# Project 2 Report

**赵心怡 19307110452**

**2022.5.3**

## 1. Train a Network on CIFAR-10

CIFAR-10 是用于视觉识别任务的广泛使用的数据集。 CIFAR-10 数据集（加拿大高级研究所）是一组通常用于训练机器学习和计算机视觉算法的图像。它是机器学习研究中使用最广泛的数据集之一。 CIFAR-10 数据集包含 10 个不同类别的 60,000 张 32 × 32 彩色图像。这 10 个不同的类别分别代表飞机、汽车、鸟类、猫、鹿、狗、青蛙、马、轮船和卡车。每个类别有 6,000 张图像。由于 CIFAR-10 中的图像是低分辨率的（32 × 32），这个数据集可以让我们快速尝试我们的模型，看它是否有效。

### 1.1 Basic Network

我们先设了一个基础的双层神经网络，设置epoch为50，优化器为SGD, loss function 为nll loss，神经网络的结构如下：

| **Model** | **total parameter** | **optimizer** | **loss function** | **L2-regularization** | **batch norm** |
|---|---|---|---|---|---|
| MyNet | 1757258 | SGD | nll | no | yes |
| **hidden size** | **kernel** | **activation** | **epoch** | **dropout rate** | **res connect** |
| 100 | 5*5+5*5 | ReLU | 50 | 0.3 | no |

因为batch_size越小需要训练的时间越长，我们在下面的实验中选取batch_size=64。

### 1.2 Batch normalization

首先加入batch norm比较加入后的结果：

| **batch norm** | **speed** | **accuracy** |
|---|---|---|
| no | 493.27s | 0.716 |
| *add batch norm* | *496.68s* | *0.718* |

发现加了batch norm以后的时间增加了但是准确率有一定提升。

### 1.3 Dropout rate

然后我们设置dropout rate: 比较了p=0.1, 0.3的情况，我们尝试在每个卷积层后都dropout一次

| dropout | speed | accuracy |
| --- | --- | --- |
| 0 | 496.68s | 0.718 |
| 0.5 | 506.95s | 0.7597 |
| 0.3 | *493.46s* | *0.7737* |

最终选取了dropout rate=0.3作为模型的最优dropout。

## 1.4 Loss function

比较不同的loss function:

| loss | speed | accuracy |
| --- | --- | --- |
| *nll* | *493.46s* | ***0.7737*** |
| crossEntropy | 490.29s | 0.7571 |

发现选取nll的loss function时准确率更高。

## 1.5 Optimizer

比较不同的optimizer，初始学习率都是0.1，结果如下. 有几个优化器没能收敛，猜测是由于对初始学习率的敏感性。

| optimizer | speed | accuracy |
| --- | --- | --- |
| *SGD*+momentum(default) | *493.46s* | *0.7737* |
| Adam | 498.81s | 0.7077 |
| RmsProp | 517.69s | 0.6341 |

最终选取SGD+momentum优化器。

## 1.6 Activation

尝试不同的激活函数：

| activation | speed | accuracy |
| --- | --- | --- |
| ReLU | *493.46s* | *0.7737* |
| tanh | 514.81s | 0.5768 |

结果显示ReLU激活函数对我们的模型效果较好。

## 1.6 Nueron & Filter

尝试不同数量的神经元/过滤器：

| neuron kernel | speed | accuracy |
| --- | --- | --- |
| 5*5+5*5 | *493.46s* | *0.7737* |
| 5*5+3*3 | 498.58s | 0.757 |
| 3*3+3*3 | 490.75s | 0.7436 |

## 1.7 Hidden layer

尝试修改hidden size大小:

| hidden size | speed | accuracy |
| --- | --- | --- |
| 100 | *485.98s* | *0.7847* |
| 300 | 492.81s | 0.7506 |

