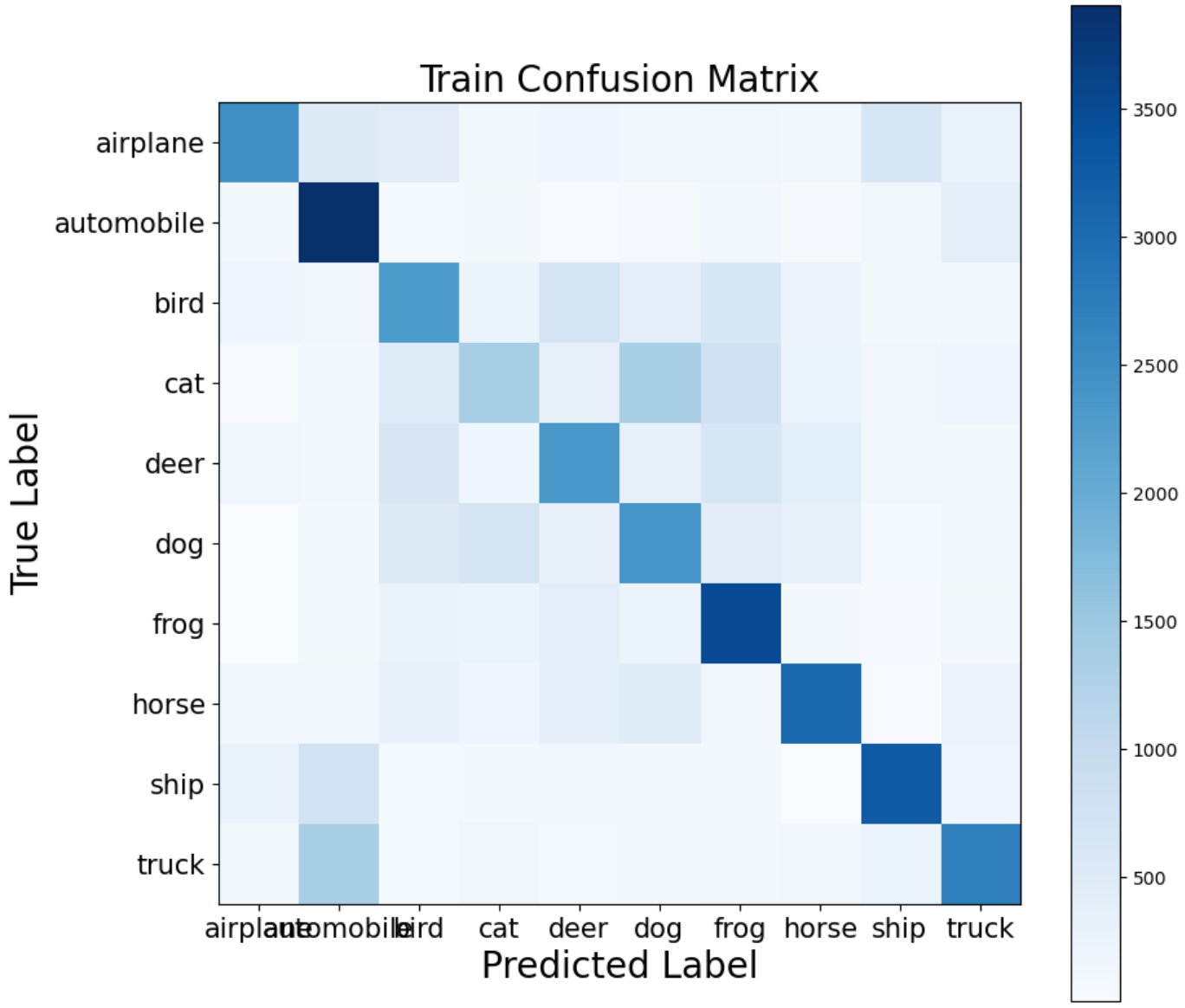
```
Test Acc: 50.190000 Time: 2.224
预测结果:
tensor([2, 2, 6, 1, 6, 1, 5, 5, 9, 6, 2, 1, 5, 6, 8, 5, 0, 9, 0, 5, 0, 0, 2, 1,
        8, 2, 0, 0, 9, 6, 9, 5, 5, 6, 6, 0, 3, 5, 7, 0, 0, 0, 5, 1, 6, 8, 4, 3,
        4, 0, 0, 5, 1, 6, 5, 0, 4, 8, 4, 0, 2, 2, 3, 6], device='cuda:0')
真实结果:
tensor([2, 0, 5, 1, 4, 1, 5, 3, 9, 6, 2, 1, 4, 2, 8, 4, 5, 9, 8, 5, 3, 0, 3, 9,
        8, 4, 7, 7, 7, 4, 9, 3, 1, 6, 2, 0, 3, 0, 7, 9, 0, 8, 6, 9, 6, 8, 4, 5,
        6, 4, 0, 5, 1, 3, 5, 0, 9, 8, 5, 0, 2, 1, 2, 6], device='cuda:0')
```

