Computer Vision: 1st lesson – The Convolutional Classifier

Have you ever wanted to teach a computer to see? In this course, exactly what you will do! In this course, you will:

* Use modern deep learning networks to build an ***image classifier*** with Keras
* Design your own ***custom convnet*** with reusable blocks
* Learn the fundamental ideas behind visual ***feature extraction***
* Master the art of ***transfer learning*** to boost your models
* Utilize ***data augmentation*** to extend your dataset

If you've taken the *Introduction to Deep Learning* course, you'll know everything you need to be successful. Now let's get started!

This course will introduce you to the fundamental ideas of computer vision. Our goal is to learn how a neural network can "understand" a natural image well-enough to solve the same kinds of problems the human visual system can solve.

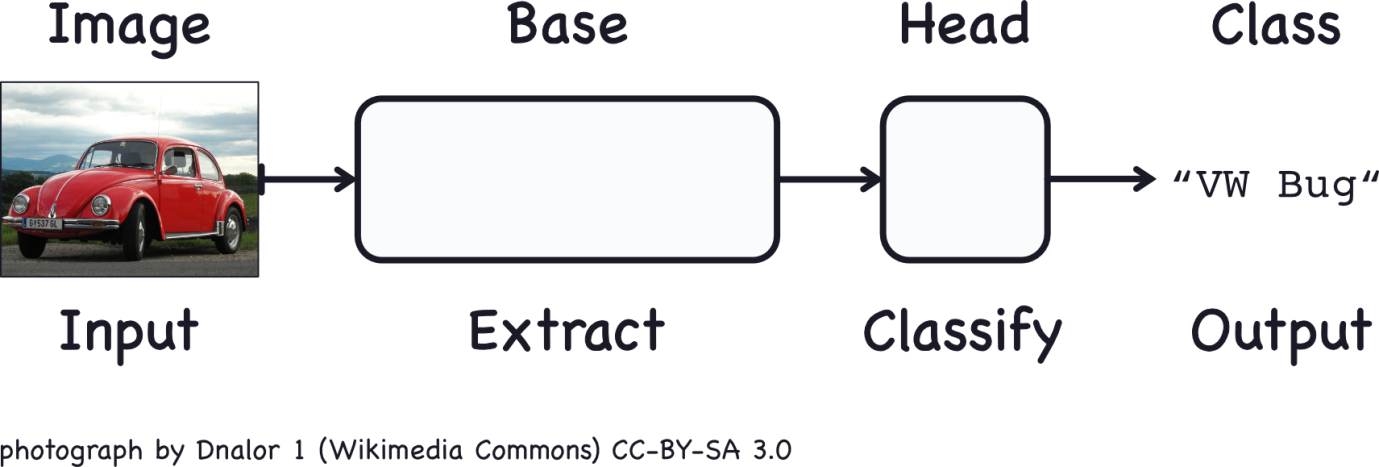
The neural networks that are best at this task are called **convolutional neural networks** (sometimes we say ***convnet*** or **CNN** instead). Convolution is the mathematical operation that gives the layers of a *convnet* their unique structure. In future lessons, you'll learn why this structure is so effective at solving computer vision problems.

We will apply these ideas to the problem of **image classification**; given a picture, can we train a computer to tell us what it's a picture of? You may have seen applications that can identify a species of plant from a photograph. That's an image classifier! In this course, you'll learn how to build image classifiers just as powerful as those used in professional applications.

While our focus will be on image classification, what you'll learn in this course is relevant to every kind of computer vision problem. At the end, you'll be ready to move on to more advanced applications like generative adversarial networks and image segmentation.

