机器学习纳米学位

毕业项目 优达学城

2018-7-10

I. 问题的定义

项目概述

本项目来源于kaggle比赛，目标是训练机器学习分类模型，对输入图片判断类别是猫还是狗， 本项目使用的主要技术的是卷积神经网络（Convolution Neural Network，CNN),卷积神经网络具有强大的自动提取图片特征的能力，广泛应用于图像分类，目标检测，语义分割等各个视觉领域。在图像分类问题上，人们已经设计出很多强大的CNN模型(VGG, Xception, ResNet等)，取得了远胜于传统SVM方法的准确率。

问题陈述

本项目面临的主要问题有

1. 模型大小与计算资源的矛盾： 通常来说，卷积神经网络的层数越深，可训练参数数量越多，模型的拟合能力越强。但是更大的模型意味着更大的显存消耗和更长的训练时间。本项目kaggle上原始数据集的图片尺寸比较大,一般在300x300pixels以上。所以层数特别深的模型难以从头开始训练。
2. 模型泛化能力的问题： 本项目提供的训练集一共有25000图片，两个类别各占一半，还要预留20%的图片作为验证集，实际用于训练的图片只有20000张。测试集图片有12500张。我们希望获得在测试集和训练集上都表现优良的分类模型，如果用拟合能力很强的大模型直接训练会出现过拟合倾向。为了保证模型的泛化能力同时不增加太多训练时间，需要合理采用正则化方法抑制模型的过拟合。

评价指标

本项目是一个二分类问题，用对数损失(Log loss,亦称交叉熵)作评价指标，交叉熵是分类问题常用的损失函数，它反映的是两个概率分布之间的差异，可以方便的进行求导，计算公式为.

