# 机器学习工程师纳米学位毕业项目
# 猫狗大战

唐沛文

2018 年 07 月 08

# 1.问题定义

## 1.1 项目概述

项目将使用深度学习算法训练一个图像识别模型，通过对大量猫和狗照片的学习，最终能够正确识别出一张图片中是一只猫还是一只狗。图像识别技术是信息时代的一门重要的技术，其产生目的是为了让计算机代替人类去处理大量的物理信息。随着计算机技术的发展，人类对图像识别技术的认识越来越深刻。图像识别技术的过程分为信息的获取、预处理、特征抽取和选择、分类器设计和分类决策。

在图像分类领域，传统机器学习去识别分类一个一个图像的标准流程是特征提取、特征筛选，最后将特征向量输入合适的分类器完成特征分类。直到 2012 年 Alex Krizhevsky 突破性的提出 AlexNet 的网络结构，借助深度学习的算法，将图像特征的提取、筛选和分类三个模块集成于一体，设计 5 层卷积层加 3 层全连接层的深度卷积神经网络结构，逐层对图像信息进行不同方向的挖掘提取，譬如浅层卷积通常获取的是图像边缘等通用特征，深层卷积获取的一般是特定数据集的特定分布特征。AlexNet 以 15.4%的创纪录低失误率夺得 2012 年 ILSVRC（ImageNet 大规模视觉识别挑战赛）的年度冠军，值得一提的是当年亚军得主的错误率为 26.2%。AlexNet 超越传统机器学习的完美一役被公认为是深度学习领域里程碑式的历史事件，一举吹响了深度学习在计算机领域爆炸发展的号角。

时间转眼来到了 2014 年，GoogleNet 横空出世，此时的深度学习，已经历 ZF-net，VGG-net 的进一步精炼，在网络的深度，卷积核的尺寸，反向传播中梯度消失问题等技术细节部分已有了详细的讨论，Google 在这些技术基础上引入了 Inception 单元，大破了传统深度神经网络各计算单元之间依次排列，即卷积层->激活层->池化层->下一卷积层的范式，将 ImageNet 分类错误率提高到了 6.7%的高水平。在网络越来越深，网络结构越来越复杂的趋势下，深度神经网络的训练越来越难，2015 年 Microsoft 大神何恺明（现就职于 Facebook AI Research）为了解决训练中准确率先饱和后降低的问题，将 residual learning 的概念引入深度学习领域，其核心思想是当神经网络在某一层达到饱和时，利用接下来的所有层去映射一个 f(x)=x 的函数，由于激活层中非线性部分的存在，这一目标几乎是不可能实现的。但 ResNet 中，将一部分卷积层短接，则当训练饱和时，接下来的所有层的目标变成了映射一个 f(x)=0 的函数，为了达到这一目标，只需要训练过程中，各训练变量值收敛至 0 即可。Resdiual learning 的出现，加深网络深度提高模型表现的前提下保证了网络训练的稳定性。2015 年，ResNet 也以 3.6%的超低错误率获得了 2015 年 ImageNet 挑战赛的冠军，这一技术也超越了人类的平均识别水平，意味着人工智能在人类舞台中崛起的开始。

本项目会使用卷积神经网络(Convolutional Neural Network)模型进行图像识别。CNN 最早被用于手写数字识别并一直保持了其在该问题的霸主地位。近年来卷积神经网络在多个方向持续发力，在语音识别、人脸识别、通用物体识别、运动分析、自然语言处理甚至脑电波分析方面均有突破。

## 1.2 问题陈述

项目属于图像识别领域的分类问题，通过对已标记数据进行监督学习，训练后的模型可以对给出图片进行分类。

项目选择的数据集是 kaggle 上的猫狗大战数据集，观察后发现数据集中存着一些明显的异常图片，理论上应该删除掉提高模型识别能力，但是由于测试集中也可能存在异常图片，如果将训练集中的异常数据去除，模型就学不会这些异常值，在最终提交时无法做出有利于分数的预测。因此在后续训练时，考虑对比去除异常图片和保留异常图片两种情况，选择更优的处理方式。

## 1.3 评价指标

本项目的最低要求是 kaggle Public Leaderboard 前 10%，同时评估指标采用 LogLoss 进行评估。LogLoss 计算公式如下：

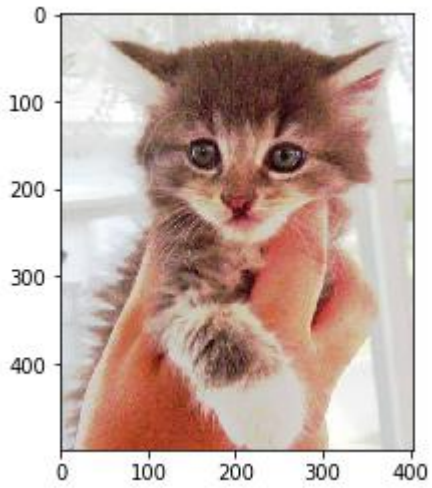
$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\hat{y}_i\right) + (1-y_i)\log\left(1-\hat{y}_i\right)\right]$$

其中 $y_i$ 表示真实情况，$\hat{y}_i$ 表示模型预测结果。模型表现越好，logloss 值越小。

# 2.分析

## 2.1 数据的探索

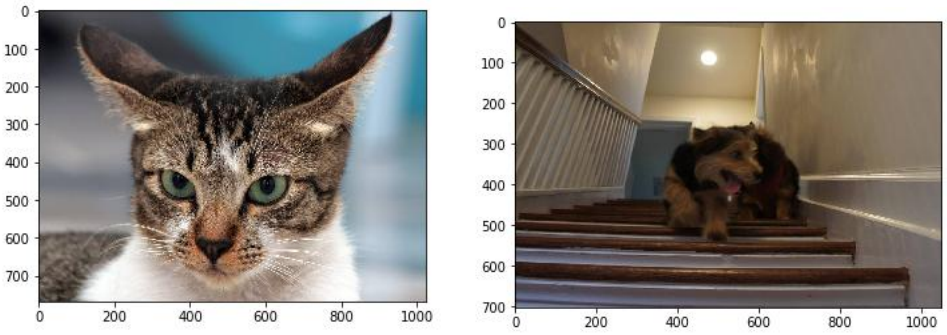
整个训练集集包含 25000 张训练图片，其中猫和狗各 12500 张，猫狗占比 1:1。随机挑选了几张数据集中的图片，如下：



