机器学习纳米学位

毕业项目: 猫狗大战
作者: Kyle Chen
日期: 20180510
版本: 20180510v1

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, 可以是任何猫或狗, 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.

问题陈述

- 解决办法所针对的具体问题
- 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.

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.

- 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%, 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;
- 在代码实现中, 可以使用keras.preprocessing中的image库加载RGB图像.
- 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.

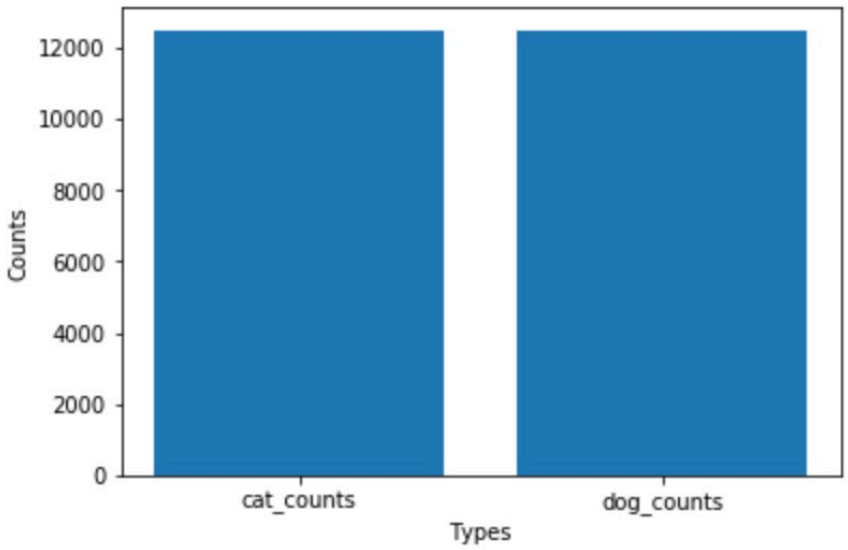
- 在获得训练集, 验证集之前, 需要将train/目录下的文件打乱, 然后按照一定的比例将其划分到训练集, 验证集中.
- 大致过了一遍数据集里面的数据之后, 有几个异常值需要删除的.

```
!rm -rf DataSet/train/cat.7377.jpg DataSet/train/cat.4085.jpg
```

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 == 0 ])
plt.bar(['cat_counts', 'dog_counts'], [cat_counts, dog_counts])
plt.xlabel('Types')
plt.ylabel('Counts')
plt.show()
```



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.
- 在使用迁移学习之前, 我们需要先尝试自己搭建CNN来处理这个问题(具体模型在基准模型中会详细探讨).
- 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, 去做拟合; 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.

基准模型

- 在使用Xception训练前, 我们还需要将其与CNN做个比较, 在其不使用迁移学

习的模型时, 是否还能有好的表现.

- 这里我们可以使用relu作为Hidden_nodes的激活函数, sigmoid作为输出函数, 在中间添加Conv2D, MaxPooling2D. 如若需要, 可以加入BN防止过拟合. 当然, 在没有实际代码经过多次测试的基础下, 暂且不能确定最优的模型框架.
- 拟合数据时需要加上validation_split = 0.3, shuffle = True两个参数.
- 这里使用LogLoss来为模型评估分数.
- 关于CNN部分可以参照可以下框架搭建:

```
cnn_model = Sequential()
shape_input = (len(data[0]), len(data[0][0]), len(data[0][0][0]))
cnn_model.add(Conv2D(filters=16, kernel_size=2, input_shape=shape_input))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=2, padding='valid'))
cnn_model.add(Dense(133, activation='relu'))
cnn_model.add(Conv2D(filters=32, kernel_size=2))
cnn_model.add(MaxPooling2D(pool_size=2, padding='valid'))
cnn_model.add(Dense(133, activation='relu'))
cnn_model.add(Conv2D(filters=64, kernel_size=2))
cnn_model.add(MaxPooling2D(pool_size=2, padding='valid'))
cnn_model.add(GlobalAveragePooling2D(dim_ordering='default'))
cnn_model.add(Dense(1, activation='sigmoid'))
cnn_model.summary()
```

- 以上模型在猫狗大战中的训练过程如下:

```
Train on 17500 samples, validate on 7500 samples
Epoch 1/5
17500/17500 [==============================] - 1908s 109ms/step - loss: 0.6724 - acc: 0.5890 - val_loss: 0.6701 - val
_acc: 0.5653

