VGG16: Deep Neural Network for Image Classification

05/31/2019

- Oxford Visual Geometry Group (VGG)

VGG16 Deep Convolutional Neural Network (Deep CNN)

- Keras VGG-16: 16 trainable layers (13 convolutional, plus 3 MLP classifier layers)
- 138.4 million parameters (Total params: 138,357,544)
- Convolutional base only (13 layers) - Total params: 14,714,688
- Pre-trained on ImageNet classification - 1,000 output image classes
- Main arguments: include_top=True/False, input_shape=(224, 224, 3) (optional)
- Final feature map output shape (7, 7, 512)

VGG16 and VGG19 developed by the Oxford Visual Geometry Group (VGG), 2012 - 2014. ImageNet performance (ICLR 2015):

- Top-1 accuracy of 75/2% (24.8 error rate)
- Top-5 accuracy of 92.5% (7.5% error rate)

Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan & Andrew Zisserman, Visual Geometry Group, Department of Engineering Science, University of Oxford, ICLR 2015. (https://arxiv.org/abs/1409.1556 (https://arxiv.org/abs/1409.1556))

VGG16 ImageNet image classification

VGG16 was state of the art in 2014. Newer deep network architectures, with far fewer parameters, and training and inference time requirements, outperform VGG16 in 2019 on ImageNet and other tasks.

accuracies (document values, refs):

- Inception V3
- Xception
- ResNet (2015)
- SSD (2018)
- YOLOv3 (11/2018)
- EfficientNet (05/2019)

VGG-16 Architecture

VGG-16 has 13 Conv2D layers, 5 MaxPooling2D, 1 flatten, and 3 output classifier layers (Dense):

```
(2) Conv2D-1 - Conv2D-2 - MaxPooling2D
(2) Conv2D-1 - Conv2D-2 - MaxPooling2D
(3) Conv2D-1 - Conv2D-2 - Conv2D-3 - MaxPooling2D
(3) Conv2D-1 - Conv2D-2 - Conv2D-3 - MaxPooling2D
(3) Conv2D-1 - Conv2D-2 - Conv2D-3 - MaxPooling2D
flatten (Flatten)
fc1 (Dense) - fc2 (Dense) - predictions (Dense)
```

Other pre-trained CNN models part of `keras.applications` :

- VGG-16, VGG-19, ResNet-50, Inception, Inception-ResNet, Xception.

In [1]:
```
# Using TF 1.13.1; TF 2.0.0-alpha0 not compatible
import numpy as np
import os
import cv2
from datetime import datetime

import matplotlib.pyplot as plt

import tensorflow as tf
import tensorflow.keras as keras

from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator

print("TensorFlow:", tf.__version__, "Keras:", keras.__version__)
```

```
TensorFlow: 1.13.1 Keras: 2.2.4-tf
```

In [2]:
```
# VGG16 Full Pre-Trained Network (Keras)
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions

vgg16_full = VGG16(weights='imagenet', include_top=True, input_shape=(224, 224, 3))
```

```
WARNING:tensorflow:From /Users/nelson/dev/anaconda3/lib/python3.7/site-packages/tensorflow/python/op
s/resource_variable_ops.py:435: colocate_with (from tensorflow.python.framework.ops) is deprecated a
nd will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

VGG16 weights (553 MB size):

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weigh
ts_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 36s
```

553 MB VGG16 model download the first time VGG16() is called from Keras.

In [3]:
```
vgg16_full.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0

```
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

VGG16 input/output tensors and classifier

VGG16 convolutional base input tensor (images): (_, 224, 224, 3)

```
input_1 (InputLayer)         (None, 224, 224, 3)       0
```

