

# 基于车辆实拍图片的 数据研究

10174503110 印张悦

10175501112 陈诺

华东师范大学 上海浦东 201208

## 目录

1 数据集介绍 .....	3
1.1 数据集来源 .....	3
1.2 数据集分析 .....	3
2 数据预处理 .....	4
2.1 图片剪裁 .....	4
2.2 图片规整化 .....	4
2.3 处理效果 .....	5
2 Keras CNN 模型 .....	5
3.1 模型搭建 .....	5
3.2 训练结果 .....	6
4 PCA DNN 模型 .....	7
4.1 导入数据 .....	7
4.2 PCA 降维 .....	7
4.3 DNN 模型 .....	8
4.3 模型测试结果 .....	8
5 机器学习算法 .....	9
5.1 决策树算法 .....	9
5.2 随机森林算法 .....	9
5.3 SVM 算法 .....	10
5.4 改进 .....	10
5.4.1 边缘检测方法 .....	10
5.4.2 采用 OpenCV 实现上述方法 .....	11
5.4.3 采用 Otus 方法进行图片处理 .....	12
5.4.4 Otus 方法+机器学习算法 .....	12
6 GAN 新车型生成 .....	13

\* 作者： 印张悦、陈诺

6.1 搭建 GAN.....	13
6.2 图片导入 .....	13
6.2 生成结果 .....	14
7 Matlab 数据处理与导出.....	16
7.1 .mat 部分数据.....	16
7.2 利用 Matlab 的 csvwrite 导出 .....	17
7.3 利用 python 导出 .....	17
8 Matlab and AlexNet.....	20
8.1 Why Matlab .....	20
8.2 Why AlexNet.....	21
8.2.1 AlexNet 发展历史 .....	21
8.2.2 Alexnet 具体细节.....	21
8.2.3 AlexNet 为什么好用 .....	23
8.2.4 AlexNet 结构分析 .....	23
9 Matlab AlexNet 预测车型.....	29
9.1 调整最后三层 .....	29
9.2 导入图片与训练.....	31
9.3 测试.....	34
9.5 准确率预测.....	35
10 小插曲：RCNN 区域检测.....	35
10.1 起因 .....	36
10.1 过程.....	37
11 总结 .....	42
参考文献.....	43

\* 作者： 印张悦、陈诺

# 1 数据集介绍

## 1.1 数据集来源

来源于 Stanford 的车辆数据集: [http://ai.stanford.edu/~jkrause/cars/car\\_dataset.html](http://ai.stanford.edu/~jkrause/cars/car_dataset.html), 共 196 类车型, 共 16,185 张图片, 8,144 张训练集图片和 8,041 张测试集图片。

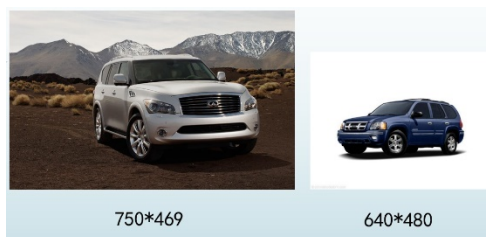
### Overview

The *Cars* dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. Classes are typically at the level of *Make, Model, Year*, e.g. 2012 Tesla Model S or 2012 BMW M3 coupe.



## 1.2 数据集分析

数据集中的图片存在大小不一、边缘区域太大、以及混入其他物体（比如人、其他车辆）等情况, 需进行剪裁和规整。



大小不一



混入其他物体

\* 作者: 印张悦、陈诺

## 2 数据预处理

### 2.1 图片剪裁

根据数据集中的图片轮廓信息，使用 OpenCV 进行剪裁，并将图片规整为 64\*64 的大小，方便进行处理。

数据集信息：

		X轴下界	Y轴下界	X轴上界	Y轴上界	标签
attribute	fname	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class
0	1	39	116	569	375	14
1	2	36	116	868	587	3
2	3	85	109	601	381	91
3	4	621	393	1484	1096	134
4	5	14	36	133	99	106
5	6	259	289	515	416	123
6	7	88	80	541	397	89
7	8	73	79	591	410	96
8	9	20	126	1269	771	167
9	10	21	110	623	367	58
10	11	51	93	601	393	49
11	12	6	62	499	286	186

采用 OpenCV 进行剪裁，剪裁时先对竖直方向进行裁剪，然后对水平方向进行裁剪：

```
def solve_image():
    for i in range(1, 8001):
        if i < 10:
            location = '000' + str(i)
        elif i < 100:
            location = '00' + str(i)
        elif i < 1000:
            location = '0' + str(i)
        else:
            location = str(i)
        img = cv2.imread('./data/0' + location + '.jpg')
        j = i - 1
        # 裁剪坐标为[y1:y2, x1:x2]
        cropped = img[int(df['bbox_y1'][j]):int(df['bbox_y2'][j]), int(df['bbox_x1'][j]):int(df['bbox_x2'][j])]
```

### 2.2 图片规整化

采用 OpenCV 将图片规整为 64\*64 的大小：

```
if height > 0 and width > 0:
    cropped = cv2.resize(cropped, (64, 64))
cv2.imwrite('./Solve_data/0' + location + '.jpg', cropped)
```

\* 作者： 印张悦、陈诺

## 2.3 处理效果

原数据集大小不一且边缘较大的情况得到了极大的改善：



## 2 Keras CNN 模型

现在图片处理中卷积神经网络应用广泛且效果不错，所以一开始我们就采用卷积神经网络进行建模。

### 3.1 模型搭建

使用 Keras 搭建 CNN 网络。

核心代码：

```
model = Sequential()
model.add(Conv2D(filters = 64, kernel_size = (2, 2), input_shape = (64, 64, 3), activation = 'relu'))
model.add(Dropout(rate=0.1))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model.add(Dropout(0.1))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Flatten()) #把多维的输入一维化

###从卷积层到全连接层的过渡###
model.add(Dense(1000, activation = 'relu'))
model.add(Dense(500, activation = 'sigmoid'))
model.add(Dense(197, activation = 'softmax'))
```

\* 作者： 印张悦、陈诺

模型介绍：

卷积层（2\*2 的卷积核）+遗忘层（每次断开节点 10%）+最大池化层+

卷积层（3\*3 的卷积核）+遗忘层（每次断开节点 10%）+最大池化层+

扁平层+三层全连接层（包括输出层）

```
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 63, 63, 64)	832
dropout_1 (Dropout)	(None, 63, 63, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 31, 31, 64)	0
conv2d_2 (Conv2D)	(None, 29, 29, 64)	36928
dropout_2 (Dropout)	(None, 29, 29, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 1000)	12545000
dense_2 (Dense)	(None, 500)	500500
dense_3 (Dense)	(None, 197)	98697
Total params: 13,181,957		
Trainable params: 13,181,957		
Non-trainable params: 0		

## 3.2 训练结果

选取 6000 个为训练集，2000 个为测试集，定义批大小为 50，世代为 10 开始训练，得到模型在测试集上的准确率为：13.12%。

```
#定义参数
epochs = 10
batch_size = 50

model.compile(loss = 'categorical_crossentropy',
              optimizer = 'adam', metrics = ['accuracy'])

train_history = model.fit(x_train_normalized, y_train_onehot,
                        validation_split = 0.1,
                        epochs = epochs, batch_size = batch_size)

score = model.evaluate(x_test_normalized, y_test_onehot, batch_size = 50)

800/800 [=====] - 0s 321us/step

print("test accuracy: {}".format(int(score[1] * 10000)/100))

test accuracy:13.12%
```

\* 作者： 印张悦、陈诺

## 4 PCA DNN 模型

面对高维数据，主成分分析提取关键特征是一个常用的方法，我们尝试采用 PCA 将高维数据映射到低维空间达到降维效果，减少数据规模，方便处理。

### 4.1 导入数据

由于算力的原因，我们不得不在原有数据集基础上规整数据为 80\*80，这样已经损失了很多信息，但由于计算能力的限制，我们不得不做出抉择。

```
def load_image():
    r = []
    g = []
    b = []
    for i in range(0, 8000):
        if i + 1 < 10:
            location = '000' + str(i+1)
        elif i + 1 < 100:
            location = '00' + str(i+1)
        elif i + 1 < 1000:
            location = '0' + str(i+1)
        else:
            location = str(i+1)
        test_image = image.load_img('./solved_data/0' + location + '.jpg', target_size = (80, 80))
        test_image = image.img_to_array(test_image)
        tmp_r = []
        tmp_g = []
        tmp_b = []

        for j in range(0, 80):
            tmp = []
            for k in range(0, 80):
                tmp.append(test_image[j][k][0])
            tmp_r.append(tmp)
        r.append(tmp_r)
```

### 4.2 PCA 降维

在之前的导入代码中就可以发现我们分成了三个不同的矩阵进行导入，这是因为彩色图片分为 RGB 三色，读取时为 RGB 张量（数据量\*图片长度\*图片宽度\*3），我们需要将其分离开来，规整为矩阵形式进行 PCA 降维。PCA 时我们选取 10 个主成分，由 Sklearn 库中的 PCA 函数实现，随后将其规整为 800\*1 的向量。

```
pca = PCA(n_components = 10, copy = False)
solved_train_r = []
solved_train_g = []
solved_train_b = []
for i in range(0, 6000):
    tmp = np.array(data_r[i])
    tmp = pca.fit_transform(tmp)
    tmp = tmp.reshape(800, 1)
    solved_train_r.append(tmp)

    tmp = np.array(data_g[i])
    tmp = pca.fit_transform(tmp)
    tmp = tmp.reshape(800, 1)
    solved_train_g.append(tmp)

    tmp = np.array(data_b[i])
    tmp = pca.fit_transform(tmp)
    tmp = tmp.reshape(800, 1)
    solved_train_b.append(tmp)
```

\* 作者： 印张悦、陈诺

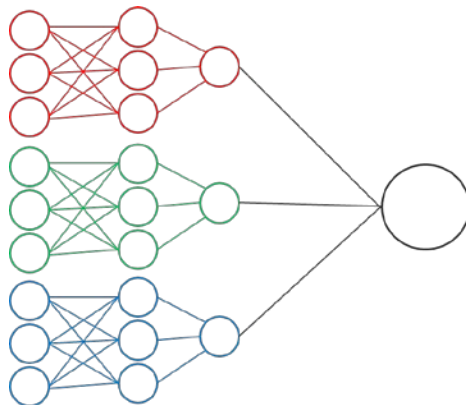


## 4.3 DNN 模型

由于降维后的数据破坏了原有数据的组织结构，因此 CNN 模型不再适用，所以我们采用 DNN 模型。

采用全连接层时我们考虑到一张图片的 R 和 G 和 B 是完全没有关系的，不应该相互连接，因此要搭建不同的神经网络，最后再把三个网络相互关联。

模型示意图如下图所示：



核心代码部分：