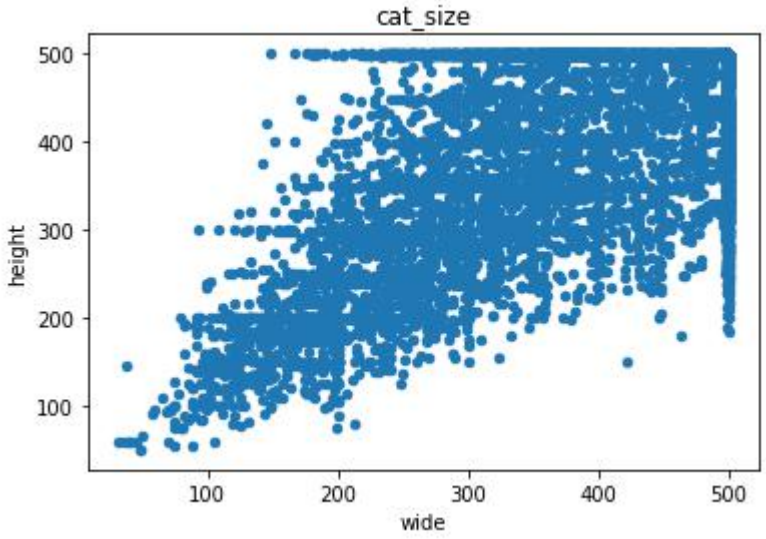


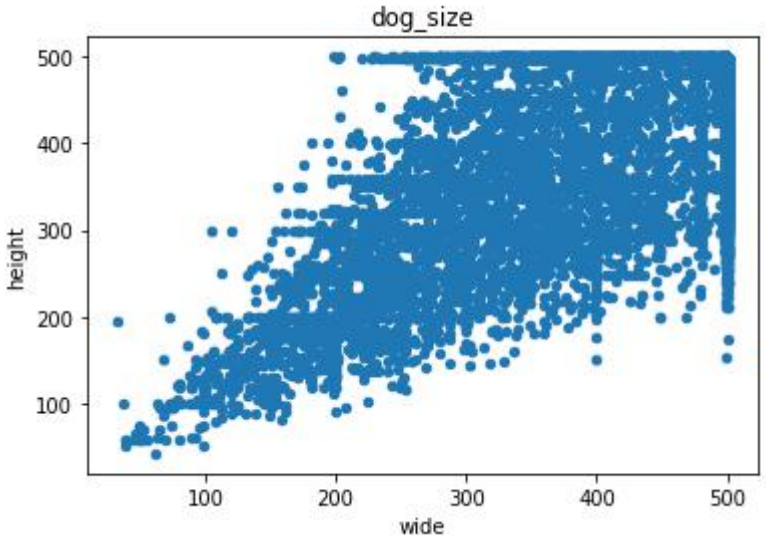
可以发现，图片的有以下特点：

1. 分辨率不同，因此在将图片读入后图片进行统一缩放到相同尺寸。分别对猫和狗训练集的图片尺寸进行统计，得到以下散点图：
   猫狗训练集中各存在一张图片尺寸明显大于其他图片为 768*1023 和 702* 1050。如下：



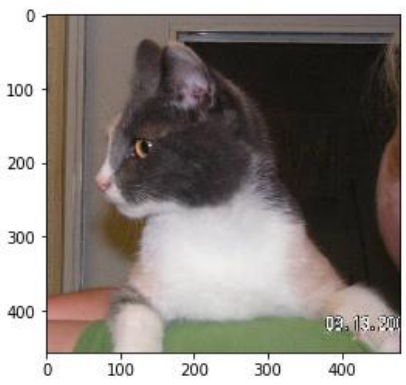
其他图片均在 500*500 以内，绘制 500*500 以内的图片大小散点图如下：



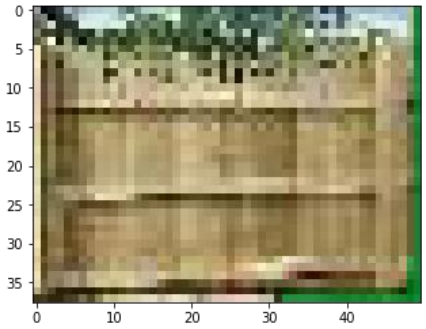



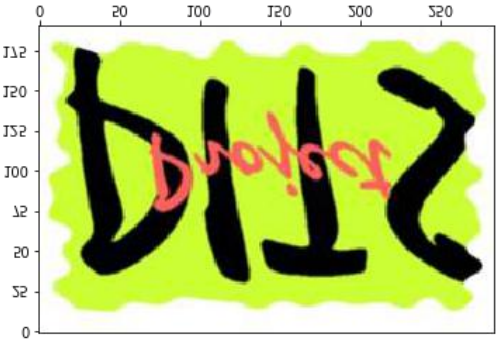

2. 有的图片背景会比较复杂，或者图片主体不是猫或者狗。如下图：

3. 主体不一定是正面照。如下图：



4. 存在异常图片，图片中不包含猫或者狗，如下图：





## 2.2 分类算法

### 2.2.1 传统机器学习算法

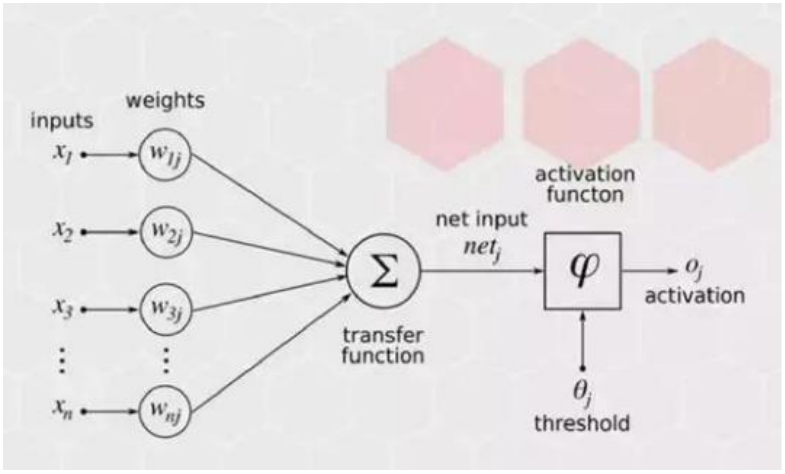
此项目解决的主要是图片的分类问题，用于分类的传统机器学习算法有很多，比如逻辑回归、支持向量机、决策树等。算法的思路都是根据输入的一张图片上每个像素组成的特征向量，通过大量图片的监督学习进行模型训练，最终可以对一张模型没有见过的图片进行分类。本项目仅识别猫和狗，因此属于二类分类问题。

对于传统的机器学习算法，由于图片的像素值很高，例如一张 244*244 RGB 三通道的图

片，像素值就有 178608，即程序输入特征向量是 178608 维。随着维度的增多，训练数据集大小需要指数级的上升，否则会出现维度灾难，导致过拟合的现象。给 20 万个维度的模型提供训练集几乎是不可能实现的。

