基于keras搭建CNN进行图像分类

<Excerpt in index | 首页摘要>
主要基于三个数据集进行基本的图像分类器的搭建帮助入门分类问题
关键词： **猫狗大战　Keras==2.0.1　TesnsorFlow后端　CPU训练**
blog： 【Keras】从两个实际任务掌握图像分类
Building powerful image classification models using very little data
面向小数据集构建图像分类模型
keras系列｜图像多分类训练与利用bottleneck features进行微调
<The rest of contents | 余下全文>

这里总结几个比较贴近实际的分类项目，从数据分析和处理说起，基于Keras，帮助掌握图像分类任务

猫狗大战

本文需要使用的Keras模块有：

- `fit_generator`：用于从Python生成器中训练网络
- `ImageDataGenerator`：用于实时数据提升

准备：

- `训练集和测试集`：kaggle上的猫狗大战
- `环境`：安装有Keras，SciPy，PIL的机器，如果有NVIDIA GPU那就更好了
- `数据集存放`：文件路径按以下存放：

```
data/
    train/
        dogs/
            dog001.jpg
            dog002.jpg
            ...
        cats/
            cat001/jpg
            cat002.jpg
            ...
    validation/
        dogs/
            dog001.jpg
```

```
                    dog002.jpg
                    ...
                cats/
                    cat001/jpg
                    cat002.jpg
                    ...
```

数据预处理与数据提升

为了尽量利用我们有限的训练数据，我们将通过一系列随机变换堆数据进行提升，这样我们的模型将看不到任何两张完全相同的图片，这有利于我们**抑制过拟合**，使得模型的**泛化能力更好**。

在Keras中，这个步骤可以通过`keras.preprocessing.image.ImageGenerator`来实现，这个类使你可以：

- 在训练过程中，设置要施行的随机变换
- 通过`.flow`或`.flow_from_directory(directory)`方法实例化一个针对图像batch的生成器，这些生成器可以被用作keras模型相关方法的输入，如`fit_generator`，`evaluate_generator`和`predict_generator`

案例如下图所示：

```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, i
mg_to_array, load_img

datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

img = load_img('data/train/cats/cat.0.jpg')  # this is a PIL image
x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3,
 150, 150)

# the .flow() command below generates batches of randomly transformed ima
ges
# and saves the results to the `preview/` directory
i = 0
for batch in datagen.flow(x, batch_size=1,
                          save_to_dir='preview', save_prefix='cat', save_
format='jpeg'):
    i += 1
    if i > 20:
        break  # otherwise the generator would loop indefinitely
```

现象：

载入数据并对数据进行预处理

```
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,    #归一化
        shear_range=0.2,    #剪切变换
        zoom_range=0.2,     #随机放大
        horizontal_flip=True)    #水平翻转
```

训练集分批载入

```
test_datagen = ImageDataGenerator(rescale=1./255)

# this is a generator that will read pictures found in
# subfolers of 'data/train', and indefinitely generate
# batches of augmented image data
train_generator = train_datagen.flow_from_directory(
        '/home/weijia.wu/workspace/Kaggle/cat_and_dog/dataset/train',  #
 this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=32,
        class_mode='binary')  # since we use binary_crossentropy loss, we
 need binary labels
```

简单的cnn搭建

数据提升是对抗过拟合问题的一个武器，但还不够，因为提升过的数据仍然是高度相关的。对抗过拟合的你应该主要关注的是模型的**“熵容量”**——模型允许存储的信息量。能够存储更多信息的模型能够利用更多的特征取得更好的性能，但也有存储不相关特征的风险。另一方面，只能存储少量信息的模型会将存储的特征主要集中在真正相关的特征上，并有更好的泛化性能。

有很多不同的方法来调整模型的“熵容量”，常见的一种选择是**调整模型的参数数目**，即模型的层数和每层的规模。另一种方法是对**权重进行正则化约束**，如L1或L2.这种约束会使模型的权重偏向较小的值。

