猫狗大战

机器学习工程师纳米学位毕业项目

桂骏马

2018 年 5 月 8 日

1. 定义

1.1 项目概览

这是 2013 年 Kaggle 一个娱乐的竞赛项目，猫狗图片分类。提供的数据集包含训练集 25000 张猫和狗图片，测试集 15000 张猫和狗的图片。用训练集图片数据，训练一个机器学习的模型，预测测试集中图片是狗还是猫。

这是一个标准的图像分类问题， 本项目采用基于卷积神经网络 CNN 的深度学习（Deep Learning）算法来构建模型。

神经网络是 2012 年重新兴起的热门算法，Geffory Hinton 的学生 Alex Krizhevsky 提出 AlexNet，在 2012 年 ILSVRC 图像分类竞赛中一举夺得冠军，top-5 错误率比上一年的冠军（SVM 算法，手动提取特征）25.7%下降了 10%左右 达到 15.3%，而且远远领先当年的第二名（26.2%）。从此之后基于神经网络的深度网络模型占据了每年的 ILSVRC 图像分类竞赛 top 排行榜。

1.2 问题陈述

Kaggle 提供的图片集，样式风格多样，图片大小不同，猫和狗的品种也很多。很好地体现真实物理世界的复杂性，对模型特征选取的挑战很高。神经网络自动学习特征的优点，非常适合做图片分类任务。
输入图片像素 RGB 数据，构建深层神经网络，第一层学习基本的边沿形状，后续层学习更复杂的结构特征如对象的轮廓，纹理，颜色等。根据学习到的特征对应的分类标签，经过大量的图片数据训练出模型，应用最终的模型对测试集数据预测每张图片是狗的概率（狗的概率以 1 表示，猫的概率以 0 表示）

1.3 评价标准

Kaggle 猫狗大战的评分标准采用对数损失 LogLoss，越小越好。

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

- n is the number of images in the test set
- $\hat{y}_i$ is the predicted probability of the image being a dog
- y_i is 1 if the image is a dog, 0 if cat
- $\log()$ is the natural (base e) logarithm

log loss 作为对预测概率的评分标准是非常合适的

1. 真实概率即测试集图片是狗的概率分布是服从伯努利分布（0-1 分布）
2. 对数损失是度量了真实概率分布与预测概率分布之间的差异，这里的差异度量用的是 KL 散度，等效于求最大似然估计。所以对数损失值越小越好！

