人工神经网络第一次大作业

实现 ConvNet

姓名： 晋一卓
学号:22336106

一．实验目的

尝试在 CIFAR-10 数据集上使用卷积网络（ConvNet）获得最佳性能。尝试调整不同参数，或添加批归一化/层归一化以及 Dropout 层（这些功能已在 annp/layers.py 中实现）以加速训练。最终，训练出一个在验证集上准确率至少达到 60%的卷积网络。最终单元格需包含模型在训练集、验证集和测试集上的准确率。

二．实验思路

可尝试的改进方向：

1. 滤波器大小：不同尺寸的滤波器可以提取不同层次的图像特征。
2. 滤波器数量：增加或减少滤波器数量是否效果更好？
3. 网络架构：通过更深的网络提升性能，在 annp/classifiers/cnn.py 文件中实现其他架构。

三．模型构建与优化过程

（一）改变网络架构：

作业中原本的 ThreeLayerConvNet 涵盖了以下流程：

卷积层→ReLU→池化→全连接→ReLU→全连接→Softmax

【在我改进构建的 DeeperConvNet 中，实现了以下优化】

1. 增加卷积层数：使用两个卷积层组成块：conv→ReLU→conv→ReLU→pool,更接近于 VGG-like 网络结构，在图像任务中表现更
2. 加入 Batch Normalization
3. 加入 Dropout

**具体实现可在 annp/classifiers/cnn.py 中查看（cnn-1.py 中保留原本 ThreeLayerConvNet 架构）*

（二）训练模型一：基准模型

在基准模型中，我使用自定义的 DeeperConvNet，具体参数设置如下（接下来的模型均在此基础上进行优化）：

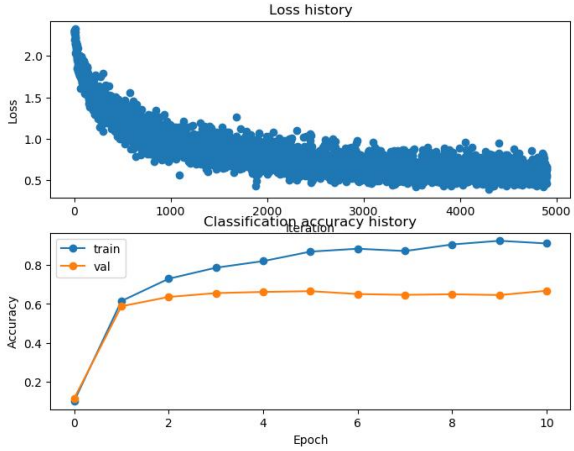
```
#模型1.基准模型
from annp.classifiers.cnn import DeeperConvNet
model = DeeperConvNet(
    input_dim=(3, 32, 32),
    num_filters=(32, 64),
    filter_size=3,
    hidden_dim=100,
    num_classes=10,
    weight_scale=1e-3,
    reg=0.001,
    use_batchnorm=False,
    use_dropout=False,
    dropout_keep_ratio=0.5,
    dtype=np.float32
)
```

```
from annp.solver import Solver
solver = Solver(model, data,
                num_epochs=10,
                batch_size=100,
                update_rule='adam',
                optim_config={'learning_rate': 1e-3},
                verbose=True,
                print_every=100)
```

【参数设置】
• num_filters：两个卷积层，分别使用 32 和 64 个卷积核
• filter_size：所有卷积层均使用 3*3 核
• hidden_dim：全连接层隐藏维度设为 100
• weight_scale：权重初始化标准差设置为 1e-3(较小，但作为基准模型较为稳定)
• use_batchnorm：初始设为 false,后续将引入并进行训练
• use_dropout：初始设为 false,后续将引入并进行训练

【优化器配置】
• update_rule：使用 Adam 优化器
• learning_rate：初始学习率设置为 1e-3
• batch_size：每批训练样本数设置为 100，适中
• num_epochs：训练总轮数设为 10

【可视化结果】



【Test accuracy】 0.6320

【模型表现】
基准模型的学习结果表现出：
在改变网络架构后，模型表现效果较好，已经满足作业要求。
1. loss 持续减少，train/val accuracy 随迭代次数提升，在 epoch1 表现最为明显
2. Train acc 收敛于>0.9，val acc 收敛于 0.6 左右

