《计算机视觉》第一次实验：物体识别或图像分类

一、实验目的

- 加强对基于Mindspore的神经网络模型构建流程的理解
- 掌握如何用Mindspore实现卷积神经网络的构建
- 学会利用checkpoint函数保存模型参数
- 掌握如何利用模型预测单张图像的分类结果

二、实验环境

- Python 3.7.5
- Mindspore 1.5
- Matplotlib 3.2.2
- Numpy 1.18.5
- 平台：华为AI平台
  数据：cifar10数据
  参考：华为计算机视觉实验1 https://edu.huaweicloud.com/roadmap/colleges.html

三、实验内容

任务一：按照华为平台实验手册进行操作

> 要求：熟悉实验环境，掌握卷积神经网络模型程序流程
> 具体：记录并观察实验结果，如损失随迭代轮数变化等

任务二：LeNet-5模型对比

> 要求：调节实验参数，进行实验对比及分析
> 具体：实验参数包括：
> 1.画出LeNet-5网络结构示意图，网络结构参数表格；
> 2.训练批次大小，迭代轮数，学习速率等

任务三：卷积神经网络模型设计

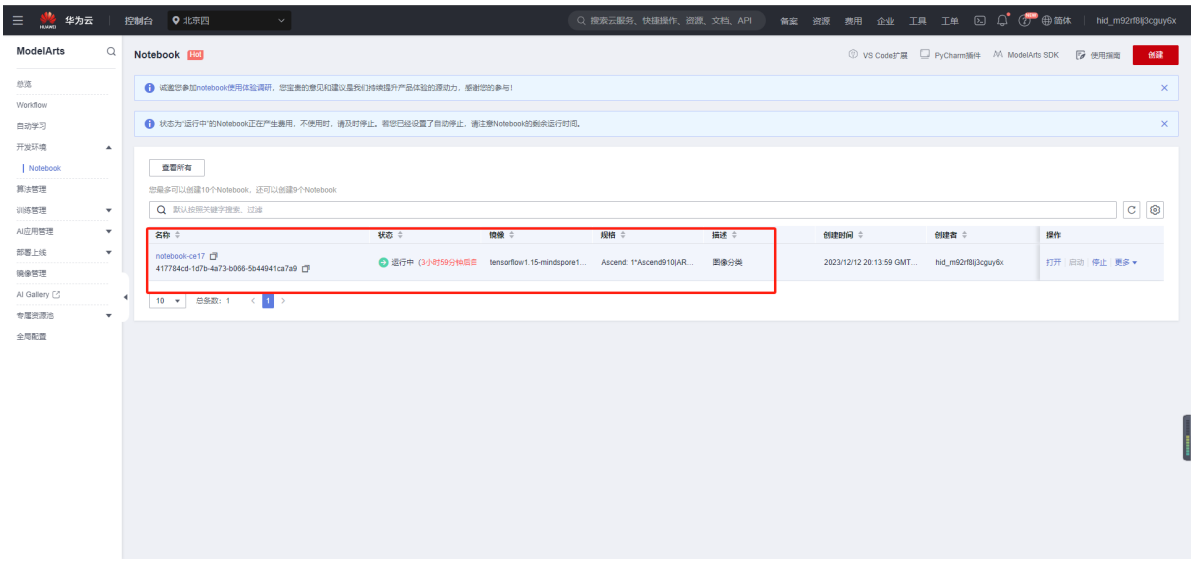
> 要求：改变网络结构，进行实验对比及分析
> 具体：
> 1.改变网络结构参数，包括卷积层数，卷积核尺寸、步长及是否填充，全连接层层数，各层节点数目等，构建一个新的LeNet-5网络，进行新网络和原网络性能对比；
> 2.添加和不添加BN层。