2. 分析

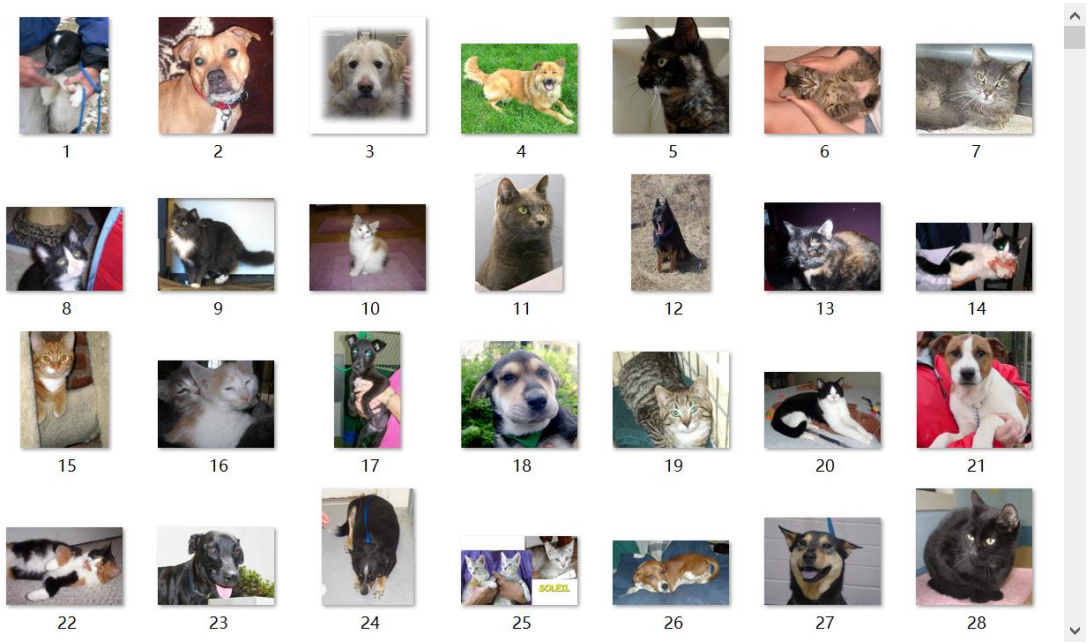
2.1 样本数据分析

训练集 25000 张图片，分别是 12500 张猫的图片和 12500 张狗的图片。图片以类别+数字作为文件名，图片格式是 jpg。

猫的图片

狗的图片

测试集 15000 张图片，以数字作为文件名，没有标签，图片格式是 jpg。

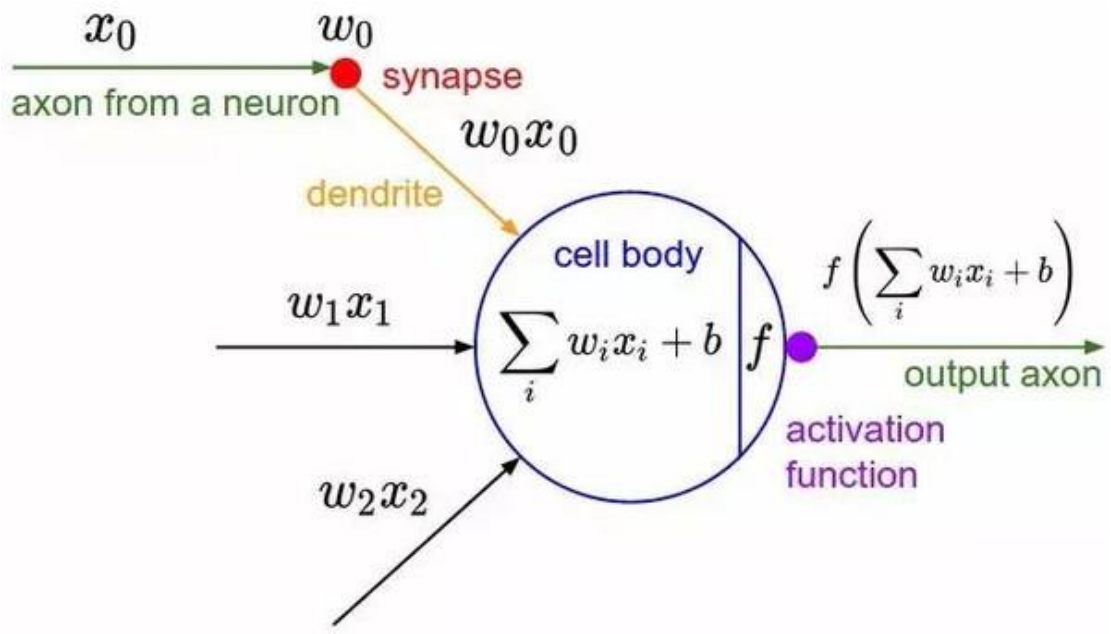


训练集是平衡样本，猫和狗各一半。有利于训练模型正确识别猫和狗的特征，不会偏向于一类。

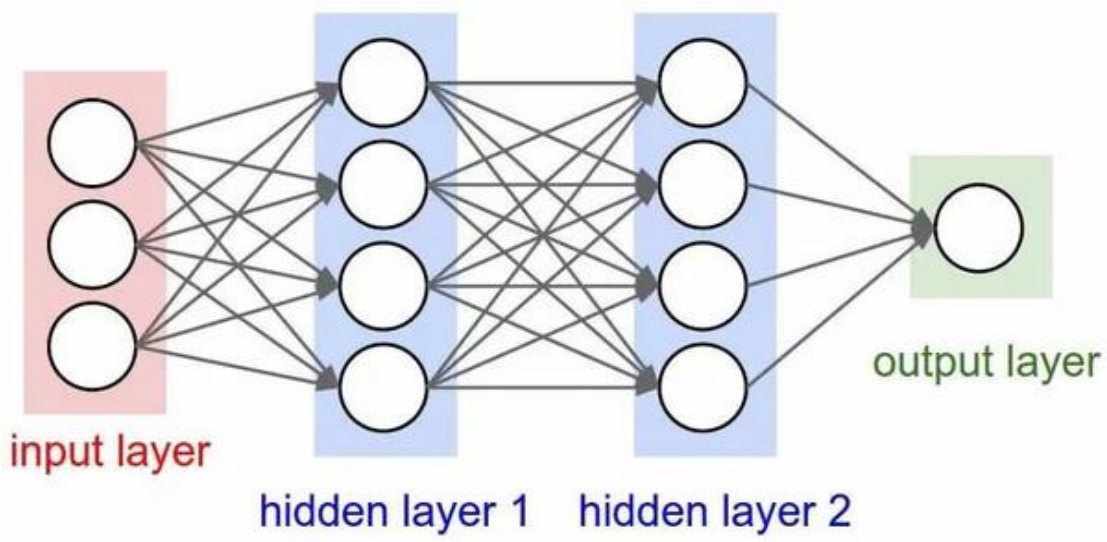
2.2 算法分析

卷积神经网络原理的介绍

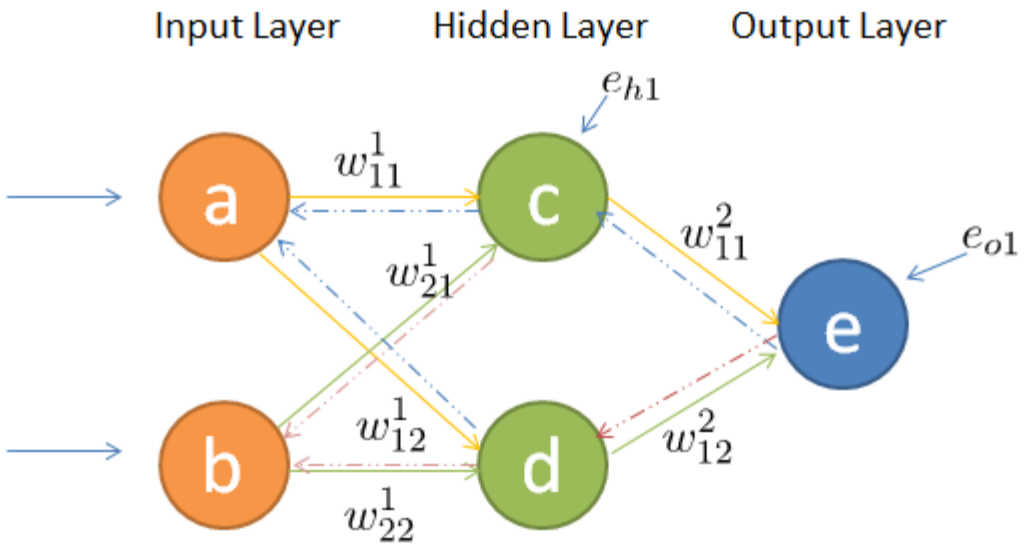
1943 年，McCulloch 和 Pitts 提出 **M-P 神经元模型**。1956 年 Frank Rosenblatt 提出了感知机(Perceptron)模型加入了权值计算。神经元接收来自 n 个其他神经元传递过来的输入信号 x，这些输入信号通过带权重 w 的连接(connection)进行传递，神经元接收到的总输入值将与神经元的阈值进行比较，然后通过“激活函数”(activation function) f 处理产生神经元的输出。感知机成为神经网络的基础单元。