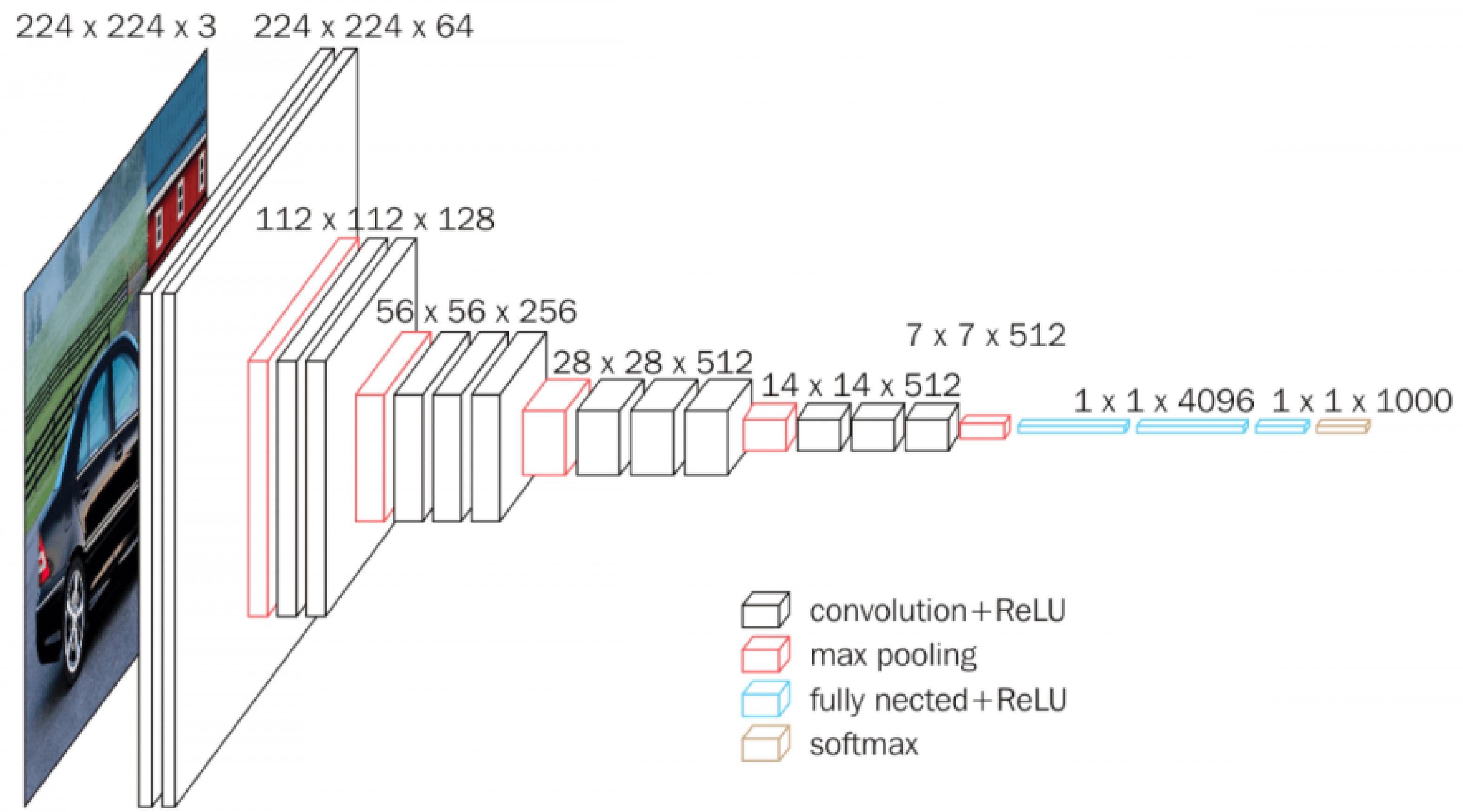
VGG16 convolutional base output tensor: (_, 7, 7, 512)

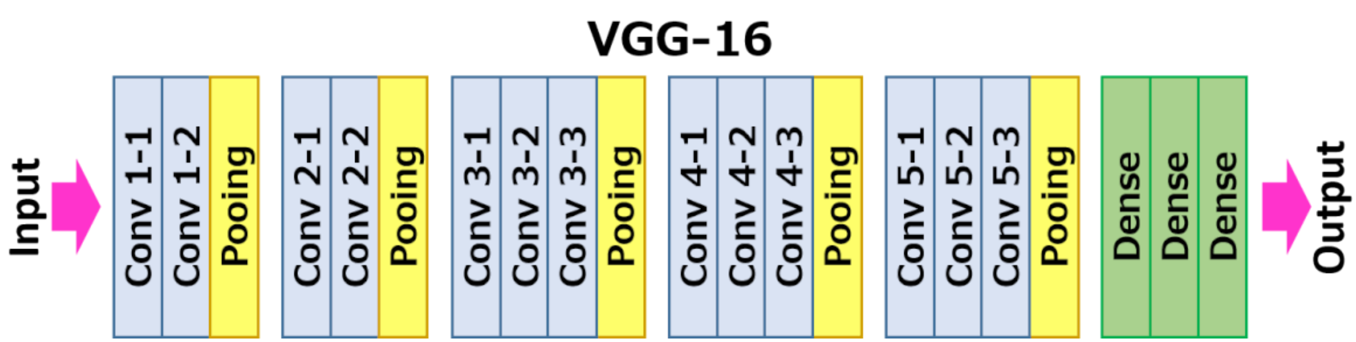
```
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
```

VGG16 output classifier input tensor (features): (_, 7, 7, 512)

```
flatten (Flatten)            (None, 25088)             0
fc1 (Dense)                  (None, 4096)              102764544
fc2 (Dense)                  (None, 4096)              16781312
predictions (Dense)          (None, 1000)              4097000
```

VGG16 Architecture (ImageNet classifier)





In [4]:
```
# vgg16_full
print("VGG16 Layers")
layer = vgg16_full.layers[0]
for layer in vgg16_full.layers:
    print(layer)
```

```
VGG16 Layers
<tensorflow.python.keras.engine.input_layer.InputLayer object at 0xb35dbf9e8>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36b76748>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x10c6b3ba8>
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0xb36b76eb8>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36ba9e48>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36bd8c50>
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0xb36c36ba8>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36c1ed68>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36c56da0>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36c965f8>
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0xb36cd1240>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36cd1208>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36d04d68>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36d3d048>
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0xb36d79c88>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36d61c18>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36d98da0>
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0xb36dd63c8>
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0xb36decf98>
<tensorflow.python.keras.layers.core.Flatten object at 0xb36e0e898>
<tensorflow.python.keras.layers.core.Dense object at 0xb36e2aba8>
<tensorflow.python.keras.layers.core.Dense object at 0xb36e45b70>
<tensorflow.python.keras.layers.core.Dense object at 0xb36e2ab00>
```