四、实验过程及结果

1、任务一：按照华为平台实验手册进行操作

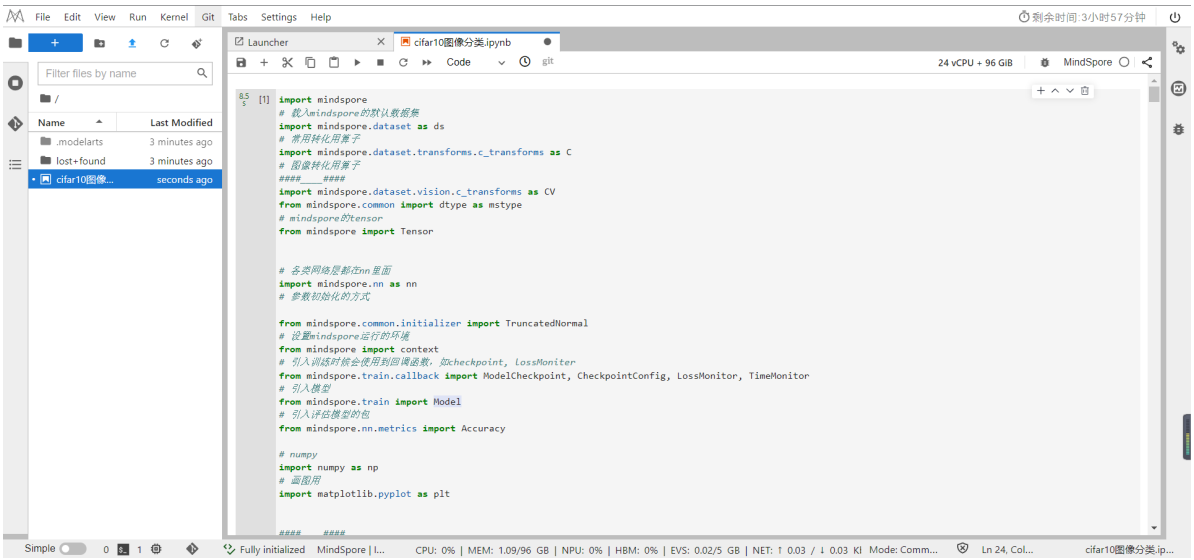
（1）创建华为云Notebook

按照助教老师的示范，创建并启动Notebook，如下图所示：



（2）导入相关实验模块

在Notebook中导入实验一所给的代码，如图所示：

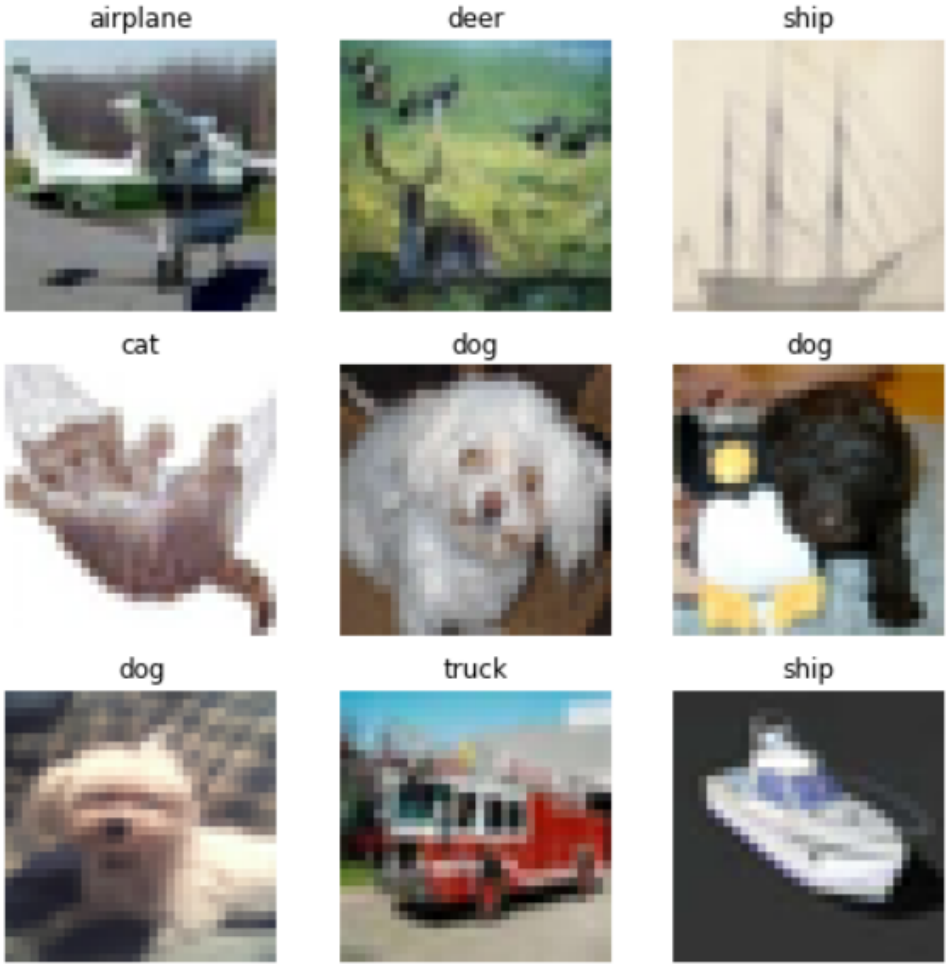


（3）数据集展示与数据初始化

下载成功提示：

```
downloading with requests
download finished
data prepared
```

数据集展示：

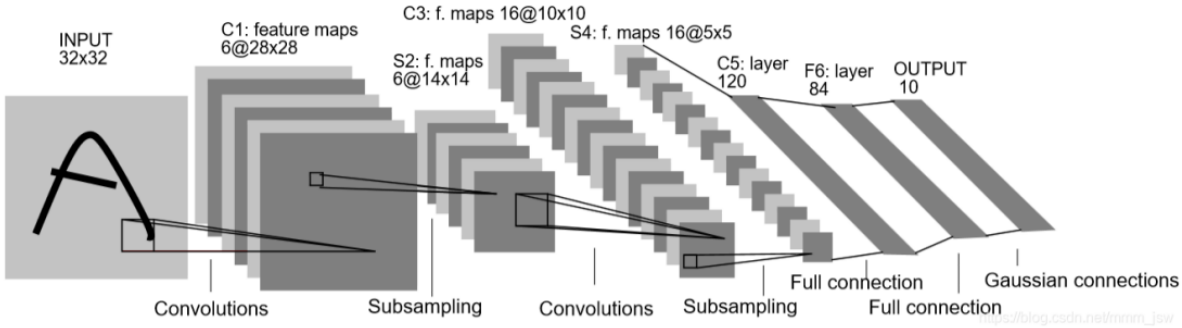


> 数据集一共有10个类，10个类完全相互排斥，且类之间没有重叠
>
> 分别是：airplane/automobile/bird/cat/deer/dog/frog/horse/ship/truck

定义数据预处理的步骤、生成训练数据集略

（4）构建网络模型

LeNet-5网络结构：



其网络结构如下：

- INPUT（输入层）：输入32*32的图片。
- C1（卷积层）：选取6个5*5卷积核(不包含偏置)，得到6个特征图，每个特征图的一个边为32−5+1=28，也就是神经元的个数由32*32=1024减小到了28*28=784。
- S2（池化层）：池化层是一个下采样层，输出14*14*6的特征图。
- C3（卷积层）：选取16个大小为5*5卷积核，得到特征图大小为10*10*16。
- S4（池化层）：窗口大小为2*2，输出5*5*16的特征图。
- F5（全连接层）：120个神经元。

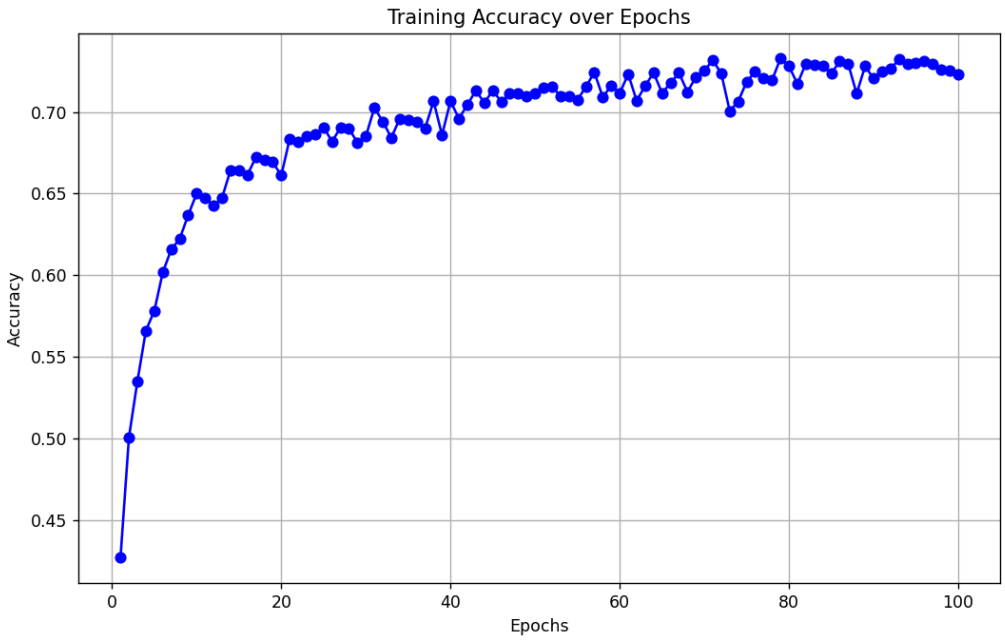
- F6（全连接层）：84个神经元。
- OUTPUT（输出层）：10个神经元，10分类问题。