```
def neural_network(solved_train_r, solved_train_g, solved_train_b, solved_test_r, solved_test_g, solved_test_b,
                  train_r, train_g, train_b, test_r, test_g, test_b, train_target, test_target):
    model1=Sequential()
    model1.add(Dense(input_dim = 800 ,units = 600,activation='relu'))
    model1.add(Dropout(0.3))
    model1.add(Dense(units = 400,activation = 'relu'))
    model1.add(Dropout(0.1))
    model1.add(Dense(units = 197,activation = 'softmax'))
    model1.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
    model1.fit(train_r, train_target, batch_size = 100, epochs = 10, validation_split = 0.1) #batch_size越大越利用GPU平行计算，速度越快
    score = model1.evaluate(test_r, test_target)
    print("test_r accuracy:{}".format(int(score[1] * 10000)/100))
    result_train_r = []
    for i in range(0, 6000):
        tmp = np.array(solved_train_r[i])
        tmp = tmp.reshape(1,800)
        tmp = model1.predict_classes(tmp)
        result_train_r.append(int(tmp))
    result_test_r = []
    for i in range(0, 2000):
        tmp = np.array(solved_test_r[i])
        tmp = tmp.reshape(1,800)
        tmp = model1.predict_classes(tmp)
        result_test_r.append(int(tmp))

    model2=Sequential()
    model2.add(Dense(input_dim = 800 ,units = 600,activation='relu'))
    model2.add(Dropout(0.3))
    model2.add(Dense(units = 400,activation = 'relu'))
    model2.add(Dropout(0.1))
    model2.add(Dense(units = 197,activation = 'softmax'))
    model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model2.fit(train_g, train_target, batch_size = 100, epochs = 10, validation_split = 0.1) #batch_size越大越利用GPU平行计算，速度越快
```

## 4.3 模型测试结果

之前提到，由于算力原因，我们将图片规整的很小（80\*80），损失了很多数据，之后我们采用 PCA 降维，仅选取 10 个主成分又损失了许多信息，导致最后的测试集准确率仅为 0.75%，相信如果能保留原有图片的大小以及选取所有的主成分进行训练，模型的准确率一定能大大改善。

\* 作者： 印张悦、陈诺

```
Epoch 1/10
6000/6000 [=====] - 1s 134us/step - loss: 14.1919 - acc: 0.0053
Epoch 2/10
6000/6000 [=====] - 0s 80us/step - loss: 7.9119 - acc: 0.0038
Epoch 3/10
6000/6000 [=====] - 0s 78us/step - loss: 5.2821 - acc: 0.0080
Epoch 4/10
6000/6000 [=====] - 1s 87us/step - loss: 5.2800 - acc: 0.0082
Epoch 5/10
6000/6000 [=====] - 0s 77us/step - loss: 5.2786 - acc: 0.0082
Epoch 6/10
6000/6000 [=====] - 0s 74us/step - loss: 5.2775 - acc: 0.0082
Epoch 7/10
6000/6000 [=====] - 0s 77us/step - loss: 5.2765 - acc: 0.0082
Epoch 8/10
6000/6000 [=====] - 0s 80us/step - loss: 5.2757 - acc: 0.0082
Epoch 9/10
6000/6000 [=====] - 1s 88us/step - loss: 5.2749 - acc: 0.0082
Epoch 10/10
6000/6000 [=====] - 0s 72us/step - loss: 5.2743 - acc: 0.0082
2000/2000 [=====] - 0s 89us/step
final_test accuracy:0.75%
```

## 5 机器学习算法

课程中我们学习了如何用机器学习算法进行分类，所以我们试着采用机器学习算法解决该分类问题。使用 `skleran` 搭建，同样，由于算力原因，我们得丢失许多数据，这次我们不得不将图片规整为更小的  $32 \times 32$ ，特征数量为  $32 \times 32 \times 3$ ，进行训练和测试。

### 5.1 决策树算法

采用决策树进行分类任务，但由于训练集的特征很多，因此在训练集上得到了极高的准确率（过拟合了）。但由于类别较多（192），而训练集（8000）较少，最后的测试集准确率仅为 1.43%

```
# 决策树算法
dt = DecisionTreeClassifier()
dt.fit(x1_train, y1_train)

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                        max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                        splitter='best')

s_train = dt.score(x1_train, y1_train)
print("训练集准确率: {}%".format(int(s_train*10000)/100))

训练集准确率: 99.9%

s_test = dt.score(x1_test, y1_test)
print("测试集准确率: {}%".format(int(s_test*10000)/100))

测试集准确率: 1.43%
```

### 5.2 随机森林算法

和决策树同样的原因，随机森林算法也没有在测试集上得到较好的准确率，在尝试了 10 次后随机森林算法在测试集上的最高准确率为 1.93%，相对决策树算法略有提升。

\* 作者： 印张悦、陈诺

```
# 随机森林算法
rf = RandomForestClassifier(n_estimators = 10, oob_score = True)
rf.fit(x1_train, y1_train)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:458: UserWarning: Some inputs do not have OOB scores.
This probably means too few trees were used to compute any reliable oob estimates.
warn("Some inputs do not have OOB scores.",
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:463: RuntimeWarning: invalid value encountered in true_divide
predictions[k].sum(axis=1)[i, np.newaxis])

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
oob_score=True, random_state=None, verbose=0, warm_start=False)

s_train = rf.score(x1_train, y1_train)
print("训练集准确率: {}".format(int(s_train*10000)/100))

训练集准确率: 99.81%

s_test = rf.score(x1_test, y1_test)
print("测试集准确率: {}".format(int(s_test*10000)/100))

测试集准确率: 0.62%

# 交叉验证准确率
s1_rf = []
s2_rf = []
for i in range(0, 10):
    rf = RandomForestClassifier(n_estimators = 20)
    rf.fit(x1_train, y1_train)
    s1_rf = rf.score(x1_train, y1_train)
    s1_rf.append(s1_rf)
    s2_rf = rf.score(x1_test, y1_test)
    s2_rf.append(s2_rf)
print("训练集最高准确率: {}".format(int(max(s1_rf)*10000)/100))
print("测试集最高准确率: {}".format(int(max(s2_rf)*10000)/100))

训练集最高准确率: 99.9%
测试集最高准确率: 1.9%
```

## 5.3 SVM 算法

SVM 算法的结果不忍直视，测试集准确率仅为 0.25%（想象一下我们一共有 196 类，随机选择选对的概率为 1/196，约等于 0.51%，也就是比随机猜的准确率还低，惨不忍睹）。

```
# SVM算法
# rbf核函数，设置数据权重
svc = SVC(kernel='rbf', class_weight='balanced')
# 训练模型
clf = svc.fit(x1_train, y1_train)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
"avoid this warning.", FutureWarning)

s_train = svc.score(x1_train, y1_train)
print("训练集准确率: {}".format(int(s_train*10000)/100))

训练集准确率: 99.9%

s_test = svc.score(x1_test, y1_test)
print("测试集准确率: {}".format(int(s_test*10000)/100))

测试集准确率: 0.25%
```

## 5.4 改进

失败原因：之所以传统的机器学习算法准确率如此低，主要还是因为传统的机器学习算法需要人工提取有效的特征，而我们以像素中的 RGB 为特征，特征数量过多，而且有效性很差，所以我们尝试优化图片，自动提取有效特征。

### 5.4.1 边缘检测方法

对于本数据集，边缘特征是一个重要的有效特征，因此尝试自动提取图片的边缘信息。

常用的边缘检测的几种方法分析：

- （1）拉普拉斯算子：提取的边缘由于数据长度的关系，很多边缘容易在图中被忽略；
- （2）Canny 算子：提取的边缘断断续续，一般适用于图片中有大范围或明显的几何结构（比如道路检测）等场合中；

\* 作者： 印张悦、陈诺

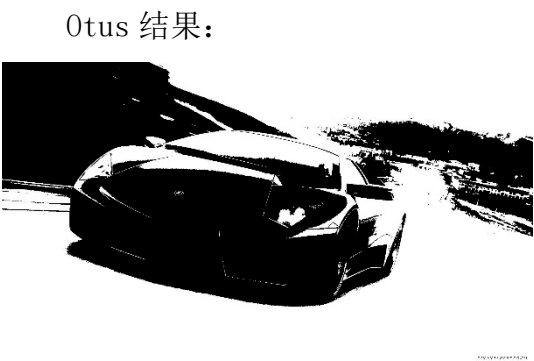
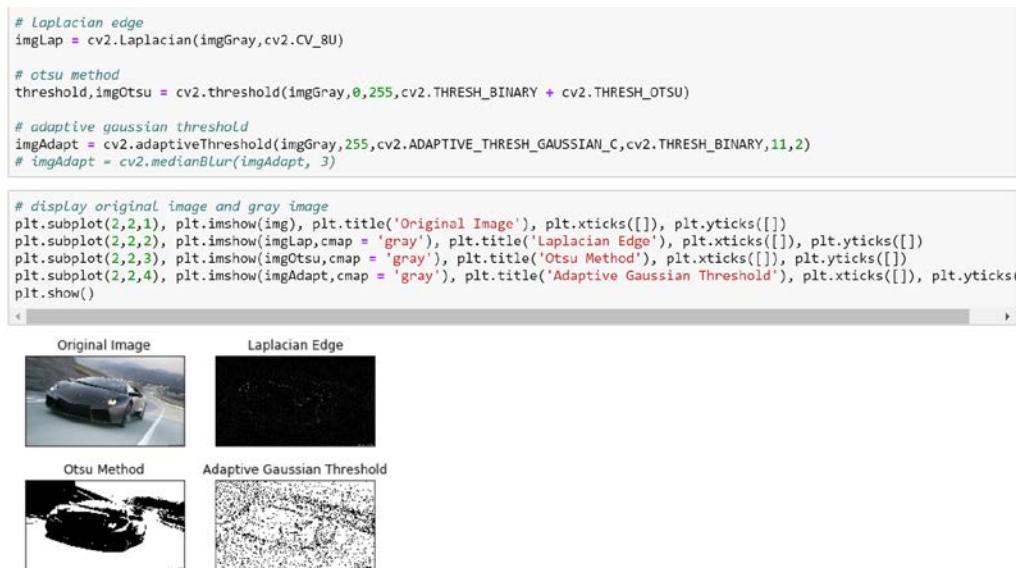
(3) 自适应高斯提取边缘：在不同的图像块内的阈值都不一样，所以对图片中不同的光照条件会有很好的适应；

(4) **Otsu 方法**：可用于提取前后背景，适用于有两个尖峰的灰度直方图的图片中。

### 5.4.2 采用 OpenCV 实现上述方法

利用 Python 的 OpenCV 库实现上述方法，由于 OpenCV 对上述方法有了很好的封装，因此实现较为简单。

下面的图片为同一张图片在拉普拉斯算子、Otsu 方法和自适应高斯提取边缘算法下的结果，显然 Otsu 方法更适用于本数据集的边缘检测和有效特征的提取。



不足之处：对于背景较复杂的图片，Otsu 方法也无法准确的提取边缘，背景的混乱导致在灰度直方图中的背景对应的灰度值投票过少，不足以形成尖峰使得前后背景分割的阈值刚好停留在尖峰的山脚附近。

\* 作者： 印张悦、陈诺



### 5.4.3 采用 Otus 方法进行图片处理

采用 Otus 方法处理我们的数据集，经过 Otus 方法处理后的图片变成了灰度图片，特征数量减少为  $32 \times 32$ ，而且边缘轮廓被较好的提取出来。

