Computer Vision: 6th lesson – Data Augmentation

Now that you've learned the fundamentals of convolutional classifiers, you're ready to move on to more advanced topics. In this lesson, you'll learn a trick that can give a boost to your image classifiers; it's called *data augmentation*.

The usefulness of cloned data:

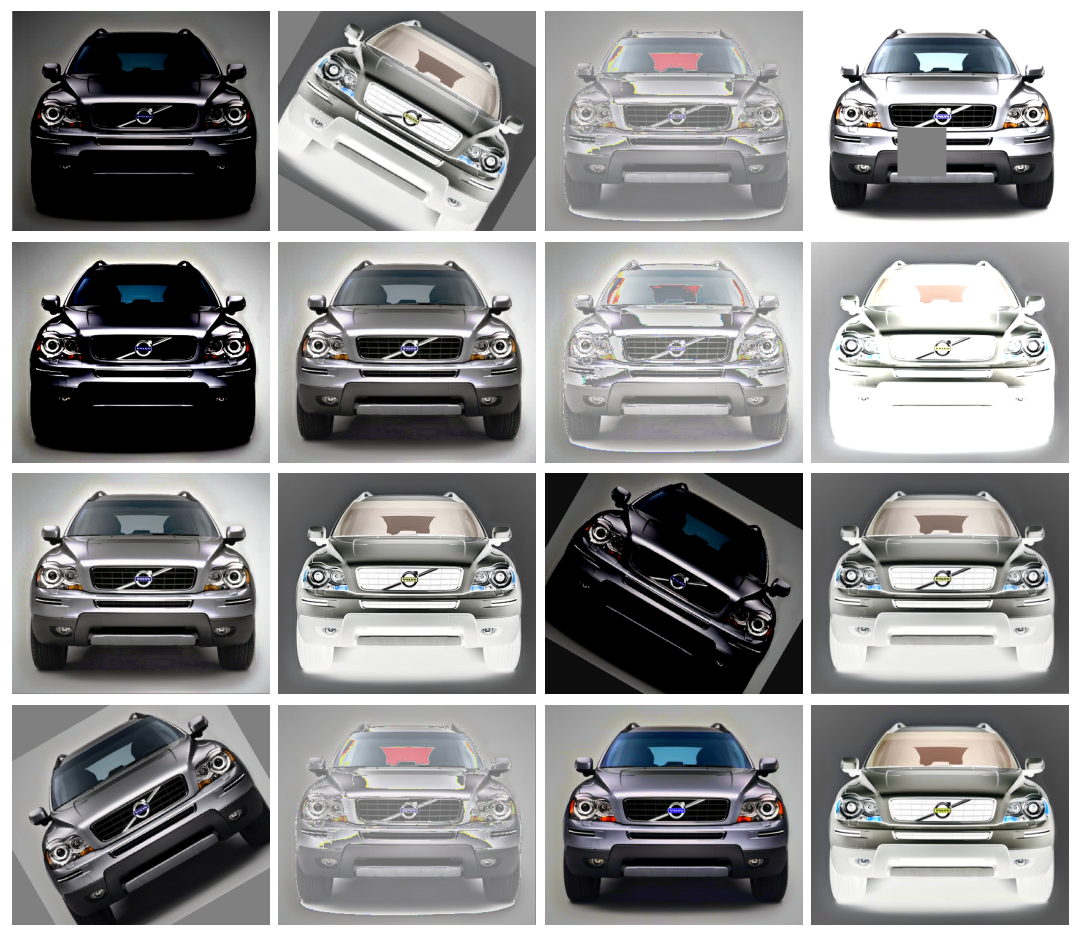
The best way to improve the performance of a machine learning model is to train it on more data. The more examples the model has to learn from, the better it will be able to recognize which differences in images matter and which do not. More data helps the model to *generalize* better.

One easy way of getting more data is to use the data you already have. If we can transform the images in our dataset in ways that preserve the class, we can teach our classifier to ignore those kinds of transformations. For instance, whether a car is facing left or right in a photo doesn't change the fact that it is a Car and not a Truck. So, if we *augment* our training data with flipped images, our classifier will learn that "left or right" is a difference it should ignore.

