上海理工大学光电信息与计算机工程学院

《机器学习》2023-2024 学年第二期
期末报告

专　　业　计算机科学与技术

学生姓名　黄旭铭

学　　号　2135050214

年　　级　大二

指导教师　彭敦陆

课程代码　2004140

成　　绩

教师签字

基于机器学习的图像分类技术

1. 项目背景

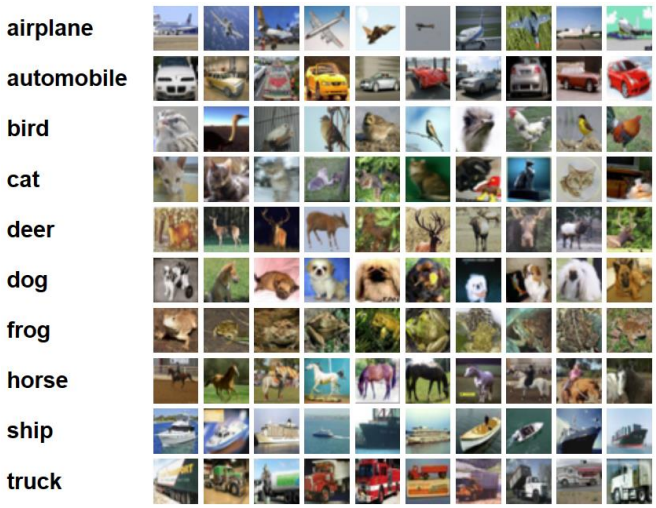
图像分类是计算机视觉领域中的一项基础任务，它涉及将图像自动分配到一个或多个类别中。与文本分类类似，图像分类的目标是构建一个模型，该模型能够识别图像中的特征，并根据这些特征将图像分类到相应的类别。图像分类是计算机视觉领域的核心问题之一，相关研究成果已广泛应用于各行各业的视觉处理中。

目前，图像分类的主流方式是采用机器学习方法（包括深度学习方法）。将数据集中的图片作为长×宽×高的三维矩阵数据输入到预先定义的模型中，然后根据得到的输出进行分类。不过现在流行的神经网络模型基本为黑盒，其可解释性较差。

本项目旨在采用机器学习方法对大量图像进行学习的基础上，实现给定图像可能的类别进行预测。

2．数据集简介

给定的数据集为 CIFAR10[图 1]，图像均为 32*32 像素，并且已分好训练集 50000 张图片，测试集 10000 张图片[图 2]。



CIFAR10 图 1

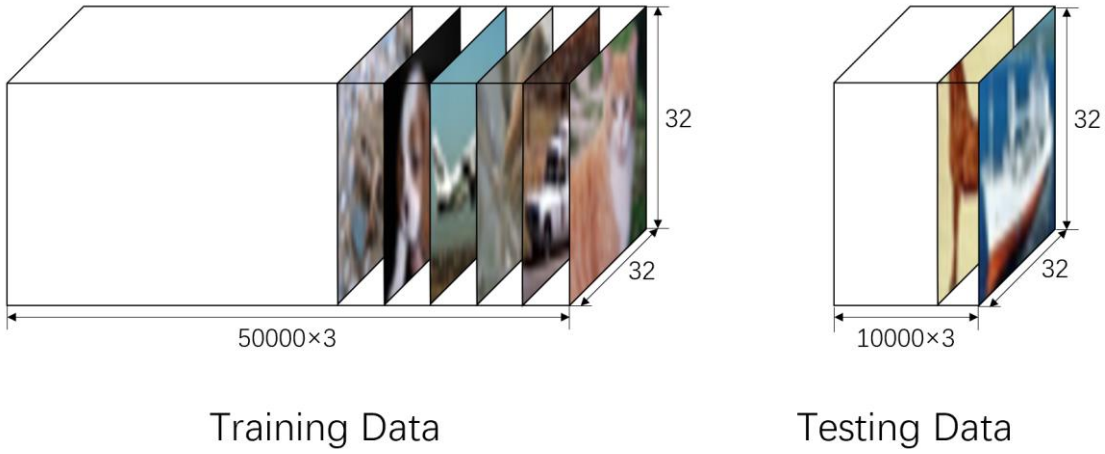


图 2

3. 数据处理

1. 使用原始像素值

处理方式：这种方法直接使用图像的灰度像素值。图像首先被转换成灰度格式，以简化处理流程并减少数据量（从三通道变为单通道）。然后，图像被展平成一个一维数组，这个数组的长度等于图像的像素数（对于 32x32 像素的图像，长度为 1024）。这种方法直接反映了图像的原始视觉内容。

数据分析：分析原始像素值主要是观察像素强度的分布，可以通过直方图来展示。不过，这种方法通常不提供关于图像内容或结构的深入洞察，因为它没有考虑像素之间的关系。

2. 颜色直方图

处理方式：颜色直方图通过统计图像中每种颜色出现的频率来提取特征。在我们的示例中，我单独为 RGB 三个通道计算了直方图，每个通道分成 32 个 bins，这样可以捕获颜色的分布情况而不仅仅是强度。然后将这三个直方图串联起来形成一个特征向量。

数据分析：通过分析颜色直方图，我们可以了解图像中颜色的使用和分布情况。例如，某一个颜色为主的图像在蓝色通道的直方图中会有较高的峰值[图 3]。但是颜色直方图对于图像的大小和某些形式的旋转是不敏感的，但它无法捕捉位置信息。

