人脸关键点检测

题目描述：

使用（96，96）的灰度图进行 15 个人脸关键点的预测。

数据集描述：

训练集包含人脸图像以及相应的关键点坐标
测试集只包含人脸图像

评测方式：

使用自己编写的模型对训练集数据进行预测，将预测结果使用提供的函数导出到 csv 文件，文件使用**完整的学号**命名。

评测指标：

对提交文件，将使用 RMSE (Root Mean Square Error)作为评价指标,公式如下：

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

实验流程

数据预处理：

1. **数据清洗：处理缺失值、异常值**
   a) 缺失值用上一个观测值填充

b) 检测缺失值：没有任何缺失值

```
In [8]: train_data.isnull().any()
```

```
Out[8]: Unnamed: 0                    False
        left_eye_center_x             False
        left_eye_center_y             False
        right_eye_center_x            False
        right_eye_center_y            False
        left_eye_inner_corner_x       False
        left_eye_inner_corner_y       False
        left_eye_outer_corner_x       False
        left_eye_outer_corner_y       False
        right_eye_inner_corner_x      False
        right_eye_inner_corner_y      False
        right_eye_outer_corner_x      False
        right_eye_outer_corner_y      False
        left_eyebrow_inner_end_x      False
        left_eyebrow_inner_end_y      False
        left_eyebrow_outer_end_x      False
        left_eyebrow_outer_end_y      False
        right_eyebrow_inner_end_x     False
        right_eyebrow_inner_end_y     False
        right_eyebrow_outer_end_x     False
        right_eyebrow_outer_end_y     False
        nose_tip_x                    False
        nose_tip_y                    False
        mouth_left_corner_x           False
        mouth_left_corner_y           False
        mouth_right_corner_x          False
        mouth_right_corner_y          False
        mouth_center_top_lip_x        False
        mouth_center_top_lip_y        False
        mouth_center_bottom_lip_x     False
        mouth_center_bottom_lip_y     False
        Image                         False
        dtype: bool
```

```
In [9]: test_data.isnull().any()
```

```
Out[9]: Unnamed: 0    False
        Image         False
        dtype: bool
```

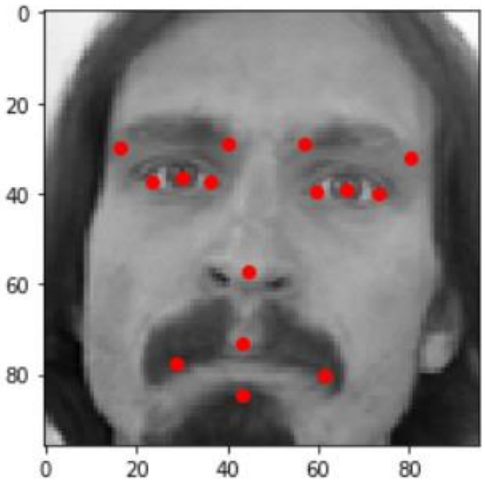
```
In [6]: train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6001 entries, 0 to 6000
Data columns (total 32 columns):
Unnamed: 0                    6001 non-null int64
left_eye_center_x             6001 non-null float64
left_eye_center_y             6001 non-null float64
right_eye_center_x            6001 non-null float64
right_eye_center_y            6001 non-null float64
left_eye_inner_corner_x       6001 non-null float64
left_eye_inner_corner_y       6001 non-null float64
left_eye_outer_corner_x       6001 non-null float64
left_eye_outer_corner_y       6001 non-null float64
right_eye_inner_corner_x      6001 non-null float64
right_eye_inner_corner_y      6001 non-null float64
right_eye_outer_corner_x      6001 non-null float64
right_eye_outer_corner_y      6001 non-null float64
left_eyebrow_inner_end_x      6001 non-null float64
left_eyebrow_inner_end_y      6001 non-null float64
left_eyebrow_outer_end_x      6001 non-null float64
left_eyebrow_outer_end_y      6001 non-null float64
right_eyebrow_inner_end_x     6001 non-null float64
right_eyebrow_inner_end_y     6001 non-null float64
right_eyebrow_outer_end_x     6001 non-null float64
right_eyebrow_outer_end_y     6001 non-null float64
nose_tip_x                    6001 non-null float64
nose_tip_y                    6001 non-null float64
mouth_left_corner_x           6001 non-null float64
mouth_left_corner_y           6001 non-null float64
mouth_right_corner_x          6001 non-null float64
mouth_right_corner_y          6001 non-null float64
mouth_center_top_lip_x        6001 non-null float64
mouth_center_top_lip_y        6001 non-null float64
mouth_center_bottom_lip_x     6001 non-null float64
mouth_center_bottom_lip_y     6001 non-null float64
Image                         6001 non-null object
```