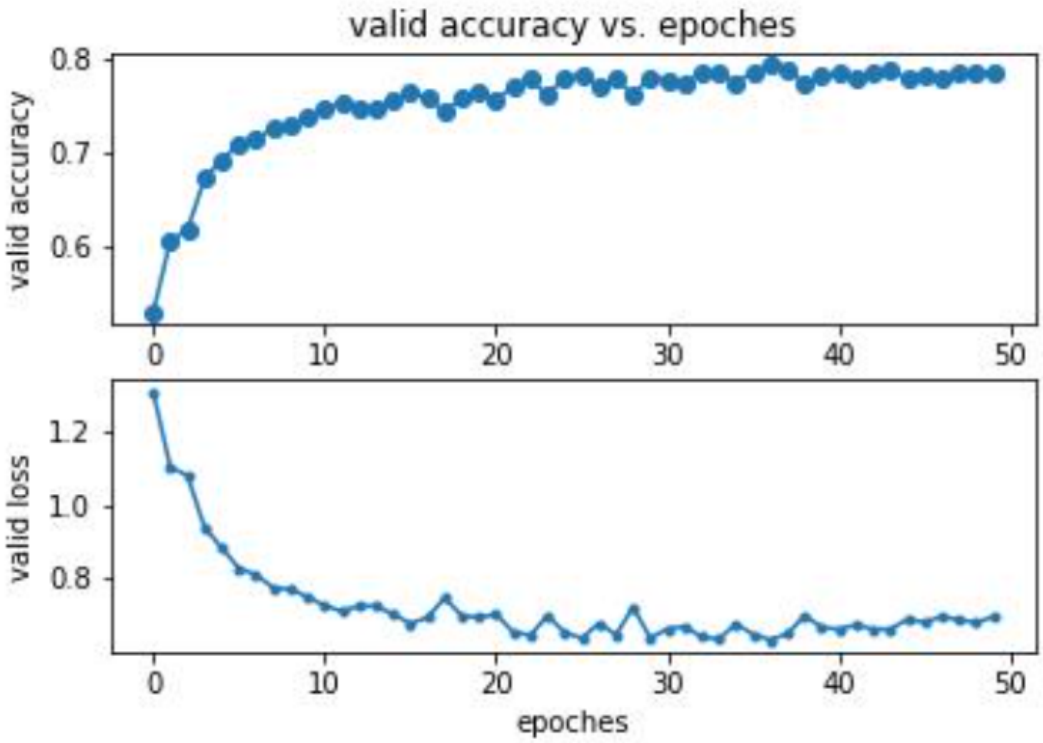
发现hidden size增加的同时准确率没有明显提升

## 1.8Best Model

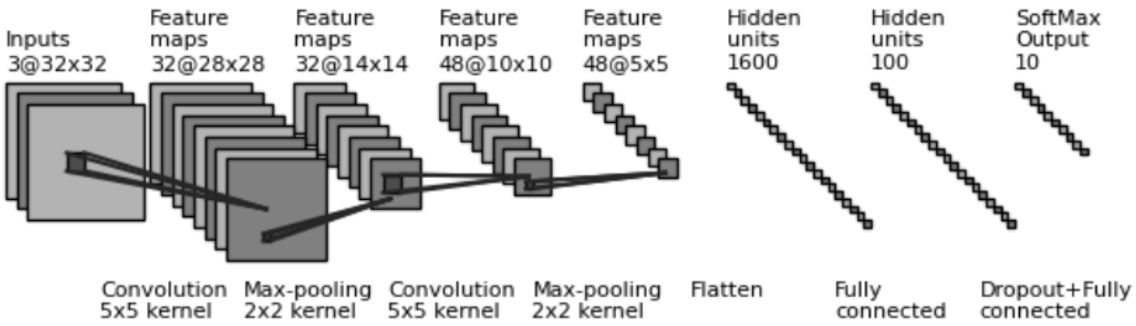
最终我们的模型达到的最优结果准确率是0.7847

```
iteration: 0;    Loss: 0.8122363686561584
...
iteration: 0;    Loss: 0.4479424059391022
valid loss:0.6731930687427521, Accuracy:0.7807
Accuracy on test set:0.7847
Finish! run time: 485.9873 s
```

最优的模型的accuracy随时间的变化:



我们的基础模型结构因为比较简单，所以可以用draw_convnet工具包画卷积图， 结果如下：




## 1.9 complex model:

基于以上的基础模型，我根据ResNet9的结构，重新写了一个改进版MyResNet9，对神经网络进行进一步优化。将神经网络扩充到了九层卷积。我们的模型与最初版本大体一致，在卷积层部分我修改了卷积核大小，将原来的kernel-size=5改成3，padding改成1，stride改成1。

我们设的Batch-size=64， epoch=50.

同时我们添加了data sugmentation, 修改优化器为SGD优化器，增加weight_decay=1e-4来进行L2-regularization。

改变不同的激活函数和损失函数的影响和上面简单的网络结果类似，没有对模型起到明显的优化。

数据增强包括了截取图像，随机的翻转核标准化等，增加数据增强显著增强了模型的泛化性能。

| **Data augmentation** | **Accuracy** |
| --- | --- |
| no augmentation (default) | 0.7907 |
| RandomCrop+HorizontalFlip | **0.845** |

我们增加了l2-regularization设置weight_decay为1e-4：

| **L2-regularization** | **accuracy** |
| --- | --- |
| no | 0.845 |
| 4e-4 | **0.8675** |

修改momentum的大小：

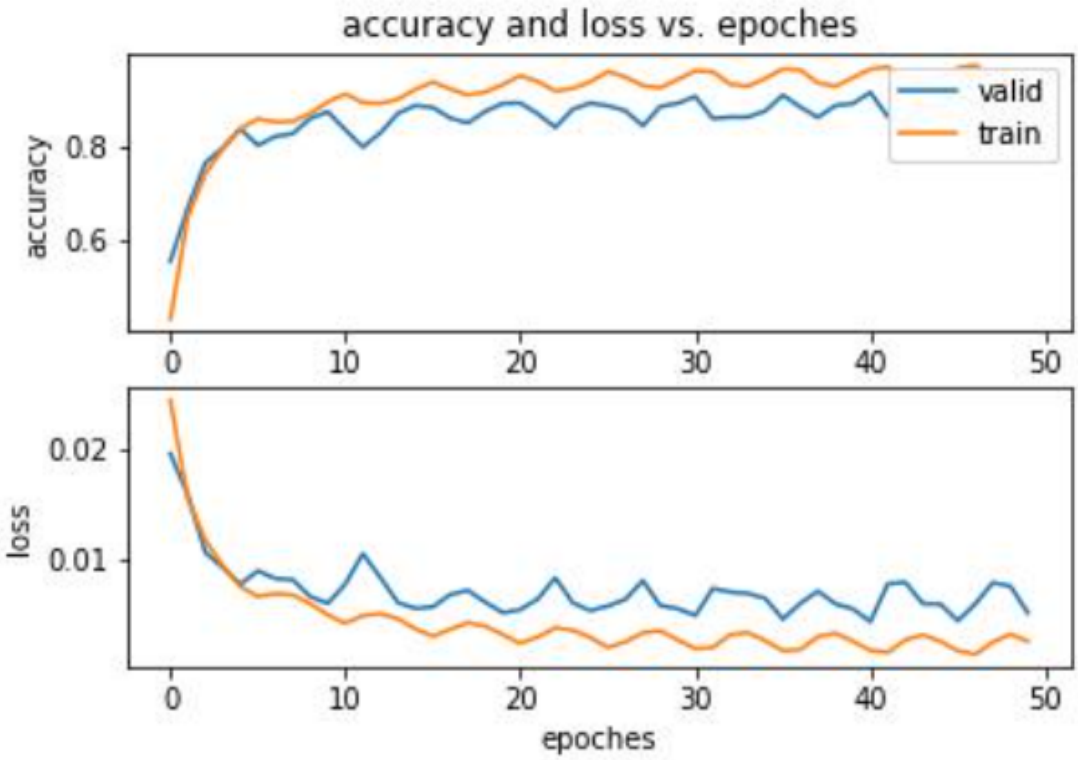
| **Momentum** | **Accuracy** |
| --- | --- |
| 0.5 | 0.8675 |
| 0.9 | **0.8749** |

我们设定了一个新的学习率调整策略，根据循环学习率策略 (CLR) 设置每个参数组的学习率。

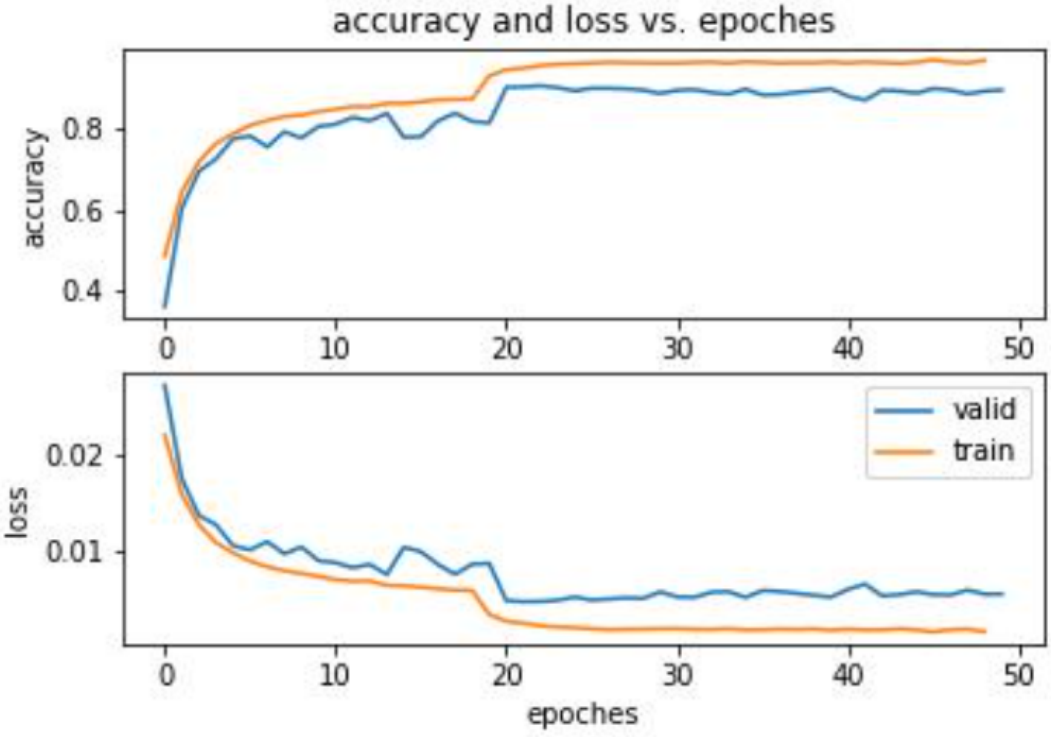
该策略以恒定频率循环两个边界之间的学习率。两个边界之间的距离可以在每次迭代或每个周期的基础上进行缩放。 CyclicLR策略在每批之后改变学习率。