后续发展出多层神经网络，拟合更为复杂的多维非线性任意函数。



1986 年 David Rumelhart 等学者出版的《平行分布处理:认知的微观结构探索》一书。书中完整地提出了 BP 算法,系统地解决了多层网络中隐单元连接权的学习问题,并在数学上给出了完整的推导。这是神经网络发展史上的里程碑，BP 算法迅速走红，掀起了神经网络的第二次高潮。



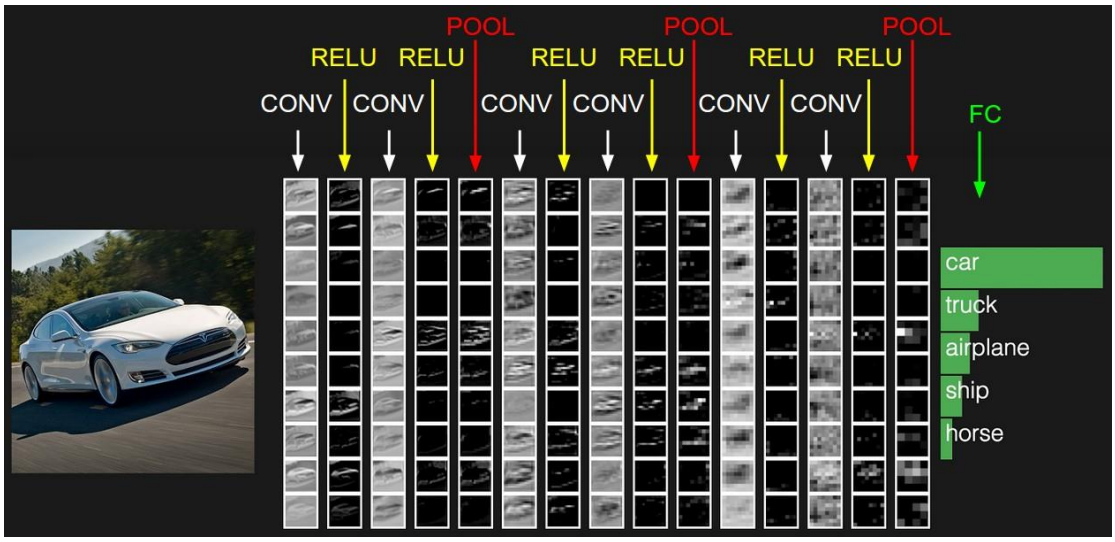
反向传播是利用链式法则递归计算表达式的梯度的方法，基于初始化权值前向计算出损失函数，得出预测值和真实值误差，并采用 Mini-batch 随机梯度下降的方法更新权值。

1988 年 Yann Lecun 在贝尔实验室 提出卷积神经网络（Convolutional Nerual Networks）对手写字母达到很高的识别率。2012 年 Alex Krizhevsky 提出 AlexNet (5 个卷积层+2 个全连接层+1 个 softmax 层) 夺得 ImageNet 冠军，迎来基于神经网络的深度学习的全面兴起，并应用到各个领域。

卷积神经网络通常包含以下几种层

- 卷积层（Convolutional layer），卷积神经网路中每层卷积层由若干卷积单元组成，每个卷积单元的参数都是通过反向传播算法优化得到的。卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网络能从低级特征中迭代提取更复杂的特征。
- 激活层（Activation layer），这一层神经的活性化函数（Activation function）现在通常使用 ReLU，某些情况下使用 sigmoid 等
- 池化层（Pooling layer），通常在卷积层之后会得到维度很大的特征，将特征切成几个区域，取其最大值或平均值，得到新的、维度较小的特征。
- 全连接层（ Fully-Connected layer），把所有局部特征结合变成全局特征，用来计算最后每一类的 Loss。

一个卷积神经网络各层应用实例：



卷积（Convolution）

卷积神经网络通过两种方法可以大幅降低全连接神经网络的参数数目

1．局部感受野

假定一个大小为 5×5 的图像，和一个 3×3 的卷积核。卷积核实际上有 9 个神经元，他们的输出又组成一个 3×3 的矩阵，称为特征图。第一个神经元连接到图像的第一个 3×3 的局部，第二个神经元则连接到第二个局部。连续扫描完整幅图像，完成一次图像的卷积运算。