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

其中y_l表示一张图片的类别被预测为狗的概率，y_i是1如果此图片的真实类别是狗，否则是0

为了对模型做正则化，抑制过拟合，给损失函数加上一项最后全连接层权值的L2范数。

$$R(w) = \lambda \sum_{i=1}^{n} |w_i^2|$$

其中 λ是L2损失函数的系数，λ越大表示正则化程度越大

完整的损失函数为

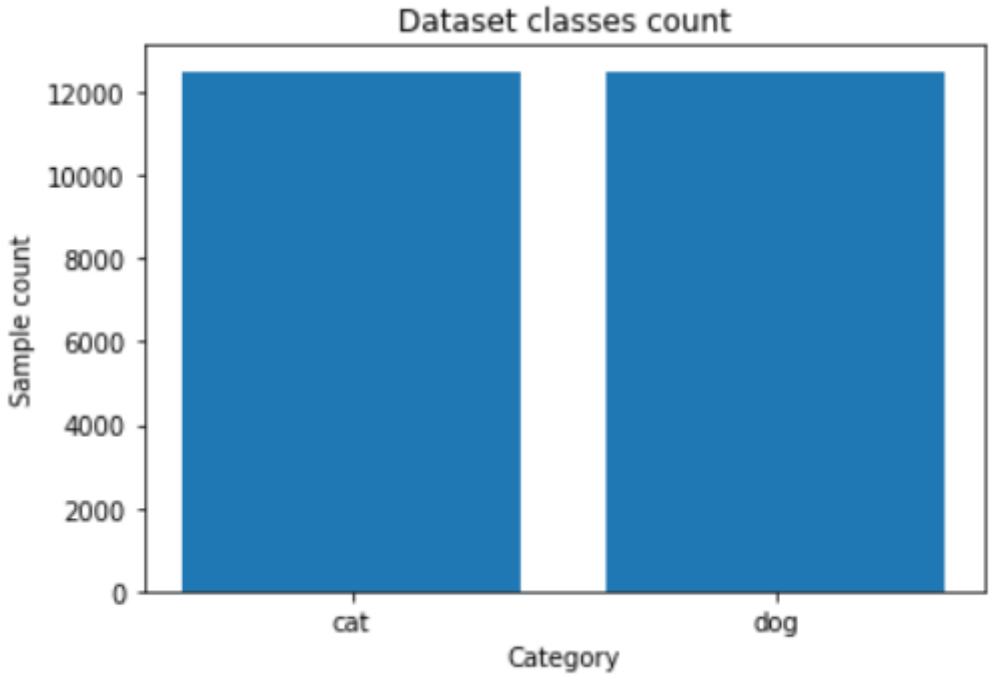
$$Loss = Logloss + R(w)$$

II. 分析

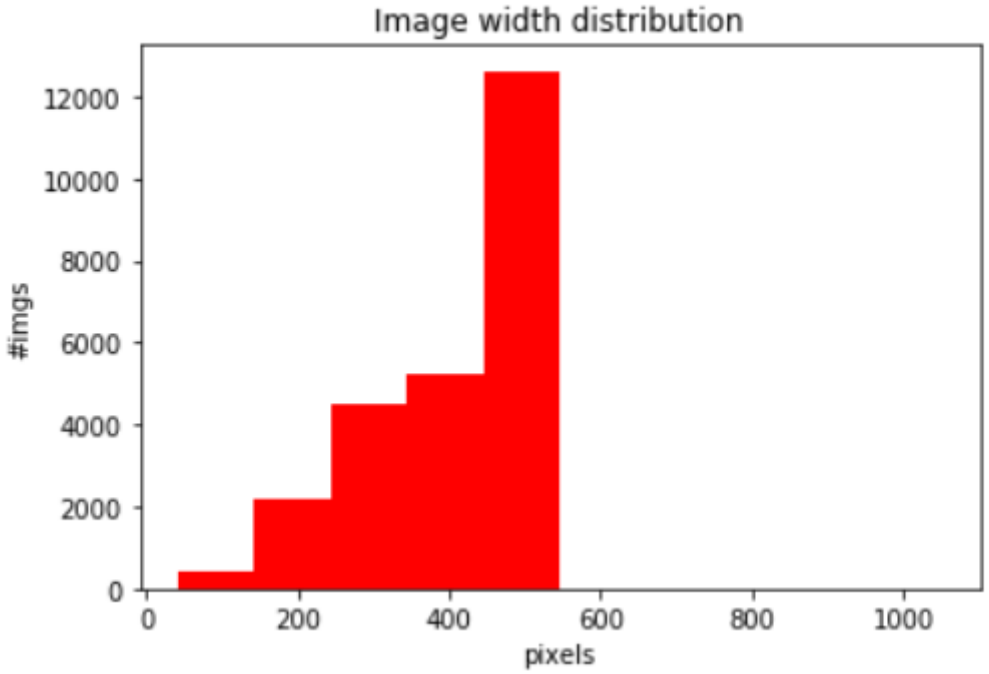
数据的探索和可视化

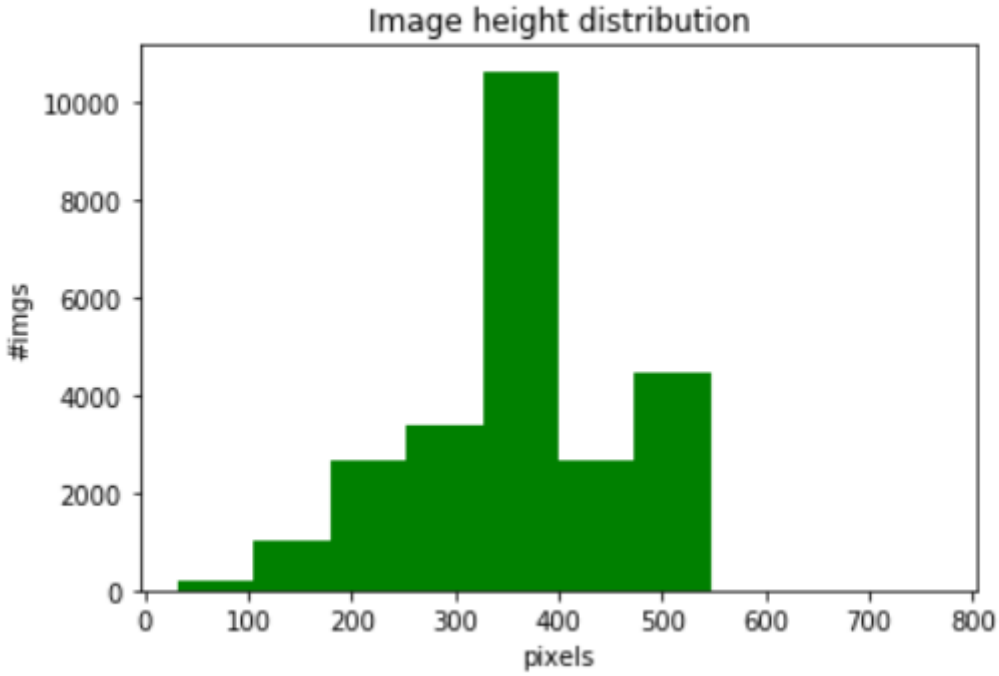
本项目使用的数据集包含训练集train文件夹和测试集test文件夹，train包含25000个图片，每张图片用 “类别.编号.jpg”命名，test包含12500图片，只用“编号.jpg”命名。为了适应keras图片生成器的读取模式，首先需要根据类别把train分为两个文件夹，每个文件夹用类别命名。 下面是训练集图片的统计信息

训练图片类别分布



训练图片的宽度和高度分布





训练图片的随机采样示例

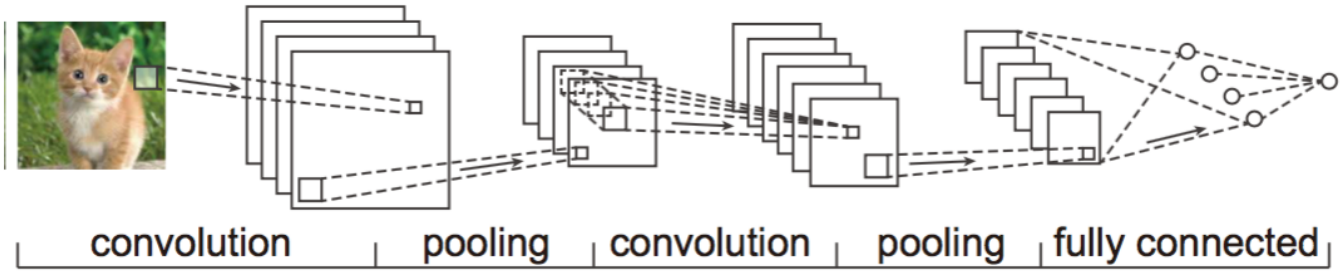
从图中可以初步看出

1. 训练图片类别经过挑选，两个各占50%，没有类别不平衡问题
2. 训练图片尺寸不一致，宽度和高度大部分分布在200-500像素之间，有个别异常大的图片尺寸600像素以上
3. 训练图片视角，光线和目标位置相差很大，有的图片是猫狗的正脸，有的包含全身，部分图片甚至有多个目标出现

算法和技术

1.卷积神经网络

本项目用到的核心算法是卷积神经网络。卷积神经网络（Convolutional Neural Network, CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。 卷积神经网络由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。



如图所示，最基本的卷积神经网络结构由卷积层，池化层和全连接层组成。 卷积层（Convolutional layer），卷积神经网络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最佳化得到的。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，更多层的网路能从低级特征中迭代提取更复杂的特征。

池化（Pooling）是卷积神经网络中另一个重要的概念，它实际上是一种形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。直觉上，这种机制能够有效地原因在于，在发现一个特征之后，它的精确位置远不及它和其他特征的相对位置的关系重要。池化层会不断地减小数据的空间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了过拟合。通常来说，CNN的卷积层之间都会周期性地插入池化层。