（5）模型训练与测试

epoch取1-100的训练结果如下：

```
============== Starting Training ==============
epoch: 1 step: 1562, loss is 1.502115249633789
{'Accuracy': 0.42715669014084506}
epoch: 2 step: 1562, loss is 1.6712968349456787
{'Accuracy': 0.5008602752880922}
epoch: 3 step: 1562, loss is 1.0625323057174683
{'Accuracy': 0.5351712548015365}
epoch: 4 step: 1562, loss is 1.2833151817321777
{'Accuracy': 0.5661211587708067}
epoch: 5 step: 1562, loss is 1.1646469831466675
{'Accuracy': 0.5782850512163893}
......
epoch: 95 step: 1562, loss is 0.4161587357521057
{'Accuracy': 0.730193661971831}
epoch: 96 step: 1562, loss is 0.5974003672599792
{'Accuracy': 0.7309939180537772}
epoch: 97 step: 1562, loss is 0.6811219453811646
{'Accuracy': 0.729433418693982}
epoch: 98 step: 1562, loss is 1.0947656631469727
{'Accuracy': 0.7261923815620999}
epoch: 99 step: 1562, loss is 0.8114985823631287
{'Accuracy': 0.725432138284251}
epoch: 100 step: 1562, loss is 0.5283129215240479
{'Accuracy': 0.7232314340588989}
```

编写代码，将训练结果绘制成训练曲线，如下所示：

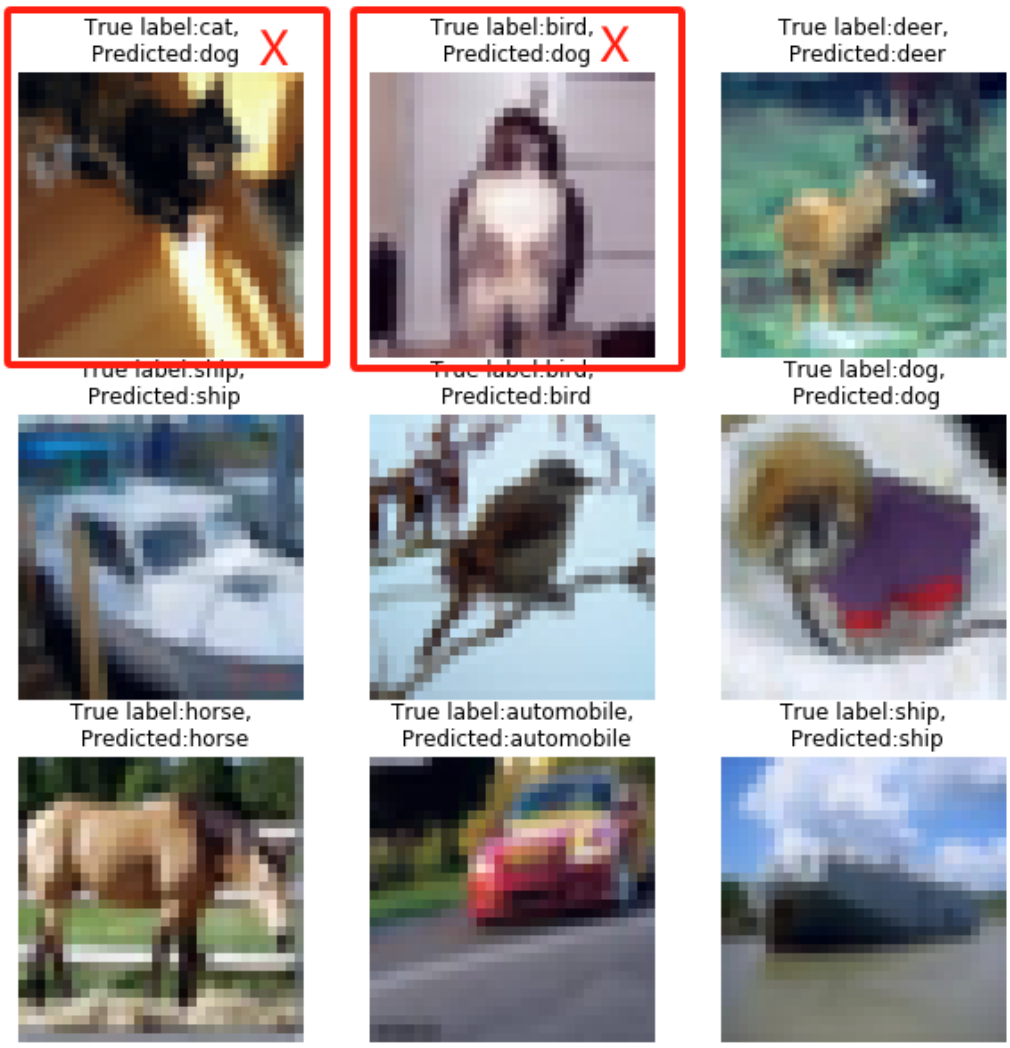


设置测试集参数并测试，准确度为：

```
test results: {'Accuracy': 0.7196514423076923}
```

由此可知，不管是训练集还是测试集准确率都只有70%左右，可见模型处于欠拟合状态。

图片类别预测与可视化，有两个预测是错误的，结果如下：



思考题：

- 彩色图像有几个颜色通道?

  3个颜色通道，一般为RGB。

- 请列举常见的颜色空间。

  RGB, HSV，Lab，CMYK等。

- 为什么我们在训练时会使用随机裁剪或者翻转的方式来处理图片?

  为了增加网络的泛化能力，即使图片只有部分或者翻转了，也依然有能力识别出来。

- 将图片归一化有什么好处?

  加快模型的训练时候的收敛速度，也可以在一定程度上避免梯度消失或者爆炸。

- 除了Adam优化器，请列举其他常见的优化器?

  SGD, BGD, MBGD, Momentum, NAG(Nesterov Accelerated Gradient)，Adagrad，Adadelta，RMSprop。

- 什么叫一个epoch?

  一个epoch指代所有的数据送入网络中完成一次前向计算及反向传播的过程。

- 如果模型过拟合，常见的处理方法有哪些?