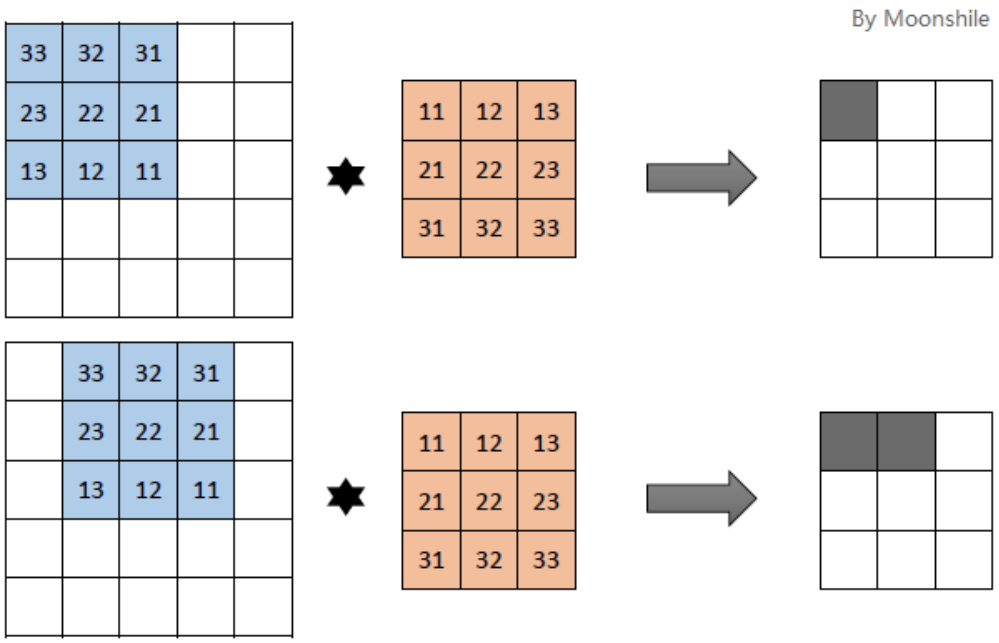
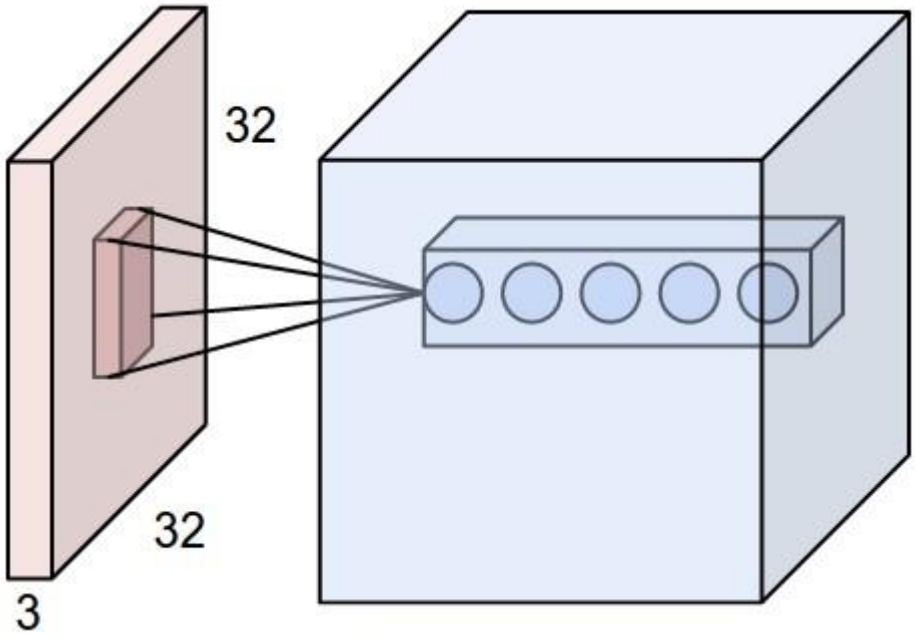


Fig. 2 Convolution Process

卷积层的参数包含一系列过滤器（filter），每个过滤器训练一个深度，有几个过滤器输出单元就具有多少深度，对应不同的特征图(feature map)

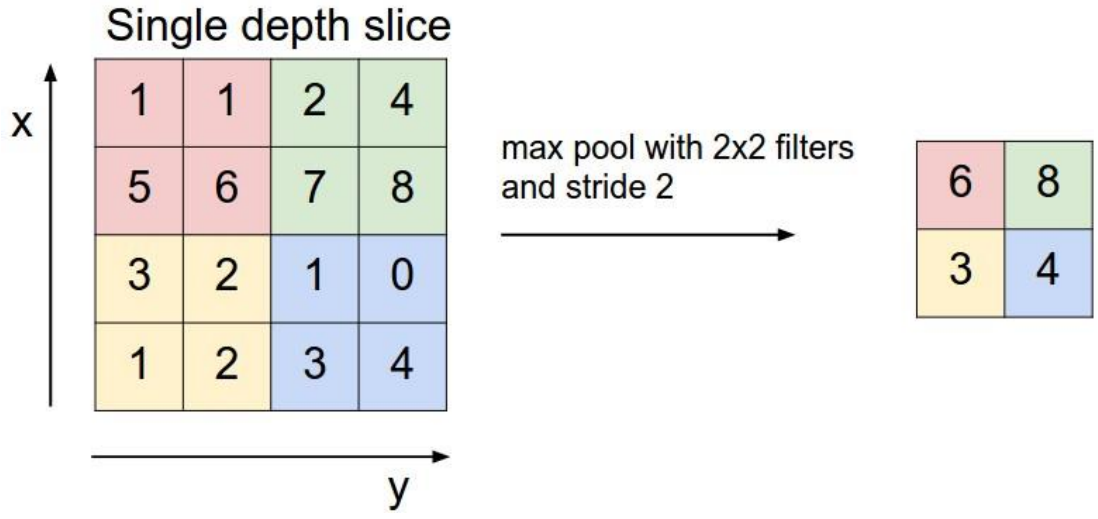


2. 参数共享(Parameter Sharing)
    1. 基于一个假设：图像不同区域的共同特征的过滤器（filter）是相同的，不会随着位置而改变特征的表示。
    2. 权值共享使得我们能更有效的进行特征抽取，因为它极大的减少了需要学习的自由变量的个数。极大的减少了计算量，便于实现大规模深层神经网络。

池化层(Pooling Layer)

- 池化（pool）即下采样（downsamples），目的是为了减少特征图的 size。池化操作对每个深度切片独立，规模一般为 2 * 2，相对于卷积层进行卷积运算。

- 池化操作将保持特征图(feature map)深度大小不变，即不改变输出特征的数目。



2.3 卷积神经网络算法实现

对于这个项目采用 Google 的 tensorflow + Keras 框架来构建模型。Keras 是一个崇尚极简、高度模块化的神经网络库，并可以同时运行在 TensorFlow 和 Theano 上。可以通过编程的方式调试模型结构和各种超参数，简化了编程的复杂度。

图像处理需要进行大量的 tensor 矩阵运算操作，并且还要调用 GPU 进行并行计算。

算法主要步骤：

1. 图像数据预处理，切分训练集，验证集，转换成 Keras Input。
2. 模型计算图构建，考虑模型的层，损失函数，优化器。
3. 训练模型，采用 mini-batch 梯度下降算法更新模型权值。
4. 根据 val-loss, accuray 等指标，调整模型参数，优化模型。
5. 对测试集图片预测概率，输出结果。

2.4 基准测试

本项目是 Kagge 竞赛题，采用 Kaggle 的 log loss 作为评分标准。由于硬件计算资源的限制，CPU Intel I7-7700, RAM 16G, GPU Nvidia GTX-1060-6G.
目标 log loss < 0.06