在我们的模型里，我们使用了很小的卷积网络，只有很少的几层，每层的滤波器数目也不多。再加上数据提升和Dropout，就差不多了。Dropout通过防止一层看到两次完全一样的模式来防止过拟合，相当于也是一种数据提升的方法。（你可以说dropout和数据提升都在随机扰乱数据的相关性）

```
model = Sequential()
model.add(Convolution2D(32, 3, 3, input_shape=( 150, 150 , 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(32, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(64, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# the model so far outputs 3D feature maps (height, width, features)

model.add(Flatten())  # this converts our 3D feature maps to 1D feature v
ectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练并得到结果

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

model.fit_generator(
        train_generator,
        samples_per_epoch=train_generator.n,
        nb_epoch=20,
        validation_data=validation_generator,
        nb_val_samples=validation_generator.n)
```

结果：

```
782/781 [==============================] - 102s 130ms/step - loss: 0.3590 - acc: 0.8520 - val_loss: 0.3109 - val_acc: 0.8723
Epoch 19/20
782/781 [==============================] - 102s 130ms/step - loss: 0.3598 - acc: 0.8520 - val_loss: 0.2871 - val_acc: 0.8777
Epoch 20/20
782/781 [==============================] - 102s 130ms/step - loss: 0.3608 - acc: 0.8511 - val_loss: 0.2702 - val_acc: 0.8927
```

fit_generator

```
fit_generator(
                generator=,
                steps_per_epoch=None,  #就是一个epoch执行多少次batch。解决训练集过大，无法一次性放入内容
                epochs=1,              #An epoch is an iteration over the entire data provided
                verbose=1,
                callbacks=None,
                validation_data=None,
                validation_steps=None,
                class_weight=None,
                max_queue_size=10,
                workers=1,
                use_multiprocessing=False,
                shuffle=True,
                initial_epoch=0)
```

猫狗大战——使用预训练网络的bottleneck特征

A more **refined approach** would be to leverage a network pre-trained on a large dataset. Such a network would have already **learned features** that are useful for most computer vision problems, and leveraging such features would allow us to reach a better accuracy than any method that would only rely on the available data.

Our strategy will be as follow: we will **only instantiate the convolutional part of the model**, everything up to the fully-connected layers. We will then run this model on our training and validation data once, recording the output (the **"bottleneck features"** from th VGG16 model: the last activation maps before the fully-connected layers) in two numpy arrays. Then we will train a small fully-connected model on top of the stored features.

```
datagen = ImageDataGenerator(rescale=1./255)
generator = datagen.flow_from_directory(
        '/home/weijia.wu/workspace/Kaggle/cat_and_dog/dataset/train',
        target_size=(150, 150),
        batch_size=32,
```

```
        class_mode=None,  # this means our generator will only yield batc
hes of data, no labels
        shuffle=False)  # our data will be in order, so all first 1000 im
ages will be cats, then 1000 dogs
# the predict_generator method returns the output of a model, given
# a generator that yields batches of numpy data
bottleneck_features_train = model.predict_generator(generator, generator.
n)
# save the output as a Numpy array
np.save(open('bottleneck_features_train.npy', 'w'), bottleneck_features_t
rain)

generator = datagen.flow_from_directory(
        '/home/weijia.wu/workspace/Kaggle/cat_and_dog/dataset/validatio
n',
        target_size=(150, 150),
        batch_size=32,
        class_mode=None,
        shuffle=False)
bottleneck_features_validation = model.predict_generator(generator, gener
ator.n)
np.save(open('bottleneck_features_validation.npy', 'w'), bottleneck_featu
res_validation)
```