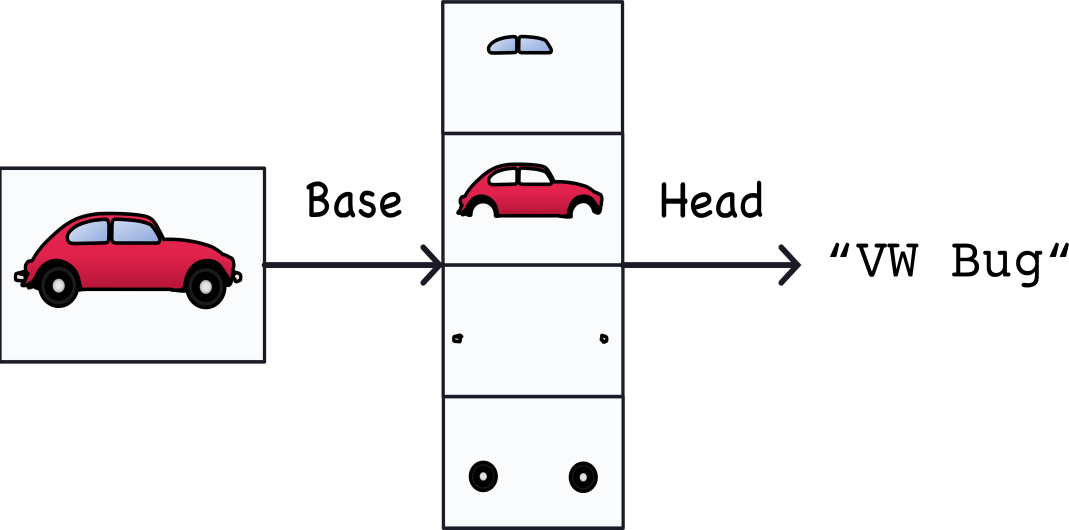
The convolutional classifier:

A *convnet* used for image classification consists of two parts: a **convolutional base** and a **dense head**.



* The base is used to *extract the features* from an image. It is formed primarily of layers performing the convolution operation, but often includes other kinds of layers as well.
* The head is used to *determine the class* of the image. It is formed primarily of dense layers, but might include other layers like dropout.

What do we mean by visual feature? A feature could be a line, a color, a texture, a shape, a pattern, or some complicated combination? The whole process goes something like this.

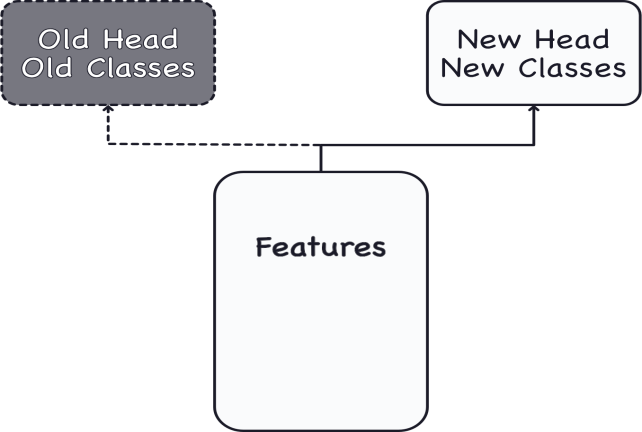


The features actually extracted look a bit different, but it gives the idea.

Training the classifier:

The goal of the network during training is to learn two things; which features to extract from an image or base, and which class goes with what features or head.

These days, *convnets* are rarely trained from scratch. More often, we reuse the base of a pretrained model. To the pretrained base we then attach an untrained head. In other words, we reuse the part of a network that has already learned to do. First, extract features and attach to it some fresh layers to learn, then classify.



* Because the head usually consists of only a few dense layers, very accurate classifiers can be created from relatively least data.
* Reusing a pre-trained model is a technique known as *transfer learning*. It is so effective, that almost every image classifier these days will make use of it.

***Case study example: Train a convnet classifier***

Throughout this course, we're going to be creating classifiers that attempt to solve the following problem; is this a picture of a *Car or of a Truck?* Our dataset is about 10,000 pictures of various automobiles, around half cars and half trucks.

***1st step: Load the data***

This next hidden cell will import some libraries and set up our data pipeline. We have a training split called ds\_train and a validation split called ds\_valid.