Epoch 00001: val_loss improved from inf to 0.67014, saving model to saved_models/weights.best.cnn.hdf5
Epoch 2/5
17500/17500 [==============================] - 1749s 100ms/step - loss: 0.6535 - acc: 0.6191 - val_loss: 0.6427 - val
_acc: 0.6325

Epoch 00002: val_loss improved from 0.67014 to 0.64273, saving model to saved_models/weights.best.cnn.hdf5
Epoch 3/5
17500/17500 [==============================] - 1754s 100ms/step - loss: 0.6445 - acc: 0.6271 - val_loss: 0.6435 - val
_acc: 0.6259

Epoch 00003: val_loss did not improve
Epoch 4/5
17500/17500 [==============================] - 1775s 101ms/step - loss: 0.6400 - acc: 0.6304 - val_loss: 0.6318 - val
_acc: 0.6348

Epoch 00004: val_loss improved from 0.64273 to 0.63181, saving model to saved_models/weights.best.cnn.hdf5
Epoch 5/5
17500/17500 [==============================] - 1796s 103ms/step - loss: 0.6375 - acc: 0.6335 - val_loss: 0.6622 - val
_acc: 0.6156

Epoch 00005: val_loss did not improve

<keras.callbacks.History at 0x49f859470>
```

- 可以看到还有很大的优化空间, 我们还可以通过迁移学习来慢慢接近目标, 进入前10%, 也就是LogLoss分数达到0.06127以下.
- 在后续章节中, 我们将尝试使用Xception来解决这个分类问题

III. 方法

数据预处理

- 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).

```
# 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:

```
# 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:

```
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:

```
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:

```
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

```
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

```
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 #
=================================================================
input_3 (InputLayer)         (None, 2048)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 2049
=================================================================
Total params: 2,049
Trainable params: 2,049
Non-trainable params: 0
_________________________________________________________________
```

- 编译模型

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

- 使用Xception训练模型

```
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])
```

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.

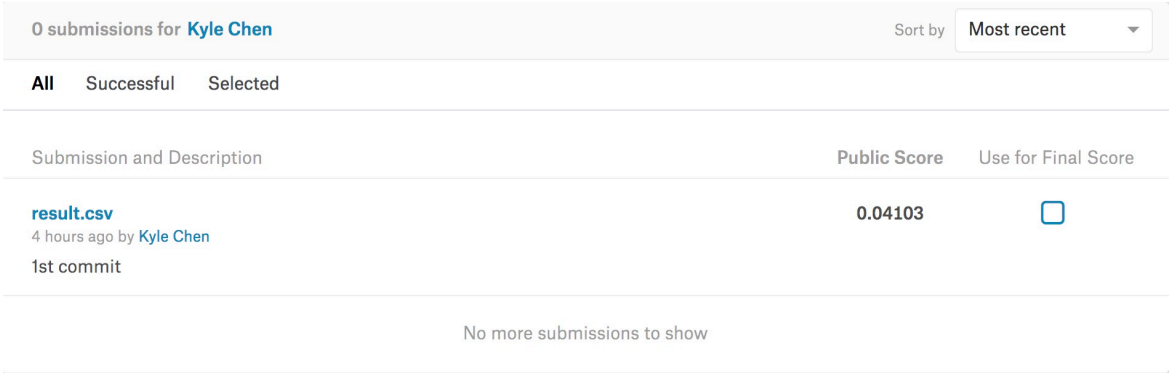
IV. 结果

模型的评价与验证

- 训练结果

