beluga

Dog Breed - Pretrained keras models(LB 0.3)

196 voters

last run 4 months ago · Python notebook · 21511 views
using data from multiple data sources · Public

Tags deep learning cnn image processing animals multiple data sources

Notebook

Transfer learning with pretrained Keras models

Although Kernel resources were increased recently we still can not train useful CNNs without GPU. The original ImageNet set has quite a few different dog classes so we can reuse CNNs with pretrained ImageNet weights. Fortunately prediction is much faster (<1s/image) making it possible to run meaningful experiments with Kaggle Kernels.

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
from os import listdir, makedirs
from os.path import join, exists, expanduser
from tqdm import tqdm
from sklearn.metrics import log_loss, accuracy_score
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.applications.resnet50 import ResNet50
from keras.applications import xception
from keras.applications import inception_v3
from keras.applications.vgg16 import preprocess_input, decode_predictions
from sklearn.linear_model import LogisticRegression
```

```
Using TensorFlow backend.
/opt/conda/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: compiletime version 3.5
of module 'tensorflow.python.framework.fast_tensor_util' does not match runtime version 3.6
  return f(*args, **kwds)
```

In [2]:
```
start = dt.datetime.now()
```

Use Keras Pretrained Models dataset

Kernels can't use network connection to download pretrained keras model weights. This dataset helps you to apply your favorite pretrained model in the Kaggle Kernel environment. You can find more details here (https://www.kaggle.com/gaborfodor/keras-pretrained-models).

We have to copy the pretrained models to the cache directory (~/.keras/models) where keras is looking for them.

In [3]:
```
!ls ../input/keras-pretrained-models/
```

```
Kuszma.JPG
imagenet_class_index.json
inception_resnet_v2_weights_tf_dim_ordering_tf_kernels.h5
inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
inception_v3_weights_tf_dim_ordering_tf_kernels.h5
inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
```

```
resnet50_weights_tf_dim_ordering_tf_kernels.h5
resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
xception_weights_tf_dim_ordering_tf_kernels.h5
xception_weights_tf_dim_ordering_tf_kernels_notop.h5
```

In [4]:

```
cache_dir = expanduser(join('~', '.keras'))
if not exists(cache_dir):
    makedirs(cache_dir)
models_dir = join(cache_dir, 'models')
if not exists(models_dir):
    makedirs(models_dir)
```

In [5]:

In [6]:

```
!ls ~/.keras/models
```

```
imagenet_class_index.json
inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
resnet50_weights_tf_dim_ordering_tf_kernels.h5
resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
xception_weights_tf_dim_ordering_tf_kernels_notop.h5
```

In [7]:

```
!ls ../input/dog-breed-identification
```

```
labels.csv  sample_submission.csv  test  train
```

Use top 16 classes

Using all the images would take more than the 1 hour kernel limit. Let's focus on the most frequent 16 breeds.

In [8]:

```
INPUT_SIZE = 224
NUM_CLASSES = 16
SEED = 1987
data_dir = '../input/dog-breed-identification'
labels = pd.read_csv(join(data_dir, 'labels.csv'))
sample_submission = pd.read_csv(join(data_dir, 'sample_submission.csv'))
print(len(listdir(join(data_dir, 'train'))), len(labels))
print(len(listdir(join(data_dir, 'test'))), len(sample_submission))
```

```
10222 10222
```

```
10222 10222
10357 10357
```

In [9]:

```
selected_breed_list = list(labels.groupby('breed').count().sort_values(by='id', ascending=Fals
e).head(NUM_CLASSES).index)
labels = labels[labels['breed'].isin(selected_breed_list)]
labels['target'] = 1
labels['rank'] = labels.groupby('breed').rank()['id']
labels_pivot = labels.pivot('id', 'breed', 'target').reset_index().fillna(0)
np.random.seed(seed=SEED)
rnd = np.random.random(len(labels))
train_idx = rnd < 0.8
valid_idx = rnd >= 0.8
y_train = labels_pivot[selected_breed_list].values
ytr = y_train[train_idx]
yv = y_train[valid_idx]
```

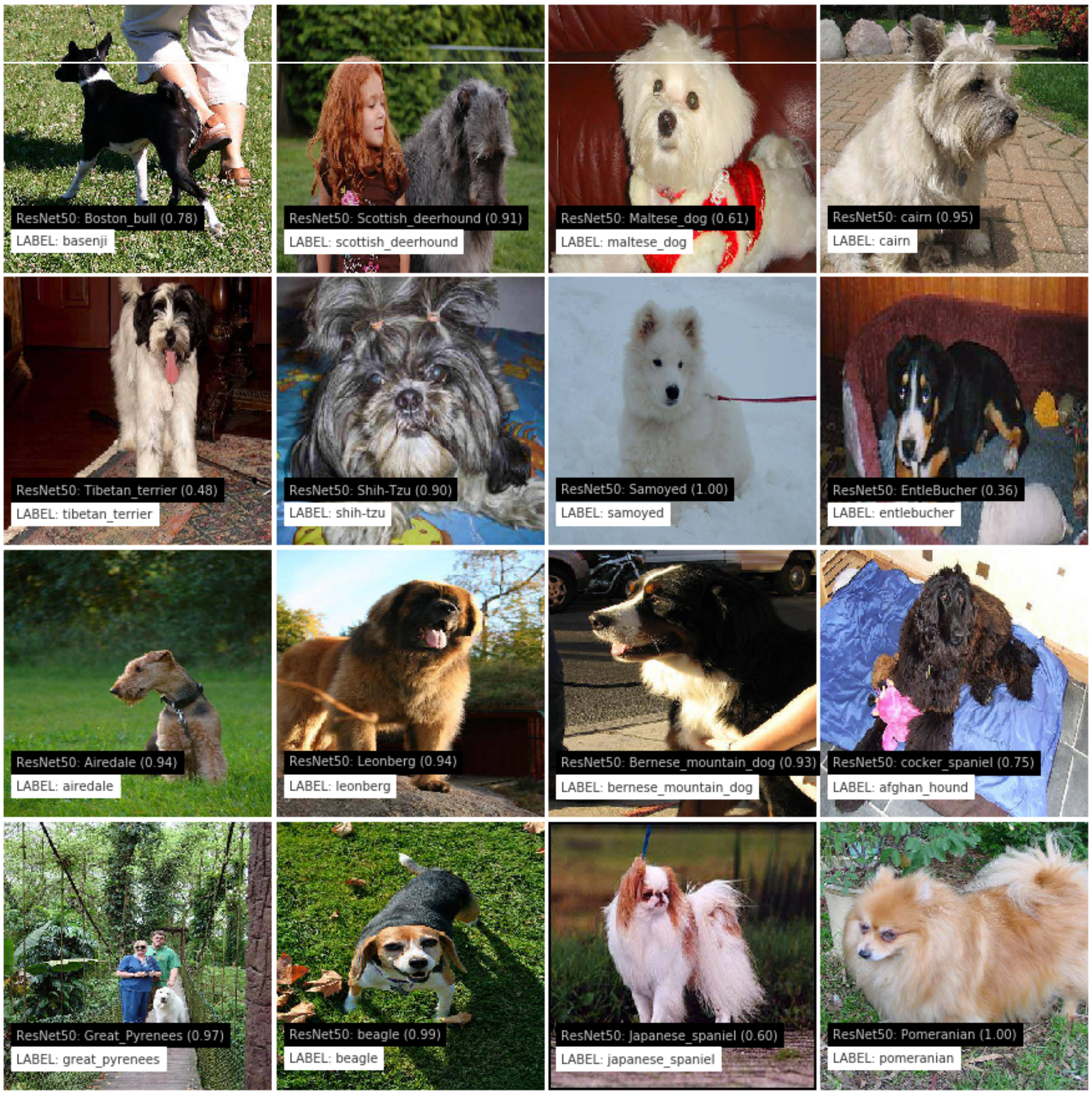
In [10]:

```
def read_img(img_id, train_or_test, size):
    """Read and resize image.
    # Arguments
        img_id: string
        train_or_test: string 'train' or 'test'.
        size: resize the original image.
    # Returns
        Image as numpy array.
    """
    img = image.load_img(join(data_dir, train_or_test, '%s.jpg' % img_id), target_size=size)
    img = image.img_to_array(img)
    return img
```

ResNet50 class predictions for example images

In [11]:

```
model = ResNet50(weights='imagenet')
j = int(np.sqrt(NUM_CLASSES))
i = int(np.ceil(1. * NUM_CLASSES / j))
fig = plt.figure(1, figsize=(16, 16))
grid = ImageGrid(fig, 111, nrows_ncols=(i, j), axes_pad=0.05)
for i, (img_id, breed) in enumerate(labels.loc[labels['rank'] == 1, ['id', 'breed']].values):
    ax = grid[i]
    img = read_img(img_id, 'train', (224, 224))
    ax.imshow(img / 255.)
    x = preprocess_input(np.expand_dims(img.copy(), axis=0))
    preds = model.predict(x)
    _, imagenet_class_name, prob = decode_predictions(preds, top=1)[0][0]
    ax.text(10, 180, 'ResNet50: %s (%.2f)' % (imagenet_class_name , prob), color='w', backgrou
ndcolor='k', alpha=0.8)
    ax.text(10, 200, 'LABEL: %s' % breed, color='k', backgroundcolor='w', alpha=0.8)
    ax.axis('off')
plt.show()
```



ResNet50: Boston_bull (0.78)
LABEL: basenji
ResNet50: Scottish_deerhound (0.91)
LABEL: scottish_deerhound
ResNet50: Maltese_dog (0.61)
LABEL: maltese_dog
ResNet50: cairn (0.95)
LABEL: cairn
ResNet50: Tibetan_terrier (0.48)
LABEL: tibetan_terrier
ResNet50: Shih-Tzu (0.90)
LABEL: shih-tzu
ResNet50: Samoyed (1.00)
LABEL: samoyed
ResNet50: EntleBucher (0.36)
LABEL: entlebucher
ResNet50: Airedale (0.94)
LABEL: airedale
ResNet50: Leonberg (0.94)
LABEL: leonberg
ResNet50: Bernese_mountain_dog (0.93)
LABEL: bernese_mountain_dog
ResNet50: cocker_spaniel (0.75)
LABEL: afghan_hound
ResNet50: Great_Pyrenees (0.97)
LABEL: great_pyrenees
ResNet50: beagle (0.99)
LABEL: beagle
ResNet50: Japanese_spaniel (0.60)
LABEL: japanese_spaniel
ResNet50: Pomeranian (1.00)
LABEL: pomeranian

Preprocessing and prediction seems to be working. 75% accuracy on these 16 images.

Extract VGG16 bottleneck features

In [12]:
```
INPUT_SIZE = 224
POOLING = 'avg'
x_train = np.zeros((len(labels), INPUT_SIZE, INPUT_SIZE, 3), dtype='float32')
for i, img_id in tqdm(enumerate(labels['id'])):
    img = read_img(img_id, 'train', (INPUT_SIZE, INPUT_SIZE))
    x = preprocess_input(np.expand_dims(img.copy(), axis=0))
    x_train[i] = x
print('Train Images shape: {} size: {:,}'.format(x_train.shape, x_train.size))
```

```
1777it [00:09, 182.36it/s]
```

```
Train Images shape: (1777, 224, 224, 3) size: 267,488,256
```

In [13]:

```
Xtr = x_train[train_idx]
Xv = x_train[valid_idx]
print((Xtr.shape, Xv.shape, ytr.shape, yv.shape))
vgg_bottleneck = VGG16(weights='imagenet', include_top=False, pooling=POOLING)
train_vgg_bf = vgg_bottleneck.predict(Xtr, batch_size=32, verbose=1)
valid_vgg_bf = vgg_bottleneck.predict(Xv, batch_size=32, verbose=1)
print('VGG train bottleneck features shape: {} size: {:,}'.format(train_vgg_bf.shape, train_vg
g_bf.size))
print('VGG valid bottleneck features shape: {} size: {:,}'.format(valid_vgg_bf.shape, valid_vg
g_bf.size))
```

```
((1409, 224, 224, 3), (368, 224, 224, 3), (1409, 16), (368, 16))
1409/1409 [==============================] - 993s 705ms/step
368/368 [==============================] - 288s 784ms/step
VGG train bottleneck features shape: (1409, 512) size: 721,408
VGG valid bottleneck features shape: (368, 512) size: 188,416
```

LogReg on VGG bottleneck features

In [14]:

```
logreg = LogisticRegression(multi_class='multinomial', solver='lbfgs', random_state=SEED)
logreg.fit(train_vgg_bf, (ytr * range(NUM_CLASSES)).sum(axis=1))
valid_probs = logreg.predict_proba(valid_vgg_bf)
valid_preds = logreg.predict(valid_vgg_bf)
```

In [15]:

```
print('Validation VGG LogLoss {}'.format(log_loss(yv, valid_probs)))
print('Validation VGG Accuracy {}'.format(accuracy_score((yv * range(NUM_CLASSES)).sum(axis=1
), valid_preds)))
```

```
Validation VGG LogLoss 0.35205974794606615
Validation VGG Accuracy 0.9184782608695652
```

Not bad, 90% accuracy for the top 16 classes. The multiclass classification with 120 classes is more difficult so these LogLoss/Accuracy scores does not translate to LB.

Extract Xception bottleneck features

In [16]:

```
INPUT_SIZE = 299
POOLING = 'avg'
x_train = np.zeros((len(labels), INPUT_SIZE, INPUT_SIZE, 3), dtype='float32')
for i, img_id in tqdm(enumerate(labels['id'])):
    img = read_img(img_id, 'train', (INPUT_SIZE, INPUT_SIZE))
    x = xception.preprocess_input(np.expand_dims(img.copy(), axis=0))
    x_train[i] = x
print('Train Images shape: {} size: {:,}'.format(x_train.shape, x_train.size))
```

```
1777it [00:09, 183.91it/s]


Train Images shape: (1777, 299, 299, 3) size: 476,596,731
```

In [17]:

```
Xtr = x_train[train_idx]
Xv = x_train[valid_idx]
print((Xtr.shape, Xv.shape, ytr.shape, yv.shape))
xception_bottleneck = xception.Xception(weights='imagenet', include_top=False, pooling=POOLING
)
train_x_bf = xception_bottleneck.predict(Xtr, batch_size=32, verbose=1)
valid_x_bf = xception_bottleneck.predict(Xv, batch_size=32, verbose=1)
print('Xception train bottleneck features shape: {} size: {:,}'.format(train_x_bf.shape, train
_x_bf.size))
print('Xception valid bottleneck features shape: {} size: {:,}'.format(valid_x_bf.shape, valid
_x_bf.size))
```

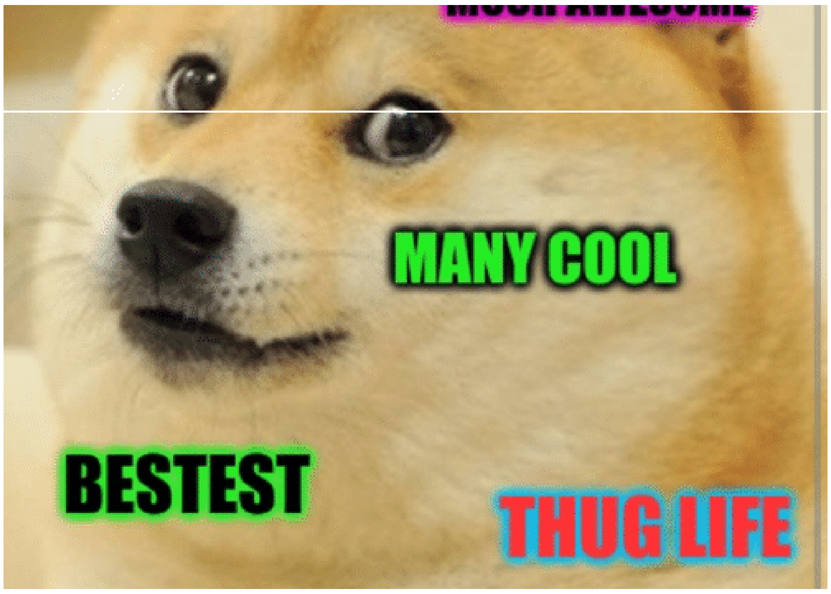
```
((1409, 299, 299, 3), (368, 299, 299, 3), (1409, 16), (368, 16))
1409/1409 [==============================] - 1053s 747ms/step
368/368 [==============================] - 268s 729ms/step
Xception train bottleneck features shape: (1409, 2048) size: 2,885,632
Xception valid bottleneck features shape: (368, 2048) size: 753,664
```

LogReg on Xception bottleneck features

In [18]:

```
logreg = LogisticRegression(multi_class='multinomial', solver='lbfgs', random_state=SEED)
logreg.fit(train_x_bf, (ytr * range(NUM_CLASSES)).sum(axis=1))
valid_probs = logreg.predict_proba(valid_x_bf)
valid_preds = logreg.predict(valid_x_bf)
print('Validation Xception LogLoss {}'.format(log_loss(yv, valid_probs)))
print('Validation Xception Accuracy {}'.format(accuracy_score((yv * range(NUM_CLASSES)).sum(ax
is=1), valid_preds)))
```

```
Validation Xception LogLoss 0.06829598590262777
Validation Xception Accuracy 0.9809782608695652
```



Much better! 98% accuracy 0.07 LogLoss.

Extract Inception bottleneck features

In [19]:
```
Xtr = x_train[train_idx]
Xv = x_train[valid_idx]
print((Xtr.shape, Xv.shape, ytr.shape, yv.shape))
inception_bottleneck = inception_v3.InceptionV3(weights='imagenet', include_top=False, pooling
=POOLING)
train_i_bf = inception_bottleneck.predict(Xtr, batch_size=32, verbose=1)
valid_i_bf = inception_bottleneck.predict(Xv, batch_size=32, verbose=1)
print('InceptionV3 train bottleneck features shape: {} size: {:,}'.format(train_i_bf.shape, tr
ain_i_bf.size))
print('InceptionV3 valid bottleneck features shape: {} size: {:,}'.format(valid_i_bf.shape, va
lid_i_bf.size))
```

```
((1409, 299, 299, 3), (368, 299, 299, 3), (1409, 16), (368, 16))
1409/1409 [==============================] - 578s 410ms/step
368/368 [==============================] - 151s 411ms/step
InceptionV3 train bottleneck features shape: (1409, 2048) size: 2,885,632
InceptionV3 valid bottleneck features shape: (368, 2048) size: 753,664
```

LogReg on Inception bottleneck features

In [20]:
```
logreg = LogisticRegression(multi_class='multinomial', solver='lbfgs', random_state=SEED)
logreg.fit(train_i_bf, (ytr * range(NUM_CLASSES)).sum(axis=1))
valid_probs = logreg.predict_proba(valid_i_bf)
valid_preds = logreg.predict(valid_i_bf)
```

In [21]:
```
print('Validation Inception LogLoss {}'.format(log_loss(yv, valid_probs)))
print('Validation Inception Accuracy {}'.format(accuracy_score((yv * range(NUM_CLASSES)).sum(a
xis=1), valid_preds)))
```

```
Validation Inception LogLoss 0.08069913954479578
Validation Inception Accuracy 0.967391304347826
```

LogReg on all bottleneck features

In [22]:

```
X = np.hstack([train_x_bf, train_i_bf])
V = np.hstack([valid_x_bf, valid_i_bf])
print('Full train bottleneck features shape: {} size: {:,}'.format(X.shape, X.size))
print('Full valid bottleneck features shape: {} size: {:,}'.format(V.shape, V.size))
logreg = LogisticRegression(multi_class='multinomial', solver='lbfgs', random_state=SEED)
logreg.fit(X, (ytr * range(NUM_CLASSES)).sum(axis=1))
valid_probs = logreg.predict_proba(V)
valid_preds = logreg.predict(V)
print('Validation Xception + Inception LogLoss {}'.format(log_loss(yv, valid_probs)))
print('Validation Xception + Inception Accuracy {}'.format(accuracy_score((yv * range(NUM_CLAS
SES)).sum(axis=1), valid_preds)))
```