| Learning rate | Accuracy |
| --- | --- |
| 0.01 | 0.8749 |
| 0.04-0.01 CyclicLR | **0.899** |
| 0.04-0.01 Linear | 0.882 |

画图发现cyclicLR的accuracy和loss曲线也呈现周期性波动，准确率的波动幅度较大。




相比较前20个epoch0.04后面改成0.01的learning rate,的准确率曲线发现准确率的提升较平缓，但是在20epoch时候会有一个准确率陡增，猜测是因为新的learning rate让模型收敛到了更好的位置。



论文中的研究表明修改某一块卷积核大小对整体的影响并不大，在此尝试微调修改模型的卷积形式

| **Network** | **Accuracy** |
| --- | --- |
| 5*5, stride=2, padding=2 | 0.899 |
| 3*3, stride=1, padding=1 | **0.9076** |

进一步修改相关的参数，因为发现仍然有过拟合现象，因此适当增加了L2-regularization。

最终的模型可以较好地解决过拟合问题，在50个epoch下的最优准确率达到91.58%。

```
iteration: 0;    Loss: 0.0100116478279232 98
valid loss:0.004894815228506923, Accuracy:0.9131
Accuracy on test set:0.9158
Finish! run time: 1128.697 s
```

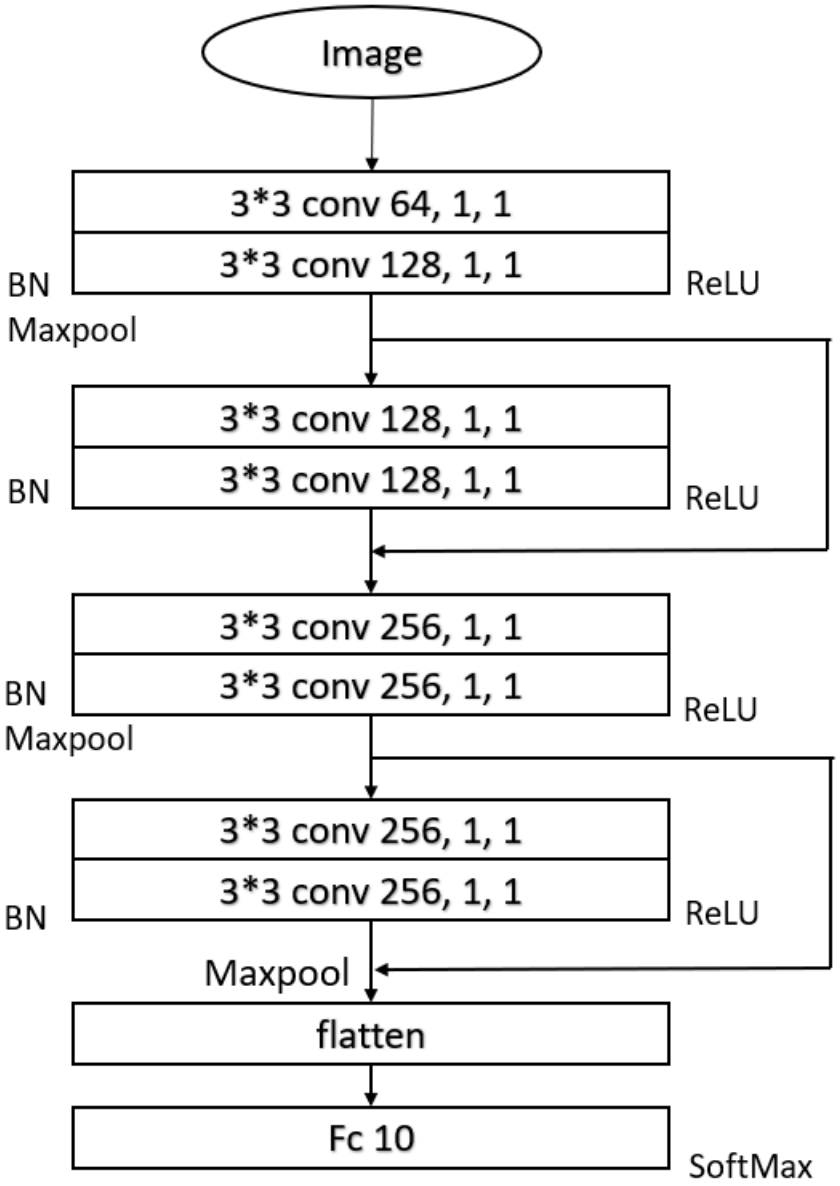
最终的模型参数：

| **Optimizer** | **L2-regularization** | **loss function** | **Momentum** |
| --- | --- | --- | --- |
| SGD | 4e-4 | crossEntropy | 0.9 |
| **Activation** | **Batch size** | **learning rate** | **Batch Norm** |
| ReLU | 50 | 0.04-0.01dynamic | yes |