In [5]:
```
# Load some test images for predictions with VGG16
image_dir = './data/images/'
image_fns = ['african_elephant.jpg', 'indian_elephant.jpg', 'cat.1.jpg', 'cat.2.jpg',
          'dog.66.jpg', 'dog.166.jpg']
image_fns_wrong = ['african_elephant.jpg', 'indian_elephant.jpg', 'cat.1.jpg', 'cat.2.jpg',
          'dog.66.jpg', 'dog.166.jpg', 'giraffe.jpg', 'man.jpg', 'woman.jpg']

image_fns_mix = ['african_elephant.jpg',
                 'african_elephant_people.jpg', 'african_elephant_jeep.jpg',
                 'dog_bike_truck.jpg', 'giraffe_zebra.jpg',
                 'person_horse_dog.jpg', 'man_woman.jpg']
image_fns_art = ['art/dama_sentada.jpg', 'art/scream.jpg']

# PIL images resized to 224x224
# Alternatives: resize to smaller dimension (width, height), crop center instead
images = []
for image_fn in image_fns_mix:
    img = image.load_img(image_dir+image_fn, target_size=(224, 224))
    img_arr = np.array(img)
    images.append(img_arr)

images_arr = np.array(images) # /255
images_pre = preprocess_input(images_arr * 1)
print("images_arr.shape:", images_arr.shape)
```

```
images_arr.shape: (7, 224, 224, 3)
```

In [6]:
```python
# VGG16 preprocessed images reshaped and streched to (224, 224, 3)

# image, preprocessed images_pre
print("images[0].shape:", images[0].shape, end=' - ')
print("images_pre.shape:", images_pre.shape)
plt.figure(figsize=(18,10))
plt.subplot(121)
plt.imshow(images[0])
plt.subplot(122)
plt.imshow(images[1])
plt.show();
```

```
images[0].shape: (224, 224, 3) - images_pre.shape: (7, 224, 224, 3)
```

In [7]:
```
# Predict classes for loaded images
predictions = vgg16_full.predict(images_pre, steps=1)
print("shape:", predictions.shape)
print("argmax[0]:", np.argmax(predictions[0]))
print(predictions)
```

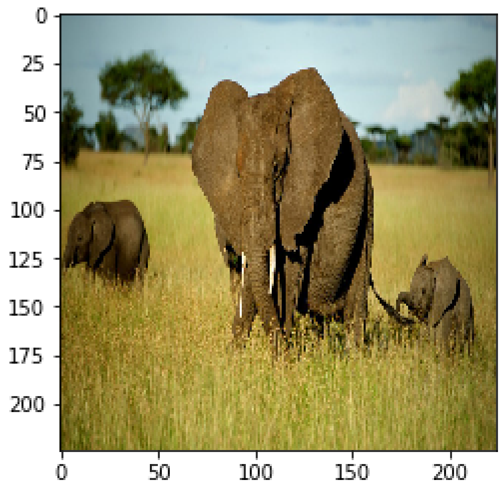
```
shape: (7, 1000)
argmax[0]: 386
[[4.60237670e-09 1.11573875e-11 2.53057453e-11 ... 2.81511014e-09
  9.52526591e-09 1.46931145e-09]
 [1.34543268e-06 9.10646114e-10 2.31577624e-09 ... 6.98862195e-08
  8.21964647e-08 9.80019532e-09]
 [9.29580946e-09 8.72959760e-11 4.27643504e-10 ... 7.91589572e-10
  9.35444500e-10 9.31806521e-10]
 ...
 [1.24174633e-07 2.25469449e-07 2.04179130e-07 ... 1.13140766e-06
  1.37783873e-05 2.84834073e-06]
 [1.04798099e-08 4.55719729e-09 1.18309423e-07 ... 7.96514055e-09
  4.42225144e-07 2.28499016e-06]
 [2.44380345e-07 2.66996665e-07 4.40396661e-07 ... 1.97017403e-06
  5.18206471e-05 7.71542022e-04]]
```

In [8]:
```
# Top 5 predictions for images[0]
top_5 = tf.keras.applications.vgg16.decode_predictions(predictions, top=5)
for class_id, name, y_proba in top_5[0]:
    print(" {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
print()
```

```
 n02504458 - African_elephant 69.72%
 n01871265 - tusker       19.20%
 n02504013 - Indian_elephant 6.60%
 n02410509 - bison        3.56%
 n02437312 - Arabian_camel 0.43%
```

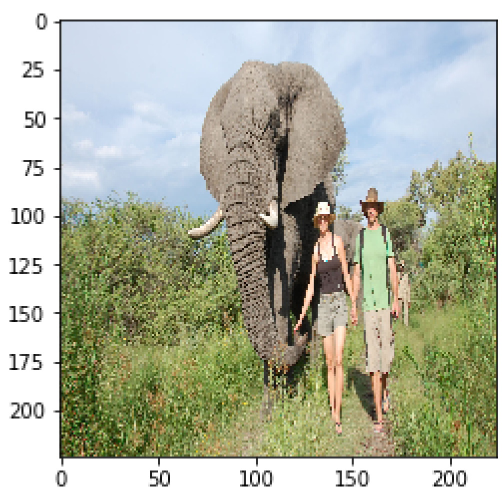
In [9]:
```
# Top 5 predictions for all images
top_5 = tf.keras.applications.vgg16.decode_predictions(predictions, top=5)
for image_index in range(len(images)):
    print("Image {}".format(image_index))
    plt.imshow(images[image_index])
    plt.show()
    for class_id, name, y_proba in top_5[image_index]:
        print(" {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
    print()
```

```
Image 0
```



