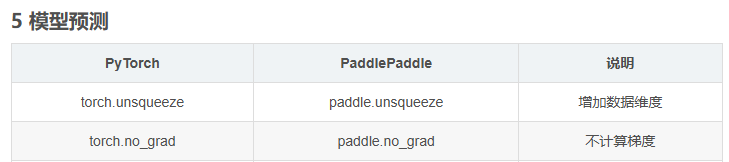
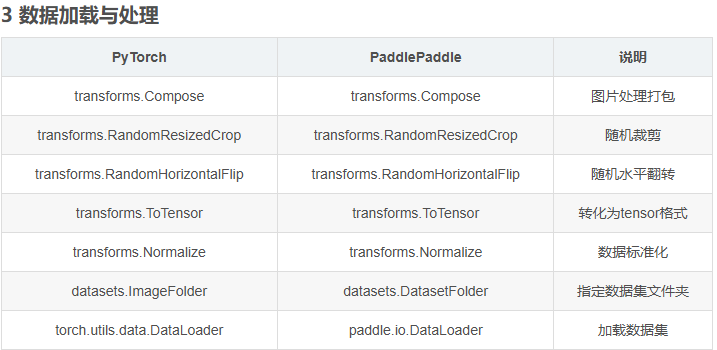
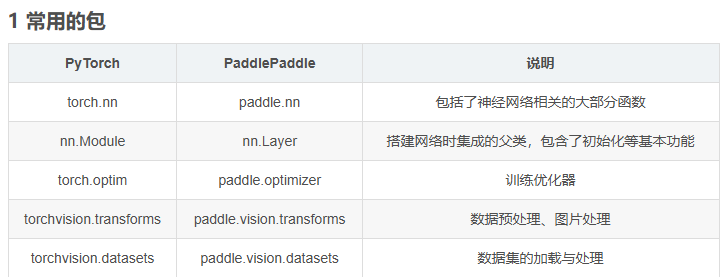
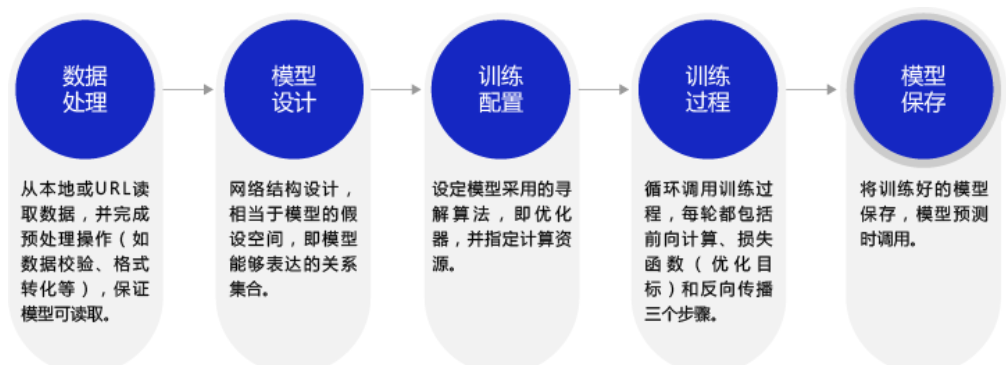
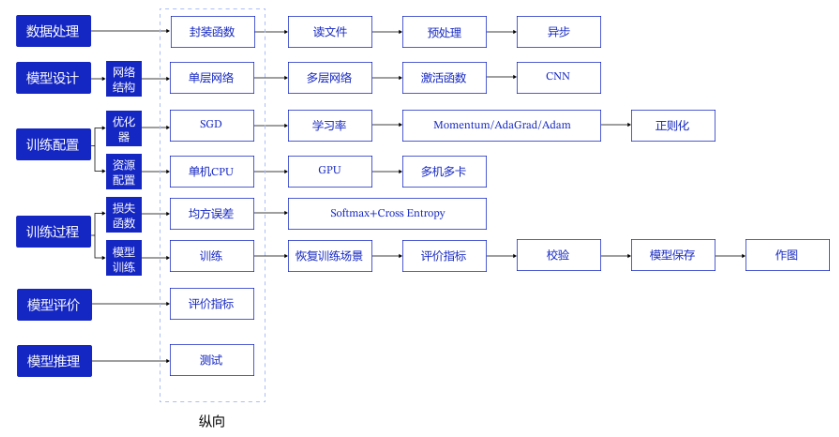
PaddlePaddle与PyTorch的转换







# 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

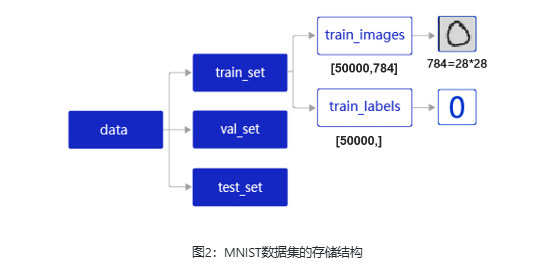
plt.show()

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))