【模型改进思路】
同时引入 bn 与 dropout,并对参数进行适当微调

（三）训练模型二：引入 bn 与 dropout

```python
#模型2：加入batchnorm与dropout
model = DeeperConvNet(
    input_dim=(3, 32, 32),
    num_filters=(32, 64),
    filter_size=3,
    hidden_dim=128,
    num_classes=10,
    weight_scale=1e-3,
    reg=0.001,
    use_batchnorm=True,
    use_dropout=True,
    dropout_keep_ratio=0.5,
    dtype=np.float32
)
```

```python
solver = Solver(model, data,
                num_epochs=10,
                batch_size=100,
                update_rule='adam',
                optim_config={'learning_rate': 1e-3},
                verbose=True,
                print_every=100)
```

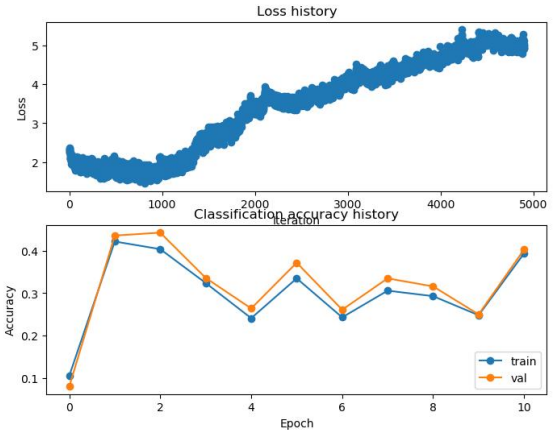
【参数设置】

- hidden_dim：全连接层隐藏维度设为 128
- use_batchnorm：设为 true
- use_dropout：设为 true

【优化器配置】

保持不变

【可视化结果】



【Test accuracy】0.1380

【模型表现】

模型学习结果表现出：

同时引入 bn 与 dropout 效果很差，loss 持续提升，train/val accuracy 在 epoch1 达到最高（不超过 0.5）,后持续波动，在 epoch4，6 达到 acc 最低值。

学习结果说明：bn 与 dropout 同时使用存在冲突风险：

1.bn 在每一个 mini-batch 的激活做归一化，依赖于真实的激活值分布；

2.dropout 会随机屏蔽神经元，导致 bn 接收到的激活分布不稳定，不可靠；

3.训练初期，bn 的统计值不稳定，dropout 的加入进一步破坏了训练信号，容易造成训练震荡（显示在我的学习模型中）或停滞。

【模型改进思路】

在小模型中，dropout 会降低模型表示能力，我设置的初始化/正则/学习率组合也不太合理，后将扩大规模，改变参数进行重试。

（四）训练模型三：采用更大规模

```
#模型3：加入batchnorm与dropout+更大规模的模型
model = DeeperConvNet(
    input_dim=(3, 32, 32),
    num_filters=(64, 128),
    filter_size=3,
    hidden_dim=512,
    num_classes=10,
    weight_scale=1e-2,
    reg=5e-4,
    use_batchnorm=True,
    use_dropout=True,
    dropout_keep_ratio=0.8,
    dtype=np.float32
)
```

```
solver = Solver(model, data,
                num_epochs=15,
                batch_size=100,
                update_rule='adam',
                optim_config={'learning_rate': 1e-3},
                verbose=True,
                print_every=100)
```

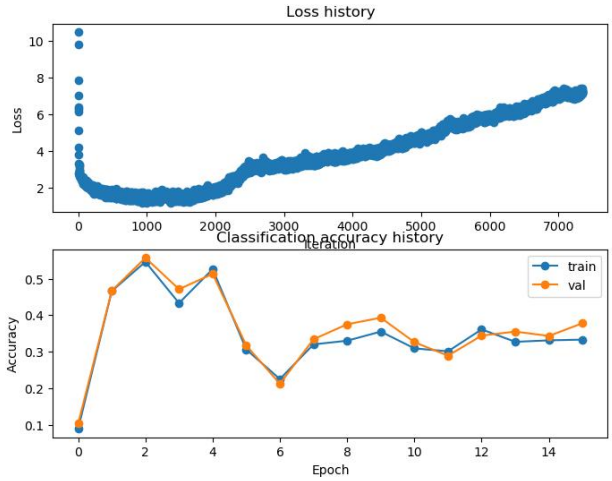
【参数设置】

- hidden_dim：全连接层隐藏维度设为 512
- num_filters：两个卷积层，分别使用 64 和 128 个卷积核
- use_batchnorm：设为 true
- use_dropout：设为 true
- weight_scale：权重初始化标准差设置为 1e-2