  L1，L2正则化，Early stopping，数据增强，dropout等。

2、任务二：LeNet-5模型对比

（1）画出LeNet-5网络结构示意图，网络结构参数表格

LeNet-5网络结构示意图：

```
Input (32x32x1)
   |
   v
Conv1 (6 filters, 5x5, stride 1)
   |
   v
Tanh Activation
   |
   v
Avg Pooling (2x2, stride 2)
   |
   v
Conv2 (16 filters, 5x5, stride 1)
   |
   v
Tanh Activation
   |
   v
Avg Pooling (2x2, stride 2)
   |
   v
Flatten
   |
   v
Fully Connected Layer (120 neurons)
   |
   v
Tanh Activation
   |
   v
Fully Connected Layer (84 neurons)
   |
   v
Tanh Activation
   |
   v
Output Layer (10 neurons, softmax)
```

LeNet-5网络结构参数表格：

Layer	Type	Output Shape	Parameters
Input	Input	(32, 32, 1)	0
Conv1	Convolution	(28, 28, 6)	(5x5x1x6) + 6
Activation1	Tanh	(28, 28, 6)	0
MaxPool1	Max Pooling	(14, 14, 6)	0
Conv2	Convolution	(10, 10, 16)	(5x5x6x16) + 16
Activation2	Tanh	(10, 10, 16)	0
MaxPool2	Max Pooling	(5, 5, 16)	0
Flatten	Flatten	400	0
FullyConnected1	Fully Connect	120	(400x120) + 120
Activation3	Tanh	120	0
FullyConnected2	Fully Connect	84	(120x84) + 84
Activation4	Tanh	84	0
Output	Fully Connect	10	(84x10) + 10

注：

- Conv1 和 Conv2 分别是第一个和第二个卷积层。
- Activation1、Activation2、Activation3、Activation4 是激活函数层，通常使用 Tanh。
- MaxPool1 和 MaxPool2 分别是第一个和第二个最大池化层。
- Flatten 将前一层的输出拉平成一维向量，用于连接全连接层。
- FullyConnected1 和 FullyConnected2 是全连接层。
- Output 是输出层，通常使用 softmax 激活函数。

参数计算方式：

- 对于卷积层，参数个数等于卷积核大小乘以输入通道数再乘以输出通道数，再加上每个输出通道的偏置项。
- 对于全连接层，参数个数等于前一层神经元个数乘以当前层神经元个数，再加上每个神经元的偏置项。

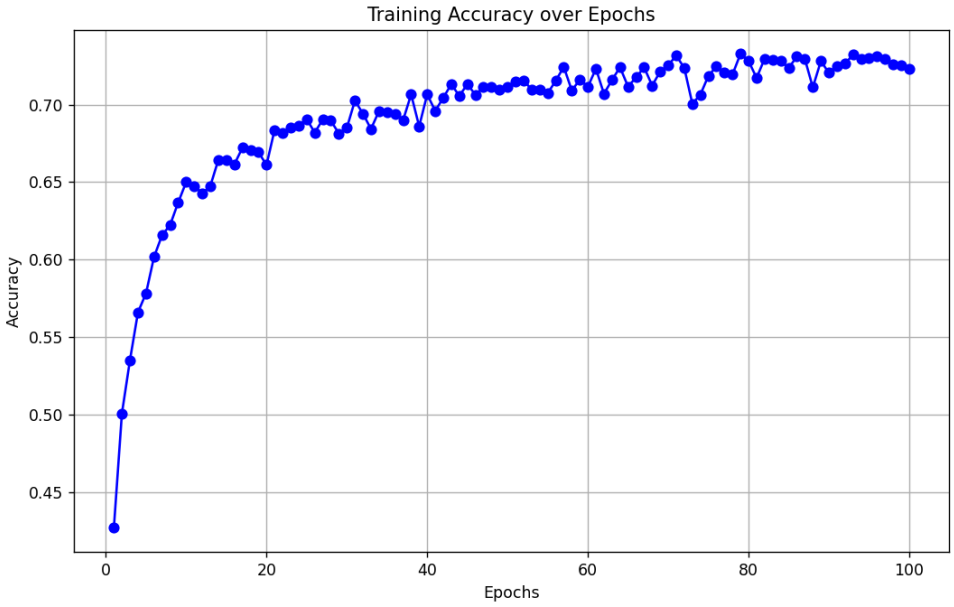
（2）调节实验参数（训练批次大小，迭代轮数，学习速率等）

改变训练批次大小（别的参数不变）：

59 ms [8]
```
data_path = os.path.join(current_path, 'data/10-batches-bin')
batch_size=64
status="train"

# 生成训练数据集
cifar_ds = get_data(data_path)
ds_train = process_dataset(cifar_ds,batch_size =batch_size, status=status)
```