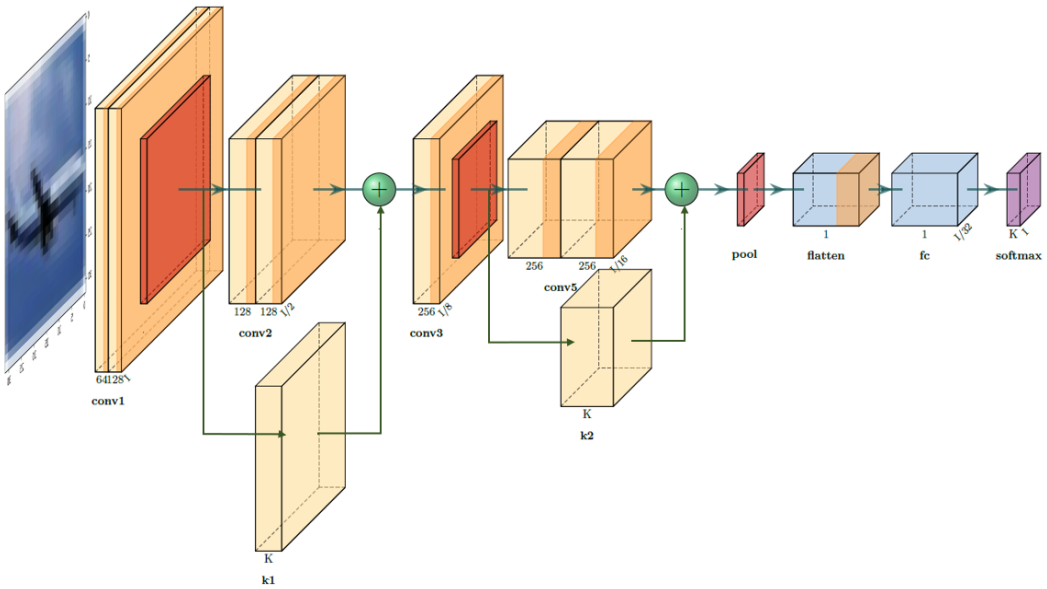
模型的参数个数为**2143808**

accuracy与loss随时间的变化，看到准确率到最后趋近于平稳，loss先快速下降然后有略微上升的趋势。

简单绘制神经网络模型结构如下：



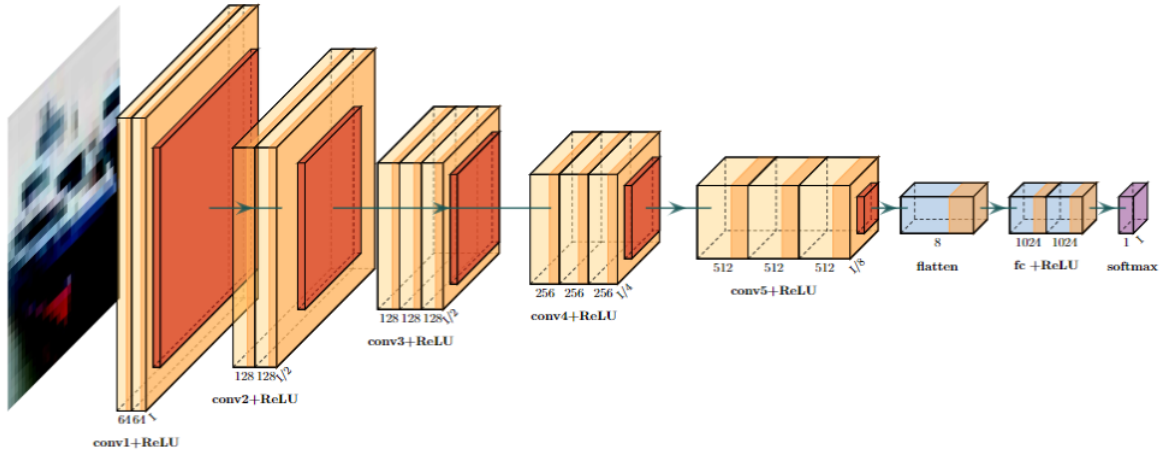

PlotNeuralNet 可以直观的画出模型的结构，而且较为美观。 我们的模型修改后的结构图大致如下：



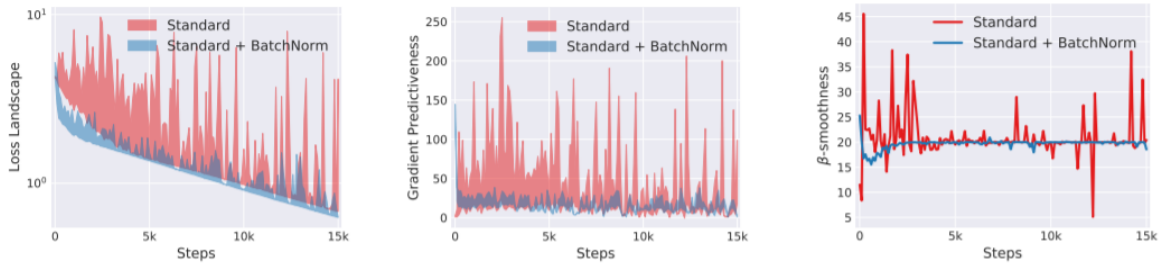

## 2. BatchNormalization VGG

batch normalization(BN) 是一种广泛采用的技术，可以更快、更稳定地训练深度神经网络 (DNN)。提高准确性和加速训练的趋势使 BN 成为深度学习中最受欢迎的技术。在高层次上，BN 是一种旨在通过稳定层输入分布来改进神经网络训练的技术。这是通过引入控制这些分布的前两个矩（均值和方差）的附加网络层来实现的。

VGG的基本结构可视化：




查找VGG的论文中，官方给出的梯度预测效果图如下：




(a) loss landscape　(b) gradient predictiveness　(c) “effective” $\beta$-smoothness