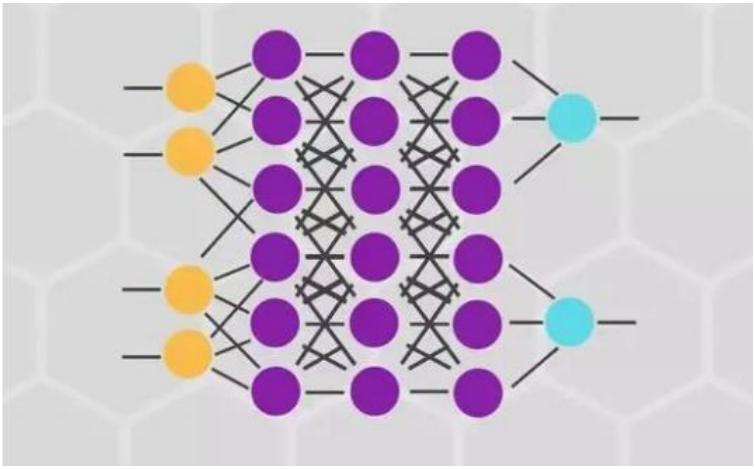
## 2.2.2 神经网络

人工神经网络(ANN)中最小也是最重要的单元叫神经元。与生物神经系统类似，这些神经元也互相连接并具有强大的处理能力。一般而言，ANNs 试图复现真实大脑的行为和过程，这也是为什么他们的结构是基于生物学观察而建模的。




每个神经元都有输入连接和输出连接。这些连接模拟了大脑中突触的行为。与大脑中突触传递信号的方式相同——信号从一个神经元传递到另一个神经元，这些连接也在人造神经元之间传递信息。每一个连接都有权重，这意味着发送到每个连接的值要乘以这个因子。再次强调，这种模式是从大脑突触得到的启发，权重实际上模拟了生物神经元之间传递的神经递质的数量。所以，如果某个连接重要，那么它将具有比那些不重要的连接更大的权重值。

由于可能有许多值进入一个神经元，每个神经元便有一个所谓的输入函数。通常，连接的输入值都会被加权求和。然后该值被传递给激活函数，激活函数的作用是计算出是否将一些信号发送到该神经元的输出。

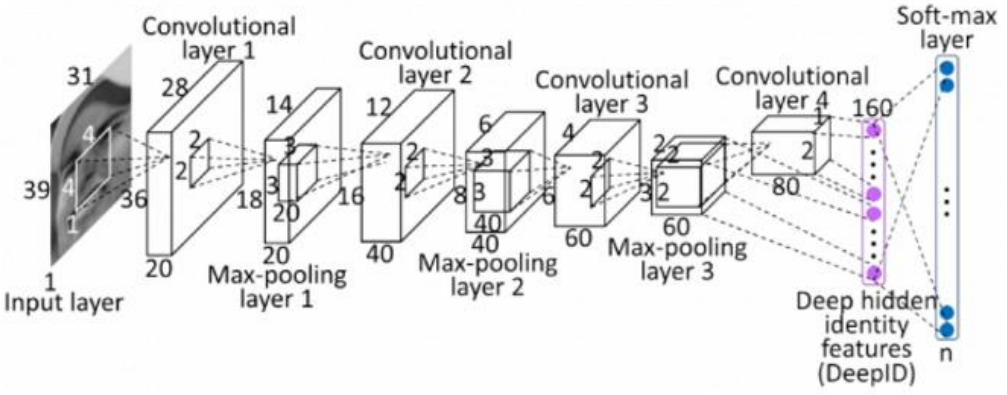


## 2.2.3 卷积神经网络

卷积神经网络（Convolutional Neural Networks-简称 CNN）是近年发展起来，并引起广泛重视的一种高效识别方法。20 世纪 60 年代，Hubel 和 Wiesel 在研究猫脑皮层中用于局部敏感和方向选择的神经元时发现其独特的网络结构可以有效地降低反馈神经网络的复杂性，继而提出了卷积神经网络。现在，CNN 已经成为众多科学领域的研究热点之一，特别是在模式分类领域，由于该网络避免了对图像的复杂前期预处理，可以直接输入原始图像，因而得到了更为广泛的应用。 K.Fukushima 在 1980 年提出的新识别机是卷积神经网络的第一个实现网络。随后，更多的科研工作者对该网络进行了改进。其中，具有代表性的研究成果是 Alexander 和 Taylor 提出的“改进认知机”，该方法综合了各种改进方法的优点并避免了耗时的误差反向传播。

一般地，CNN 的基本结构包括两层，其一为特征提取层，每个神经元的输入与前一层的局部接受域相连，并提取该局部的特征。一旦该局部特征被提取后，它与其它特征间的位置关系也随之确定下来；其二是特征映射层，网络的每个计算层由多个特征映射组成，每个特征映射是一个平面，平面上所有神经元的权值相等。特征映射结构采用影响函数核小的 sigmoid 函数作为卷积网络的激活函数，使得特征映射具有位移不变性。此外，由于一个映射面上的神经元共享权值，因而减少了网络自由参数的个数。卷积神经网络中的每一个卷积层都紧跟着一个用来求局部平均与二次提取的计算层，这种特有的两次特征提取结构减小了特征分辨率。

CNN 主要用来识别位移、缩放及其他形式扭曲不变性的二维图形。由于 CNN 的特征检测层通过训练数据进行学习，所以在使用 CNN 时，避免了显示的特征抽取，而隐式地从训练数据中进行学习；再者由于同一特征映射面上的神经元权值相同，所以网络可以并行学习，这也是卷积网络相对于神经元彼此相连网络的一大优势。卷积神经网络以其局部权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性，其布局更接近于实际的生物神经网络，权值共享降低了网络的复杂性，特别是多维输入向量的图像可以直接输入网络这一特点避免了特征提取和分类过程中数据重建的复杂度。



有以下几点需要说明：

1. 卷积的作用在于，通过卷积可以压缩空间的维度同时不断增加深度，使深度信息可以表示复杂的语意，也就是可以使用卷积提取显著特征。并且可以有效减少特征维数，显著降低计算量。

2. CNN 中 Convolution 层，即为卷积层。如果卷积层的输入层（input layer）维度值为 W， 滤波器（filter）的维度值为 F (height * width * depth)， stride 的数值为 S， padding 的数值为 P， 下一层的维度值可用如下公式表示: (W−F+2P)/S+1。

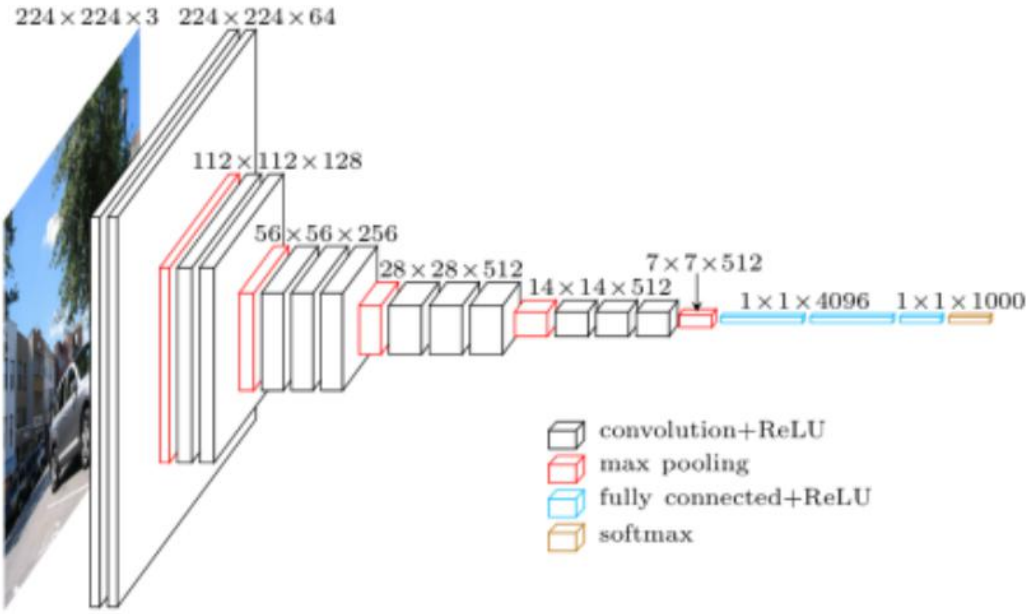
3. CNN 中的 GlobalAveragePooling2D 层，即为全局平均池化层。是对输入数据在每个维度上取全局平均值作为输出。最终起到降低输出大小和避免过拟合的作用。