【优化器配置】

- num_epochs：增加迭代次数，设为 15

【可视化结果】



【Test accuracy】0.1280

【模型表现】

模型学习结果表现出：loss 持续提升，train/val accuracy 在 epoch2 达到最大（0.6 左右）,并在 epoch6 达到最低，后在 epoch8-15 相对稳定于 0.35。

说明同时引入 bn 和 dropout 的方案不可行，增大规模与迭代数也无法消解两者的冲突。

【模型改进思路】

学习结果说明：小模型同时引入 bn 与 dropout 不合理，故尝试只使用 bn,禁用 dropout，同时增大 hidden_dim，修改其他参数为适合 bn 的初始化数值。

（五）训练模型四：使用 bn，禁用 dropout,同时增大 hidden

```
#模型4：仅使用batchnorm,禁用dropout,增大hidden
model = DeeperConvNet(
    input_dim=(3, 32, 32),
    num_filters=(32, 64),
    filter_size=3,
    hidden_dim=128,
    num_classes=10,
    weight_scale=1e-2,
    reg=1e-4,
    use_batchnorm=True,
    use_dropout=False,
    dtype=np.float32
)
```

```
solver = Solver(
    model,
    data,
    num_epochs=20,
    batch_size=100,
    update_rule='adam',
    optim_config={'learning_rate': 1e-3},
    print_every=100,
    verbose=True
)
```

【参数配置】

- hidden_dim：全连接层隐藏维度设为 128
- num_filters：两个卷积层，分别使用 32 和 64 个卷积核
- use_batchnorm：设为 true
- use_dropout：设为 false
- weight_scale：权重初始化标准差设置为 1e-2

【优化器配置】

- num_epochs：增加迭代次数，设为 20

由于主机异常关闭，无法展示该模型的可视化结果，故对训练日志进行特征总结：

【训练日志总结】

1. 初始阶段（epoch0-2）：初始 loss 较高，对应 val acc 仅为 12.3%，模型很快在 epoch1 收敛到 train acc53.8%,训练集准确率达到 56。0%；到 epoch2,val acc 达到 58.3%，随后停滞；
2. 中期阶段（epoch3-5）：准确率持续小幅度上涨，在 epoch4 达到最高：训练集 62.2%，验证集 59.7%。从 epoch5 开始出现严重的过拟合现象，训练准确率持续下降，验证集明显回落（52.3%）
3. 后期阶段（epoch6-20）多个 epoch 的训练准确率持续下降，训练崩溃，损失值出现剧烈震荡，后几轮稍有回升但未达到最佳状态。

【Test accuracy】0.1070

【模型表现】

对模型学习结果进行分析：

在 epoch0-4 val acc 持续上升，且在 epoch4 达到了最佳值；但从 epoch5 开始，性能下降，train/val acc 同步退步，长期在 50%到 55%之间震荡。

这说明模型前期收敛良好，train acc 达到 60%以上，但在中后期，由于 epoch4 就逼近最优，此时继续使用同样的学习率进行学习，参数更新过大就导致偏离好的解，过拟合，性能也就退化。

【模型改进思路】

迭代 20 次不仅没有得到更好的效果，反而更差，浪费时间资源，所以我将添加 early stopping 与保存最佳模型的功能，降低初始学习率，添加上学习率衰减的策略。

（六）训练模型五：添加 early stopping+动态学习率衰减的更优模型

```
model = DeeperConvNet(
    input_dim=(3, 32, 32),
    num_filters=(32, 64),
    filter_size=3,
    hidden_dim=128,
    num_classes=10,
    weight_scale=1e-2,
    reg=1e-4,
    use_batchnorm=True,
    use_dropout=False,
    dtype=np.float32
```

```
solver = Solver(
    model,
    data,
    num_epochs=1,
    batch_size=100,
    update_rule='adam',
    optim_config={'learning_rate': 5e-4},
    print_every=100,
    verbose=False
)
```

【参数设置】

- hidden_dim 设为 128
- weight_scale 设为 1e-2，采用更大的初始权重
- reg=1e-4:减弱正则化，降低过拟合风险
- use_batchnorm：设置为 true
- use_dropout：设置为 false
- learning_rate；设为 5e-4，采用更小的学习率+bn 策略，确保训练更稳定

