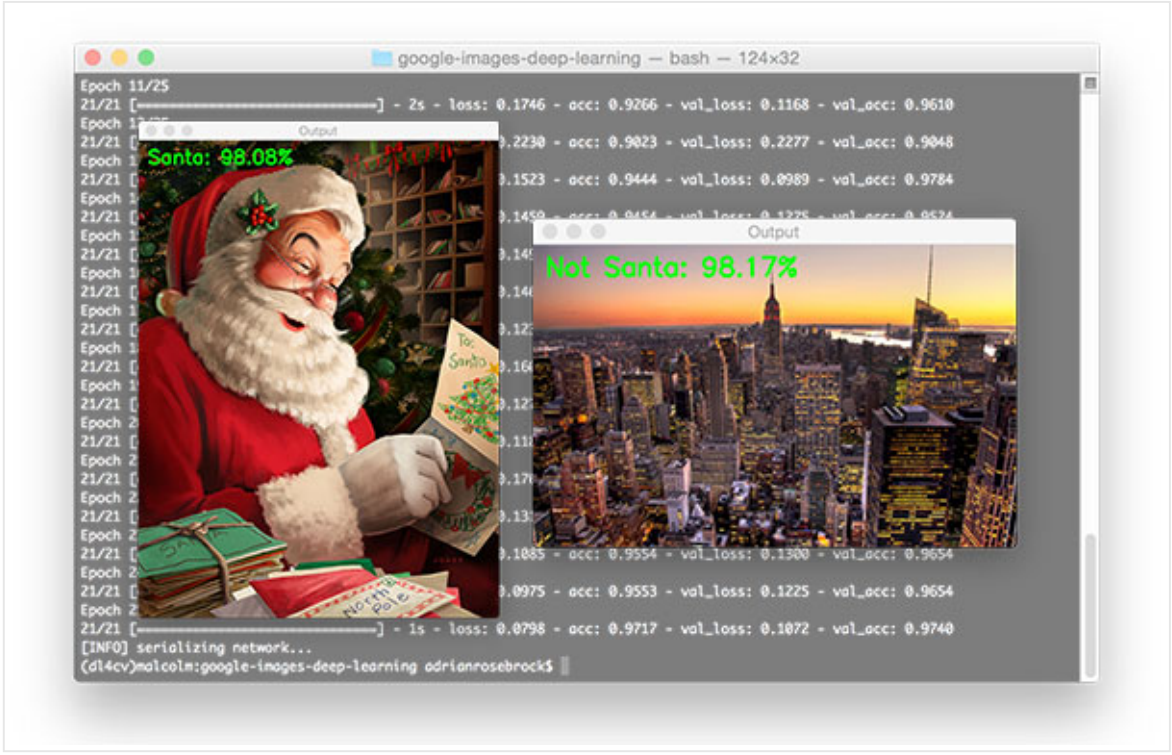
pyimagesearch
be awesome at building image search engines

Image classification with Keras and deep learning

by **Adrian Rosebrock** on December 11, 2017 in **Deep Learning**, **Keras**, **Tutorials**

Click here to download the source code to this post.



The Christmas season holds a special place in my heart.

Not because I'm particularly religious or spiritual. Not because I enjoy cold weather. And certainly not because I relish the taste of eggnog (the consistency alone makes my stomach turn).

Instead, Christmas means a lot to me because of my dad.

As I mentioned in a post a few weeks ago, I had a particularly rough childhood. There was a lot of mental illness in my family. I had to grow up fast in that environment and there were times where I missed out on the innocence of being a kid and living in the moment.

But somehow, through all that struggle, my dad made Christmas a glowing beacon of happiness.

Perhaps one of my favorite memories as a kid was when I was in kindergarten (5-6 years old). I had just gotten off the bus, book bag in hand.

I was walking down our long, curvy driveway where at the bottom of the hill I saw my dad laying out Christmas lights which would later decorate our house, bushes, and trees, transforming our home into a Christmas wonderland.