全连接层(Fully connected layer)是卷积神经网络中负责最后分类任务的神经网络。它通常位于网络的后端，负责把前面卷积层提取出的特征向量映射到损失函数的输入中。全连接层也是通过反向求导传播梯度的方式训练的，因此可以很方便的与卷积层联合在一起训练

2. 迁移学习

本项目用到的另一个关键技术是迁移学习(Transfer learning).所谓迁移学习，就是能让现有的模型算法稍加调整即可应用于一个新的领域和功能的一项技术。前文中已经提到，本数据集的图片分辨率较高，如果从头开始搭建深度卷积网络训练可能要花费很长的时间才能收敛到比较理想的效果。好在人们已经在ImageNet数据集上训练了大量表现良好的模型，而ImageNet数据集的1000个类别中包含大量动物类，猫和狗都是其中之一。因此使用ImageNet上预训练过的网络就是顺其自然的选择了。

预训练网络(Pretrained model)的前几层卷积通常用来提取图片的高层次特征，如最基本的边缘，纹理等。更深的卷积层用来提取更细致的特征，网络的最后几层用来做分类，把提取到的特征映射到概率空间。常用的预训练模型有VGG16, InceptionV3, Xception, ResNet等。下图是各个预训练模型在ImageNet数据集上的表现和模型大小

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

可以看出，通常模型的准确率高复杂度也越高，其中InceptionV3模型有非常不错的准确率并且模型体积也较小，是用于迁移学习的极佳选择。

InceptionV3是基于GoogLeNet提出的改进网络，整个模型有46层，11个Inception模块，Inception模块的作用是代替人工确定卷积层中的过滤器类型或者确定是否需要创建卷积层和池化层，即：不需要人为决定使用哪个过滤器，是否需要池化层等，由网络自行决定这些参数，可以给网络添加所有可能值，将输出连接起来，网络自己学习它需要什么样的参数。InceptionV3的并联方式在不大量增加参数的情况下保持了性能，同时用Batch Normalzation层缓解了梯度消失的问题。

基准指标

本项目采用kaggle上测试集的最终提交结果作为评判标准，打分标准是对数损失函数。项目采用的基准指标是kaggle排行榜的前10%，即交叉熵低于0.061

III. 方法

数据预处理

InceptionV3的预训练模型要求输入图片像素归一化到[-1,1]之间，在ImageNet上图片大小统一缩放到299x299，为了采用迁移学习，本项目的图片也需要做相同的预处理，keras中有方便的预处理函数可以调用。图片经过InceptionV3前向传播后需要经过GlobalAveragePooling层池化，得到长度为2048的特征向量再送入分类器网络进行分类。除此之外，图片不需要做其他处理，一个泛化能力足够强的深度学习模型应该能适应各种视角，光照等条件带来的不确定性。为了节省训练时间，本项目也没有使用数据增强。

执行过程

本项目使用keras 2.0.6版本的深度学习工具。为了保证代码的可读性和可复用性，定义了一个类InceptionV3专门用来做迁移学习，构造函数中可指定weight_decay,drop_rate等超参数控制模型的正则化程度，可以选择预训练的网络权值并指定冻结多少层做fine-tune。

```
class InceptionV3:

    weights_path = "saved_weights"
    models_path = "saved_models"
    input_shape = (299,299,3)

    def __init__(self, num_classes=2,pretrained=True,
                 drop_rate=0.25,weight_decay=0):
        self.num_classes = num_classes
        self.drop_rate = drop_rate
        self.weight_decay = weight_decay
        self.cls_model = None
        self.pretrained = pretrained

        input_tensor = Input(InceptionV3.input_shape)
        x = Lambda(inception_v3.preprocess_input)(input_tensor)
        if self.pretrained:
            self.backend = inception_v3.InceptionV3(weights='imagenet',
                                                    include_top=False, input_tensor=x)
        else:
            self.backend = inception_v3.InceptionV3(weights=None,
                                                   include_top=False,input_tensor=x)

        if not os.path.exists(InceptionV3.weights_path):
            os.mkdir(InceptionV3.weights_path)
        if not os.path.exists(InceptionV3.models_path):
            os.mkdir(InceptionV3.models_path)
```

```
def build(self, fine_tune=True,layer_to_freeze=281, lr=0.001):
    self.base_model = Model(input=self.backend.input, output=GlobalAveragePooling2D()(self.backend.output))

    self.cls_model = Sequential()
    self.cls_model.add(Dropout(0.5,input_shape=self.base_model.output.shape.as_list()[1:]))
    self.cls_model.add(Dense(self.num_classes, activation='softmax',kernel_regularizer=regularizers.l2(self.weight_decay)))

    if fine_tune:
        for i,layer in enumerate(self.base_model.layers):
            if i<layer_to_freeze:
                layer.trainable = False

    self.cls_model_compile(lr)
```