```
def solve_image(imgFile, imgSave):
    img = cv2.imread(imgFile)
    cRange = 256
    img = cv2.resize(img, (32, 32))
    rows, cols, channels = img.shape

    # convert color space from bgr to gray
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # otsu method
    threshold, imgOtsu = cv2.threshold(imgGray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    new_im = Image.fromarray(imgOtsu.astype(np.uint8))
    new_im.save(imgSave)
```

```
def load_image():
    T = []
    for i in range(1, 8001):
        if i < 10:
            location = '000' + str(i)
        elif i < 100:
            location = '00' + str(i)
        elif i < 1000:
            location = '0' + str(i)
        else:
            location = str(i)
        imgFile = './solved_data/0' + location + '.jpg'
        imgSave = './Black_data/0' + location + '.jpg'
        solve_image(imgFile, imgSave)
```

```
load_image()
```

### 5.4.4 Otus 方法+机器学习算法

对比之前得分最低的 SVM 的算法，新的 SVM 算法在调参优化和选用处理过的数据集的帮助下准确率提高了 3 倍（虽然还是很低），但这个小幅度的提升说明我们的处理还是有效

\* 作者： 印张悦、陈诺

果的。如果我们能进一步提取轮廓线条、车子主要部位（如车标等）等有效特征，相信我们的准确率还会大大提高。

```
# 调参优化
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
svc = SVC(kernel='rbf', class_weight='balanced',)
c_range = np.logspace(-5, 15, 11, base=2)
gamma_range = np.logspace(-9, 3, 13, base=2)
# 网格搜索交叉验证的参数范围, cv=3, 3折交叉
param_grid = [{'kernel': ['rbf'], 'C': c_range, 'gamma': gamma_range}]# 参数列表, 表明参数范围
grid = GridSearchCV(svc, param_grid, cv=3, n_jobs=-1)# 寻找最优参数, 传递给模型
# 训练模型
clf = grid.fit(x1_train, y1_train)
s_train = grid.score(x1_train, y1_train)
print("训练集准确率: {}".format(int(s_train*10000)/100))
s_test = grid.score(x1_test, y1_test)
print("测试集准确率: {}".format(int(s_test*10000)/100))
```

训练集准确率: 99.89%  
测试集准确率: 0.75%

这里也表现出在选用传统机器学习算法时选取有效特征的重要性。

## 6 GAN 新车型生成

兰博基尼跑车一直以优美的线条和霸气的外观广受人们的喜爱，在学习了 GAN 生成新的手写数字图片后迫不及待的想能否用 GAN 来生成一辆自己的兰博基尼？怀揣着这一想法，我们选取一系列背景简单的兰博基尼图片搭建 GAN 进行训练。

### 6.1 搭建 GAN

我们采用最简单的 GAN 模型，以下是最关键的生成器和判别器代码：

生成器	判别器
<pre>def get_generator(noise_img, n_units, out_dim, reuse=False, alpha=0.01):     """     生成器     noise_img: 生成器的输入     n_units: 隐层单元个数     out_dim: 生成器输出tensor的size, 这里应该为32*32=784     alpha: leaky ReLU系数     """     with tf.variable_scope("generator", reuse=reuse):         # hidden layer         hidden1 = tf.layers.dense(noise_img, n_units)         # Leaky ReLU         hidden1 = tf.maximum(alpha * hidden1, hidden1)         # dropout         hidden1 = tf.layers.dropout(hidden1, rate=0.2)          # logits &amp; outputs         logits = tf.layers.dense(hidden1, out_dim)         outputs = tf.tanh(logits)      return logits, outputs</pre>	<pre>def get_discriminator(img, n_units, reuse=False, alpha=0.01):     """     判别器     n_units: 隐层结点数量     alpha: Leaky ReLU系数     """     with tf.variable_scope("discriminator", reuse=reuse):         # hidden layer         hidden1 = tf.layers.dense(img, n_units)         hidden1 = tf.maximum(alpha * hidden1, hidden1)          # logits &amp; outputs         logits = tf.layers.dense(hidden1, 1)         outputs = tf.sigmoid(logits)      return logits, outputs</pre>

### 6.2 图片导入

还是我们一直面临的问题，算力不够，我们只能将原本 1920\*1080 的高清图片规整为 56\*56。

\* 作者： 印张悦、陈诺



```
def solve_image():
    batch_images = []
    for i in range(1, 11):
        imgFile = './GAN/' + str(i) + '.jpg'
        # Load an original image
        img = cv2.imread(imgFile)
        # color value range
        cRange = 256
        rows, cols, channels = img.shape
        # convert color space from bgr to gray
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (56, 56))
        imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # otsu method
        threshold, imgOtsu = cv2.threshold(imgGray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
        from PIL import Image
        new_im = Image.fromarray(imgOtsu.astype(np.uint8))
        # new_im.show()
        new_im.save('./solved_GAN/' + str(i) + '.jpg')
        # 将图片拉伸至一维
        batch_images.append(imgOtsu.reshape((1, 56*56)))
    return batch_images
```

## 6.2 生成结果

选取世代为 230，开始训练模型，以下为主要代码：

```
# batch_size
batch_size = 1
# 训练迭代轮数
epochs = 230
# 抽取样本数
n_sample = 25

# 存储测试样例
samples = []
# 存储Loss
losses = []
# 保存生成器变量
saver = tf.train.Saver(var_list = g_vars)
# 开始训练
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for e in range(epochs):
        for batch_i in range(10):

            batch_images = images[batch_i]
            # 对图像像素进行scale，这是因为tanh输出的结果介于(-1,1)，real和fake图片共享discriminator的参数
            batch_images = batch_images*2 - 1

            # generator的输入噪声
            batch_noise = np.random.uniform(-1, 1, size=(batch_size, noise_size))

            # Run optimizers
            _ = sess.run(d_train_opt, feed_dict={real_img: batch_images, noise_img: batch_noise})
            _ = sess.run(g_train_opt, feed_dict={noise_img: batch_noise})

            # 每一轮结束计算Loss
            train_loss_d = sess.run(d_loss,
                                    feed_dict = {real_img: batch_images,
                                                  noise_img: batch_noise})

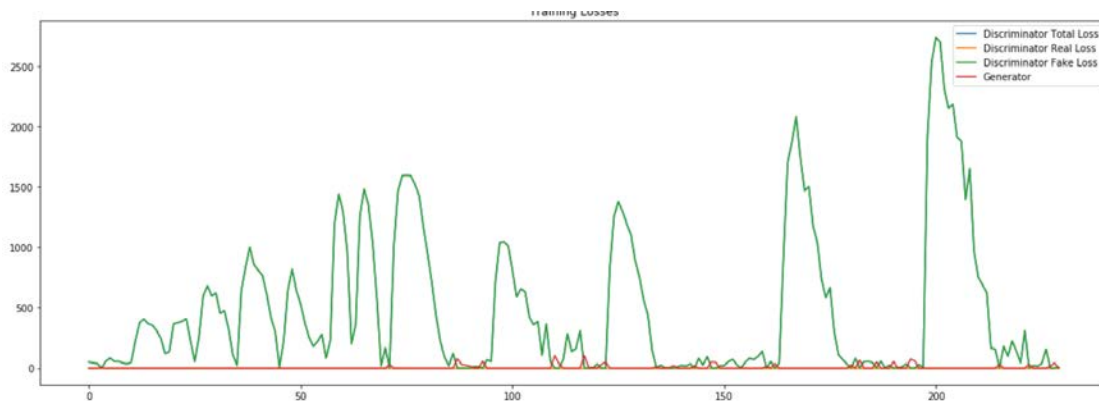
            # real img Loss
            train_loss_d_real = sess.run(d_loss_real,
                                         feed_dict = {real_img: batch_images,
                                                       noise_img: batch_noise})

            # fake img Loss
            train_loss_d_fake = sess.run(d_loss_fake,
                                         feed_dict = {real_img: batch_images,
                                                       noise_img: batch_noise})

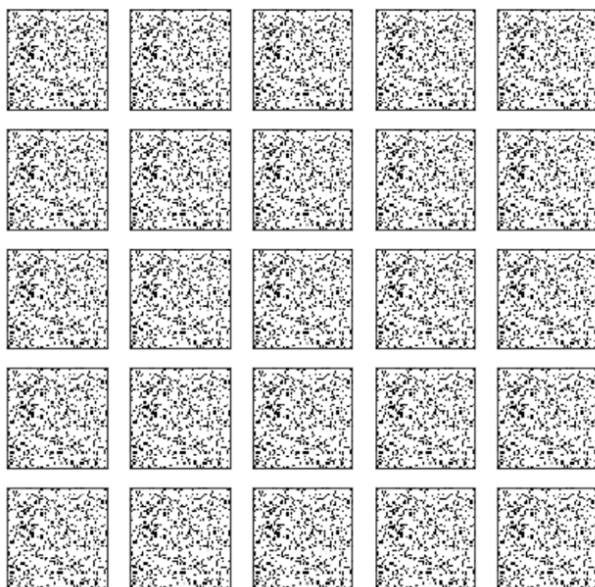
            # generator Loss
            train_loss_g = sess.run(g_loss,
                                    feed_dict = {noise_img: batch_noise})
```

训练结束后查看损失和世代的关系，发现损失的波动非常大：

\* 作者： 印张悦、陈诺

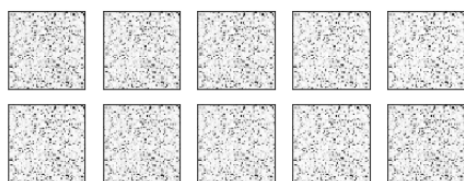


最后一轮生成的结果：



显然，结果非常糟糕，我们尝试选用不同的学习率和世代进行训练，期望得到好的结果：