3. 方法和具体实现

深度神经网络，逐层提取图像特征，损失函数 binary_crossentropy，最后一层用 sigmoid 实现 2 分类。需要很深的网络，且图像的 size 较大（通常 229 x 229 x 3 或者 224 x 224 x 3 等）。权值参数较多，计算量很大。
而 ImageNet 图像分类的算法很成熟，采用预训练模型进行迁移学习 效率很高。

3.1 模型选择

模型信息

模型	大小	Top1 准确率	Top5 准确率	参数数目	深度
Xception	88MB	0.790	0.945	22,910,480	126

模型	大小	Top1 准确率	Top5 准确率	参数数目	深度
VGG16	528MB	0.715	0.901	138,357,544	23
VGG19	549MB	0.727	0.910	143,667,240	26
ResNet50	99MB	0.759	0.929	25,636,712	168
InceptionV3	92MB	0.788	0.944	23,851,784	159
IncetionResNetV2	215MB	0.804	0.953	55,873,736	572
MobileNet	17MB	0.665	0.871	4,253,864	88

VGG16 和 VGG19 是比较早期的模型，准确率相对较低，且参数数目多。
所以考虑 InceptionV3， Xception， ResNet50 这 3 个模型 进行迁移学习。

3.2 数据预处理

先下载数据文件 train.zip, test.zip，解压缩并存放到本地文件夹。

```
# extract train image to folder 'train' and 'test'
train_path = 'train'
test_path = 'test'
# extract train
if not isdir(train_path):
    zipFile = zipfile.ZipFile('train.zip')
    zipFile.extractall()
    zipFile.close()

# move cat's image to folder 'train/cat/', dog's image to folder 'train/dog'
if not isdir('train/cat'):
    os.mkdir('train/cat')
if not isdir('train/dog'):
    os.mkdir('train/dog')

for filename in os.listdir('train'):
    match_obj1 = re.search(r'cat.[0-9]*.jpg',filename)
    if match_obj1:
        shutil.move('train/'+filename, 'train/cat/')

    match_obj2 = re.search(r'dog.[0-9]*.jpg',filename)
    if match_obj2:
        shutil.move('train/'+filename, 'train/dog/'+filename)

# extract test image to folder 'test'
if not isdir('test'):
    zipFile = zipfile.ZipFile('test.zip')
    zipFile.extractall()
    zipFile.close()
    os.mkdir('test/test')
    for file in os.listdir('test'):
        if not os.path.isdir(file):
            shutil.move('test/'+file,'test/test/')
```

1. 训练集 25000 张图片，一半是猫，一半是狗，平衡样本。
2. 要切分成 train, validation 2 个部分。通常把 80% 样本作为 train, 20% 作为 validation. 切分的时候要注意 train, validation 也要保持猫和狗的样本平衡，以免最终学习到的特征偏向于狗或者猫。
3. 要把样本随机打乱，相当于洗牌。排除样本顺序对训练学习产生影响，提高模型的泛化能力。

```
X_catfile = ['train/cat/cat.%d.jpg' % i for i in range(12500)]
X_dogfile = ['train/dog/dog.%d.jpg' % i for i in range(12500)]
# print(X_catfile[:10])
# set the validation split = 0.2
# then split train_set into X_train,X_val, y_train, y_val
splitpoint = int(12500 * 0.8)
# print(splitpoint)
X_catfile_train,X_catfile_val = X_catfile[:splitpoint],X_catfile[splitpoint:]
X_dogfile_train,X_dogfile_val = X_dogfile[:splitpoint],X_dogfile[splitpoint:]

X_train = np.array(X_catfile_train + X_dogfile_train)
y_train = np.array([0 for i in range(splitpoint)] + [1 for i in range(splitpoint)])
X_val = np.array(X_catfile_val + X_dogfile_val)
y_val = np.array([0 for i in range(splitpoint,12500)] + [1 for i in range(splitpoint,12500)])

train_idx = np.arange(20000)
val_idx = np.arange(5000)
random.seed(2018)
random.shuffle(train_idx)
# print(train_idx)
random.seed(2018)
random.shuffle(val_idx)
X_train,y_train = X_train[train_idx],y_train[train_idx]
X_val,y_val = X_val[val_idx],y_val[val_idx]
#print(X_train[:10])
#print(y_train[:10])
#print(y_train.sum())    #一半是猫，一半是狗，total sum = 10,000
```

InceptionV3，Xception 模型的默认输入图片 size 是 299x299，而 ResNet50 模型的默认输入图片 size 是 224 x 224。在读入图片时，3 个模型的处理方式是不同的。所以要分别生成 2 个 shape 的 Input tensor。
另外考虑到 GPU 显存 total 只有 6GB，没办法一次载入所有 train, validation 图片，要分 batch 传入图片数据。

3.3 迁移学习模型构建

1. 导入 InceptionV3，Xception，ResNet50，去掉 top layer 作为 base model。且冻结预训练模型的权值。
2. 为了提高预测的精确度降低损失，融合 3 个模型，采用 keras.layers.concatenate 层。
3. 防止过拟合，通常是增加 Dropout。实际调参时发现 loss 值收敛速度很慢，且在 batch_size 小的情况下比如 25，会造成 val_loss 震荡。改用 batch_normalization，不但能抑制过拟合，还加快收敛。

