手写数字识别与夸克、胶子喷注分类

庞龙刚@华中师范大学

第十届华大 QCD 讲习班

学习目标

1. 接触成熟的机器学习库 tensorflow，keras
2. 使用 keras 构建简单的手写数字识别代码
3. 使用深度神经网络分类 Quark 与 Gluon 喷注

学习内容

1. 简单的手写数字识别，内容基于 keras 例子代码：
   https://keras.io/examples/vision/mnist_convnet/
2. 将此卷积神经网络 (convolution neural network) 用作夸克胶子喷注分类任务
3. 搭建一个简单的点云神经网络，用作夸克胶子喷注分类

In [1]:
```
import numpy as np
import pandas as pd
from tqdm import tqdm
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import mplhep as hep

hep.styles.use("ATLAS")
```

手写数字识别

准备数据

手写数字识别任务：深度学习界的 “hello world”

目标：对手写数字 0 到 9 共 10 个数字进行识别

训练样本：60000 个

测试样本：10000 个

每个样本： 28 * 28 个像素， 一个标签（0-9）

In [2]:
```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

```
# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

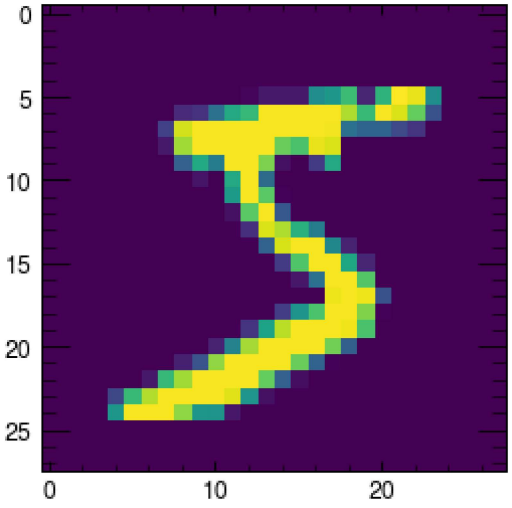
```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

In [3]:
```
x_train.shape
#plt.imshow(x_train[])
```

Out[3]:
```
(60000, 28, 28, 1)
```

In [4]:
```
# 观察第一个样本及其标签
plt.imshow(x_train[0, :, :, 0])
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x23ea8600460>
```



In [5]:
```
# to_categorical 函数将 5 变成 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
# 这种表示称为 one-hot 表示， 用于多个类别的分类任务

print(y_train[0])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

构建卷积神经网络

这里使用 keras 中的 Sequential 函数，将输入、隐藏层、输出层 list 转换为神经网络。

In [20]:
```
def create_model(input_shape, num_classes):
    model = keras.Sequential(
        [
            keras.Input(shape=input_shape),
```

```
            layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation="softmax"),
        ]
    )
    return model


model = create_model(input_shape=(28, 28, 1), num_classes=10)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

训练卷积神经网络

设定几个 callback （回调函数），功能

1. csv_logger 记录每个 epoch 的训练精度和验证精度
2. reduce_lr 动态调节学习率，阶梯状衰减
3. save_best 保存验证误差最小时的模型参数，实现早停，防止过拟合时 validation error 逐渐增大

In [7]:

```
def mycallbacks(version=1):
    '''生成3个回调函数，用于记录训练进程、动态调节学习率、早停
    :version: 模型版本，影响日志与模型参数文件保存路径'''
    csv_logger = keras.callbacks.CSVLogger("training_v%s.log"%version)

    reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                factor=0.2, patience=20, min_lr=0.000001)

    save_best =  keras.callbacks.ModelCheckpoint(
        filepath="model_v%s.hdf5"%version,
        verbose=1, save_best_only=True)

    return [csv_logger, reduce_lr, save_best]
```

In [8]:
```
# 分类一般使用交叉熵损失函数 cross entropy
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"