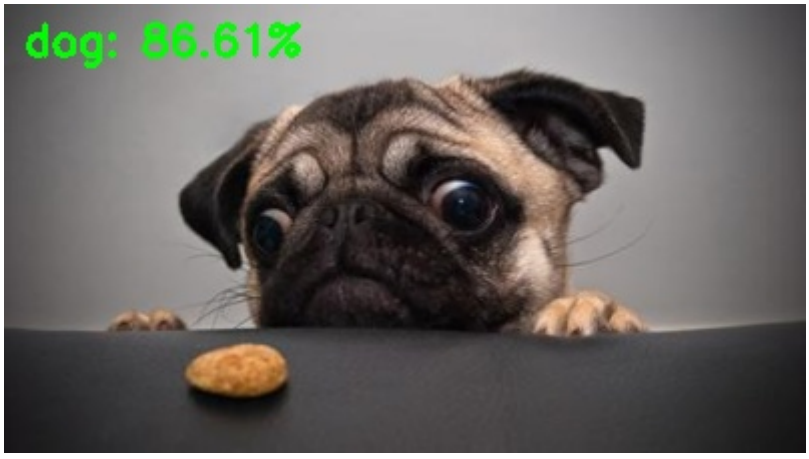
We can then load our saved data and train a small fully-connected model:

```
train_data = np.load(open('bottleneck_features_train.npy'))
# the features were saved in order, so recreating the labels is easy
train_labels = np.array([0] * 12498 + [1] * 12498)

validation_data = np.load(open('bottleneck_features_validation.npy'))
validation_labels = np.array([0] * 466 + [1] * 466)

model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(train_data, train_labels,
          nb_epoch=20, batch_size=32,
          validation_data=(validation_data, validation_labels))
```

对单张图片进行预测:

```
image = cv2.imread(args)
orig = image.copy()

# pre-process the image for classification
image = cv2.resize(image, (norm_size, norm_size))
image = image.astype("float") / 255.0
image = img_to_array(image)
image = np.expand_dims(image, axis=0)
# classify the input image
result = model.predict(image)[0]
proba = np.max(result)
label = str(np.where(result==proba)[0])
if(proba>0.5):
    label = 'dog'
else:
    label = 'cat'
label = "{}: {:.2f}%".format(label, proba * 100)

if 1:
    # draw the label on the image
    output = imutils.resize(orig, width=400)
    cv2.putText(output, label, (10, 25),cv2.FONT_HERSHEY_SIMPLEX,
        0.7, (0, 255, 0), 2)
```

结果:



猫狗大战——fine-tune

Fine-tune是指以一个预训练好的网络为基础，在新的数据集上**重新训练一小部分权重。**在这个实验中，fine-tune分三个步骤:

- 导入vgg-16并载入权重
- 将之前定义的全连接网络加在模型的顶部，并载入权重
- 冻结vgg16网络的一部分参数

Notice:

- 只选择fine-tune最后的卷积块，而不是整个网络，这是为了防止过拟合。整个网络具有巨大的熵容量，因此具有很高的过拟合倾向。由底层卷积模块学习到的特征更加一般，更加不具有抽象性，因此我们要保持前两个卷积块（学习一般特征）不动，只fine-tune后面的卷积块（学习特别的特征）。
- fine-tune应该在很低的学习率下进行，通常使用SGD优化而不是其他自适应学习率的优化算法，如RMSProp。这是为了保证更新的幅度保持在较低的程度，以免毁坏预训练的特征。

在vgg后面添加我们的模型：

```
from keras.applications import VGG16  #直接导入已经训练好的VGG16网络
base_model = VGG16(input_shape=(150,150,3),include_top=False, weights='im
agenet')
```

```
# build a classifier model to put on top of the convolutional model
top_model = Sequential()
top_model.add(Flatten(input_shape=base_model.output_shape[1:]))
top_model.add(Dense(256, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(1, activation='sigmoid'))
model = Model(inputs=base_model.input, outputs=top_model(base_model.outpu
t))
```

卷积层参数冻结：

将最后一个卷积块前的卷积层参数冻结

```
# set the first 25 layers (up to the last conv block)
# to non-trainable (weights will not be updated)
for layer in model.layers[:25]:
    layer.trainable = False

# compile the model with a SGD/momentum optimizer
# and a very slow learning rate.
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.SGD(lr=1e-4, momentum=0.9),
              metrics=['accuracy'])
```

训练：

以很低的学习率进行训练

```
# fine-tune the model
model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.n ,
        epochs=30,
        validation_data=validation_generator,
        validation_steps=validation_generator.n)
```