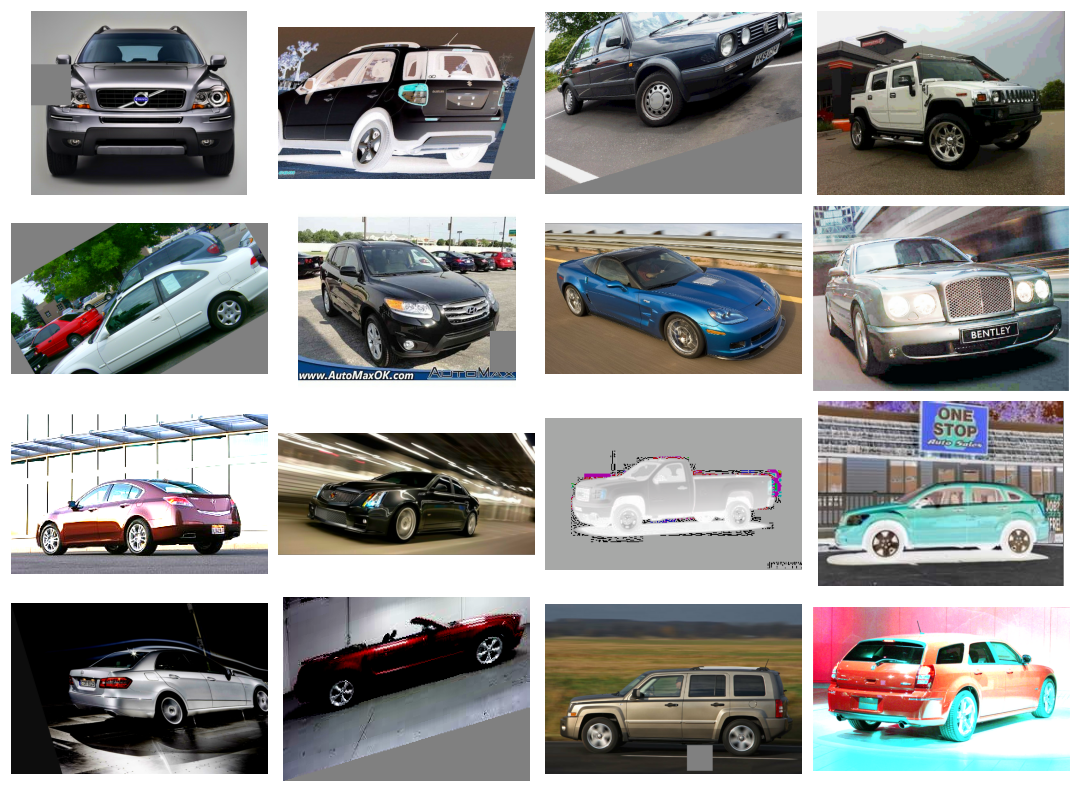
And that's the whole idea behind data augmentation; add in some extra fake data that looks reasonably like the real data and your classifier will improve.

Using data augmentation:

Typically, many kinds of transformation are used when augmenting a dataset. These might include rotating the image, adjusting the color or contrast, warping the image, or many other things, usually applied in combination. Here is a sample of the different ways a single image might be transformed.



Data augmentation is usually done *online*, meaning, as the images are being fed into the network for training. Recall that training is usually done on mini-batches of data. This is what a batch of 16 images might look like when data augmentation is used.



Each time an image is used during training, a new random transformation is applied. This way, the model is always seeing something a little different than what it's seen before. This extra variance in the training data is what helps the model on new data.

It's important to remember though that not every transformation will be useful on a given problem. Most importantly, whatever transformations you use should not mix up the classes. If you were training a digit recognizer, for instance, rotating images would mix up '9's and '6's. In the end, the best approach for finding good augmentations is the same as with most ML problems: try it and see!

***Case study example: Training with data augmentation***

Keras lets you augment your data in two ways. The first way is to include it in the data pipeline with a function like ImageDataGenerator. The second way is to include it in the model definition by using Keras's *preprocessing* layers. This is the approach that we'll take. The primary advantage for us is that the image transformations will be computed on the GPU instead of the CPU, potentially speeding up training.

*# Imports*

import os, warnings

import matplotlib.pyplot as plt

from matplotlib import gridspec

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing import image\_dataset\_from\_directory

*# Reproducability*

def set\_seed(seed=31415):

np.random.seed(seed)

tf.random.set\_seed(seed)

os.environ['PYTHONHASHSEED'] = str(seed)

*#os.environ['TF\_DETERMINISTIC\_OPS'] = '1'*

set\_seed()

