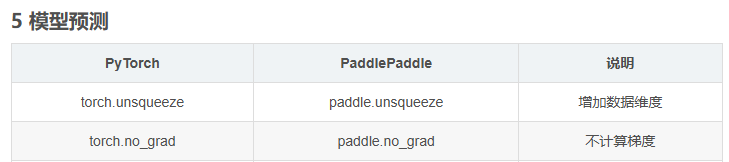
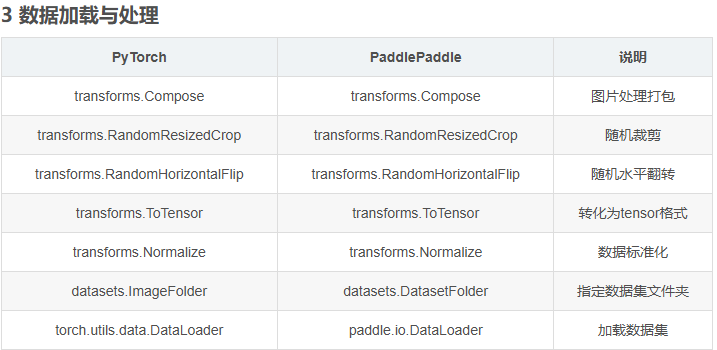
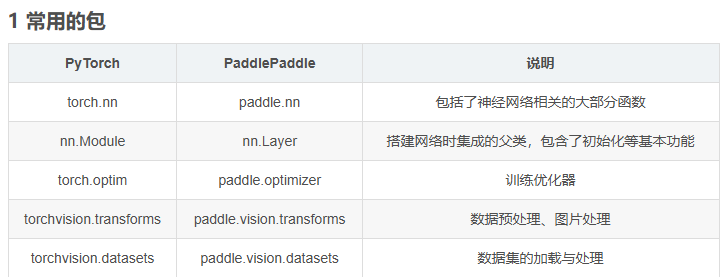
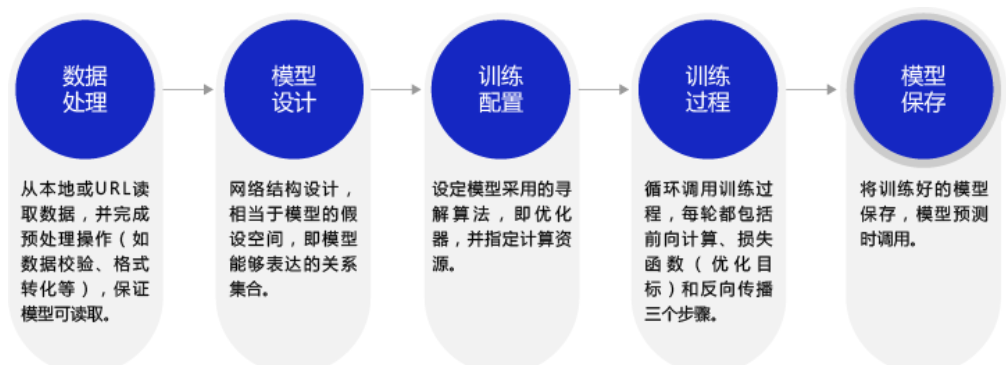
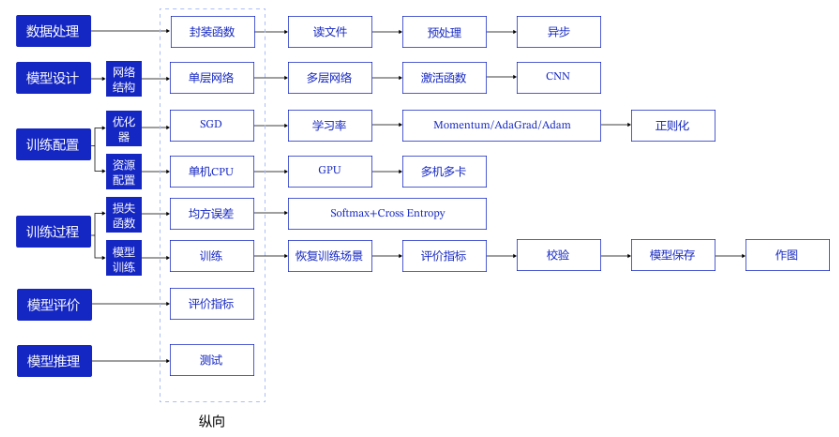
PaddlePaddle与PyTorch的转换