3. 局部二值模式（LBP）

处理方式：LBP 是一种描述图像局部纹理特征的方法。对于图像中的每个像素，LBP 算法将其邻域像素的灰度值与该像素的灰度值比较，生成一个二进制数。这些二进制数的直方图形成了一个特征描述符。在我们的例子中，我们选择了半径为 1 的 8 个邻域点。

数据分析：LBP 特征有助于分析图像的纹理信息，例如纹理的粗糙度或平滑度。通过观察不同模式的分布，LBP 特征对图像的纹理特征有更好的理解，对光照变化相对稳定，但相似的纹理和亮度会混淆边界[图 4]。

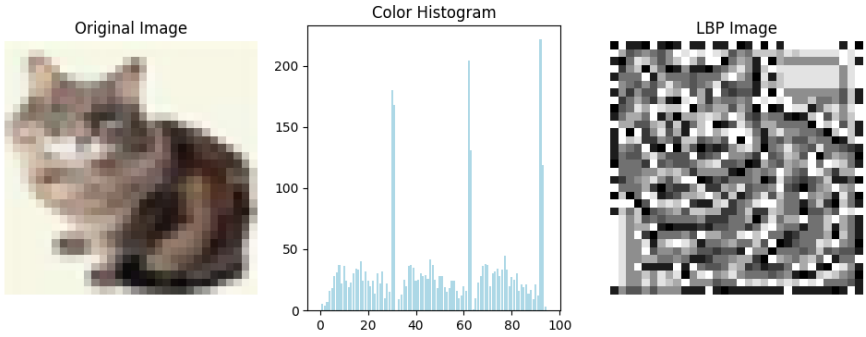


图 3

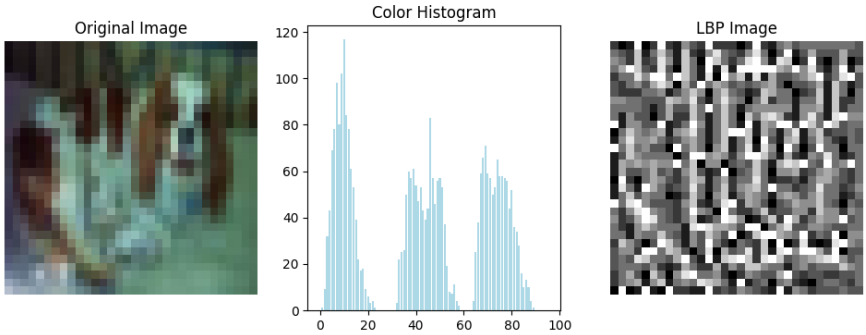


图 4

通过分析，我摒弃了颜色直方图和局部二值模式的数据处理，采用原始像素值以保留尽可能的图像信息。代码如下：

```
#灰度图
image = Image.open("path_to_your_image.jpg").resize((32, 32))
image = np.array(image)
gray_image = rgb2gray(image)

# 使用原始像素值
def extract_pixel_values(image):
    return image.flatten()

#颜色直方图
def extract_color_histogram(image, bins=32, range=[0, 256]):
    # R, G, B 三通道直方图
```

```
    histogram = [np.histogram(image[:,:,i], bins=bins, range=range)[0]
                 for i in range(3)]
    return np.concatenate(histogram)

#二值模式特征
def extract_lbp_features(image, P=8, R=1):
    lbp_image = local_binary_pattern(image, P, R, method="uniform")
    lbp_hist, _ = np.histogram(lbp_image.ravel(), bins=np.arange(0, P * R + 3),
range=(0, P * R + 2))
    return lbp_hist

```

4. 传统机器学习方法

1. 支持向量机（SVM）

基本模型定义为特征空间上间隔最大的线性分类器，间隔最大使它具有一定的泛化能力。主要思想是在特征空间中寻找一个超平面，使得两个类别的数据在该超平面两侧，且到超平面的距离最大。

通过三个小时的运行后，放弃继续运行。剖析背后原因有二：一是时间复杂度过高 $O(f \cdot s^2) = O(32 \times 32 \times 3 \times 50000^2)$，二是因为梯度下降需要储存所有样本参数导致内存占用过高。