```
Full train bottleneck features shape: (1409, 4096) size: 5,771,264
Full valid bottleneck features shape: (368, 4096) size: 1,507,328
Validation Xception + Inception LogLoss 0.07085797531382107
Validation Xception + Inception Accuracy 0.9755434782608695
```

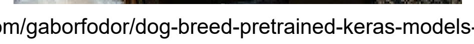
Training this model on the full dataset would give 0.3 LogLoss on LB.

Check errors

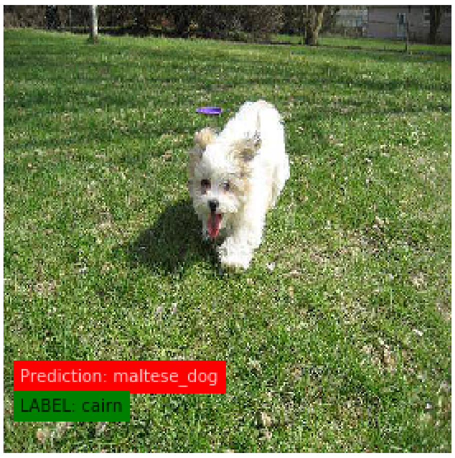
We still have a few misclassification errors.

In [23]:

```
valid_breeds = (yv * range(NUM_CLASSES)).sum(axis=1)
error_idx = (valid_breeds != valid_preds)
for img_id, breed, pred in zip(labels.loc[valid_idx, 'id'].values[error_idx],
                                [selected_breed_list[int(b)] for b in valid_preds[error_idx]],
                                [selected_breed_list[int(b)] for b in valid_breeds[error_idx
]]):
    fig, ax = plt.subplots(figsize=(5,5))
    img = read_img(img_id, 'train', (299, 299))
    ax.imshow(img / 255.)
    ax.text(10, 250, 'Prediction: %s' % pred, color='w', backgroundcolor='r', alpha=0.8)
    ax.text(10, 270, 'LABEL: %s' % breed, color='k', backgroundcolor='g', alpha=0.8)
    ax.axis('off')
    plt.show()
```

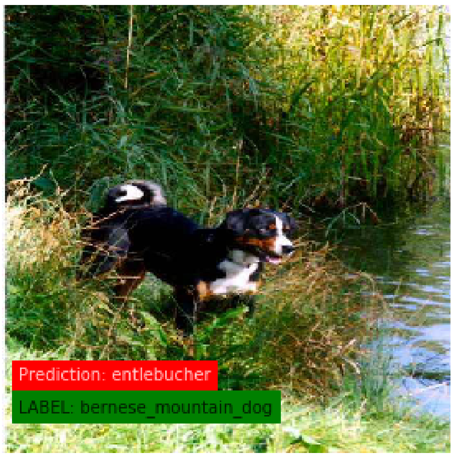


Prediction: tibetan_terrier
LABEL: shih-tzu


Prediction: maltese_dog
LABEL: cairn

Prediction: scottish_deerhound
LABEL: airedale

Prediction: pomeranian
LABEL: shih-tzu


Prediction: entlebucher
LABEL: bernese_mountain_dog

Prediction: shih-tzu
LABEL: cairn

Prediction: tibetan_terrier
LABEL: shih-tzu

```
In [24]:
end = dt.datetime.now()
print('Total time {} s.'.format((end - start).seconds))
print('We almost used the one hour time limit.')
```

```
Total time 3401 s.
We almost used the one hour time limit.
```

Did you find this Kernel useful?
Show your appreciation with an upvote

196

Comments (45) All Comments Sort by Hotness

Click here to enter a comment...

yuquan.li • Posted on Latest Version • 2 months ago • Options • Reply 1

Appreciate your sharing

yuquan.li • Posted on Latest Version • 2 months ago • Options • Reply 1

orry, help me please. Can i load just, i.e., 10 images from 120 classes?

yuquan.li • Posted on Latest Version • 2 months ago • Options • Reply 1

good job

Stanwar • Posted on Version 17 • 4 months ago • Options • Reply 3

Thank you so much for this beluga. I'd been meaning to learn how to do this for a while and I found it very easy to follow. It's one of my favourite kernels for sure.

I wondered if you or anyone else has tried (and had any joy with) synthetic images? I tried the technique at the link below but it seemed to worsen fit.

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Hemant • Posted on Version 18 • 4 months ago • Options • Reply 1

Thanks, for the kernel, it is worthy for beginners

HyungsukKang • Posted on Version 13 • 6 months ago • Options • Reply 5

It is a really nice kernel to study with :0 However, forked kernel does not work on predicting with Resnet50 model due to the order of declaring x within predicting pipeline. So I fixed the code :)

the declaring x in pipeline predicting with Resnet50 model should be from:

```
x = preprocess_input(img.copy())
ax.imshow(img / 255.)
x = np.expand_dims(img, axis=0)
```

to:

```
x = np.expand_dims(img, axis=0)
ax.imshow(img / 255.)
x = preprocess_input(x)
```

beluga • Posted on Version 14 • 6 months ago • Options • Reply 1

Thanks for notice! Fixed.

ranjiewen • Posted on Version 18 • 4 months ago • Options • Reply 0

i don't know the code different, can you explain??

Bostjan Mrak • Posted on Version 5 • 6 months ago • Options • Reply 6

Great job! I have similar kernel in drafts but I didn't publish it yet, gladly you make it and users probably won't learn anything new from my kernel. Bottleneck just works and this method can give very good results. Highest accuracy is with xception, although xception is one of the slowest networks, it's well worth to wait :)

ranjiewen • Posted on Version 15 • 5 months ago • Options • Reply 0

thanks , i know 1*1 convolutional called bottleneck . and why there called bottleneck, this only get last conv feature map?

liuchunhui • Posted on Version 17 • 4 months ago • Options • Reply 0

1*1 conv often used for reduce connections between two conv layer, for example conv1-->conv2, to conv1-->1*1 cov-->conv2. I think shape of 1*1 conv is smaller than shapes of conv1 and conv2, just like bottle neck.

Sayan • Posted on Latest Version • 3 months ago • Options • Reply 1

A 1x1 convolution helps in reducing the number of channels, whereas pooling layers reduces the height and the width of the convolution volume.

Fredust • Posted on Version 13 • 6 months ago • Options • Reply 0

nice

Nicky_ua • Posted on Version 14 • 5 months ago • Options • Reply 0

I can't download files from input data

Dhanush Kamath • Posted on Version 14 • 5 months ago • Options • Reply 0

I see that you have applied Logistic Regression on the bottleneck features of the images. Why not attach a new fully connected layer at the end of the pretrained convnet and perform transfer learning?

beluga • Posted on Version 17 • 4 months ago • Options • Reply 1

Sure, additional fully connected layer would be better though it needs more time to finetune.

Dhanush Kamath • Posted on Version 14 • 5 months ago • Options • Reply 0

DenysFilippov • Posted on Version 14 • 5 months ago • Options • Reply 0

Cool

alyssa • Posted on Version 14 • 5 months ago • Options • Reply 0

Thanks!

Alexander Kireev • Posted on Version 14 • 5 months ago • Options • Reply 0

Thank You) brilliant work) good support for beginners

Alexander Kireev • Posted on Version 14 • 5 months ago • Options • Reply 0

Sorry, help me please. Can i load just, i.e., 10 images from 120 classes? This can help me to work with whole set and save kernel from interrupting.

5 months ago

This Comment was deleted.

5 months ago

This Comment was deleted.

5 months ago

This Comment was deleted.

beluga • Posted on Version 17 • 4 months ago • Options • Reply 1

I have a similar kernel for seedling classification. I use fix sample per class there https://www.kaggle.com/gaborfodor/seedlings-pretrained-keras-models

GunaPandian • Posted on Version 14 • 5 months ago • Options • Reply 0

Clear and precise ...thanks for it

Mark Conrad • Posted on Version 15 • 5 months ago • Options • Reply 0

Thanks for the kernel!

RichardBJ • Posted on Version 15 • 5 months ago • Options • Reply 0

Pretty cool... I'm impressed that the misclassifications are sort of "easy mistakes" to make, even for a dog expert!

AnkitPaliwal • Posted on Version 18 • 4 months ago • Options • Reply 0

That's a quite informative kernel for beginners in deep learning. Thank you for sharing beluga

Aydin Ayanzadeh • Posted on Version 18 • 4 months ago • Options • Reply 0

Really great kernel. its very worthy for start.

Ajit Puthenputhu... • Posted on Version 18 • 4 months ago • Options • Reply 0

Thanks for the kernel!

Atul A • Posted on Version 18 • 4 months ago • Options • Reply 0

Hi @Beluga - thanks - this is super useful. If I'm creating my own Kernel, how do I upload my own data or pre-trained models to it? Thanks!

beluga • Posted on Version 18 • 4 months ago • Options • Reply 0

You could create a public dataset with your data and then use that within your kernel. https://www.kaggle.com/product-feedback/32423

