MACHINE LEARNING MASTERY

Machine Learning in OpenCV

7-Day Mini-Course

Stefania Cristina
Adrian Tam

Disclaimer

The information contained within this eBook is strictly for educational purposes. If you wish to apply ideas contained in this eBook, you are taking full responsibility for your actions.

The author has made every effort to ensure the accuracy of the information within this book was correct at time of publication. The author does not assume and hereby disclaims any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from accident, negligence, or any other cause.

Credits

Founder: Jason Brownlee
Author: Stefania Cristina and Adrian Tam
Technical Reviewers: Yoyo Chan, Amy Lam, and Darci Heikkinen

Copyright

Edition: v1.01

Find the latest version of this guide online at: http://MachineLearningMastery.com

Contents

Before We Get Started...

Machine learning is an amazing tool for many tasks. OpenCV is a great library for manipulating images. It would be great if we can put them together.

In this crash course, you will learn from examples how to make use of machine learning and the image processing API from OpenCV to accomplish some goals. This mini-course is intended for practitioners who are already comfortable with programming in Python, know the basic concept of machine learning, and have some background in manipulating images. Let's get started.

This is a long tutorial.

Who Is This Mini-Course for?

Before we get started, let's make sure you are in the right place. The list below provides some general guidelines as to who this course was designed for. Don't panic if you don't match these points exactly, you might just need to brush up in one area or another to keep up.

- ▷ **Developers that know how to write a little code**. This means that it is not a big deal for you to get things done with Python and know how to setup the ecosystem on your workstation (a prerequisite). It does not mean you're a wizard coder, but it does mean you're not afraid to install packages and write scripts.
- ▷ **Developers that know a little machine learning**. This means you know about some common machine learning algorithms like regression or neural networks. It does not mean that you are a machine learning PhD, just that you know the landmarks or know where to look them up.
- ▷ **Developers that know a bit about image processing**. This means you know how to read an image file, how to manipulate a pixel, and how to crop a sub-image. Preferably using OpenCV. It does not mean that you are an image processing expert but you understand that digital images are arrays of pixels.

This mini-course is not a textbook on machine learning, OpenCV, or digital image processing. Rather, it is a project guideline that takes you step-by-step from a developer with minimal knowledge to a developer who can confidently use machine learning in OpenCV.

Mini-Course Overview (What to Expect)

This mini-course is divided into 7 parts. Each lesson was designed to take the average developer about 30 minutes. You might finish some much sooner and other you may choose to go deeper and spend more time. You can complete each part as quickly or as slowly as you like. A comfortable schedule may be to complete one lesson per day over seven days. Highly recommended. The topics you will cover over the next 7 lessons are as follows:

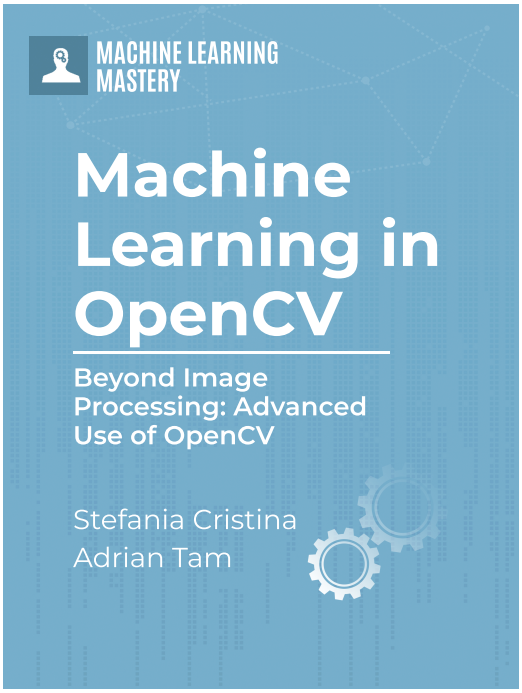
- ▷ **Lesson 1**: Introduction to OpenCV.
- ▷ **Lesson 2**: Read and Display Images Using OpenCV.
- ▷ **Lesson 3**: Finding Circles
- ▷ **Lesson 4**: Extracting Subimages
- ▷ **Lesson 5**: Matching Pennies
- ▷ **Lesson 6**: Building a Coin Classifier
- ▷ **Lesson 7**: Using DNN Module in OpenCV

This is going to be a lot of fun. You're going to have to do some work though, a little reading, a little research and a little programming. You want to learn machine learning and computer vision right?