I took off like a rocket, carelessly running down the driveway (as only a child can), unzipped winter coat billowing behind me as I ran, shouting:

"Wait for me, dad!"

I didn't want to miss out on the decorating festivities.

For the next few hours, my dad patiently helped me u em
out, and then watched as I haphazardly threw the ligh es
my size), ruining any methodical, well-planned decora

Once I was finished he smiled proudly. He didn't need ng
was the best he had ever seen.

This is just one example of the many, *many* times my
else may have been going on in the family).

He probably didn't even know he was crafting a lifelor
me happy.

Each year, when Christmas rolls around, I try to slow

Without my dad, I wouldn't be where I am today — an
childhood.

In honor of the Christmas season, I'd like to dedicate

Even if you're busy, don't have the time, or simply don't care about deep learning (the subject matter of today's tutorial), slow down and give this blog post a read, if for nothing else than for my dad.

I hope you enjoy it.

Looking for the source code to this post?
Jump right to the downloads section.

Image classification with Keras and deep learning

This blog post is part two in our three-part series of building a Not Santa deep learning classifier (i.e., a deep learning model that can recognize if Santa Claus is in an image or not):

1. **Part 1:** Deep learning + Google Images for training data
2. **Part 2:** Training a Santa/Not Santa detector usin
3. **Part 3:** Deploying a Santa/Not Santa deep learni

In the first part of this tutorial, we'll examine our "Santa" and "Not Santa" datasets.

Together, these images will enable us to train a Convolutional Neural Network using Python and Keras to detect if Santa is in an image.

Once we've explored our training images, we'll move on to training the seminal LeNet architecture. We'll use a smaller network architecture to ensure readers without expensive GPUs can still follow along with this tutorial. This will also ensure beginners can understand the fundamentals of deep learning with Convolutional Neural Networks with Keras and Python.

Finally, we'll evaluate our *Not Santa* model on a series of images, then discuss a few limitations to our approach (and how to further extend it).

Our "Santa" and "Not Santa" dataset

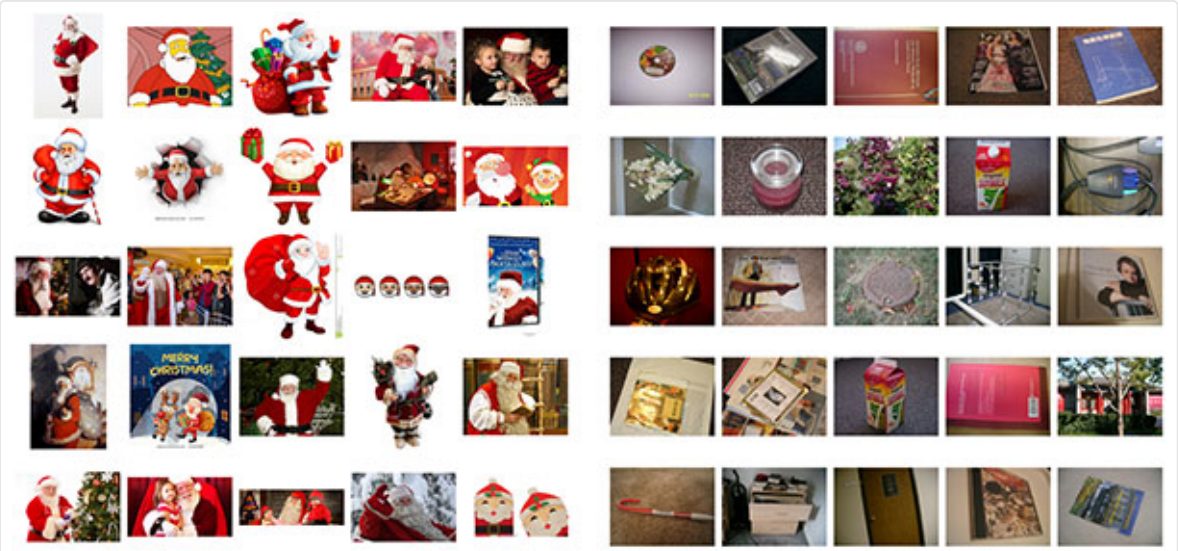


Figure 1: A subset of the dataset for the *Not Santa* detector. Included are (*left*) Santa images extracted from Google Images and (*right*) images sampled from the UKBench dataset.

