

机器学习纳米学位

毕业项目: 猫狗大战
作者: Kyle Chen
日期: 20180519
版本: 20180519v1

I. 问题的定义

项目概述

- 项目涉及的相关研究领域
- 在猫狗大战项目研究中, 重点研究了深度学习在图像识别中的应用. 猫狗大战是一个典型的二分类应用场景, 主要用于将图片中的猫, 狗区分出来. 在此项目中, 输入是一张相片, 相片中, 可以是猫或狗, 当其中出现猫时, 期望预测结果为猫. 如若为狗时, 期望预测结果为狗.
- 在现实生活中, 不乏很多多分类问题, 但是我们只有在对二分类问题非常了解的情况下, 才能对多分类问题有更深的理解, 也对往后处理分类问题落地起到了至关重要的作用.
- 在图像识别类项目中, 使用迁移学习模型能大大提升识别的准确度. 在Keras中, 集成了VGG16, VGG19, ResNet50, Inception V3, Xception预训练模型. 我们可以直接使用它们来对我们当前的分类问题做训练.
- 在Xception迁移学习模型中, 我们使用到的ImageNet数据集是按照WordNet架构组织的大规模带标签图像数据集. 大约1500万张图片, 2.2万类, 每张图片都经过严格的人工筛选与标记. ImageNet类似于图像所有引擎.

问题陈述

- 解决办法所针对的具体问题
- 在开始之前, 就有一个比较棘手的问题, 那就是关于判断异常数据. 这个过程我会在后续的文档中提及. 这里只是大致介绍一下, 我们可以使用预训练模型

将有猫狗的图片筛选出来,但是这里有个百分比的优化问题,我们需要通过不断的优化代码中的top参数,让其准确的将这些异常数据给筛选出来,最后删除.

- 在此项目中,我们需要解决针对图像的训练与分类问题,这是一个有监督学习的二分类问题.首先,要先对训练集中的数据进行训练;在多次训练与学习中,提升对数据预测的准确度;其次,在训练完成之后,对测试集进行预测与评分.
- 这里,我们并不会对Xception已经训练好的层级做训练,我们仅仅是利用预训练模型来导出特征权重,并针对后续添加的全连接层与输出层做训练.

评价指标

- 可以使用模型在测试集中的得分(LogLoss)来对指标评估.
- 在kaggle页面中,也为我们提供了验证LogLoss函数:

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

\hat{y} 表示我们预测出来的结果, y 表示图片的正确归类, n 表示样本个数.

- 在使用LogLoss函数作为评估指标时,分值越低,代表模型的表现越好.
- 当然,很重要的一点,在最后,需要进入kaggle猫狗大战挑战中的前10%(由于此比赛已结束,目前只能在private leaderboard获取评分,并以猫狗大战的public leaderboard top 10%的分数为基准,作为评分标准), LogLoss分数大概在0.06127以下.

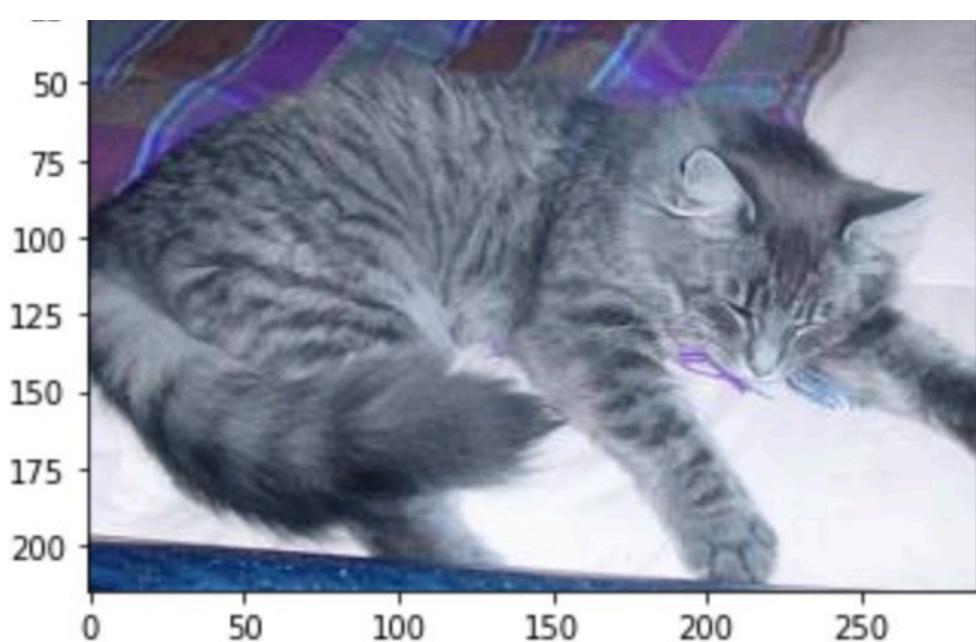
II. 分析

数据的探索

- 在此项目中,输入应为一张图片.图片中可以是猫或狗.当出现狗时,则期待分类器能将其归类为狗这一类;否则,我们期待我们的分类器能将其归类为猫这个类型;
- 随机抽查20张训练集中的图片文件:

DataSet/train/dog.9522.jpg

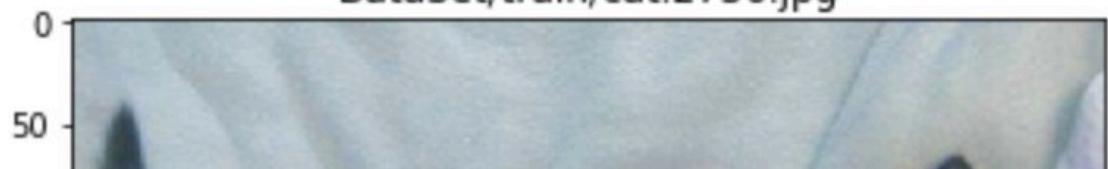


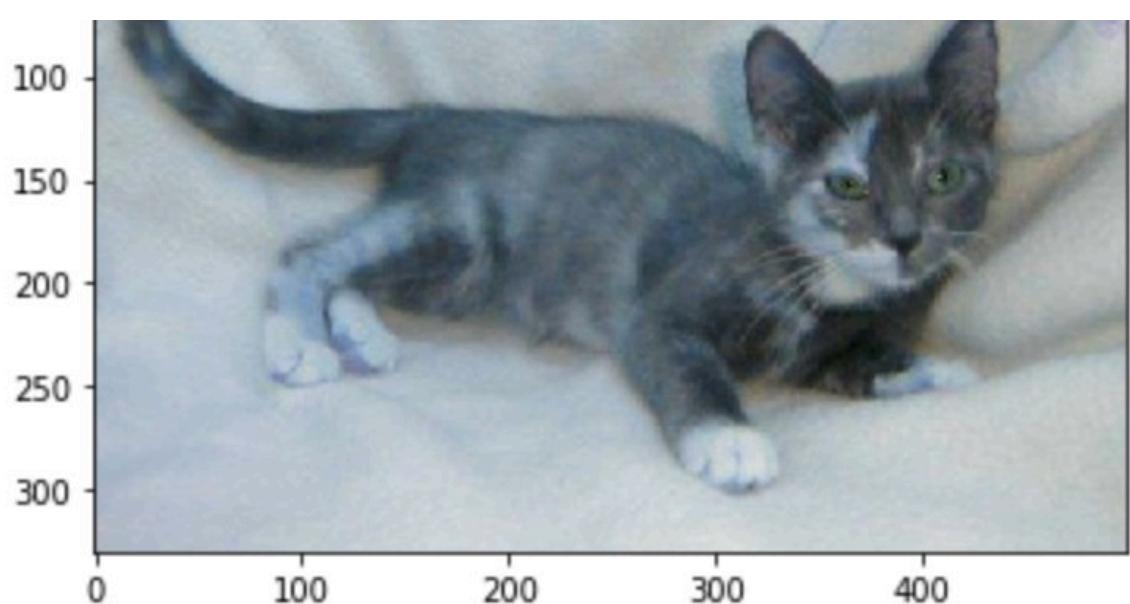


DataSet/train/cat.2736.jpg



DataSet/train/dog.330.jpg

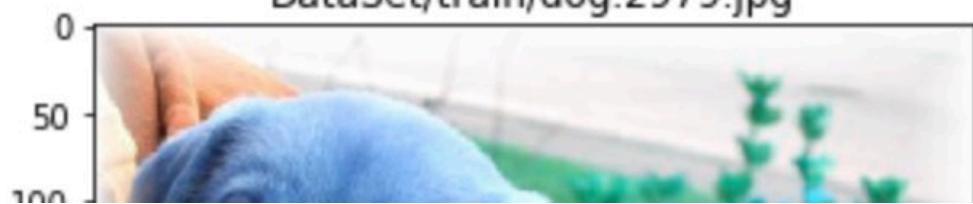




DataSet/train/dog.6556.jpg

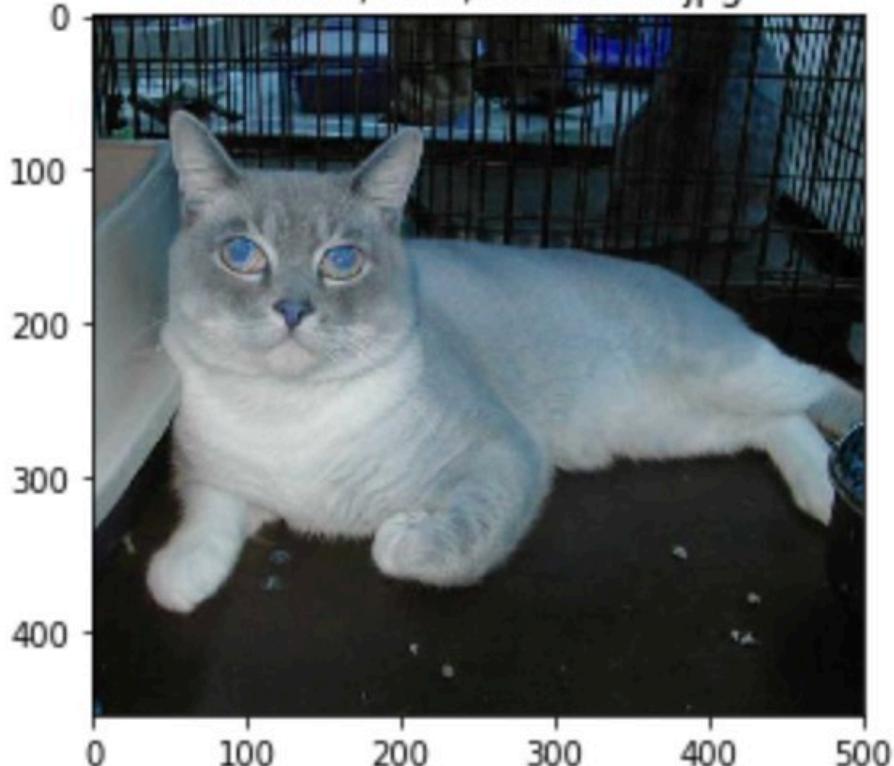


DataSet/train/dog.2979.jpg





DataSet/train/cat.10598.jpg

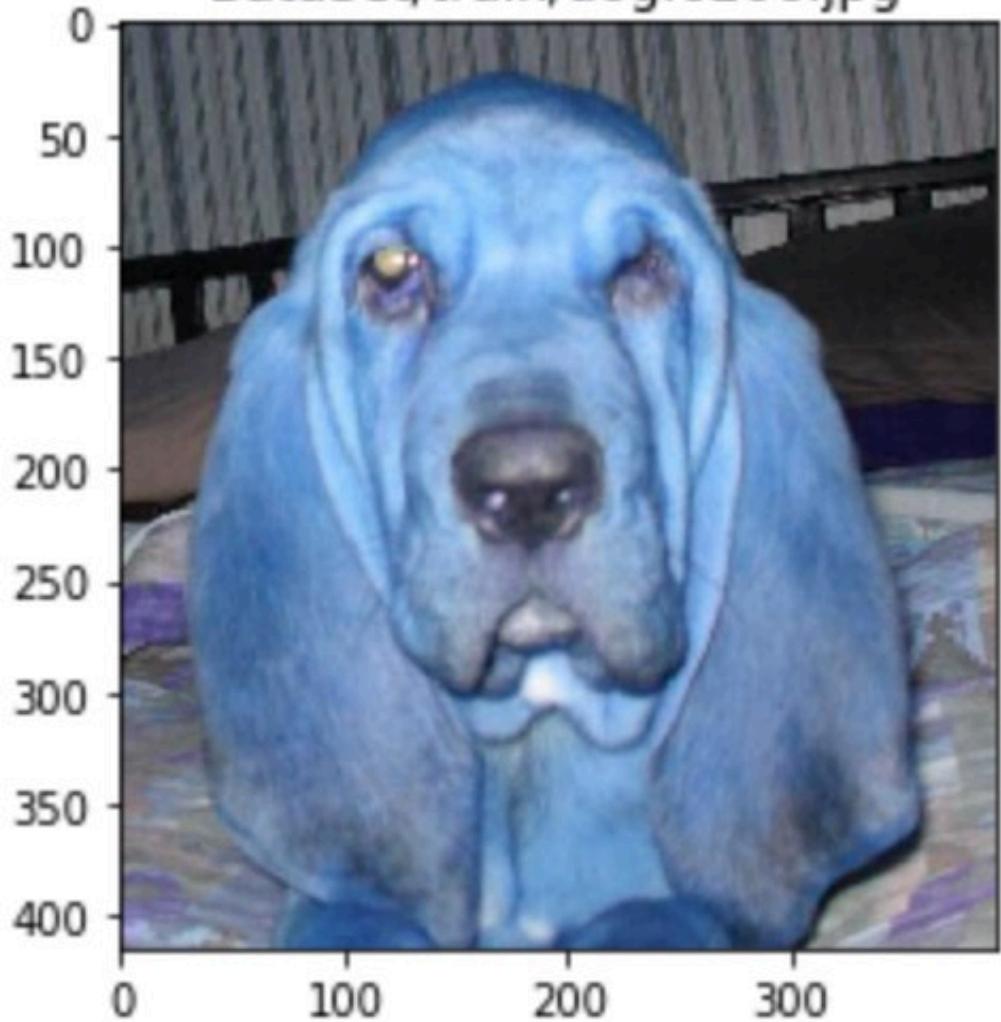


DataSet/train/dog.6336.jpg





DataSet/train/dog.6206.jpg

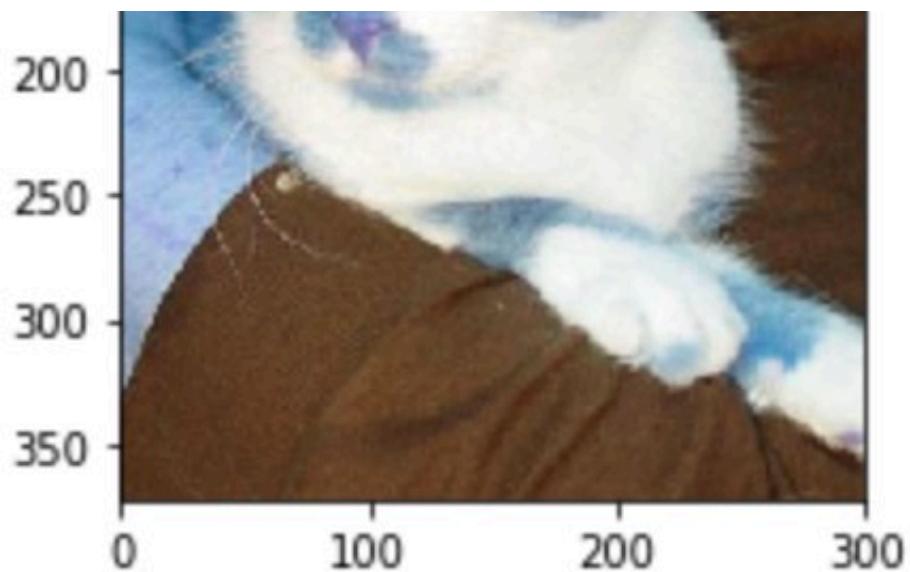


DataSet/train/cat.1930.jpg



DataSet/train/cat.7974.jpg

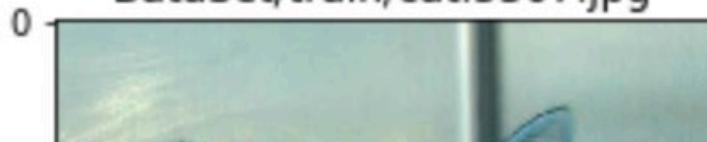


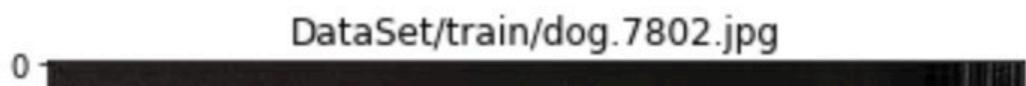
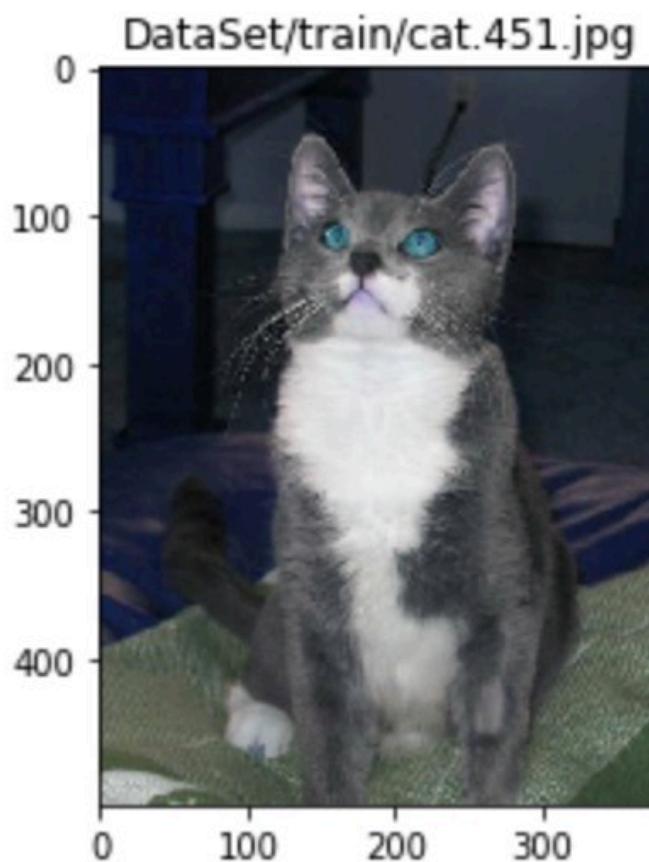
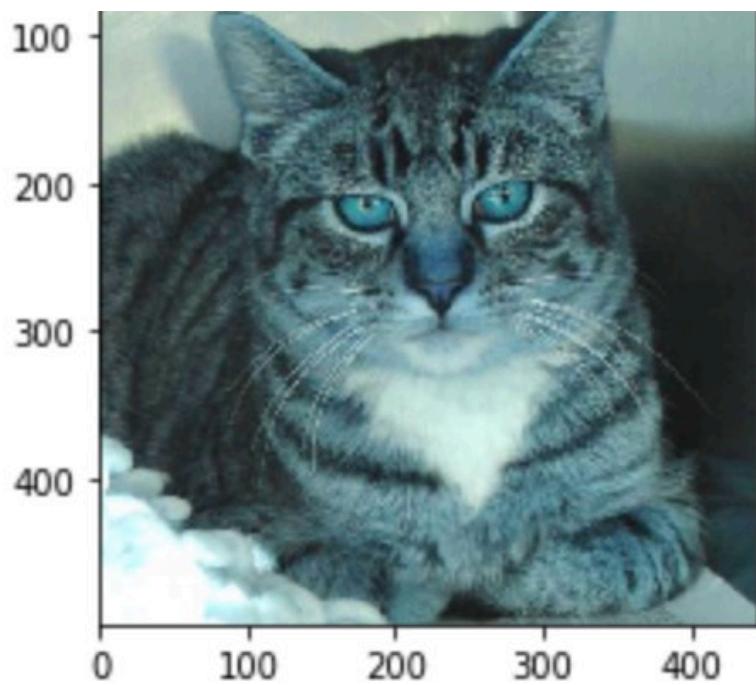


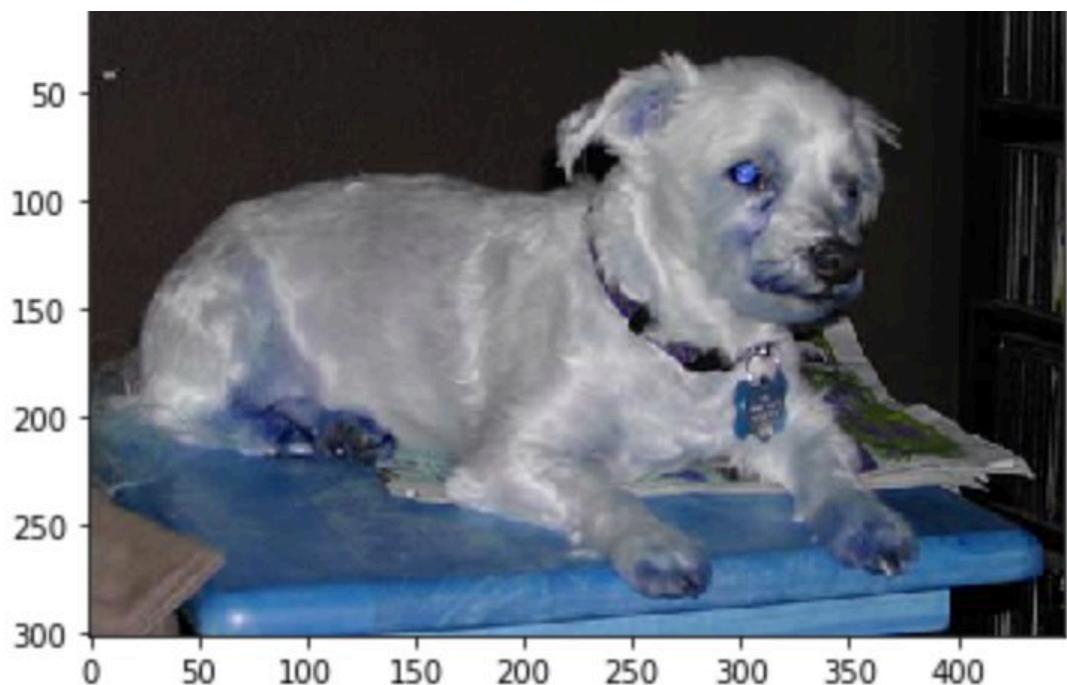
DataSet/train/cat.10771.jpg



DataSet/train/cat.5307.jpg



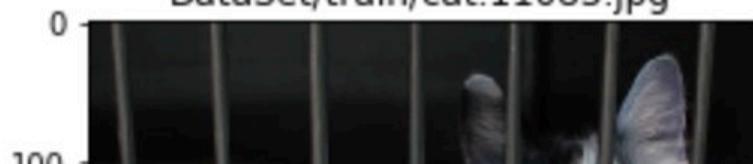


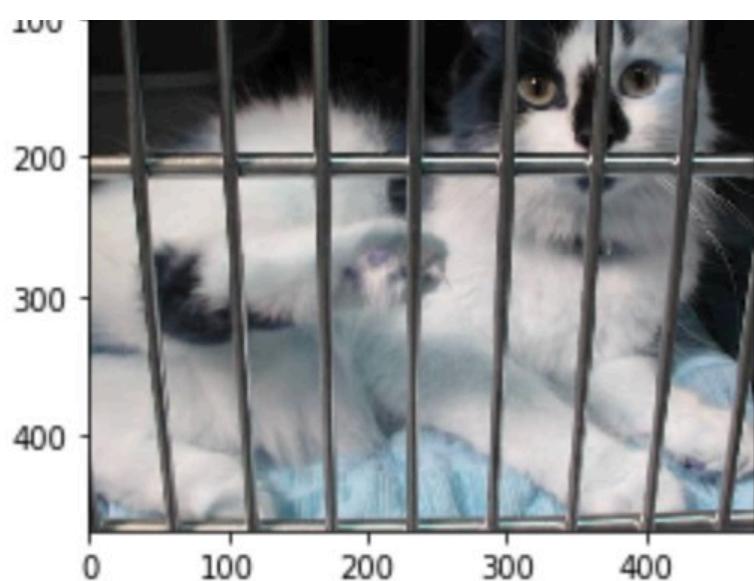


DataSet/train/dog.11108.jpg



DataSet/train/cat.11083.jpg

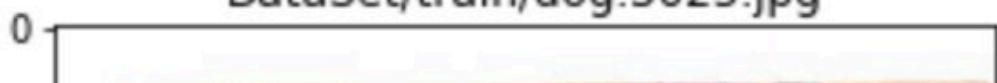


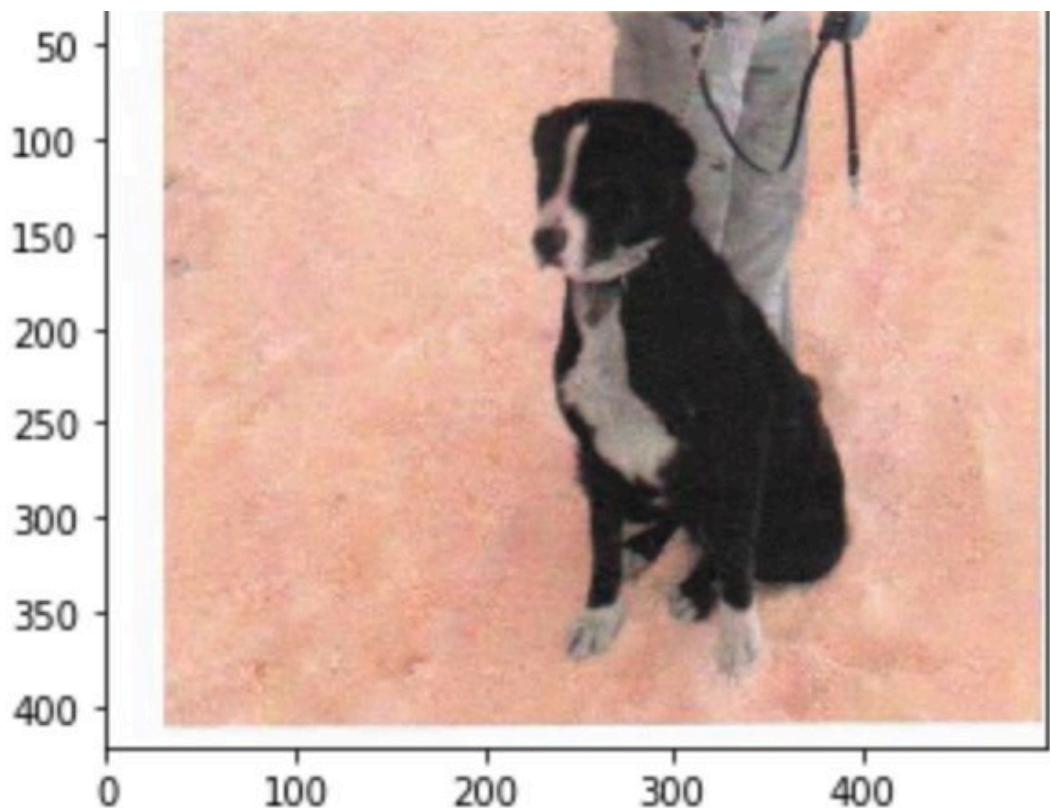


DataSet/train/dog.11786.jpg



DataSet/train/dog.3025.jpg





- 不难发现, 这里的图像宽高比不一定一致, 所以我们会在加载数据的时候限定图像的尺寸, 比如说(299, 299, 3).
- 在代码实现中, 可以使用keras.preprocessing中的image库加载RGB图像, keras.preprocessing.image中的ImageDataGenerator也为我们提供了简单易用的flow_from_directory()函数.
- 研究下kaggle给我们提供的样本:

```
→ dogs-vs-cats-redux-kernels-edition ✘ ls -ahl train/ | grep -i cat |  
head -n 3  
-rw-r--r--    1 Kyle  staff   12K Sep 20  2013 cat.0.jpg  
-rw-r--r--    1 Kyle  staff   16K Sep 20  2013 cat.1.jpg  
-rw-r--r--    1 Kyle  staff   34K Sep 20  2013 cat.10.jpg  
  
→ dogs-vs-cats-redux-kernels-edition ✘ ls -ahl train/ | grep -i dog |  
head -n 3  
-rw-r--r--    1 Kyle  staff   31K Sep 20  2013 dog.0.jpg  
-rw-r--r--    1 Kyle  staff   24K Sep 20  2013 dog.1.jpg  
-rw-r--r--    1 Kyle  staff   12K Sep 20  2013 dog.10.jpg
```

不难发现, 样本中的Y, 就是文件的prefix = [cat | dog], 标签和样本是绑定在一起的, 这方便了我们对样本打乱.

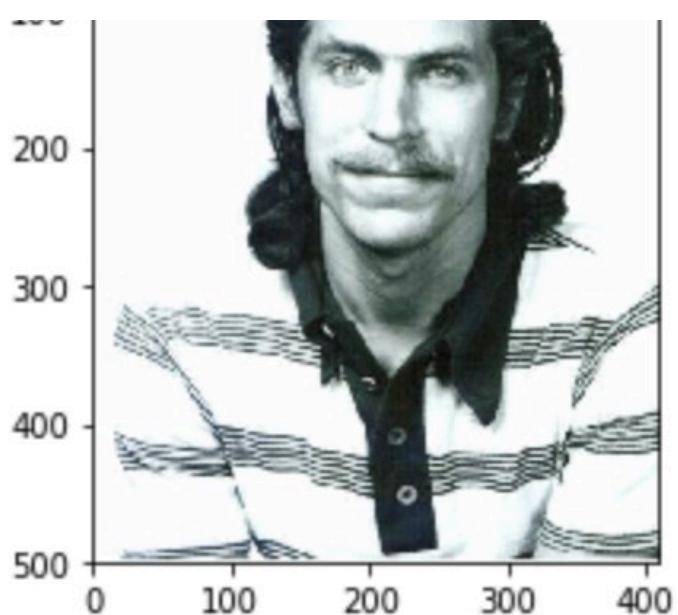
- 接着我们统计下训练集中猫, 狗的类型分布:

```
→ dogs-vs-cats-redux-kernels-edition ✘ find train/ -name "cat*" | wc -l  
12500  
→ dogs-vs-cats-redux-kernels-edition ✘ find train/ -name "dog*" | wc -l  
12500
```

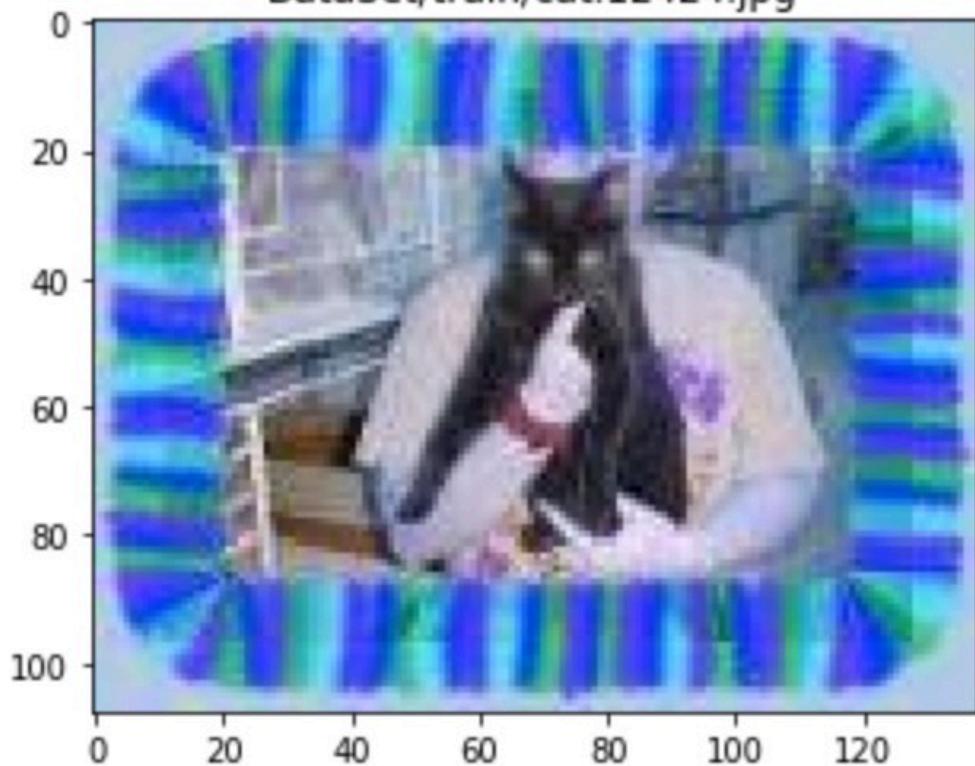
可以发现, 这里的猫狗是均匀分布的, 我们可以在训练集中取一部分来作为训练集, 另外一部分作为验证集.