Here's a tip: All the answers to these lessons can be found on this blog `http://MachineLearningMastery.com`. Use the search feature.

Hang in there, don't give up!

If you would like me to step you through each lesson in great detail (and much more), take a look at my book: **Machine Learning in OpenCV:**



Learn more here:
https://machinelearningmastery.com/machine-learning-opencv/

Lesson 01
Introduction to OpenCV

01

OpenCV is a popular open source library for processing images. It has API bindings in Python, C++, Java, and Matlab. It comes with thousands of functions and implemented many advanced image processing algorithms. If you're using Python, a common alternative to OpenCV is PIL (Python Imaging Library, or its successor, Pillow). Compared to PIL, OpenCV has a richer set of features and is often faster because it is implemented in C++.

This mini-course is to apply machine learning in OpenCV. In later lessons of this mini-course, you will also need TensorFlow/Keras and tf2onnx library in Python.

In this lesson your goal is to install OpenCV.

For a basic Python environment, you can install packages using `pip`. To install OpenCV using `pip`, together with TensorFlow and tf2onnx, you can use:

```
sudo pip install opencv-python tensorflow tf2onnx
```

OpenCV is named as package `opencv-python` in PyPI, but it contains only the "free" algorithms and main modules. There is also the package `opencv-contrib-python`, which also included the "extra" modules. These extra modules are less stable and not well-tested. If you prefer to install the latter, you should use the following commmand instead:

```
sudo pip install opencv-contrib-python tensorflow tf2onnx
```

However, if you're using Anaconda or miniconda environment, the name of the package is just `opencv`, which you can install with the command `conda install`.

To know your OpenCV installation works, you can simply run a simple script and check its version:

```
import cv2
print(cv2.version.opencv_version)
```

Listing 01.1: Checking the version of OpenCV

Learn more about OpenCV, you can start with its online documentation, https://docs.opencv.org/4.x/.

Your Task

Repeat the above code to make sure you have OpenCV correctly installed. Can you also print the version of TensorFlow module by adding a few lines to the code?

Next

In the next lesson, you will use OpenCV to read and display an image.

Lesson 02
Read and Display Image Using OpenCV

02

OpenCV, the Open Source Computer Vision Library, is a powerful tool for image processing and computer vision tasks. But before diving into complex algorithms, let's master the basics: reading and displaying images.

Reading an image using OpenCV is to use `cv2.imread()` function. It takes the path to the image file and returns a NumPy array. The array would usually be three-dimensional, in height×width×channel and each element is an unsigned 8-bit integer. "Channel" is usually BGR (blue-green-red) in OpenCV. But if you prefer to load the image in grayscale, you can add an extra parameter, such as:

```
import cv2

image = cv2.imread("path/filename.jpg", cv2.IMREAD_GRAYSCALE)
print(image.shape)
```

Listing 02.1: Reading an image

The code above will print the array dimension. While we usually describe an image as width×height, array dimension is described as height×width. If the image is read in grayscale, there is only one channel and hence the output will be a two-dimensional array. If you removed the second argument to make it simply `cv2.imread("path/filename.jpg")`, the array should be in shape height×width×3 for three channels of BGR.

To display an image, you can use OpenCV's `cv2.imshow()` function. This will create a window to display the image. But this window will not be shown unless you ask OpenCV to wait for you to interact with the window. Normally, you use:

```
...
cv2.imshow("My image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Listing 02.2: Display an image and wait for a key press

The `cv2.imshow()` takes the window title as its first argument. The image to display should be in BGR channel order. The `cv2.waitKey()` function will wait for your key press for certain milliseconds as specified in its function argument. If zero, it will wait indefinitely. The key

press will be returned as its codepoint in integer, which in this case, you ignored it. As a good practice, you closed the window before this program ends.

Putting it all together:

```
import cv2

image = cv2.imread("path/filename.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imshow("My image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Listing 02.3: Reading and displaying an image

Your Task

Modify the code above to point the path to an image in your disk and try it out. How can you modify the code above to wait until Esc key is pressed but ignoring all other keys? (Hint: The code point for Esc key is 27)

Next

In the next lesson, you will see how to find patterns in an image.

Lesson 03
Finding Circles

03

Since a digital image is represented as a matrix, you can devise your algorithm and check each pixel of the image to identify if some pattern exists in the image. Over the years, a lot of clever algorithms have been invented and you can learn some of them in any digital image processing textbook.

In this mini-course, you're going to solve a simple problem: Given an image with many coins, identify and count a particular type of coin. Coins are circles. To identify circles in an image, one promising algorithm is to use the Hough Circle Transform.

Hough Transform is an algorithm that makes use of *gradient* information of an image. Therefore, it works on a grayscale image rather than a color image. To convert a colored image to grayscale, you can use the `cv2.cvtColor()` function from OpenCV. Because Hough Transform is based on gradient information, it would be sensitive to image noise. Applying Gaussian blur is a usual preprocessing step to reduce noise for Hough Transform. In code, you apply the following for a BGR image you read:

```
...
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (25,25), 1)
```

Listing 03.1: Converting an image to grayscale and apply Gaussian blur

Here the Gaussian blur is applied using a 25×25 kernel. You can use a smaller or larger kernel depending on the level of noise in the image.

Hough Circle Transform is to find circles from an image, using the following function:

```
circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT, dp=1, minDist=100,
                           param1=80, param2=60, minRadius=90, maxRadius=150)
```

Listing 03.2: Hough Circle Transform

There are a lot of parameters. The first argument is the grayscale image and the second is the algorithm to use. The rest are as follows:

- `dp`: Ratio of image resolution to accumulator resolution. Use 1.0 to 2.0 normally.
- `minDist`: Minimum distance between centers of detected circles. The lesser the value, the more false positives you will get.

- ▷ `param1`: This is the threshold for the Canny edge detector
- ▷ `param2`: When algorithm `cv2.HOUGH_GRADIENT` is used, this is the accumulator threshold. The lesser the value, the more false positives.
- ▷ `minRadius` and `maxRadius`: Minimum and maximum circle radius to detect

The returned value from the function `cv2.HoughCircles()` is a NumPy array of circles, represented as rows containing the center coordinates and the radius.

Let's try with a sample image:

- ▷ https://machinelearningmastery.com/wp-content/uploads/2024/01/coins-1.jpg

Download the image, save it as `coins-1.jpg`, and run the following code:

```
import cv2
import numpy as np

image_path = "coins-1.jpg"
img = cv2.imread(image_path)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (25,25), 1)
circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT,
                           dp=1, minDist=100,
                           param1=80, param2=60, minRadius=90, maxRadius=150)
if circles is not None:
    for c in np.uint16(np.round(circles[0])):
        cv2.circle(img, (c[0], c[1]), c[2], (255,0,0), 10)
        cv2.circle(img, (c[0], c[1]), 2, (0,0,255), 20)
cv2.imshow("Circles", img)
cv2.waitKey()
cv2.destroyAllWindows()
```

Listing 03.3: Detecting and marking circles

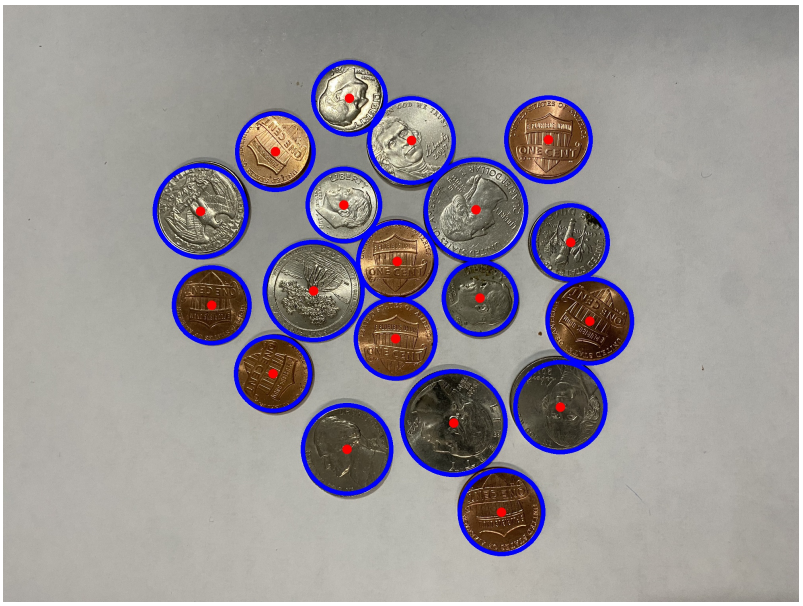


Figure 03.1: Detected circles in blue with centers in red

The code above first rounded off and converted the detected circle data into integers. Then draw the circles on the original image accordingly. From the illustration above, you can see how well the Hough Circle Transform helps you find the coins in the image.

Further Readings

P. E. Hart. "How the Hough Transform was Invented". *IEEE Signal Processing Magazine*, 26(6), Nov. 2009, pp. 18–22. DOI: 10.1109/msp.2009.934181.

R. O. Duda and P. E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". *Comm. ACM*, 15, Jan. 11–15. DOI: 10.1145/361237.361242.

Your Task

The detection is sensitive to the parameters you provided to `cv2.HoughCircles()` function. Try to modify the parameters and see how it results. You can also try to find the best parameters for a different picture, especially one with different lighting conditions or different resolutions.

Next

In the next lesson, you will see how you can extract coins from the image based on the detected circles.

Lesson 04
Extracting Subimages

04

The image you read using OpenCV is a NumPy array of shape height×width×channel. To extract part of the image, you can simply use the NumPy slicing syntax. For example, from a BGR colored image, you can extract the red channel with:

```
red = img[:, :, 2]
```

Listing 04.1: Extracting Red channel

Therefore, to extract part of an image, you can use

```
subimg = img[y0:y1, x0:x1]
```

Listing 04.2: Extracting Red channel

which you will get a rectangular portion of a larger image. Remember that in matrices, you first count the vertical elements (pixels) from top to bottom, then count the horizontal from left to right. Hence you should describe the y-coordinate range first in the slicing syntax.

Let's modify the code in the previous lesson, to extract each coin we found:

```
import cv2
import numpy as np

image_path = "coins-1.jpg"
img = cv2.imread(image_path)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (25,25), 1)
circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT, dp=1, minDist=100,
                           param1=80, param2=60, minRadius=90, maxRadius=150)
for c in np.uint16(np.round(circles[0])):
    x, y, r = c
    subimg = img[y-r:y+r, x-r:x+r]
    cv2.imshow("Coin", subimg)
    cv2.waitKey(0)
cv2.destroyAllWindows()
```

Listing 04.3: Finding the overall accuracy

This code extracts a square subimage for each circle the Hough Transform found. Then the subimage is displayed in a window and wait for your key until the next one is displayed.

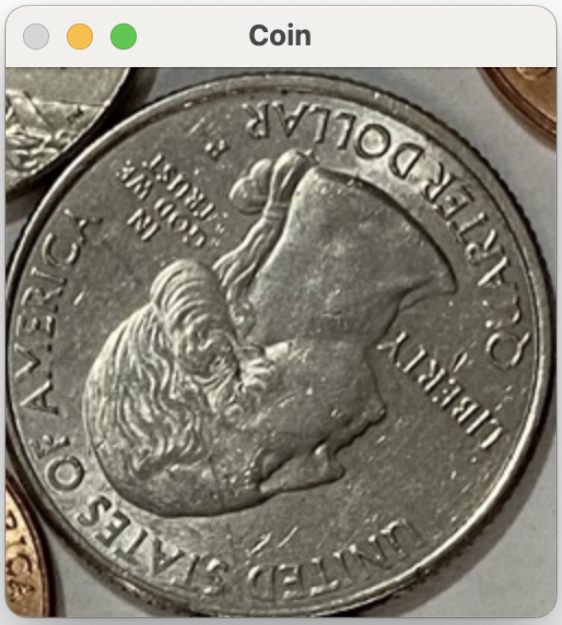


Figure 04.1: OpenCV window displaying a detected coin

Your Task

Run this code. You will notice that each circle detected may be of a different size so are the subimages extracted. How can you resize the subimages to a consistent size before displaying them in the window?

Next

In the next lesson, you will learn to compare the extracted subimages to a reference image.

Lesson 05
Matching Pennies

05

Our task is to identify and count the penny coins from an image. You can find the image of a U.S. penny from Wikipedia:

▷ https://upload.wikimedia.org/wikipedia/commons/thumb/d/d7/Lincoln-Cent-Reverse-sheild.png/240px-Lincoln-Cent-Reverse-sheild.png

With this as the reference image, how can you compare the identified coin with the reference? This is a problem more difficult than it sounds. Used coins can be rusty, dull, or with scratches. The coins in the picture can be rotated. It is not easy to compare pixels and tell if they are the same coin.

A better way is to use keypoint matching algorithms. There are several keypoint algorithms in OpenCV. Let's try with the ORB, which is an invention from the OpenCV team. Downloading the reference image from the link above as `penny.png`, you can extract keypoints and keypoint descriptors with the following code:

```
import cv2

reference_path = "penny.png"
sample = cv2.imread(reference_path)
orb = cv2.ORB_create(nfeatures=500)
kp, ref_desc = orb.detectAndCompute(sample, None)
```

Listing 05.1: Extracting keypoint descriptors using ORB

The tuple `kp` are keypoint objects, but not as important as the `ref_desc` array, which is the *keypoint descriptors*. This is a NumPy array with shape $K \times 32$ for K keypoints detected. Each keypoint's ORB descriptor is a vector of 32 integers.

If you get the descriptor from another image, you can compare it to see if any keypoints match. You should not expect an exact match in descriptors. Instead, you can apply the *Lowe's ratio test* to decide if the keypoints are matched:

```
...
bf = cv2.BFMatcher()
kp, desc = orb.detectAndCompute(coin_gray, None)
matches = bf.knnMatch(ref_desc, desc, k=2)
count = len([1 for m, n in matches if m.distance < 0.80*n.distance])
```

Listing 05.2: Lowe's ratio test

Here, you use a brute-force matcher from `cv2.BFMatcher()` to run the kNN algorithm to get the two nearest neighbors from each reference keypoint to the keypoints from the candidate image. The vector distance is then compared (such that a shorter distance means a better match). The Lowe's ratio test is to tell if the match is good enough. You may try a different constant than 0.8 above. We count the number of good matches, which we will claim the coin is identified if enough good match is found.

Below is the full code, which we will show and identify each coin found using Matplotlib. Since Matplotlib expects a colored image in RGB channels instead of BGR channels, you need to convert the image using `cv2.cvtColor()` before displaying it with `plt.imshow()`.

```
import math
import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path = "coins-1.jpg"
reference_path = "penny.png"

img = cv2.imread(image_path)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (25,25), 1)
circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT, dp=1, minDist=100,
                           param1=80, param2=60, minRadius=90, maxRadius=150)

sample = cv2.imread(reference_path)
orb = cv2.ORB_create(nfeatures=500)
bf = cv2.BFMatcher()
kp, ref_desc = orb.detectAndCompute(sample, None)

plt.figure(2)
N = len(circles[0])
rows = math.ceil(N / 4)
for i, c in enumerate(np.uint16(np.around(circles[0]))):
    x, y, r = c
    coin = img[y-r:y+r, x-r:x+r]
    coin_gray = cv2.cvtColor(coin, cv2.COLOR_BGR2GRAY)
    kp, desc = orb.detectAndCompute(coin_gray, None)
    matches = bf.knnMatch(ref_desc, desc, k=2)
    count = len([1 for m, n in matches if m.distance < 0.80*n.distance])
    plt.subplot(rows, 4, i+1)
    plt.imshow(cv2.cvtColor(coin, cv2.COLOR_BGR2RGB))
    plt.title(f"{count}")
    plt.axis('off')
plt.show()
```

Listing 05.3: Identify penny coins using ORB keypoints

Figure 05.1: Detected coins and the number of keypoints matched

You can see that the number of keypoints matched cannot provide a clear metric to help identify penny coins from other coins. Can you think of other algorithm other than kNN that might be helpful?

Further Readings

Feature Matching. OpenCV.
https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html
How does the Lowe's ratio test work? Stack Overflow.
https://stackoverflow.com/questions/51197091/how-does-the-lowes-ratio-test-work

Your Task

The code above uses ORB keypoints. You can also try with SIFT keypoints. How to modify the code above? Would you see the change in the number of keypoints matched? Also, since ORB features are vectors. Can you build a logistic regression classifier to identify good keypoints such that you do not need to rely on a reference image?

Next

In the next lesson, you will work on a better coin identifier.

Lesson 06
Building a Coin Classifier

06

Provided with an image of a coin and identifying if it is a U.S. penny coin is easy for humans but not so for computers. It is known that the best way to do such classification is by machine learning. You should first extract feature vectors from the image, then run a machine learning algorithm as a classifier to tell if it is a match or not.

Deciding which feature to use is a hard problem itself. But if you use the convolutional neural network, you can let the machine learning algorithm figure out the features by itself. But to train a neural network, you need data. Fortunately, you don't need a lot. Let's see how you can build one.

Firstly, you can get a picture of some coins here:

- ▷ `https://machinelearningmastery.com/wp-content/uploads/2024/01/coins-2.jpg`
- ▷ `https://machinelearningmastery.com/wp-content/uploads/2024/01/coins-3.jpg`

Figure 06.1: Picutre of coins to be extracted as dataset to train a neural network

Save them as `coins-2.jpg` and `coins-3.jpg`, then extract the images of coins, using Hough Circle Transform, and save them into a directory `dataset`:

```
import cv2
import numpy as np

idx = 1
for image_path in ["coins-2.jpg", "coins-3.jpg"]:
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (25,25), 1)
    circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT,
                               dp=1, minDist=100,
                               param1=80, param2=60, minRadius=90, maxRadius=150)

    for c in np.uint16(np.around(circles[0])):
        x, y, r = c
        cv2.imwrite(f"dataset/{idx}.jpg", img[y-r:y+r, x-r:x+r])
        idx += 1
```

Listing 06.1: Extract coins image into a dataset

There are not many images. You can manually tag each image as a penny coin (positive) or not (negative). One way is to move the positive samples into the sub-directory `dataset/pos` and the negative samples into `dataset/neg`. For your convenience, you can find a copy of tagged images in the zip file:

▷ https://machinelearningmastery.com/wp-content/uploads/2024/01/coins.zip

With these, let's build a convolutional neural network for this binary classification problem, using Keras and TensorFlow.

During the data preparation phase, you will read each positive and negative image sample. To make the convolutional neural network simpler, you fixed the input size to 256×256 pixels by resizing the image first. To increase variations in the dataset, you can rotate each image by 90, 180, and 270 degrees and add to the dataset (it is easy since the image samples are all square). Then, you can make use of the `train_test_split()` function from scikit-learn to separate the dataset into training and test sets in a ratio of 7:3.

To create the model, you can use the classific architecture of multiple Conv2D layers with MaxPooling, then followed by Dense layers at the output. Note that it is a binary classification model. Hence at the final output layer, you should use sigmoid activation.

At training, you can simply use a large number of iterations (e.g., `epochs=200`) with early stopping so you don't need to worry about underfitting. You should monitor for the loss evaluated at the test set to be sure not to run into overfitting. In code, this is how you can train a model and save the model as `penny.h5`:

```
import glob
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Flatten
from tensorflow.keras.models import Sequential
```

```
images = []
labels = []
for filename in glob.glob("dataset/pos/*"):
    img = cv2.imread(filename)
    img = cv2.resize(img, (256,256))
    images.append(img)
    labels.append(1)
    for _ in range(3):
        img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
        images.append(img)
        labels.append(1)
for filename in glob.glob("dataset/neg/*"):
    img = cv2.imread(filename)
    img = cv2.resize(img, (256,256))
    images.append(img)
    labels.append(0)
    for _ in range(3):
        img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
        images.append(img)
        labels.append(0)

images = np.array(images)
labels = np.array(labels)
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.3)

model = Sequential([
    Conv2D(16, (5,5), input_shape=(256,256,3), padding="same", activation="relu"),
    MaxPooling2D((2,2)),
    Conv2D(32, (5,5), activation="relu"),
    MaxPooling2D((2,2)),
    Conv2D(64, (5,5), activation="relu"),
    MaxPooling2D((2,2)),
    Conv2D(128, (5,5), activation="relu"),
    Flatten(),
    Dense(256, activation="relu"),
    Dense(1, activation="sigmoid")
])

# Training
earlystopping = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
model.compile(loss="binary_crossentropy", optimizer="adagrad", metrics=["accuracy"])
model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=200, batch_size=32, callbacks=[earlystopping])
model.save("penny.h5")
```

Listing 06.2: Train a penny classifier using Keras

Observe its output, you should easily see the accuracy reach above 90% with not many iterations.

Your Task

Run the code above and create a trained model in `penny.h5`, which you will use in the next lesson. You can modify the model design and see if you can improve the accuracy. Some ideas you can try are: using a different number of Conv2D-MaxPooling layers, different sizes of each layer other than 16-32-64-128 above, or using an activation function other than ReLU.

Next

In the next lesson, you will convert the model you created in Keras to use in OpenCV.

Lesson 07
Using DNN Module in OpenCV

07

Given you have built a convolutional neural network in the previous lesson, you can now use it together with OpenCV. It is easier for OpenCV to consume your model if you first convert it into ONNX format. To do so, you would need the tf2onnx module from Python. Once you installed it, you can convert the model with the following command:

```
python -m tf2onnx.convert --keras penny.h5 --output penny.onnx
```

Listing 07.1: Converting a Keras model into ONNX format

As you saved your Keras model as `penny.h5`, this command will create the file `penny.onnx`.

With the ONNX model file, you can now use the `cv2.dnn` module from OpenCV. The usage is as follows:

```
import cv2

net = cv2.dnn.readNetFromONNX("penny.onnx")
net.setInput(blob)
output = float(net.forward())
```

Listing 07.2: Using the DNN module from OpenCV

That is, you create a neural network object with OpenCV, assign the input, and run the model with `forward()` to fetch the output, which according to how you designed your model, is a floating point value between 0 and 1. As a convention in neural networks, the input is *batched* even if you provide only one input sample. Hence you should add a batch dimension to the images before sending them to the neural network.

Now let's see how you can achieve the goal of counting pennies. You can start by modifying the code from Lesson 05, as follows:

```
import math
import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path = "coins-1.jpg"
```

```
model_path = "penny.onnx"

img = cv2.imread(image_path)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (25,25), 1)
circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT,
                           dp=1, minDist=100,
                           param1=80, param2=60, minRadius=90, maxRadius=150)

plt.figure(2)
N = len(circles[0])
rows = math.ceil(N / 4)
net = cv2.dnn.readNetFromONNX("penny2.onnx")
for i, c in enumerate(np.uint16(np.around(circles[0]))):
    x, y, r = c
    coin = img[y-r:y+r, x-r:x+r]
    coin = cv2.resize(coin, (256,256))
    blob = coin[np.newaxis, ...]
    net.setInput(blob)
    score = float(net.forward())
    plt.subplot(rows, 4, i+1)
    plt.imshow(cv2.cvtColor(coin, cv2.COLOR_BGR2RGB))
    plt.title(f"{score:.2f}")
    plt.axis('off')
plt.show()
```