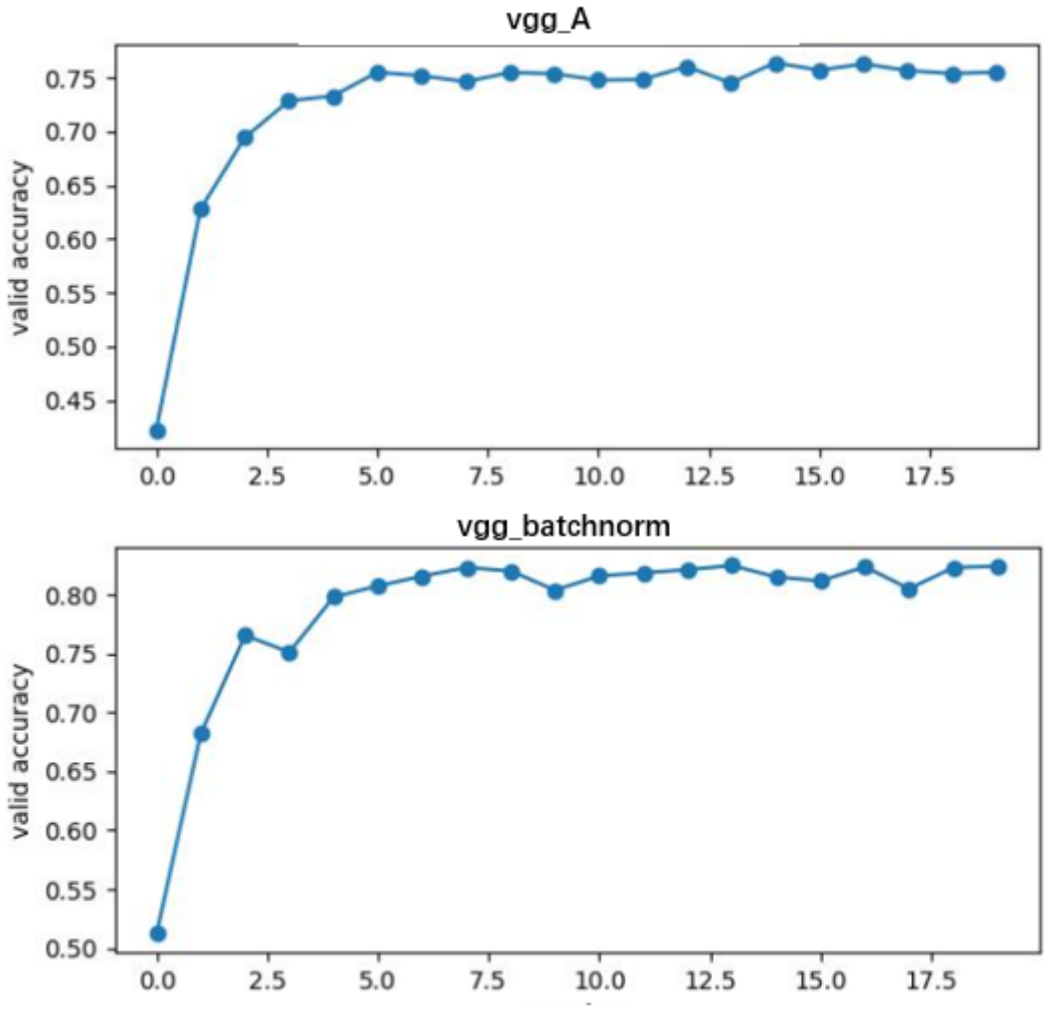
我在标准VGG模型上测试了batch norm的效果，给定超参数：

| optim | loss function | epoch num | batch size | activation |
|---|---|---|---|---|
| SGD | CrossEntropy | 20 | 64 | ReLU |

模型的精度如下：

| Model | speed | test accuracy |
|---|---|---|
| vgg | 14.53s/epoch | 0.764 |
| vgg_batchnorm | 18.34s/epoch | **0.8251** |

绘制valid accuracy 随epoch增加的变化趋势图：



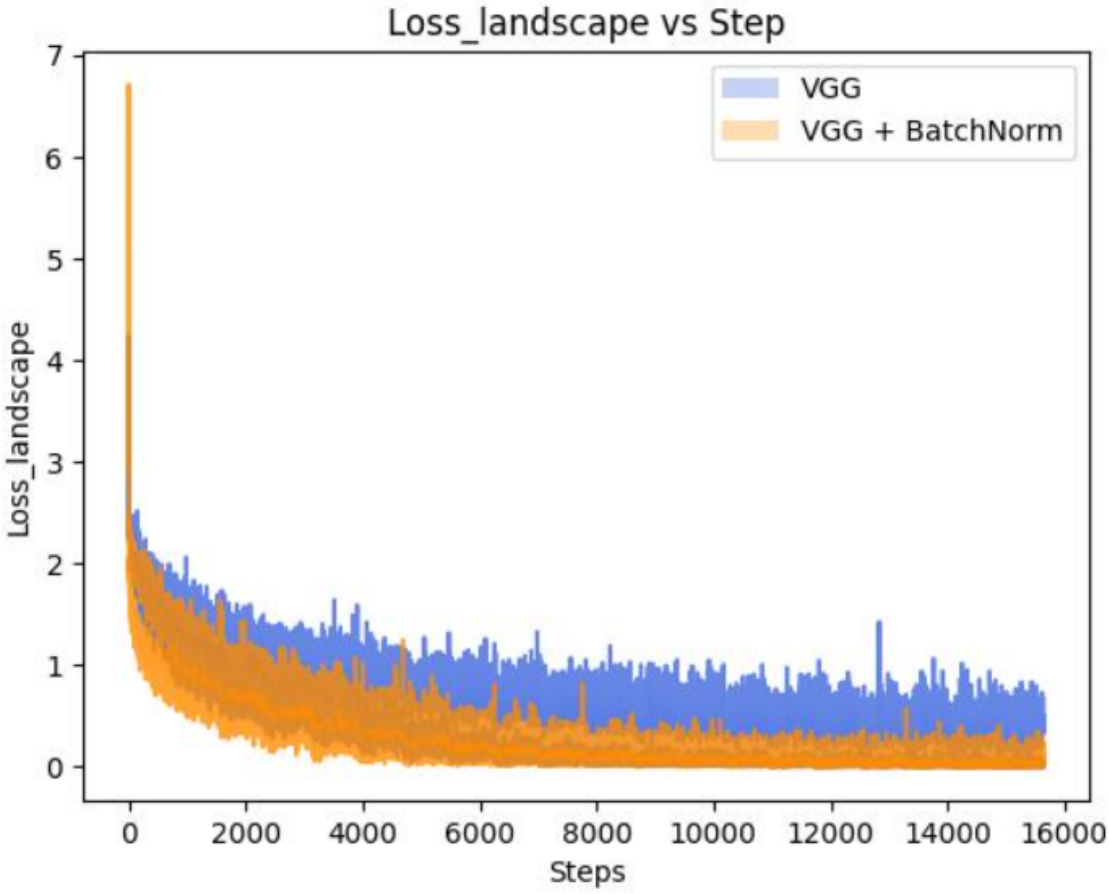

从结果可以看出，batch normalization可以有效提升模型的准确率，而且模型的精度提升更快。说明BN可以加速模型收敛。对损失的下降速度在下一部分讨论。

## 2.2 Loss Landscape

首先我创建了一个学习率列表来储存不同的步长，对每个步长都进行训练和保存模型

我们选择的学习率为：[2e-3,1e-3, 5e-4,1e-4], 在每一步时用loss的最小值min_loss, 和最大值max_loss, 储存在min_curve和max_curve的数组中。

然后用matplotlib.plt函数绘制折线，用fill between 的方法填充中间的区域。我们可以看到VGG-A+BN 收敛 速度更快，并且在不同的学习速率下波动较小。