Eg：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。

### 读入数据并划分数据集

在本地./work/目录下读取文件名称为mnist.json.gz的MNIST数据，并拆分成训练集、验证集和测试集，代码实现如下。

*# 声明数据集文件位置*

datafile = './work/mnist.json.gz'

print('loading mnist dataset from {} ......'.format(datafile))

*# 加载json数据文件*

data = json.load(gzip.open(datafile))

print('mnist dataset load done')

*# 读取到的数据区分训练集，验证集，测试集*

train\_set, val\_set, eval\_set = data

*# 观察训练集数据*

imgs, labels = train\_set[0], train\_set[1]

print("训练数据集数量: ", len(imgs))

*# 观察验证集数量*

imgs, labels = val\_set[0], val\_set[1]

print("验证数据集数量: ", len(imgs))

*# 观察测试集数量*

imgs, labels = val= eval\_set[0], eval\_set[1]print("测试数据集数量: ", len(imgs))print(len(imgs[0]))

### 训练样本乱序、生成批次数据

imgs, labels = train\_set[0], train\_set[1]

print("训练数据集数量: ", len(imgs))

*# 获得数据集长度*

imgs\_length = len(imgs)

*# 定义数据集每个数据的序号，根据序号读取数据*

index\_list = list(range(imgs\_length))

*# 读入数据时用到的批次大小*

BATCHSIZE = 100

*# 随机打乱训练数据的索引序号*

random.shuffle(index\_list)

*# 定义数据生成器，返回批次数据*

**def** **data\_generator**():

imgs\_list = []

labels\_list = []

**for** i **in** index\_list:

*# 将数据处理成希望的类型*

img = np.array(imgs[i]).astype('float32')

label = np.array(labels[i]).astype('float32')

imgs\_list.append(img)

labels\_list.append(label)

**if** len(imgs\_list) == BATCHSIZE:

*# 获得一个batchsize的数据，并返回*

**yield** np.array(imgs\_list), np.array(labels\_list)

*# 清空数据读取列表*

imgs\_list = []

labels\_list = []

*# 如果剩余数据的数目小于BATCHSIZE，*

*# 则剩余数据一起构成一个大小为len(imgs\_list)的mini-batch*

**if** len(imgs\_list) > 0:

**yield** np.array(imgs\_list), np.array(labels\_list)

**return** data\_generator

*# 声明数据读取函数，从训练集中读取数据*

train\_loader = data\_generator

*# 以迭代的形式读取数据*

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

image\_data, label\_data = data

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

*# 打印数据shape和类型*

print("打印第一个batch数据的维度:")

print("图像维度: {}, 标签维度: {}".format(image\_data.shape, label\_data.shape))

**break**

### 校验数据有效性

机器校验：

imgs\_length = len(imgs)

**assert** len(imgs) == len(labels), \

"length of train\_imgs({}) should be the same as train\_labels({})".format(len(imgs), len(labels))

人工校验：

*# 声明数据读取函数，从训练集中读取数据*

train\_loader = data\_generator*# 以迭代的形式读取数据***for** batch\_id, data **in** enumerate(train\_loader()):

image\_data, label\_data = data

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

*# 打印数据shape和类型*

print("打印第一个batch数据的维度，以及数据的类型:")

print("图像维度: {}, 标签维度: {}, 图像数据类型: {}, 标签数据类型: {}".format(image\_data.shape, label\_data.shape, type(image\_data), type(label\_data)))

**break**

### 封装数据读取与处理函数

### 基础的数据增强/增广方法和实践

#### 亮度调整

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

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

**from** paddle.vision.transforms **import** functional **as** F

img\_path = "/home/aistudio/work/cat.jpg"

image=Image.open(img\_path)

*# adjust\_brightness对输入图像进行亮度值调整*

new\_img = F.adjust\_brightness(image, 0.4)

*# 显示图像*

display(image.resize((300,400)))

display(new\_img.resize((300,400)))

#### **色调调整**

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

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

**from** paddle.vision.transforms **import** functional **as** F

img\_path = "/home/aistudio/work/cat.jpg"

image=Image.open(img\_path)

*# adjust\_hue对输入图像进行色调的调整*

F.adjust\_hue(image, 0.1)

*# 显示图像*

display(image.resize((300,400)))

display(new\_img.resize((300,400)))

#### **随机旋转**

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

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

**from** paddle.vision.transforms **import** RandomRotation

img\_path = "/home/aistudio/work/cat.jpg"

image=Image.open(img\_path)

*# RandomRotation依据90度，按照均匀分布随机产生一个角度对图像进行旋转*

transform = RandomRotation(90)

new\_img = transform(image)

*# 显示图像*

display(image.resize((300,400)))

display(new\_img.resize((300,400)))

## **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

### 经典的全连接神经网络

[paddle.summary(net, input\_size, dtypes=None)](https://www.paddlepaddle.org.cn/documentation/docs/zh/api/paddle/summary_cn.html" \l "summary" \t "https://www.paddlepaddle.org.cn/tutorials/projectdetail/_blank)函数能够打印网络的基础结构和参数信息。

