Data pre-processing and data augmentation

In order to make the most of our few training examples, we will "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better.

In Keras this can be done via the keras. preprocessing.image.ImageDataGenerator class. This class allows you to:

* configure random transformations and normalization operations to be done on your image data during training
* instantiate generators of augmented image batches (and their labels) via. flow (data, labels) or. flow\_from\_directory(directory). These generators can then be used with the Keras model methods that accept data generators as inputs, fit\_generator, evaluate\_generator and predict\_generator.

Let's look at an example right away:

from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

rescale=1./255,

shear\_range=0.2,

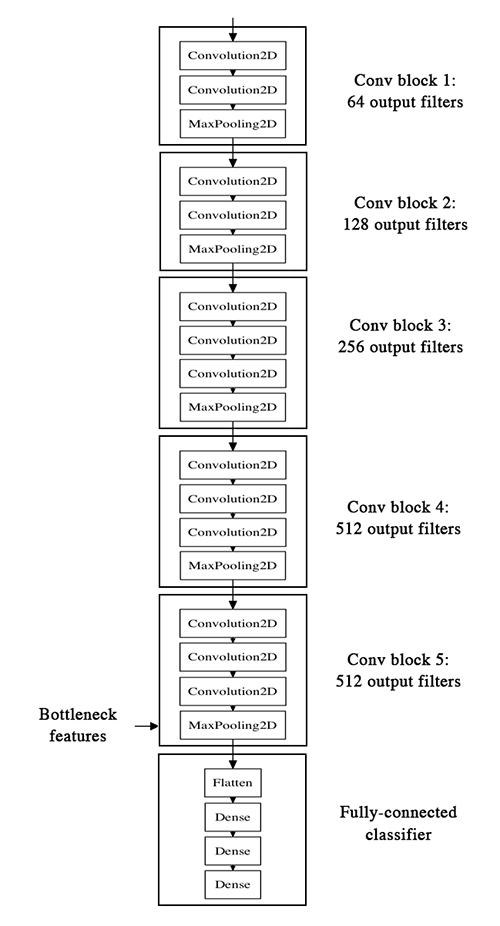
zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest')

* rotation\_range is a value in degrees (0-180), a range within which to randomly rotate pictures
* width\_shift and height\_shift are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally
* rescale is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our models to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255. factor.
* shear\_range is for randomly applying [shearing transformations](https://en.wikipedia.org/wiki/Shear_mapping)
* zoom\_range is for randomly zooming inside pictures
* horizontal\_flip is for randomly flipping half of the images horizontally --relevant when there are no assumptions of horizontal assymetry (e.g. real-world pictures).
* fill\_mode is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

Here's what the VGG16 architecture looks like:



from keras.models import Sequential:

**Sequential** API is used to create **models** layer-by-layer. Functional API is an alternative approach of creating more complex **models**. Functional **model**, you can define multiple input or output that share layers. First, we create an instance for **model** and connecting to the layers to access input and output to the **model**.

**from keras.preprocessing.image import ImageDataGenerator**

**Keras ImageDataGenerator** is a gem! It lets you augment your images in real-time while your model is still training! You can apply any random transformations on each training image as it is passed to the model. This will not only make your model robust but will also save up on the overhead memory!

What does keras preprocessing image do?

The function **will** run after the **image is** resized and augmented. The function should take one argument: one **image** (Numpy tensor with rank 3), and should output a Numpy tensor with the same shape. ... It defaults to the image\_data\_format value found in your **Keras** config file at ~/.

from keras.applications.vgg16 import preprocess\_input

What is import Preprocess\_input?

When you **import preprocess\_input** from the correct module (the module of the selected model, such as from keras.applications.vgg16 **import preprocess\_input** , you have the function that properly transforms a standard image into an appropriate input

Keras works with batches of images. So, the first dimension is used for the number of samples (or images) you have.

When you load a single image, you get the shape of one image, which is (size1,size2,channels).

In order to create a batch of images, you need an additional dimension: (samples, size1,size2,channels)

The preprocess\_input function is meant to adequate your image to the format the model requires.

from keras.models import Model:

What is a model in keras?

, **Keras model** represents the actual neural network **model**. **Keras** provides a two mode to create the **model**, simple and easy to use Sequential API as well as more flexible and advanced Functional API.

from keras.layers import Input, Lambda, Dense, Flatten

What does the dense layer do in keras?

**Dense layer** is the regular deeply connected neural network **layer**. It is most common and frequently used **layer**. **Dense layer does** the below operation on the input and return the output.

What is the use of lambda layer in keras?

The **Lambda layer** exists so that arbitrary expressions can be **used** as a **Layer** when constructing Sequential and Functional API models. **Lambda layers** are best suited for simple operations or quick experimentation. For more advanced **use** cases, follow this guide for subclassing tf. **keras**.

What is flatten layer in keras?

Advertisements. **Flatten** is used to **flatten** the input. For example, if **flatten** is applied to **layer** having input shape as (batch\_size, 2,2), then the output shape of the **layer** will be (batch\_size, 4) **Flatten** has one argument as follows **keras**.**layers**.**Flatten**(data\_format = None)

from glob import glob

**glob** (short for global) is used to return all file paths that match a specific pattern. We can use **glob** to search for a specific file pattern, or perhaps more usefully, search for files where the filename matches a certain pattern by using wildcard characters

vgg = VGG16(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)

Note: each Keras Application expects a specific kind of input preprocessing. For VGG16, call tf.keras.applications.vgg16.preprocess\_input on your inputs before passing them to the model. vgg16.preprocess\_input will convert the input images from RGB to BGR, then will zero-center each color channel with respect to the ImageNet dataset, without scaling.

**Arguments**

* **include\_top**: whether to include the 3 fully-connected layers at the top of the network.
* **weights**: one of None (random initialization), 'imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded.
* **input\_tensor**: optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model.
* **input\_shape**: optional shape tuple, only to be specified if include\_top is False (otherwise the input shape has to be (224, 224, 3) (with channels\_last data format) or (3, 224, 224) (with channels\_first data format). It should have exactly 3 input channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value.
* **pooling**: Optional pooling mode for feature extraction when include\_top is False. - None means that the output of the model will be the 4D tensor output of the last convolutional block. - avg means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor. - max means that global max pooling will be applied.
* **classes**: optional number of classes to classify images into, only to be specified if include\_top is True, and if no weights argument is specified.
* **classifier\_activation**: A str or callable. The activation function to use on the "top" layer. Ignored unless include\_top=True. Set classifier\_activation=None to return the logits of the "top" layer. When loading pretrained weights, classifier\_activation can only be None or "softmax".

# **VGG16 – Convolutional Network for Classification and Detection**



**VGG16** is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to [ILSVRC-2014](http://www.image-net.org/challenges/LSVRC/2014/results). It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU’s.



The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv.  layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalisation (LRN), such normalization does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time.