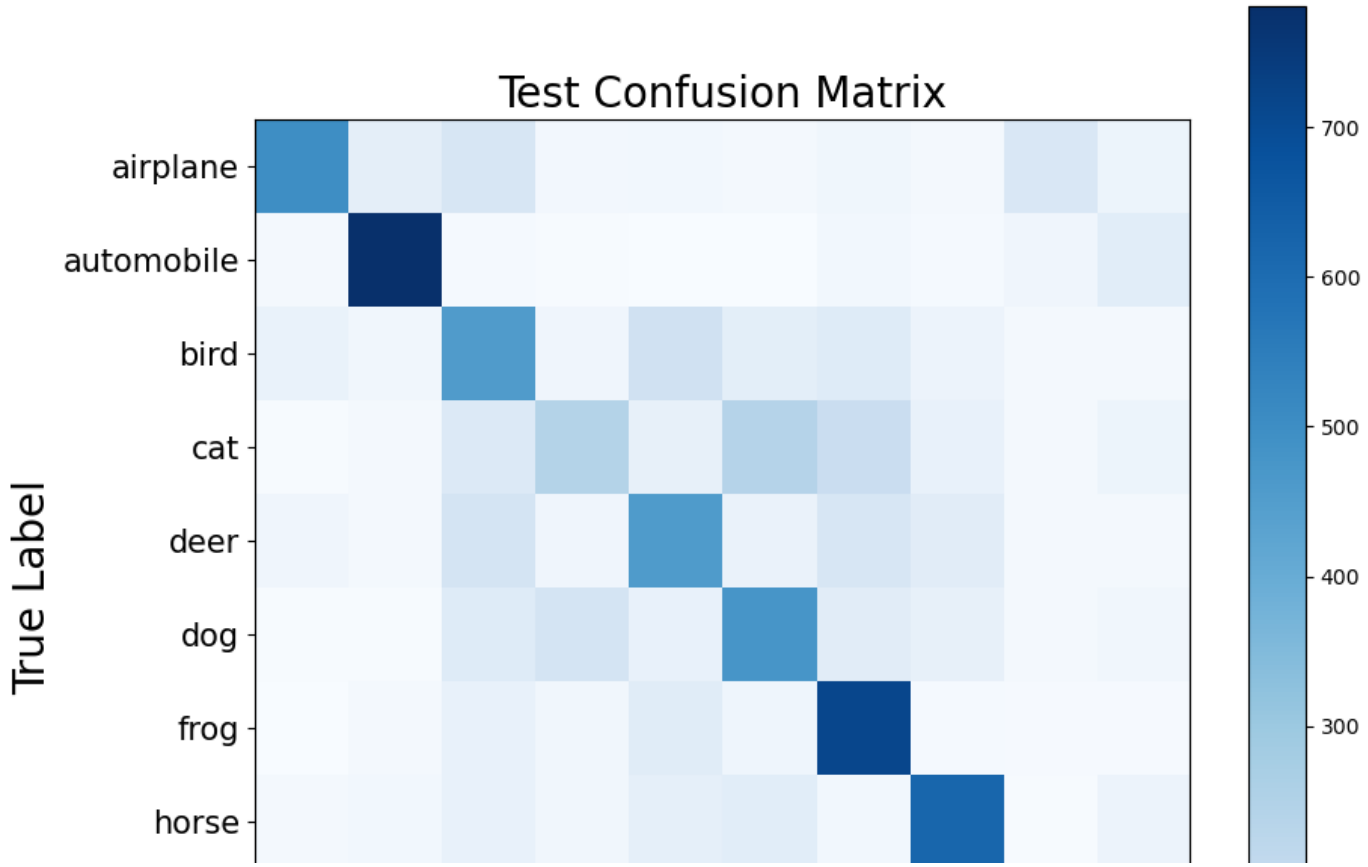
2.4 混淆矩阵

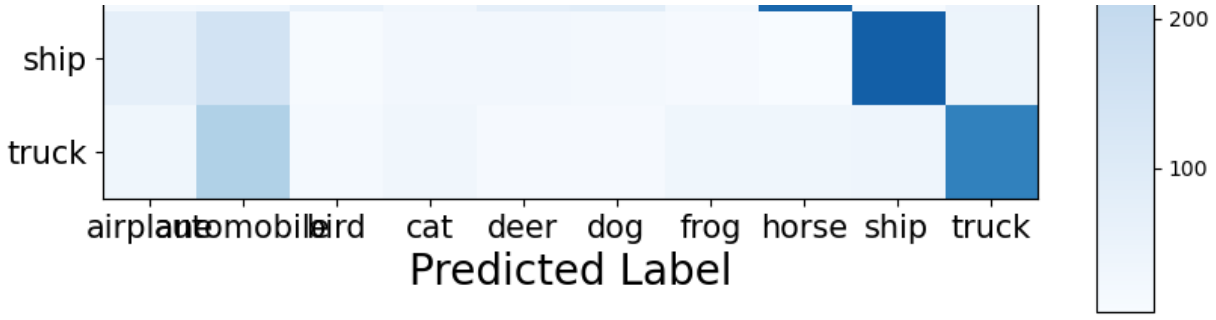
使用sklearn的confusion_metrix对训练集与测试集进行混淆矩阵的绘制，直观的反应预测的准确率。
对于这种多分类的问题，混淆矩阵可以很直观的反应各个类别的训练情况，对于CIFAR-10来说，对于"Dog""Deer""Cat"的识别较为差劲，误识别为"Frog"的概率很高，后续可进一步探究如何偏差训练。

```
train_confusion_matrix = torch.zeros(NUM_CLASSES, NUM_CLASSES)
test_confusion_matrix = torch.zeros(NUM_CLASSES, NUM_CLASSES)

def plot_confusion_matrix(cm, classes, title='Confusion Matrix', cmap=plt.cm.Blues):
    plt.figure(figsize=(10, 10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=20)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, fontsize=15)
```

```
    plt.yticks(tick_marks, classes, fontsize=15)
    plt.xlabel('Predicted Label', fontsize=20)
    plt.ylabel('True Label', fontsize=20)
    plt.show()

def get_confusion_matrix(model, loader, confusion_matrix):
    model.eval()
    for data in loader:
        inputs, labels = data
        if cuda:
            inputs, labels = inputs.cuda(), labels.cuda()
        inputs = inputs.view(inputs.size(0), -1)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        for i in range(len(labels)):
            confusion_matrix[labels[i], predicted[i]] += 1
    return confusion_matrix

train_confusion_matrix = get_confusion_matrix(model, train_loader, train_confusion_matrix)
test_confusion_matrix = get_confusion_matrix(model, test_loader, test_confusion_matrix)

plot_confusion_matrix(train_confusion_matrix.numpy(), label_names, title='Train Confusion Matrix')
plot_confusion_matrix(test_confusion_matrix.numpy(), label_names, title='Test Confusion Matrix')
```


Train Confusion Matrix
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
True Label
Predicted Label
3500
3000
2500
2000
1500
1000
500


Test Confusion Matrix
airplane
automobile
bird
cat
deer
dog
frog
horse
True Label
700
600
500
400
300