- 在获得训练集, 验证集之前, 需要将train/目录下的文件打乱, 然后按照一定的比例将其划分到训练集, 验证集中.
- 当然, 在开始读取训练集数据之前, 我们需要对train/目录下的文件先做异常值的判断与处理. 这里我们采用Xception预训练模型先对图片做一个预测, 当图片中前n项预测结果都与猫狗类型不匹配时, 我们则将其当做异常数据处理. 在这里, 我尝试了很多次不同的top n, 并将其输出查看筛选出来的图片. 最终采用top 50作为一个界限, 发现如果在预测的前50位排名并未出现猫狗的时候, 基本上都是些难以判断的异常数据. 在开始对Xception结果中猫狗做判断之前, 我们还需要对Xception的top n结果做一个处理, 例如说, 哪些分类是猫, 哪些分类是狗, 我们需要对所有的分类结果汇总, 并处理成便于解析与使用的格式. 然后再将Xception预测出来的top 50结果与分类中的猫狗结果做匹配, 如若其中有猫或狗的分类, 我们则视为此图片正常, 否则视为异常值. 这里预训练模型的输出类型多达1000个, 所以在猫狗类型的识别文件中, 应该也有1000行对应的数据, 其中包含猫狗, 但在其之外的各种类型数据也包含其中. 在这里展现我们筛选出来的34个异常值:



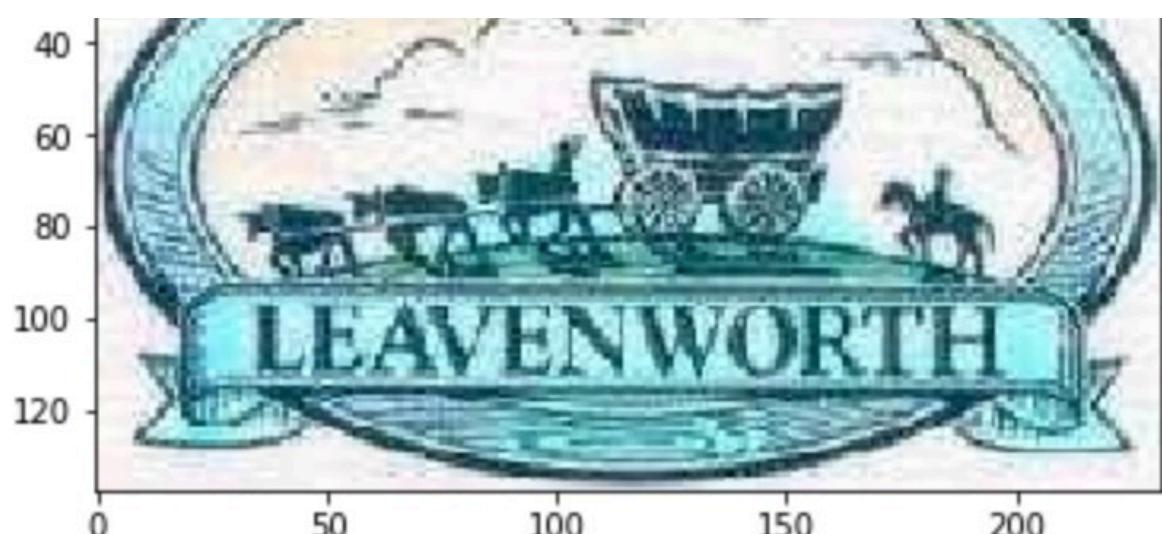


DataSet/train/cat.12424.jpg

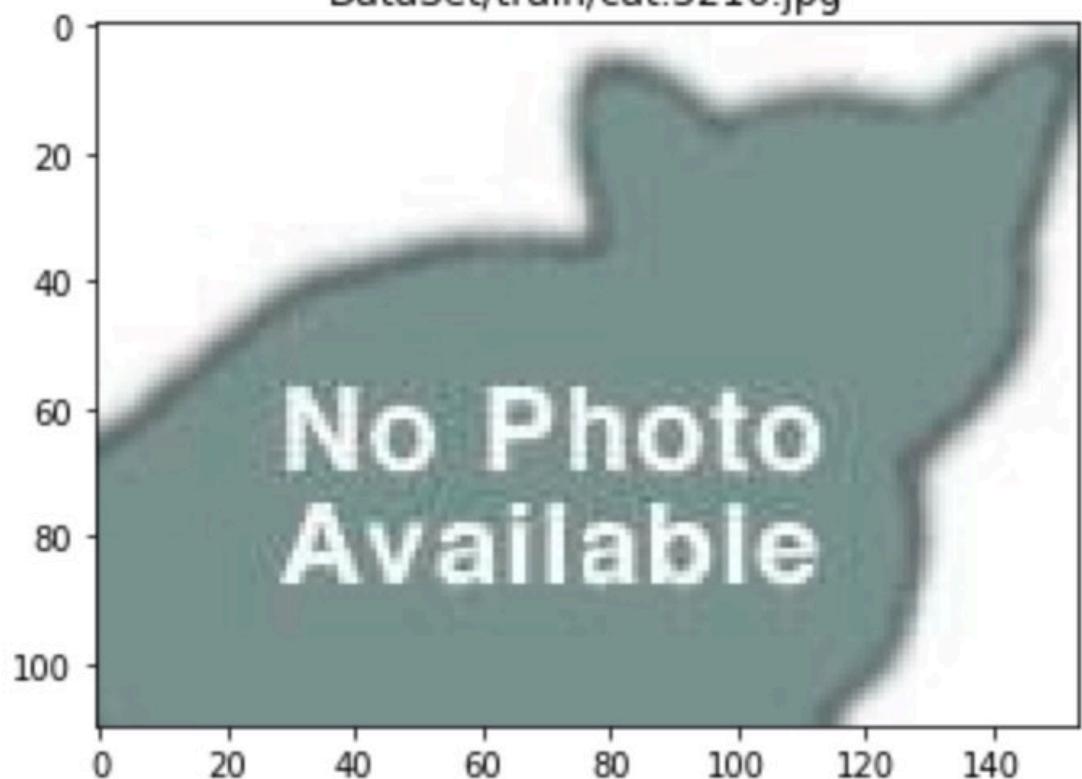


DataSet/train/dog.11299.jpg

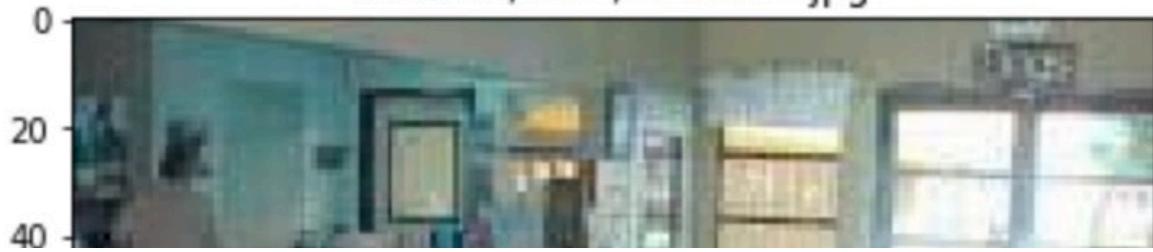


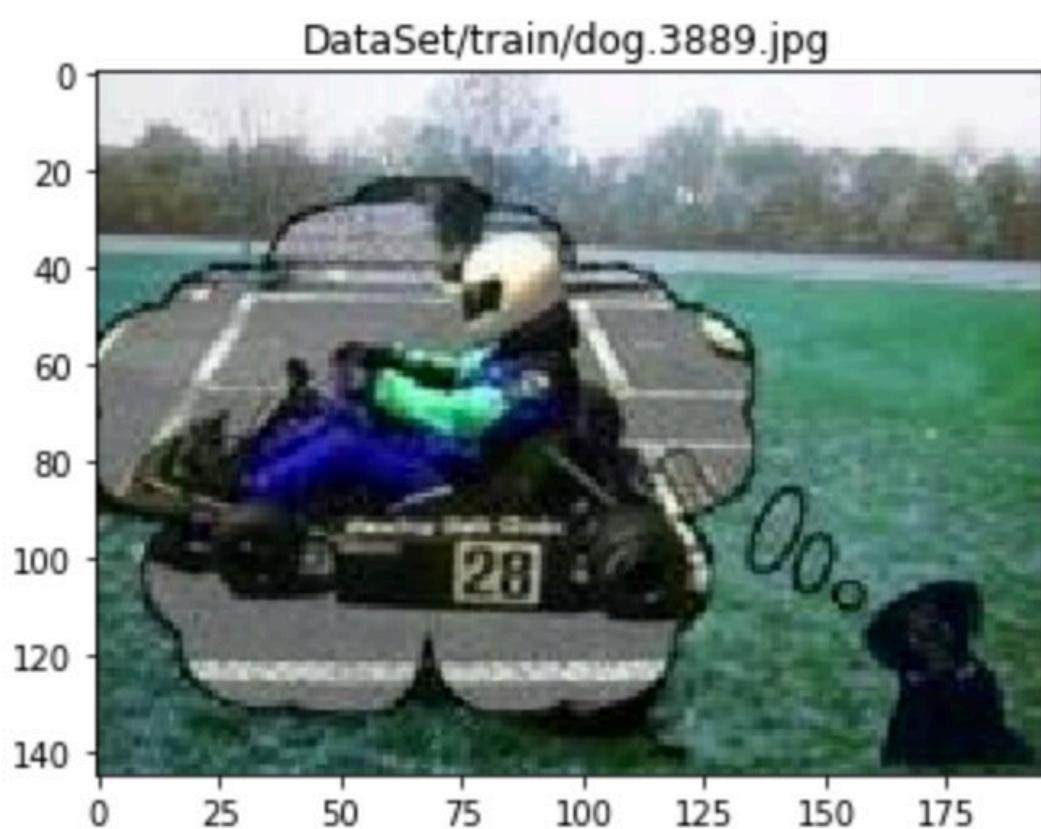


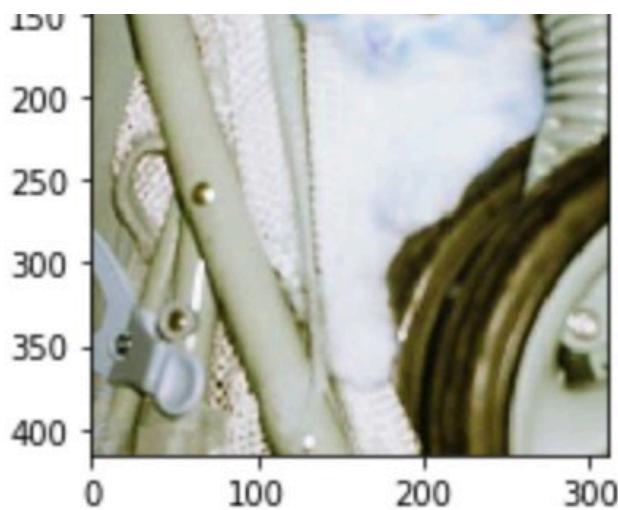
DataSet/train/cat.3216.jpg



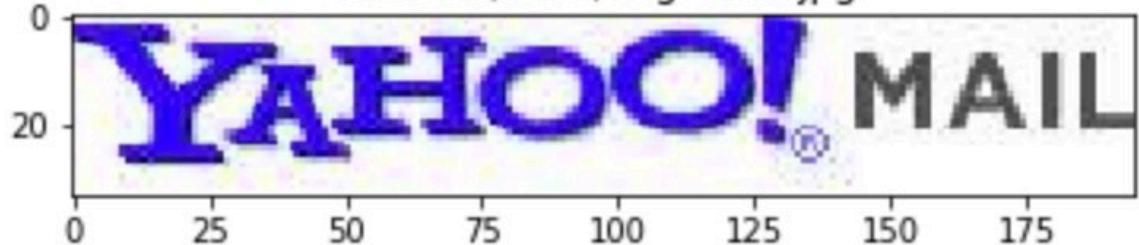
DataSet/train/cat.4338.jpg



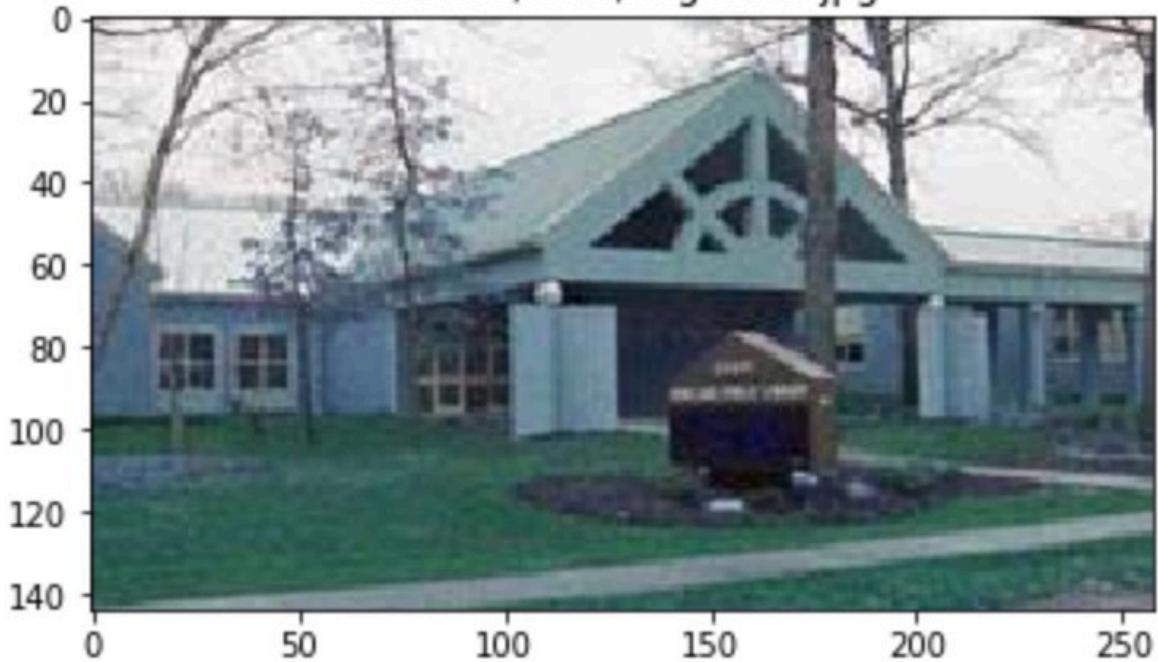




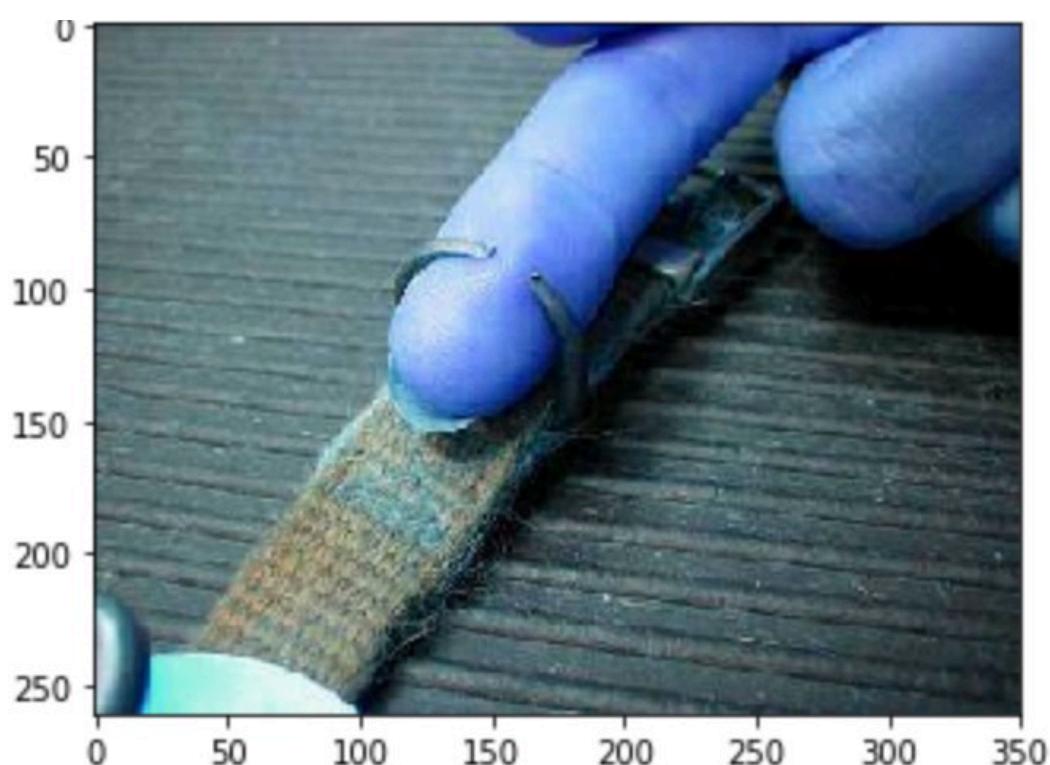
DataSet/train/dog.4367.jpg



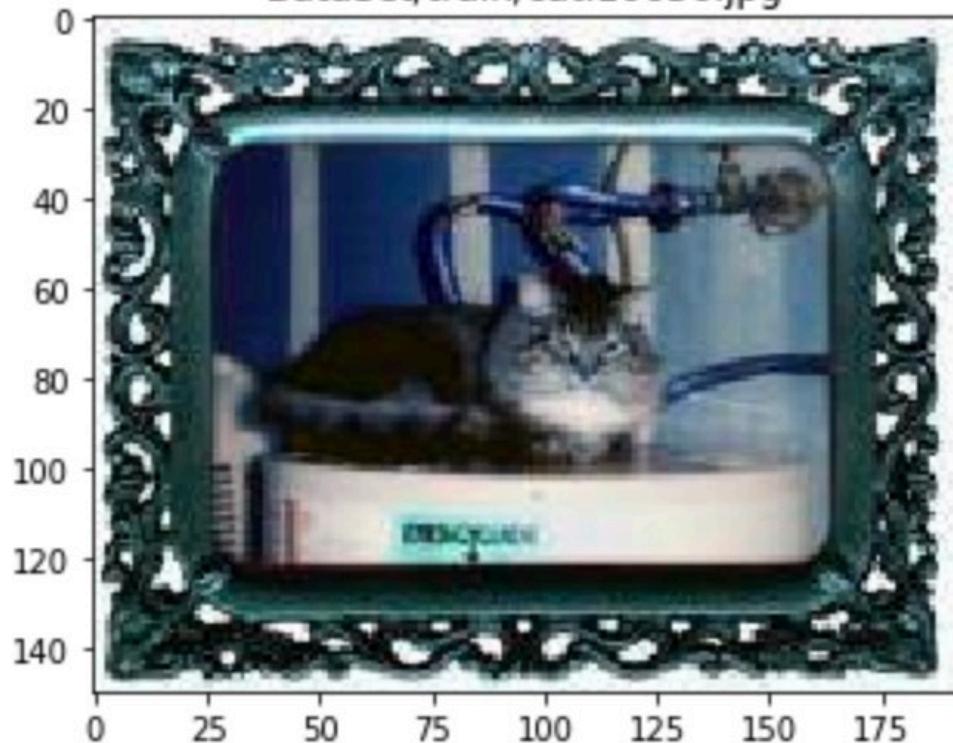
DataSet/train/dog.6475.jpg



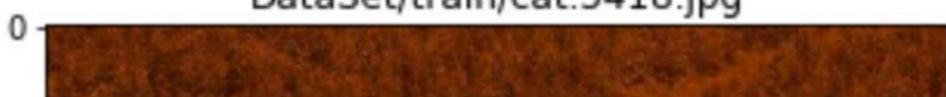
DataSet/train/dog.10801.jpg

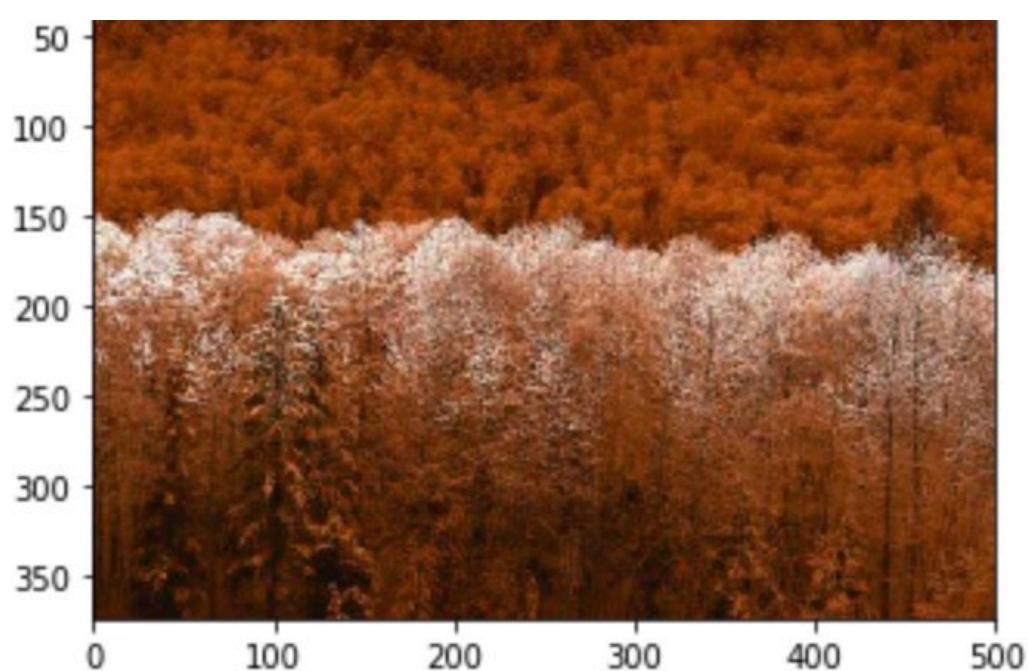


DataSet/train/cat.10636.jpg

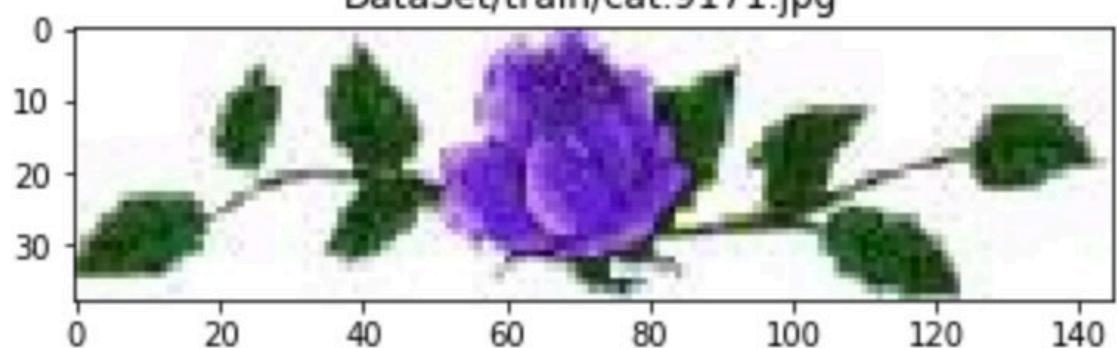


DataSet/train/cat.5418.jpg



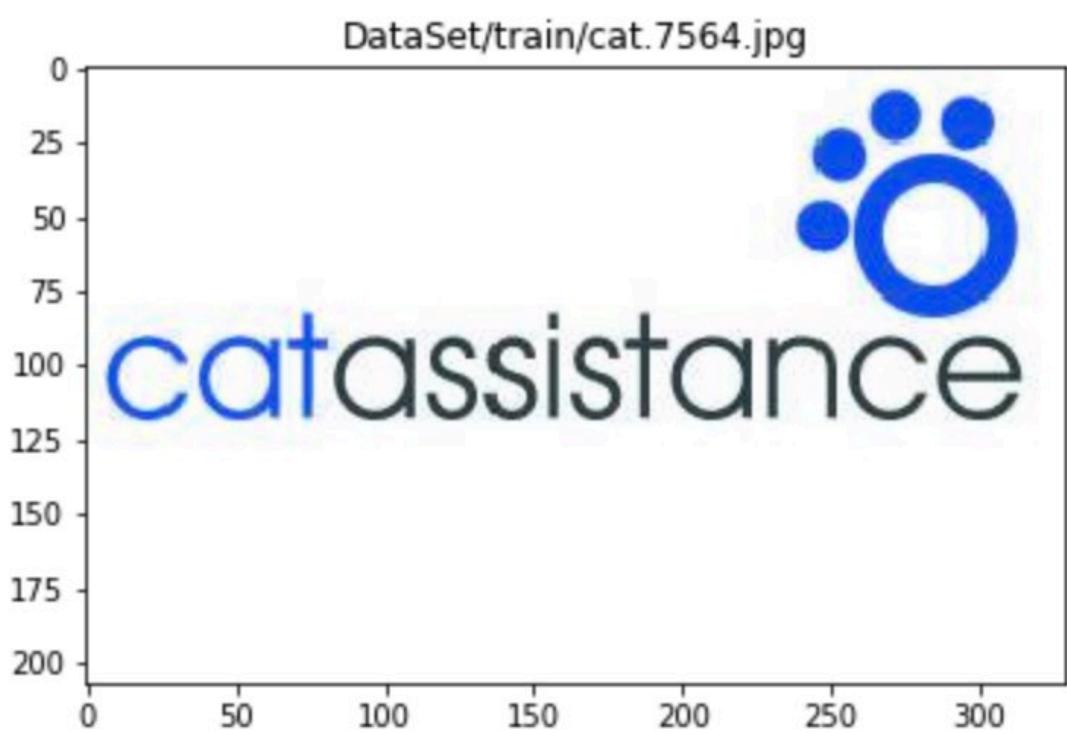
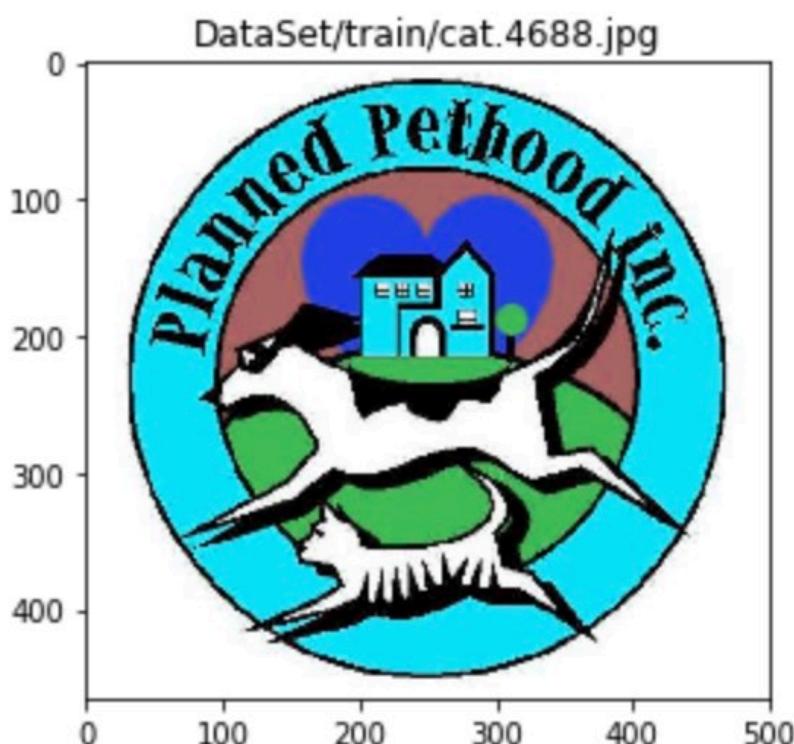
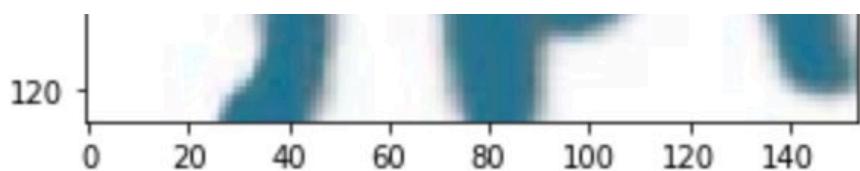


DataSet/train/cat.9171.jpg

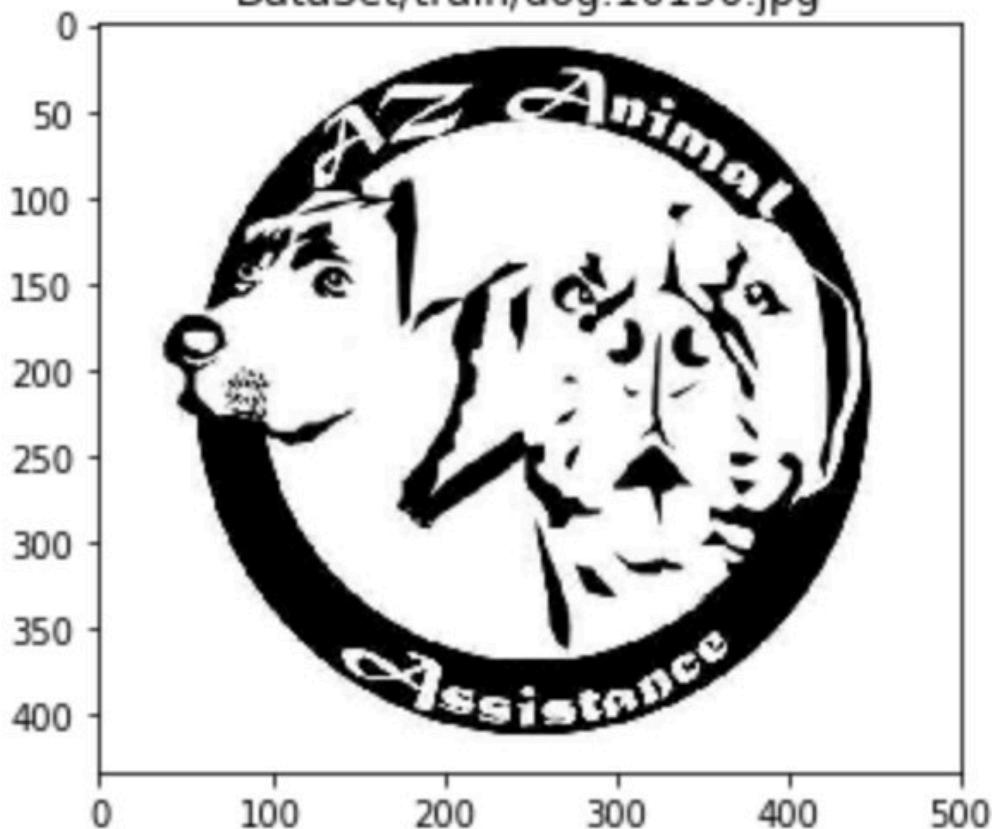


DataSet/train/dog.8898.jpg

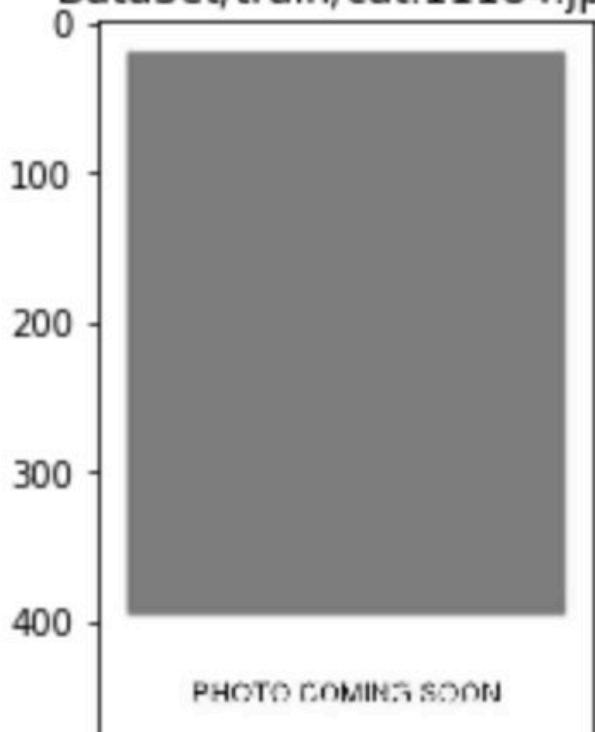


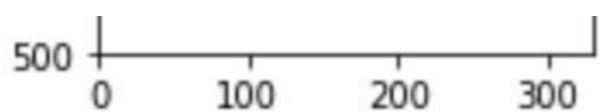


DataSet/train/dog.10190.jpg



DataSet/train/cat.11184.jpg

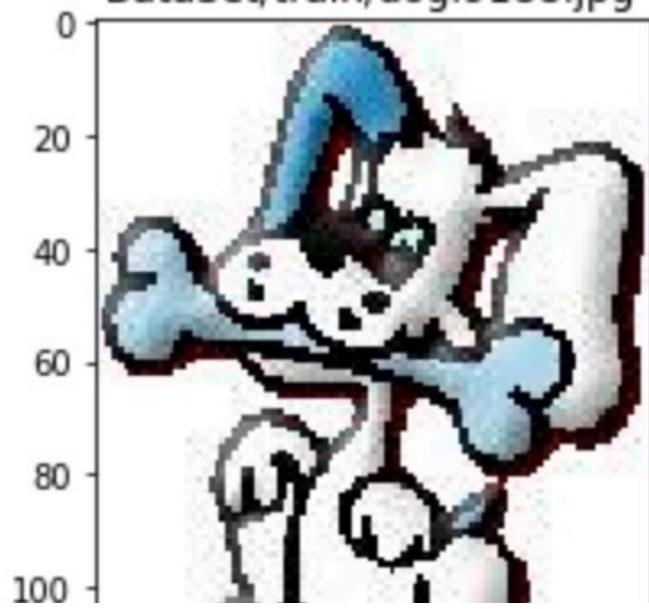


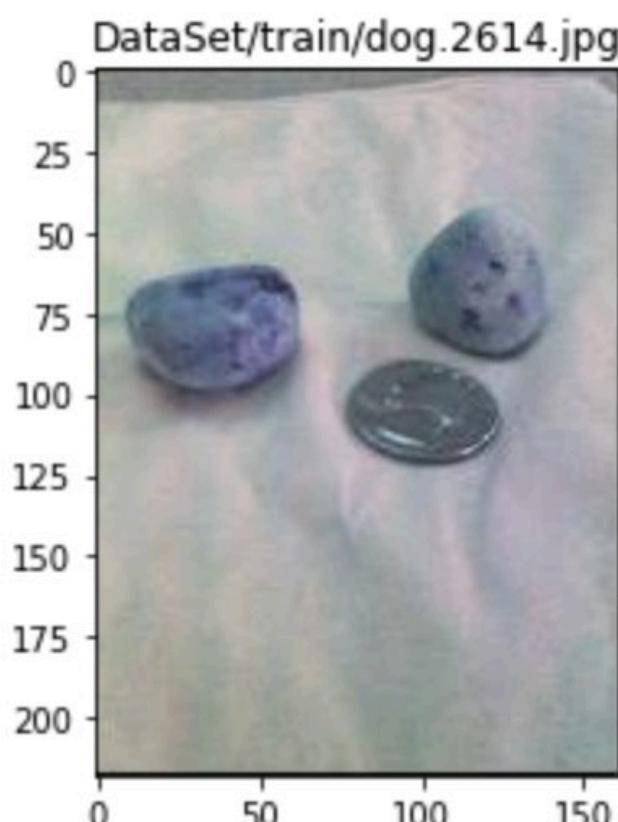
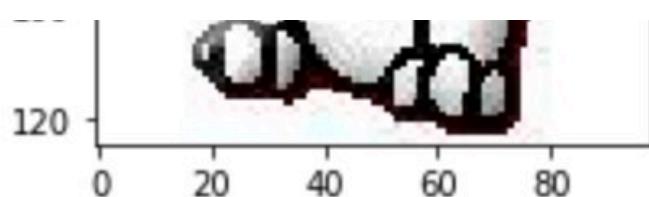