结果:

```
24996/24996 [==============================] - 234s 9ms/step - loss: 0.3111 - acc: 0.9055 - val_loss: 0.0948 - val_acc: 0.9710
Epoch 2/30
24996/24996 [==============================] - 234s 9ms/step - loss: 0.3275 - acc: 0.9051 - val_loss: 0.0948 - val_acc: 0.9710
Epoch 3/30
24996/24996 [==============================] - 234s 9ms/step - loss: 0.3179 - acc: 0.9074 - val_loss: 0.0948 - val_acc: 0.9710
```

猫狗大战——杨培文

用InceptionV3，ResNet50，Xception提取特征然后再用于后续训练

通过 Keras 搭建一个深度卷积神经网络来识别一张图片是猫还是狗，在验证集上的准确率可以达到 99.6%

导出特征向量：

提高我们的模型表现。那么一种有效的方法是综合各个不同的模型，从而得到不错的效果，兼听则明。如果是直接在一个巨大的网络后面加我们的全连接，那么训练10代就需要跑十次巨大的网络，而且我们的卷积层都是不可训练的，那么这个计算就是浪费的。所以我们可以**将多个不同的网络输出的特征向量先保存下来，以便后续的训练**

```
def write_gap(MODEL, image_size, lambda_func=None):
    width = image_size[0]
    height = image_size[1]
    input_tensor = Input((height, width, 3))
    x = input_tensor
    if lambda_func:
        x = Lambda(lambda_func)(x)
```

```
    base_model = MODEL(input_tensor=x, weights='imagenet', include_top=Fa
lse)
    model = Model(base_model.input, GlobalAveragePooling2D()(base_model.o
utput))

    gen = ImageDataGenerator()
    train_generator = gen.flow_from_directory("train2", image_size, shuff
le=False,
                                              batch_size=16)
    test_generator = gen.flow_from_directory("test2", image_size, shuffle
=False,
                                             batch_size=16, class_mode=No
ne)

    train = model.predict_generator(train_generator, train_generator.nb_s
ample)
    test = model.predict_generator(test_generator, test_generator.nb_samp
le)
    with h5py.File("gap_%s.h5"%MODEL.func_name) as h:
        h.create_dataset("train", data=train)
        h.create_dataset("test", data=test)
        h.create_dataset("label", data=train_generator.classes)
```

载入特征向量：

```
for filename in ["gap_ResNet50.h5", "gap_Xception.h5", "gap_InceptionV3.h
5"]:
    with h5py.File(filename, 'r') as h:
        X_train.append(np.array(h['train']))
        X_test.append(np.array(h['test']))
        y_train = np.array(h['label'])

X_train = np.concatenate(X_train, axis=1)
X_test = np.concatenate(X_test, axis=1)

X_train, y_train = shuffle(X_train, y_train)  #打乱顺序
```

建立模型：

```
input_tensor = Input(X_train.shape[1:])
x = input_tensor
x = Dropout(0.5)(x)
x = Dense(1, activation='sigmoid')(x)
model = Model(input_tensor, x)
```

```
model.compile(optimizer='adadelta',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

结果：

```
Epoch 1/8
20000/20000 [==============================] - 8s 414us/step - loss: 0.1267 - acc: 0.9632 - val_loss: 0.0312 - val_acc: 0.9942
Epoch 2/8
20000/20000 [==============================] - 2s 92us/step - loss: 0.0336 - acc: 0.9911 - val_loss: 0.0194 - val_acc: 0.9942
Epoch 3/8
20000/20000 [==============================] - 2s 92us/step - loss: 0.0231 - acc: 0.9929 - val_loss: 0.0166 - val_acc: 0.9946
```

猫狗大战——迁移学习

加载数据：

```
np.random.seed(2017)
n = 25000
X = np.zeros((n, 224, 224, 3), dtype=np.uint8)
y = np.zeros((n, 1), dtype=np.uint8)
for i in tqdm(range(n/2)):
    X[i] = cv2.resize(cv2.imread('/home/weijia.wu/workspace/Kaggle/cat_an
d_dog/dataset2/train2/cat/cat.%d.jpg' % i), (224, 224))
    X[i+n/2] = cv2.resize(cv2.imread('/home/weijia.wu/workspace/Kaggle/ca
t_and_dog/dataset2/train2/dog/dog.%d.jpg' % i), (224, 224))
y[n/2:] = 1
```

构建模型：