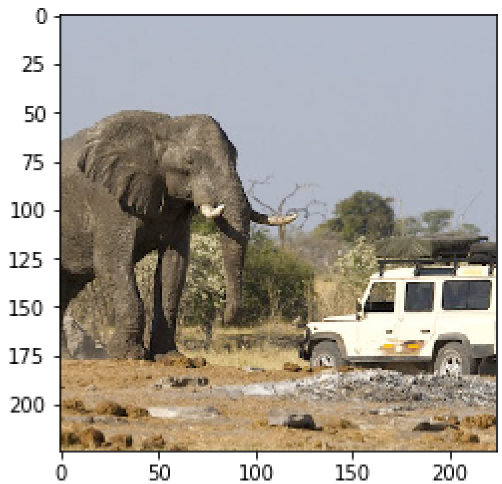
```
n02504458 - African_elephant 69.72%
n01871265 - tusker       19.20%
n02504013 - Indian_elephant 6.60%
n02410509 - bison        3.56%
n02437312 - Arabian_camel 0.43%
```

```
Image 1
```



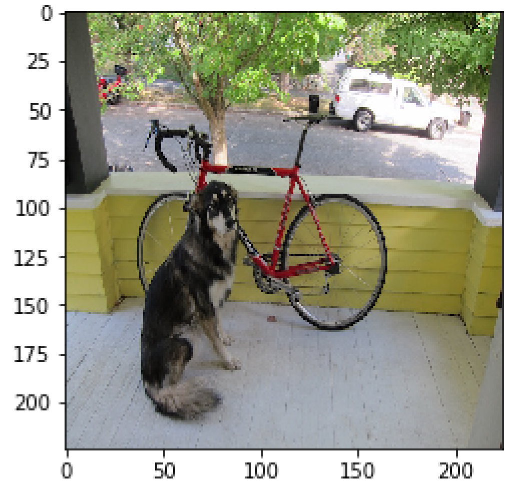
```
n01871265 - tusker       53.20%
n02504458 - African_elephant 43.39%
n02504013 - Indian_elephant 3.27%
n01704323 - triceratops  0.08%
n02963159 - cardigan     0.01%
```

Image 2



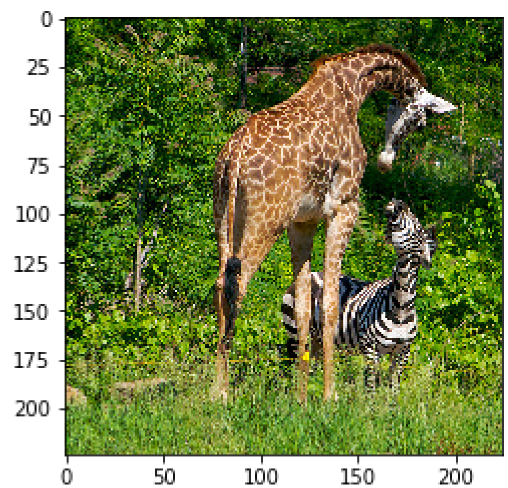
```
n02504458 - African_elephant 86.31%
n01871265 - tusker       11.22%
n02504013 - Indian_elephant 2.44%
n03594945 - jeep         0.01%
n02486410 - baboon       0.00%
```

Image 3



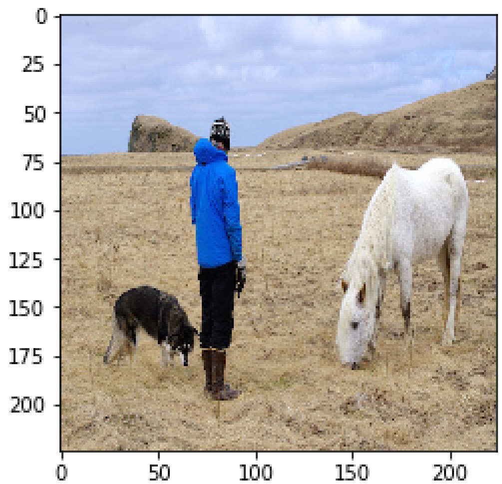
```
n02110063 - malamute       32.37%
n02110185 - Siberian_husky 21.75%
n02109961 - Eskimo_dog     15.27%
n03218198 - dogsled        5.32%
n02106166 - Border_collie  4.22%
```

```
Image 4
```