iest • Posted on Version 18 • 4 months ago • Options • Reply 0

Thank you so much.. I learn a lot from this kernel

guillermo • Posted on Latest Version • 3 months ago • Options • Reply 0

Thank you so much for your kernel. I noob and I'm try to understand some basic things,,, A question is, why can you use ResNet50 directly and the others models you have to extract the features to apply and LogicRegression later? You use VGG you train the network with the dog images and get the weights to extract the features, besides you exclude the last layer (include_top=False). After you use LG with this weights, I guess that it's like to replace the last layer of VGG for this LG. Why don't use just VGG? Why don't you flatten the last layer and connect to a dense layer?

morenoh149 • Posted on Latest Version • 2 months ago • Options • Reply 0

All of the pretrained models shipped with keras ship with the 'imagenet' weights and classes out of the box (https://keras.io/applications/). As such all of the models could have been used directly.

For those curious, the keras models use 1000 for the default value of the 'classes' parameter, same as the 1000 categories on the imagenet 2012 challenge http://image-net.org/challenges/LSVRC/2012/browse-synsets I wrote a small script and can verify that the 120 dog breeds listed at https://www.kaggle.com/c/dog-breed-identification/data are a subset of the 1000 categories in the imagenet 2012 challenge.

This kernel must be inspired by https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

> We will use the VGG16 architecture, pre-trained on the ImageNet dataset --a model previously featured on this blog. Because the ImageNet dataset contains several "cat" classes (persian cat, siamese cat...) and many "dog" classes among its total of 1000 classes, this model will already have learned features that are relevant to our classification problem. In fact, it is possible that merely recording the softmax predictions of the model over our data rather than the bottleneck features would be enough to solve our dogs vs. cats classification problem extremely well. However, the method we present here is more likely to generalize well to a broader range of problems, including problems featuring classes absent from ImageNet.

and

> The reason why we are storing the features offline rather than adding our fully-connected model directly on top of a frozen convolutional base and running the whole thing, is computational effiency. Running VGG16 is expensive, especially if you're working on CPU, and we want to only do it once. Note that this prevents us from using data augmentation.

A dense final layer was probably not used so the kernel could run on a VM cpu under an hour.

OneTree • Posted on Latest Version • 3 months ago • Options • Reply 0

A good Kernel, needs a lot effort.

cxlcc • Posted on Latest Version • 3 months ago • Options • Reply 0

Appreciate your sharing, Beluga.

Given the run time limit of Kernel, it is hard to train 120 classes in one go as you mentioned.

Is it possible to complete the training of 120 classes in several runs by storing the weights at the end of one run and reloading the weights and continue training at another run?

morenoh149 • Posted on Latest Version • 2 months ago • Options • Reply 0

It is probably possible, but at that point you are probably better off running your experiments on your own cloud like Amazon Sagemaker, databricks, crestle, floydhub or paperspace.

TARS • Posted on Latest Version • 2 months ago • Options • Reply 0

Nice job mate!

Abinesh Sankar • Posted on Latest Version • 2 months ago • Options • Reply 0

How do I predict on the test set??

Qitao Shi • Posted on Latest Version • 2 months ago • Options • Reply 0

Thank you very much!

saurabh agrawal • Posted on Latest Version • a month ago • Options • Reply 0

In the last part, you have trained logistic regression on combined output of inception and xception both. I can see that results are better but did not understand really why. What is the hypothesis for this?

Lucky-Rathore • Posted on Latest Version • 15 days ago • Options • Reply 0

hats off.

Song • Posted on Latest Version • 7 days ago • Options • Reply 0

thanks

© 2018 Kaggle Inc

Our Team Terms Privacy Contact/Support