```
base_model = ResNet50(input_tensor=Input((224, 224, 3)), weights='imagene
t', include_top=False)

for layers in base_model.layers:
    layers.trainable = False

x = GlobalAveragePooling2D()(base_model.output)
x = Dropout(0.25)(x)
x = Dense(1, activation='sigmoid')(x)
model = Model(base_model.input, x)
```

训练模型：

```
Epoch 1/5
20000/20000 [==============================] - 134s 7ms/step - loss: 0.1504 - acc: 0.9455 - val_loss: 0.0644 - val_acc: 0.9776
Epoch 2/5
20000/20000 [==============================] - 133s 7ms/step - loss: 0.0888 - acc: 0.9673 - val_loss: 0.0539 - val_acc: 0.9804
Epoch 3/5
20000/20000 [==============================] - 133s 7ms/step - loss: 0.0780 - acc: 0.9703 - val_loss: 0.0502 - val_acc: 0.9808
```

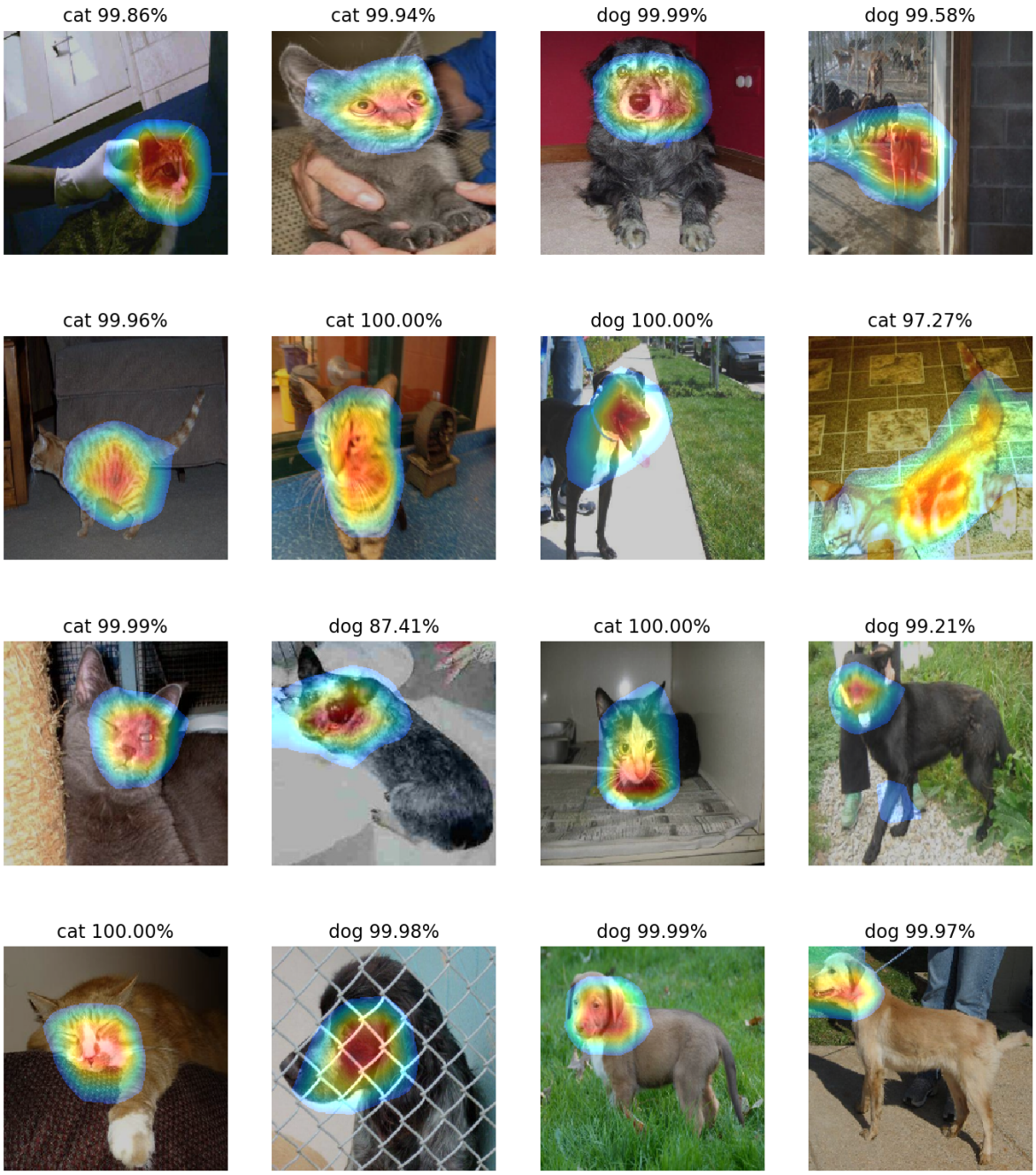
CAM：

CAM可视化

在Learning Deep Features for Discriminative Localization这篇文章中，作者提出了CNN网络除了具有很强的图片处理，分类能力；同时还能够针对图片中的**关键部分进行定位**，这个过程被称为Class Activation Mapping，简称CAM。