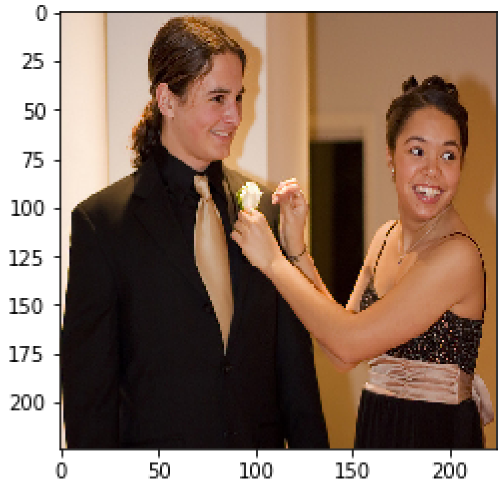
```
n02391049 - zebra         43.26%
n01518878 - ostrich       28.91%
n02130308 - cheetah       8.68%
n02423022 - gazelle       6.10%
n02422699 - impala        2.27%
```

Image 5




```
n02437616 - llama         68.91%
n02105412 - kelpie        6.04%
n02111500 - Great_Pyrenees 4.89%
n02412080 - ram           4.70%
n02104029 - kuvasz        4.00%
```

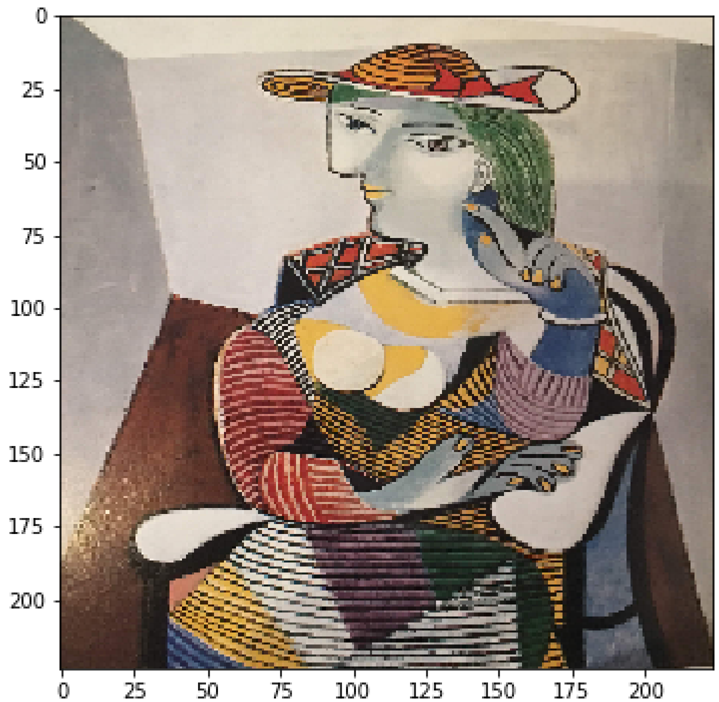
Image 6




```
n10148035 - groom        39.19%
n02883205 - bow_tie      27.01%
n04350905 - suit         11.94%
n03450230 - gown         8.47%
n03770439 - miniskirt    2.00%
```

In [11]:
```
# Another image (painting)
image_path = image_dir+image_fns_art[0]
print("image_path:", image_path)
img = image.load_img(image_path, target_size=(224, 224))
img_arr = np.array(img)
images = np.expand_dims(img_arr, axis=0)
images_pre = preprocess_input(images)
plt.figure(figsize=(6,6))
plt.imshow(img)
plt.show()

# Print object predictions
predictions = vgg16_full.predict(images_pre, steps=1)
for class_id, name, y_proba in top_5[image_index]:
    print(" {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
```

```
image_path: ./data/images/art/dama_sentada.jpg
```



```
n10148035 - groom        39.19%
n02883205 - bow_tie      27.01%
n04350905 - suit         11.94%
n03450230 - gown         8.47%
n03770439 - miniskirt    2.00%
```

VGG16 Transfer learning: Custom output classifier

Using VGG16 convolutional base + Dense 25088 x 256 x 1 classifier

- 13 convolutional layers from VGG16
- 2 fully-connected layers of binary output classifier
- Trainable params: 21,137,729 (14,714,688 base; 6,423,041 classifier)
- Image input shape does not affect size of the network, only size of output feature tensor

```
In [12]: # Import VGG16 convolutional base, with original input shape (224, 224, 3)
         K.clear_session()
         vgg16_base_224 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

VGG16 base (59 MB)

```
Initial Download:
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weigh
ts_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 7s
```

59 MB model download the first time VGG16 base is called from Keras.

Model: "vgg16_base_224"

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
...
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Output shape

```
vgg16_base_224.output_shape: ( _, 7, 7, 512)
```

In [13]:
```
vgg16_base_224.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0         
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
```