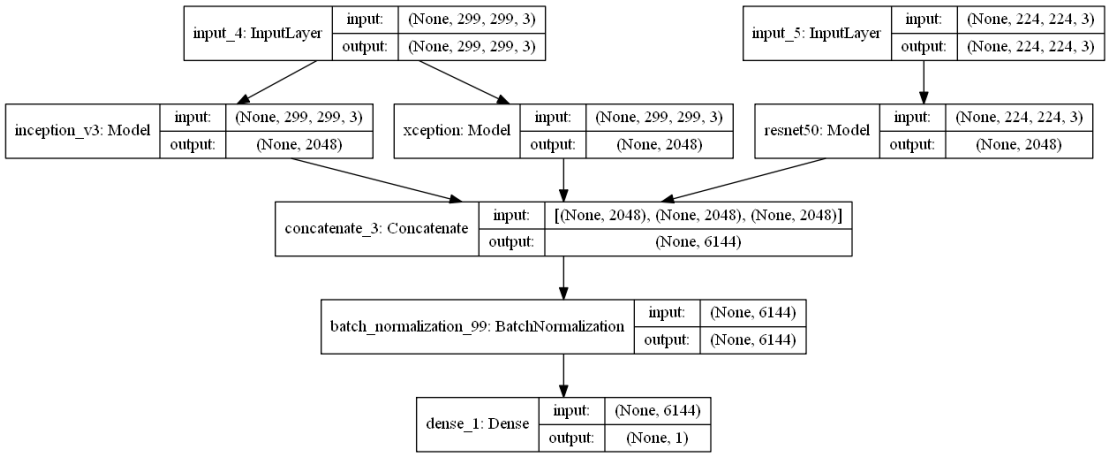
,

```
#from IPython.display import SVG
#from keras.utils.vis_utils import model_to_dot
#from keras.utils import plot_model

inception_V3 = InceptionV3(include_top = False, weights = 'imagenet', input_shape = (299, 299,3),pooling='avg')
for layer in inception_V3.layers:
    layer.trainable = False

xception = Xception(include_top= False, weights='imagenet',input_shape=(299,299,3),pooling='avg')
for layer in xception.layers:
    layer.trainable = False

resnet50 = ResNet50(include_top= False, weights='imagenet',input_shape=(224,224,3),pooling='avg')
for layer in resnet50.layers:
    layer.trainable = False

input1 = Input((299,299,3))
input2 = Input((224,224,3))

x1 = inception_V3(input1)
x2 = xception(input1)
x3 = resnet50(input2)

x = keras.layers.concatenate([x1,x2,x3])
x = BatchNormalization()(x)
# x = Dropout(0.5)(x)
x = Dense(1,activation = 'sigmoid')(x)
tune_model = Model(inputs=[input1,input2],outputs=x)
optimizer_adadelta = keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=1e-10)
tune_model.compile(loss = 'binary_crossentropy', optimizer =optimizer_adadelta, metrics =['accuracy'])
#plot_model(tune_model, to_file='model_add_BN.png',show_shapes = True)
#SVG(model_to_dot(tune_model).create(prog='dot', format='svg'))
tune_model.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_4 (InputLayer)         (None, 299, 299, 3)       0
_________________________________________________________________________________________________
input_5 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________________________________________
inception_v3 (Model)         (None, 2048)              21802784    input_4[0][0]
_________________________________________________________________________________________________
xception (Model)             (None, 2048)              20861480    input_4[0][0]
_________________________________________________________________________________________________
resnet50 (Model)             (None, 2048)              23587712    input_5[0][0]
_________________________________________________________________________________________________
concatenate_3 (Concatenate)  (None, 6144)              0           inception_v3[1][0]
                                                                   xception[1][0]
                                                                   resnet50[1][0]
_________________________________________________________________________________________________
batch_normalization_99 (BatchNo (None, 6144)           24576       concatenate_3[0][0]
_________________________________________________________________________________________________
dense_1 (Dense)              (None, 1)                 6145        batch_normalization_99[0][0]
=================================================================================================
Total params: 66,282,697
Trainable params: 18,433
Non-trainable params: 66,264,264
```

迁移学习，base model 的参数冻结，不参与训练。只需要训练 18,433 个权值参数。

2 个 input, 1 个 output 端到端的融合模结构图如下



为了进一步提高模型的泛化能力，防止几轮(epoch)就过拟合，val-loss 无法继续下降。对训练集的图像进行数据增强，使用下面这些数据增强方法

1. 随机剪裁
2. 随机旋转
3. 随机剪切
4. 水平翻转
5. 垂直翻转

```python
def random_crop(img, ratio=1.0):
    #随机剪裁, Ratio是裁剪后与裁剪前的面积比
    if random.random() < 1:
        h,w = img.shape[:2]
        #长宽比随机扰动变量 hw_delta
        hw_delta = np.random.uniform(-0.05,0.05)
        hw_mult = 1 + hw_delta

        w_crop = int(round(w*np.sqrt(ratio*hw_mult)))

        if w_crop > w:
            w_crop = w

        h_crop = int(round(h*np.sqrt(ratio/hw_mult)))
        if h_crop > h:
            h_crop = h

        x0 = np.random.randint(0,w-w_crop+1)
        y0 = np.random.randint(0,h-h_crop+1)

        img = img[y0:y0+h_crop,x0:x0+w_crop]

    return img

def random_rotate(img, angle=0.0):
    if random.random() < 1:
        h,w = img.shape[:2]
        # angle = np.random.uniform(-angle,angle)
        # angle cycle is 360
        angle %= 360
        # 用openCV 内置函数计算仿射矩阵
        M_rotate = cv2.getRotationMatrix2D((w/2,h/2),angle,1)
        # 得到旋转后的图片
        img = cv2.warpAffine(img, M_rotate, (w, h))
    return img

def random_shear(img,angle=0.0):
    if random.random() < 1:
        h,w = img.shape[:2]
        #产生随机剪切角度值
        # angle = np.random.uniform(-angle,angle)
        # x轴剪切变换，角度值=angle
        theta = angle * np.pi / 180
        M_shear = np.array([[1,np.tan(theta), 0],[0,1,0]],dtype = np.float32)
        img = cv2.warpAffine(img, M_shear, (w, h))
    return img

def random_h_flip(img, horizontal_flip=False):
    if horizontal_flip == True:
        if random.random() < 1:
            img = cv2.flip(img, 1)
    return img

def ramdom_v_flip(img, vertical_flip=False):
    if vertical_flip == True:
        if random.random() < 1:
            img = cv2.flip(img,0)
    return img
```

```
def train_generator(X=None, y=None, batch_size=32,ratio=None,rotate=None,shear=None,
                    horizontal_flip=False,vertical_flip=False):
    params = {}
    if ratio is not None:
        params['ratio'] = ratio
    if rotate is not None:
        params['rotate'] = rotate
    if shear is not None:
        params['shear'] = shear
    if horizontal_flip:
        params['horizontal_flip'] = horizontal_flip
    if vertical_flip:
        params['vertical_flip'] = vertical_flip

    if params:
        func = []
        for param in params.keys():
            func.append(param)
    func = random.choice(func)
    # print(func,params[func])

    function_map = { 'ratio':random_crop,
                     'rotate':random_rotate,
                     'shear':random_shear,
                     'horizontal_flip':random_h_flip,
                     'vertical_flip':ramdom_v_flip
                    }

    datalen = len(X)
    counter = datalen//batch_size

    X_299 = np.zeros((datalen,299,299,3),dtype=np.uint8)
    X_224 = np.zeros((datalen,224,224,3),dtype=np.uint8)

    # print("\nwaiting for generating train samples......\n")
    for i in range(datalen):
        X_img = cv2.imread(X[i])
        X_img = function_map[func](X_img,params[func])
        X_299[i] = cv2.resize(X_img,(299,299))
        X_224[i] = cv2.resize(X_img,(224,224))
    # print("generated %d train samples\n"%datalen)

    random.seed(2018)
    index = np.arange(counter)
    random.shuffle(index)

    while (True):
        for i in range(counter):
            yield [(X_299[index[i] * batch_size:(index[i] + 1) * batch_size]-127.5)/127.5,
                   (X_224[index[i] * batch_size:(index[i] + 1) * batch_size]-127.5)/127.5], \
                     y[index[i] * batch_size:(index[i] + 1) * batch_size]
```