4. CNN 中常用激活函数 ReLU(Rectified Linear Unit)，公式为 $f(x)=\max(0,x)$ 。相比传统的神经网络激活函数如 Logistic sigmoid，ReLU 具备更有效率的梯度下降以及反向传播，避免了梯度爆炸和梯度消失问题。同时简化计算过程，没有了其他复杂激活函数中诸如指数函数的影响。

5. Dropout 是一项正则化技术，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。dropout 是 CNN 中防止过拟合提高效果的一个大杀器。它强迫一个神经单元，和随机挑选出来的其他神经单元共同工作，达到好的效果。消除减弱了神经元节点间的联合适应性，增强了泛化能力。

6. 优化算法 SGD，随机梯度下降。相对于批量梯度下降法，将进行 N 次迭代直到目标函数收敛，每次迭代都将对 m 个样本进行计算，计算量大。SGD 每次迭代仅对一个样本计算梯度，直到收敛。通常来说， SGD 中参数更新的计算不使用单个样本，而是少量或说一小批样本。这样做的原因有两个：第一，这降低了参数更新过程中的方差，使收敛的过程更稳定，第二，可以很好地将成本和梯度的计算矢量化，于是在计算中可以利用到深度优化的矩阵运算。

7. 项目使用 VGG16 网络作为基准模型，VGG 结构如下图所示：

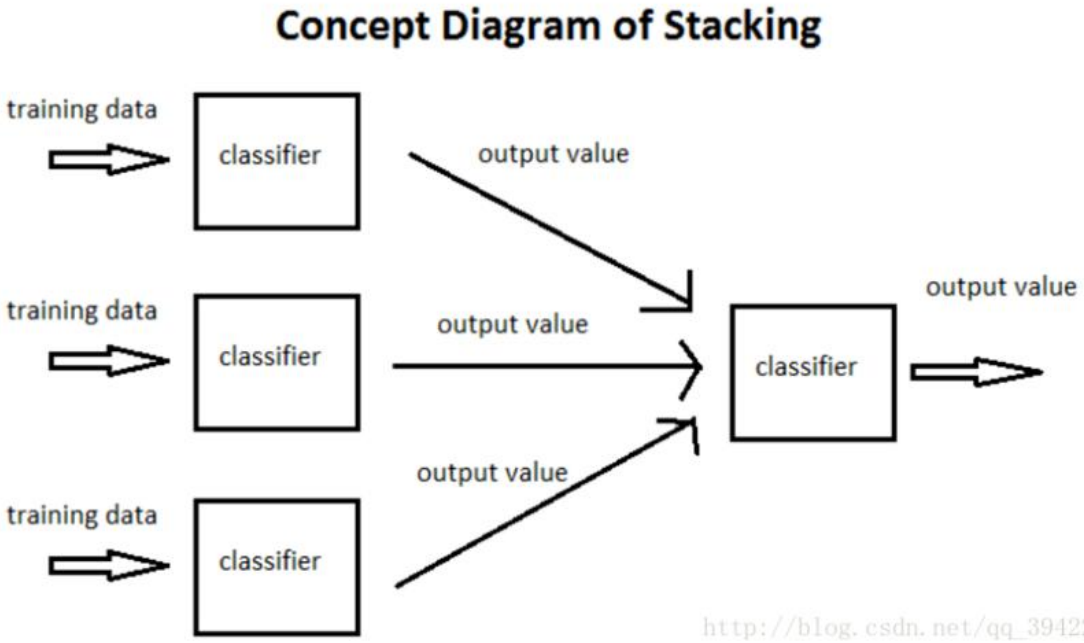



由于 VGG16 只有卷积池化堆叠，结构简单并且提取的特征很有用，非常适合作为基础模型。

8. 迁移学习指在大多数情况下，面对某一领域的某一特定问题，不可能找到足够充分

的训练数据，这是业内一个普遍存在的事实。但是，得益于一种技术的帮助，从其他数据源训练得到的模型，经过一定的修改和完善，就可以在类似的领域得到复用，这一点大大缓解了数据源不足引起的问题，而这一关键技术就是迁移学习。迁移学习的基本思路是利用预训练模型，即已经通过现成的数据集训练好的模型（这里预训练的数据集可以对应完全不同的待解问题，例如具有相同的输入，不同的输出）。开发者需要在预训练模型中找到能够输出可复用特征（feature）的层次（layer），然后利用该层次的输出作为输入特征来训练那些需要参数较少的规模更小的神经网络。

9. 模型融合指通常可以在各种不同的机器学习任务中使结果获得提升。顾名思义，模型融合就是综合考虑不同模型的情况，并将它们的结果融合到一起。事实上，往往将几个效果还可以的模型的预测结果融合，取得的效果要比多个精细调优的模型分别预测的效果好。具体 Stacking 思想如下图所示：




使用多个基分类器，再在基分类器预测的基础上进行分类。融合可以通过汇总各个基模型提取的特征，在汇总特征的基础上进行分类预测，达到提升性能的目的。

## 2.2.4 技术

项目使用 Keras 作为计算框架，是一个以 CNTK、TensorFlow 或 Theano 为计算后台的深度学习建模环境。Keras 在实际应用中有如下显著优点：

1. Keras 在设计时以人为本，强调快速建模，用户能快速地将所需模型的结构映射到 Keras 代码中，尽可能减少编写代码的工作量。特别对于成熟的模型类型，加快开发速度。
2. 支持显著常见的结构，如卷积神经网络、时间递归神经网络等。
3. 高度模块化，用户几乎能够任意组合各个模块来构建所需的模型。
4. 能在 CPU 和 GPU 之间无缝切换。

本项目使用 TensorFlow 作为 keras 的后台深度学习框架。

## 2.3 基准指标

将使用预训练的 VGG16[1]的迁移学习作为基准模型，锁定卷积层，在卷积层后接入 dropout 层和全连接层。最终提交的模型需要超过 VGG16 基准模型。

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, label, shuffle=True, test_size=0.2, random_state=42)