### 卷积神经网络

定义卷积层

## **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)

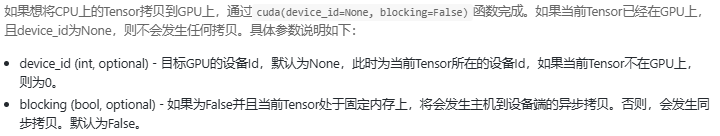
# 定义优化器，使用随机梯度下降SGD优化器，学习率设置为0.001

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

四种比较成熟的优化算法：SGD、Momentum、AdaGrad和Adam

[paddle.device.get\_device](https://www.paddlepaddle.org.cn/documentation/docs/zh/api/paddle/device/get_device_cn.html" \l "get-device" \t "https://www.paddlepaddle.org.cn/tutorials/projectdetail/_blank)功能返回当前程序运行的全局设备

paddle.device.set\_device （‘device’），设置在GPU上训练还是CPU上训练。



## **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')

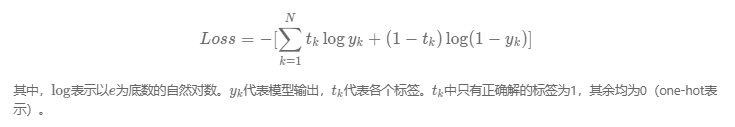
plt.xlabel("freq", fontsize='large')

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

plt.show()

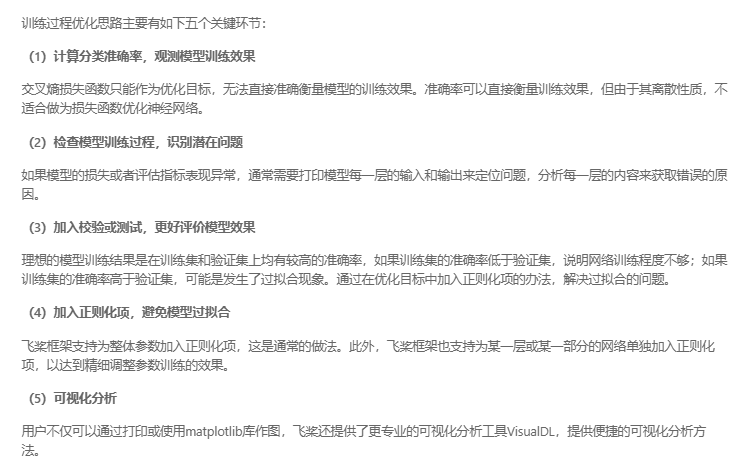
plot(loss\_list)

## 损失函数——交叉熵函数(常用于分类问题)



loss = F.cross\_entropy(predicts, labels)

全连接神经网络、卷积神经网络，在模型的最后阶段，都是使用Softmax进行处理。



## 计算模型的分类准确率

paddle.metric.Accuracy输入参数input为预测的分类结果predict，输入参数label为数据真实的label



## 检查模型训练过程，识别潜在训练问题

使用check\_shape变量控制是否打印“尺寸”，验证网络结构是否正确。使用check\_content变量控制是否打印“内容值”，验证数据分布是否合理。

## 加入校验或测试，更好评价模型效果

* ****训练集**** ：用于训练模型的参数，即训练过程中主要完成的工作。
* ****验证集**** ：用于对模型超参数的选择，比如网络结构的调整、正则化项权重的选择等。
* ****测试集**** ：用于模拟模型在应用后的真实效果。因为测试集没有参与任何模型优化或参数训练的工作，所以它对模型来说是完全未知的样本。在不以校验数据优化网络结构或模型超参数时，校验数据和测试数据的效果是类似的，均更真实的反映模型效果。



## 加入正则化项，避免模型过拟合

过拟合的原因是模型过于敏感，而训练数据量太少或其中的噪音太多。

### （1）正则化项

为所有参数加上统一的正则化项,仅在优化器中设置weight\_decay参数即可实现。使用参数coeff调节正则化项的权重，权重越大时，对模型复杂度的惩罚越高。

opt = paddle.optimizer.Adam(learning\_rate=0.01, weight\_decay=paddle.regularizer.L2Decay(coeff=1e-5), parameters=model.parameters())

### 暂退法Dropout

[paddle.nn.functional.dropout](https://www.paddlepaddle.org.cn/documentation/docs/zh/api/paddle/nn/functional/dropout_cn.html" \l "dropout" \t "https://www.paddlepaddle.org.cn/tutorials/projectdetail/_blank) 可以实现Dropout操作。其输入是一个张量，即需要丢弃数据的神经元参数。该API的axis参数用于指定对输入张量进行Dropout操作的轴。默认为None，即对输入张量x中的任意元素，以丢弃概率p随机将一些元素输出置0。若参数axis不为None，则以一定的概率从图像特征或语音序列中丢弃掉整个通道。



## 可视化分析

将模型训练效果等数据可视化作图可选用两种工具：Matplotlib库和VisualDL

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





## 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'))