DataSet/train/cat.3672.jpg



DataSet/train/dog.9188.jpg





The Caring Containment Professionals.TM

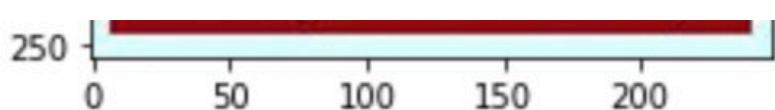
0 100 200 300 400

DataSet/train/dog.1895.jpg



DataSet/train/dog.10161.jpg



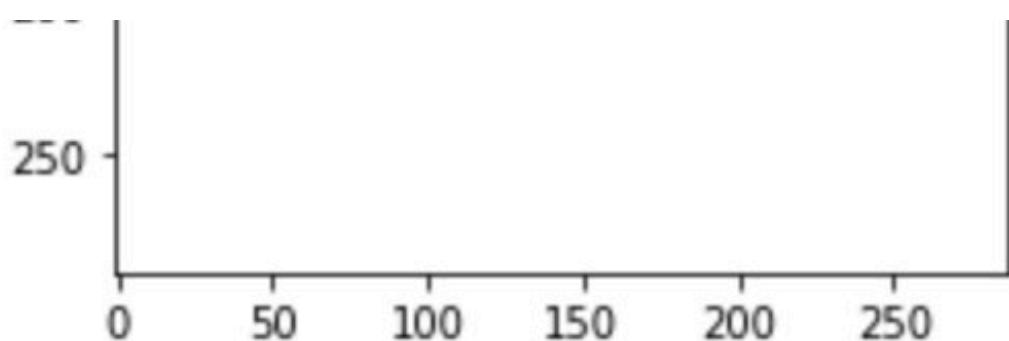


DataSet/train/dog.5604.jpg

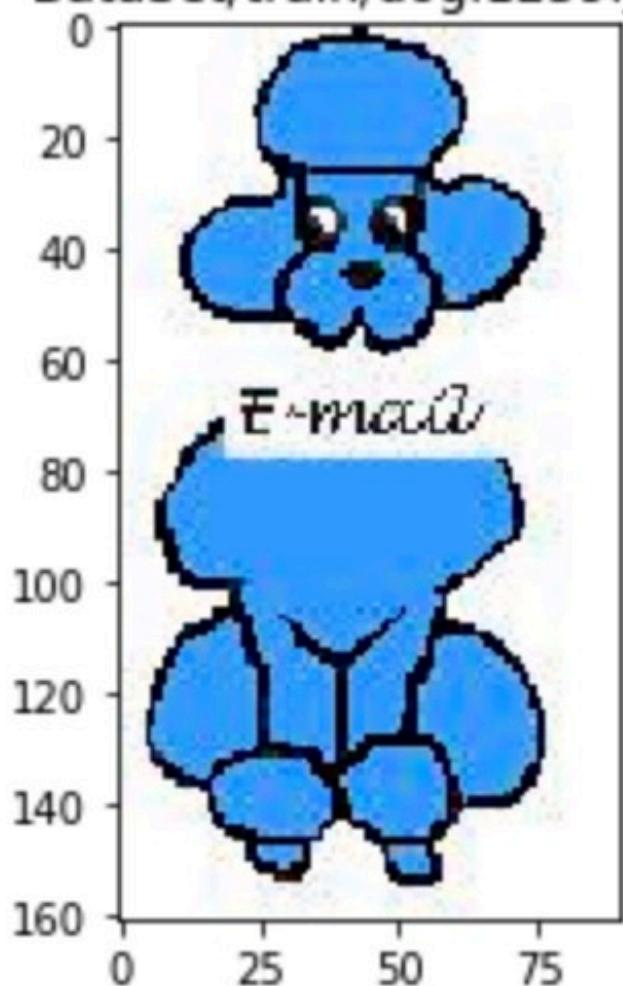


DataSet/train/cat.8456.jpg



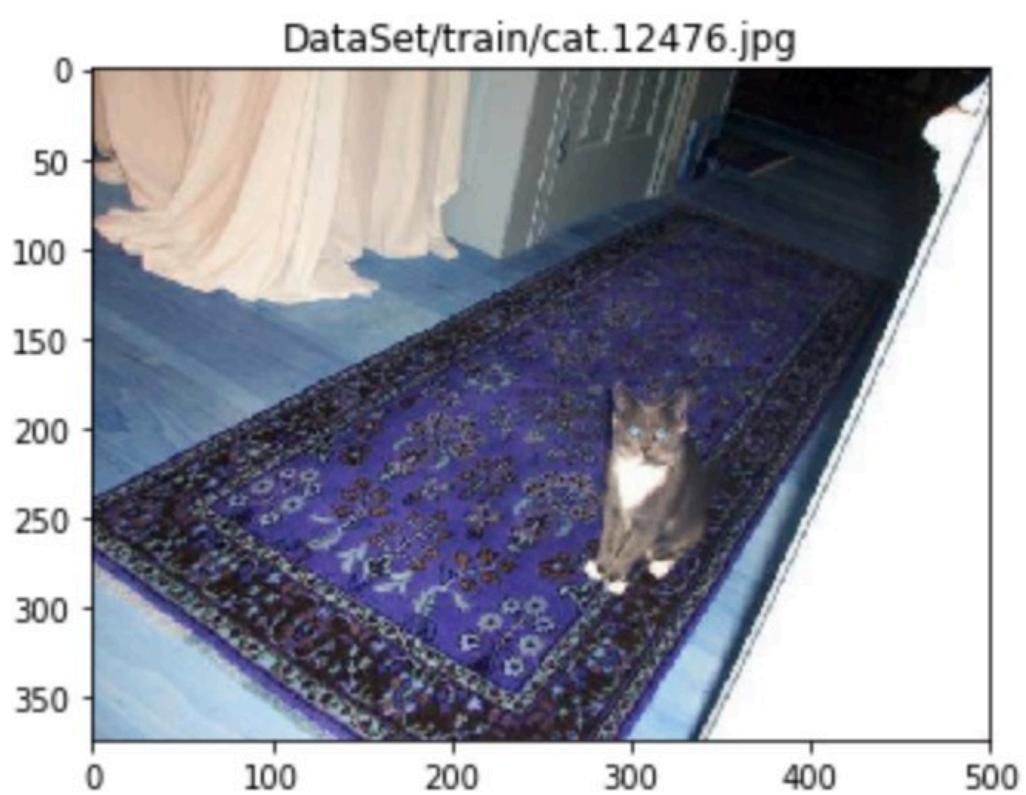
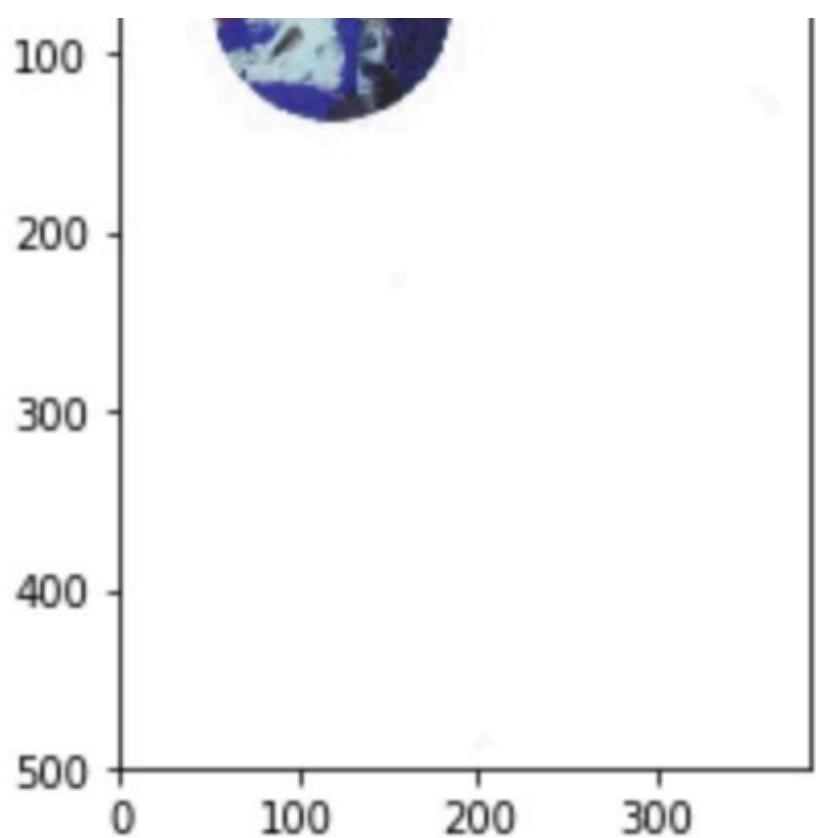


DataSet/train/dog.1259.jpg



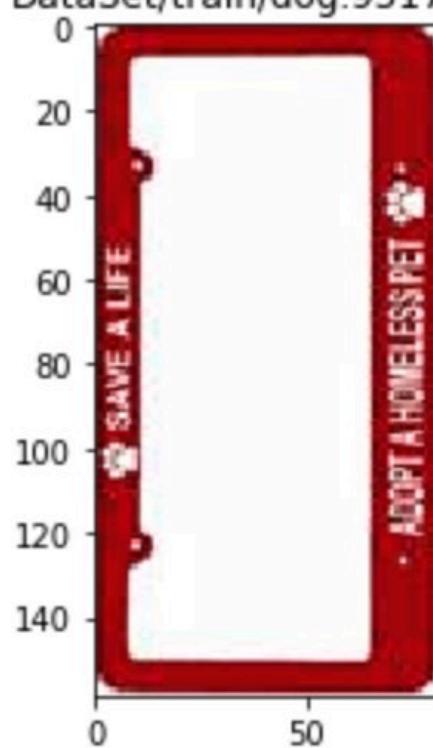
DataSet/train/cat.10029.jpg



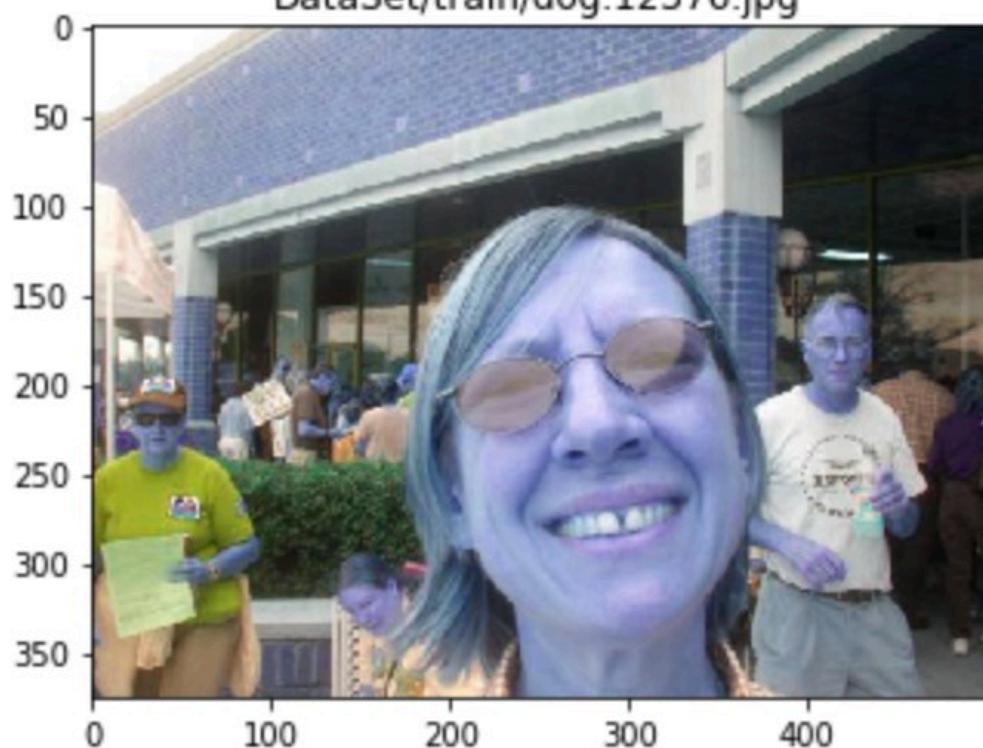


DataSet/train/cat.12476.jpg

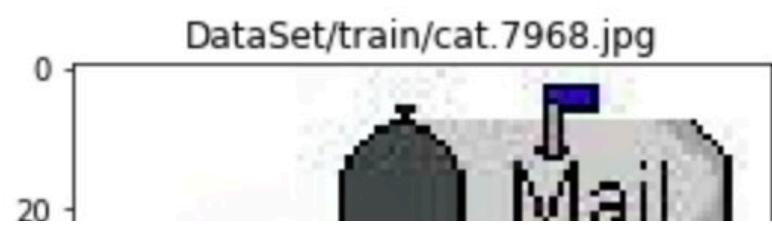
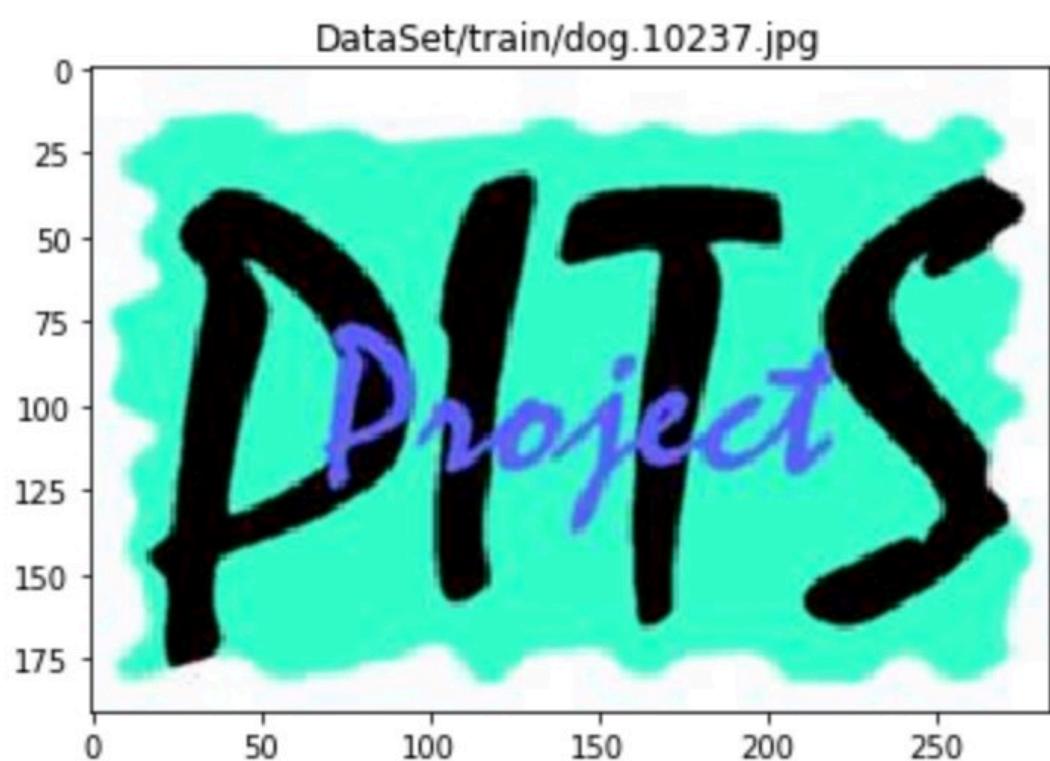
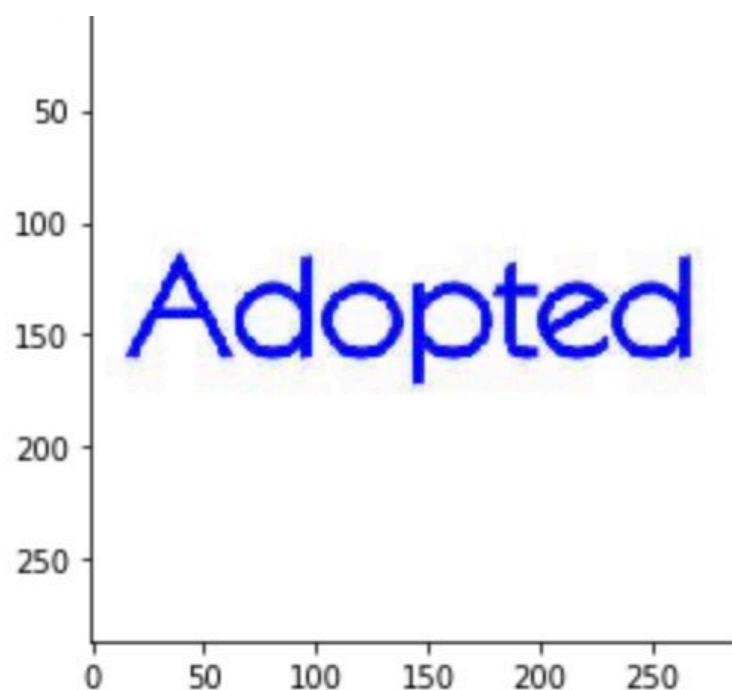
DataSet/train/dog.9517.jpg