#VGG16 base
x = Input(shape=(244, 244, 3))
x = Lambda(vgg16.preprocess_input)(x)

base_model = VGG16(include_top=False, input_tensor=x, weights='imagenet', pooling='avg')
for layer in base_model.layers:
    layer.trainable = False

# 自定义分类层
y = Dropout(0.2)(base_model.output)
y = Dense(1, activation='sigmoid', kernel_initializer='he_normal')(y)

model = Model(inputs=base_model.input, outputs=y, name='Transfer_Learning')
sgd = SGD(lr=1e-3, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

```
log_name = '/dogcat-EP{epoch:02d}-LOSS{val_loss:.4f}.h5'
log_dir = datetime.now().strftime('transfer_model_%Y%m%d_%H%M')
if not os.path.exists(log_dir):
    os.mkdir(log_dir)

es = EarlyStopping(monitor='val_loss', patience=20)
mc = ModelCheckpoint(log_dir + log_name, monitor='val_loss', save_best_only=True)
tb = TensorBoard(log_dir=log_dir)

model.fit(x=X_train, y=y_train, batch_size=16, epochs=10, validation_data=(X_val, y_val), callbacks=[es, mc, tb])
```

VGG16 迁移学习得到的测试集 logloss 为 0.08173。同时由于项目要求至少达到 public leaderboard top 10%，因此达到 public leaderboard top 10%也是项目的基准之一。

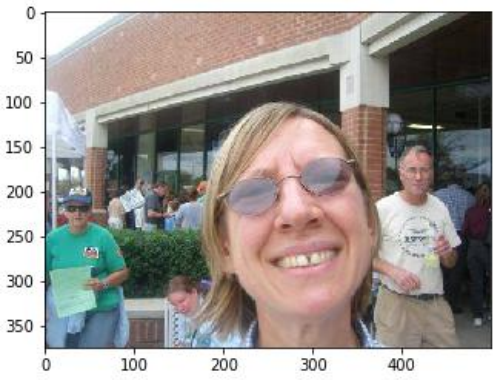
# 3.具体方法

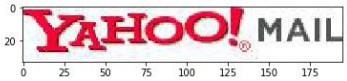
## 3.1 数据预处理

1. 图像处理：项目使用 vgg16 作为基准模型，vgg16 的默认输入大小是 224*224，因此在基准模型训练是将图片大小统一调整为 224*224。对于后面的融合训练，Xception[6]、Inception_v3[3]、Inception_resnet_v2[4]模型的默认输入大小是 299*299，因此将图片尺寸统一调整为 299*299。
2. 图片读入时，由于数据量很大，造成按 299*299 读入内存时，内存不足。因此在做模型融合时，将训练集分为了 3 个子集，每次仅读入一个子集，将子集通过预训练模型加工结果写入到文件后再处理下一个子集。确保不会出现内存溢出。
3. 对应训练数据，根据文件名称，解析出文件对应的是狗还是猫，并打上标签供后续训练使用。使用 sklearn 的 train_test_split 函数将读入的训练集分为训练集和交叉验证集。其中训练集 20000 张，交叉验证集 5000 张，测试集 12500 张。
4. 归一化处理：对于不同的 keras 预训练模型，有不同的图像归一化过程。Xception、Inception_v3、Inception_resnet_v2 是将所有像素值归一化到[-1,1]。其他模型是像素减去 ImageNet 所有图像在 RGB 通道上的像素均值 [103.939, 116.779, 123.68]进行归一化。我们在模型训练前调用 kaggle 中各个模型对应的 preprocess_input 函数，使输入数据能够满足训练模型的要求。
5. 异常数据检查：通过使用 Xception[6]、ResNet50[2]、InceptionResNetV2[4]三个预训练模型，对训练集进行预测，取 top80 的结果。与 imagnet 分类中的狗和猫集合进行

对比，筛选出预测结果中未出现在狗和猫集合的图片。共 50 张，经过分析基本都是异常图片。观察有发现有以下几种情况：

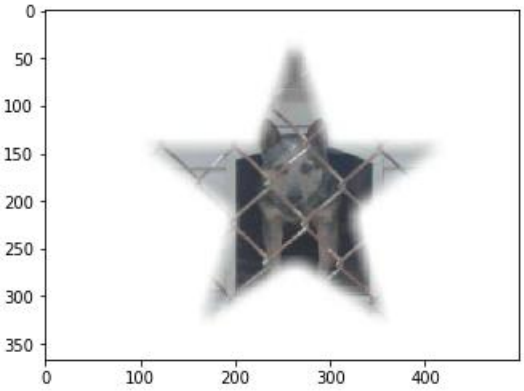
1） 图片中不包含猫或者狗：



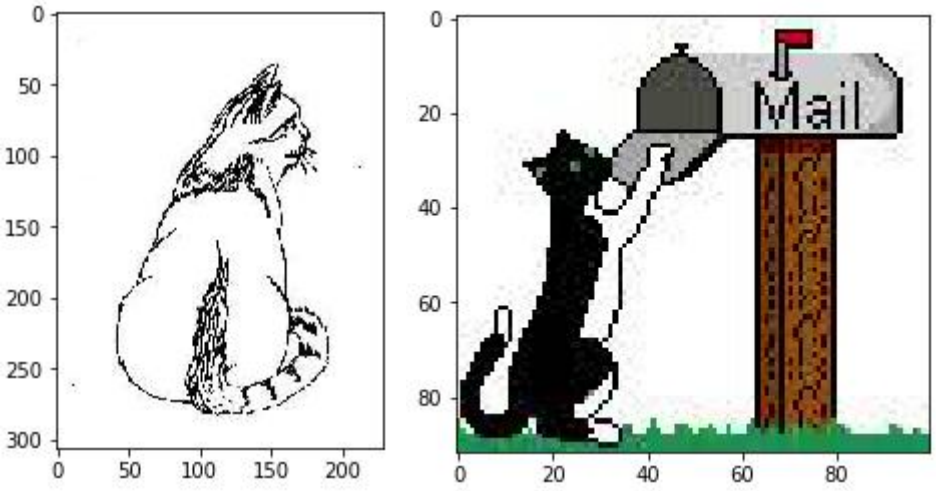



2） 背面过于复杂或者主体不明显：

3） 识别目标在图像中占比过小：



4） 图片不是真实的照片，为卡通图片：




6. 对于监测出的异常图片，考虑到异常图片在测试集中也同样存在，如果将训练集中的异常数据去除，模型就学不会这些异常值，在最终提交时无法做出有利于分数的预测。因此在后续训练时，考虑对比去除异常图片和保留异常图片两种情况，选择更优的处理方式。