## 2.3 Gradient Prediction

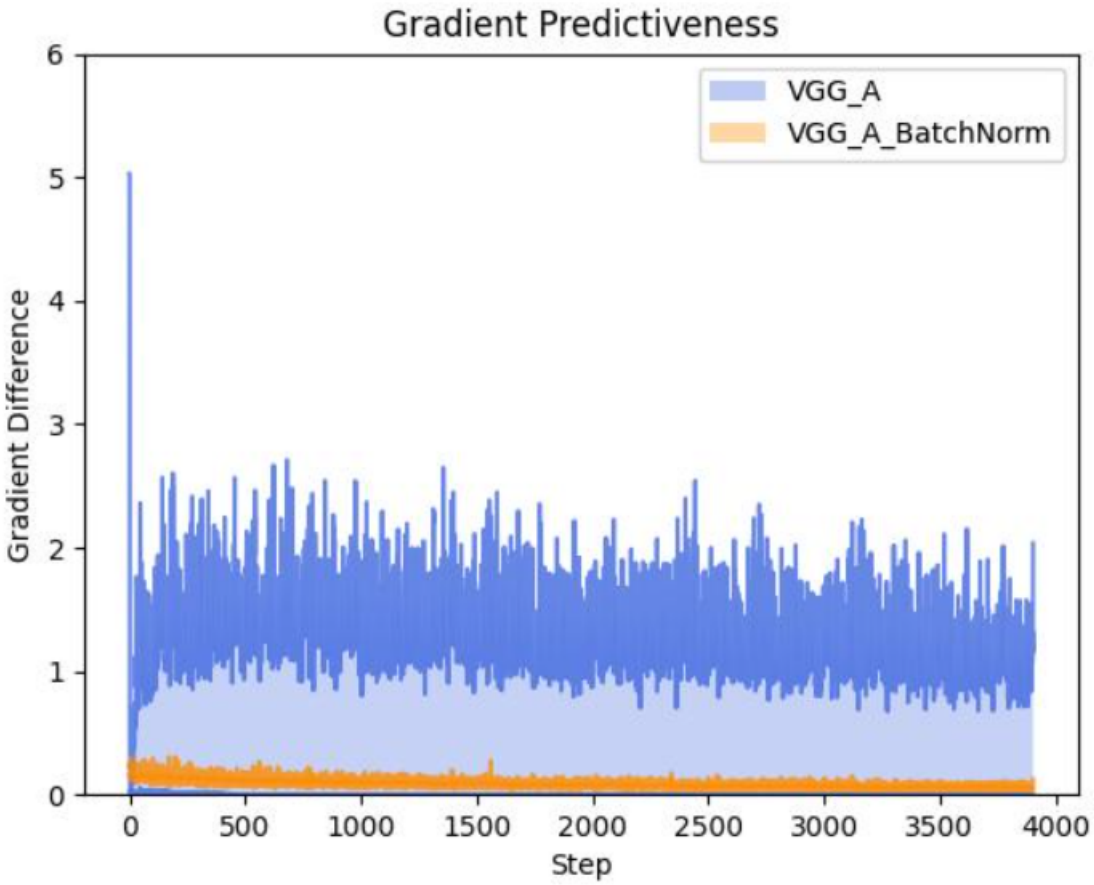
模型的梯度预测能力如下图所示，同样选择学习率为：[2e-3,1e-3, 5e-4,1e-4]

梯度误差可以用迭代前后的梯度张量的二范数表示：

$$Gradient\_Difference = ||grad_{k+1} - grad_k||_2, k = 1, 2, \dots$$

类似于loss landscape，我们也进行了可视化，结果如下

可以看到有BN时，VGG-A的梯度预测比没有BN时稳定得多。




## 2.4 Effective $\beta$-Smoothness

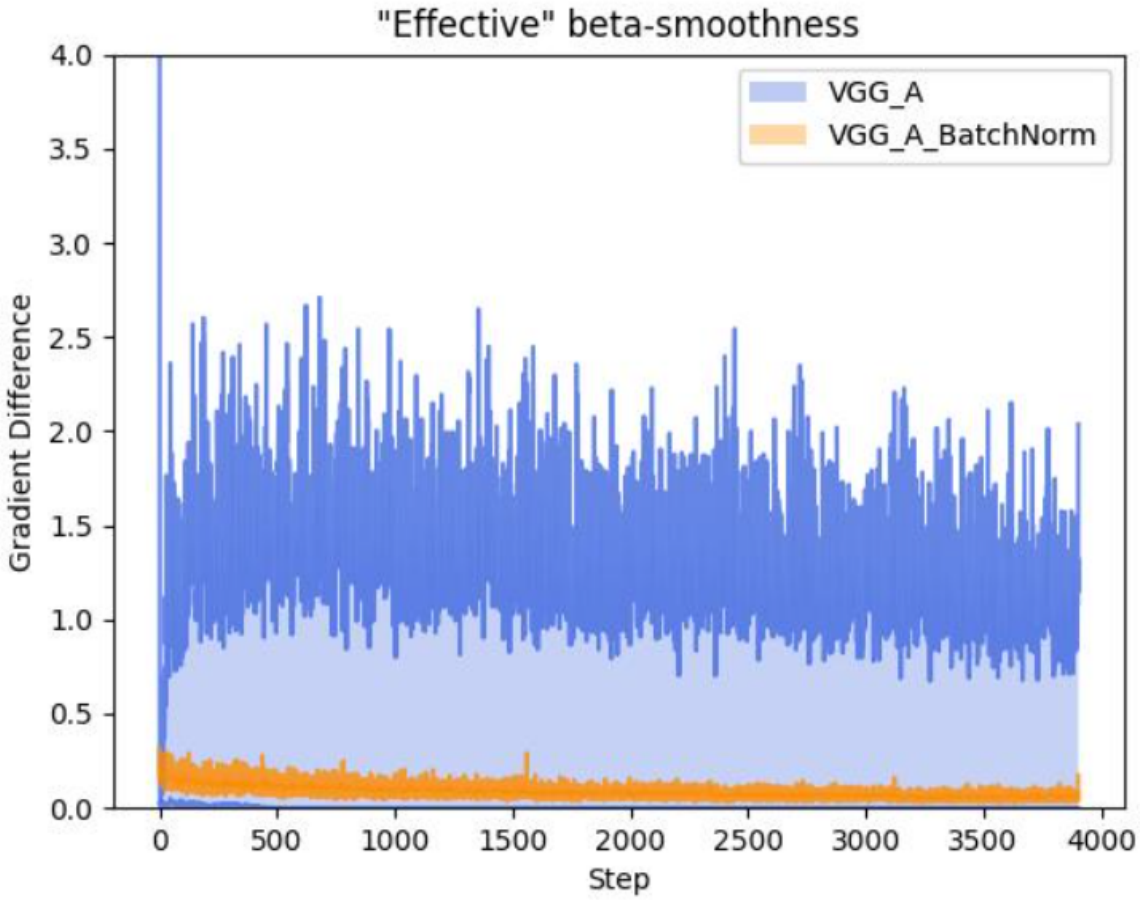
beta-smooth的条件是其梯度函数是$beta$-Lipschitz的，通过推导可以得到

$$\beta_i = max \frac{|| \bigtriangledown f_{x_i}(w_i) - \bigtriangledown f_{x_i}(w_i)||}{lr_i|| \bigtriangledown f_{x_i}(w_i)||}, i = 1, 2, 3...$$