```
Non-trainable params: 0
_________________________________________________________________
```

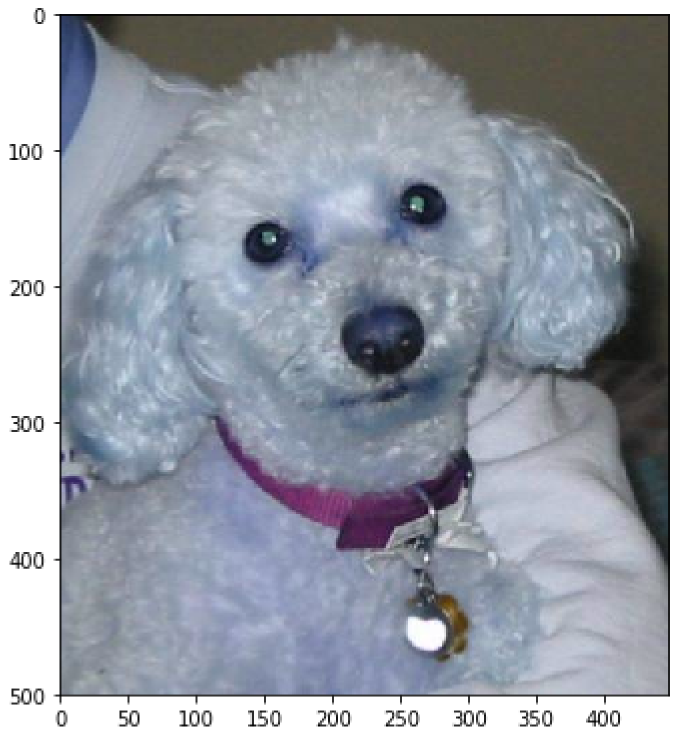
Kaggle image classification challenge: sample dogs vs. cats

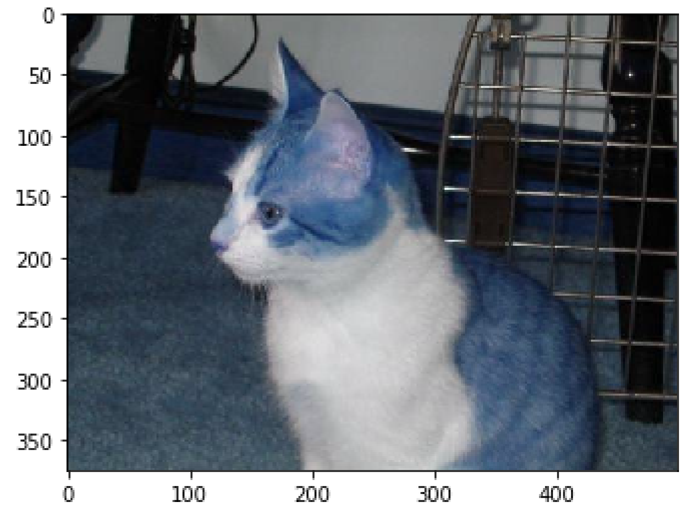
- Kaggle competition URL: https://kaggle.com/c/dogs-vs-cats (https://kaggle.com/c/dogs-vs-cats)
- dogs vs. cats data: https://kaggle.com/c/dogs-vs-cats/data (https://kaggle.com/c/dogs-vs-cats/data)

Based on *Deep Learning with Python*, François Chollet, 2017, Manning Publications, (Chapter 5)
https://www.amazon.com/Deep-Learning-Python-Francois-Chollet/dp/1617294438 (https://www.amazon.com/Deep-Learning-Python-Francois-Chollet/dp/1617294438)

In [14]:
```
# Sample kaggle/dogs-vs-cats images
kaggle_dogcats_dir = '/Users/nelson/dev/datasets/cv/kaggle/dogs-vs-cats_small/'

train_dir = os.path.join(kaggle_dogcats_dir, 'train/')
dog_fns = os.listdir(train_dir + 'dogs')
cat_fns = os.listdir(train_dir + 'cats')
dog_img = cv2.imread(train_dir + 'dogs/' + dog_fns[0])
cat_img = cv2.imread(train_dir + 'cats/' + cat_fns[0])

print("dog_img.shape:", dog_img.shape, end=' - ')
print("cat_img.shape:", cat_img.shape)
print("images will be reshaped to (224, 224, 3)")
plt.figure(figsize=(12,10))
plt.subplot(121)
plt.imshow(dog_img)
plt.subplot(122)
plt.imshow(cat_img)
plt.show();
```

```
dog_img.shape: (500, 448, 3) - cat_img.shape: (375, 499, 3)
images will be reshaped to (224, 224, 3)
```





MLP Dense 25088 x 256 x 1 output classifier for VGG16 top layer

MLP Dense 25088 x 256 x 1 binary classifier - model.summary:

```
Layer (type)                 Output Shape              Param #   
=================================================================
dense_6 (Dense)              (None, 256)               6422784   
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 257       
=================================================================
Total params: 6,423,041
Trainable params: 6,423,041
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
# MLP Dense 8192 x 256 x 1 output classifier for VGG16 top layer

vgg16_features_shape = vgg16_base_224.output_shape[1:]
print("vgg16_features_shape:", vgg16_features_shape)
print("mlp_input_shape:", np.product(vgg16_features_shape))
mlp_input_dim = np.product(vgg16_features_shape) # 7*7*512 = 25088

mlp = models.Sequential()
mlp.add(layers.Dense(256, activation='relu', input_dim=mlp_input_dim))
mlp.add(layers.Dropout(0.5))
mlp.add(layers.Dense(1, activation='sigmoid'))

mlp.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy', metrics=['acc'])
```