## 3.2 实现

### 3.2.1 使用单个预训练模型（基准模型）

使用 Imagnet 数据集预训练的 VGG16 模型进行迁移学习。大概步骤如下：

1. 导入预训练模型及相应的权重。
2. 将预训练模型的所有卷积层锁定避免在训练是被修改。在模型后面添加 dropout 层和全连接层，用于分类训练。

```
#VGG16 base
x = Input(shape=(244, 244, 3))
x = Lambda(vgg16.preprocess_input)(x)

base_model = VGG16(include_top=False, input_tensor=x, weights='imagenet', pooling='avg')
for layer in base_model.layers:
    layer.trainable = False

# 自定义分类层
y = Dropout(0.2)(base_model.output)
y = Dense(1, activation='sigmoid', kernel_initializer='he_normal')(y)

model = Model(inputs=base_model.input, outputs=y, name='Transfer_Learning')
sgd = SGD(lr=1e-3, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

3. 训练模型。

```
log_name = '/dogcat-EP{epoch:02d}-LOSS{val_loss:.4f}.h5'
log_dir = datetime.now().strftime('transfer_model_%Y%m%d_%H%M')
if not os.path.exists(log_dir):
    os.mkdir(log_dir)

es = EarlyStopping(monitor='val_loss', patience=20)
mc = ModelCheckpoint(log_dir + log_name, monitor='val_loss', save_best_only=True)
tb = TensorBoard(log_dir=log_dir)

model.fit(x=X_train, y=y_train, batch_size=16, epochs=10, validation_data=(X_val, y_val), callbacks=[es, mc, tb])
```

4 个模型经过 10 代训练后，验证集准确率均在 0.98 左右，测试集 logloss 为 0.08173。

## 3.2.2 模型融合

为了进一步提升模型识别能力，考虑将多个预训练模型的输出结果进行合并，将各个模型关注的点汇总在一起。再通过 dropout 层和全连接层进行分类训练。

大致步骤如下：

1. 选取 kaggle 给出的预训练模型中，准确率最高的 3 个模型。Xception、InceptionV3 和 IncetionResNetV2。

| 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 |
| 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 |

2. 载入三个预训练的模型，在每个模型后面增加全局平均池化层。
3. 对每个模型，将训练数据通过模型进行预测，得到输出的特征向量，并存入 h5 文件中。

```
def export_gap(MODEL, preprocess=None):
    x = Input((244, 244, 3))
    if preprocess:
        x = Lambda(preprocess)(x)
    model = MODEL(input_tensor=x, weights='imagenet', include_top=False, pooling='avg')

    train_gap = model.predict(X, batch_size=256)
    test_gap = model.predict(X_test, batch_size=256)

    with h5py.File("gap_%s.h5" % MODEL.__name__, 'w') as f:
        f.create_dataset('train', data=train_gap)
        f.create_dataset('test', data=test_gap)
```

4. 对三个模型得到的 3 个特征向量进行合并，生成一个大的特征向量。

```
train = []
test = []
for file in ['gap_InceptionV3.h5', 'gap_InceptionResNetV2.h5', 'gap_Xception.h5']:
    with h5py.File(file, 'r') as f:
        train.append(np.array(f['train']))
        test.append(np.array(f['test']))
```

```
Feature Vector Shape for Model #0: (25000, 2048)
```

```
train = np.concatenate(train, axis=1)
test = np.concatenate(test, axis=1)
print(train.shape)
```

```
(25000, 5632)
```

5. 将特征向量连接 dropout 层和全连接层。

```
inputs = Input((X_train.shape[1],))
x = inputs
x = Dropout(0.25)(x)
y = Dense(1, activation='sigmoid')(x)

model = Model(inputs=inputs, outputs=y)
model.compile(loss='binary_crossentropy', optimizer='adadelta', metrics=['accuracy'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        (None, 5632)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 5632)              0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 5633
=================================================================
Total params: 5,633
Trainable params: 5,633
Non-trainable params: 0
_________________________________________________________________
```

6. 进行分类训练

```
model.fit(x=X_train, y=y_train, batch_size=128, epochs=10, validation_data=(X_val, y_val))
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 5s 255us/step - loss: 0.0824 - acc: 0.9718 - val_loss: 0.0256 - val_acc: 0.9916
Epoch 2/10
20000/20000 [==============================] - 1s 60us/step - loss: 0.0213 - acc: 0.9934 - val_loss: 0.0227 - val_acc: 0.9934
Epoch 3/10
20000/20000 [==============================] - 1s 60us/step - loss: 0.0174 - acc: 0.9947 - val_loss: 0.0180 - val_acc: 0.9946
Epoch 4/10
20000/20000 [==============================] - 1s 61us/step - loss: 0.0160 - acc: 0.9950 - val_loss: 0.0270 - val_acc: 0.9910
Epoch 5/10
20000/20000 [==============================] - 1s 60us/step - loss: 0.0132 - acc: 0.9960 - val_loss: 0.0166 - val_acc: 0.9948
Epoch 6/10
20000/20000 [==============================] - 1s 61us/step - loss: 0.0124 - acc: 0.9959 - val_loss: 0.0152 - val_acc: 0.9950
Epoch 7/10
20000/20000 [==============================] - 1s 59us/step - loss: 0.0118 - acc: 0.9964 - val_loss: 0.0159 - val_acc: 0.9952
Epoch 8/10
20000/20000 [==============================] - 1s 59us/step - loss: 0.0097 - acc: 0.9972 - val_loss: 0.0151 - val_acc: 0.9952
Epoch 9/10
20000/20000 [==============================] - 1s 60us/step - loss: 0.0089 - acc: 0.9971 - val_loss: 0.0192 - val_acc: 0.9942
Epoch 10/10
20000/20000 [==============================] - 1s 60us/step - loss: 0.0080 - acc: 0.9978 - val_loss: 0.0180 - val_acc: 0.9948
```

测试集 logloss 为 0.03951，明显优于单模型，因此后续考虑在此模型基础上进行进一步的优化。

## 3.3 改进

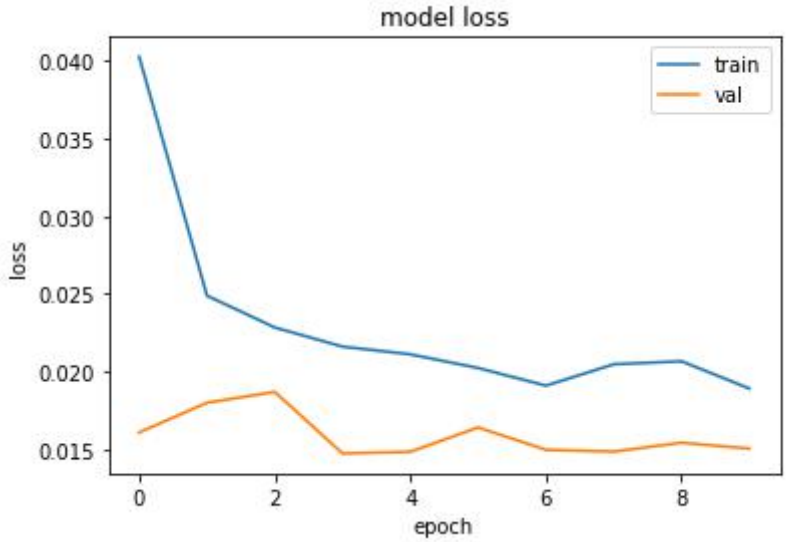
对于融合模型，可调节超参如：优化器选择、优化器学习率、epochs、batch_size、dropout 等参数。由于网格搜索同时对这些参数进行调节速度会非常慢，我们采用每次只对一个参数进行网格搜索，在每个参数上找到最优的配置。

### 3.3.1 Adadelta 优化器参数优化

优化器采用 Adadelta，batch_size 分别在 32、64、128、256，epochs 在 5、10、15、20，dropout 在 0.1、0.2、0.3、0.4、0.5、0.6、0.7、0.8、0.9。在每个参数上分别进行网格搜索。