```
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>
```

- 测试集Kaggle得分, 进入top 10%, LogLoss分数 < 0.06127:



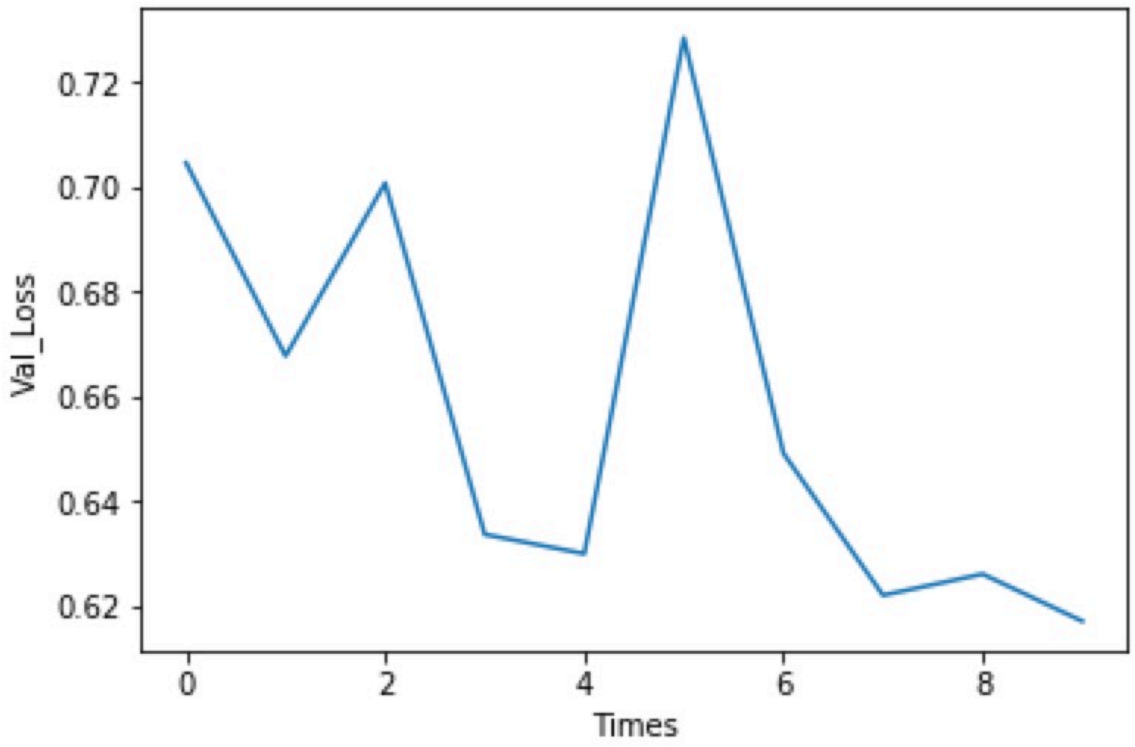
合理性分析

- 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.

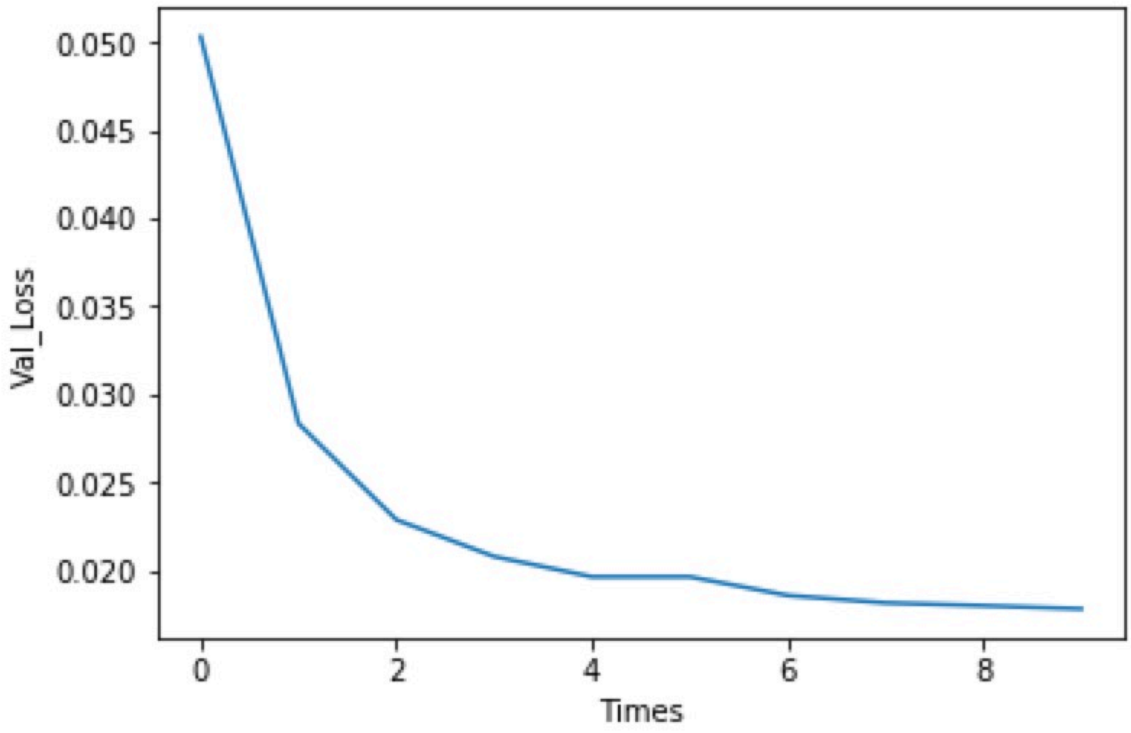
V. 项目结论

结果可视化

- 使用CNN时的val_loss学习曲线



- 使用Xception时的val_loss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.

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.
- 利用自动化工具识别异常数据.

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