*# Set Matplotlib defaults*

plt.rc('figure', autolayout=True)

plt.rc('axes', labelweight='bold', labelsize='large',

titleweight='bold', titlesize=18, titlepad=10)

plt.rc('image', cmap='magma')

warnings.filterwarnings("ignore") *# to clean up output cells*

*# Load training and validation sets*

ds\_train\_ = image\_dataset\_from\_directory(

'../input/car-or-truck/train',

labels='inferred',

label\_mode='binary',

image\_size=[128, 128],

interpolation='nearest',

batch\_size=64,

shuffle=True,

)

ds\_valid\_ = image\_dataset\_from\_directory(

'../input/car-or-truck/valid',

labels='inferred',

label\_mode='binary',

image\_size=[128, 128],

interpolation='nearest',

batch\_size=64,

shuffle=False,

)

*# Data Pipeline*

def convert\_to\_float(image, label):

image = tf.image.convert\_image\_dtype(image, dtype=tf.float32)

return image, label

AUTOTUNE = tf.data.experimental.AUTOTUNE

ds\_train = (

ds\_train\_

.map(convert\_to\_float)

.cache()

.prefetch(buffer\_size=AUTOTUNE)

)

ds\_valid = (

ds\_valid\_

.map(convert\_to\_float)

.cache()

.prefetch(buffer\_size=AUTOTUNE)

)

Found 5117 files belonging to 2 classes.

Found 5051 files belonging to 2 classes.

To illustrate the effect of augmentation, we'll just add a couple of simple transformations to the model from the first step.

from tensorflow import keras

from tensorflow.keras import layers

*# these are a new feature in TF 2.2*

from tensorflow.keras.layers.experimental import preprocessing

pretrained\_base = tf.keras.models.load\_model(

'../input/cv-course-models/cv-course-models/vgg16-pretrained-base',

)

pretrained\_base.trainable = False

model = keras.Sequential([

*# Preprocessing*

preprocessing.RandomFlip('horizontal'), *# flip left-to-right*

preprocessing.RandomContrast(0.5), *# contrast change by up to 50%*

*# Base*

pretrained\_base,

*# Head*

layers.Flatten(),

layers.Dense(6, activation='relu'),

layers.Dense(1, activation='sigmoid'),

])

And now we'll start the training!

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['binary\_accuracy'],

)

history = model.fit(

ds\_train,

validation\_data=ds\_valid,

epochs=30,

verbose=0,

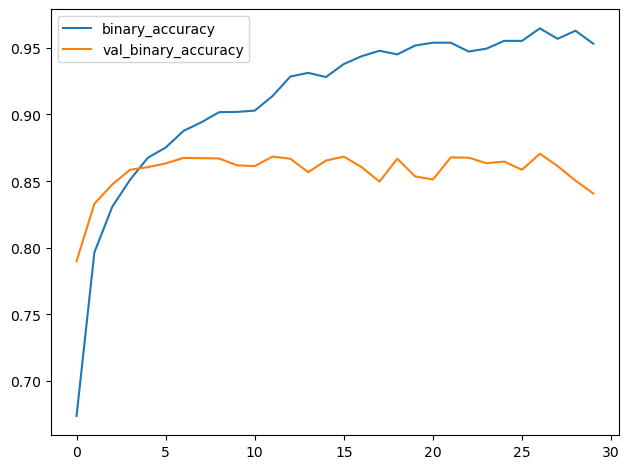
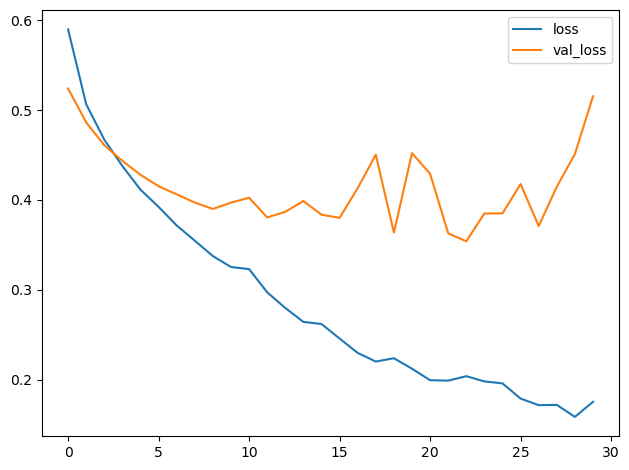
)

import pandas as pd

history\_frame = pd.DataFrame(history.history)

history\_frame.loc[:, ['loss', 'val\_loss']].plot()

history\_frame.loc[:, ['binary\_accuracy', 'val\_binary\_accuracy']].plot();



The training and validation curves in the model from the first step diverged fairly quickly, suggesting that it could benefit from some regularization. The learning curves for this model were able to stay closer together, and we achieved some modest improvement in validation loss and accuracy. This suggests that the dataset did indeed benefit from the augmentation.