当beta的值越小说明模型平滑程度越好。

有效的beta平滑模型训练结果如图4所示。

我们可以看到加了BN，VGG-A的平滑度比没有BN时稳定得多。




# 3 Extra Bonus 2

如今深度学习的巨大成功依赖于过度参数化。过度参数化虽然有利于训练过程，但由于参数量大，给我们使用深度神经网络带来了困难。我们可以使用 DessiLBI 进行前向选择，而不是使用后向选择方法来获得紧凑的模型，从而可以同时获得经过训练的过参数化模型和稀疏结构。在这个部分中我通过实验证明DessiLBI的改善。

# 3.1 DessiLBI + LeNET on MNIST

首先我们用Lenet模型对数据进行分类，在https://github.com/DessiLBI2020/DessiLBI/tree/master/DessiLBI/中作者给的代码中提供了超参数如下：

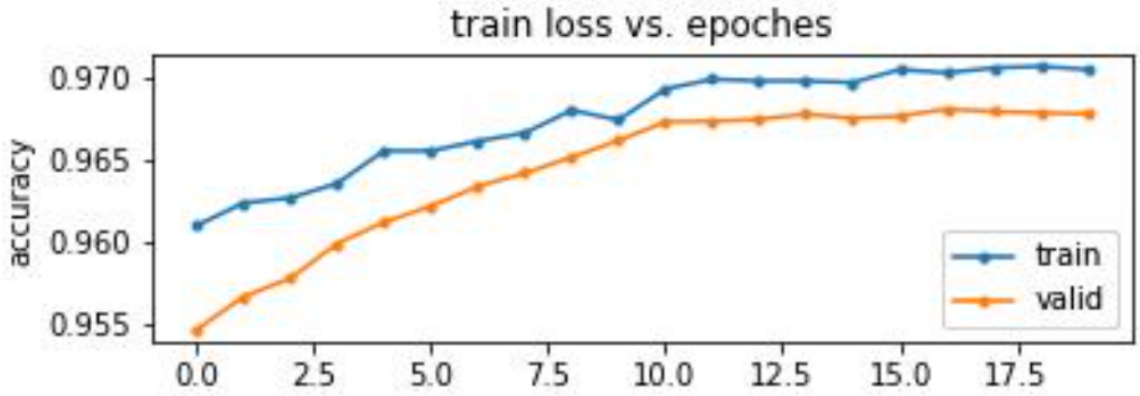
| optim | kappa | mu | batch size | epoch | lr |
|---|---|---|---|---|---|
| DessiLBI | 1 | 20 | 128 | 30 | 1*0.1^(1+epoch/20) |

初始的lr比较小,为了方便比较，我们先训练20个epoch得到一个pretrained model. 准确率在95.24%

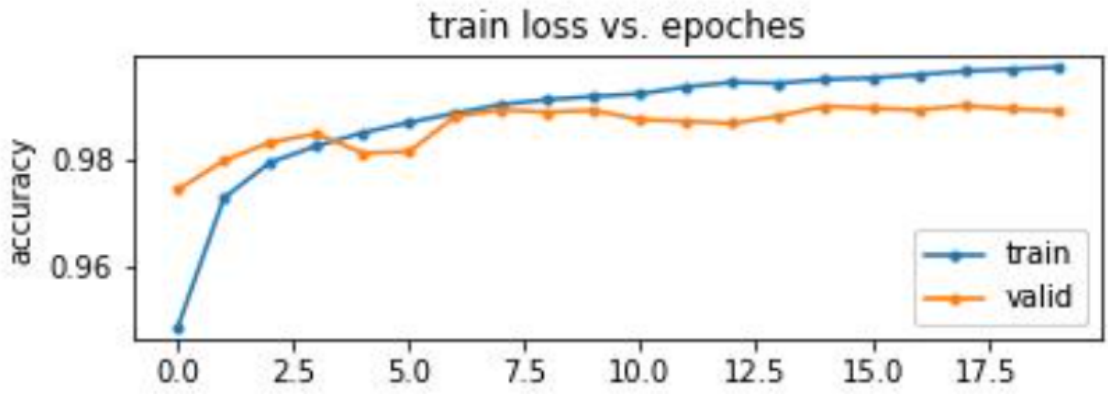
和SGD方法准确率比较结果,sgd的learning rate设置为0.04

| Optimizer | Speed | Accuracy |
|---|---|---|
| SGD | 3m45s | **0.9824** |
| Dessilbi | 3m55s | 0.9706 |

**DessiLBI**的准确率提升：




**SGD**的准确率提升：




DessiLBI的模型准确率似乎没有SGD好，猜测原因是超参数的选取，学习率太小导致训练精度和验证的精度都不再上升。同时epoch的设置可能也太小，模型未达到拟合。但也说明了确实Dessibli可以减少过参数化问题。

为了证明过参数化问题确实存在而且DessilLBI确实可以让模型有更好的泛化性能，我们对第三个卷积层作剪枝。

准确率变化如下表：

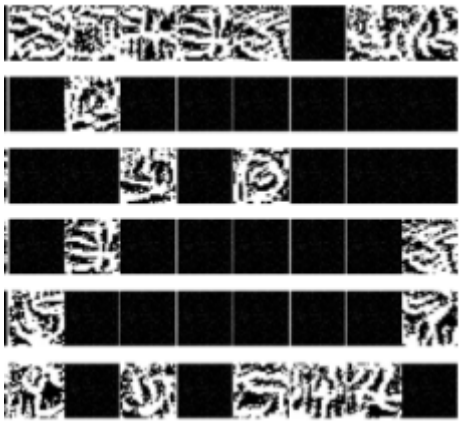
| **剪枝程度** | **0%** | **10%** | **20%** | **30%** | **40%** | **50%** | **60%** |
|---|---|---|---|---|---|---|---|
| conv3 | 0.9706 | 0.9704 | 0.9702 | 0.97 | 0.9653 | 0.9395 | 0.8925 |
| conv3+fc1 | 0.9706 | 0.9706 | 0.9707 | 0.9714 | 0.9659 | 0.9347 | 0.8837 |

可以看到经过剪枝模型的准确率有所降低但是基本变化不大。

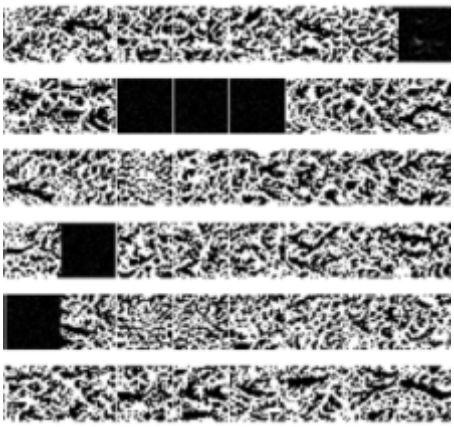
在剪枝40%以上时，猜测是因为关键的权重被剪枝了，模型准确率下降很快。

观察conv3过滤器模块的可视化：

Dessilbi：



SGD:



Dessilbi比SGD的黑色块更多，总体颜色更深。

这种结果似乎可以说明 DessiLBI 在不牺牲精度的情况下享有稀疏的过滤器选择。

# 3.2 Combine with adam

在优化深度网络的过程中，可以考虑使用 Adam 或其他自适应梯度方法来更新 DessiLBI 的 W。

按照project2附录中Adam Implementation提供的代码，我对opt中的step函数进行修改，保存在slbi-opt-adam函数中。

## Test with Mnist

使用lenet5模型作为我们训练的模型，

超参数如下：

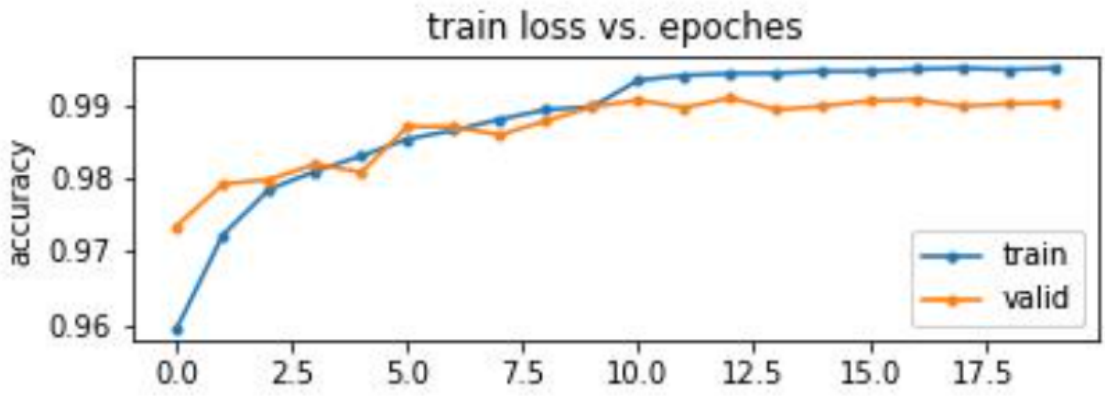
| **lr** | **kappa** | **mu** | **betas** |
|---|---|---|---|
| 0.0003 | 1 | 20 | (0.9,0.999) |
| **eps** | **weight_decay** | **dampening** | **epoch** |
| 1e-8 | 0 | 0 | 20 |

混合了adam后模型的具体准确率如下：

| Model | accuracy |
| --- | --- |
| **opt+adam** | **0.9903** |
| opt | 0.9706 |

在时间耗费上两个方法接近一致。

dessiLBI+adam的训练的accuracy和loss曲线：




从结果上将DessiLBI 和Adam优化器结合确实可以提升模型的准确率。发现在某些时间内验证的准确率会比训练准确率高，猜测是模型较简单存在欠拟合问题。

## Test with Cifar10

同样，我们用cifar10数据集来验证adam优化器算法的效果。

我们用问题2中VGGwith batchnorm的模型来训练。

超参数如下：

| lr | kappa | mu | betas |
| --- | --- | --- | --- |
| 0.04 | 1 | 500 | (0.9,0.999) |
| **eps** | **weight_decay** | **dampening** | **epoch** |
| 1e-8 | 0.0005 | 0 | 20 |

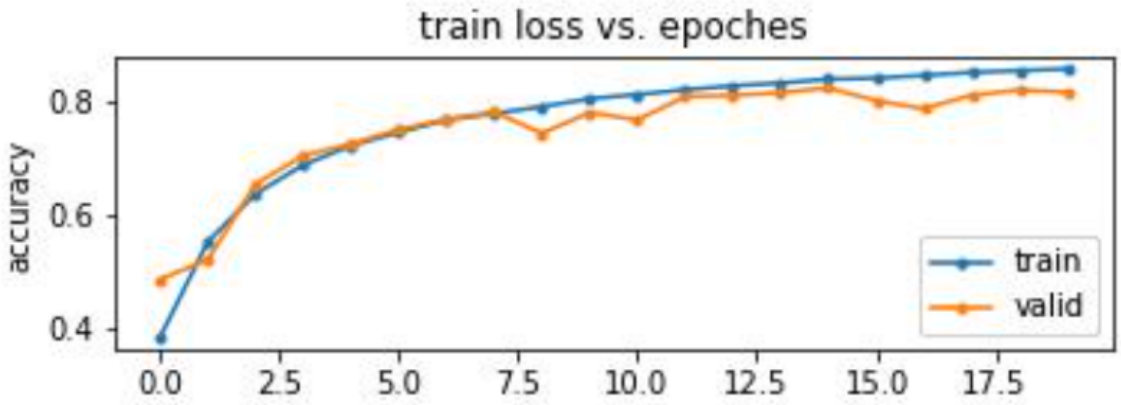
在第二问中已经讨论并得到了SGD优化器下VGG模型的准确率为0.8251，因此在这块直接比较DessiLBI和DessiLBI+ADAM的区别。

混合了adam后模型的具体准确率如下：

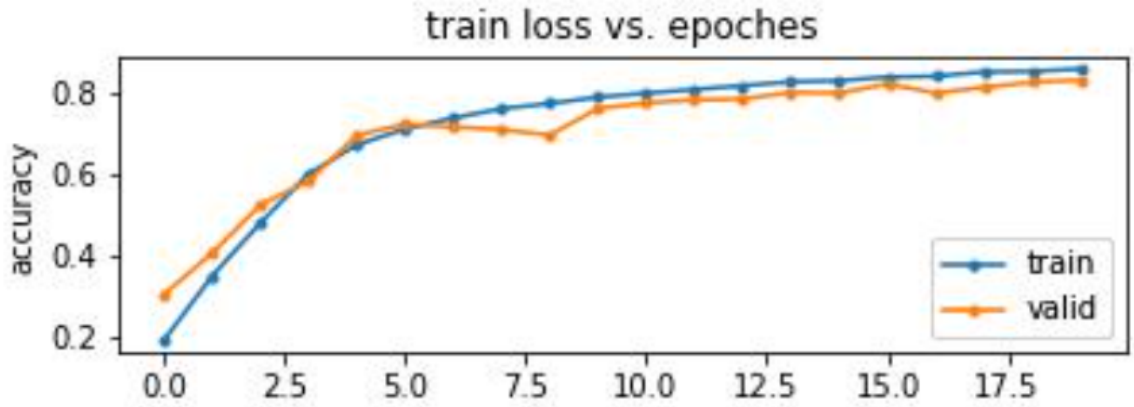
| Model | accuracy |
| --- | --- |
| **opt+adam** | **0.8345** |
| opt | 0.8185 |

画出两个模型准确率的提升曲线:

**DessiLBI：**




**DessiLBI+Adam：**




从结果上也可以得出，将DessiLBI 和Adam优化器结合确实可以提升模型的准确率。发现在某些时间内验证的准确率会比训练准确率高，猜测是模型较简单存在欠拟合问题。