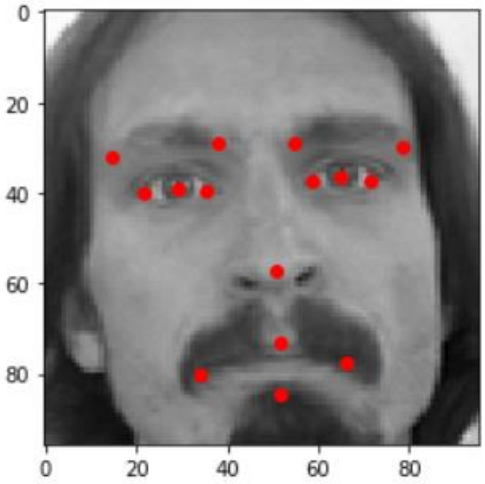
2. **数据增强：**

为了防止过拟合、以及获得更大的训练集，将图片翻转，点沿图片中垂线做轴对称变换，对应标签交换（例如左眼中心和右眼中心坐标交换）。获得两倍的数据集，最终将所有数据混洗。（代码见 DataPreprocessing.ipynb）

```
In [10]: show_result(X_train[0].reshape(96,96),y_train[0])
```




```
In [24]: show_result(X_add[0].reshape(96,96),y_add[0])
```



3. **数据规范化（特征缩放）**
   将像素点除以 255，使其缩放到区间[0, 1]，标签坐标缩放到区间[-1 , 1]。
   原因：使训练更快，避免异常值影响训练

模型设计:

1. CNN

优化器 Adam，earlystop，缩放的数据集，batchsize=256，激活函数 ReLU
三个卷积层，filter 数分别是 32，64，128，选用了较小的卷积核，大小分别是 3x3, 2x2, 2x2，使用 padding 使输出大小等于输入大小
每个卷积层后接一个池化层，`pool_size` 都是 `2x2`，步长为 `2`，不采用 `padding`
池化层之后接 `Dropout=0.1`，防止过拟合

之后是 3 层全连接层，节点数为 1000、1000、30，中间 dropout 0.5

```
model.add(Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding='same', input_shape=(96, 96, 1)))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Dropout(0.1))
# BLOCK 2
model.add(Conv2D(filters=64, kernel_size=(2, 2), strides=(1, 1), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Dropout(0.2))

# BLOCK 1
model.add(Conv2D(filters=128, kernel_size=(2, 2), strides=(1, 1), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Dropout(0.3))

# full connected #私自改了参数 4096个节点pc跑不出来
model.add(Flatten())
model.add(Dense(1000, activation='relu',kernel_initializer=RandomUniform(minval=-0.05, maxval=0.05, seed=None),bias_initializer=Constant(value=0.01)))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='relu',kernel_initializer=RandomUniform(minval=-0.05, maxval=0.05, seed=None),bias_initializer=Constant(value=0.01)))
model.add(Dense(30)) # 最终输出15个2维坐标
```

Model:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 96, 96, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 48, 48, 32)	0
dropout_1 (Dropout)	(None, 48, 48, 32)	0
conv2d_2 (Conv2D)	(None, 48, 48, 64)	8256
max_pooling2d_2 (MaxPooling2	(None, 24, 24, 64)	0
dropout_2 (Dropout)	(None, 24, 24, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	32896
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 128)	0
dropout_3 (Dropout)	(None, 12, 12, 128)	0
flatten_1 (Flatten)	(None, 18432)	0
dense_1 (Dense)	(None, 512)	9437696
dropout_4 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
dense_3 (Dense)	(None, 30)	7710

Total params: 9,618,206
Trainable params: 9,618,206
Non-trainable params: 0
_____

训练结果：

1. 采用缩放后的数据，全连接层节点数：1000、1000、30
   结果：提交后 RMSE=1.8，并且模型文件较大
   分析：全连接层节点太多导致模型较大