*# 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(31415)

*# 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.

Let’s take a look at a few examples from the training set.

import matplotlib.pyplot as plt

***2nd step: Define pre-trained base***

The most commonly used dataset for pre-training is ImageNet, a large dataset of many kind of natural images. Keras includes a variety models pre-trained on ImageNet in its applications module. The pre-trained model we'll use is called **VGG16**.

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

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

)

pretrained\_base.trainable = False

***3rd step: Attach head***

Next, we attach the classifier head. For this example, we'll use a layer of hidden units (the first Dense layer) followed by a layer to transform the outputs to a probability score for class 1, Truck. The Flatten layer transforms the two dimensional outputs of the base into the one dimensional inputs needed by the head.

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

pretrained\_base,

layers.Flatten(),

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

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

])

***4th step: Train the model***

Finally, let's train the model. Since this is a two-class problem, we'll use the binary versions of crossentropy and accuracy. The adam optimizer generally performs well, so we'll choose it as well.

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['binary\_accuracy'],

)

history = model.fit(

ds\_train,

validation\_data=ds\_valid,

epochs=30,

verbose=0,

)

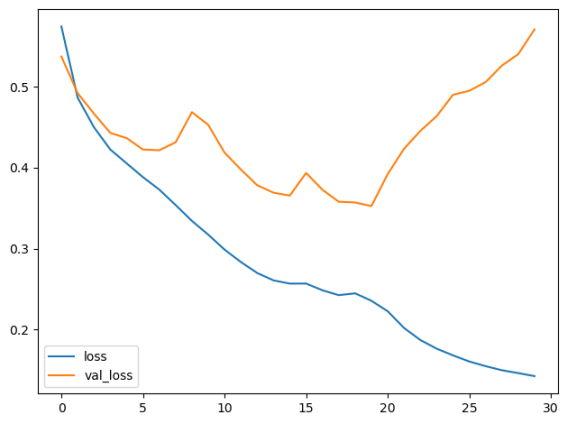
When training a neural network, it's always a good idea to examine the loss and metric plots. The history object contains this information in a dictionary history.history. We can use pandas to convert this dictionary to a DataFrame and plot it with a built-in method.

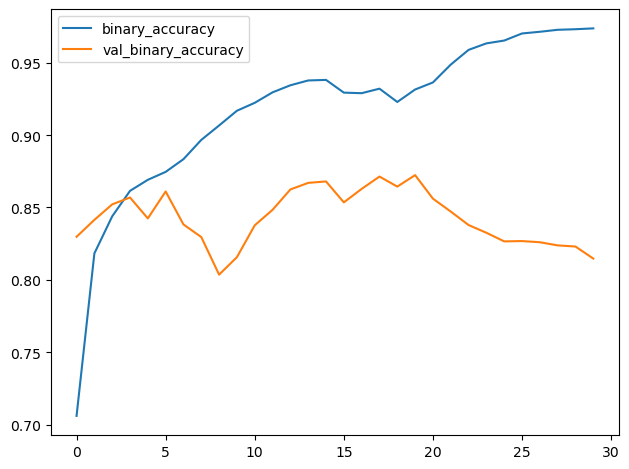
import pandas as pd

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

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

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





In this lesson, we learned about the structure of a *convnet* classifier; a **head** to act as a classifier atop of a **base** which performs the feature extraction.

The head, essentially, is an ordinary classifier like you learned about in the introductory course. For features, it uses those features extracted by the base. This is the basic idea behind convolutional classifiers: that we can attach a unit that performs feature engineering to the classifier itself.

This is one of the big advantages deep neural networks have over traditional machine learning models; given the right network structure, the deep neural net can learn how to engineer the features it needs to solve its problem. For the next few lessons, we'll take a look at how the convolutional base accomplishes the feature extraction. Then, you'll learn how to apply these ideas and design some classifiers of your own.