2.5 分析网络参数

先把Output放在最前面

Acc = 22.55 有dropout 有softmax batch_size = 64 四层MLP epochs = 10 lr = 0.001

```
Test Acc: 22.550000 Time: 1.892
预测结果:
tensor([0, 0, 6, 7, 1, 6, 7, 7, 6, 1, 6, 0, 0, 0, 0, 6, 0, 1, 0, 7, 1, 6, 0, 1,
        0, 0, 6, 6, 7, 0, 7, 6, 6, 1, 6, 0, 6, 1, 7, 0, 6, 6, 0, 6, 6, 7, 1, 1,
        1, 1, 6, 6, 0, 1, 1, 7, 1, 1, 0, 0, 1, 6, 6, 1], device='cuda:0')
真实结果:
tensor([3, 0, 3, 3, 3, 6, 6, 0, 5, 2, 6, 0, 0, 2, 2, 7, 8, 4, 9, 5, 7, 6, 8, 9,
        0, 8, 6, 3, 3, 5, 5, 2, 1, 0, 4, 2, 2, 0, 5, 2, 7, 4, 8, 3, 3, 5, 0, 9,
        4, 8, 5, 4, 9, 7, 3, 5, 8, 3, 8, 0, 0, 9, 3, 5], device='cuda:0')
```

Acc = 23.00 有dropout 有softmax batch_size = 64 三层MLP epochs = 50 lr = 0.001

```
Test Acc: 23.000000 Time: 1.719
预测结果:
tensor([9, 9, 8, 7, 8, 7, 8, 0, 7, 7, 5, 8, 8, 8, 0, 7, 7, 7, 5, 8, 8, 7, 9, 5,
        5, 8, 5, 1, 9, 5, 1, 5, 5, 8, 8, 9, 8, 7, 8, 7, 1, 1, 7, 1, 8, 5, 7, 7,
        9, 5, 1, 7, 8, 8, 7, 9, 7, 8, 9, 8, 7, 1, 7, 1], device='cuda:0')
真实结果:
tensor([0, 3, 8, 0, 4, 4, 5, 8, 6, 7, 5, 8, 8, 8, 0, 6, 2, 9, 7, 0, 8, 4, 4, 2,
        5, 2, 5, 8, 0, 2, 1, 8, 5, 2, 8, 3, 8, 2, 0, 2, 2, 1, 4, 3, 5, 3, 2, 3,
        7, 5, 1, 3, 7, 0, 6, 1, 2, 9, 7, 1, 4, 9, 5, 3], device='cuda:0')
```

Acc = 48.51 激活函数Sigmoid 有dropout 有softmax batch_size = 64 三层MLP epochs = 10 lr = 0.001

```
Test Acc: 48.510000 Time: 1.667
预测结果:
tensor([0, 0, 8, 5, 0, 8, 1, 7, 7, 3, 5, 2, 9, 7, 2, 6, 0, 0, 0, 6, 2, 5, 4, 7,
        4, 4, 9, 7, 6, 2, 4, 4, 7, 7, 3, 8, 6, 0, 2, 1, 8, 7, 3, 3, 8, 0, 0, 7,
        3, 9, 0, 5, 4, 3, 7, 8, 1, 3, 6, 8, 0, 4, 0, 7], device='cuda:0')
真实结果:
tensor([0, 0, 8, 3, 0, 7, 8, 2, 7, 3, 0, 2, 1, 7, 5, 6, 8, 8, 0, 6, 2, 3, 4, 7,
        6, 7, 3, 9, 7, 8, 3, 4, 5, 7, 3, 1, 7, 2, 2, 8, 0, 7, 7, 4, 0, 0, 5, 7,
        5, 0, 0, 5, 4, 5, 2, 8, 1, 2, 2, 8, 0, 2, 8, 7], device='cuda:0')
```

Acc = 52.08 有dropout 无softmax batch_size = 64 四层MLP epochs = 10 lr = 0.001

```
Test Acc: 52.080000 Time: 1.852
预测结果:
tensor([7, 7, 6, 2, 4, 8, 9, 9, 1, 1, 3, 6, 0, 0, 1, 4, 8, 2, 7, 5, 8, 8, 7, 7,
        6, 4, 2, 8, 8, 2, 2, 9, 8, 1, 7, 9, 5, 5, 5, 6, 5, 8, 1, 4, 5, 0, 8, 3,
        0, 2, 5, 0, 4, 5, 4, 5, 9, 8, 0, 6, 4, 9, 1, 1], device='cuda:0')
真实结果:
tensor([4, 7, 8, 2, 4, 8, 9, 1, 1, 1, 3, 1, 0, 0, 1, 3, 8, 2, 5, 3, 5, 8, 7, 7,
```

```
        6, 2, 2, 8, 8, 0, 6, 3, 8, 1, 5, 3, 0, 5, 6, 6, 4, 8, 1, 4, 7, 4, 2, 3,
        0, 6, 5, 0, 4, 3, 2, 3, 1, 0, 7, 6, 4, 1, 1, 1], device='cuda:0')
```