In order to train our Not Santa deep learning model, we require two sets of images:

- Images *containing* Santa ("Santa").
- Images that *do not contain* Santa ("Not Santa").

Last week we used our Google Images hack to quickly grab training images for deep learning networks.

In this case, we can see a sample of the 461 images containing Santa gathered using technique (**Figure 1**, *left*).

I then randomly sampled 461 images that do not contain Santa (**Figure 1**, *right*) from the UKBench dataset, a collection of ~10,000 images used for building and evaluating Content-based Image Retrieval (CBIR) systems (i.e., image search engines).

Used together, these two image sets will enable us to train our *Not Santa* deep learning model.

Your first image classifier with Convolutional Neural Networks and Keras

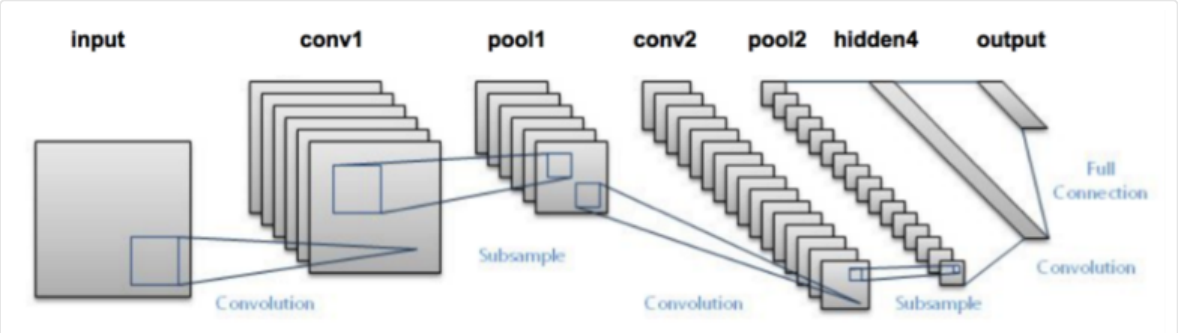


Figure 2: The LeNet architecture consists of two sets of convolutional, activation, and pooling layers, followed by a fully-connected layer, activation, another fully-connected, and finally a softmax classifier. We'll be implementing this network architecture using Keras and Python (image source).

The LetNet architecture is an excellent "first image classifier" for Convolutional Neural Networks. Originally designed for classifying handwritten digits, we can easily extend it to other types of images as well.

This tutorial is meant to be an introduction to image classification using deep learning, Keras, and Python so I will not be discussing the inner-workings of each layer. If you are interested in taking a deep dive into deep learning, please take a look at my book, *Deep Learning for Computer Vision with Python*, where I discuss deep learning in detail (and with lots of code + practical, hands-on implementations as well).

Let's go ahead and define the network architecture. Open up a new file name it `lenet.py` , and insert the following code:

Note: *You'll want to use the* ***"Downloads"*** *section of this post to download the source code + example images before running the code. I've included the code below as a matter of completeness, but you'll want to ensure your directory structure matches mine.*

Image classification with Keras and deep learning — Python

```
1  # import the necessary packages
2  from keras.models import Sequential
3  from keras.layers.convolutional import Conv2D
4  from keras.layers.convolutional import MaxPooling2D
5  from keras.layers.core import Activation
6  from keras.layers.core import Flatten
7  from keras.layers.core import Dense
8  from keras import backend as K
9
10 class LeNet:
11     @staticmethod
12     def build(width, height, depth, classes):
13         # initialize the model
14         model = Sequential()
15         inputShape = (height, width, depth)
16
17         # if we are using "channels first", update the input shape
18         if K.image_data_format() == "channels_first":
19             inputShape = (depth, height, width)
```