```
vgg16_features_shape: (7, 7, 512)
mlp_input_shape: 25088
```

In [17]:
```
# MLP summary
mlp.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 256)               6422784
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 6,423,041
Trainable params: 6,423,041
Non-trainable params: 0
_________________________________________________________________
```

VGG16 Image Preprocessing

Preprocess image dataset with vgg16_base pretrained VGG16 model head.

- Input image size: (224, 224, 3)
- Output feature size: (7, 7, 512)

In [18]:
```
# Image feature extraction
# kaggle_dogcats_dir = '/Users/nelson/dev/datasets/cv/kaggle/dogs-vs-cats_small'

train_dir = os.path.join(kaggle_dogcats_dir, 'train')
validation_dir = os.path.join(kaggle_dogcats_dir, 'validation')
test_dir = os.path.join(kaggle_dogcats_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

def extract_features(directory, sample_count):
    # features = np.zeros(shape=(sample_count, 4, 4, 512))
    features = np.zeros(shape=(sample_count, 7, 7, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='binary')   # TBD: ImageDataGenerator, make multinomial
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = vgg16_base_224.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels

# Timing
start_time = datetime.now()
print("VGG16 conv_base Start:", str(start_time))

train_features, train_labels = extract_features(train_dir, 2000)
train_time = datetime.now()
print("Training features done:", str(train_time))

validation_features, validation_labels = extract_features(validation_dir, 1000)
val_time = datetime.now()
print("Validation features done:", str(val_time))

test_features, test_labels = extract_features(test_dir, 1000)
```

```
# Time
end_time = datetime.now()
print("Test features done/End:", str(end_time))
print("VGG16 conv_base Total seconds:", str(end_time-start_time))
str(start_time), str(end_time), (end_time-start_time)
```

```
VGG16 conv_base Start: 2019-07-24 06:05:10.639047
Found 2000 images belonging to 2 classes.
Training features done: 2019-07-24 06:13:25.406459
Found 1000 images belonging to 2 classes.
Validation features done: 2019-07-24 06:17:40.675229
Found 1000 images belonging to 2 classes.
Test features done/End: 2019-07-24 06:21:56.844798
VGG16 conv_base Total seconds: 0:16:46.205751
```

Out[18]:
```
('2019-07-24 06:05:10.639047',
 '2019-07-24 06:21:56.844798',
 datetime.timedelta(seconds=1006, microseconds=205751))
```

VGG16 Base: Time for feature extraction (CPU)

- 4,000 total images (2,000 training, 1,000 validation, 1,000 test)
- 224 x 224 x 3 resolutiion

Total time: 15:51 minutes (1010 seconds) - 0.25 s/image (on CPU, i7 quad-core)

In [19]:
```
# Features shape: VGG16 conv_base output feature map shape (7, 7, 512)
print("train_features.shape:", train_features.shape)
print("validation_features.shape:", validation_features.shape)
print("test_features.shape:", test_features.shape)
```

```
train_features.shape: (2000, 7, 7, 512)
validation_features.shape: (1000, 7, 7, 512)
test_features.shape: (1000, 7, 7, 512)
```

In [20]:
```
# Reshape features: extracted features from (samples, 7, 7, 512) to (samples, 25088)
train_features = np.reshape(train_features, (2000, 7*7*512))
validation_features = np.reshape(validation_features, (1000, 7*7*512))
test_features = np.reshape(test_features, (1000, 7*7*512))

print("MLP reshaped train_features.shape:", train_features.shape)
```

```
MLP reshaped train_features.shape: (2000, 25088)
```

Output MLP Classifier Training and Evaluation

Train output MLP Classifier with pre-processed image from VGG16 model head.

- Classifier input feature size: (4, 4, 512)
- Output: MLP sigmoid output Dense layer

In [22]:
```
# Time
start_time = datetime.now()

history = mlp.fit(train_features, train_labels, epochs=20, batch_size=20,
                    validation_data=(validation_features, validation_labels))

# Time
end_time = datetime.now()
print("MLP Dense 25088*256*1 - Total seconds:", str(end_time-start_time))
```

```
Train on 2000 samples, validate on 1000 samples
Epoch 1/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.3842 - acc: 0.8275 - val_loss:
0.2900 - val_acc: 0.8920
Epoch 2/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.2770 - acc: 0.8880 - val_loss:
0.2628 - val_acc: 0.8970
Epoch 3/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.2318 - acc: 0.9070 - val_loss:
0.2366 - val_acc: 0.9030
Epoch 4/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.1934 - acc: 0.9245 - val_loss:
0.2304 - val_acc: 0.9050
Epoch 5/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.1649 - acc: 0.9445 - val_loss:
0.2416 - val_acc: 0.8950
Epoch 6/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.1436 - acc: 0.9505 - val_loss:
0.2158 - val_acc: 0.9070
Epoch 7/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.1287 - acc: 0.9505 - val_loss:
0.2068 - val_acc: 0.9120
Epoch 8/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.1103 - acc: 0.9630 - val_loss:
0.2022 - val_acc: 0.9150
Epoch 9/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0948 - acc: 0.9685 - val_loss:
0.2193 - val_acc: 0.9080
Epoch 10/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0937 - acc: 0.9710 - val_loss:
0.2259 - val_acc: 0.9020
Epoch 11/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0775 - acc: 0.9780 - val_loss:
0.2030 - val_acc: 0.9130
Epoch 12/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0657 - acc: 0.9820 - val_loss:
0.2047 - val_acc: 0.9110
Epoch 13/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0589 - acc: 0.9830 - val_loss:
0.2104 - val_acc: 0.9090
Epoch 14/20
2000/2000 [==============================] - 4s 2ms/sample - loss: 0.0549 - acc: 0.9850 - val_loss:
0.2110 - val_acc: 0.9090
```