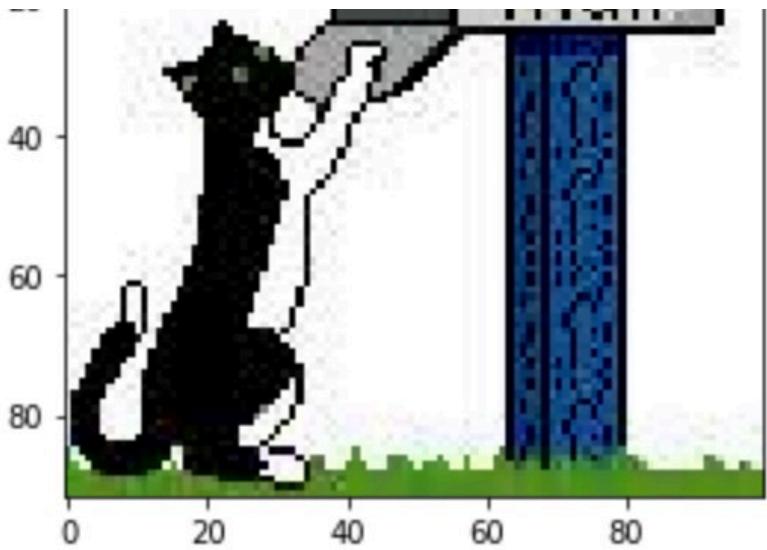


DataSet/train/dog.12376.jpg



DataSet/train/dog.8736.jpg



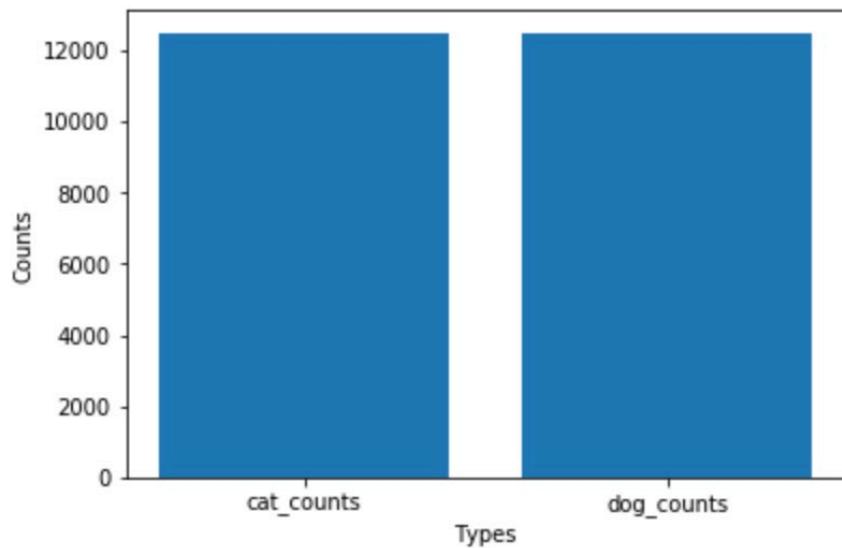


- 不难发现, 异常值的类型比较乱, 这里有卡通, 字母, 人物, 甚至还有占比极小的猫/狗图片.
- 注意, 这里的整个过程是全自动化去操作的, 但是在人为确认模型top n的准确度能满足我们的预测需求的情况下, 最终确定的 $n = 50$. 可以发现, 最终自动删除的这些图是符合我们的需求(异常值)的.

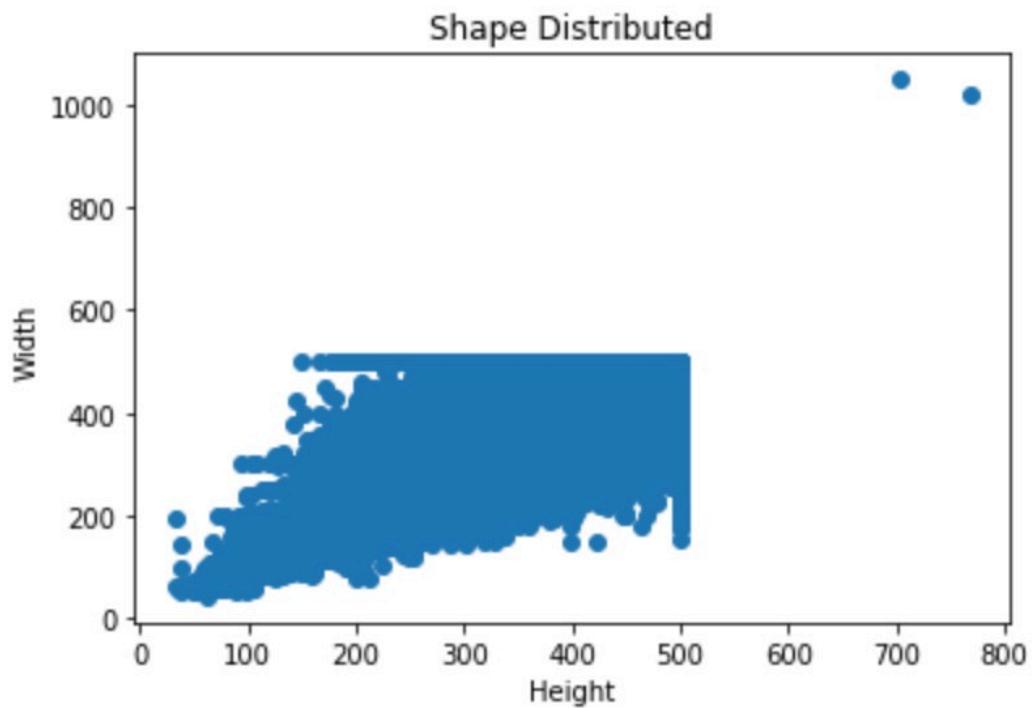
探索性可视化

- 如图, 去除异常值后的猫狗样本基本分布均匀.

```
cat_counts = sum([ 1 for x in label if x == 0 ])
dog_counts = sum([ 1 for x in label if x == 1 ])
plt.bar(['cat_counts', 'dog_counts'], [cat_counts, dog_counts])
plt.xlabel('Types')
plt.ylabel('Counts')
plt.show()
```



- 由于原始图像宽高比不一定一致, 所以我们会在加载数据的时候限定图像的尺寸, 例如(299, 299, 3), 下图直观地展示了数据集中的尺寸比例分布情况:



算法和技术

- 在一切开始前, 我们需要准备我们的数据集. 在猫狗大战这个项目中, 可以直接从kaggle上下载DataSet. 在安装[kaggle api](#)之后, 我们可以直接在终端执行:

```
→ Dogs_vs_Cats ✘ kaggle competitions download -c dogs-vs-cats-redux-kernels-edition
```

- 接下来, 我们需要将DataSet中的数据分成训练集, 验证集, 测试集. 由于测试集已经被单独存放到test/目录下, 仅需将train/目录下的文件分成训练集与验证集, 可以参考比例7:3来划分. 当然除了shuffle(因为Model.fit()中的shuffle是在validation_split之后才做的), 验证集可以放到fit的时候, 用validation_split参数自动去生成.
- 迁移学习就是利用已经训练好的模型参数, 到新的模型上, 帮助新的模型训练. 我们可以将迁移学习中已经学习到的模型参数通过特征权重导出的方式来分享给新的模型, 以加快新模型的学习效率. 迁移学习并不是万能的, 要想使用它, 很大一部分取决于是否能站在前人的肩膀上, 迁移学习可以使预训练的深度网络对同一领域的不同应用生效, 例如说我们这次要做的猫狗分类问题, 就属于图像识别领域的一个子集. 在我们接下来的程序中, 会尝试Xception进行迁移学习模型训练, Xception中的ImageNet数据集是按照WordNet架构组织的大规模带标签图像数据集, 站在前人的肩膀上, 能为大幅提升程序的学习效率与准确率.
- 使用keras框架构建深度卷积神经网络, 这里我们使用Xception进行迁移学习训练, 在第一次调用时会自动到github上下载相关的训练好的特征权重模型, 供我们后面训练使用. 在构建模型时, 并不是直接将其加载进来就能直接使用, 我们需要将其嵌入我们需要训练的模型中去. 例如说, 猫狗大战, 是一个二分类问题, 所以我们需要将最终的预测结果修改为两类(猫/狗).
- 这里有两种方法可以将Xception融入到模型中去, 一种是直接将其加载到我们的模型中, 构建好模型, 选择冻结其中的某些层, 做出拟合; 一种是将我们的图片作为输入, 用Xception去对图片进行预测, 最终导出处理后的特征权重, 再构建后续的框架, 并将其作为输入.
- 在这里, 将不会直接把Xception融入到模型中去. 为了提高训练效率, 我们使用Xception导出特征权重, 再使用新的特征权重去做拟合.

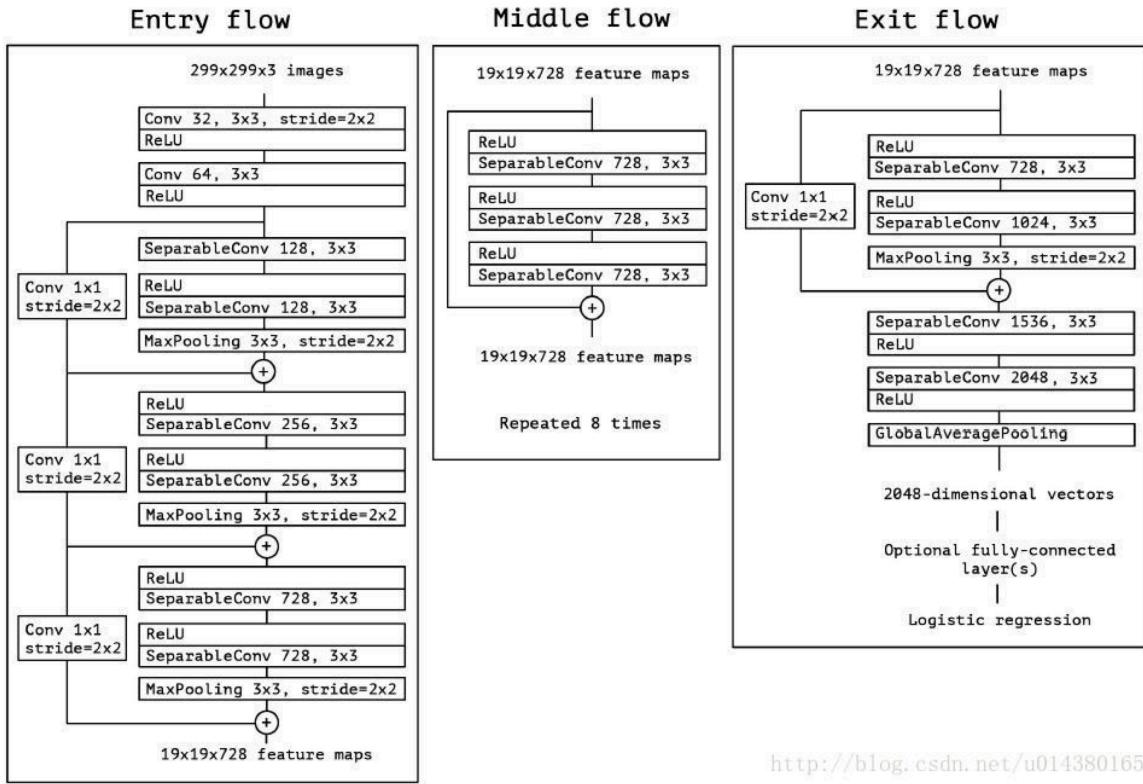
- 在训练完成后, 通过模型在验证集上的LogLoss分数表现, 选取最优的模型, 将此模型用于测试集的预测, 最终获取评分(需要将预测结果上传至kaggle), 作为最终得分.
- 在compile过程中, 使用了adadelta作为优化器, 此优化器针对Adagrad做了改进.
- 关于各种迁移学习模型之间的比较, 后续, 我们选择了准确率比较高, 并且数据集较小的Xception来解决这个分类问题:

模型	大小	Top-1 准确率	Top-5 准确率	参数数量	深度
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

基准模型

- 这里使用猫狗大战的Top 10%作为基准分数, 也就是LogLoss分值达到0.06127以下.
- 这里选取Xception作为基准模型, Xception是google继Inception后提出的对Inception v3的另一种改进, 主要是采用depthwise separable convolution来替换原来Inception v3中的卷积操作.
- Xception结构图:

Figure 5. The Xception architecture: the data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization [7] (not included in the diagram). All SeparableConvolution layers use a depth multiplier of 1 (no depth expansion).



III. 方法

数据预处理

- 关于异常值的处理, 这部分我们在前面已经提及到了, 在这里不再过多的介绍, 直接上代码, 关于函数与代码段的定义, 注释里面已经写的非常详细了:

```

# 获取ImageNetClasses.csv文件中的猫狗分类列表
def get_rightClass(fp):
    rightClass = []
    with open(fp, 'r') as r:
        reader = csv.reader(r)
        for line in reader:
            if (line[1] == '猫') or (line[1] == '狗'):
                rightClass.append(line[0])
    return(rightClass)

# 读取图片数据
def read_img(fp, img_size):
    img = image.load_img(fp, target_size = img_size)
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    return(x)

# 从目录中加载图片数据
def read_img_dir(flist, img_size):
    result = [ read_img(fp, img_size) for fp in tqdm(flist) ]
    return(result)

# 获取相关图片的预测结果
def get_predict(model, x, top = 10):
    pred = model.predict(x)
    return(decode_predictions(pred, top = top)[0])

# 获取所有图片预测结果
def get_predict_dir(model, X, top = 10):
    result = [ get_predict(model, x, top) for x in tqdm(X) ]
    return(result)

img_size = (299, 299, 3)
INClass_FP = 'ImageNetClasses/ImageNetClasses.csv'
flist = os.listdir(TRAIN_DIR)
flist = [ '{}/{}'.format(TRAIN_DIR, x) for x in flist ]
rightClass = get_rightClass(INClass_FP)
X = read_img_dir(flist, img_size)

# 注意，这里一定要把Xception模型的定义给提取出来，因为这一步需要一点时间
model = Xception(weights='imagenet')
pred = get_predict_dir(model, X, 50)

```

100% |██████████| 25000/25000 [02:34<00:00, 161.71it/s]
100% |██████████| 25000/25000 [06:32<00:00, 63.67it/s]

```

# 获取能match到猫狗类型的预测总数
def get_right_list(pred):
    right_list = []
    for item in pred:
        pred_list = [x[0] for x in item]
        pred_list = [1 for x in pred_list if x in rightClass]
        right_list.append(sum(pred_list))
    return(right_list)

# 获取异常数据的文件名
right_list = get_right_list(pred)
exp_flist = []
for index, value in enumerate(right_list):
    if value == 0:
        exp_flist.append(flist[index])

# 看下一共有多少张异常图片
print(len(exp_flist))

# 将图片输出看看
for fp in exp_flist:
    show_pic(fp)

```

- 在使用Xception导出特征权重前, 尝试直接用Model将Xception模型载入, 在后面加入了GAP与sigmoid输出层, 在未经冻结中间层的情况下, 直接进行训练, 最终的结果并未比直接使用CNN训练后的预测结果好. 这里, 我们直接使用Xception导出权重特征, 相当于使用Model加载Xception后, 冻结Xception中已经训练好的中间层, 直接将其结果作为我们训练模型的输入. 但是如果每次都去冻结Xception, 这样会造成后面训练的时间过长, 对计算能力不强的机器不大友好. 顾我们这里, 直接将其通过Xception, 导出特征权重, 并传入我们的模型做分类.
- 为了方便导入数据, 这里将重新定义几个目录(将train下的数据分类并拷贝至transfer/train/{cat,dog}, test拷贝至DataSet/transfer/test/pic/目录下), 注意, 这里的训练集是经过异常值剔除的.

```

# 将train下的数据分类并拷贝至transfer/train/{cat,dog}, test拷贝至DataSet/transfer/test/pic/目录下
!mkdir DataSet/transfer/train/cat -p
!mkdir DataSet/transfer/train/dog
!mkdir DataSet/transfer/test/pic -p
!find DataSet/train -name 'cat.*' -exec cp -rf {} DataSet/transfer/train/cat/ \;
!find DataSet/train -name 'dog.*' -exec cp -rf {} DataSet/transfer/train/dog/ \;
!find DataSet/test -name '*.jpg' -exec cp -rf {} DataSet/transfer/test/pic/ \;

```

- 使用flow_from_directory()函数导入图片与分类数据, 由于输入图片大小分布不一, 这里我们将对输入的图片大小进行调整. 在Xception中(299, 299)大小的图片数据作为输入, 模型在此时表现最好. 此函数会对输入数据进行归一化处理, 经过处理后, 数据会在(0, 1)之间. 在这一步, 我们还暂时不需要将训练集/测试集的顺序打乱:

```
# run load data, change size to Xception default (299,299)
img_size = (299, 299)
gen = ImageDataGenerator()
X_train_gen = gen.flow_from_directory(TRANSFER_TRAIN_DIR, img_size, shuffle = False,
                                         batch_size = 16)
X_test_gen = gen.flow_from_directory(TRANSFER_TEST_DIR, img_size, shuffle = False,
                                         batch_size = 16, classes = None)

Found 25000 images belonging to 2 classes.
Found 12500 images belonging to 1 classes.
```

- 构建Xception权重导出模型, 这里我们在后面添加了一个GAP为我们后面的全连接层做准备. 在这里添加GAP是因为需要防止受其他干扰项干扰, 特别是当图中其他干扰项占比特别大的时候. 在使用GAP之前, 还对于GAP(GlobalAveragePool)与GMP(GlobalMaxPool)做了对比. 使用GMP的时候, 当这里干扰项非常大, 例如猫与人在图中的占比, 当人的占比特别大时, 使用GMP会输出人; 但是如若使用GAP, 则会输出我们想要的猫:

```
input_tensor = Input((img_size[0], img_size[1], 3))
input_tensor = Lambda(xception.preprocess_input)(input_tensor)
Xception_base = Xception(input_tensor = input_tensor,
                           weights = 'imagenet', include_top = False)
Xception_model = Model(Xception_base.input, GlobalAveragePooling2D()(Xception_base.output))
Xception_model.summary()
```