```
#gridsearch 选取batch_size
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import log_loss, make_scorer
from keras.wrappers.scikit_learn import KerasClassifier

def create_model(dropout, optimizer='adadelta'):
    inputs = Input((train.shape[1],))
    x = inputs
    x = Dropout(dropout)(x)
    y = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=inputs, outputs=y)
    model.compile(loss='binary_crossentropy', optimizer='adadelta', metrics=['accuracy'])
    return model

model = KerasClassifier(build_fn=create_model, verbose=0)

batch_size = [32, 64, 128, 256]
epochs = [10]
dropout = [0.25]

scorer = make_scorer(log_loss, greater_is_better=False)

param_grid = dict(batch_size=batch_size, nb_epoch=epochs, dropout=dropout)
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring=scorer)

grid_result = grid.fit(train, label)
```

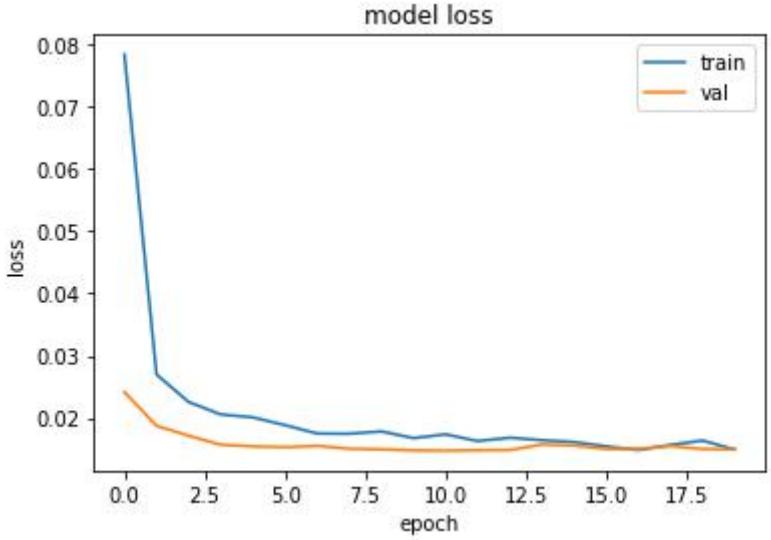
最终得到的最优组合为：batch_size=32, dropout=0.7, epoch=10。学习曲线如下：




在测试集上最终的 logloss 为 0.03924。

## 3.3.2 RMSprop 优化器参数优化

优化器采用 RMSprop，batch_size 分别在 32、64、128、256，epochs 在 5、10、15、20，dropout 在 0.1、0.2、0.3、0.4、0.5、0.6、0.7、0.8、0.9，学习率为 0.0001,、0.001、0.01、0.1。在每个参数上分别进行网格搜索。最终得到的最优组合为：lr=0.0001,batch_size=32, dropout=0.5, epoch=20。学习曲线如下：




在测试集上最终的 logloss 为 0.03916。

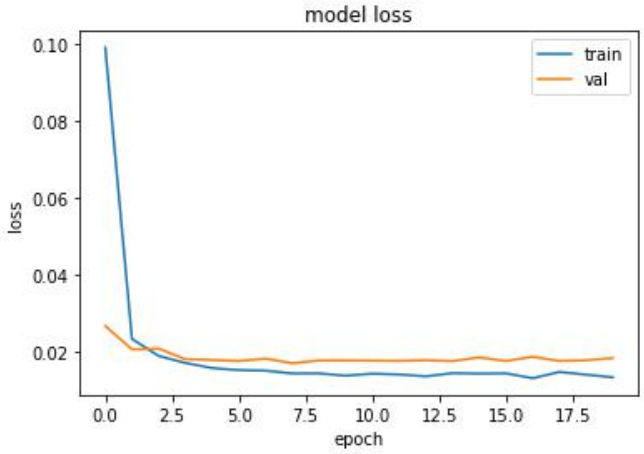
## 3.3.3 异常图片处理

在之前的分析中考虑到异常图片在测试集中也同样存在，如果将训练集中的异常数据去除，模型就学不会这些异常值，在最终提交时无法做出有利于分数的预测。因此没有将异常图片剔除。这部分我们将采用剔除掉异常数据的训练集，进行模型训练。对比剔除前后的模型性能，决定是否将训练集中异常图片排除掉。

根据之前的异常图片检查，在训练集中发现了 50 张异常图片，将他们清理之后，使用之前参数调优得到的 RMSprop 最优参数进行模型训练。最终在测试集的 logloss 为 0.03979，在测试集表现较差。可能是由于测试集中也包含异常图片导致，因此考虑不删除训练集中的异常图片。

## 3.3.4 图片尺寸修改为 299*299

将图片尺寸修改为 299*299 后，使用前面选择的最优参数 lr=0.0001,batch_size=32, dropout=0.5, epoch=20 进行了重新训练。学习曲线如下：



最终在测试集的 logloss 为 0.03680。排在 kaggle Public Leaderboard 的第 7 名。说明使用预训练模型默认的输入图片尺寸能够达到更好的效果。

# 4.结果

## 4.1 模型评价与验证

最终模型与预期结果一致，相比基准模型提升一倍以上。最终模型的超参通过网格搜索的方式，对模型中涉及的优化器选择、优化器学习率、epochs、batch_size、dropout 等参数进行调节。考虑到在 CNN 上使用网格搜索的速度非常慢，因此我们采用的是一种贪心搜索的方式，每次搜索一个参数的多项配置，在这个参数上选择最优的配置。经过调优后，最终测试集 logloss 由融合模型最初的 0.03951 提升至 0.03916。