model.fit(x_train, y_train,
          batch_size=32,           # 将训练样本按 每 32 个作为一个小批次（mini batch)
          epochs=10,               # 神经网络遍历所有的训练数据 10 次（每个样本使用了
          validation_split=0.1,    # 使用 90% 的数据训练，10% 的数据验证
          callbacks=mycallbacks(version=1)
         )
```

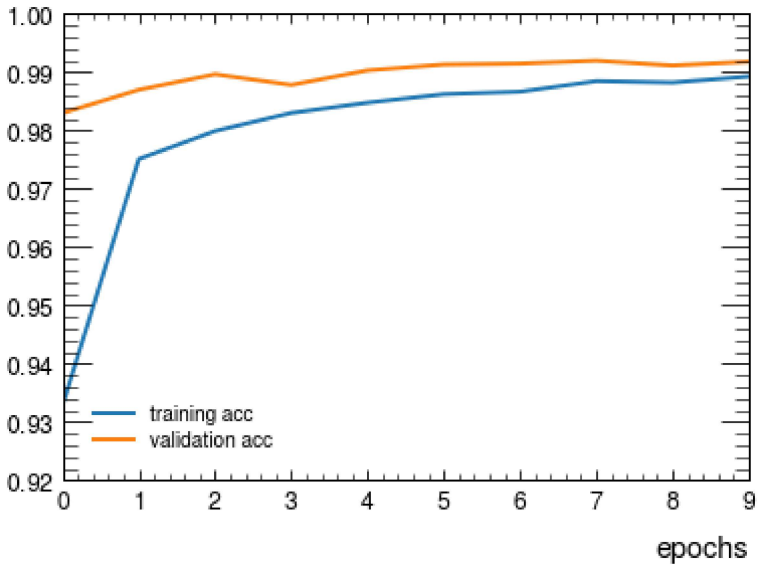
```
Epoch 1/10
1681/1688 [============================>.] - ETA: 0s - loss: 0.2177 - accuracy: 0.93
27
Epoch 00001: val_loss improved from inf to 0.05295, saving model to model_v1.hdf5
1688/1688 [==============================] - 10s 6ms/step - loss: 0.2174 - accuracy:
0.9329 - val_loss: 0.0530 - val_accuracy: 0.9830 - lr: 0.0010
Epoch 2/10
1688/1688 [==============================] - ETA: 0s - loss: 0.0816 - accuracy: 0.97
51
Epoch 00002: val_loss improved from 0.05295 to 0.04637, saving model to model_v1.hdf
5
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0816 - accuracy:
0.9751 - val_loss: 0.0464 - val_accuracy: 0.9870 - lr: 0.0010
Epoch 3/10
1683/1688 [============================>.] - ETA: 0s - loss: 0.0644 - accuracy: 0.98
00
Epoch 00003: val_loss improved from 0.04637 to 0.03636, saving model to model_v1.hdf
5
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0644 - accuracy:
0.9799 - val_loss: 0.0364 - val_accuracy: 0.9897 - lr: 0.0010
Epoch 4/10
1681/1688 [============================>.] - ETA: 0s - loss: 0.0545 - accuracy: 0.98
30
Epoch 00004: val_loss did not improve from 0.03636
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0546 - accuracy:
0.9830 - val_loss: 0.0393 - val_accuracy: 0.9878 - lr: 0.0010
Epoch 5/10
1685/1688 [============================>.] - ETA: 0s - loss: 0.0480 - accuracy: 0.98
48
Epoch 00005: val_loss improved from 0.03636 to 0.03146, saving model to model_v1.hdf
5
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0480 - accuracy:
0.9848 - val_loss: 0.0315 - val_accuracy: 0.9903 - lr: 0.0010
Epoch 6/10
1683/1688 [============================>.] - ETA: 0s - loss: 0.0424 - accuracy: 0.98
62
Epoch 00006: val_loss improved from 0.03146 to 0.03032, saving model to model_v1.hdf
5
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0425 - accuracy:
0.9862 - val_loss: 0.0303 - val_accuracy: 0.9913 - lr: 0.0010
Epoch 7/10
1688/1688 [==============================] - ETA: 0s - loss: 0.0406 - accuracy: 0.98
67
Epoch 00007: val_loss did not improve from 0.03032
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0406 - accuracy:
0.9867 - val_loss: 0.0303 - val_accuracy: 0.9915 - lr: 0.0010
Epoch 8/10
1680/1688 [============================>.] - ETA: 0s - loss: 0.0366 - accuracy: 0.98
85
Epoch 00008: val_loss improved from 0.03032 to 0.02955, saving model to model_v1.hdf
5
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0366 - accuracy:
0.9885 - val_loss: 0.0295 - val_accuracy: 0.9920 - lr: 0.0010
Epoch 9/10
1684/1688 [============================>.] - ETA: 0s - loss: 0.0355 - accuracy: 0.98
83
Epoch 00009: val_loss did not improve from 0.02955
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0355 - accuracy:
0.9882 - val_loss: 0.0296 - val_accuracy: 0.9912 - lr: 0.0010
Epoch 10/10
1683/1688 [============================>.] - ETA: 0s - loss: 0.0345 - accuracy: 0.98
93
Epoch 00010: val_loss improved from 0.02955 to 0.02790, saving model to model_v1.hdf
5
```