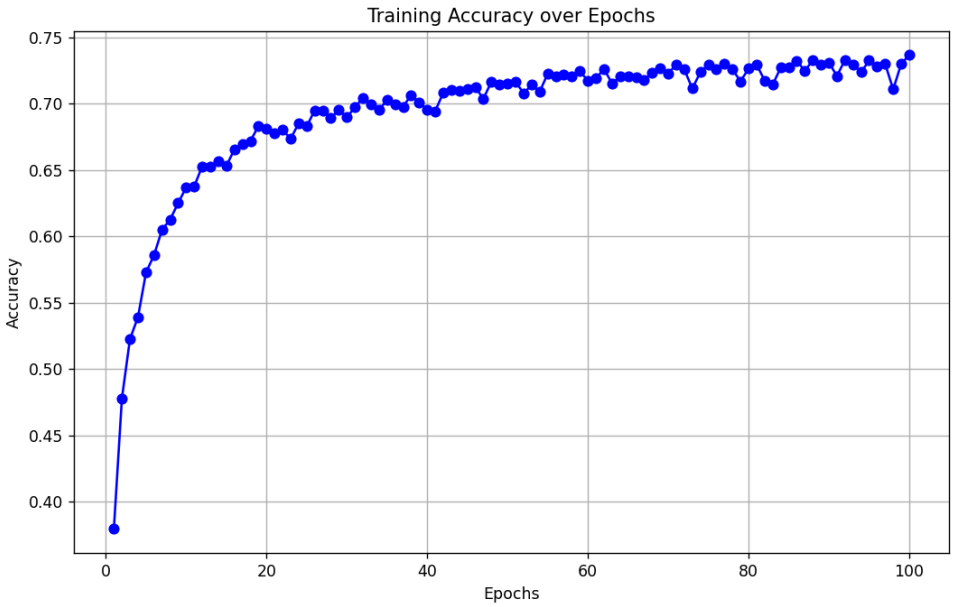
batch_size = 32, 64, 128

- batch_size = 32



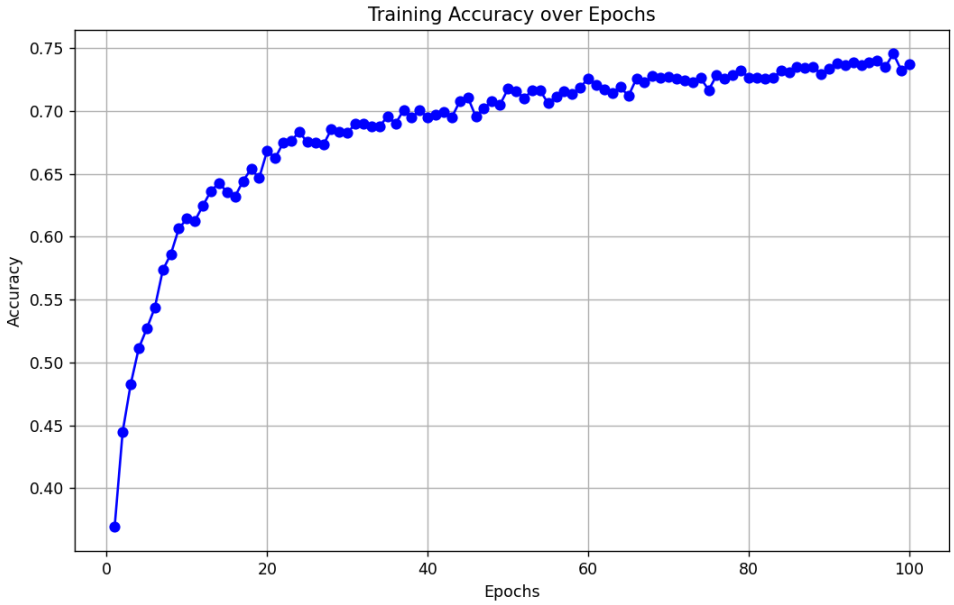

```
test results: {'Accuracy': 0.7196514423076923}
```

- batch_size = 64




```
test results: {'Accuracy': 0.725761217948718}
```

- batch_size = 128




```
test results: {'Accuracy': 0.7335576923076923}
```

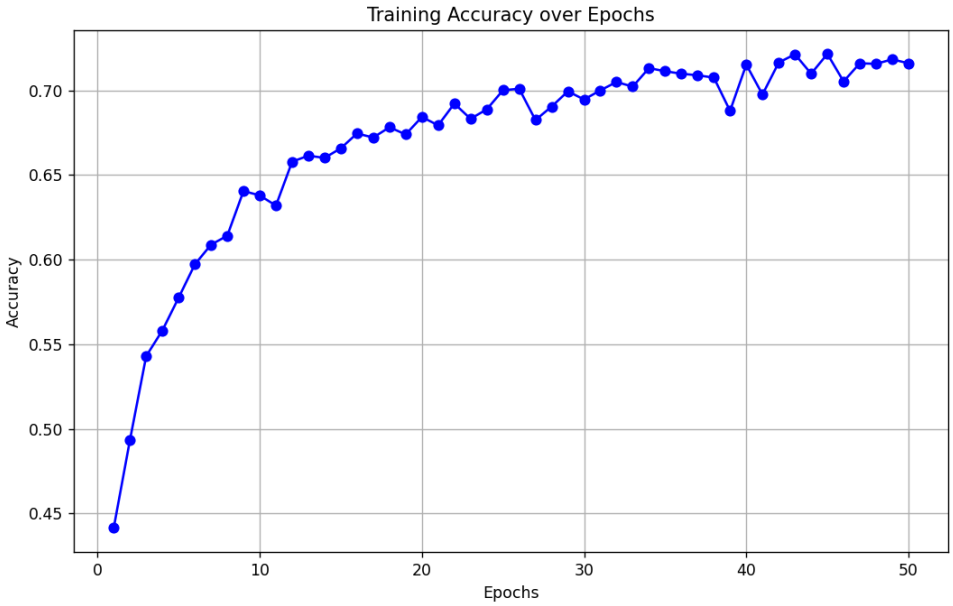
结论：上述实验表明，**当batch_size取32、64、128时，随着batch_size增大，损失略有减少，测试集的准确率有所提高，模型准确率有微小提高。**可能是因为较大的 batch_size 可能导致更稳定的梯度估计，从而更平滑地更新模型参数。且模型在每个更新步骤中看到更多的样本，有助于学习更一般化的特征，提高模型对测试集的泛化能力。然而这种提高是非常微小的，在实践中应当**选择合适的batch_size，并不是越大越好。**（查阅资料发现，在一些情况下，较小的 `batch_size` 可能会表现得更好）

改变迭代轮数（别的参数不变）：

```
# 设置CheckpointConfig，callback函数，save_checkpoint_steps=训练总数/batch_size
config_ck = CheckpointConfig(save_checkpoint_steps=1562,
                             keep_checkpoint_max=10)
ckpoint_cb = ModelCheckpoint(prefix="checkpoint_lenet_original", directory='./results',config=config_ck)
# 建立可训练模型
model = Model(network = network, loss_fn=net_loss,optimizer=net_opt, metrics={"Accuracy": Accuracy()})
eval_per_epoch = 1
epoch_per_eval = {"epoch": [], "acc": []}
eval_cb = EvalCallBack(model, ds_train, eval_per_epoch, epoch_per_eval)
print("============== Starting Training ==============")
model.train(50, ds_train,callbacks=[ckpoint_cb, LossMonitor(per_print_times=1),eval_cb],dataset_sink_mode=dataset_sink_mode)
```

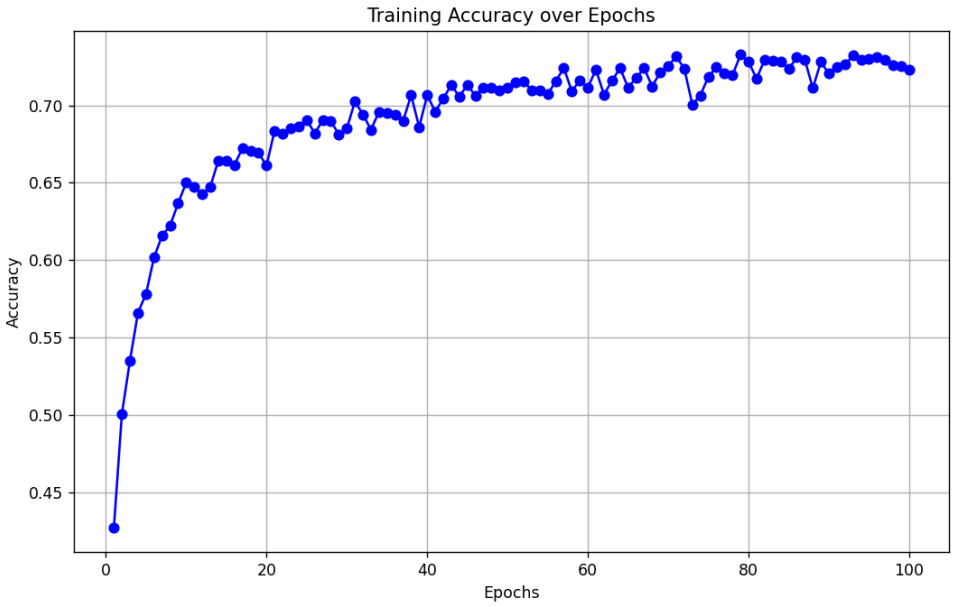
epoch = 50, 100, 150

- epochs = 50



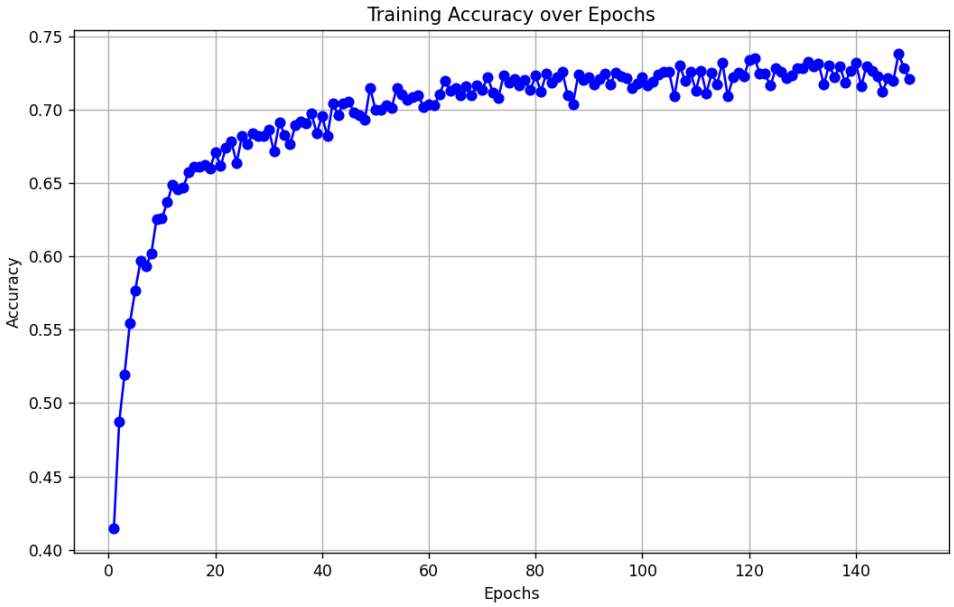

```
test results: {'Accuracy': 0.6913612355953905}
```

- epochs = 100




```
test results: {'Accuracy': 0.7196514423076923}
```

- epochs = 150




```
test results: {'Accuracy': 0.7392508012820513}
```

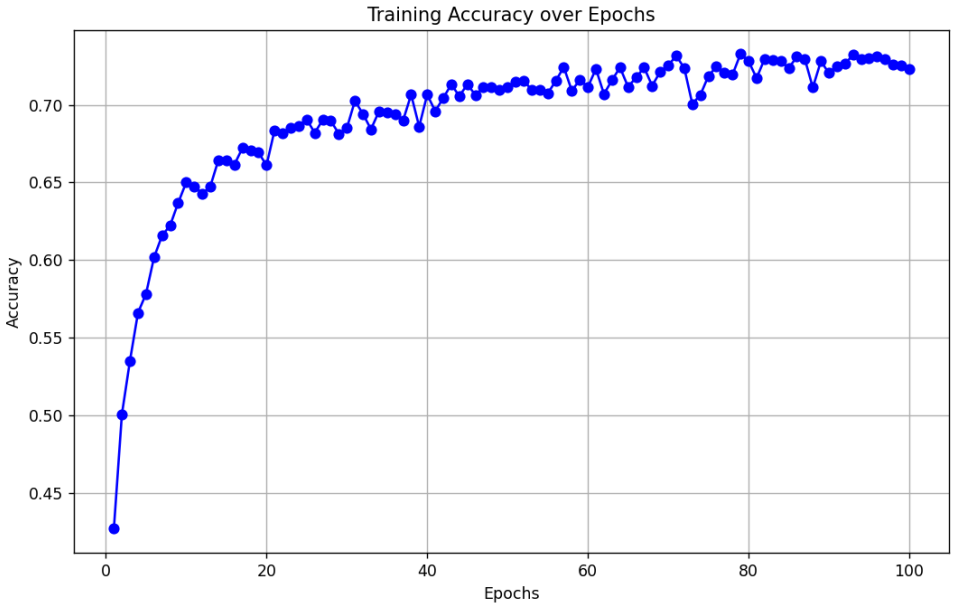
结论：上述实验表明，**当epochs分别取50、100、150时，随着epochs增大，训练集的准确率略有提高，模型准确率有微小提高。** 增加 epochs 提供了更多的学习时间，使得模型有更多机会学习数据的特征，适应数据的复杂性，尤其是对于复杂的任务和大型数据集。然而，**增加 epochs 并不总是会导致性能提高，而且可能会增加训练时间，甚至导致过拟合。**实践中，需要通过验证集的性能来监控模型的泛化能力，以确定何时停止训练。

改变学习速率（别的参数不变）：

```
# 返回当前设备
device_target = mindspore.context.get_context('device_target')
# 确定图模型是否下沉到芯片上
dataset_sink_mode = True if device_target in ['Ascend','GPU'] else False
# 设置模型的设备与图的模式
context.set_context(mode=context.GRAPH_MODE, device_target=device_target)
# 使用交叉熵函数作为损失函数
net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
# 优化器为Adam
net_opt = nn.Adam(params=network.trainable_params(), learning_rate=0.003)
# 监控每个epoch训练的时间
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
```

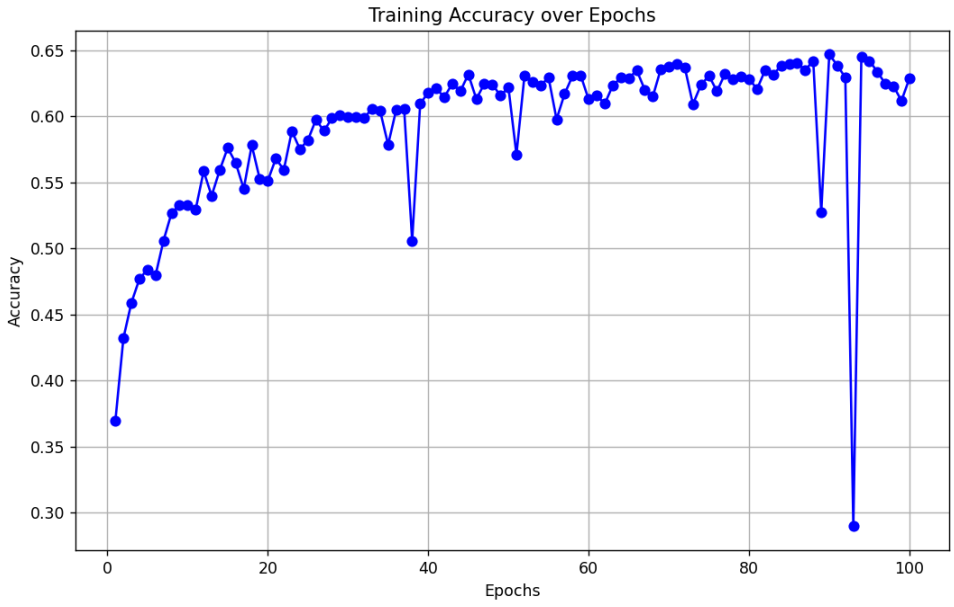
learning_rate = 0.001, 0.003, 0,005

- learning_rate = 0.001



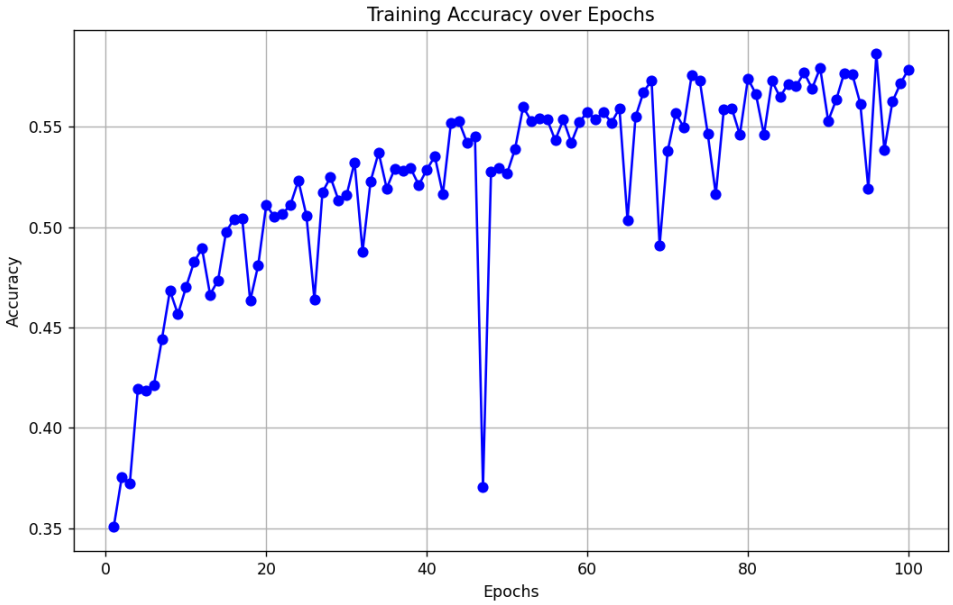

```
test results: {'Accuracy': 0.7196514423076923}
```

- learning_rate = 0.003




```
test results: {'Accuracy': 0.6365184294871795}
```

- learning_rate = 0.005




```
test results: {'Accuracy': 0.5743189102564102}
```

结论：上述实验表明， **当learning_rate分别取0.001、0.003、0.005时，随着learning_rate增大，训练集的准确率降低，损失增多，测试集的准确率降低，模型准确率有降低。**因为增大学习率可能导致训练不稳定，甚至导致性能下降： 较大的学习率可能导致梯度爆炸或梯度消失，导致模型参数偏离最优解或模型学习缓慢。实践中应当选择合适的学习率，通常通过尝试不同的学习率进行实验，找到在给定问题上表现最佳的学习率。也可以**使用自适应学习率算法，** 例如，Adam、Adagrad等自适应学习率算法可以在训练过程中自动调整学习率。

调节最优化方法，由Adam改为SGD:

3、任务三：卷积神经网络模型设计

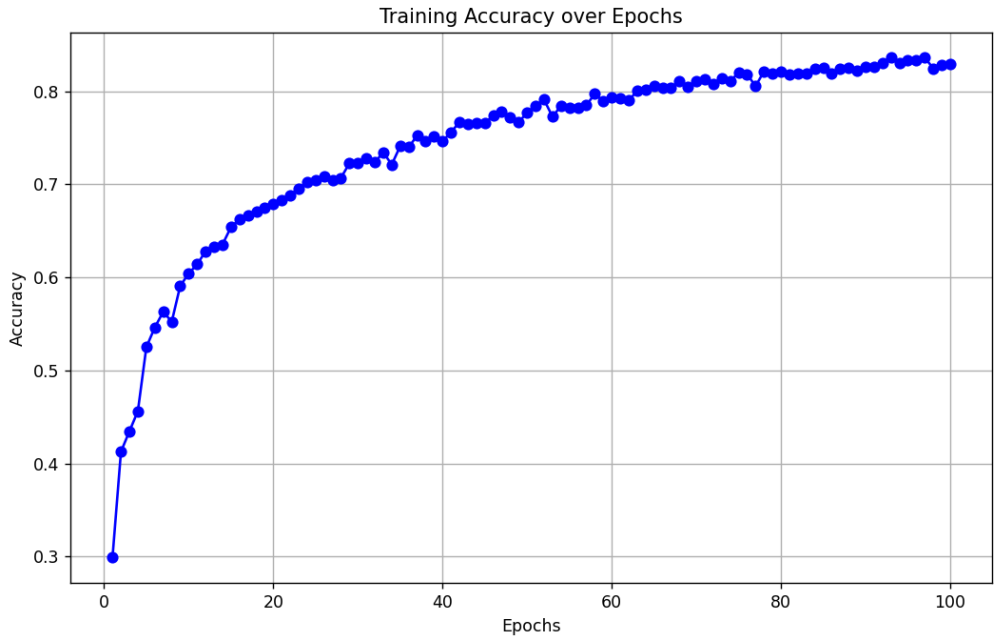
（1）改变网络结构参数

> 所有的卷积核从5*5变成3*3
>
> 增加了两层卷积层，提升模型的非线性映射能力
>
> 提升了卷积核数量至64核，使模型可以提取更多的特征

修改代码:

```
class LeNet5(nn.Cell):
    """
    Lenet network
    Args:
        num_class (int): Num classes. Default: 10.
    Returns:
        Tensor, output tensor
    Examples:
        >>> LeNet(num_class=10)
    """
    def __init__(self, num_class=10, channel=3):
        super(LeNet5, self).__init__()
```

```
        self.num_class = num_class
        # 1.所有的卷积核从5*5变成3*3。
        # 2.增加了两层卷积层，提升模型的非线性映射能力。
        # 3.提升了卷积核数量为64核
        self.conv1_1 = conv(channel, 8, 3)
        self.conv1_2 = conv(8, 16, 3)
        self.conv2_2 = conv(32, 64, 3)
        self.conv2_2 = conv(32, 64, 3)
        self.fc1 = fc_with_initialize(64 * 8 * 8, 120)
        self.fc2 = fc_with_initialize(120, 84)
        self.fc3 = fc_with_initialize(84, self.num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

    def construct(self, x):
    def construct(self, x):
        x = self.conv1_1(x)
        #x = self.bn2_1(x)
        x = self.relu(x)
        x = self.conv1_2(x)
        #x = self.bn2_2(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv2_1(x)
        #x = self.bn2_3(x)
        x = self.relu(x)
        x = self.conv2_2(x)
        #x = self.bn2_4(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        #x = self.bn1_1(x)
        x = self.relu(x)
        x = self.fc2(x)
        #x = self.bn1_2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
# 构建网络
network = LeNet5(10)
```




```
test results: {'Accuracy': 0.805136547088623}
```