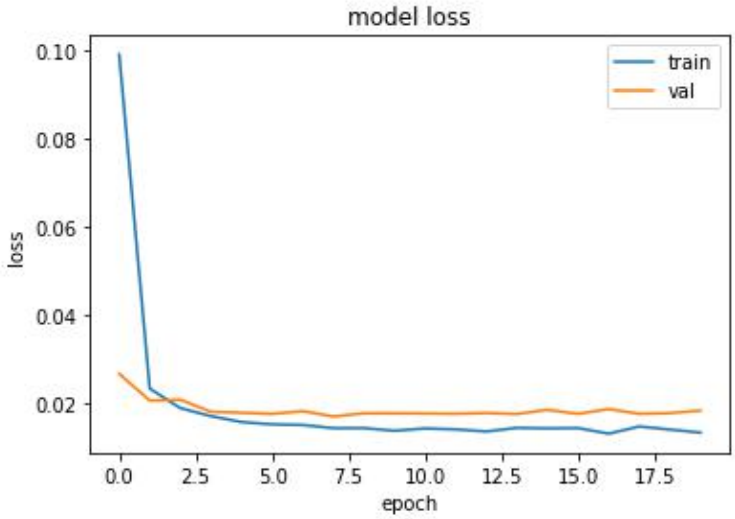
并且还对训练集中的异常数据剔除前后的测试集表现进行了对比，发现剔除前为 0.03951，剔除后为 0.03979。说明在训练数据中的一些微小的异常数据，不会对最终结果产生较大的影响。具有很好的鲁棒性。

最后将输入图片统一调整至 299*299 后，测试 logloss 为 0.03680。说明使用预训练模型默认的输入图片尺寸可以得到更优的解。

## 4.2 结果分析

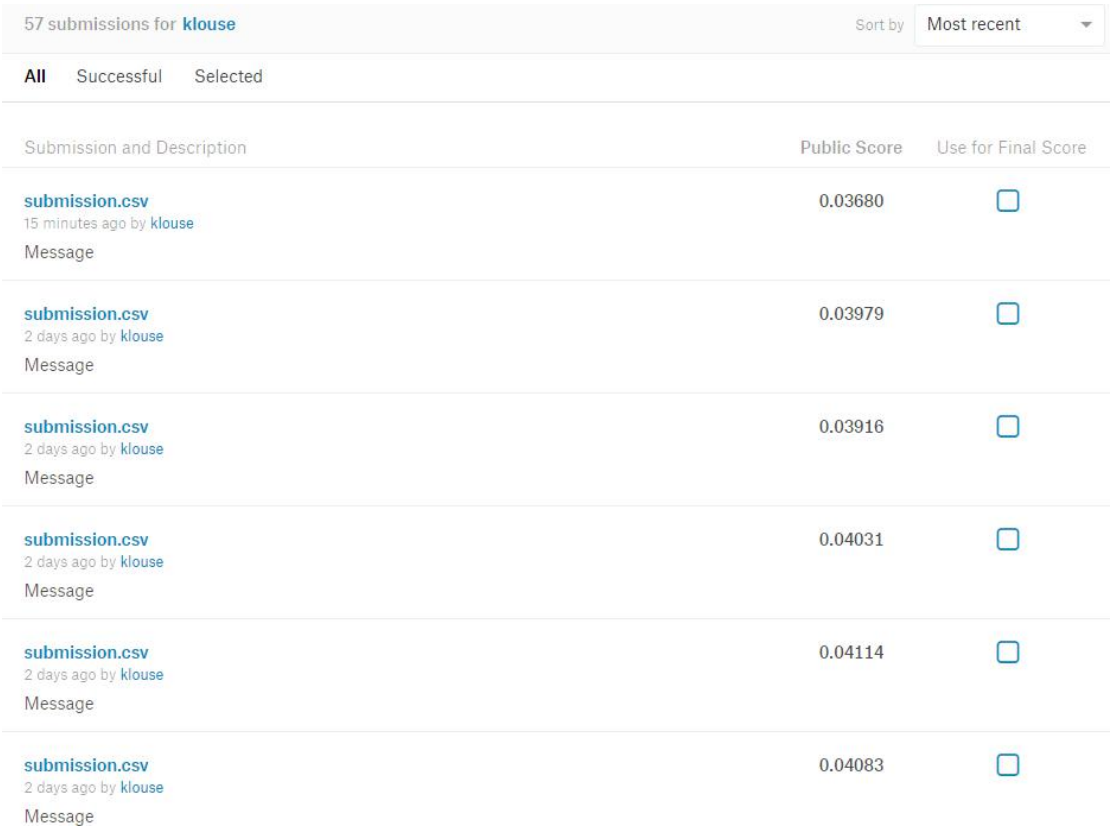
相比基准模型 VGG16 迁移模型的 0.08173 测试集表现，融合模型有明显的提升至 0.03680。且在 kaggle public leaderboard 排名 7 名。达到了项目的 public leaderboard 前 10% 的要求。

最终模型的学习曲线如下:




可以看出，随着 epoch 的增加，训练集 loss 和验证 loss 在逐步下降，而验证集基本趋于稳定，并趋近与训练集 loss。说明模型未出现过拟合或欠拟合的问题，模型参数配置合理。

在 kaggle 上测试结果如下：




# 5.结论

## 5.1 总结

项目采用迁移学习的方式，利用预训练的模型做为自己模型的输入，充分利用预训练模型的特征提取能力，站在别人的肩膀上，从而大大缩短了建模的难度并且有效的提高了模型性能。

首先利用预训练模型的预测能力，对训练集中的图片进行预测。对预测结果进行检查，发现图片中没有出现猫和狗，或者背景过于复杂，主体不明显的图片。基本可以替代人眼进行检查，只需人工对模型筛选后的图片进行检查，大大提升了工作效率。

其次使用 VGG16 单模型进行迁移学习，在测试集的 logloss 为 0.08173。为了进一步提升模型性能，使用模型融合的思想，将多个模型的特征输出结果进行合并，作为自己模型的输入。各个预模型之间可以起到互相补充的作用，使最终输入到模型的特征向量可以全面的表现出图片各方面的特征，在使用 Xception、InceptionV3 和 IncetionResNetV2 模型进行融合后，测试集 logloss 为 0.03951 有明显的提升。

最后在融合模型上，项目测试了不同的超参组合，在优化器选择、优化器学习率、epochs、batch_size、dropout 进行调优，最终测试集 logloss 为 0.03680。成功达到设定的目标，排名在 kaggle public leaderboard 7 名。并且模型具有很好的鲁棒性，可以适应训练集中微小的异常数据。

## 5.2 持续改进

后续可以从以下方面进行改进：

1. 更多的数据：引入更多的数据集，也可以通过数据增强（Data Augmentation）、对抗生成网络（Generative Adversarial Networks）等方式来对数据集进行扩充，进一步提升模型的鲁棒性。
2. 多目标检测：通过一些先进的目标识别算法，如 RCNN、Fast-RCNN、Faster-RCNN 或 Mask R-CNN 等，来完成一张照片中同时出现多个目标的检测任务。再对每个目标进行单独的识别。
3. 目前处理的方式是将图片全部读入内存，再调用 cnn 进行预测。这种方法的问题是，当训练集过大时，内存无法装下所有训练集，导致内存溢出。为解决海量数据的问题，可以使用 ImageDataGenerator 的 flow_from_directory(directory)函数，以文件夹路径为参数,生成经过数据提升/归一化后的数据,在一个无限循环中产生数据。

# 参考文献

**[1] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition.**

**[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition.**

**[3] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision.**

**[4] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.**

**[5] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Densely Connected Convolutional Networks**

**[6] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions.**