于是改用随机梯度下降进行优化（SGD），训练时间为 87.78s，准确度为 16.72%。效果很差，尝试一下解释：

如图[5, 6]，Loss 值居高不下，且只有二类的效果较好（airplane，truck）并严重混淆其他类。

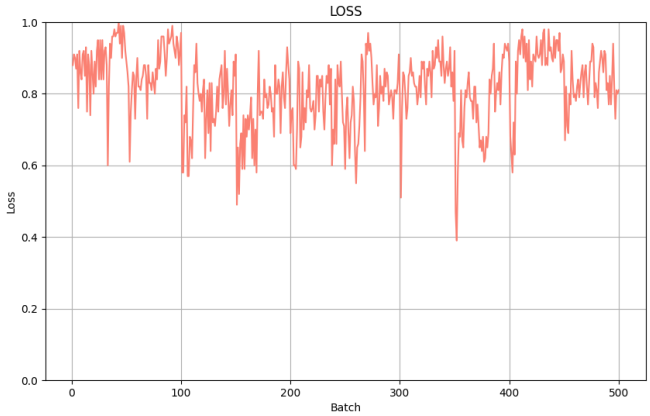


图 5

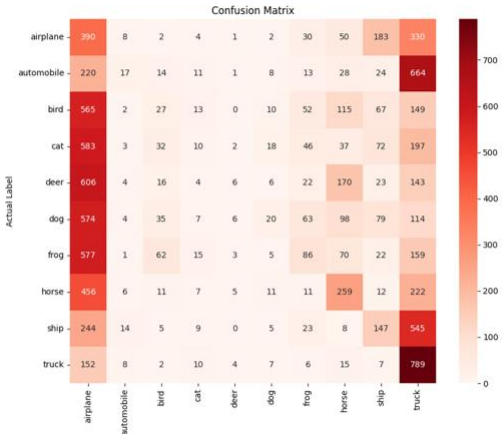


图 6

因此容易得出 Loss 函数卡在局部最优点图[7]，从而寻找下一个方法。

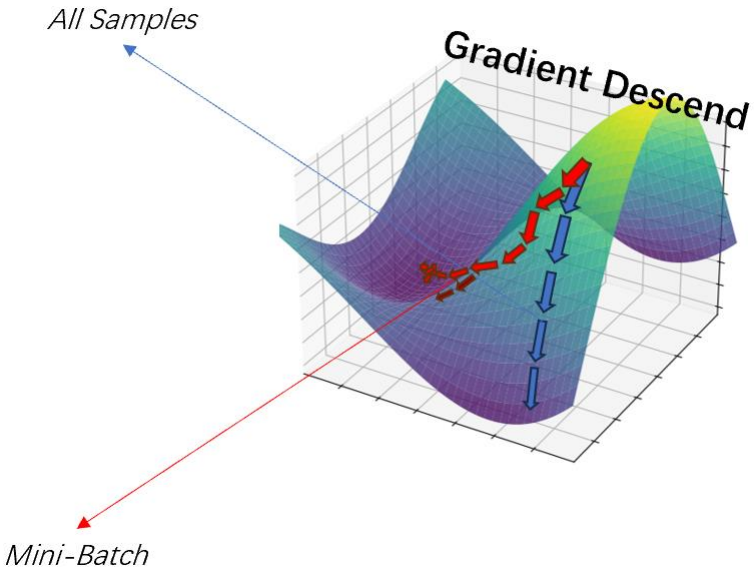


图 7

```
# Initialize the SGD model
model = SGDClassifier(loss='hinge')

# Train the model in batches
batch_size = 100
n_batches = int(np.ceil(X_train.shape[0] / batch_size))
total_start_time = time()
accuracies = []

for i in tqdm(range(n_batches), desc="Training model"):
```

```
    start = i * batch_size
    end = min((i + 1) * batch_size, X_train.shape[0])
    model.partial_fit(X_train[start:end], y_train[start:end],
classes=np.unique(y_train))
    batch_accuracy = model.score(X_train[start:end], y_train[start:end])


# # Initialize the SVM model
# model = SVC(kernel='linear')
# model.fit(X_train, y_train)

```

2. K-最近邻（K-NN）

K-NN 在分类问题中，对于一个给定的测试样本，基于欧氏距离，在训练集中找到与该样本最近的 K 个点，然后根据这 K 个点所属的类别通过投票等方式进行预测。

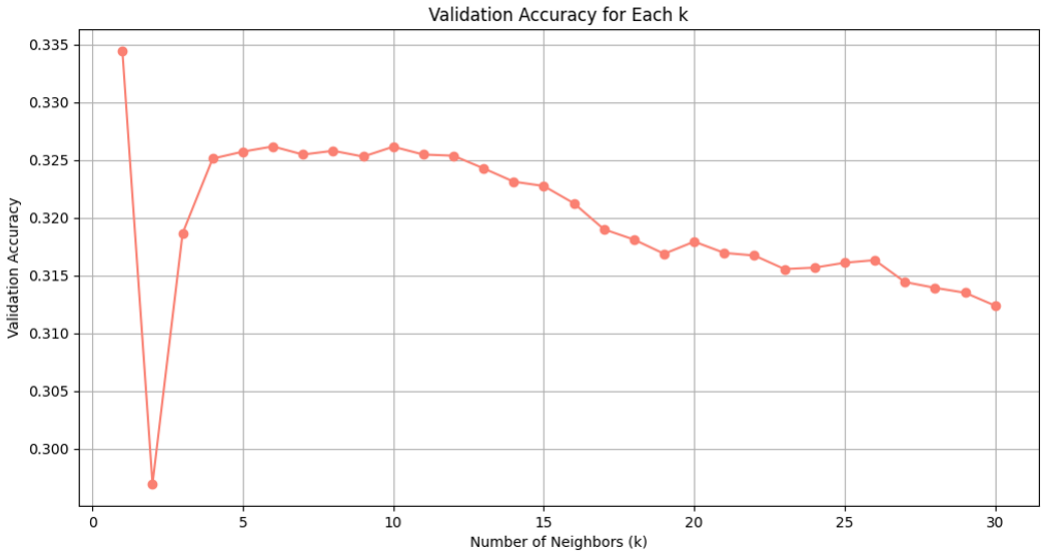


图 8

通过网格搜索，绘制 k_best 图：发现 k=1 为异常值，受验证集的划分影响大；相比 k=6，在十分类中 k=10 更加稳定，具有健壮性。测试集验证 Precision(k=1) = 41%，Precision(k=6) = 44%，Precision (k=10) = 46%，证明以上结论的合理性。

```
# Parameters for k
```

```
2. k_range = list(range(1, 31))
3. param_grid = dict(n_neighbors=k_range)
4.
5. # Initialize K-NN Classifier
6. knn = KNeighborsClassifier()
7.
8. # GridSearchCV for k_best
9. grid = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy',
return_train_score=False)
10. grid.fit(X_train, y_train)
```