```
In [16]: _ = view_samples(-1, samples) # 显示最后一轮的outputs
```



尝试多次后结果依旧不是那么理想。

总结分析：之前由于算力的原因，我们将图片规整成了极小的规模，丢失了大部分信息。而且由于车辆图片的轮廓线条十分复杂，数据量十分高，这个模型原本用于处理手写数字识别这样简单轮廓的图片，想要保持这么复杂的轮廓线条也十分困难，而且也不难发现我们的损失波动很大，始终无法收敛，还未逼近到最优值就结束了。那么高维度的图片还是应该在深度学习服务器上处理，这样可以保持图片原有的规模以及迭代更多次数，生成高质量的图片。

\* 作者： 印张悦、陈诺



# 7 Matlab 数据处理与导出

这里因为 python 部分需要用到.csv 文件,而数据原格式是.mat,故需经过一番处理。诚然,数据处理与导出按理应是较为简单的,但是对于.mat 存放的表格,则需要多一些操作。

## 7.1 .mat 部分数据

.mat 文件保存的是一些(由 matlab 程序产生/生成的)变量,由于可以是任何类型的变量,所以它可以存表格,结构体,甚至神经网络等。

cars\_train\_annos 文件存着 annotations 结构体数组,其中包含对原数据集的车辆轮廓的四个顶点坐标,车型分类(标签)以及文件地址,由['fname','bbox\_x1','bbox\_y1','bbox\_x2','bbox\_y2','class']组成,详情如下。

annotations

1x8144 struct

Fields	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname
1	39	116	569	375	14	'00001.jpg'
2	36	116	868	587	3	'00002.jpg'
3	85	109	601	381	91	'00003.jpg'
4	621	393	1484	1096	134	'00004.jpg'
5	14	36	133	99	106	'00005.jpg'
6	259	289	515	416	123	'00006.jpg'
7	88	80	541	397	89	'00007.jpg'
8	73	79	591	410	96	'00008.jpg'
9	20	126	1269	771	167	'00009.jpg'
10	21	110	623	367	58	'00010.jpg'
11	51	93	601	393	49	'00011.jpg'
12	6	62	499	286	186	'00012.jpg'
13	30	36	418	307	135	'00013.jpg'
14	31	246	778	540	85	'00014.jpg'
15	32	77	589	379	193	'00015.jpg'
16	27	49	611	396	172	'00016.jpg'
17	39	52	233	150	14	'00017.jpg'
18	3	8	190	147	73	'00018.jpg'
19	247	287	1366	761	192	'00019.jpg'
20	17	281	961	596	57	'00020.jpg'

COMMAND WINDOW

class\_names 为标签对应车型名,详情如下。

\* 作者： 印张悦、陈诺

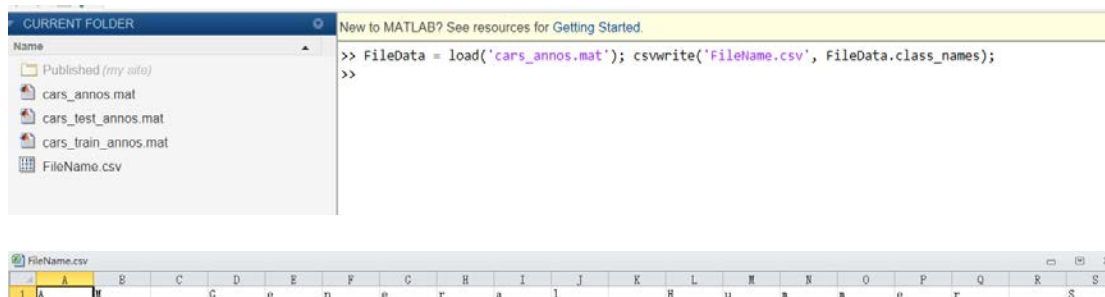
名称	值
annotations	1x16185 struct
class_names	1x196 cell

1	2	3	4	5	6	7	8	9	10	11	12
1	AM Gene...	Acura RL ...	Acura TL ...	Acura TL ...	Acura TS...	Acura Int...	Acura ZD...	Aston Ma...	Aston Ma...	Aston Ma...	Audi R ...
2											
3											
4											
5											
6											
7											
8											
9											

## 7.2 利用 Matlab 的 csvwrite 导出

开始时尝试用 Matlab 自带的库 `csvwrite` 进行导出，但是其似乎只对数字有效，对字符串会产生一个一个字符截断的效果，因此该方法失效，尝试用 `python` 进行导出。效果如下：



## 7.3 利用 python 导出

```

In [52]: annos = sio.loadmat('cars_train_annos.mat')
annos['annotations'][0][0]

Out[52]: (array([[39]], dtype=uint8), array([[116]], dtype=uint8), array([[569]], dtype=uint16), array([[375]], dtype=uint16), array([[14]], dtype=uint8), array(['00001.jpg'], dtype='<U9'))

```

可以看到其中 `cars_train_annos` 由 `['fname','bbox_x1','bbox_y1','bbox_x2','bbox_y2','class']` 组成。且由于每个单个元素莫名地被组织成列表形式（如数据 116 被整合成 `[[116]]`），故需进行处理。

```

In [27]: annos = sio.loadmat('cars_annos.mat')
print(annos)

{'_header_': b'MATLAB 5.0 MAT-file, Platform: GLNX64, Created on: Sat Feb 28 19:34:55 2015', '_version_': '1.0', '_globals_': [],
'annotations': array([[array(['car_ims/000001.jpg'], dtype='<U18'), array([[112]], dtype=uint8), array([[7]], dtype=uint8), array([[853]], dtype=uint16), array([[1717]], dtype=uint16), array([[1]], dtype=uint8), array([[0]], dtype=uint8)),
(array(['car_ims/000002.jpg'], dtype='<U18'), array([[48]], dtype=uint8), array([[24]], dtype=uint8), array([[441]], dtype=uint16), array([[202]], dtype=uint8), array([[1]], dtype=uint8), array([[0]], dtype=uint8)),
(array(['car_ims/000003.jpg'], dtype='<U18'), array([[7]], dtype=uint8), array([[4]], dtype=uint8), array([[277]], dtype=uint16), array([[180]], dtype=uint8), array([[1]], dtype=uint8), array([[0]], dtype=uint8)),
...,
(array(['car_ims/016183.jpg'], dtype='<U18'), array([[25]], dtype=uint8), array([[32]], dtype=uint8), array([[587]], dtype=uint16), array([[359]], dtype=uint16), array([[196]], dtype=uint8), array([[1]], dtype=uint8)),
(array(['car_ims/016184.jpg'], dtype='<U18'), array([[56]], dtype=uint8), array([[60]], dtype=uint8), array([[208]], dtype=uint8), array([[186]], dtype=uint8), array([[196]], dtype=uint8), array([[1]], dtype=uint8)),
(array(['car_ims/016185.jpg'], dtype='<U18'), array([[1]], dtype=uint8), array([[1]], dtype=uint8), array([[200]], dtype=uint8), array([[131]], dtype=uint8), array([[196]], dtype=uint8), array([[1]], dtype=uint8))]],
dtype=[('relative_in_path', 'O'), ('bbox_x1', 'O'), ('bbox_y1', 'O'), ('bbox_x2', 'O'), ('bbox_y2', 'O'), ('class', 'O'), ('test', 'O')]),
'class_names': array([array(['AM General Hummer SUV 2000'], dtype='<U26'),
array(['Acura RL Sedan 2012'], dtype='<U19'),

```

\* 作者： 印张悦、陈诺

```
path = annos["annotations"][:,i][0][0][0].split(".")
```

其中 ‘:’ 其实就是第 0 个数据，就与上面说的莫名被组织成列表一样。

代码如下

```
import numpy as np
import scipy.io as sio
import os
import cv2
import pandas as pd
# from keras.applications.inception_v3 import InceptionV3
# from keras.preprocessing import image
# from keras.models import Model
# from keras.layers import Dense, GlobalAveragePooling2D
# from keras import backend as K
import matplotlib.pyplot as plt
def get_labels():
    annos = sio.loadmat('cars_train_annos.mat')
    _, total_size = annos["annotations"].shape#get size
    print("total sample size is ", total_size)
    labels = np.zeros((total_size, 6))
    for i in range(total_size):
        path = annos["annotations"][:,i][0][5][0].split(".")#5th col split file name and '.jpg' by '.'
so as to discard invalid non-alphabetic path
        id = int(path[0][:]) - 1#id is to get the number of labels corresponding to the file name
        labels[id,0] = id+1
        for j in range(5):
            labels[id, j+1] = int(annos["annotations"][:,i][0][j][0])#get other numeric data
    return labels

if __name__ == "__main__":
    labels = get_labels().T#transpose for reset_index
    # print(labels)
    # np.savetxt('annotations.csv', labels, delimiter = ',')
    dfdata = pd.DataFrame(labels)
    # dfdata=dfdata.rename(index={112:1},inplace=False)
    dfdata = dfdata.reset_index(drop=False)#add index
    # dfdata=dfdata.rename(index={1:"test"},inplace=False)
    dfdata.index=['fname','bbox_x1','bbox_y1','bbox_x2','bbox_y2','class']#rename index

    dfdata.index.name = 'image'
    dfdata=dfdata.drop('index',axis=1,inplace=False)#drop invalid index part
    dfdata.columns.name = 'attribute'
    print(dfdata.T)
```

\* 作者： 印张悦、陈诺

```
dfdata = dfdata.T
datapath1 = 'annotations.csv'
dfdata.to_csv(datapath1, index=True)
效果：
```