```
for i in range(16):
    plt.subplot(4, 4, i+1)
    img = cv2.imread('/home/weijia.wu/workspace/Kaggle/cat_and_dog/datase
t2/test/test/%d.jpg' % random.randint(1, 12500))
    img = cv2.resize(img, (224, 224))
    x = img.copy()
    x.astype(np.float32)
    out, prediction = model2.predict(np.expand_dims(x, axis=0))

    prediction = prediction[0]
    out = out[0]

    if prediction < 0.5:
        plt.title('cat %.2f%%' % (100 - prediction*100))
    else:
        plt.title('dog %.2f%%' % (prediction*100))

    cam = (prediction - 0.5) * np.matmul(out, weights)
    cam -= cam.min()
    cam /= cam.max()
    cam -= 0.2
    cam /= 0.8

    cam = cv2.resize(cam, (224, 224))
    heatmap = cv2.applyColorMap(np.uint8(255*cam), cv2.COLORMAP_JET)
    heatmap[np.where(cam <= 0.2)] = 0

    out = cv2.addWeighted(img, 0.8, heatmap, 0.4, 0)

    plt.axis('off')
    plt.imshow(out[:,:,::-1])
```

现象：



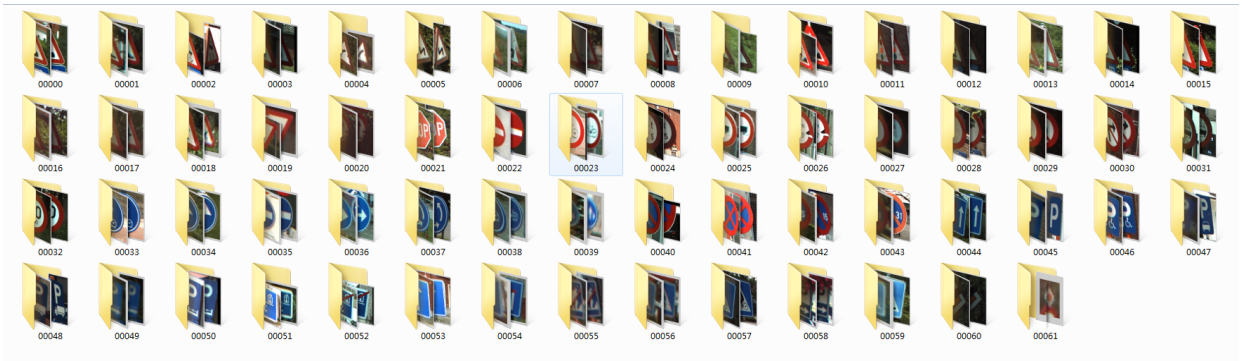
交通标志分类

准备数据