另外，优化器的学习率等超参数也对模型的训练过程有影响。Adadelta 的学习率是自适应的，特点：训练初中期，加速很快 训练后期，反复在局部最小值附近抖动。Val_loss 到 0.1 附近震荡，无法再下降。尝试把默认参数 epsilon=1e-06，调整到 1e-10。初期收敛的慢一些，后期 loss 可以持续下降到 0.06 以下。

为了防止训练 epoch 过多，后面过拟合 val_loss 不下降甚至反而增加。Kears 很方便增加 Early stop 回调函数，监测 val_loss 如果连续 3 次不下降就停止训练。

4. 结果

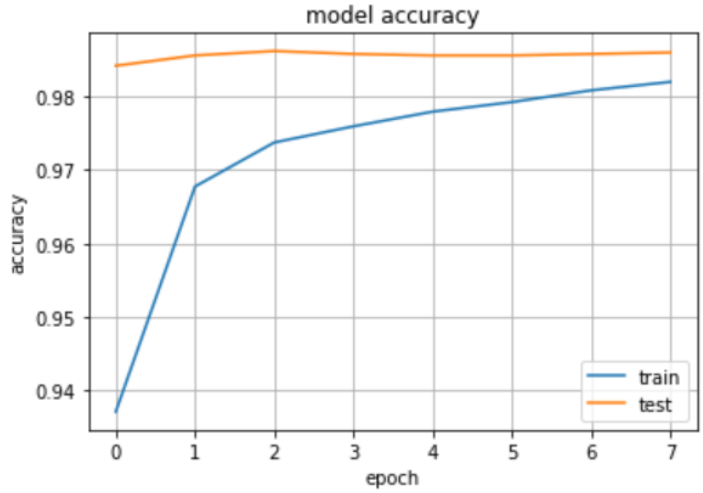
经过数据增强，调参数最终训练结果如下

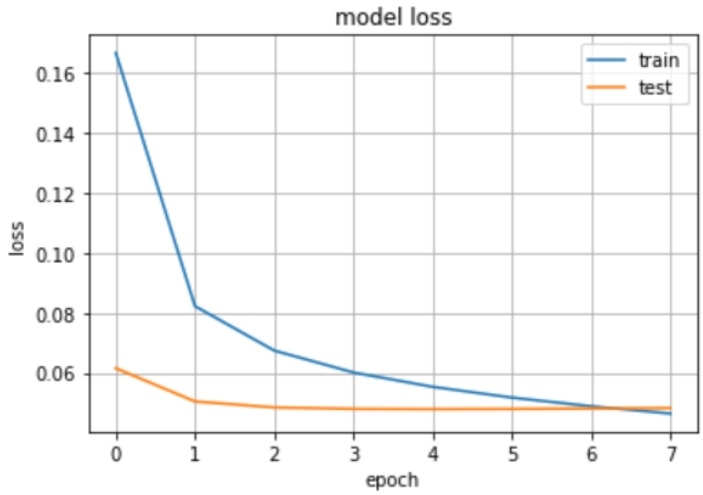
```
# in case, computer crash in a specified epoch, save checkpointer of model
# restore a previous model to continue traing from the specified epoch.
# tune_model = load_model('./model/tune_model.h5')

checkpointer = ModelCheckpoint('./model/tune_model.h5', monitor='val_loss', verbose=0, save_best_only=True,
                               save_weights_only=False, mode='auto', period=1)
earlystop = EarlyStopping(monitor='val_loss', patience=3, verbose=0, mode='auto')

logger = CSVLogger('./model/log.csv', separator=',', append=True)

# if restore previous model, may train from specified epoch by parameters "initial_epoch= "
model_history = tune_model.fit_generator(train_generator(X_train, y_train, batch_size,0.9,20,20,True,True),
                                         steps_per_epoch=len(y_train)//batch_size, epochs=epochs,
                                         validation_data = val_generator(X_val, y_val, batch_size),
                                         validation_steps=len(y_val)//batch_size,
                                         callbacks = [earlystop,checkpointer,logger],
                                         verbose=1)
```

```
Epoch 1/40
400/400 [==============================] - 1169s 3s/step - loss: 0.1668 - acc: 0.9371 - val_loss: 0.0616 - val_acc: 0.9842
Epoch 2/40
400/400 [==============================] - 772s 2s/step - loss: 0.0823 - acc: 0.9678 - val_loss: 0.0505 - val_acc: 0.9856
Epoch 3/40
400/400 [==============================] - 769s 2s/step - loss: 0.0675 - acc: 0.9738 - val_loss: 0.0485 - val_acc: 0.9862
Epoch 4/40
400/400 [==============================] - 761s 2s/step - loss: 0.0601 - acc: 0.9760 - val_loss: 0.0480 - val_acc: 0.9858
Epoch 5/40
400/400 [==============================] - 763s 2s/step - loss: 0.0554 - acc: 0.9780 - val_loss: 0.0479 - val_acc: 0.9856
Epoch 6/40
400/400 [==============================] - 764s 2s/step - loss: 0.0518 - acc: 0.9793 - val_loss: 0.0480 - val_acc: 0.9856
Epoch 7/40
400/400 [==============================] - 763s 2s/step - loss: 0.0489 - acc: 0.9809 - val_loss: 0.0481 - val_acc: 0.9858
Epoch 8/40
400/400 [==============================] - 773s 2s/step - loss: 0.0464 - acc: 0.9820 - val_loss: 0.0483 - val_acc: 0.9860
```