Lines 2-8 handle importing our required Python packages. The `Conv2D` class is responsible for performing convolution. We can use the `MaxPooling2D` class for max-pooling operations. As the name suggests, the `Activation` class applies a particular activation function. When we are ready to `Flatten` our network topology into fully-connected, `Dense` layer(s) we can use the respective class names.

The `LeNet` class is defined on **Line 10** followed by the `build` method on **Line 12**. Whenever I defined a new Convolutional Neural Network architecture I like to:

- Place it in its own class (for namespace and organizational purposes)
- Create a static `build` function that builds the architecture itself

The `build` method, as the name suggests, takes a number of parameters, each of which I discuss below:

- `width` : The width of our input images
- `height` : The height of the input images
- `depth` : The number of channels in our input images (`1` for grayscale single channel images, `3` for standard RGB images which we'll be using in this tutorial)
- `classes` : The total number of classes we want to recognize (in this case, two)

We define our `model` on **Line 14**. We use the `Sequential` class since we will be *sequentially* adding layers to the `model` .

Line 15 initializes our `inputShape` using *channels last* ordering (the default for TensorFlow). If you are using Theano (or any other backend to Keras that assumes *channels first* ordering), **Lines 18 and 19** properly update the `inputShape` .

Now that we have initialized our model, we can start adding layers to it:

Image classification with Keras and deep learning — Python

```
21          # first set of CONV => RELU => POOL layers
22          model.add(Conv2D(20, (5, 5), padding="same",
23              input_shape=inputShape))
24          model.add(Activation("relu"))
25          model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

Lines 21-25 creates our first set of `CONV => RELU => POOL` layers.

The `CONV` layer will learn 20 convolution filters, each of which are *5×5*.

We then apply a ReLU activation function followed by *2×2* max-pooling in both the *x* and *y* direction with a stride of two. To visualize this operation, consider a sliding window that "slides" across the activation volume, taking the max operation over each region, while taking a step of two pixels in both the horizontal and vertical direction.

Let's define our second set of `CONV => RELU => POOL` layers:

Image classification with Keras and deep learning — Python

```
27          # second set of CONV => RELU => POOL layers
28          model.add(Conv2D(50, (5, 5), padding="same"))
29          model.add(Activation("relu"))
30          model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

This time we are learning *50 convolutional filters* rather than the *20 convolutional filters* as in the previous layer set. It's common to see the number of `CONV` filters learned ***increase*** the ***deeper*** we go in the network architecture.

Our final code block handles flattening out the volume into a set of fully-connected layers:

Image classification with Keras and deep learning — Python

```
32          # first (and only) set of FC => RELU layers
33          model.add(Flatten())
34          model.add(Dense(500))
35          model.add(Activation("relu"))
36
```

```
37          # softmax classifier
38          model.add(Dense(classes))
39          model.add(Activation("softmax"))
40
41          # return the constructed network architecture
42          return model
```

On **Line 33** we take the output of the preceding `MaxPooling2D` layer and flatten it into a single vector. This operation allows us to apply our dense/fully-connected layers.

Our fully-connected layer contains 500 nodes (**Line 34**) which we then pass through another nonlinear ReLU activation.

Line 38 defines another fully-connected layer, but this one is special — the number of nodes is equal to the number of `classes` (i.e., the classes we want to recognize).

This `Dense` layer is then fed into our softmax classifier which will yield the *probability* for each class.

Finally, **Line 42** returns our fully constructed deep learning + Keras image classifier to the calling function.

Training our Convolutional Neural Network image classifier with Keras

Let's go ahead and get started training our image classifier using deep learning, Keras, and Python.