```
Epoch 15/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0471 - acc: 0.9910 - val_loss:
0.2104 - val_acc: 0.9140
Epoch 16/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0429 - acc: 0.9905 - val_loss:
0.2095 - val_acc: 0.9120
Epoch 17/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0370 - acc: 0.9920 - val_loss:
0.2059 - val_acc: 0.9170
Epoch 18/20
2000/2000 [==============================] - 4s 2ms/sample - loss: 0.0310 - acc: 0.9955 - val_loss:
0.2084 - val_acc: 0.9170
Epoch 19/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0281 - acc: 0.9960 - val_loss:
0.2207 - val_acc: 0.9160
Epoch 20/20
2000/2000 [==============================] - 3s 2ms/sample - loss: 0.0260 - acc: 0.9970 - val_loss:
0.2161 - val_acc: 0.9220
MLP Dense 25088*256*1 - Total seconds: 0:01:06.887846
```

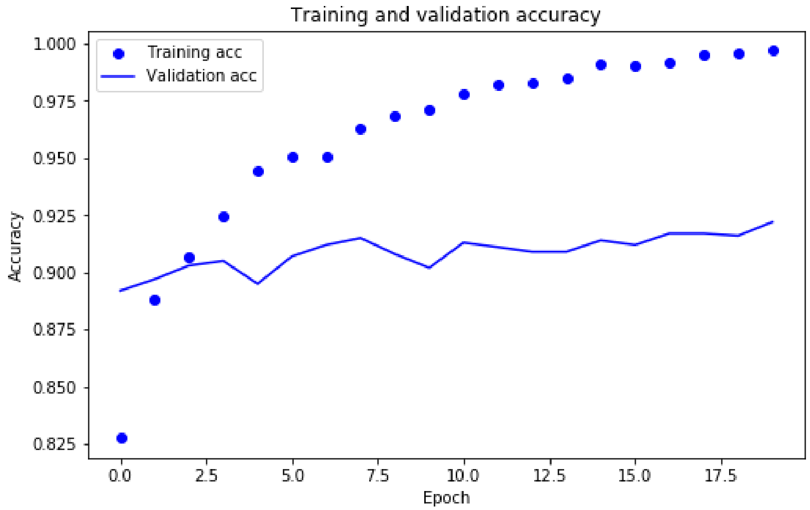
MLP output classifier: Time for classification (CPU)

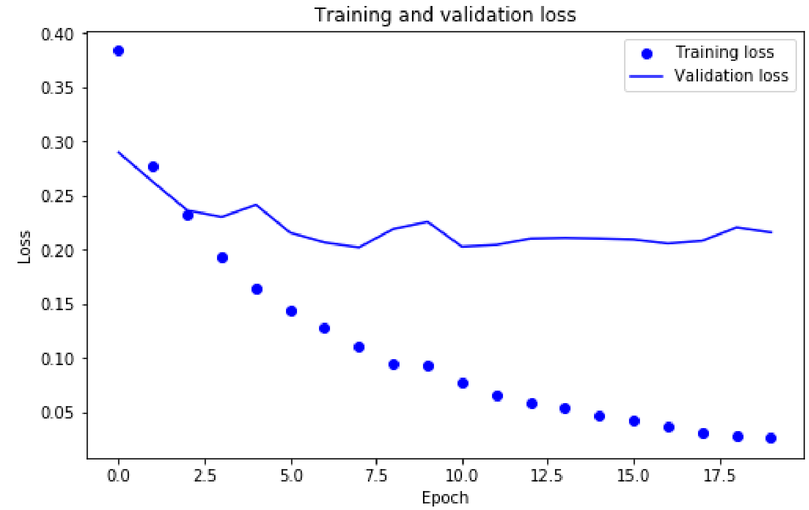
- 2,000 training images; 1,000 validation images
- 25,088 (7x7x512) input feature vector
- Binary output, epochs=20, batch_size=20

Total time: 1:06 minutes (66 seconds)

In [23]:
```
# Visualize accuracy, loss on training and validation sets

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))
accuracy_template = "Epochs: {}, Best acc: {}, Best val_acc: {}"
print(accuracy_template.format(len(epochs), np.max(acc), np.max(val_acc)))

plt.figure(figsize=(18,5))
plt.subplot(121)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(122)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epochs: 20, Best acc: 0.996999979019165, Best val_acc: 0.921999990940094
```





In [24]:
```
# Model evaluation
print("test_features.shape:", test_features.shape)
test_loss, test_acc = mlp.evaluate(test_features, test_labels, steps=50)
print('vgg16_model/custom output MLP - test acc:', test_acc)
```

```
test_features.shape: (1000, 25088)
50/50 [==============================] - 5s 102ms/step - loss: 0.2115 - acc: 0.9120
vgg16_model/custom output MLP - test acc: 0.912
```

Custom trained VGG16 image classifier

Test set accuracy: 91.2%

Evaluation test time

- 1,000 test images
- Total time: 1:39 minutes (5 seconds) - 5 miliseconds/image

In [ ]: