Computer Vision: 5th lesson – Custom Convolutional Networks

Now that you've seen the layers a *convnet* uses to extract features, it's time to put them together and build a network of your own!

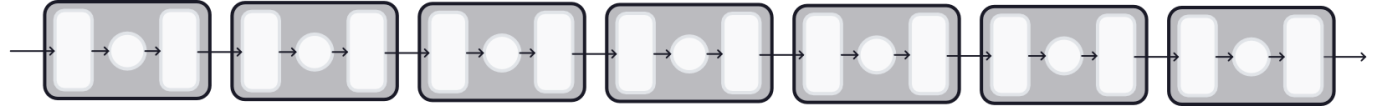
Simple-to-refined:

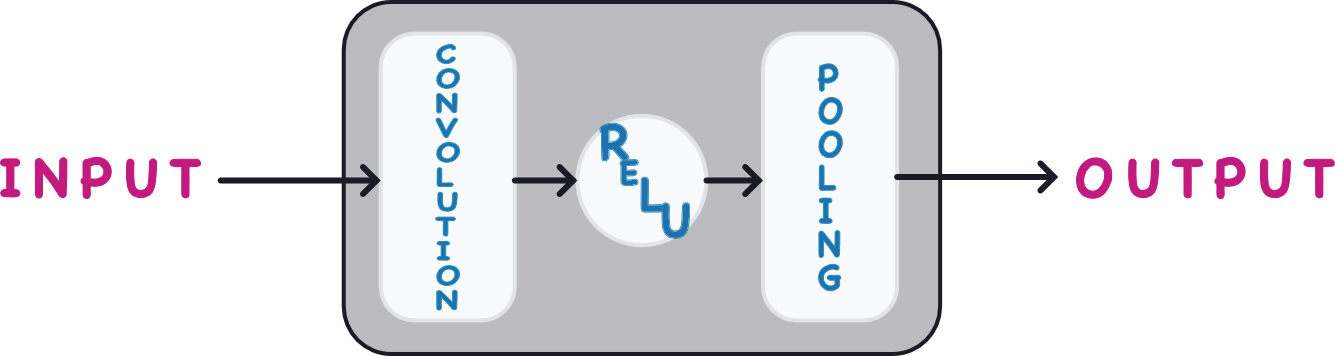
In the last three lessons, we saw how convolutional networks perform feature extraction through three operations: filter, detect, and condense. A single round of feature extraction can only extract relatively simple features from an image, things like simple lines or contrasts. These are too simple to solve most classification problems. Instead, *convnets* will repeat this extraction over and over, so that the features become more complex and refined as they travel deeper into the network.



Convolutional blocks:

It does this by passing them through long chains of convolutional blocks which perform this extraction. These convolutional blocks are stacks of Conv2D and MaxPool2D layers, whose role in feature extraction we learned about in the last few lessons.





***Case study example: Design a convnet***

Let's see how to define a deep convolutional network capable of engineering complex features. In this example, we'll create a Keras Sequence model and then train it on our Cars dataset.

*# 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()

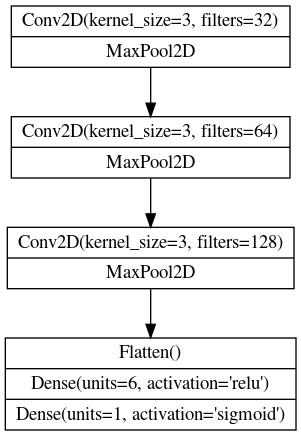
.prefetch(buffer\_size=AUTOTUNE)

)

Found 5117 files belonging to 2 classes.

Found 5051 files belonging to 2 classes.

Here is a diagram of the model we'll use:



Now we'll define the model. See how our model consists of three blocks of Conv2D and MaxPool2D layers (the base) followed by a head of Dense layers. We can translate this diagram more or less directly into a Keras Sequential model just by filling in the appropriate parameters.

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

*# First Convolutional Block*

layers.Conv2D(filters=32, kernel\_size=5, activation="relu", padding='same',

*# give the input dimensions in the first layer*

*# [height, width, color channels(RGB)]*

input\_shape=[128, 128, 3]),

layers.MaxPool2D(),

*# Second Convolutional Block*

layers.Conv2D(filters=64, kernel\_size=3, activation="relu", padding='same'),

layers.MaxPool2D(),

*# Third Convolutional Block*

layers.Conv2D(filters=128, kernel\_size=3, activation="relu", padding='same'),

layers.MaxPool2D(),

*# Classifier Head*

layers.Flatten(),

layers.Dense(units=6, activation="relu"),

layers.Dense(units=1, activation="sigmoid"),

])

model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 128, 128, 32) 2432

max\_pooling2d (MaxPooling2D (None, 64, 64, 32) 0

)

conv2d\_1 (Conv2D) (None, 64, 64, 64) 18496

max\_pooling2d\_1 (MaxPooling (None, 32, 32, 64) 0

2D)

conv2d\_2 (Conv2D) (None, 32, 32, 128) 73856

max\_pooling2d\_2 (MaxPooling (None, 16, 16, 128) 0

2D)

flatten (Flatten) (None, 32768) 0

dense (Dense) (None, 6) 196614

dense\_1 (Dense) (None, 1) 7

=================================================================

Total params: 291,405

Trainable params: 291,405

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Notice in this definition is how the number of filters doubled block-by-block: 32, 64, 128. This is a common pattern. Since the MaxPool2D layer is reducing the size of the feature maps, we can afford to increase the quantity we create.

We can train this model just like the model from Lesson 1: compile it with an optimizer along with a loss and metric appropriate for binary classification.

model.compile(

optimizer=tf.keras.optimizers.Adam(epsilon=0.01),

loss='binary\_crossentropy',

metrics=['binary\_accuracy']

)

history = model.fit(

ds\_train,

validation\_data=ds\_valid,

epochs=40,

verbose=0,

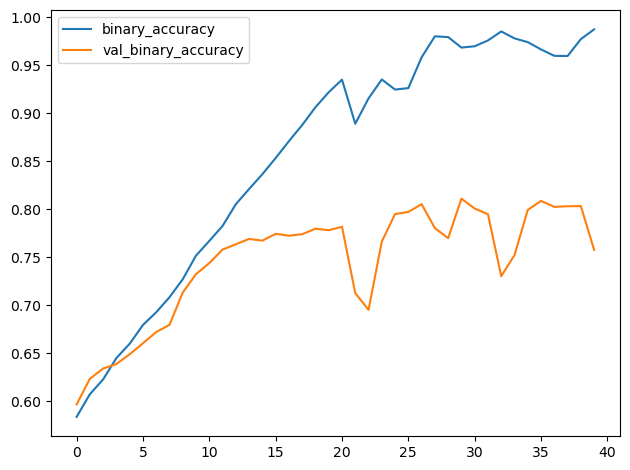
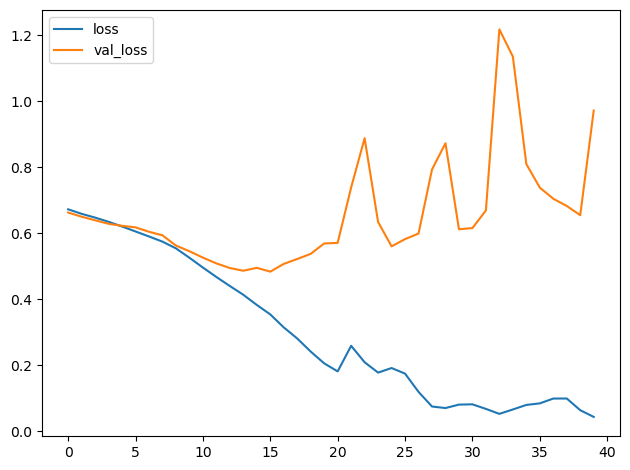
)

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();



This model is much smaller than the VGG16 model from Lesson 1; only 3 convolutional layers versus the 16 of VGG16. It was nevertheless able to fit this dataset fairly well. We might still be able to improve this simple model by adding more convolutional layers, hoping to create features better adapted to the dataset. This is what we'll try in the exercises.