- 利用Xception训练特征向量, 这一步其实就是为后面的预测节约时间, 提高模型学习效率. 其实跟前面的异常值剔除的思路差不多, 就是将经过预处理的图片特征走Xception过一遍. 后面不需要对Xception中已经训练好的模型进行修改, 所以直接导出特征向量, 对于提高后续程序的执行效率还是很有必要的:

```
X_train = Xception_model.predict_generator(X_train_gen, verbose=1)
X_test = Xception_model.predict_generator(X_test_gen, verbose=1)
```

```
1563/1563 [=====] - 181s 116ms/step
782/782 [=====] - 91s 116ms/step
```

- 导出特征权重, 就是将上一步的结果保存到文件中, 方便后面调用:

```

with h5py.File('saved_models/weights.Xception.hdf5') as fp:
    fp.create_dataset('train', data = X_train)
    fp.create_dataset('test', data = X_test)
    fp.create_dataset('label', data = X_train_gen.classes)

```

执行过程

- 导入我们刚刚训练好的特征向量. 注意, 这里一定要先对训练集中的数据与标签打乱, 因为我们在后面训练的时候, 会按照比例将其划分为训练集与验证集, 虽然model.fit()函数中自带了shuffle, 但是深究一下, 会发现它是先split再去shuffle的, 这样就会造成验证集中样本分布不均匀, 导致预测结果不准确. 所以这里我用了sklearn中的shuffle, 直接手动做了shuffle:

```

X_train = []
X_test = []
with h5py.File('saved_models/weights.Xception.hdf5', 'r') as fp:
    X_train.append(np.array(fp['train']))
    X_test.append(np.array(fp['test']))
    y_train = np.array(fp['label'])

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

```

- 使用Xception训练好的特征向量构建模型, 这里添加了Dropout和一个全连接层作为输出层. 一般情况下推荐Dropout使用0.5, 由于这是一个二分类问题, 所以使用Sigmoid作为最后全连接层的处理函数:

```

input_tensor = Input(X_train.shape[1:])
Xception_model = Model(input_tensor, Dropout(0.5)(input_tensor))
Xception_model = Model(Xception_model.input, Dense(1, activation = 'sigmoid')(Xception_model.output))
Xception_model.summary()

```

Layer (type)	Output Shape	Param #
<hr/>		
input_3 (InputLayer)	(None, 2048)	0
dropout_1 (Dropout)	(None, 2048)	0
dense_4 (Dense)	(None, 1)	2049
<hr/>		
Total params: 2,049		
Trainable params: 2,049		
Non-trainable params: 0		

- 编译模型, 这里使用了adadelta优化器. adadelta通过设置窗口, 只使用部分时间的梯度积累, 解决了adagrad中的过早结束优化过程的问题:

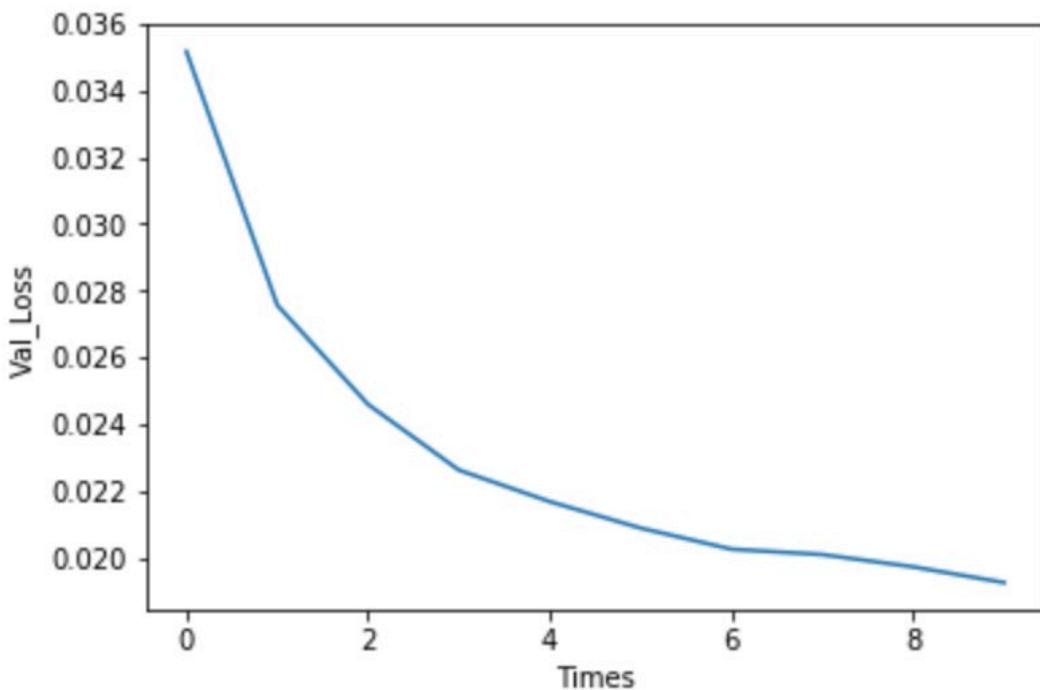
```
Xception_model.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])
```

- 使用Xception训练模型, fit时, 将原训练集中的数据以7:3的比例划分为训练集与验证集, 并保存最好的训练结果:

```
epochs = 10
batch_size = 128
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Xception.hdf5',
                                verbose=1, save_best_only=True)
Xception_model.fit(X_train, y_train, validation_split = 0.3,
                    epochs = epochs, batch_size = batch_size, verbose=1,
                    callbacks=[checkpointer])
```

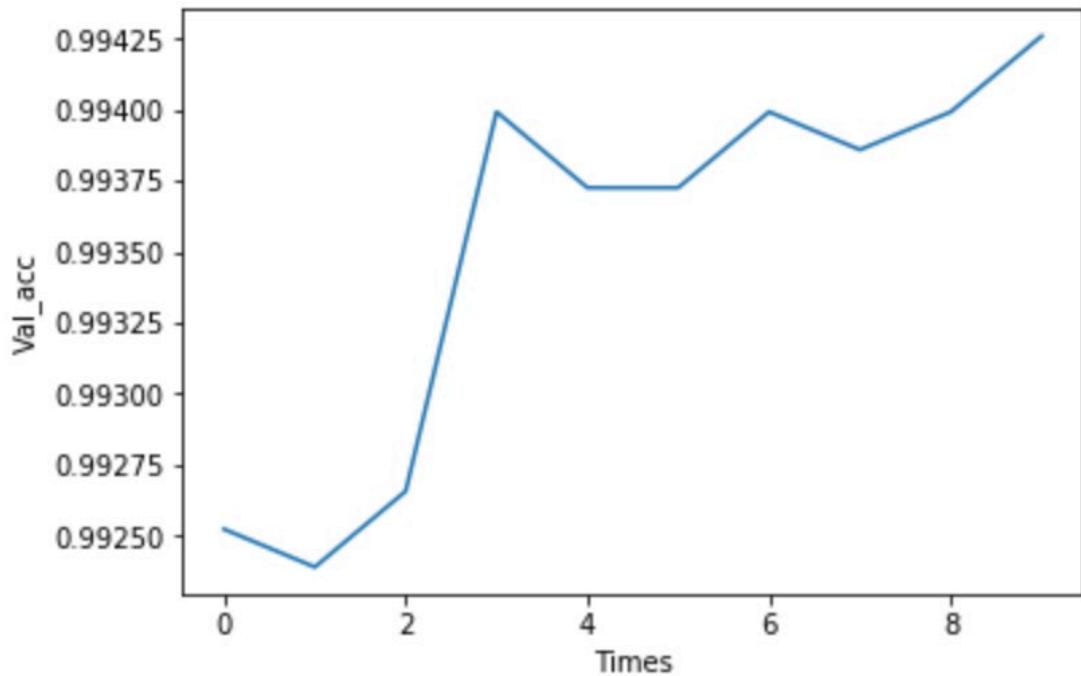
- 在这里, 我们尝试了两种优化器, 并对其的学习过程进行可视化: adagrad优化器模型学习曲线:

```
# val_loss学习曲线
plt.plot(hist_Xception.history['val_loss'])
plt.xlabel('Times')
plt.ylabel('Val_Loss')
plt.show()
```



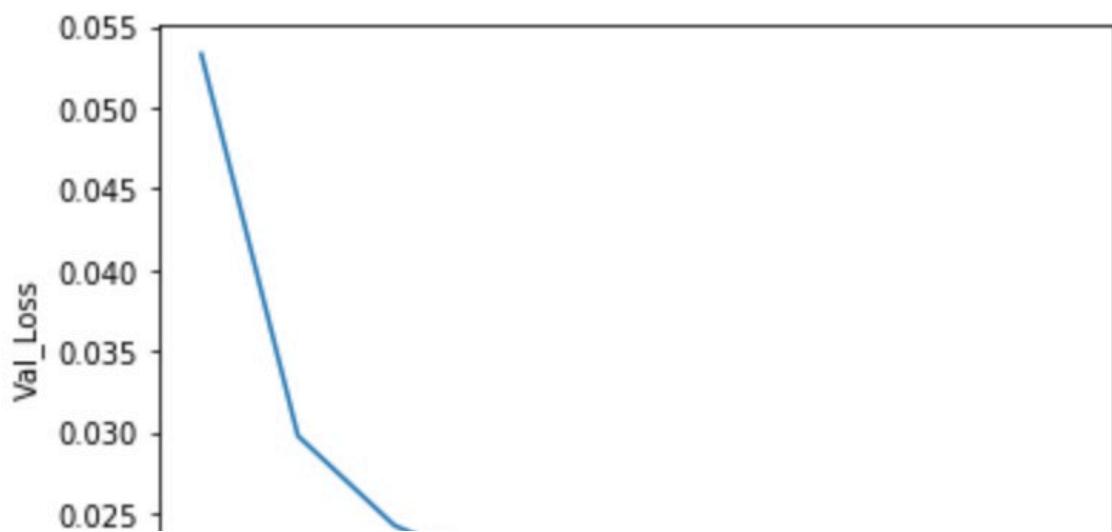
```
# val_loss学习曲线
plt.plot(hist_Xception.history['val_acc'])
plt.xlabel('Times')
```

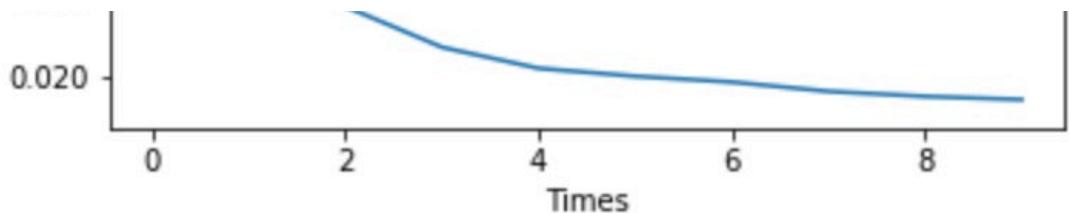
```
plt.ylabel('Val_acc')  
plt.show()
```



adadelta优化器模型学习曲线:

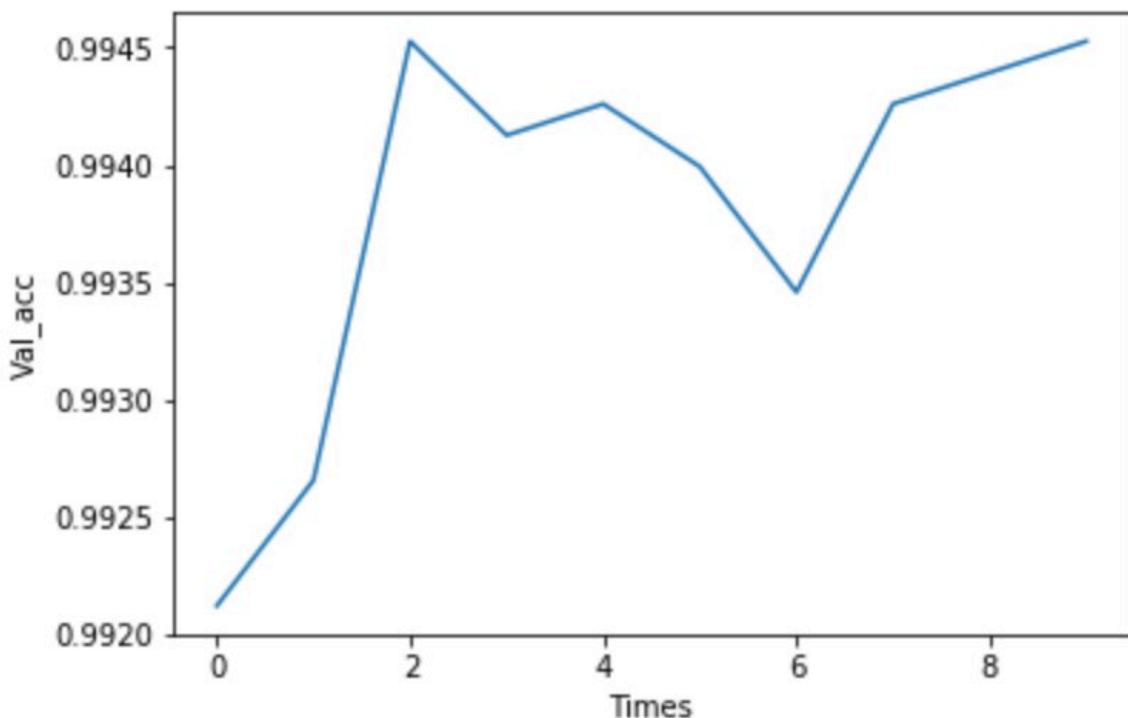
```
# val_loss 学习曲线  
plt.plot(hist_Xception.history['val_loss'])  
plt.xlabel('Times')  
plt.ylabel('Val_Loss')  
plt.show()
```





```
# val_loss学习曲线
```

```
plt.plot(hist_Xception.history[ 'val_acc' ])
plt.xlabel( 'Times' )
plt.ylabel( 'Val_acc' )
plt.show()
```



- 可以发现使用adadelta优化器进行调优比adagrad优化器进行调优的收敛会更快, 学习的速率会更加的快. 所以最终我们选择使用adadelta优化器进行调优.

完善

- 在这里踩过一个坑, 必须先做shuffle再去fit, 不能直接使用fit中的shuffle. 因

为fit中的shuffle是先按照比例分训练集与数据集,再去shuffle,这样就会导致训练集与验证集中的样本分布不均匀.

IV. 结果

模型的评价与验证

- 训练结果
- 在经过多次训练测试后,发现基于Xception的模型能够稳定的对数据进行学习与拟合,准确率也逐步提升.

```
Epoch 00002: val_loss improved from 0.05176 to 0.02914, saving model to saved_models/weights.best.Xception.hdf5
Epoch 3/10
17500/17500 [=====] - 1s 32us/step - loss: 0.0299 - acc: 0.9927 - val_loss: 0.0229 - val_ac
c: 0.9936

Epoch 00003: val_loss improved from 0.02914 to 0.02293, saving model to saved_models/weights.best.Xception.hdf5
Epoch 4/10
17500/17500 [=====] - 1s 31us/step - loss: 0.0264 - acc: 0.9928 - val_loss: 0.0209 - val_ac
c: 0.9936

Epoch 00004: val_loss improved from 0.02293 to 0.02090, saving model to saved_models/weights.best.Xception.hdf5
Epoch 5/10
17500/17500 [=====] - 1s 31us/step - loss: 0.0241 - acc: 0.9930 - val_loss: 0.0198 - val_ac
c: 0.9935

Epoch 00005: val_loss improved from 0.02090 to 0.01984, saving model to saved_models/weights.best.Xception.hdf5
Epoch 6/10
17500/17500 [=====] - 1s 32us/step - loss: 0.0214 - acc: 0.9939 - val_loss: 0.0188 - val_ac
c: 0.9941

Epoch 00006: val_loss improved from 0.01984 to 0.01880, saving model to saved_models/weights.best.Xception.hdf5
Epoch 7/10
17500/17500 [=====] - 1s 32us/step - loss: 0.0213 - acc: 0.9937 - val_loss: 0.0188 - val_ac
c: 0.9937

Epoch 00007: val_loss did not improve from 0.01880
Epoch 8/10
17500/17500 [=====] - 1s 32us/step - loss: 0.0202 - acc: 0.9938 - val_loss: 0.0180 - val_ac
c: 0.9939

Epoch 00008: val_loss improved from 0.01880 to 0.01802, saving model to saved_models/weights.best.Xception.hdf5
Epoch 9/10
17500/17500 [=====] - 1s 32us/step - loss: 0.0200 - acc: 0.9942 - val_loss: 0.0177 - val_ac
c: 0.9945

Epoch 00009: val_loss improved from 0.01802 to 0.01771, saving model to saved_models/weights.best.Xception.hdf5
Epoch 10/10
17500/17500 [=====] - 1s 32us/step - loss: 0.0201 - acc: 0.9940 - val_loss: 0.0178 - val_ac
c: 0.9937

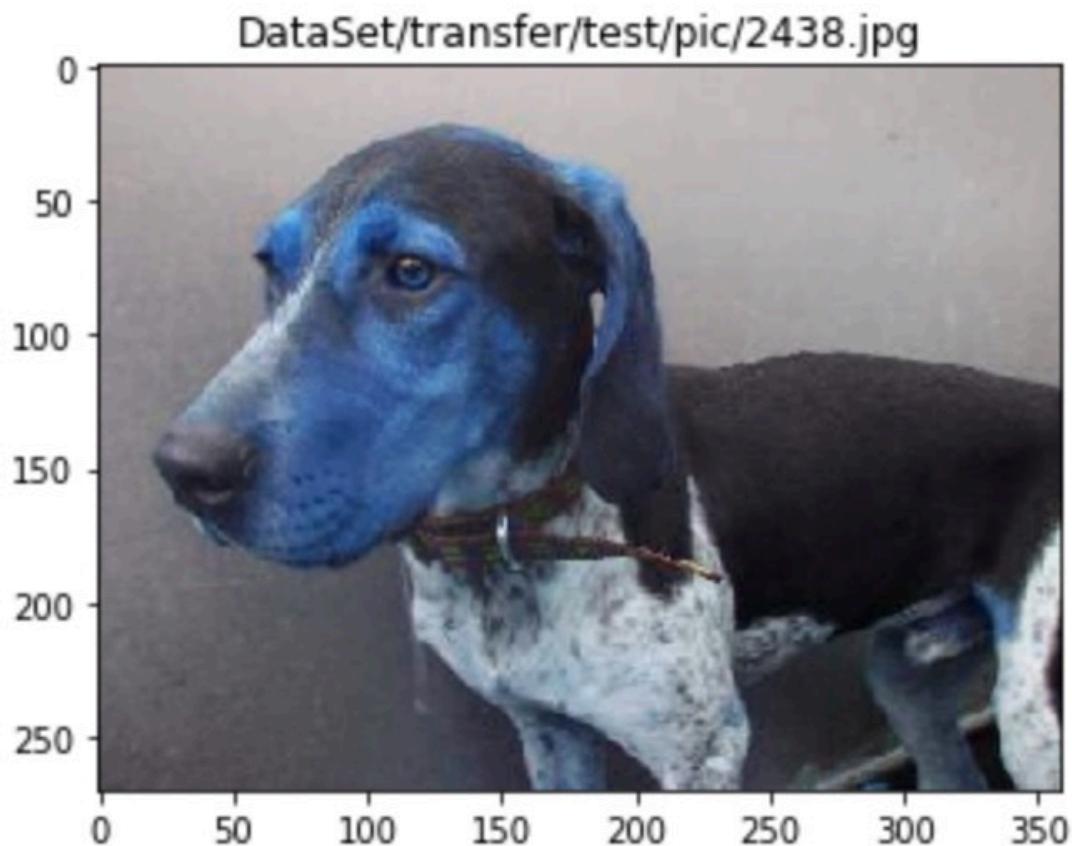
Epoch 00010: val_loss did not improve from 0.01771
Out[49]: <keras.callbacks.History at 0x7f3c539c89b0>
```