一开始采用冻结模型前面所有卷积层的方法，只用小学习率训练最后的分类层，但是试验发现把整个模型载入显存再冻结权值训练的方法，训练时间依然很长，主要原因是keras的生成器需要不断去磁盘中读取图片，经过预处理层，卷积层，池化层的前向传播才能到达分类层。反复循环这个过程会造成进程的IO等待时间变长。显卡和CPU使用率来回跳动。

```
Found 20000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
Epoch 1/6
1250/1250 [==============================] - 477s - loss: 0.0787 - acc: 0.9720 - val_loss: 0.0417 - val_acc: 0.9838
Epoch 2/6
1250/1250 [==============================] - 477s - loss: 0.0387 - acc: 0.9860 - val_loss: 0.0379 - val_acc: 0.9843
Epoch 3/6
1250/1250 [==============================] - 477s - loss: 0.0217 - acc: 0.9932 - val_loss: 0.0428 - val_acc: 0.9870
Epoch 4/6
1250/1250 [==============================] - 478s - loss: 0.0150 - acc: 0.9951 - val_loss: 0.0455 - val_acc: 0.9860
Epoch 5/6
1250/1250 [==============================] - 478s - loss: 0.0132 - acc: 0.9954 - val_loss: 0.0502 - val_acc: 0.9850
Epoch 6/6
1250/1250 [==============================] - 477s - loss: 0.0076 - acc: 0.9977 - val_loss: 0.0607 - val_acc: 0.9839
```

后来经过试验，决定把整个训练集和测试集的图片全部分批传入预训练网络，把输入的特征向量保存到磁盘，再单独构建一个全连接的分类模型训练。在训练的时候相当于把整个训练集的特征向量全部导入了内存，这样省去了IO等待时间，同时占用的显存大大减少，在普通显卡上几秒钟中就能训练出一个分类模型。

```
@staticmethod
def save_weights(train_dir, test_dir, batch_size=32):
    gen = ImageDataGenerator()
    train_generator = gen.flow_from_directory(train_dir, (InceptionV3.input_shape[0],InceptionV3.input_shape[1]),
                                              shuffle=False, batch_size=batch_size)

    test_generator = gen.flow_from_directory(test_dir, (InceptionV3.input_shape[0],InceptionV3.input_shape[1]),
                                             shuffle=False, batch_size=batch_size,classes=None,class_mode=None)
    X_train =self.base_model.predict_generator(train_generator,
                                               steps=(train_generator.samples)//batch_size,verbose=1)
    X_test = self.base_model.predict_generator(test_generator,
                                               steps=(test_generator.samples)//batch_size,verbose=1)
    with h5py.File(os.path.join(InceptionV3.weights_path,"inception_v3.hdf5")) as f:
        f.create_dataset("train", data=X_train)
        f.create_dataset("test", data=X_test)
        f.create_dataset("label", data=train_generator.classes)
```