***Note:** Be sure to scroll down to the **"Downloads"** section to grab the code + training images. This will enable you to follow along with the post and then train your image classifier using the dataset we have put together for you.*

Open up a new file, name it `train_network.py` , and insert the following code (or simply follow along with the code download):

Image classification with Keras and deep learning — Python

```
1  # set the matplotlib backend so figures can be saved in the background
2  import matplotlib
3  matplotlib.use("Agg")
4
5  # import the necessary packages
6  from keras.preprocessing.image import ImageDataGenerator
7  from keras.optimizers import Adam
8  from sklearn.model_selection import train_test_split
9  from keras.preprocessing.image import img_to_array
10 from keras.utils import to_categorical
11 from pyimagesearch.lenet import LeNet
12 from imutils import paths
13 import matplotlib.pyplot as plt
14 import numpy as np
15 import argparse
16 import random
17 import cv2
18 import os
```

On **Lines 2-18** we import required packages. There packages enable us to:

1. Load our image dataset from disk
2. Pre-process the images
3. Instantiate our Convolutional Neural Network
4. Train our image classifier

Notice that on **Line 3** we set the `matplotlib` backend to `"Agg"` so that we can save the plot to disk in the background. This is important if you are using a headless server to train your network (such as an

Azure, AWS, or other cloud instance).

From there, we parse command line arguments:

```
Image classification with Keras and deep learning                    Python
20 # construct the argument parse and parse the arguments
21 ap = argparse.ArgumentParser()
22 ap.add_argument("-d", "--dataset", required=True,
23     help="path to input dataset")
24 ap.add_argument("-m", "--model", required=True,
25     help="path to output model")
26 ap.add_argument("-p", "--plot", type=str, default="plot.png",
27     help="path to output accuracy/loss plot")
28 args = vars(ap.parse_args())
```

Here we have two required command line arguments, `--dataset` and `--model`, as well as an optional path to our accuracy/loss chart, `--plot`.

The `--dataset` switch should point to the directory containing the images we will be training our image classifier on (i.e., the "Santa" and "Not Santa" images) while the `--model` switch controls where we will save our serialized image classifier after it has been trained. If `--plot` is left unspecified, it will default to `plot.png` in this directory if unspecified.

Next, we'll set some training variables, initialize lists, and gather paths to images:

```
Image classification with Keras and deep learning                    Python
30 # initialize the number of epochs to train for, initial learning rate,
31 # and batch size
32 EPOCHS = 25
33 INIT_LR = 1e-3
34 BS = 32
35
36 # initialize the data and labels
37 print("[INFO] loading images...")
38 data = []
39 labels = []
40
41 # grab the image paths and randomly shuffle them
42 imagePaths = sorted(list(paths.list_images(args["dataset"])))
43 random.seed(42)
44 random.shuffle(imagePaths)
```

On **Lines 32-34** we define the number of training epochs, initial learning rate, and batch size.

Then we initialize data and label lists (**Lines 38 and 39**). These lists will be responsible for storing our the images we load from disk along with their respective class labels.

From there we grab the paths to our input images followed by shuffling them (**Lines 42-44**).

Now let's pre-process the images:

```
Image classification with Keras and deep learning                    Python
46 # loop over the input images
47 for imagePath in imagePaths:
48     # load the image, pre-process it, and store it in the data list
49     image = cv2.imread(imagePath)
50     image = cv2.resize(image, (28, 28))
51     image = img_to_array(image)
52     data.append(image)
53
54     # extract the class label from the image path and update the
55     # labels list
56     label = imagePath.split(os.path.sep)[-2]
```

```
57      label = 1 if label == "santa" else 0
58      labels.append(label)
```

This loop simply loads and resizes each image to a fixed *28×28* pixels (the spatial dimensions required for LeNet), and appends the image array to the `data` list (**Lines 49-52**) followed by extracting the class `label` from the `imagePath` on **Lines 56-58**.