```
1688/1688 [==============================] - 11s 6ms/step - loss: 0.0346 - accuracy: 0.9893 - val_loss: 0.0279 - val_accuracy: 0.9918 - lr: 0.0010
```

Out[8]:
```
<tensorflow.python.keras.callbacks.History at 0x23ea8d57820>
```

In [9]:
```
def check_performance(version):
    history = pd.read_csv("training_v%s.log"%version)
    plt.plot(history["epoch"], history["accuracy"], label="training acc")
    plt.plot(history["epoch"], history["val_accuracy"], label="validation acc")
    plt.legend(loc='best')
    plt.xlabel("epochs")
```

In [10]:
```
check_performance(version=1)
```



夸克胶子喷注分类 (Quark Gluon Jet classification)

数据集： https://zenodo.org/record/3164691#.YGwuCOj7Q2w

下载：QG_jets.npz

保存位置：当前目录/data/QG_jets.npz

根据夸克与胶子碎裂成的末态强子，判断这些强子来自于夸克还是胶子

```
There are 20 files in each dataset, each in compressed NumPy
format. Files including charm and bottom jets have 'withbc' in
their filename. There are two arrays in each file

X: (100000,M,4), exactly 50k quark and 50k gluon jets, randomly
sorted, where M is the max multiplicity of the jets in that file
(other jets have been padded with zero-particles), and the
features of each particle are its pt, rapidity, azimuthal angle,
and pdgid.
y: (100000,), an array of labels for the jets where gluon is 0 and
quark is 1.
If you use this dataset, please cite this Zenodo record as well as
the corresponding paper:
```

P. T. Komiske, E. M. Metodiev, J. Thaler, Energy Flow Networks: Deep Sets for Particle Jets, JHEP 01 (2019) 121, arXiv:1810.05165.

读入数据 （Know your data)

In [11]:
```
with np.load("data/QG_jets.npz") as dat:
    x_qg = dat["X"]
    y_qg = dat["y"]
```

In [12]:
```
# 139 为喷注中最大粒子数量， 4 表示存储了 (pt, rapidity, phi, pdgid) 信息
# 数据中如果一个喷注总的粒子数小于 139， 用 (0, 0, 0, 0) 补全
x_qg.shape
```

Out[12]:
```
(100000, 139, 4)
```

横动量 p_t, 快度(rapidity，一般用 y 表示) 以及方位角 ϕ 是对粒子四动量 (E, p_x, p_y, p_z) 的另一种表示。

$$p_t = \sqrt{p_x^2 + p_y^2} \tag{1}$$

$$\phi = \arctan2(p_y, p_x) \tag{2}$$

$$y = \frac{1}{2}\ln\frac{E + p_z}{E - p_z} \tag{3}$$

In [13]:
```
y_qg.shape
```

Out[13]:
```
(100000,)
```

In [14]:
```
# 输出训练样本 0 的前三行数据：pt, rapidity, phi, pdgid
# pdgid=22 表示 光子， pdgid = -211 表示 pion- 介子