Acc = 54.21 无dropout 无softmax batch_size = 64 四层MLP epochs = 10 lr = 0.001

```
Test Acc: 54.210000 Time: 2.260
预测结果:
tensor([7, 6, 7, 8, 5, 7, 5, 7, 1, 7, 3, 2, 1, 3, 4, 9, 6, 4, 9, 1, 6, 5, 0, 1,
        1, 6, 8, 9, 2, 6, 7, 2, 6, 9, 5, 0, 6, 2, 6, 6, 1, 7, 7, 8, 5, 1, 7, 3,
        1, 3, 9, 1, 6, 7, 8, 0, 0, 0, 8, 2, 1, 1, 6, 6], device='cuda:0')
真实结果:
tensor([4, 6, 7, 2, 3, 7, 5, 7, 1, 7, 6, 0, 1, 3, 3, 8, 5, 7, 9, 9, 2, 7, 4, 1,
        1, 3, 8, 1, 6, 9, 7, 2, 3, 9, 5, 0, 6, 2, 6, 6, 1, 7, 7, 8, 5, 8, 6, 7,
        1, 6, 9, 1, 2, 7, 0, 0, 2, 1, 8, 7, 1, 1, 2, 6], device='cuda:0')
```

Acc = 59.55 无dropout 无softmax batch_size = 64 四层MLP epochs = 100 lr = 0.001

```
Test Acc: 59.550000 Time: 2.293
预测结果:
tensor([5, 2, 5, 4, 2, 6, 9, 8, 6, 6, 0, 9, 8, 7, 3, 9, 7, 9, 4, 4, 0, 8, 2, 6,
        7, 9, 5, 7, 8, 9, 8, 9, 4, 8, 2, 8, 7, 5, 9, 0, 0, 4, 8, 4, 0, 7, 4, 3,
        1, 2, 9, 2, 2, 4, 2, 7, 0, 3, 9, 8, 9, 1, 5, 0], device='cuda:0')
真实结果:
tensor([5, 2, 8, 7, 5, 6, 9, 0, 4, 9, 8, 6, 8, 7, 5, 9, 7, 1, 5, 4, 0, 8, 2, 6,
        5, 9, 8, 7, 8, 2, 9, 9, 7, 8, 4, 8, 7, 5, 6, 0, 0, 4, 8, 4, 7, 7, 2, 4,
        8, 2, 9, 4, 0, 4, 2, 7, 0, 4, 9, 9, 9, 8, 2, 0], device='cuda:0')
```

2.5.1 网络深度

四层网络收效较好
对于这种多分类问题，当使用ReLU这种受层数影响不大的激活函数时，合理选择网络深度能收获更好的训练效果，当然，针对CIFAR-10这种较小的数据集，目前实验结果证明四层是一个比较理想的层数，不过这同样和隐藏层大小有关，只要特征丢失较少，收效就还不错。

2.5.2 激活函数

ReLU收效较好
只尝试了Sigmoid和ReLU函数，由于本模型深度不深，所以不会出现梯度消失的情况，故两种函数在这个问题上表现都还不错。

2.5.3 神经元数量

Num_Class = 10
对于MLP，神经元数量的设置通常是一个关键因素。太少的神经元可能无法捕捉复杂的模式，而太多的神经元可能导致过拟合。因为是十分类问题，所以就使用了10个神经元，同时，隐藏层的对接也要合适。

2.6 关于softmax的使用

实验结果

在使用了CrossEntropy的条件下，最后一层加入Softmax函数，准确率严重下降，从50%左右降至20%左右

原因分析

1. Torch.nn.CrossEntropyLoss() 食用指南

   计算公式如下:

$$\ell(x, y) = L = l_1, \ldots, l_N{}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1y_n \neq \text{ignore}_{\text{index}}$$

2. Torch.nn.Soft(dim) 食用指南

   公式如下:

$$\mathrm{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

3. 很明显，`Torch`自己的交叉熵损失函数中已经进行了`Softmax`的归一化操作，再来一次就耦合了。