2. 采用数据增强后的数据集，数据缩放
   全连接层节点减少为 512、256、30，dropout=0.2，采用 early stop
   结果变差，本地结果：loss=0.0013，val_loss=0.0030
   分析：
   - 全连接层节点数过少
   - 训练轮数过少，节点欠拟合

2. 标准 VGG-16

根据牛津大学的视觉几何组（Visual Geometry Group）和 Google DeepMind 公司的研究员一起研发的的深度卷积神经网络 VGG 和论文《VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION》搭建 VGG-16

模型分析：

特征图的空间分辨率单调递减，特征图的通道数单调递增，使得输入图像在维度上流畅地转换到分类向量。用通俗的语言讲，就是特征图尺寸单调递减，特征图数量单调递增。从上面的模型图上来看，立体方块的宽和高逐渐减小，但是厚度逐渐增加。

一些其它的特点如下：

1. 选择采用 3x3 的卷积核是因为 3x3 是最小的能够捕捉像素 8 邻域信息的尺寸。

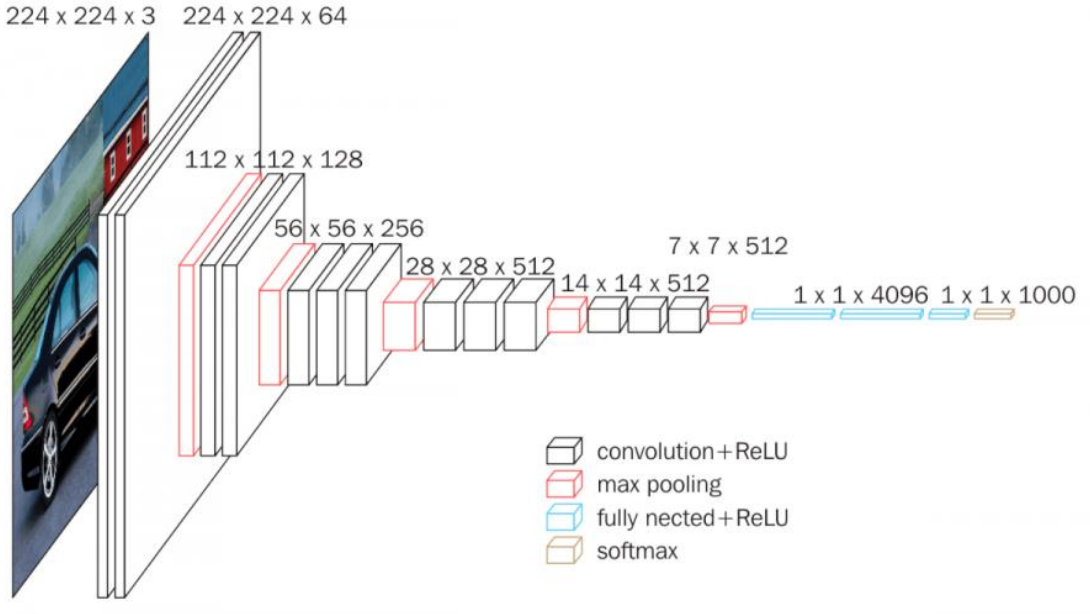
2. 使用 1x1 的卷积核目的是在不影响输入输出的维度情况下，对输入进行形变，再通过 ReLU 进行非线性处理，提高决策函数的非线性。

3. 2 个 3x3 卷积堆叠等于 1 个 5x5 卷积，3 个 3x3 堆叠等于 1 个 7x7 卷积，感受野大小不变，而采用更多层、更小的卷积核可以引入更多非线性（更多的隐藏层，从而带来更多非线性函数），提高决策函数判决力，并且带来更少参数。

4. 每个 VGG 网络都有 3 个 FC 层，5 个池化层，1 个 softmax 层。

在 FC 层中间采用 dropout 层，防止过拟合。

模型示意图（论文里的模型）



参数:

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

全连接层：

采用高斯分布初始化（std=0.005），bias 常数初始化（0.1）

一些修改：

- 为了减小模型，将模型最后的全连接层改为了 512、128、30
- 由于是预测模型，故将最后一层节点数设为 30，并去掉了 softmax 层
- 优化器使用 SGD
- 使用了未缩放的原数据