print(x_qg[0, 0:3, :])
```

```
[[ 2.68769142e-01  3.56903171e-01  4.74138734e+00  2.20000000e+01]
 [ 1.60076377e-01 -2.55609533e-01  4.55022910e+00  2.20000000e+01]
 [ 1.14868731e+00 -6.24380156e-02  4.50385377e+00 -2.11000000e+02]]
```

In [15]:
```
# 标签: 0 for gluon; 1 for quark
y_qg[:20]
```

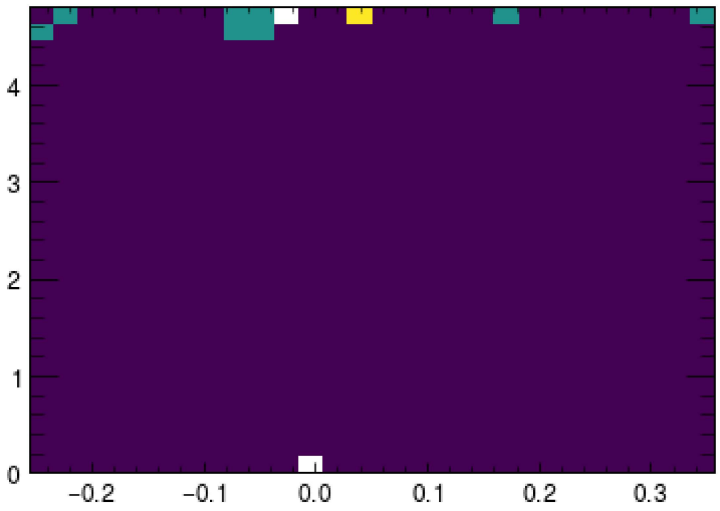
Out[15]:
```
array([1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0.,
       0., 1., 0.])
```

In [16]:
```
y_qg_onehot = keras.utils.to_categorical(y_qg, 2)
```

数据可视化 （Know your data)

In [17]:
```
def visualize(eid=0):
    '''对第 eid 个样本进行可视化，探索数据'''
    dat = x_qg[eid]
    pt = dat[:, 0]
    rapidity = dat[:, 1]
    phi = dat[:, 2]
    plt.hist2d(rapidity, phi,
               bins=28, cmin=0, cmax=5)
    # 这里为了可视化将 count>cmax 的像素设为 nan
```

In [18]:
```
visualize(eid=0)
```



数据预处理

In [22]:
```
xmin, xmax = -1.7, 1.7
ymin, ymax = 0,  2*np.pi
# 将所有图像都对齐到 |y| < 1.7, phi in [0, 2pi] 范围
ranges = [[xmin, xmax], [ymin, ymax]]

def to_img(eid=0, bins=28):
    dat = x_qg[eid]
    rapidity = dat[:, 1]
    phi = dat[:, 2]
    # 将 (eta, phi) 分布转化为 2D 图像
    img, _, _ = np.histogram2d(rapidity, phi,
                    bins = bins,
                    range=ranges,
                    normed=True)

    return img
```

In [23]:
```
num_events = len(x_qg)

x_qg_imgs = [to_img(i) for i in tqdm(range(num_events))]
```

```
100%|██████████| 100000/100000 [00:18<00:00, 5440.50it/s]
```

In [24]:
```
# x_qg_imgs 中每张图的最大值远大于 1,
# 使用所有事例中, 2d image 中像素最大的值进行缩放
x_qg_imgs = np.array(x_qg_imgs)
```

In [25]:
```
x_qg_imgs[0].max()
```

Out[25]:
```
31.946835635779856
```

In [26]:
```
x_qg_imgs = x_qg_imgs / x_qg_imgs.max()
```

In [27]:
```
x_qg_imgs[0].max()
```

Out[27]:
```
0.8768115942028987
```

In [28]:
```
x_qg_imgs.shape
```

Out[28]:
```
(100000, 28, 28)
```

In [29]:
```
# 为了使用 mnist 所示卷积神经网络，将数据增加一个维度
x_qg_imgs = x_qg_imgs.reshape(100000, 28, 28, -1)
```

In [30]:
```
y_qg_onehot[0]
```

Out[30]:
```
array([0., 1.], dtype=float32)
```

将数据分成训练集与测试集

这里我们使用 scikit-learn 中的

sklearn.model_selection.train_test_split

将数据集划分成 训练集 与 测试集。

In [31]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
x_train, x_test, y_train, y_test = train_test_split(x_qg_imgs, y_qg_onehot, test_siz
```