结论：**改变网络结构参数后的模型在训练集和测试集上的正确率均有较大提升。**通过调整激活函数、层数或神经元数量等参数，可能引入更多非线性关系，使得模型能够更好地捕捉数据中的复杂结构。

（2）添加BN层

在新网络的基础上，添加BN层，代码如下：

```
class LeNet5_2(nn.Cell):
    """
    Lenet network
    Args:
        num_class (int): Num classes. Default: 10.
    Returns:
        Tensor, output tensor
    Examples:
        >>> LeNet(num_class=10)
    """
    def __init__(self, num_class=10, channel=3):
        super(LeNet5_2, self).__init__()
        # 1.所有的卷积核从5*5变成3*3。
        # 2.增加了两层卷积层，提升模型的非线性映射能力。
        # 3.提升了卷积核数量为64核
        # 4.在每一层网络层中加入BatchNormalization层
        self.num_class = num_class
        self.conv1_1 = conv(channel, 8, 3)
        self.bn2_1 = nn.BatchNorm2d(num_features=8)
        self.conv1_2 = conv(8, 16, 3)
        self.bn2_2 = nn.BatchNorm2d(num_features=16)
        self.conv2_1 = conv(16, 32, 3)
        self.bn2_3 = nn.BatchNorm2d(num_features=32)
        self.conv2_2 = conv(32, 64, 3)
        self.bn2_4 = nn.BatchNorm2d(num_features=64)
        self.fc1 = fc_with_initialize(64*8*8, 120)
```

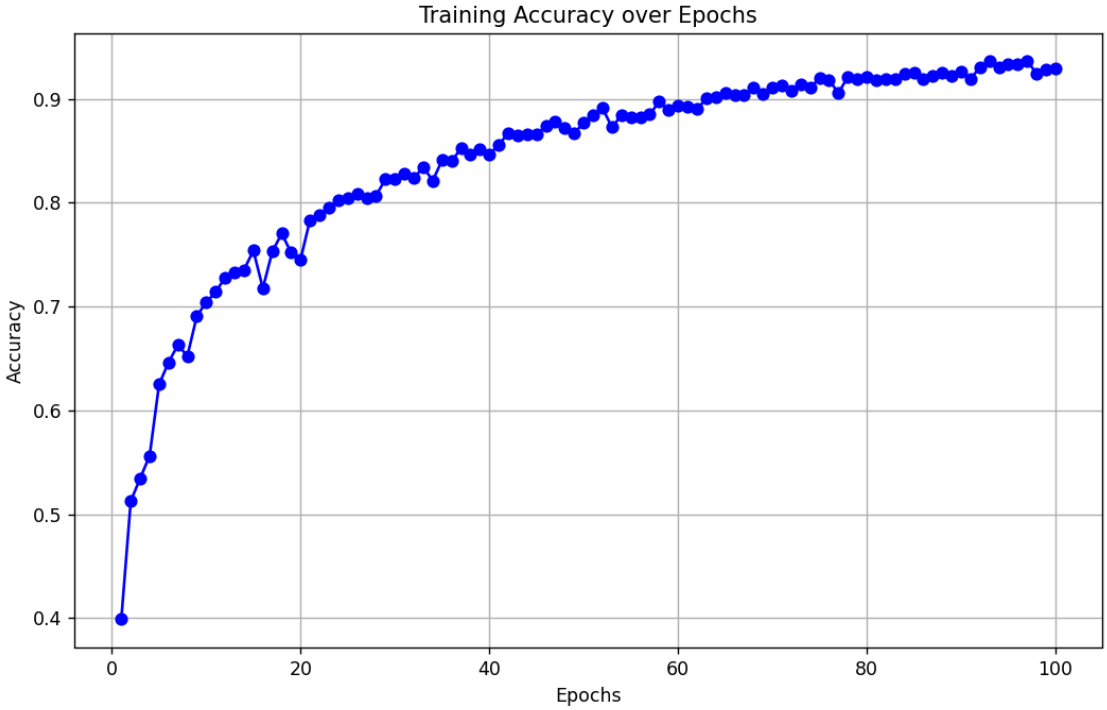
```
        self.bn1_1 = nn.BatchNorm1d(num_features=120)
        self.fc2 = fc_with_initialize(120, 84)
        self.bn1_2 = nn.BatchNorm1d(num_features=84)
        self.fc3 = fc_with_initialize(84, self.num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

    def construct(self, x):
        x = self.conv1_1(x)
        x = self.bn2_1(x)
        x = self.relu(x)
        x = self.conv1_2(x)
        x = self.bn2_2(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv2_1(x)
        x = self.bn2_3(x)
        x = self.relu(x)
        x = self.conv2_2(x)
        x = self.bn2_4(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1_1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.bn1_2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
```

epoch取1-100的训练结果如下：

```
============== Starting Training ==============
epoch: 1 step: 312, loss is 1.6064125299453735
{'Accuracy': 0.39893830128205127}
epoch: 2 step: 312, loss is 1.4331836700439453
{'Accuracy': 0.5126201923076923}
epoch: 3 step: 312, loss is 1.195136547088623
{'Accuracy': 0.5345552884615384}
epoch: 4 step: 312, loss is 0.9828018546104431
{'Accuracy': 0.5562900641025641}
epoch: 5 step: 312, loss is 1.2179136276245117
{'Accuracy': 0.625}
……
epoch: 95 step: 312, loss is 0.3170055150985718
{'Accuracy': 0.9333934294871795}
epoch: 96 step: 312, loss is 0.5585994720458984
{'Accuracy': 0.9339943910256411}
epoch: 97 step: 312, loss is 0.41265806555747986
{'Accuracy': 0.9363982371794872}
epoch: 98 step: 312, loss is 0.13842231035232544
{'Accuracy': 0.9247796474358975}
```

```
epoch: 99 step: 312, loss is 0.48193466663360596
{'Accuracy': 0.9287860576923077}
epoch: 100 step: 312, loss is 0.5555638074874878
{'Accuracy': 0.9295873397435898}
```

编写代码，将训练结果绘制成训练曲线，如下所示：



设置测试集参数并测试，准确度为：

```
test results: {'Accuracy': 0.9743189102564102}
```

图片类别预测与可视化，有一个预测是错误的，结果如下：

结论：**在每一层网络层后都加入BN层后，模型在训练集和测试集上的正确率进一步明显提高。**因为BN层能够让加速网络的收敛速度，同时BN让网络训练变得更容易。此外调参过程也变得简单，对于初始化要求没那么高，而且可以使用大的学习率等，而没有使用BN的话，更大的学习率就可能导致训练发散，大学习率又反过来作用到训练速度上，加速了收敛速度，两者相辅相成。