模型调参

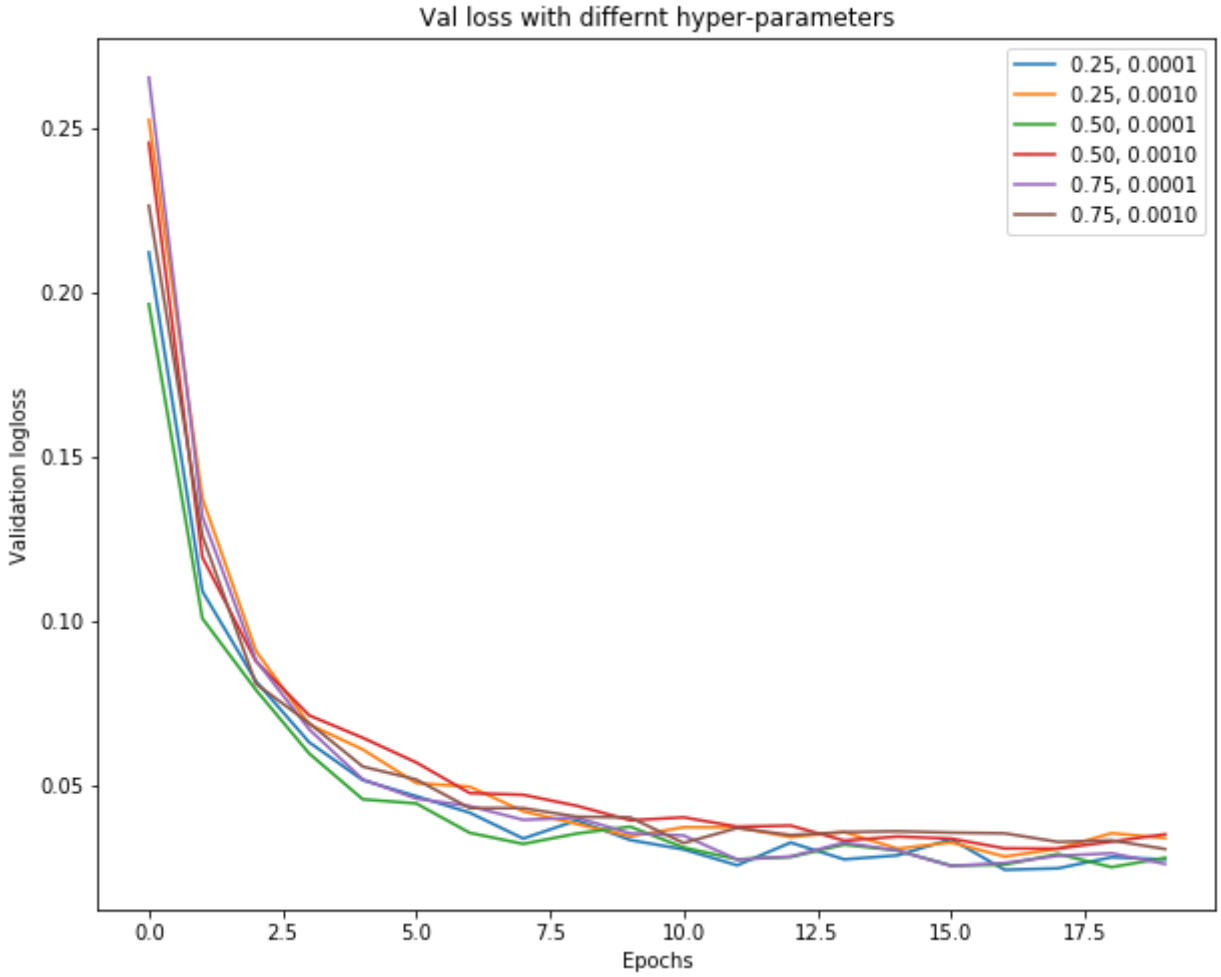
前文已经提到，训练集的规模较小，为了抑制过拟合趋势，需要采用一定的正则化手段，本项目主要采用了Dropout和L2损失函数的方法做正则化。其中Dropout的概率和L2损失函数的系数是模型的两个重要超参数。为了找到最优的参数组合，选择在验证集上用网格搜索法就行调参，得到6个临时模型。

```
drop_rate_set = [0.25, 0.5, 0.75]
weight_decay_set = [1e-4, 1e-3]
val_loss_records = []
val_acc_records = []
train_loss_records = []

i = 1
for drop_rate in drop_rate_set:
    for weight_decay in weight_decay_set:
        model = InceptionV3(num_classes=2, pretrained=True,
                 drop_rate=drop_rate,weight_decay=weight_decay)
        model.build(fine_tune=True, layer_to_freeze=313, lr=1e-4)
        history = model.fit(X_train, y_train, batch_size=64, epochs=20, validation_split=0.2)
        model.save_model("model_"+str(i)+".hdf5")
        i += 1
        val_loss_records.append(history.history['val_loss'])
        val_acc_records.append(history.history['val_acc'])
        train_loss_records.append(history.history['loss'])

        del model
        gc.collect()
```