# 数字识别建模实例

在数据处理前，首先要加载飞桨平台与“手写数字识别”模型相关的类库，实现方法如下

**import** paddle

**from** paddle.nn **import** Linear

**import** paddle.nn.functional **as** F

**import** os

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** gzip

**import** json

## 数据处理

一般涉及如下五个环节:读入数据、划分数据集、生成批次数据、训练样本集乱序、校验数据有效性

常见数据集：

# 设置数据读取器，API自动读取MNIST数据训练集

train\_dataset = paddle.vision.datasets.MNIST(mode='train')

train\_data\_0 = np.array(train\_dataset[0][0])

train\_label\_0 = np.array(train\_dataset[0][1])

# 显示第一batch的第一个图像import matplotlib.pyplot as plt

plt.figure("Image") # 图像窗口名称

plt.figure(figsize=(2,2))

plt.imshow(train\_data\_0, cmap=plt.cm.binary)

plt.axis('on') # 关掉坐标轴为 off

plt.title('image') # 图像题目

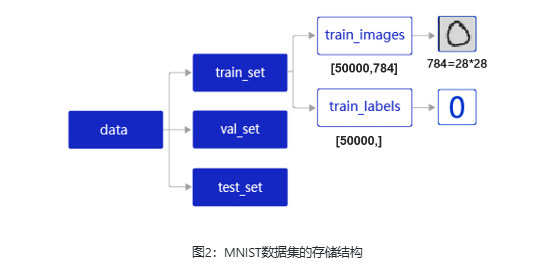
plt.show()

print("图像数据形状和对应数据为:", train\_data\_0.shape)

print("图像标签形状和对应数据为:", train\_label\_0.shape, train\_label\_0)

print("\n打印第一个batch的第一个图像，对应标签数字{}".format(train\_label\_0))

MNIST数据集的存储结构



****data****包含三个元素的列表：训练集train\_set、验证集val\_set、测试集test\_set，分别为50 000条训练样本、10 000条验证样本和10 000条测试样本。每条样本数据都包含手写数字的图像和对应的标签。

****train\_set****包含两个元素的列表：train\_images、train\_labels。

* ****train\_images****：[50 000, 784]的二维列表，包含50 000张图片。每张图片用一个长度为784的向量表示，内容是28×2828×28像素的灰度值（黑白图片）。
* ****train\_labels****：[50 000, ]的列表，表示这些图片对应的分类标签，即0~9之间的一个数字。

## **2.**模型设计

# 定义mnist数据识别网络结构，同房价预测网络

class MNIST(paddle.nn.Layer):

def \_\_init\_\_(self):

super(MNIST, self).\_\_init\_\_()

# 定义一层全连接层，输出维度是1

self.fc = paddle.nn.Linear(in\_features=784, out\_features=1)

# 定义网络结构的前向计算过程

def forward(self, inputs):

outputs = self.fc(inputs)

return outputs

## **3.**训练配置

# 声明网络结构

model = MNIST()

def train(model):

# 启动训练模式

model.train()

# 加载训练集 batch\_size 设为 16

train\_loader=paddle.io.DataLoader(paddle.vision.datasets.MNIST(mode='train'),batch\_size=16,shuffle=True)

# 定义优化器，使用随机梯度下降SGD优化器，学习率设置为0.001

opt=paddle.optimizer.SGD(learning\_rate=0.001,parameters=model.parameters())

## **4.**训练过程

# 图像归一化函数，将数据范围为[0, 255]的图像归一化到[0, 1]

def norm\_img(img):

# 验证传入数据格式是否正确，img的shape为[batch\_size, 28, 28]

assert len(img.shape) == 3

batch\_size, img\_h, img\_w = img.shape[0], img.shape[1], img.shape[2]