```
total sample size is 8144
image      fname  bbox_x1  bbox_y1  bbox_x2  bbox_y2  class
attribute
0          1.0    39.0    116.0    569.0    375.0    14.0
1          2.0    36.0    116.0    868.0    587.0    3.0
2          3.0    85.0    109.0    601.0    381.0    91.0
3          4.0   621.0    393.0   1484.0   1096.0   134.0
4          5.0    14.0    36.0    133.0    99.0    106.0
5          6.0   259.0    289.0    515.0    416.0   123.0
6          7.0    88.0    80.0    541.0    397.0    89.0
7          8.0    73.0    79.0    591.0    410.0    96.0
8          9.0    20.0   126.0   1269.0    771.0   167.0
9         10.0    21.0   110.0    623.0    367.0    58.0
10         11.0    51.0    93.0    601.0    393.0    49.0
11         12.0     6.0    62.0    499.0    286.0   186.0
12         13.0    30.0    36.0    418.0    307.0   135.0
13         14.0    31.0   246.0    778.0    540.0    85.0
14         15.0    32.0    77.0    589.0    379.0   193.0
15         16.0    27.0    49.0    611.0    396.0   172.0
16         17.0    39.0    52.0    233.0    150.0    14.0
17         18.0     3.0     8.0    190.0    147.0    73.0
18         19.0   247.0   287.0   1366.0    761.0   192.0
19         20.0    17.0   281.0    961.0    596.0    57.0
20         21.0    17.0   156.0    695.0    375.0    79.0
21         22.0   212.0   538.0   1893.0   1131.0    36.0
22         23.0    11.0    28.0    476.0    234.0   120.0
23         24.0    53.0   126.0    973.0    621.0   170.0
24         25.0    34.0    87.0    567.0    343.0   194.0
25         26.0    30.0   174.0    598.0    379.0   134.0
26         27.0    80.0   107.0    606.0    336.0   184.0
27         28.0    45.0   115.0    585.0    382.0    86.0
28         29.0    28.0   139.0   1564.0   1126.0   180.0
```

获取车型部分代码如下：

```
import numpy as np
import scipy.io as sio
import os
import cv2
import pandas as pd
import matplotlib.pyplot as plt
def get_labels():
    annos = sio.loadmat('cars_annos.mat')
    _, total_size = annos["class_names"].shape
    print("total sample size is ", total_size)
    #labels = np.zeros(total_size)#pd.DataFrame(data, index=index, columns=columns)
    labels= pd.DataFrame(columns=[1],index=[i for i in range(1,total_size+1)])
    for i in range(total_size):
        labels[1][i+1]=annos["class_names"][0][i][0]
    return labels

if __name__ == "__main__":
```

\* 作者： 印张悦、陈诺

```

labels = get_labels()#.T
dfdata = pd.DataFrame(labels)
print(dfdata)

```

效果:

```

total sample size is 196
1
1          AM General Hummer SUV 2000
2          Acura RL Sedan 2012
3          Acura TL Sedan 2012
4          Acura TL Type-S 2008
5          Acura TSX Sedan 2012
6          Acura Integra Type R 2001
7          Acura ZDX Hatchback 2012
8      Aston Martin V8 Vantage Convertible 2012
9      Aston Martin V8 Vantage Coupe 2012
10     Aston Martin Virage Convertible 2012
11     Aston Martin Virage Coupe 2012
12         Audi RS 4 Convertible 2008
13         Audi A5 Coupe 2012
14         Audi TTS Coupe 2012
15         Audi R8 Coupe 2012
16         Audi V8 Sedan 1994
17         Audi 100 Sedan 1994
18         Audi 100 Wagon 1994
19         Audi TT Hatchback 2011
20         Audi S6 Sedan 2011
21         Audi S5 Convertible 2012
22         Audi S5 Coupe 2012
23         Audi S4 Sedan 2012
24         Audi S4 Sedan 2007
25         Audi TT RS Coupe 2012
26     BMW ActiveHybrid 5 Sedan 2012
27     BMW 1 Series Convertible 2012
28     BMW 1 Series Coupe 2012
29     BMW 3 Series Sedan 2012

```

最终的'annotations.csv'与 'class\_names.csv'为所求文件。

## 8 Matlab and AlexNet

### 8.1 Why Matlab

1 平时作业用的都是 Python，现用 Matlab 是一种挑战。

2 上学期的导论作业也做了 Python 与 Matlab 两个版本，发现在数据处理上 Matlab 精确度更高更鲁棒，更胜一筹（平时使用效果亦是如此），现亦想尝试做对比观察效果差异（在神经网络上）。

3 对于 AlexNet 以及 RCNN 等，Matlab 封装得很好，且利用 Matlab 的库

- Neural Network Toolbox 11.1
- Computer Vision System Toolbox 8.1
- Deep Learning Toolbox Model for AlexNet Network
- Image Processing Toolbox 10.2
- Optimization Toolbox 8.1
- Statistics and Machine Learning Toolbox 11.3

可以很好地解决神经网络、机器视觉、图像处理、统计等问题，加之平时测试代码习惯

\* 作者： 印张悦、陈诺

用 matlab，故选择了 matlab。

## 8.2 Why AlexNet

AlexNet 是一种预训练的卷积神经网络（CNN），已经对来自 ImageNet 数据集（<http://image-net.org/index>）的约 120 万张图像进行了训练。

其中有 5 个卷积层，2 层全连接和一层分类，可以将图像分为 1000 个对象类别（例如键盘，鼠标，咖啡杯，铅笔）。

它不是最新的（后者有 GoogleNet, VGG, Deep Residual Learning），但是是最经典的 CNN 之一。（如 GoogleNet 用了更多卷积）

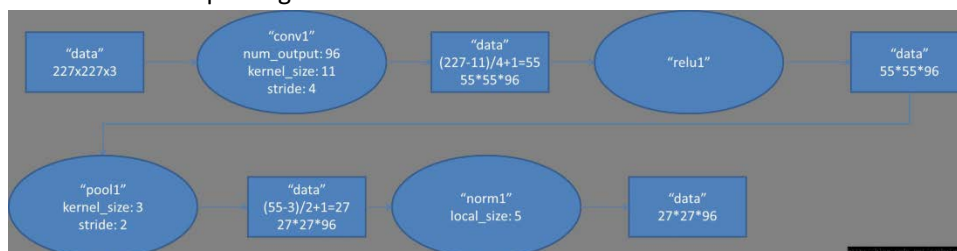
使用 AlexNet 可以对卷积神经网络的发展突破点有最全面的认识（详见 8.2.3）（因为之后模型反而简化了），所以选择了 AlexNet。

### 8.2.1 AlexNet 发展历史

- LeNet (1990s): 最初的卷积神经网络。
- 1990s to 2012: 在上世纪 90 年代后期至 2010 年初期，卷积神经网络进入孵化期。随着数据量和计算能力的逐渐发展，卷积神经网络可以处理的问题变得越来越有趣。
- AlexNet (2012) – 在 2012, Alex Krizhevsky（与其他人）发布了 AlexNet，它是比 LeNet 更深更宽的版本，并在 2012 年的 ImageNet 大规模视觉识别大赛（ImageNet Large Scale Visual Recognition Challenge, ILSVRC）中以巨大优势获胜。这对于以前的方法具有巨大的突破，当前 CNN 大范围的应用也是基于这个工作。

### 8.2.2 Alexnet 具体细节

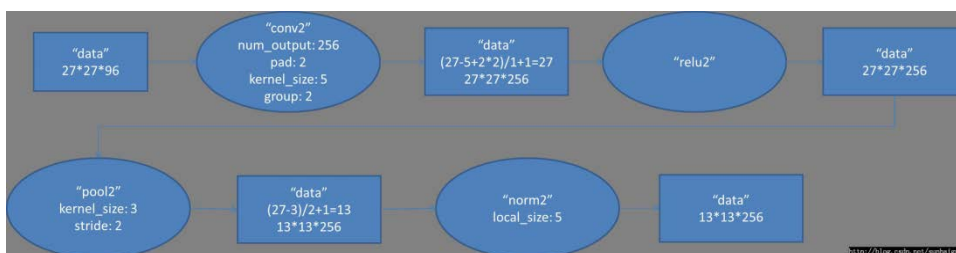
(1) conv - relu - pooling - LRN



具体计算都在图里面写了，要注意的是 input 层是 227\*227，而不是 paper 里面的 224\*224，这里可以算一下，主要是 227 可以整除后面的 conv1 计算，224 不整除。如果一定要用 224 可以通过自动补边实现，不过在 input 就补边感觉没有意义，补得也是 0。

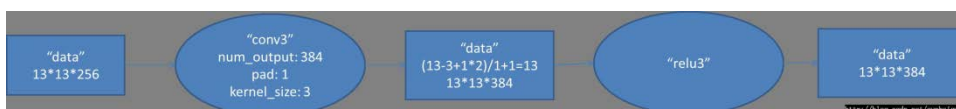
(2) conv - relu - pool - LRN

\* 作者： 印张悦、陈诺

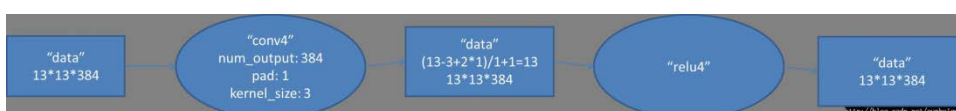


和上面基本一样，唯独需要注意的是  $\text{group}=2$ ，这个属性强行把前面结果的 feature map 分开，卷积部分分成两部分做。

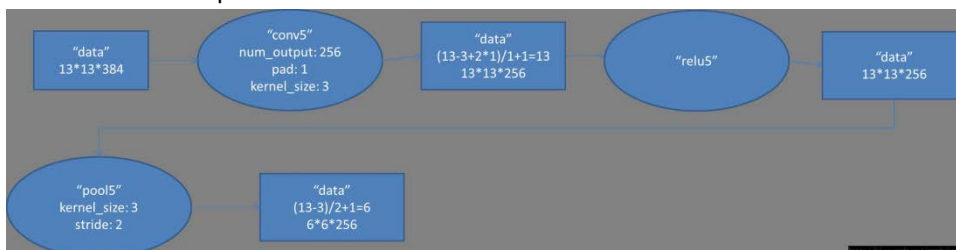
### (3) conv - relu



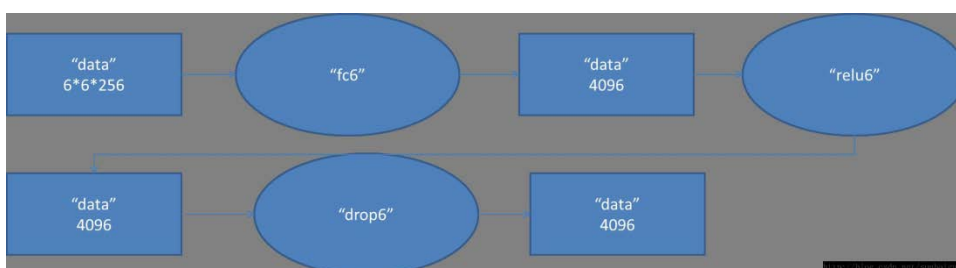
### (4) conv-relu



### (5) conv - relu - pool

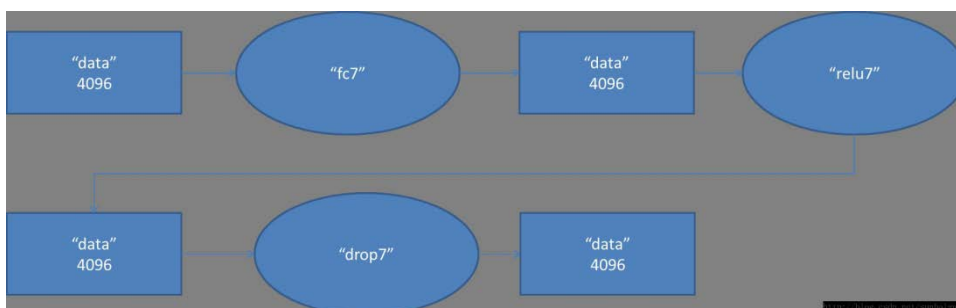


### (6) fc - relu - dropout



这里有一层特殊的 dropout 层，在 alexnet 中是说在训练的以  $1/2$  概率使得隐藏层的某些 neuron 的输出为 0，这样就丢到了一半节点的输出，BP 的时候也不更新这些节点。

### (7) fc - relu - dropout



### (8) fc - softmax

\* 作者： 印张悦、陈诺



### 8.2.3 AlexNet 为什么好用

1 首次将 ReLU 发扬光大 (比 LeNet 的 Sigmoid 好用)

2 Dropout 防止过拟合

3 用最大池化代替平均池化 避免模糊化效果

4 LRN 层增加模型泛化能力 (有争议)

但是 LRN 在 2015 年 Very Deep Convolutional Networks for Large-Scale Image Recognition 中提到 LRN 基本没什么用。

All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) non-linearity. We note that none of our networks (except for one) contain Local Response Normalisation (LRN) normalisation (Krizhevsky et al., 2012): as will be shown in Sect. 4, such normalisation does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time. Where applicable, the parameters for the LRN layer are those of (Krizhevsky et al., 2012).

后来 LRN 部分就被删去了，所以之后的手搭 AlexNet 没用到 LRN。

### 8.2.4 AlexNet 结构分析

imagenet-caffe-alex.mat 是当时训练 1000 个类时的神经网络：

有 classes, layers, normalization 三个工作区。classes 是一个 struct，下面有 classes.name

和 classes.description，都是 1\*1000 的 cell。classes.name 对于着类的标签例

如 'n02099712'，classes.description 对应着相应的类如 'Labrador retriever'。