In [33]:
```
x_train.shape
```

Out[33]:
```
(90000, 28, 28, 1)
```

训练卷积神经网络进行夸克胶子喷注分类

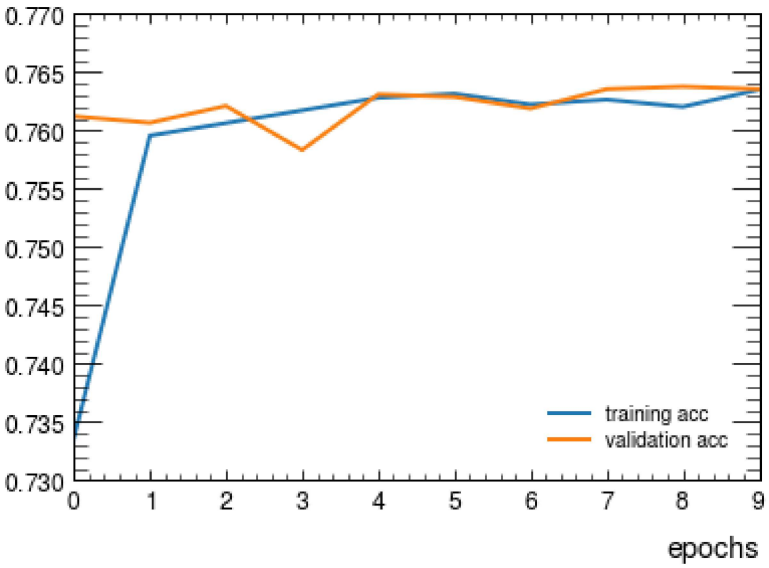
In [34]:
```
# 构造卷积神经网络，进行夸克胶子喷注分类
# 2 分类任务，将 num_classes 设置为 2
qg_model = create_model(input_shape=(28, 28, 1), num_classes=2)
qg_model.summary()

# 分类一般使用交叉熵损失函数 cross entropy
qg_model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accura

qg_model.fit(x_train, y_train,
          batch_size=32,           # 将训练样本按 每 32 个作为一个小批次（mini batch)
          epochs=10,               # 神经网络遍历所有的训练数据 10 次（每个样本使用了
          validation_split=0.1,    # 使用 90% 的数据训练，10% 的数据验证
          callbacks=mycallbacks(version=2)
         )
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1600)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1600)              0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 3202
=================================================================
Total params: 22,018
Trainable params: 22,018
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
2524/2532 [============================>.] - ETA: 0s - loss: 0.5407 - accuracy: 0.73
35
Epoch 00001: val_loss improved from inf to 0.50198, saving model to model_v2.hdf5
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5407 - accuracy:
0.7335 - val_loss: 0.5020 - val_accuracy: 0.7612 - lr: 0.0010
Epoch 2/10
2530/2532 [============================>.] - ETA: 0s - loss: 0.5082 - accuracy: 0.75
96
Epoch 00002: val_loss improved from 0.50198 to 0.50002, saving model to model_v2.hdf
5
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5082 - accuracy:
0.7596 - val_loss: 0.5000 - val_accuracy: 0.7607 - lr: 0.0010
Epoch 3/10
2524/2532 [============================>.] - ETA: 0s - loss: 0.5058 - accuracy: 0.76
05
Epoch 00003: val_loss did not improve from 0.50002
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5056 - accuracy:
0.7606 - val_loss: 0.5000 - val_accuracy: 0.7621 - lr: 0.0010
Epoch 4/10
2527/2532 [============================>.] - ETA: 0s - loss: 0.5035 - accuracy: 0.76
17
Epoch 00004: val_loss did not improve from 0.50002
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5035 - accuracy:
0.7617 - val_loss: 0.5051 - val_accuracy: 0.7583 - lr: 0.0010
Epoch 5/10
2526/2532 [============================>.] - ETA: 0s - loss: 0.5028 - accuracy: 0.76
28
Epoch 00005: val_loss improved from 0.50002 to 0.49625, saving model to model_v2.hdf
5
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5028 - accuracy:
0.7628 - val_loss: 0.4963 - val_accuracy: 0.7631 - lr: 0.0010
Epoch 6/10
2530/2532 [============================>.] - ETA: 0s - loss: 0.5034 - accuracy: 0.76
31
Epoch 00006: val_loss improved from 0.49625 to 0.49513, saving model to model_v2.hdf
5
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5034 - accuracy:
0.7632 - val_loss: 0.4951 - val_accuracy: 0.7629 - lr: 0.0010
Epoch 7/10
2531/2532 [============================>.] - ETA: 0s - loss: 0.5019 - accuracy: 0.76
22
```

```
Epoch 00007: val_loss did not improve from 0.49513
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5019 - accuracy:
0.7622 - val_loss: 0.4987 - val_accuracy: 0.7619 - lr: 0.0010
Epoch 8/10
2526/2532 [============================>.] - ETA: 0s - loss: 0.5009 - accuracy: 0.76
26
Epoch 00008: val_loss did not improve from 0.49513
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5008 - accuracy:
0.7627 - val_loss: 0.4960 - val_accuracy: 0.7636 - lr: 0.0010
Epoch 9/10
2531/2532 [============================>.] - ETA: 0s - loss: 0.5011 - accuracy: 0.76
20
Epoch 00009: val_loss improved from 0.49513 to 0.49474, saving model to model_v2.hdf
5
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5011 - accuracy:
0.7620 - val_loss: 0.4947 - val_accuracy: 0.7638 - lr: 0.0010
Epoch 10/10
2531/2532 [============================>.] - ETA: 0s - loss: 0.5013 - accuracy: 0.76
35
Epoch 00010: val_loss did not improve from 0.49474
2532/2532 [==============================] - 17s 7ms/step - loss: 0.5013 - accuracy:
0.7635 - val_loss: 0.5003 - val_accuracy: 0.7636 - lr: 0.0010
```