整体模型设计：

```
    # BLOCK 1
model.add(Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding='same', input_shape=(96, 96, 1)))
model.add(Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1),
activation='relu',padding='same'))

model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))

# BLOCK 2
model.add(Conv2D(filters=128, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding='same'))
model.add(Conv2D(filters=128, kernel_size=(3, 3), strides=(1, 1),
activation='relu',padding='same'))

model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))

# BLOCK 3
model.add(Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding='same'))
model.add(Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1),
activation='relu',padding='same'))
model.add(Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1),
activation='relu',padding='same'))

model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))

# BLOCK 4
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding='same'))
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1),
activation='relu',padding='same'))
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding='same'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))

# BLOCK 5
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding='same'))
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1),
activation='relu',padding='same'))
model.add(Conv2D(filters=512, kernel_size=(3, 3), strides=(1, 1),
activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))

# full connected #改了参数， 4096 个节点 pc 跑不出来
model.add(Flatten())
model.add(Dense(512,
activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.005,
seed=None),bias_initializer=Constant(value=0.1)))
model.add(Dropout(0.2))
model.add(Dense(128,
activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.005,
seed=None),bias_initializer=Constant(value=0.1)))
model.add(Dropout(0.1))
model.add(Dense(30,
activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.005,
seed=None),bias_initializer=Constant(value=0.1))) # 最终输出 15 个 2 维坐标
```

结果：

Loss 很大，并且 loss 下降很慢，于是手动终止了训练

可能原因分析：

1. 数据未缩放
2. SGD 收敛太慢
3. SGD 导致 loss 有严重的震荡

   SGD 对每个样本进行一次参数更新

   缺点:SGD 因为更新比较频繁，会造成 cost function 有严重的震荡。例如，在训练中刚开始 loss 从 2000 直接涨到 20000，能说明 cost function 有严重的震荡
4. 激活函数使用的是 ReLU，没有用 `LeakyReLU` 和 `BatchNormalization()`
   a) ReLU 的缺点：

      训练的时候很"脆弱"，很容易就"die"了

      例如，一个非常大的梯度流过一个 ReLU 神经元，更新过参数之后，这个神经元再也不会对任何数据有激活现象了，那么这个神经元的梯度就永远都会是 0.

      如果 learning rate 很大，那么很有可能网络中的 40% 的神经元都"dead"了。

3. 优化的 VGG-16

模型设计：在 VGG-16 的基础上做了一些改动

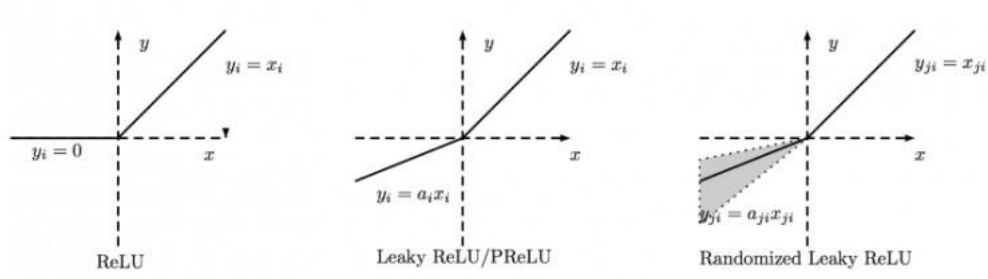
1. 卷积层 filter 数改为 32、64、96、128、256、512。因为图片大小比论文里所用的样本小，并且是黑白的，适当减少 filter 数对结果影响较小，并且减小了模型大小，减少了模型参数，训练更快。