```
convnet = helperImportMatConvNet('imagenet-caffe-alex.mat')
convnet.Layers
```

\* 作者： 印张悦、陈诺



变量 - convnet.Layers				
convnet.Layers				
	1	2	3	
1	1x1 ImageInputLayer			
2	1x1 Convolution2DLayer			
3	1x1 ReLULayer			
4	1x1 CrossChannelNormalizationLayer			
5	1x1 MaxPooling2DLayer			
6	1x1 Convolution2DLayer			
7	1x1 ReLULayer			
8	1x1 CrossChannelNormalizationLayer			
9	1x1 MaxPooling2DLayer			
10	1x1 Convolution2DLayer			
11	1x1 ReLULayer			
12	1x1 Convolution2DLayer			
13	1x1 ReLULayer			
14	1x1 Convolution2DLayer			
15	1x1 ReLULayer			
16	1x1 MaxPooling2DLayer			
17	1x1 FullyConnectedLayer			
18	1x1 ReLULayer			
19	1x1 FullyConnectedLayer			
20	1x1 ReLULayer			
21	1x1 FullyConnectedLayer			
22	1x1 SoftmaxLayer			
23	1x1 ClassificationOutputLayer			

```
>> load('imagenet-caffe-alex.mat')
>> open imagenet-caffe-alex.mat
```

ans =

包含以下字段的 [struct](#):

```
layers: {1×21 cell}
classes: [1×1 struct]
normalization: [1×1 struct]
```

（输入输出层没算进去）  
 其中一个 class 对应一千个名称和描述

\* 作者： 印张悦、陈诺

classes

classes.name

classes.description

1x1 struct 包含 2 个字段

字段

值

name

1x1000 cell

description

1x1000 cell

classes.name

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	n01440764	n01443537	n01484850	n01491361	n01494475	n01496331	n01498041	n01514668	n01514859	n01518878	n01530575	n01531178	n01532829	n01534433

classes.description

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	tench, Tin...	goldfish, ...	great whi...	tiger shar...	hammerh...	electric ra...	stingray	cock	hen	ostrich, S...	bramblin...	goldfinch...	house fin...	junco, sn...

Normalization 对应一些图片参数(注意 input 层是 227\*227)

变量 - normalization													
classes													
classes.name													
classes.description													
normalization													
1x1 struct 包含 5 个字段													
averagelImage	227x227x3 double												
keepAspect	0												
border	[29,29]												
imageSize	[227,227,3]												
interpolation	'bicubic'												

Layer 就是对应的层，并且在训练完后会有权重。

变量 - layers{1, 1}													
classes													
classes.name													
classes.description													
normalization													
layers													
layers{1, 1}													
layers{1, 1}													
weights	1x2 cell												
pad	[0,0,0,0]												
type	'conv'												
name	'conv1'												
stride	[4,4]												

以下是当年最优模型给出的 alexnet 的 CNN 结构的网站。

\* 作者： 印张悦、陈诺

## Index of /matconvnet/models/beta16

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">Parent Directory</a>		-	
<a href="#">imagenet-caffe-alex.mat</a>	2015-11-21 11:29	234M	
<a href="#">imagenet-caffe-ref.mat</a>	2015-11-21 11:29	234M	
<a href="#">imagenet-googlenet-d...&gt;</a>	2015-11-21 11:29	52M	
<a href="#">imagenet-vgg-f.mat</a>	2015-11-21 11:29	233M	
<a href="#">imagenet-vgg-m-1024.mat</a>	2015-11-21 11:29	334M	
<a href="#">imagenet-vgg-m-128.mat</a>	2015-11-21 11:29	317M	
<a href="#">imagenet-vgg-m-2048.mat</a>	2015-11-21 11:29	354M	
<a href="#">imagenet-vgg-m.mat</a>	2015-11-21 11:29	394M	
<a href="#">imagenet-vgg-s.mat</a>	2015-11-21 11:29	394M	
<a href="#">imagenet-vgg-verydee...&gt;</a>	2015-11-21 11:29	529M	
<a href="#">imagenet-vgg-verydee...&gt;</a>	2015-11-21 11:30	549M	
<a href="#">pascal-fcn16s-dag.mat</a>	2015-11-21 11:30	514M	
<a href="#">pascal-fcn32s-dag.mat</a>	2015-11-21 11:30	519M	
<a href="#">pascal-fcn8s-dag.mat</a>	2015-11-21 11:30	513M	
<a href="#">pascal-fcn8s-tvg-dag...&gt;</a>	2015-11-21 11:30	513M	
<a href="#">vgg-face.mat</a>	2015-11-21 11:30	1.0G	

而 matlab 库

Deep Learning Toolbox Model for AlexNet Network

### File Exchange

MATLAB Central ▾ Files Authors My File Exchange Contribute About



### Deep Learning Toolbox Model for AlexNet Network

by MathWorks Deep Learning Toolbox Team **STAFF**

Pretrained AlexNet network model for image classification

Overview

net = alexnet

convnet = helperImportMatConvNet(net)

convnet.Layers

\* 作者： 印张悦、陈诺

1	'input'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'fc7'	Fully Connected	4096 fully connected layer
20	'relu7'	ReLU	ReLU
21	'fc8'	Fully Connected	1000 fully connected layer
22	'prob'	Softmax	softmax
23	'classificationLayer'	Classification Output	crossentropyex with 'n01440764' and 999 other classes

也是 23 层的，其中有 5 个卷积层，2 层全连接和一层分类。

但是，我两者都没用，而选择照着进行手搭，这样改中间层方便一些。

```
function [AlexLayer,opts]=alexnet()
```

```
inputLayer = imageInputLayer([227 227 3],'Name','Input');%指定图像大小 227*227*3
```

```
middleLayers = [
```

```
convolution2dLayer([11 11], 96,'NumChannels',3,'Stride',4,'Name','conv1','Padding',0)
```

%过滤器的高度和宽度 过滤器数量（连接到同一输入区域的神经元数量 其决定了特征图的数量）

```
reluLayer('Name','relu1')
```

```
crossChannelNormalizationLayer(5,'Name','norm1')
```

```
maxPooling2dLayer(3, 'Stride', 2,'Name','pool1','Padding',0)
```

```
convolution2dLayer([5 5], 256, 'NumChannels',48,'Padding', 2,'Name','conv2','Stride',1)
```

```
reluLayer('Name','relu2')
```

```
crossChannelNormalizationLayer(5,'Name','norm2')
```

```
maxPooling2dLayer(3, 'Stride',2,'Name','pool2','Padding',0)
```

```
convolution2dLayer([3 3], 384, 'NumChannels',256,'Padding', 1,'Name','conv3','Stride',1)
```

```
reluLayer('Name','relu3')
```

```
convolution2dLayer([3 3], 384,'NumChannels',192, 'Padding', 1,'Name','conv4','Stride',1)
```

```
reluLayer('Name','relu4')
```

```
convolution2dLayer([3 3], 256, 'NumChannels',192,'Padding', 1,'Name','conv5','Stride',1)
```

```
reluLayer('Name','relu5')
```

```
maxPooling2dLayer(3, 'Stride',2,'Name','pool5','Padding',0)
```

\* 作者： 印张悦、陈诺

```

];

finalLayers = [
fullyConnectedLayer(4096,'Name','fc6')
reluLayer('Name','relu6')
%caffe 中有这一层
dropoutLayer(0.5,'Name','dropout6')

fullyConnectedLayer(4096,'Name','fc7')
reluLayer('Name','relu7')
%caffe 中有这一层
dropoutLayer(0.5,'Name','dropout7')