Listing 07.3: Identify penny coins using ORB keypoints

Instead of using ORB and counting the good keypoints for a match, you used the convolutional neural network to read the sigmoidal output. The output is as follows:

Figure 07.1: Detected coins and the match score by the neural network

You can see quite good a result with this model. All the pennies are identified with a score close to 1. The negative samples are not as good (probably because we did not provide enough

negative samples). Let's use 0.9 as the score cut-off value and rewrite the program to give a count:

```
import math
import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path = "coins-1.jpg"
model_path = "penny.onnx"

img = cv2.imread(image_path)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (25,25), 1)
circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT, dp=1, minDist=100,
                           param1=80, param2=60, minRadius=90, maxRadius=150)
positive = 0
negative = 0
net = cv2.dnn.readNetFromONNX(model_path)
for i, c in enumerate(np.uint16(np.around(circles[0]))):
    x, y, r = c
    coin = img[y-r:y+r, x-r:x+r]
    coin = cv2.resize(coin, (256,256))
    blob = coin[np.newaxis, ...]
    net.setInput(blob)
    score = float(net.forward())
    if score >= 0.9:
        positive += 1
    else:
        negative += 1
print(f"{positive} out of {positive+negative} coins identified are pennies")
```

Listing 07.4: Counting pennies

Your Task

Run the above code to test it out. You can try it with another picture like this one:

▷ https://machinelearningmastery.com/wp-content/uploads/2024/01/coins-4.jpg

Can you modify the code above to report the count by reading images continuously from your webcam?

This was the final lesson.

Final Word Before You Go...

You made it. Well done! Take a moment and look back at how far you have come:

- ▷ You discovered OpenCV as a machine learning library in addition to its image processing capabilities.
- ▷ You made use of OpenCV to extract image features as numerical vectors, which is the basis for any machine learning algorithm.
- ▷ You built a neural network model and converted it to use with OpenCV.
- ▷ Finally, you build a penny counter program. While not perfect, it demonstrates how you can combine OpenCV with machine learning.

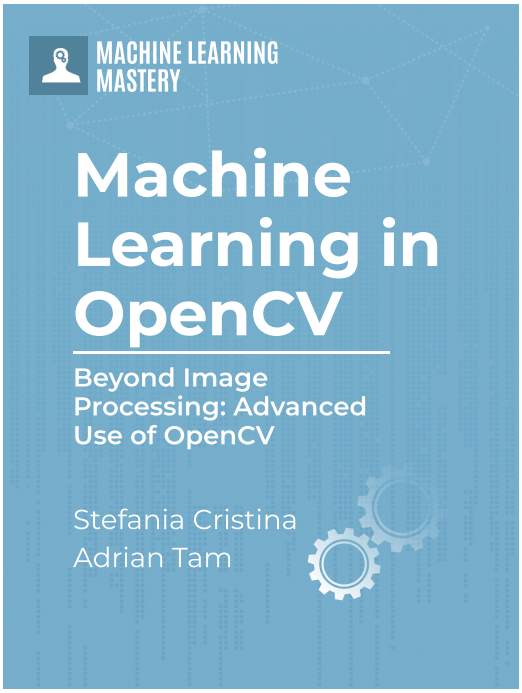
Don't make light of this, you have come a long way in a short amount of time. This is just the beginning of your computer vision journey with machine learning. Keep practicing and developing your skills.

How Did You Go With The Mini-Course?

Did you enjoy this mini-course?
Do you have any questions or sticking points?

Let me know, send me an email at: **jason@MachineLearningMastery.com**

If you would like me to step you through each lesson in great detail (and much more), take a look at my book: **Machine Learning in OpenCV:**



Learn more here:
https://machinelearningmastery.com/machine-learning-opencv/