下面是不同参数的模型在验证集上的表现对比

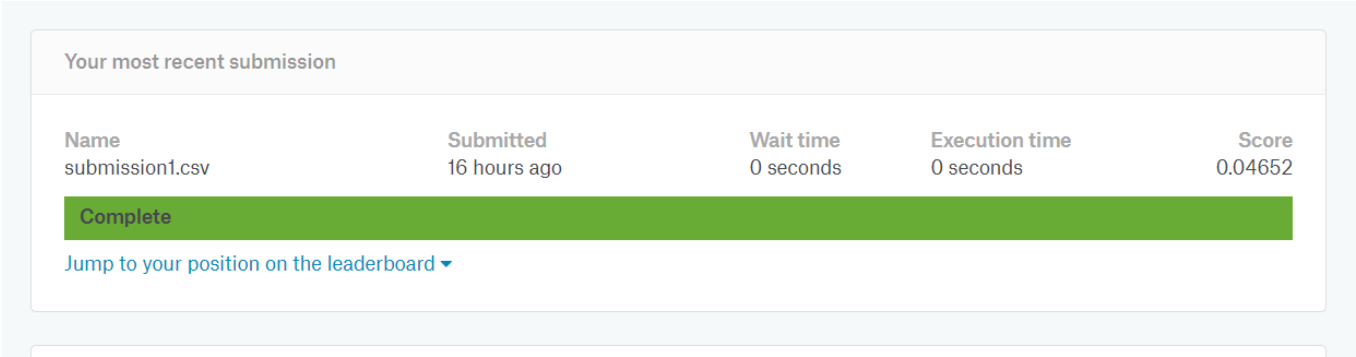


可以看到，不同参数下模型的表现整体上很接近，相比来说，验证集误差最低的是drop_out_rate=0.25, weight_deacy=0.0001的模型，在20个迭代次数里Logloss一直平稳下降，没有出现较大的过拟合，这也是最终选择的最优模型