# 归一化图像数据

img = img / 255

# 将图像形式reshape为[batch\_size, 784]

img = paddle.reshape(img, [batch\_size, img\_h\*img\_w])

return img

**import** paddle

*# 确保从paddle.vision.datasets.MNIST中加载的图像数据是np.ndarray类型*

paddle.vision.set\_image\_backend('cv2')

*# 声明网络结构*

model = MNIST()

**def** **train**(model):

*# 启动训练模式*

model.train()

*# 加载训练集 batch\_size 设为 16*

train\_loader=paddle.io.DataLoader(paddle.vision.datasets.MNIST(mode='train'),

batch\_size=16,

shuffle=True)

*# 定义优化器，使用随机梯度下降SGD优化器，学习率设置为0.001*

opt=paddle.optimizer.SGD(learning\_rate=0.001,parameters=model.parameters())

EPOCH\_NUM = 10

loss\_list = []

**for** epoch **in** range(EPOCH\_NUM):

**for** batch\_id, data **in** enumerate(train\_loader()):

images = norm\_img(data[0]).astype('float32')

labels = data[1].astype('float32')

*#前向计算的过程*

predicts = model(images)

*# 计算损失*

loss = F.square\_error\_cost(predicts, labels)

avg\_loss = paddle.mean(loss)

*#每训练了1000批次的数据，打印下当前Loss的情况*

**if** batch\_id % 1000 == 0:

loss = avg\_loss.numpy()[0]

loss\_list.append(loss)

print("epoch\_id: {}, batch\_id: {}, loss is: {}".format(epoch, batch\_id, loss))

*#后向传播，更新参数的过程*

avg\_loss.backward()

opt.step()

opt.clear\_grad()

**return** loss\_list

loss\_list = train(model)

paddle.save(model.state\_dict(), './mnist.pdparams')

**绘制损失函数变化曲线：**

**def** **plot**(loss\_list):

plt.figure(figsize=(10,5))

freqs = [i **for** i **in** range(len(loss\_list))]

*# 绘制训练损失变化曲线*

plt.plot(freqs, loss\_list, color='#e4007f', label="Train loss")

*# 绘制坐标轴和图例*

plt.ylabel("loss", fontsize='large')

plt.xlabel("freq", fontsize='large')

plt.legend(loc='upper right', fontsize='x-large')

plt.show()

plot(loss\_list)

## 模型测试

在模型测试之前，需要先从’./work/example\_0.png’文件中读取样例图片，并进行归一化处理。

*# 导入图像读取第三方库*

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**from** PIL **import** Image

img\_path = './work/example\_0.jpg'

*# 读取原始图像并显示*

im = Image.open('./work/example\_0.jpg')

plt.imshow(im)

plt.show()

*# 将原始图像转为灰度图*

im = im.convert('L')

print('原始图像shape: ', np.array(im).shape)

*# 使用Image.ANTIALIAS方式采样原始图片*

im = im.resize((28, 28), Image.ANTIALIAS)

plt.imshow(im)

plt.show()

print("采样后图片shape: ", np.array(im).shape)

*# 读取一张本地的样例图片，转变成模型输入的格式***def** **load\_image**(img\_path):

*# 从img\_path中读取图像，并转为灰度图*

im = Image.open(img\_path).convert('L')

*# print(np.array(im))*

im = im.resize((28, 28), Image.ANTIALIAS)

im = np.array(im).reshape(1, -1).astype(np.float32)

*# 图像归一化，保持和数据集的数据范围一致*

im = 1 - im / 255

**return** im

*# 定义预测过程*

model = MNIST()

params\_file\_path = 'mnist.pdparams'

img\_path = './work/example\_0.jpg'

*# 加载模型参数*

param\_dict = paddle.load(params\_file\_path)

model.load\_dict(param\_dict)

*# 灌入数据*

model.eval()

tensor\_img = load\_image(img\_path)

result = model(paddle.to\_tensor(tensor\_img))

print('result',result)

*# 预测输出取整，即为预测的数字，打印结果*

print("本次预测的数字是", result.numpy().astype('int32'))