Out[34]:
```
<tensorflow.python.keras.callbacks.History at 0x23ea635c160>
```

In [35]:
```
check_performance(version=2)
```



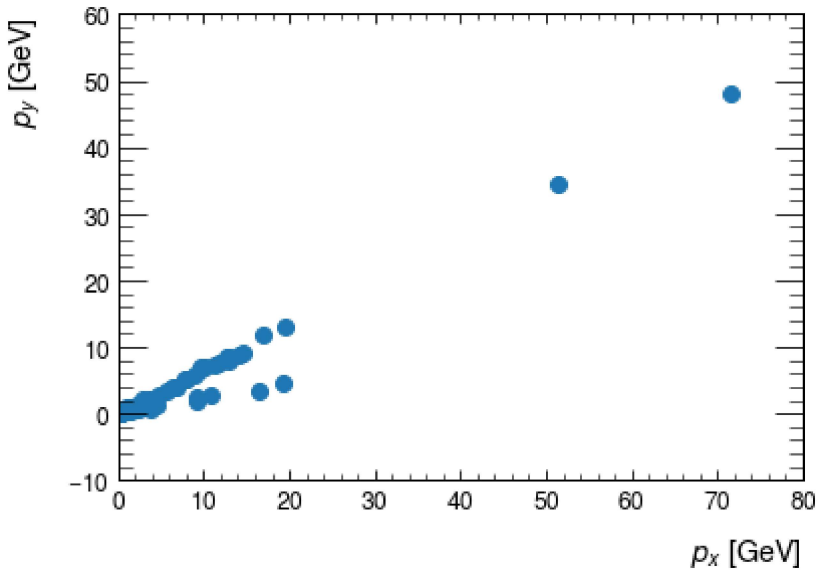
点云网络用于夸克胶子喷注分类

卷积网络需要使用 2d histogram 把制作出粒子分布的图像，会损失一部分输入数据信息。

点云网络可以直接使用所有粒子的四动量以及粒子种类信息，它的输入是每个事件的所有粒子列表。

In [36]:
```
def pointcloud_plot(eid=0):
    dat = x_qg[2]
    pt = dat[:, 0]
    phi = dat[:, 2]
    x = pt * np.cos(phi)
    y = pt * np.sin(phi)
    plt.scatter(x, y)
    plt.xlabel(r"$p_x$ [GeV]")
```

```
    plt.ylabel(r"$p_y$ [GeV]")

pointcloud_plot(eid=5)
```



In [37]:
```
def model_pointcloud(features=4, activ='relu', padding_size=139):
    '''
    :features: 每个粒子的特征个数
    :activ: 激活函数，keras 自带
    :padding_size: 单事件中最大粒子数
    '''
    S_input = layers.Input(shape=(padding_size, features), dtype='float')
    O_seq = layers.Conv1D(256, 1, activation=activ)(S_input)
    O_seq = layers.Conv1D(128, 1, activation=activ)(O_seq)
    O_seq = layers.Conv1D(128, 1, activation=activ)(O_seq)
    O_seq = layers.GlobalMaxPooling1D()(O_seq)
    O_seq = layers.Dense(32, activation=activ)(O_seq)
    O_seq = layers.Dense(2, activation='softmax')(O_seq)

    model = keras.models.Model(inputs=S_input, outputs=O_seq)

    return model
```