【优化器配置】

- num_epochs:设为最多 20 轮，由于添加 early stopping，可提前终止。

【功能优化】

1. early stopping 策略：设置 patience=5，即若验证集连续 5 个 epoch 未提升就提前终止

```
if val_acc > best_val_acc:
    best_val_acc = val_acc
    best_params = {k: v.copy() for k, v in model.params.items()}
    no_improve_cnt = 0
```

2. 动态学习率衰减：每 5 个 epoch 学习率将减半，解决后期 loss 波动过大的问题

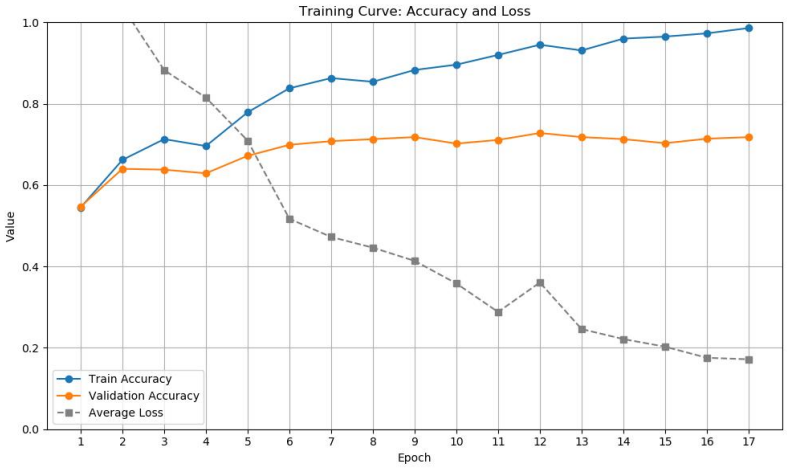
```
if (epoch + 1) % 5 == 0:
    for k in solver.optim_configs:
        solver.optim_configs[k]['learning_rate'] *= 0.5
    new_lr = solver.optim_configs[k]['learning_rate']
```

**由于模型训练结束后异常退出，无法计算 test_accuracy;*

**我将手动提取出每个 epoch 的 loss,train acc 与 val acc，并进行可视化。可视化结果如下，虽然没有更为精准的结果，但是能够正确反映该模型的学习效果更为良好。*

【可视化结果】

```
epochs = list(range(1, 18))
avg_loss = [1.2817, 1.0384, 0.8827, 0.8150, 0.7093, 0.5172, 0.4725, 0.4462, 0.4137,
            0.3582, 0.2882, 0.3606, 0.2461, 0.2215, 0.2028, 0.1756, 0.1718]
train_acc = [0.5450, 0.6620, 0.7130, 0.6960, 0.7790, 0.8380, 0.8630, 0.8540, 0.8830,
             0.8960, 0.9200, 0.9450, 0.9310, 0.9600, 0.9650, 0.9730, 0.9860]
val_acc = [0.5470, 0.6400, 0.6380, 0.6290, 0.6720, 0.6990, 0.7080, 0.7130, 0.7180,
           0.7020, 0.7110, 0.7280, 0.7180, 0.7130, 0.7030, 0.7140, 0.7180]
```



【模型表现】
模型学习结果表现出：
1. epoch 1-4：train/val acc 都有所升高，且 loss 平稳下降;
2. epoch 5-12：train/val acc 稳定提升，同时 val acc 在 epoch12 达到最佳 0.728；
3. epoch 12-17:train acc 持续提升，最终达到 0.986，val acc 稳定在 0.71 附近震荡。
4. 对 loss 进行分析：初始 loss 在 2.3 左右，整体 loss 随着 epoch 增加显著下降，且 loss 与 train acc 增长趋势相匹配；
结果说明：模型对训练集表现越来越好，但是验证集表现停滞，可能出现轻微过拟合迹象。

【模型改进思路】
在后续的模型优化中，将重点考虑 val acc：使用 bn,提高感受野，稍微增强正则化项。同时为避免程序异常退出导致的数据缺失问题，将加入 checkpoint 选项已保存训练数据。

四．最终模型展示

```
model = DeeperConvNet(
    input_dim=(3, 32, 32),
    num_filters=(32, 64),
    filter_size=3,
    hidden_dim=256,              # 增大隐藏层容量
    num_classes=10,
    weight_scale=1e-2,
    reg=5e-4,                    # 增强正则化
    use_batchnorm=True,
    use_dropout=False,
    dtype=np.float32
)
```

```
solver = Solver(
    model,
    data,
    num_epochs=1,
    batch_size=100,
    update_rule='adam',
    optim_config={'learning_rate': 5e-4},
    print_every=100,
    verbose=False
)
```