%196 种车
fullyConnectedLayer(196,'Name','fc8')
softmaxLayer('Name','softmax')
classificationLayer('Name','classification')
];

AlexLayer=[inputLayer
    middleLayers
    finalLayers];

opts = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'InitialLearnRate', 0.001, ...
    'LearnRateSchedule', 'piecewise', ...
    'MaxEpochs', 500, ...
    'MiniBatchSize', 100, ...
    'Verbose', true);
end

```

\* 作者： 印张悦、陈诺

1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench' and 999 other classes

其中多加了两个 **dropout** 层，作用是对神经网络单元按一定改了将其暂时从网络中丢弃，能防止过拟合。

## 9 Matlab AlexNet 预测车型

### 9.1 调整最后三层

接着上文，我们要对其进行微调，那么其实前面的卷积层都不用改，要改的就是最后的一个全连接层，要把它改成我们的层。由于车型一共是 **196** 种，所以全连接的输出也得改成 **196**，后面再接上一个 **softmax** 层和一个 **classificationLayer**,并定义训练方式及运行。

```
function [AlexLayer_New , optionsTransfer]=FineTune(AlexNet)%改最后三层
AlexNet_reduce = AlexNet.Layers(1:end-3);
%add
Last3Layers = [
fullyConnectedLayer(196,'Name','fc8','WeightLearnRateFactor',10, 'BiasLearnRateFactor',20)
softmaxLayer('Name','softmax')
classificationLayer('Name','classification')
];
AlexLayer_New=[AlexNet_reduce
    Last3Layers];

optionsTransfer = trainingOptions('sgdm',...%有动量的随机梯度下降
    'MaxEpochs',10,...
```

\* 作者： 印张悦、陈诺

```

        'InitialLearnRate',0.0005,...
        'Verbose',true,'MiniBatchSize', 100);%MiniBatchSize 根据显卡内存而定
end
%%

```

## 新层级

23x1 [Layer](#) array with layers:

1	'input'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'fc7'	Fully Connected	4096 fully connected layer
20	'relu7'	ReLU	ReLU
21	'fc8'	Fully Connected	196 fully connected layer
22	'softmax'	Softmax	softmax
23	'classification'	Classification Output	crossentropyex

\* 作者： 印张悦、陈诺

```
optionsTransfer =
```

[TrainingOptionsSGDM](#) - 属性:

```
            Momentum: 0.9000
        InitialLearnRate: 5.0000e-04
LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
            MaxEpochs: 10
        MiniBatchSize: 100
            Verbose: 1
        VerboseFrequency: 50
        ValidationData: []
ValidationFrequency: 50
        ValidationPatience: 5
            Shuffle: 'once'
        CheckpointPath: ''
ExecutionEnvironment: 'auto'
        WorkerLoad: []
        OutputFcn: []
            Plots: 'none'
        SequenceLength: 'longest'
SequencePaddingValue: 0
```

## 9.2 导入图片与训练

我们取前 6000 个作为训练集。导入训练集 6000 张图，给每张图加上标签，并且观察标签有没有成功贴上，`unique` 返回 196 说明训练集已贴上标签。

```
%%
```

```
[AlexLayer_New , optionsTransfer]=FineTune(AlexNet)
```

```
%%
```

```
traindata=imageDatastore('cars_train_cropped(227_227)','LabelSource','none')
```

```
%%
```

```
%load 'cars_meta.mat';
```

```
%annotations.class 即为标注信息
```

```
ac=[annotations.class]
```

```
* 作者： 印张悦、陈诺
```



```

traindata.Labels=categorical(ac(1:6000))%取前 6000 个进行测试
%traindata.Labels=categorical([class_names])
%%
unique(traindata.Labels)%看数量是不是 196 个

```

数据集给出的识别出车的部分，然后 `resize` 成大小 `227*227`，将图片导出到 `cars_train_cropped(227_227)` 中。

```

>> load('cars_train_annos.mat')

>> ac=[annotations.class]

>> traindata.Labels=categorical(ac(1:6000))

traindata =

  ImageDatastore - 属性:

      Files: {
          'G:\cars lab\code\cars_train_cropped(227_227)\00001.jpg';
          'G:\cars lab\code\cars_train_cropped(227_227)\00002.jpg';
          'G:\cars lab\code\cars_train_cropped(227_227)\00003.jpg'
          ... and 5997 more
      }
      Labels: [14; 3; 91 ... and 5997 more categorical]
AlternateFileSystemRoots: {}
      ReadSize: 1
      ReadFcn: [function_handle]

>> unique(traindata.Labels)

ans =

    196×1 categorical 数组

```

下表为 `annotations`，其中前四列为车辆边界数据，具体会在第十章介绍。

\* 作者： 印张悦、陈诺

变量 - annotations

annotations

1x8144 struct 包含 6 个字段

字段	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname
1	39	116	569	375	14	'00001.jpg'
2	36	116	868	587	3	'00002.jpg'
3	85	109	601	381	91	'00003.jpg'
4	621	393	1484	1096	134	'00004.jpg'
5	14	36	133	99	106	'00005.jpg'
6	259	289	515	416	123	'00006.jpg'
7	88	80	541	397	89	'00007.jpg'
8	73	79	591	410	96	'00008.jpg'
9	20	126	1269	771	167	'00009.jpg'
10	21	110	623	367	58	'00010.jpg'
11	51	93	601	393	49	'00011.jpg'

之后进行训练。

%%

AlexNet\_New=trainNetwork(traindata,AlexLayer\_New,optionsTransfer)

```
>> AlexNet_New=trainNetwork(traindata,AlexLayer_New,optionsTransfer)
Training on single CPU.
Initializing image normalization.
=====
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Base Learning |
|       |          | (hh:mm:ss)  | Accuracy   | Loss        | Rate          |
|=====|=====|=====|=====|=====|=====|
| 1 | 1 | 00:00:07 | 0.00% | 5.5827 | 0.0005 |
| 1 | 50 | 00:04:10 | 15.00% | 3.6646 | 0.0005 |
| 2 | 100 | 00:08:16 | 54.00% | 1.8063 | 0.0005 |
| 3 | 150 | 00:12:22 | 73.00% | 0.9562 | 0.0005 |
| 4 | 200 | 00:16:30 | 91.00% | 0.4251 | 0.0005 |
| 5 | 250 | 00:20:37 | 94.00% | 0.2490 | 0.0005 |
| 5 | 300 | 00:24:43 | 95.00% | 0.1786 | 0.0005 |
| 6 | 350 | 00:28:50 | 95.00% | 0.1752 | 0.0005 |
| 7 | 400 | 00:32:57 | 100.00% | 0.0356 | 0.0005 |
| 8 | 450 | 00:37:04 | 97.00% | 0.0621 | 0.0005 |
| 9 | 500 | 00:41:19 | 100.00% | 0.0181 | 0.0005 |
| 10 | 550 | 00:45:26 | 99.00% | 0.0185 | 0.0005 |
| 10 | 600 | 00:49:34 | 98.00% | 0.0382 | 0.0005 |
|=====|=====|=====|=====|=====|=====|

AlexNet_New =
|
| SeriesNetwork - 属性:
|
|     Layers: [23x1 nnet.cnn.layer.Layer]
```

\* 作者： 印张悦、陈诺

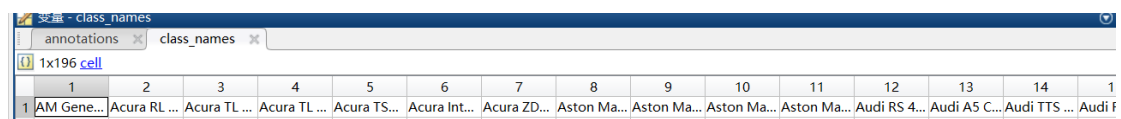
可以看到，对于  $227 \times 227 \times 3$  的 6000 张图，一个 25 层的 AlexNet 只需要训练不到 50 分钟，可见 Matlab 的速度之快。

## 9.3 测试

接下来让 8144 张图中后 2000 张作为测试集，并进行测试。

读入车型文件

```
>> load('cars_meta.mat')
```



1	2	3	4	5	6	7	8	9	10	11	12	13	14	1
1 AM Gene...	Acura RL ...	Acura TL ...	Acura TL ...	Acura TS...	Acura Int...	Acura ZD...	Aston Ma...	Aston Ma...	Aston Ma...	Aston Ma...	Audi RS 4...	Audi A5 C...	Audi TTS ...	Audi f

然后封装测试代码

```
%%  
function test(fileRoad,AlexNet_New,class_names)#封装测试  
testImage=imread(fileRoad);  
testImage_=imresize(testImage,[227 227]);  
TypeNum=classify(AlexNet_New,testImage_);  
TypeName=class_names(TypeNum);  
disp(TypeName);  
figure;  
imshow(testImage);  
end  
%%
```

测试准确率

```
aclass=[annotations.class];  
afname={annotations.fname};  
%取后两千个进行测试  
true=0;  
all=2000;  
for n = 6001:(6001+all)  
    TypeNum=test(['cars_train_nocrop/' afname{n}],AlexNet_New,class_names);  
    cstr=cellstr(TypeNum(1));  
    strnum=cstr{1};  
    numb=str2num(strnum);  
    if (numb==aclass(n))
```

\* 作者： 印张悦、陈诺

```

        true=true+1;
    end
end
disp('correct rate');
disp(true/all);

```

The screenshot shows the MATLAB editor with the file `test_m.m` open. The script contains a loop for testing 2000 images and a function `test` that uses `AlexNet_New` for classification. The command window on the right displays a table of training progress and the final test results.

Epoch	Iteration	Time Elapsed (hh:mm:ss)
1	1	00:00:07
1	50	00:04:10
2	100	00:08:16
3	150	00:12:22
4	200	00:16:30
5	250	00:20:37
5	300	00:24:43
6	350	00:28:50
7	400	00:32:57
8	450	00:37:04
9	500	00:41:19
10	550	00:45:26
10	600	00:49:34

Below the table, the command window shows the output of the `test` function:

```

AlexNet_New =
SeriesNetwork - 属性:
    Layers: [23x1 rnet.cnn.layer.Layer]

>> test_
correct rate
    0.5110

```

2000 张图测试，只花了不到 3 分钟。

## 9.5 准确率预测

可以看到最终准确率为 **51.10%**，比 python 的其他神经网络的最高结果 13% 高出了很大一截，体现了 AlexNet 与 CNN 的优势之处。（当然与其他神经网络需要把 `rgb` 分开也有关）。

## 10 小插曲：RCNN 区域检测

这里对上文提及的边界进行补充说明。

\* 作者： 印张悦、陈诺

annotations						
1x8144 struct						
Fields	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname
1	39	116	569	375	14	'00001.jpg'
2	36	116	868	587	3	'00002.jpg'
3	85	109	601	381	91	'00003.jpg'
4	621	393	1484	1096	134	'00004.jpg'
5	14	36	133	99	106	'00005.jpg'
6	259	289	515	416	123	'00006.jpg'
7	88	80	541	397	89	'00007.jpg'
8	73	79	591	410	96	'00008.jpg'
9	20	126	1269	771	167	'00009.jpg'
10	21	110	623	367	58	'00010.jpg'
11	51	93	601	393	49	'00011.jpg'
12	6	62	499	286	186	'00012.jpg'
13	30	36	418	307	135	'00013.jpg'
14	31	246	778	540	85	'00014.jpg'
15	32	77	589	379	193	'00015.jpg'
16	27	49	611	396	172	'00016.jpg'
17	39	52	233	150	14	'00017.jpg'
18	3	8	190	147	73	'00018.jpg'
19	247	287	1366	761	192	'00019.jpg'
20	17	281	961	596	57	'00020.jpg'
COMMAND WINDOW						

## 10.1 起因

首先肯定的是，如果直接对原图形进行训练，会有很大一部分的背景影响对车辆的识别，而体现不出车辆区域部分的重要性，比如在图片数据集中有很多这样的图，极为影响训练效果。



\* 作者： 印张悦、陈诺

上图为图片 00331



上图为图片 00041

这里我的想法是利用 RCNN 进行车辆区域检测（根据现有的车的轮廓数据）并生成四个坐标点。

虽然比较多余（因为数据集里已经给了区域），但是可以直观了解到现有的轮廓数据的准确性，也了解了原图片的区域范围是如何用 RCNN 生成的。

## 10.1 过程

由于这部分不是本项目重点，故篇幅作了限制。

既然用到 CNN 就自然想到拿 cifar10 开刀（第三次大作业所用的数据集）。

恰好 Matlab 在 computer vision 包里放了用于训练 cifar10 的神经网络 cifar10Net，由于该任务极为追求效果，所以选择直接利用现有模型。

找到了对应的.mat 并 load 进来

\* 作者： 印张悦、陈诺

```

>> load('cifar10NetRCNN.mat', 'cifar10NetRCNN')
>> cifar10Net=cifar10NetRCNN
|
cifar10Net =

    rcnnObjectDetector - 属性:

        Network: [1x1 SeriesNetwork]
    RegionProposalFcn: @rcnnObjectDetector.proposeRegions
        ClassNames: {'CarPosition' 'Background'}

>> cifar10Net.Network.Layers

ans =

    15x1 Layer array with layers:

     1 'imageinput'    Image Input          32x32x3 images with 'zerocenter' normalization
     2 'conv'          Convolution          32 5x5x3 convolutions with stride [1 1] and padding [2 2 2 2]
     3 'relu'          ReLU                  ReLU
     4 'maxpool'        Max Pooling           3x3 max pooling with stride [2 2] and padding [0 0 0 0]
     5 'conv_1'         Convolution          32 5x5x32 convolutions with stride [1 1] and padding [2 2 2 2]
     6 'relu_1'         ReLU                  ReLU
     7 'maxpool_1'       Max Pooling           3x3 max pooling with stride [2 2] and padding [0 0 0 0]
     8 'conv_2'         Convolution          64 5x5x32 convolutions with stride [1 1] and padding [2 2 2 2]
     9 'relu_2'         ReLU                  ReLU
    10 'maxpool_2'       Max Pooling           3x3 max pooling with stride [2 2] and padding [0 0 0 0]
    11 'fc'             Fully Connected       64 fully connected layer
    12 'relu_3'         ReLU                  ReLU
    13 'fc_rcnn'        Fully Connected       2 fully connected layer
    14 'softmax'        Softmax               softmax
    15 'classoutput'    Classification Output  crossentropyex with classes 'CarPosition' and 'Background'

```

可以看到只有 3 个卷积层，总共的层数也少了 8 层，权重参数肯定是下降了一个数量级。

现在将最后的全连接层改成自己训练的 RCNN 层：

（Matlab 中 trainRCNNObjectDetector 这个函数可以将自己训练的 RCNN 层换到原来神经网络的层）

\* 作者： 印张悦、陈诺



```

1 %%
2 load('cifar10NetRCNN.mat', 'cifar10NetRCNN');
3 cifar10Net=cifar10NetRCNN;
4 cifar10Net.Network.Layers
5 %%
6 bxl=[annotations.bbox_x1];
7 bx2=[annotations.bbox_x2];
8 by1=[annotations.bbox_y1];
9 by2=[annotations.bbox_y2];
10 data=[bx1 bx2 by1 by2];
11
12 options=trainingOptions('sgdm', ...
13     'Momentum', 0.9, ...
14     'InitialLearnRate', 0.005, ...
15     'LearnRateSchedule', 'piecewise', ...
16     'LearnRateDropFactor', 0.1, ...
17     'LearnRateDropPeriod', 128, ...
18     'L2Regularization', 0.01, ...
19     'MaxEpochs', 100, ...
20     'MiniBatchSize', 100, ...
21     'Verbose', true);
22 cifar10NetRCNN_new=trainRCNNObjectDetector(data, cifar10Net, options, ...
23     'NegativeOverlapRange', [0 0.2], 'PositiveOverlapRange', [0.7 1])
24 %可以改现有的该网络的全连接层为我训练的RCNN层
25 %%
26 testImage=imread('cars_train_nocrop/00020.jpg');
27 testImage_=imresize(testImage, [227 227]);
28 TypeNum=classify(cifar10NetRCNN_new, testImage_);
29 TypeName=class_names(TypeNum);
30 disp(TypeName);
31 figure;
32 imshow(testImage);

```

这里的 data 要处理成这样的形式，以匹配 trainRCNNObjectDetector 函数。

\* 作者： 印张悦、陈诺



CarPosition
[39,116,530,259]
[36,116,832,471]
[85,109,516,272]
[621,393,863,703]
[14,36,119,63]
[259,289,256,127]
[88,80,453,317]
[73,79,518,331]
[20,126,1249,645]
[21,110,602,257]
[51,93,550,300]
[6,62,493,224]
[30,36,388,271]
[31,246,747,294]
[32,77,557,302]
[27,49,584,347]

换完后利用训练好的 `cifar10NetRCNN_new` 进行测试。

```

25      %%
26 —    testImage=imread('cars_train_nocrop/00020.jpg');
27 —    testImage_=imresize(testImage,[227 227]);
28 —    TypeNum=classify(cifar10NetRCNN_new,testImage_);
29 —    TypeName=class_names(TypeNum);
30 —    disp(TypeName);
31 —    figure;
32 —    imshow(testImage);

```

**Classify**

的训练过程:

\* 作者： 印张悦、陈诺

	90		7250		10257.54		0.0001		100.00%		0.000500
	91		7300		10333.43		0.0001		100.00%		0.000500
	91		7350		10405.00		0.0001		100.00%		0.000500
	92		7400		10479.20		0.0001		100.00%		0.000500
	92		7450		10551.19		0.0001		100.00%		0.000500
	93		7500		10620.86		0.0001		100.00%		0.000500
	94		7550		10694.82		0.0001		100.00%		0.000500
	94		7600		10769.87		0.0002		100.00%		0.000500
	95		7650		10840.41		0.0001		100.00%		0.000500
	96		7700		10912.29		0.0001		100.00%		0.000500
	96		7750		10986.63		0.0001		100.00%		0.000500
	97		7800		11061.37		0.0001		100.00%		0.000500
	97		7850		11134.11		0.0001		100.00%		0.000500
	98		7900		11207.58		0.0001		100.00%		0.000500
	99		7950		11283.43		0.0102		99.00%		0.000500
	99		8000		11359.77		0.0001		100.00%		0.000500
	100		8050		11433.39		0.0001		100.00%		0.000500

以及最后结果：



可以看到对新图识别的效果相对还行，由于原始图片数据已有边界，故就不画蛇添足了。

\* 作者： 印张悦、陈诺

# 11 总结

- 1.采用 OpenCV 进行数据预处理，将图片规整为 64\*64。
- 2.用 Keras 搭建主流的 CNN 模型，得到了不错的训练效果。
- 3.面对高维数据，采用 PCA 降维，用 Keras 搭建 DNN 模型进行训练，由于丢失了大量数据，训练效果不佳。
- 4.机器学习算法，选取 RGB 像素为特征，由于特征过多，导致模型训练效果不佳。
- 5.采用 OpenCV 的 Otus 进行背景分离，再次使用机器学习算法得到了不错的提升。
- 6.采用 GAN 生成新的兰博基尼图片，由于算力原因将图片规整为 56\*56，丢失了大部分数据以及轮廓线条，因此最后生成的图片与理想差距甚远。

写在最后：通过本次实验，对于人工智能的许多方面有了一个大致地了解，了解了很多机器学习算法，如：决策树、随机森林、SVM 等，也自己搭建了自己的神经网络，了解了不同类型的神经网络的应用，如 CNN、DNN、GAN 等。在搭建过程中遇到了很多困难，如数据如何放入神经网络，用何种形式放入神经网络进行训练，在同学老师的帮助下也都一一克服。在实验过程中对于这个数据集也产生了很多新想法，如 PCA 降维+DNN 模型处理图片、Otus 边缘检测以提取有效特征提高传统机器学习算法的准确率等，通过实验实践了自己的想法，觉得颇有收获。

印张悦

2019 年 6 月 13 日星期四

- 7.对于.mat 结构体中字符串数据，利用 csvwrite 导出会产生错误，因此用 python 处理后导出。
- 8.通过使用 AlexNet 的结构，使准确率达到了 51.10%，相比自己搭建的模型是巨大的飞跃。
- 9.在 python 与 matlab 效率方面，matlab 对于矩阵运算的优化着实惊人，可以轻松处理 10.227\*227\*3 的 6000 张图，一个 25 层的 AlexNet 只需要训练不到 50 分钟，对 2000 张图进行测试也仅需要 3 分钟。
- 11.由于有背景占较大部分的图片，故需要先通过 RCNN 进行区域检测，以框定范围。

陈诺

2019 年 6 月 20 日星期四

## 参考文献

- [1] <http://m.myexception.cn/perl-python/1849015.html>
- [2] <https://blog.csdn.net/xiewenbo/article/details/79354812>