3. 随机森林

随机森林是一种集成学习方法，它通过 bagging 构建 100 个决策树并将它们的预测结果进行汇总来提高分类的准确率和稳定性。如图[9]。

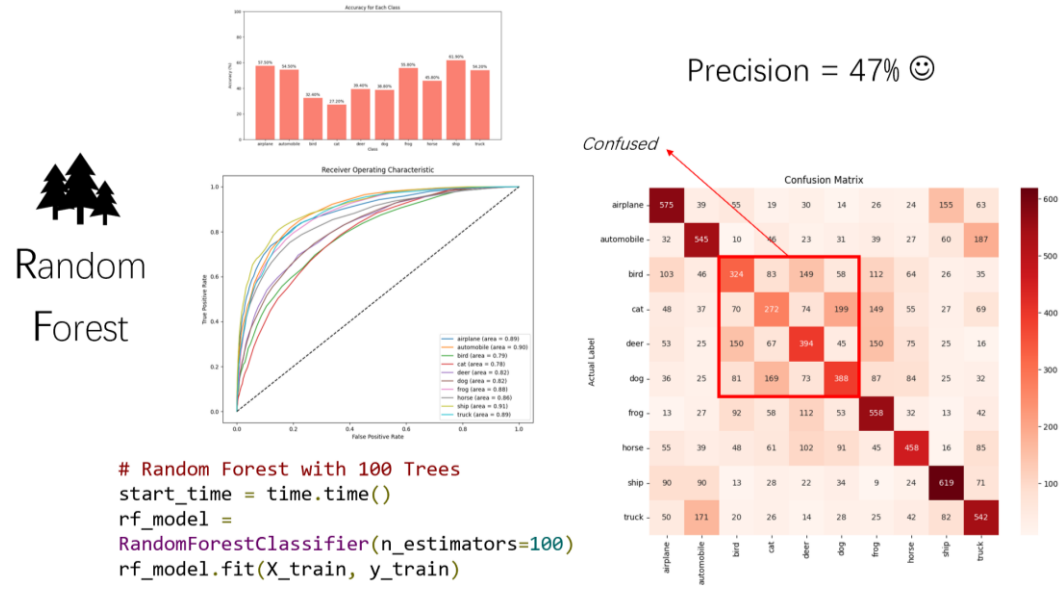


图 9

通过混淆矩阵发现，随机森林清晰地将动物与机器划分，相比其他两个机器学习方法，这是一个显著的进步。

```
1. # Train Random Forest
2. rf_model = RandomForestClassifier(n_estimators=100) # 100 trees
3. rf_model.fit(X_train, y_train)
```

5. 深度学习

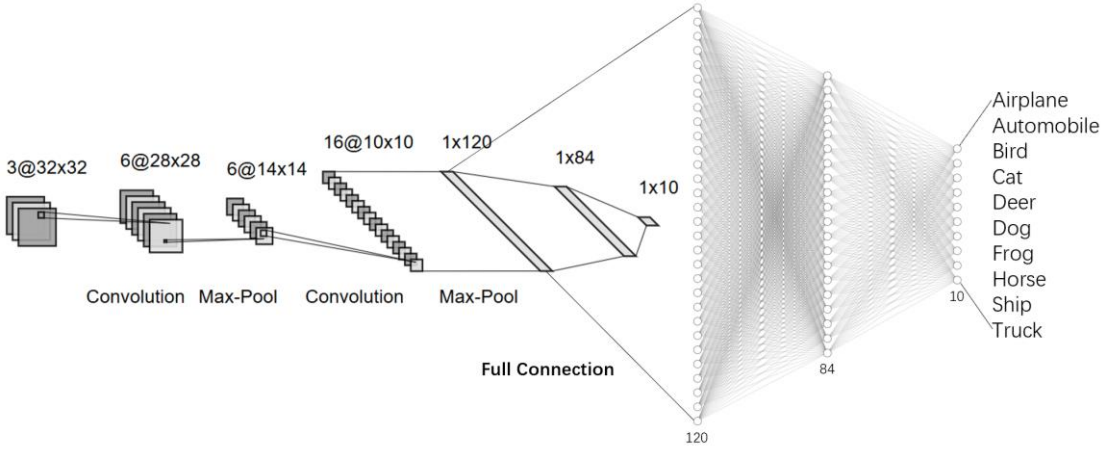
深度学习是一种机器学习技术，它通过建立和训练多层次的神经网络模型来处理和分析数据。这些模型受到人类大脑的启发，尤其是对信息处理方式的模仿。

深度学习的核心是一个称为“神经网络”的结构，它包含多层处理单元（称为神经元），每一层都能从输入数据中提取不同层次的特征。输入层接收原始数据，如图像、声音或文本，随后的隐藏层（可以是一层或多层）处理这些数据，最后输出层给出模型的判断或预测结果。

随着数据从输入层向输出层传递，每一层都会对数据进行转换和抽象，从而使模型能够学习复杂和抽象的数据表征。深度学习在许多领域中已经显示出非常好的效果，例如图像和语音识别、自然语言处理、和医学诊断等。通过大量的数据和强大的计算能力，深度学习能够实现比传统机器学习方法更精确的预测和决策。

1. EasyNet

a. EasyNet 的网络架构



b. 激活函数

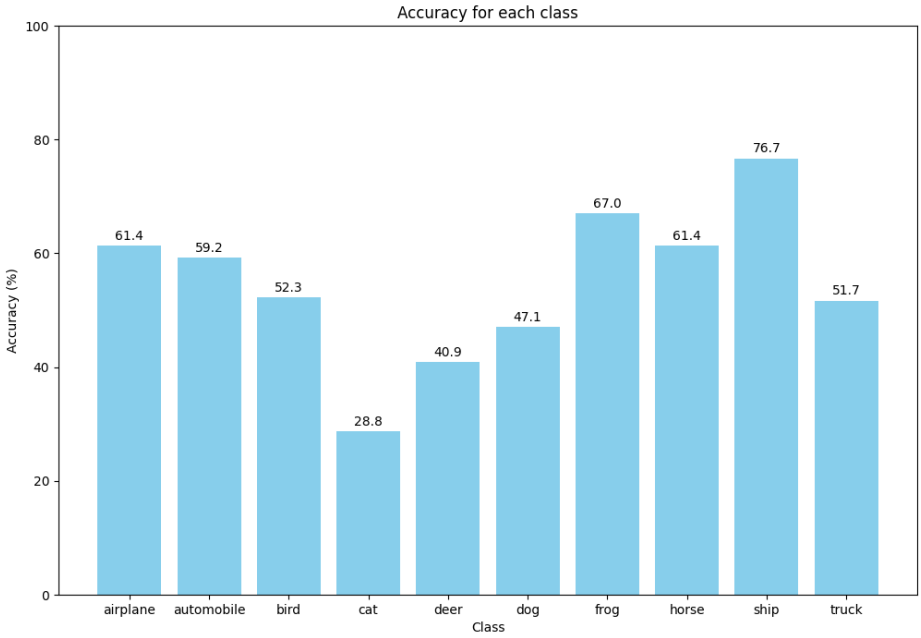
网络使用 ReLU 激活函数增加非线性，防止网络只能学习线性关系。ReLU 函数有助于加速神经网络的训练，同时减少梯度消失的问题。

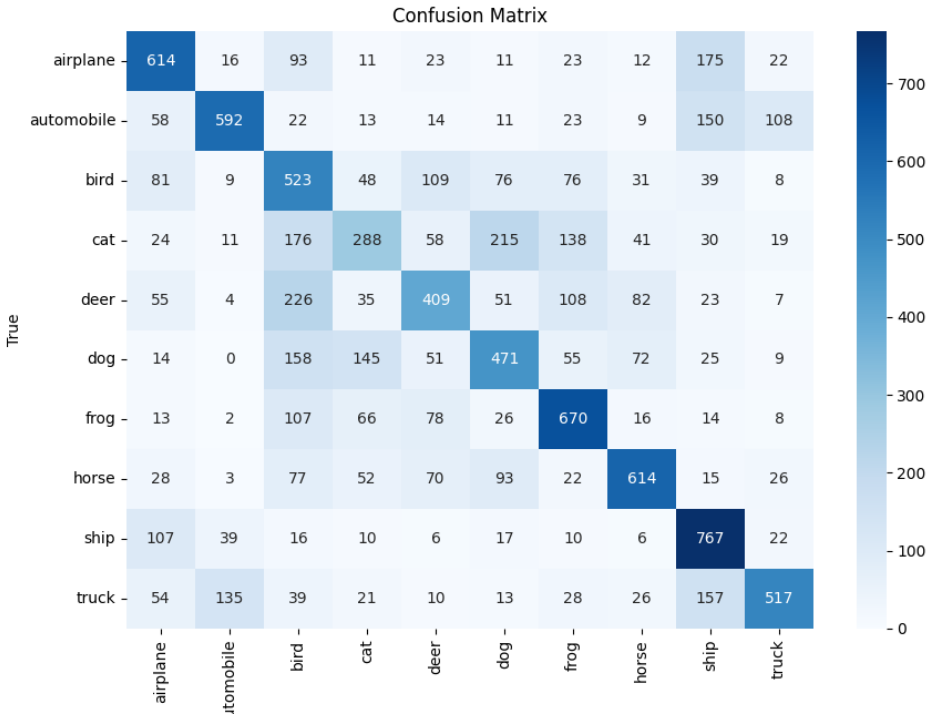
c. 特点与局限性

简单有效：这种类型的网络对于基本图像分类任务来说是简单有效的，尤其是在较小的数据集上达到了很少的训练时间。

局限性：

网络层数较少，可能不足以处理复杂的图像分类任务或大规模的数据集。由于没有提供 Batch Normalization 或 Dropout，模型面临过拟合问题。





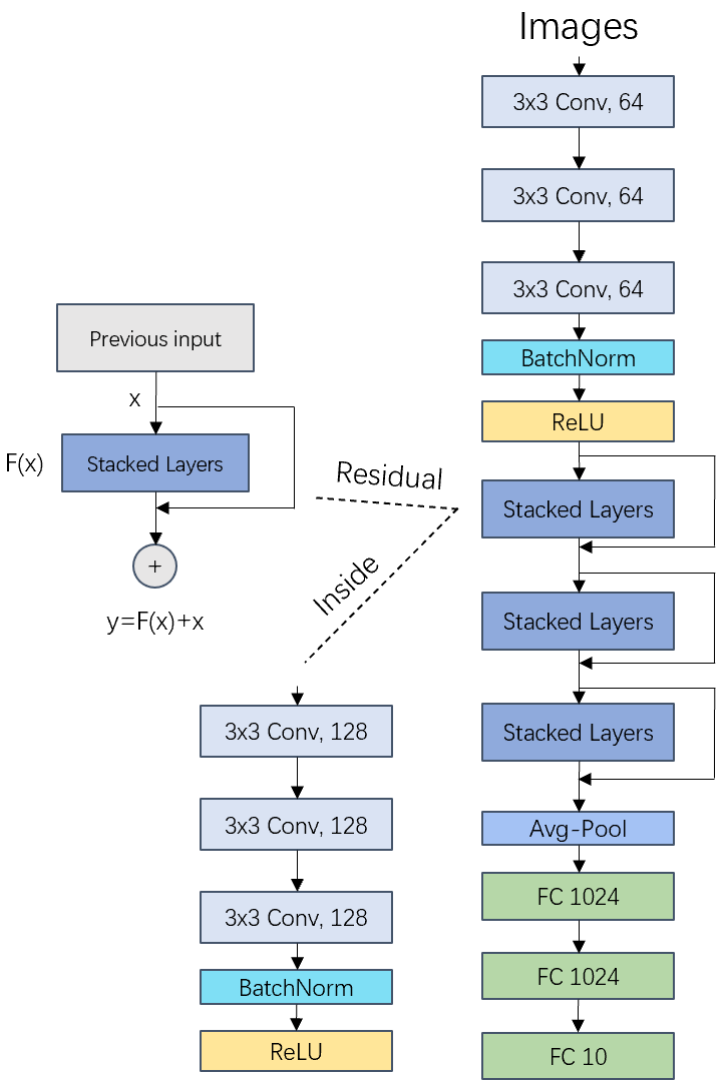
架构代码如下：

```
1. class Net(nn.Module):

2.     def __init__(self):
```

```
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

2. ResMacNet

ResMacNet 网络架构融合了传统的残差网络（ResNet）的特点[1]，并对其中的部分层进行了修改和优化，以改善性能和效率。以下是对其关键组件的分析：



基本卷积层：网络的初始部分包括三个连续的卷积层，每个层使用了 3x3 的卷积核，这有助于捕捉输入图像的低层次特征。这里没有使用步长来缩减尺寸，而是通过后续的下采样层来降低特征图的维度，保持了更多的原始信息。

残差块（ResBlock）：

下采样：每个 ResBlock 的开始通过一个 1x1 的卷积层以步长为 2 进行下采样，旨在减少数据的空间维度，同时提高特征通道数，这有助于增强模型的表达能力。

卷积层：每个残差块包括三个卷积层，前两层用于深入提取特征，而第三层则用于准备与残差连接相加的特征图。

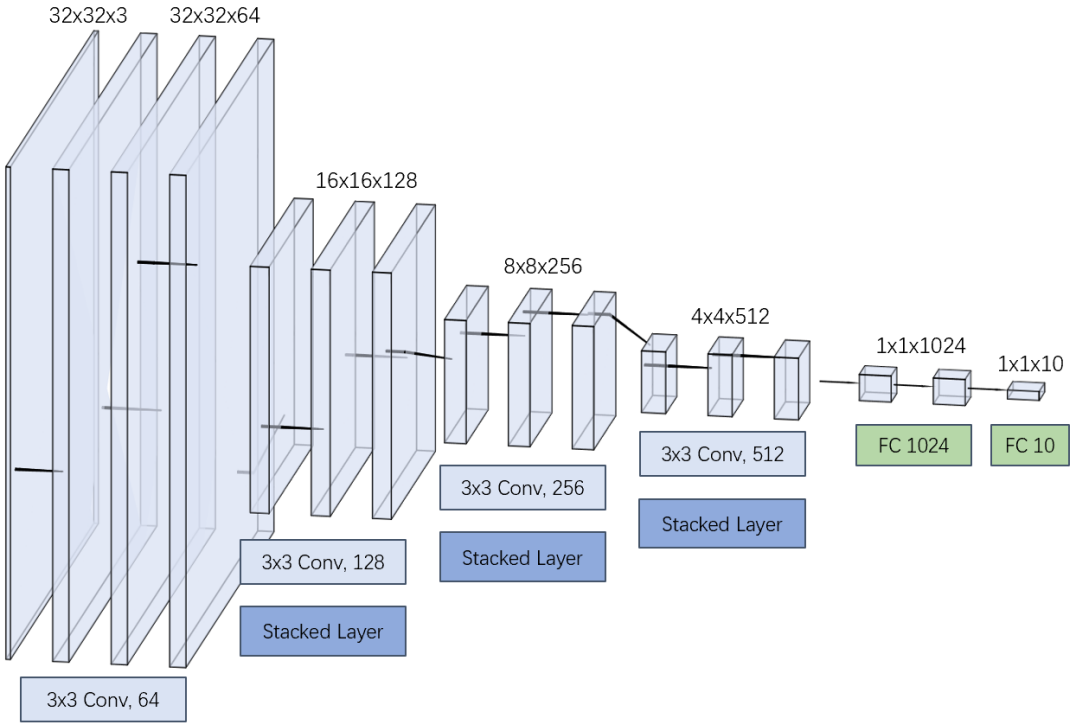
标准化和激活：每个块的输出通过批量归一化（BatchNorm）处理，然后通过 ReLU 激活函数。批量归一化有助于加快训练速度，减少模型对初始权重的敏感性。

全局平均池化层（GAP）：在网络末端使用全局平均池化替代了传统的全连接层，这有助于减少模型参数数量和计算量，同时减少过拟合的风险。GAP 通过对每个特征图进行平均池化来生成一个单一的特征值，从而保留了重要的空间特征。

全连接层：

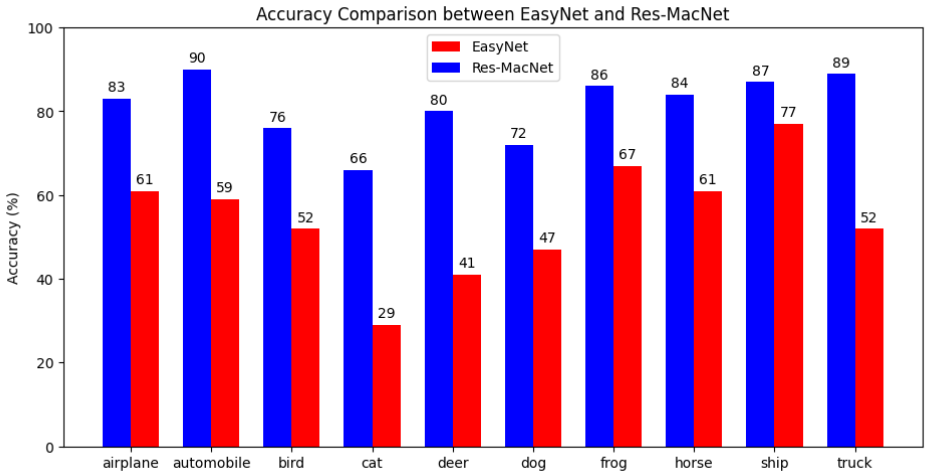
这部分网络使用了两个具有 ReLU 激活和 Dropout 正则化的全连接层，目的是进一步整合特征并防止过拟合。Dropout 通过随机“丢弃”一部分神经元的激活值来实现。

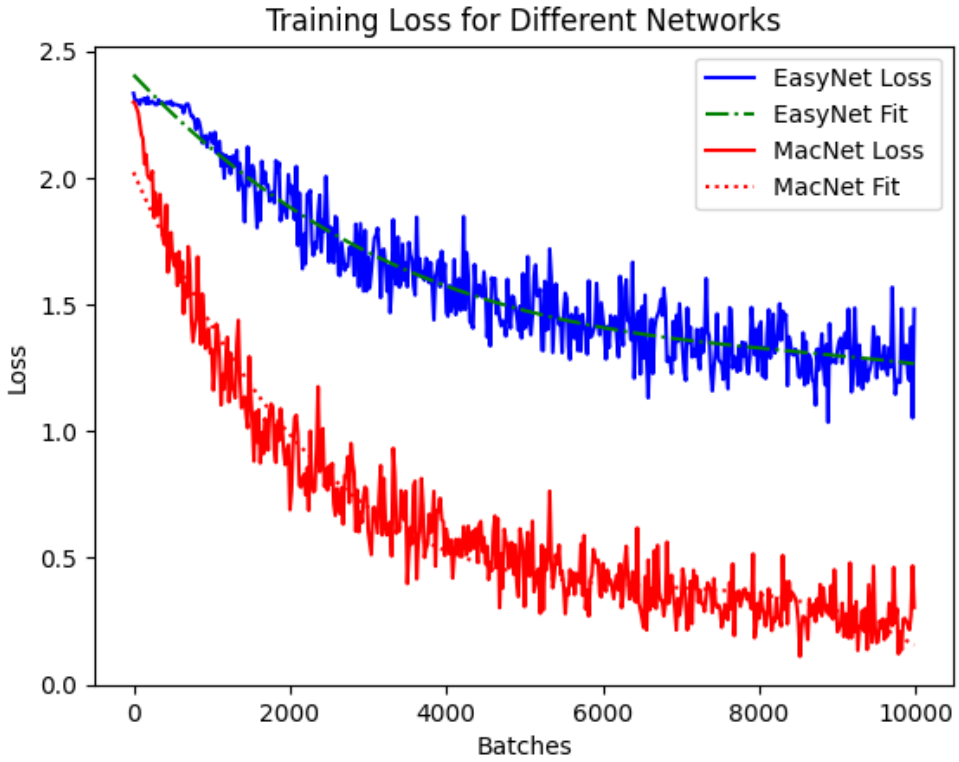
最终的全连接层将特征映射到类别数目 10，对应 CIFAR-10 的十个类别。



综上所述，ResMacNet 是一个通过整合残差连接和有效的特征抽取策略来优化性能的深度学习模型，非常适合处理复杂的图像分类任务。这种网络结构在保持计算效率的同时增加了模型的深度和复杂性，有助于学习更复

杂的特征表示。





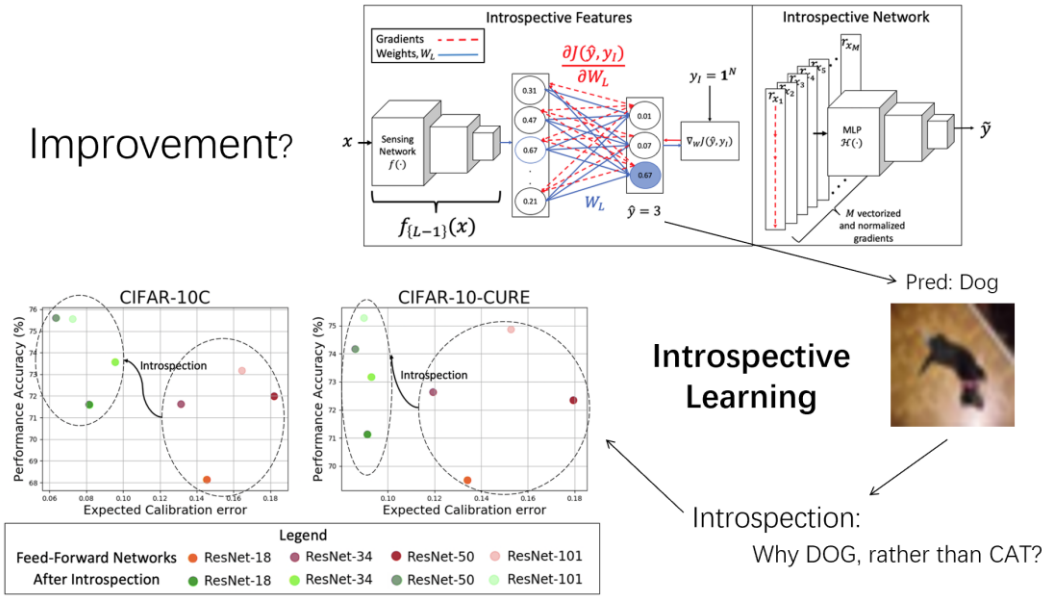
对比两个神经网络，ResMacNet 有更好的准确度和 Loss 下降。

```
1. import torch.nn as nn
2. import torch.nn.functional as F
3.
4. class ResBlock(nn.Module):
5.     def __init__(self, in_channel, out_channel):
```

```
 6.         super(ResBlock, self).__init__()
 7.         self.conv1 = nn.Conv2d(in_channel, out_channel, 3, 2, 1, bias=False)  #
stride=2 reduces size
 8.         self.conv2 = nn.Conv2d(out_channel, out_channel, 3, 1, 1, bias=False)
 9.         self.conv3 = nn.Conv2d(out_channel, out_channel, 3, 1, 1, bias=False)
10.         self.bn1 = nn.BatchNorm2d(out_channel)
11.         self.relu1 = nn.ReLU(inplace=True)
12.         self.pool = nn.Conv2d(in_channel, out_channel, 1, 2, 0, bias=False)
13.
14.     def forward(self, x):
15.         res = x
16.         res = self.pool(res)
17.         x = self.conv1(x)
18.         x = self.conv2(x)
19.         x = self.conv3(x)
20.         x = self.bn1(x)
21.         x = x + res
22.         x = self.relu1(x)
23.         return x
24.
25. class Net(nn.Module):
26.     def __init__(self):
27.         super(Net, self).__init__()
28.         self.conv1 = nn.Conv2d(3, 64, 3, 1, 1, bias=False)
29.         self.conv2 = nn.Conv2d(64, 64, 3, 1, 1, bias=False)
30.         self.conv3 = nn.Conv2d(64, 64, 3, 1, 1, bias=False)
31.         self.bn1 = nn.BatchNorm2d(64)
32.         self.relu1 = nn.ReLU(inplace=True)
33.
34.         self.block1 = ResBlock(64, 128)
35.         self.block2 = ResBlock(128, 256)
36.         self.block3 = ResBlock(256, 512)
37.
```

```
38.         self.gap = nn.AdaptiveAvgPool2d(1)
39.         self.fc1 = nn.Linear(512, 1024)
40.         self.relu2 = nn.ReLU(inplace=True)
41.         self.drop1 = nn.Dropout()
42.         self.fc2 = nn.Linear(1024, 1024)
43.         self.relu3 = nn.ReLU(inplace=True)
44.         self.drop2 = nn.Dropout()
45.         self.fc3 = nn.Linear(1024, 10)
46.
47.     def forward(self, x):
48.         x = self.conv1(x)
49.         x = self.conv2(x)
50.         x = self.conv3(x)
51.         x = self.bn1(x)
52.         x = self.relu1(x)
53.
54.         x = self.block1(x)
55.         x = self.block2(x)
56.         x = self.block3(x)
57.
58.         x = self.gap(x)
59.         x = x.view(x.size(0), -1)
60.         x = self.fc1(x)
61.         x = self.relu2(x)
62.         x = self.drop1(x)
63.         x = self.fc2(x)
64.         x = self.relu3(x)
65.         x = self.drop2(x)
66.         x = self.fc3(x)
67.
68.         return x
69.
```

6. 改进



根据[2]提出的自省是学习，可提高二次分类的准确率，以及模型的稳健性。

7. 心得体会

这个项目是我人生第一个完整的项目，结局还是令人满意的。所谓的科研应该是不断发现问题并尝试解决的循环，抑或是自己不满意并一直改进的迭代。这个项目我遇到了很多问题与不满意，好在最后都一一解决了，除了最后的自省学习还没有尝试，为下一次的学习留个空间。还有其实训练和调试只占了总时间的一半不到吧，大量时间花费在数据可视化以及项目流程可视化上了。后来反思觉得，这是必要的，毕竟咱搞数据的最终的目的还是要将结论展示给不懂数据的外行人看；况且彭老师也指出，第一次做好了，后面会一直做的好，而且更好更快。

参考文献

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[2] Prabhushankar, M., & AlRegib, G. (2022). Introspective learning: A two-stage approach for inference in neural networks. *Advances in Neural Information Processing Systems*, *35*, 12126-12140.