【参数设置】
- hidden_dim 设置为 256，增加判别能力
- reg 设置为 5e-4 加强正则，缓解过拟合
- learning rate 设置为 5e-4 稳定训练

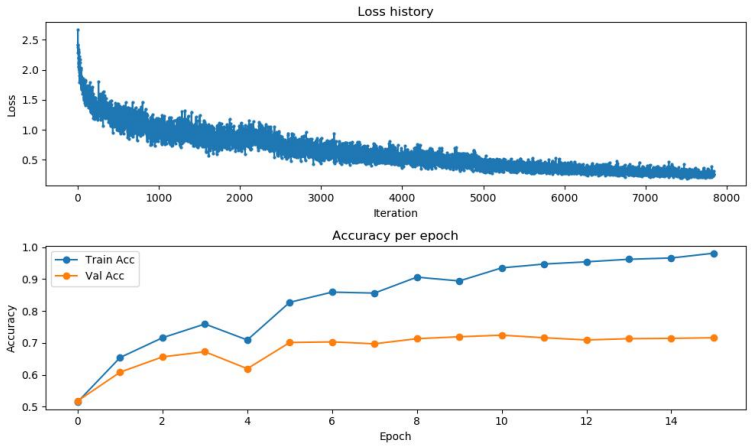
【优化器配置】
最大可迭代次数设置为 30 次，可提前结束

【功能优化】

1. 启用 early stopping
2. 启用学习率衰减，每五个 epoch 学习率进行减半
3. 启用 checkpoint，在训练过程中保存 loss/acc，防止程序异常导致信息丢失

```
checkpoint_data = {
    'epoch': epoch + 1,
    'train_acc_history': train_acc_history,
    'val_acc_history': val_acc_history,
    'loss_history': loss_history,
    'best_val_acc': best_val_acc,
    'best_params': best_params
}
with open(ckpt_path, 'wb') as f:
    pickle.dump(checkpoint_data, f)
    print("Checkpoint saved.")
```

【可视化结果】



【Test accuracy】0.6920

【模型表现】

模型学习结果表现出：

1. loss 曲线整体下降平稳，趋势一致，波动收敛良好：在从约 iteration 0 到 3000，loss 下降明显，iteration 3000 后进入缓慢下降阶段，最终去尽在 0.3-0.4 左右，loss 虽有抖动，但是呈现出指数衰减趋势，训练稳定。
2. 训练集训练集准确率经过 10 个 epoch 快速增长至 95%以上，最终逼近 0.98，与 loss 降低的趋势高度一致，模型对训练集表现出极强拟合能力。
3. 训练集准确率在 epoch 0-6 迅速增长至 71%左右，之后从 epoch 7-15 趋于稳定，存在轻微震荡，范围在 0.71-0.725 之间波动。与 train acc 持续上升存在偏差

【总结】

与前一个模型相比，该模型在结构和性能上差异不大，但验证准确率略有下降（约 0.05 左右），说明*当前的参数调整对性能提升效果有限*。然而，该模型在测试集上依然能够达到超过 70% 的准确率，具备较强的实际表现能力。

此外，该模型在训练过程中*引入了早停（Early Stopping）和模型检查点（Checkpoint）机制，不仅提高了训练的鲁棒性，也避降低过拟合带来的负面影响*。从可视化结果来看，损失函数和准确率变化曲线清晰完整，训练过程稳定，模型调试过程具有良好的可解释性。

综上考虑，尽管验证性能略有波动，但综合训练效果、稳定性与功能完整性，该模型被选定为最终使用的模型。

五．未来优化方向

若不考虑实验架构的限制，后期可以对最终模型进行以下方面的优化：

1. 启用 GPU 进行加速，提升速度与试验效率，将当前框架迁移至支持 CUDA 的框架；
2. 加深网络结构，增加卷积层数，采用更深的 ConvNet 架构（如 VGG）；
3. 引入残差连接，提高稳定性（参考 ResNet）；
4. 使用 L2 正则化。GroupNorm 等方法尝试提升稳定性避免过拟合；
5. 优化训练策略，改进当前学习率衰减策略，使用更加先进的优化器等。