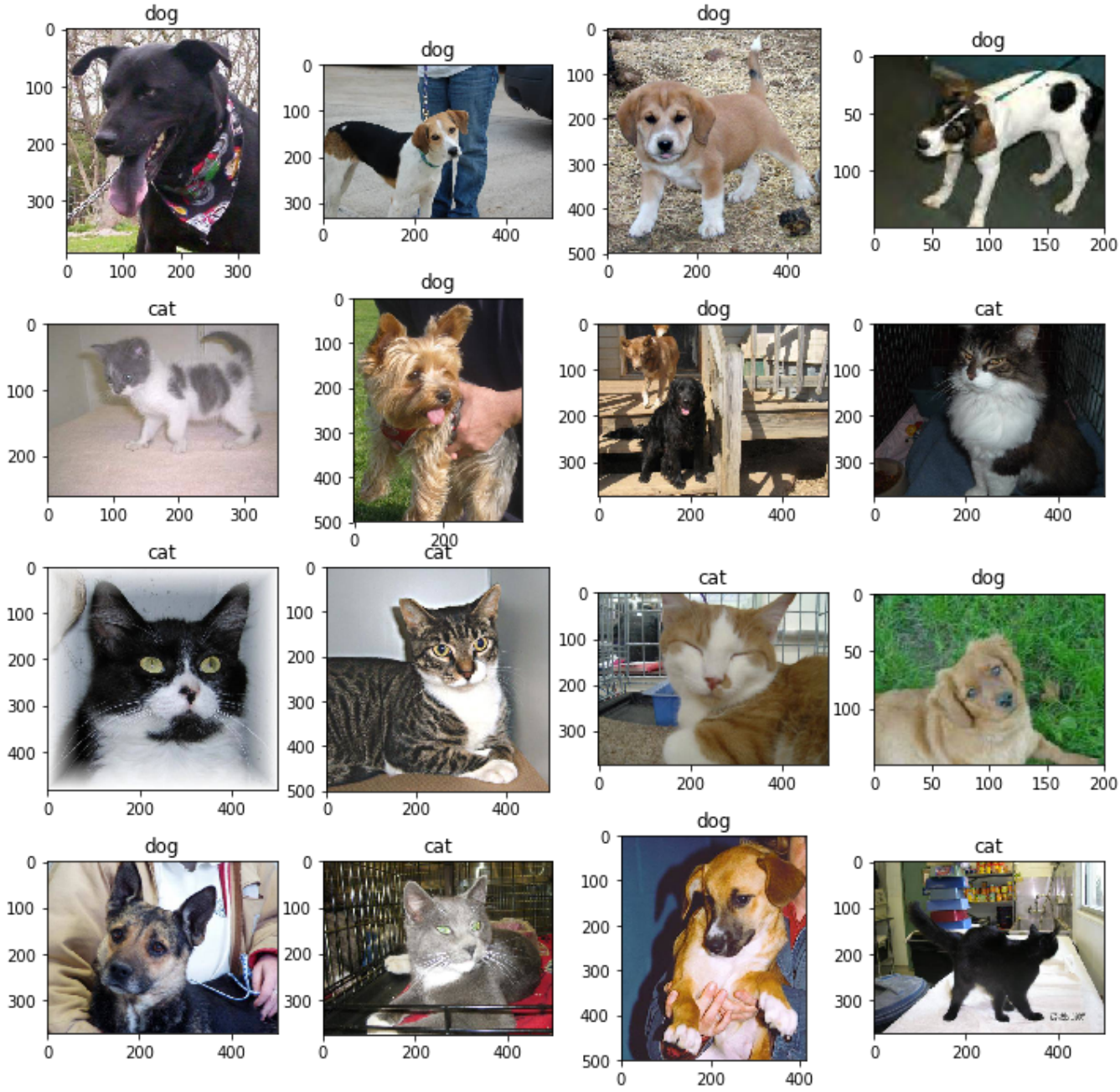
结果

模型验证与可视化

让调参得到的最优模型在测试集上进行预测，得到的结果提交到kaggle平台上，最终结果Logloss=0.04652,排名第36名，进入前10%。



下面是模型对测试集图片做出的预测



可以看出，模型在绝大多数图片上都做出了正确的预测，确实具有很强的分类能力，鲁棒性很高

结论

对项目的思考

本项目是经典的用卷积神经网络实现的图片分类任务。从头开始寻找可行的方案并最终实现它是充满挑战的。项目最大的困难在于一开始深度神经网络的训练时间很长，显存需求很大，无法用个人笔记本运行。一开始AWS EC2上计算没有找到合适的fine-tune方法，尝试冻结各种层都导致严重的过拟合，做了很多无效的计算。最终发现只要重新训练分类层就能取得很好的效果。而且训练速度很快，可以方便的调参。最终的结果总体上符合期望。

需要作出的改进

项目可以提高的地方还很多，比如尝试与ResNet, VGG等模型做融合，采用数据增强扩大训练集，进一步提高模型泛化能力等等