绘制训练曲线

```
# summarize history for accuracy
plt.plot(model_history.history['acc'])
plt.plot(model_history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid()
plt.legend(['train', 'test'], loc='lower right')
plt.show()
# summarize history for loss
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.grid()
plt.legend(['train', 'test'], loc='upper right')
plt.show()
```





训练曲线 train loss 不断下降，validation loss 也随之下降。到 Epoch 8 基本稳定，val_loss 到 0.0483 无法再下降，停止训练。

最后用 fine-tune 的模型，对测试集图片进行预测

```
import pandas as pd

test_datagen = ImageDataGenerator()
test_299 = test_datagen.flow_from_directory(
        'test/',
        target_size=(299, 299),
        batch_size=25,
        shuffle = False,
        class_mode=None)
test_224 = test_datagen.flow_from_directory(
        'test/',
        target_size=(224, 224),
        batch_size=25,
        shuffle = False,
        class_mode=None)

def test_generator():
    while (True):
        X_299 = test_299.next()
        X_224 = test_224.next()
        yield [(X_299-127.5)/127.5,(X_224-127.5)/127.5]

predictions = tune_model.predict_generator(test_generator(),steps = 500, verbose =1)
predictions = predictions.clip(min=0.005,max=0.995)

output = pd.read_csv('sample_submission.csv')
for i, filename in enumerate(test_299.filenames):
    # print(filename)
    index = int(filename[filename.rfind('\\') + 1 : filename.rfind('.')])
    output.loc[index-1 ,'label'] = predictions[i]
```

```
output.to_csv('predictions_Final.csv',index=None)
```

```
Found 12500 images belonging to 1 classes.
Found 12500 images belonging to 1 classes.
500/500 [==============================] - 424s 848ms/step
```

测试集 预测结果如下表

	A	B	C	D	E	F	G
1	id	label					
2	1	0.991448					
3	2	0.995					
4	3	0.995					
5	4	0.995					
6	5	0.005					
7	6	0.005					
8	7	0.005					
9	8	0.005					
10	9	0.005					
11	10	0.005					
12	11	0.005					
13	12	0.909621					
14	13	0.005					
15	14	0.005					
16	15	0.005					
17	16	0.005					
18	17	0.987284					
19	18	0.995					
20	19	0.005					
21	20	0.005					

predictions_Final

提交 Kaggle，得到 log loss 0.05517

5. 总结

通过这个猫狗图片识别的项目，从问题描述，数据分析，构建模型，结果预测系统的学习了运用 卷积神经网络模型 解决实际问题。在有限的计算资源条件下，取得不错的效果 Log Loss 0.05517。且模型稳定，参数调整合理，排除了过拟合，局部极值震荡等问题。数据增强要考虑是否会破坏图片的特征，比如随机剪裁可能会把关键特征的部分裁掉，影响特征学习的结果。样品随机打乱是必要的，否则样本的分布特性会影响模型学习结果。

分析最终预测结果 其中预测错误的图片或者概率偏差较大的图片是有问题的。即使是人来判断也会难以判断。

图片 571，预测概率 0.343， 应该是学习到猫的特征

图片 915，预测概率 0.335， 即使是人也很难判断

图片 1055，预测概率 0.426， 应该是学习到玩具的特征。

图片 1267，预测概率 0.399，这个图片本来就有问题

图片 2636，预测概率 0.067， 完全看不起图片中是猫还是狗

值得思考，后续有待改进的地方

大量图片 Numpy 数组 对内存的占用较高，考虑到工程项目的应用环境，要在不显著降低预测精度 accuray 的前提下，改进算法减少内存占用。

1. 训练数据类型可以考虑从 float64 减低精度到 float32,甚至 float16。
2. 训练数据 预处理后 以 HDF5 文件格式存到硬盘，在 generator 中分批读入。

参考文献

François Chollet
Xception: Deep Learning with Depthwise Separable Convolutions
arXiv:1610.02357

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Deep Residual Learning for Image Recognition
arXiv:1512.03385

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna
Rethinking the Inception Architecture for Computer Vision
arXiv:1512.00567

Udacity 机器学习（进阶）
https://cn.udacity.com/

CS231n: Convolutional Neural Networks for Visual Recognition
http://vision.stanford.edu/teaching/cs231n/syllabus.html

tensorflow 中文社区
http://www.tensorfly.cn/tfdoc/api_docs/python/nn.html

Keras:基于 Python 的深度学习库
http://keras-cn.readthedocs.io/en/latest/