In [38]:
```
model3 = model_pointcloud()
```

In [39]:
```
# 分类一般使用交叉熵损失函数 cross entropy
model3.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy

model3.fit(x_qg, y_qg_onehot,
           batch_size=32,           # 将训练样本按 每 32 个作为一个小批次（mini batch）
           epochs=10,               # 神经网络遍历所有的训练数据 10 次（每个样本使用了
           validation_split=0.1,    # 使用 90% 的数据训练，10% 的数据验证
           callbacks=mycallbacks(version=3)
          )
```

```
Epoch 1/10
2811/2813 [============================>.] - ETA: 0s - loss: 1.2952 - accuracy: 0.67
45
Epoch 00001: val_loss improved from inf to 0.60334, saving model to model_v3.hdf5
2813/2813 [==============================] - 29s 10ms/step - loss: 1.2949 - accurac
y: 0.6745 - val_loss: 0.6033 - val_accuracy: 0.6912 - lr: 0.0010
Epoch 2/10
2809/2813 [============================>.] - ETA: 0s - loss: 0.5539 - accuracy: 0.72
74
Epoch 00002: val_loss improved from 0.60334 to 0.54390, saving model to model_v3.hdf
5
2813/2813 [==============================] - 29s 10ms/step - loss: 0.5539 - accurac
y: 0.7274 - val_loss: 0.5439 - val_accuracy: 0.7327 - lr: 0.0010
Epoch 3/10
2809/2813 [============================>.] - ETA: 0s - loss: 0.5476 - accuracy: 0.72
98
Epoch 00003: val_loss did not improve from 0.54390
2813/2813 [==============================] - 29s 10ms/step - loss: 0.5477 - accurac
y: 0.7297 - val_loss: 0.5534 - val_accuracy: 0.7249 - lr: 0.0010
Epoch 4/10
2809/2813 [============================>.] - ETA: 0s - loss: 0.5528 - accuracy: 0.72
70
Epoch 00004: val_loss did not improve from 0.54390
2813/2813 [==============================] - 29s 10ms/step - loss: 0.5528 - accurac
y: 0.7270 - val_loss: 0.5502 - val_accuracy: 0.7265 - lr: 0.0010
Epoch 5/10
2812/2813 [============================>.] - ETA: 0s - loss: 0.5441 - accuracy: 0.73
16
Epoch 00005: val_loss did not improve from 0.54390
2813/2813 [==============================] - 30s 11ms/step - loss: 0.5441 - accurac
y: 0.7316 - val_loss: 0.5510 - val_accuracy: 0.7319 - lr: 0.0010
Epoch 6/10
2810/2813 [============================>.] - ETA: 0s - loss: 0.5432 - accuracy: 0.73
29
Epoch 00006: val_loss improved from 0.54390 to 0.53940, saving model to model_v3.hdf
5
2813/2813 [==============================] - 29s 10ms/step - loss: 0.5433 - accurac
y: 0.7328 - val_loss: 0.5394 - val_accuracy: 0.7357 - lr: 0.0010
Epoch 7/10
2810/2813 [============================>.] - ETA: 0s - loss: 0.5390 - accuracy: 0.73
52
Epoch 00007: val_loss improved from 0.53940 to 0.53935, saving model to model_v3.hdf
5
2813/2813 [==============================] - 29s 10ms/step - loss: 0.5390 - accurac
y: 0.7352 - val_loss: 0.5394 - val_accuracy: 0.7357 - lr: 0.0010
Epoch 8/10
2812/2813 [============================>.] - ETA: 0s - loss: 0.5347 - accuracy: 0.73
95
Epoch 00008: val_loss improved from 0.53935 to 0.53800, saving model to model_v3.hdf
5
2813/2813 [==============================] - 29s 10ms/step - loss: 0.5347 - accurac
y: 0.7395 - val_loss: 0.5380 - val_accuracy: 0.7363 - lr: 0.0010
Epoch 9/10
2813/2813 [==============================] - ETA: 0s - loss: 0.5299 - accuracy: 0.74
14
Epoch 00009: val_loss improved from 0.53800 to 0.53098, saving model to model_v3.hdf
5
2813/2813 [==============================] - 30s 11ms/step - loss: 0.5299 - accurac
y: 0.7414 - val_loss: 0.5310 - val_accuracy: 0.7443 - lr: 0.0010
Epoch 10/10
2810/2813 [============================>.] - ETA: 0s - loss: 0.5255 - accuracy: 0.74
54
Epoch 00010: val_loss did not improve from 0.53098
```

```
2813/2813 [==============================] - 30s 10ms/step - loss: 0.5255 - accurac
y: 0.7454 - val_loss: 0.5327 - val_accuracy: 0.7389 - lr: 0.0010
```

Out[39]: <tensorflow.python.keras.callbacks.History at 0x23ea9ada550>

In [40]:
```
check_performance(version=3)
```