- 在训练完成后,我们通过代码随机展现了一些预测的结果:



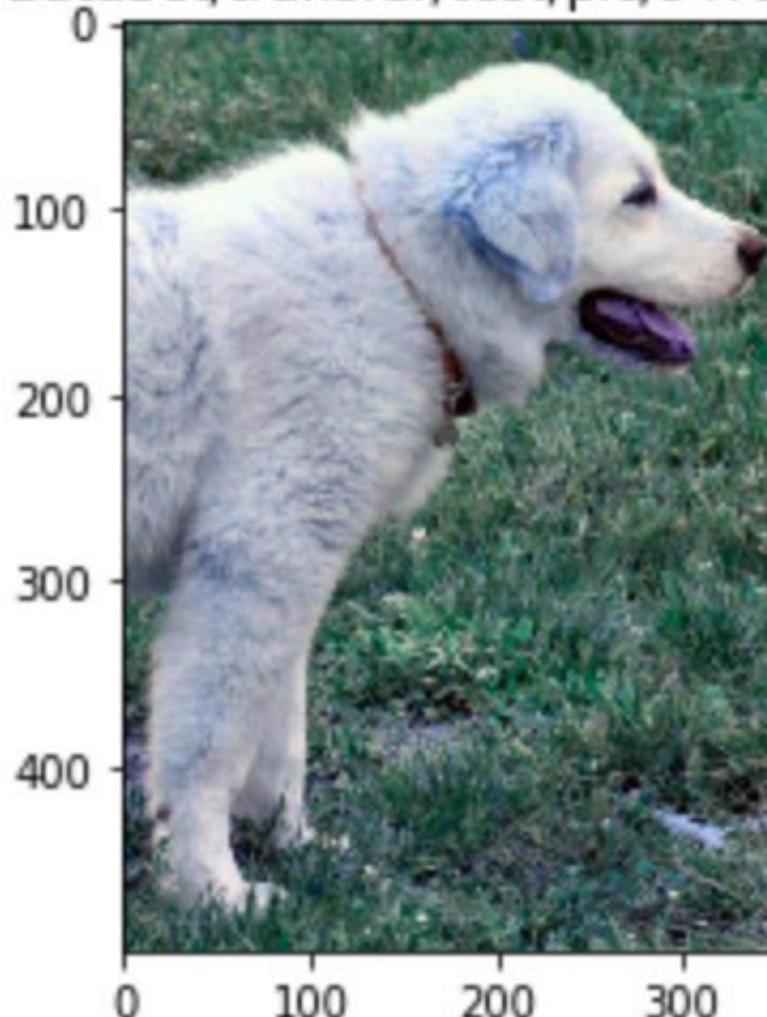


[0 . 005]



[0 . 995]

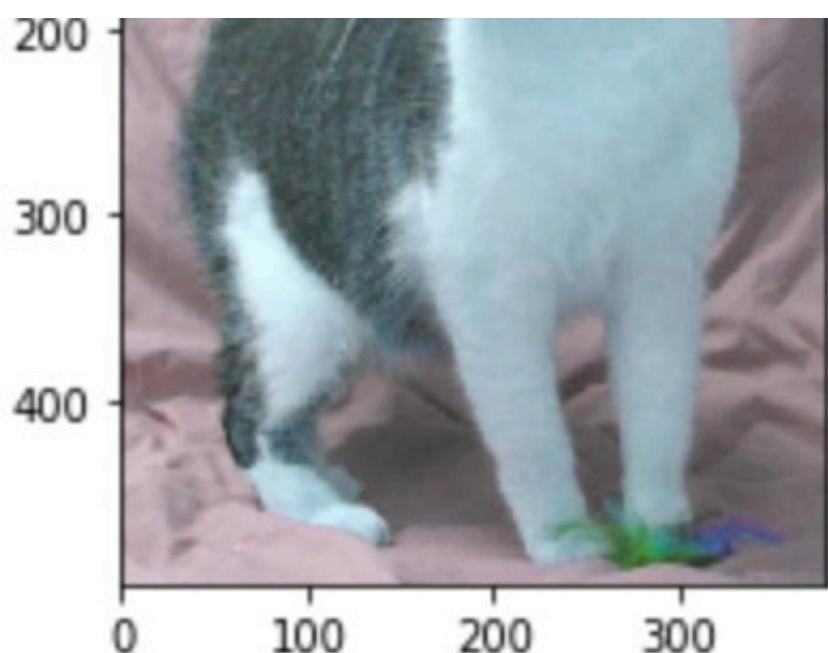
DataSet/transfer/test/pic/9472.jpg



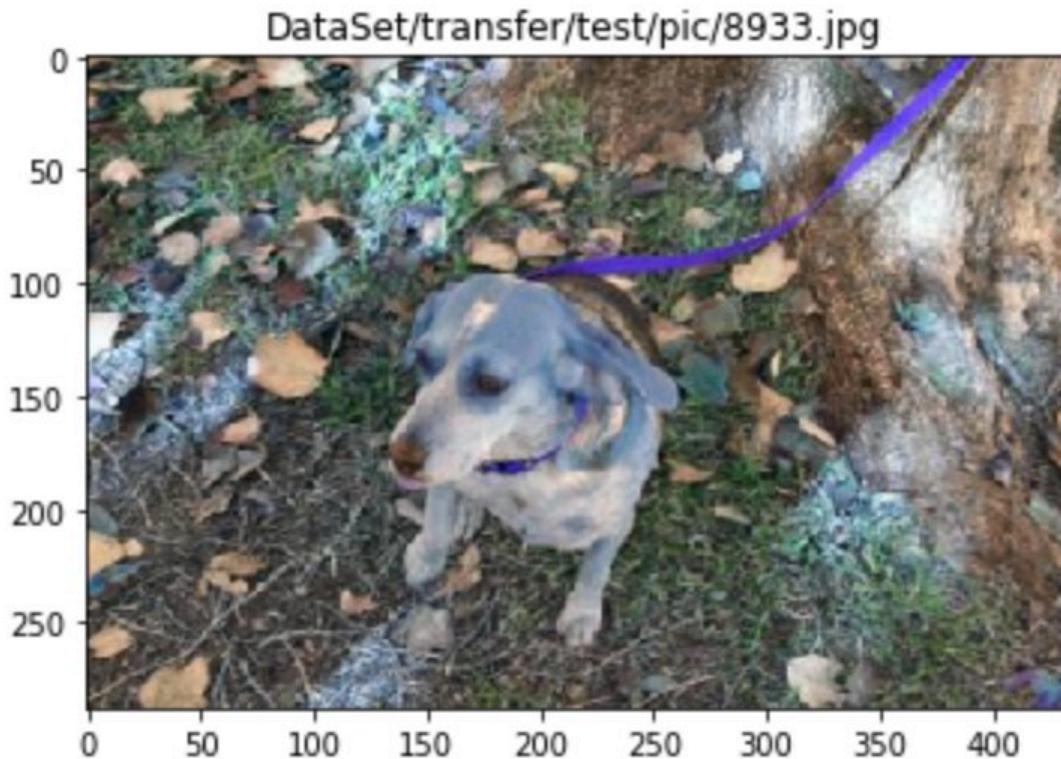
[0.995]

DataSet/transfer/test/pic/3958.jpg





[0.005]



[0.995]

- 当结果接近于1的时候, 为确定为狗的概率, 概率越发趋近于零的时候, 为确定为猫的概率. 综合上述, 可以发现, 我们模型对于识别猫狗的准确度还是很高的, 目前抽查的五个样例的准确率为100%. 模型还是很健壮的.
- 测试集Kaggle得分, 进入top 10%, LogLoss分数 < 0.06127, 达到了我们基准模型的阈值:

0 submissions for Kyle Chen		Sort by	Most recent
All	Successful	Selected	
Submission and Description	Public Score	Use for Final Score	
result.csv 4 hours ago by Kyle Chen 1st commit	0.04103	<input type="checkbox"/>	
No more submissions to show			

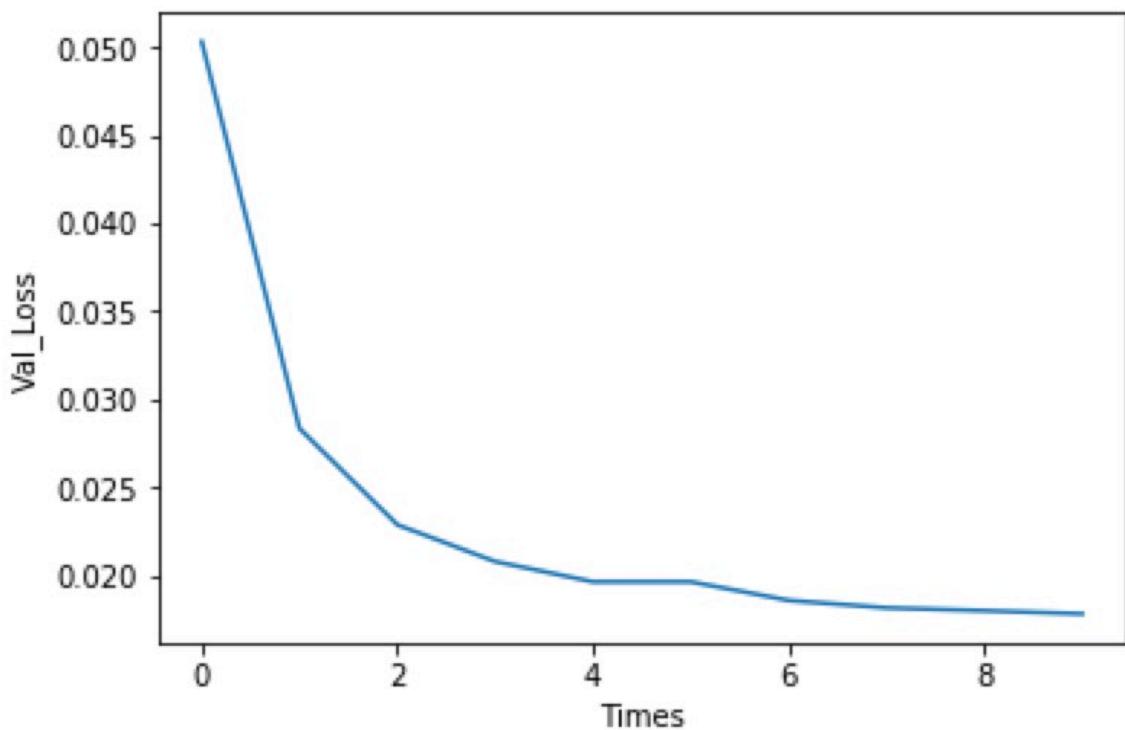
合理性分析

- 可以发现, 模型是有在慢慢收敛并且能很好的拟合数据, 最后得到了很不错的LogLoss分数, 最终也达到了top 10的目标.

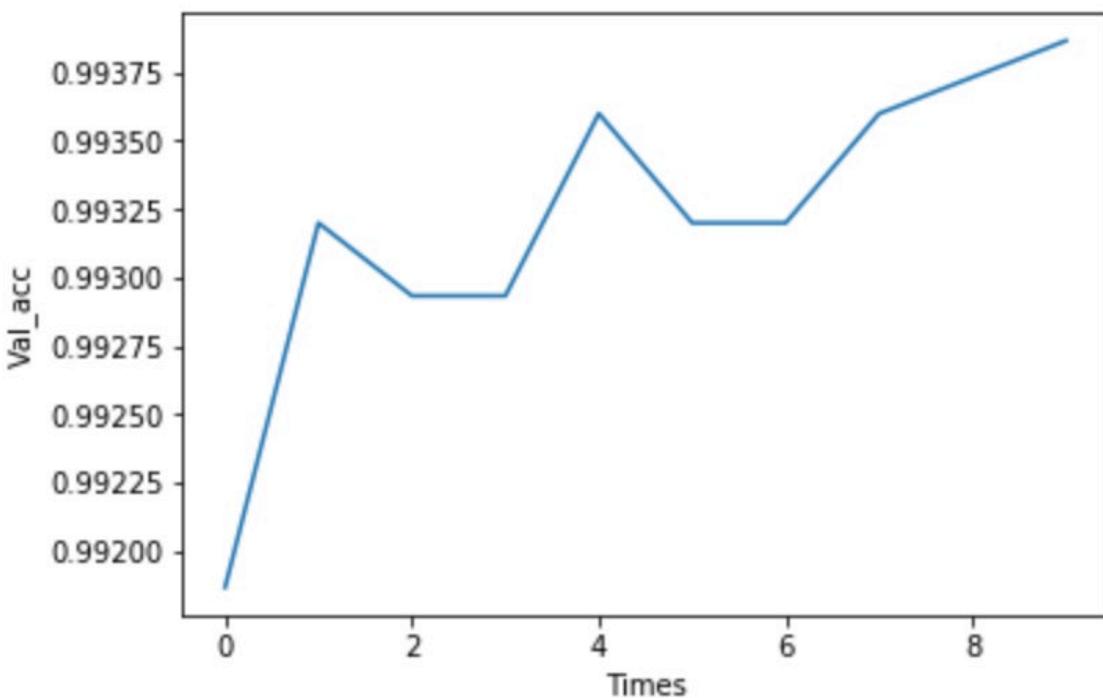
V. 项目结论

结果可视化

- 使用adadelta优化器时的val_loss学习曲线



- 使用adadelta优化器时的val_acc学习曲线



对项目的思考

- 对于此项目, 能跑进top 10%还仅仅是一个开端. 迁移学习有很多很棒的模型, 还可以研究. 对于新手来说, 这个准确率, 还不算差, 随着经验日积月累, 对调优这块可能还会有更多更深刻的认识.
- 对于机器学习, 目前比较难的地方应该还是在于落地. 在结束此次课程后, 将主要研究如何将机器学习/深度学习融入到具体的自动化场景中.
- 在此项目中, 对于新手来说, 比较困难的地方在于数据清洗与优化器的选择. 对于数据清洗部分, 异常数据的判断, 容易让人陷入死胡同, 在得到一个可靠的模型之前, 可能无法去针对猫狗去做一个判断, 但是换一个思路, 我们直接从预训练模型入手, 仿佛思路就清晰了许多. 对于优化器的选择, 需要不断的尝试与实验, 特别是对于经验不够丰富的新手来说, 在补充阅读一定文档与实验的情况下, 可能会更加需要老司机的指点.

需要作出的改进

- 可以考虑融合多个模型来提取特征权重, 最终实现更好的效果.
- 在可用性方面, 可以研究下如何在手机或者平板设备上投产.

引用

[1] Dogs vs. Cats Redux: Kernels Edition Rules:

<https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/rule>

[2] François Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, 4 Apr 2017: <https://arxiv.org/abs/1610.02357>

[3] Sergey Ioffe, Christian Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2 Mar 2015:

<https://arxiv.org/abs/1502.03167>

[4] Nitish Srivastava, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 11/13 2014:

<http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>