2. 使用 LeakyReLU

   Leaky ReLu 是给所有负值赋予一个非零斜率。Leaky ReLU 激活函数是在声学模型（2013）中首次提出的。以数学的方式表示为：

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{a_i} & \text{if } x_i < 0, \end{cases}$$

ai 是（1，+∞）区间内的固定参数。



3. BatchNormalization：

   深度神经网络训练过程中使得每一层神经网络的输入保持相同分布

整体模型：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 96, 96, 32)        288
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 96, 96, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 96, 96, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 96, 96, 32)        9216
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 96, 96, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 96, 96, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 48, 48, 64)        18432
```

leaky_re_lu_3 (LeakyReLU)	(None, 48, 48, 64)	0
batch_normalization_3 (Batch	(None, 48, 48, 64)	256
conv2d_4 (Conv2D)	(None, 48, 48, 64)	36864
leaky_re_lu_4 (LeakyReLU)	(None, 48, 48, 64)	0
batch_normalization_4 (Batch	(None, 48, 48, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 24, 24, 64)	0
conv2d_5 (Conv2D)	(None, 24, 24, 96)	55296
leaky_re_lu_5 (LeakyReLU)	(None, 24, 24, 96)	0
batch_normalization_5 (Batch	(None, 24, 24, 96)	384
conv2d_6 (Conv2D)	(None, 24, 24, 96)	82944
leaky_re_lu_6 (LeakyReLU)	(None, 24, 24, 96)	0
batch_normalization_6 (Batch	(None, 24, 24, 96)	384
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 96)	0
conv2d_7 (Conv2D)	(None, 12, 12, 128)	110592
leaky_re_lu_7 (LeakyReLU)	(None, 12, 12, 128)	0
batch_normalization_7 (Batch	(None, 12, 12, 128)	512
conv2d_8 (Conv2D)	(None, 12, 12, 128)	147456
leaky_re_lu_8 (LeakyReLU)	(None, 12, 12, 128)	0
batch_normalization_8 (Batch	(None, 12, 12, 128)	512
max_pooling2d_4 (MaxPooling2	(None, 6, 6, 128)	0
conv2d_9 (Conv2D)	(None, 6, 6, 256)	294912
leaky_re_lu_9 (LeakyReLU)	(None, 6, 6, 256)	0

```
_________________________________________________________________
batch_normalization_9 (Batch (None, 6, 6, 256)         1024
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 6, 6, 256)         589824
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 6, 6, 256)         0
_________________________________________________________________
batch_normalization_10 (Batc (None, 6, 6, 256)         1024
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 3, 3, 512)         1179648
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 3, 3, 512)         0
_________________________________________________________________
batch_normalization_11 (Batc (None, 3, 3, 512)         2048
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 3, 3, 512)         2359296
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 3, 3, 512)         0
_________________________________________________________________
batch_normalization_12 (Batc (None, 3, 3, 512)         2048
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               2359808
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 30)                15390
=================================================================
Total params: 7,268,670
Trainable params: 7,264,318
Non-trainable params: 4,352
_________________________________________________________________
```

训练结果：

1. 使用未缩放的原始数据训练，迭代 50 次，batch size=256，优化器 Adam，划分 0.3%的数据用于测试
   提交结果：RMSE=1.6888

2. 使用未缩放的原始数据训练，迭代 50 次，batch size=256，优化器 Adam，划分 0.1%的数据用于测试

使用 early stop，在测试集上结果没有变好的迭代超过 5 次则停止训练，保存最好的权重。

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
```

最终只迭代了 24 次，结果变差。

a) 本地 loss

```
loss: 11.3739 - val_loss: 3.9829
```

b) 原因分析：训练停止过早，模型欠拟合，patience 设置太小

3. 使用缩放的原始数据训练，迭代 50 次，batch size=256，优化器 Adam，划分 0.1%的数据用于测试
   提交结果：RMSE=1.6616
   分析：
   使用规范化后缩放的数据集结果稍有提升，并且训练更快

4. 增大 batch_size：预测 epoch 的速度加快，但是结果达到相同精度所需训练论述增多。
5. 减小 batch_size：结果无法收敛

其他想尝试但没有实现的解决方案：

1. VGG-16 全连接层改为全局平均池化，这样可以减少参数数量，训练更快
2. 集成学习：因为一个模型的大小已经超过限制的 50M 了，故没有进行集成学习

实验小结

1. 了解了图像识别的原理，学习了卷积神经网络的原理，并自己搭建了卷积神经网络用于训练，巩固了所学知识。
2. 主要学习了 LeNet5、VGG 两种卷积神经网络，同时也了解了 AlexNet、NIN、GoogLeNet、ResNet、Faster R-CNN 等模型。
3. 学习了数据增强的方式，将图片翻转以获得更大数据集
4. 学习 Keras 和 TensorFlow
5. 配置了 GPU，Nvidia CUDA 10.0 用于加速运算