```
data/
    train/
        00000/
```

```
                    01153_00000.png
                    01153_00001.png
                    ...
                00001/
                    00025_00000.png
                    00025_00001.png
                    ...
                ...
            test/
                00000/
                    00017_00000.png
                    00017_00001.png
                    ...
                00001/
                    00252_00000.png
                    00252_00001.png
                    ...
                ...
```

每个文件夹的名字就是其标签。



搭建LeNet:

```
def build(width, height, depth, classes):
        # initialize the model
        model = Sequential()
        inputShape = (height, width, depth)
        # if we are using "channels last", update the input shape
        if K.image_data_format() == "channels_first":   #for tensorflow
            inputShape = (depth, height, width)
        # first set of CONV => RELU => POOL layers
        model.add(Conv2D(20, (5, 5),padding="same",input_shape=inputShap
e))  #卷积
        model.add(Activation("relu"))
    #特定的激活函数类型
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    #最大池化
        #second set of CONV => RELU => POOL layers
        model.add(Conv2D(50, (5, 5), padding="same"))
```

```
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # first (and only) set of FC => RELU layers
        model.add(Flatten())                    #将输入“压平”，用于卷积层到全连
接层的过渡，
        model.add(Dense(500))                   #全连接层（500个神经元）
        model.add(Activation("relu"))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model
```

载入数据

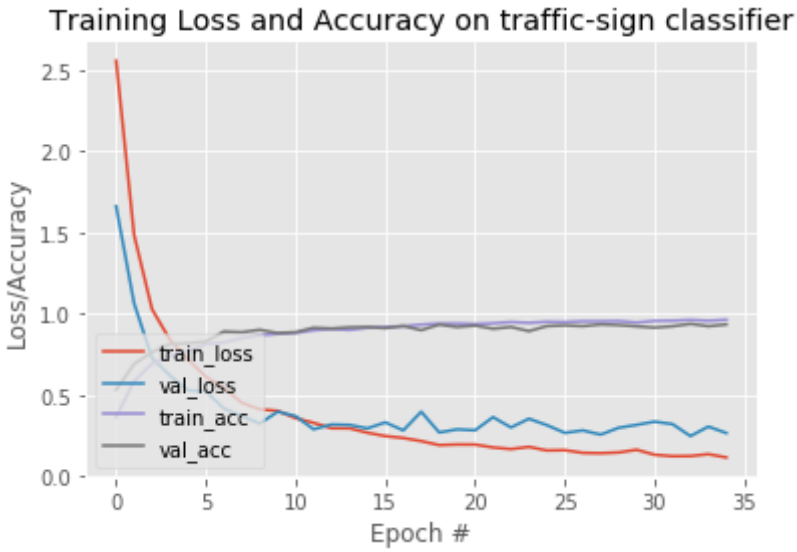
norm_size（图片归一化尺寸）是根据你得到的数据集，经过分析后得出的，这个数据集大多数图片的尺度都需要在这个范围内。

```
def load_data(path):
    print('[INFO] loading images...')
    data = []
    labels = []
    # grad the image paths and randomly shuffle them
    imagePaths = sorted(list(paths.list_images(path)))
    random.seed(42)
    random.shuffle(imagePaths)
    # loop over the imput image
    for imagePath in imagePaths:
        image = cv2.imread(imagePath)
        image = cv2.resize(image, (norm_size,norm_size))
        image = img_to_array(image)
        data.append(image)

        # extract the class label from the image path and update the labe
l list
        label = int(imagePath.split(os.path.sep)[-2])    #得到文件名
        labels.append(label)
    # scale the raw pixel intensities to the range(0,1)
    data = np.array(data,dtype="float") / 255.0
    labels = np.array(labels)

    labels = to_categorical(labels,num_classes=CLASS_NUM)
    return data,labels
```

训练算法

```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="categorical_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# train the network
print("[INFO] training network...")
H = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=1)
```

结果:



可以提高的操作:

- 1、图片归一化大小是否合适
- 2、可以考虑更深更好的网络，比如VGG、GoogleNet等
- 3、数据增强部分还可以做一做