We are able to perform this class label extraction since our dataset directory structure is organized in the following fashion:

```
Image classification with Keras and deep learning                    Shell
 1  |--- images
 2  |    |--- not_santa
 3  |    |    |--- 00000000.jpg
 4  |    |    |--- 00000001.jpg
 5  ...
 6  |    |    |--- 00000460.jpg
 7  |    |--- santa
 8  |    |    |--- 00000000.jpg
 9  |    |    |--- 00000001.jpg
10  ...
11  |    |    |--- 00000460.jpg
12  |--- pyimagesearch
13  |    |--- __init__.py
14  |    |--- lenet.py
15  |    |    |--- __init__.py
16  |    |    |--- networks
17  |    |    |    |--- __init__.py
18  |    |    |    |--- lenet.py
19  |--- test_network.py
20  |--- train_network.py
```

Therefore, an example `imagePath` would be:

```
Image classification with Keras and deep learning                    Shell
1 images/santa/00000384.jpg
```

After extracting the `label` from the `imagePath`, the result is:

```
Image classification with Keras and deep learning                    Shell
1 santa
```

I prefer organizing deep learning image datasets in this manner as it allows us to efficiently organize our dataset and parse out class labels without having to use a separate index/lookup file.

Next, we'll scale images and create the training and testing splits:

```
Image classification with Keras and deep learning                    Python
60  # scale the raw pixel intensities to the range [0, 1]
61  data = np.array(data, dtype="float") / 255.0
62  labels = np.array(labels)
63
64  # partition the data into training and testing splits using 75% of
65  # the data for training and the remaining 25% for testing
66  (trainX, testX, trainY, testY) = train_test_split(data,
67      labels, test_size=0.25, random_state=42)
68
69  # convert the labels from integers to vectors
70  trainY = to_categorical(trainY, num_classes=2)
71  testY = to_categorical(testY, num_classes=2)
```

On **Line 61** we further pre-process our input data by scaling the data points from *[0, 255]* (the minimum and maximum RGB values of the image) to the range *[0, 1]*.

We then perform a training/testing split on the data using 75% of the images for training and 25% for testing (**Lines 66 and 67**). This is a typical split for this amount of data.

We also convert labels to vectors using one-hot encoding — this is handled on **Lines 70 and 71**.

Subsequently, we'll perform some data augmentation, enabling us to generate "additional" training data by randomly transforming the input images using the parameters below:

Image classification with Keras and deep learning — Python

```
73 # construct the image generator for data augmentation
74 aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
75     height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
76     horizontal_flip=True, fill_mode="nearest")
```

Data augmentation is covered in depth in the Practitioner Bundle of my new book, *Deep Learning for Computer Vision with Python*.

Essentially **Lines 74-76** create an image generator object which performs random rotations, shifts, flips, crops, and sheers on our image dataset. This allows us to use a smaller dataset and still achieve high results.

Let's move on to training our image classifier using deep learning and Keras.

Image classification with Keras and deep learning — Python

```
78 # initialize the model
79 print("[INFO] compiling model...")
80 model = LeNet.build(width=28, height=28, depth=3, classes=2)
81 opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
82 model.compile(loss="binary_crossentropy", optimizer=opt,
83     metrics=["accuracy"])
84
85 # train the network
86 print("[INFO] training network...")
87 H = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
88     validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
89     epochs=EPOCHS, verbose=1)
90
91 # save the model to disk
92 print("[INFO] serializing network...")
93 model.save(args["model"])
```

We've elected to use LeNet for this project for two reasons:

1. LeNet is a small Convolutional Neural Network that is easy for beginners to understand
2. We can easily train LeNet on our Santa/Not Santa dataset without having to use a GPU
3. If you want to study deep learning in more depth (including ResNet, GoogLeNet, SqueezeNet, and others) please take a look at my book, *Deep Learning for Computer Vision with Python.*

We build our LeNet model along with the `Adam` optimizer on **Lines 80-83**. Since this is a two-class classification problem we'll want to use binary cross-entropy as our loss function. If you are performing classification with > *2* classes, be sure to swap out the `loss` for `categorical_crossentropy` .

Training our network is initiated on **Lines 87-89** where we call `model.fit_generator` , supplying our data augmentation object, training/testing data, and the number of epochs we wish to train for.

Line 93 handles serializing the model to disk so we later use our image classification *without* having to retrain it.

Finally, let's plot the results and see how our deep learning image classifier performed:

Image classification with Keras and deep learning — Python

```python
# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
N = EPOCHS
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy on Santa/Not Santa")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

Using matplotlib, we build our plot and save the plot to disk using the `--plot` command line argument which contains the path + filename.

To train the *Not Santa* network (after using the ***"Downloads"*** section of this blog post to download the code + images), open up a terminal and execute the following command:

Image classification with Keras and deep learning — Shell

```
$ python train_network.py --dataset images --model santa_not_santa.model
Using TensorFlow backend.
[INFO] loading images...
[INFO] compiling model...
[INFO] training network...
Epoch 1/25
1s - loss: 0.7623 - acc: 0.4926 - val_loss: 0.6342 - val_acc: 0.4978
Epoch 2/25
1s - loss: 0.6022 - acc: 0.6705 - val_loss: 0.4542 - val_acc: 0.8182
Epoch 3/25
1s - loss: 0.4749 - acc: 0.8070 - val_loss: 0.4767 - val_acc: 0.7532
...
Epoch 23/25
1s - loss: 0.1085 - acc: 0.9554 - val_loss: 0.1300 - val_acc: 0.9654
Epoch 24/25
1s - loss: 0.0975 - acc: 0.9553 - val_loss: 0.1225 - val_acc: 0.9654
Epoch 25/25
1s - loss: 0.0798 - acc: 0.9717 - val_loss: 0.1072 - val_acc: 0.9740
[INFO] serializing network...
```

As you can see, the network trained for 25 epochs and we achieved high accuracy (***97.40%*** testing accuracy) and low loss that follows the training loss, as is apparent from the plot below:

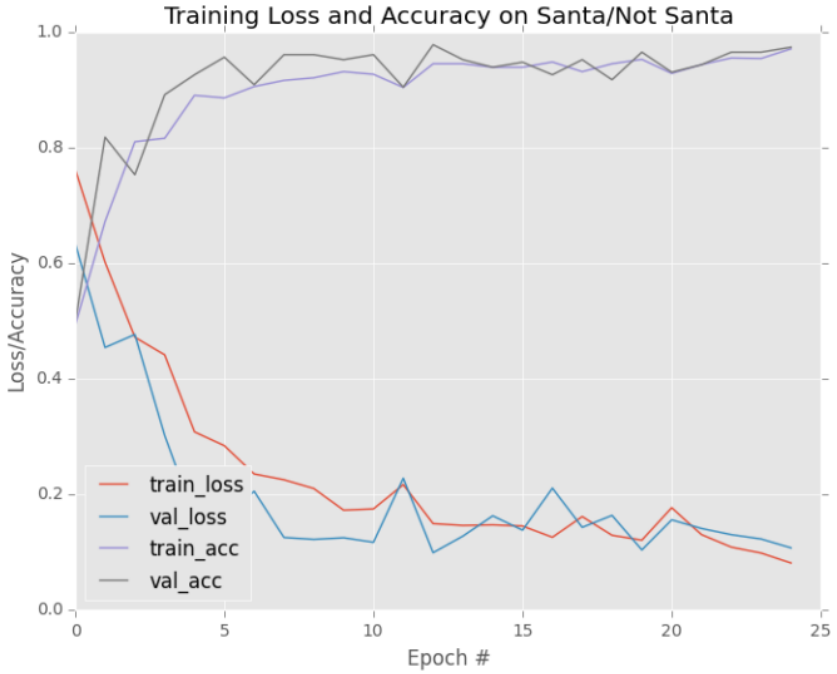


Figure 3: Training our image classifier using deep learning, Keras, and Python.

Evaluating our Convolutional Neural Network image classifier

The next step is to evaluate our *Not Santa* model on example images *not* part of the training/testing splits.

Open up a new file, name it `test_network.py` , and let's get started:

Image classification with Keras and deep learning — Python

```
# import the necessary packages
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import numpy as np
import argparse
import imutils
import cv2
```

On **Lines 2-7** we import our required packages. Take special notice of the `load_model` method — this function will enable us to load our serialized Convolutional Neural Network (i.e., the one we just trained in the previous section) from disk.

Next, we'll parse our command line arguments:

Image classification with Keras and deep learning — Python

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,
    help="path to trained model model")
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
args = vars(ap.parse_args())
```

We require two command line arguments: our `--model` and a input `--image` (i.e., the image we are going to classify).

From there, we'll load the image and pre-process it:

Image classification with Keras and deep learning — Python

```
# load the image
```

```
18 image = cv2.imread(args["image"])
19 orig = image.copy()
20
21 # pre-process the image for classification
22 image = cv2.resize(image, (28, 28))
23 image = image.astype("float") / 255.0
24 image = img_to_array(image)
25 image = np.expand_dims(image, axis=0)
```

We load the `image` and make a copy of it on **Lines 18 and 19**. The copy allows us to later recall the original image and put our label on it.

Lines 22-25 handling scaling our image to the range *[0, 1]*, converting it to an array, and addding an extra dimension (**Lines 22-25**).

As I explain in my book, *Deep Learning for Computer Vision with Python*, we train/classify images in *batches* with CNNs. Adding an extra dimension to the array via `np.expand_dims` allows our image to have the shape `(1, width, height, 3)`, assuming channels last ordering.

If we forget to add the dimension, it will result in an error when we call `model.predict` down the line.

From there we'll load the *Not Santa* image classifier model and make a prediction:

Image classification with Keras and deep learning — Python

```
27 # load the trained convolutional neural network
28 print("[INFO] loading network...")
29 model = load_model(args["model"])
30
31 # classify the input image
32 (notSanta, santa) = model.predict(image)[0]
```

This block is pretty self-explanatory, but since this is where the heavy lifting of this script is performed, let's take a second and understand what's going on under the hood.

We load the *Not Santa* model on **Line 29** followed by making a prediction on **Line 32**.

And finally, we'll use our prediction to draw on the `orig` image copy and display it to the screen:

Image classification with Keras and deep learning — Python

```
34 # build the label
35 label = "Santa" if santa > notSanta else "Not Santa"
36 proba = santa if santa > notSanta else notSanta
37 label = "{}: {:.2f}%".format(label, proba * 100)
38
39 # draw the label on the image
40 output = imutils.resize(orig, width=400)
41 cv2.putText(output, label, (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,
42     0.7, (0, 255, 0), 2)
43
44 # show the output image
45 cv2.imshow("Output", output)
46 cv2.waitKey(0)
```

We build the label (either "Santa" or "Not Santa") on **Line 35** and then choose the corresponding probability value on **Line 36**.

The `label` and `proba` are used on **Line 37** to build the label text to show at the image as you'll see in the top left corner of the output images below.

We resize the images to a standard width to ensure it will fit on our screen, and then put the label text on the image (**Lines 40-42**).

Finally, on **Lines 45**, we display the output image until a key has been pressed (**Line 46**).

Let's give our *Not Santa* deep learning network a try:

```
Image classification with Keras and deep learning                    Python
1 $ python test_network.py --model santa_not_santa.model \
2     --image examples/santa_01.png
```

Figure 4: Santa has been detected with 98% confidence using our Keras image classifer.

By golly! Our software thinks it is good ole' St. Nick, so it really must be him!

Let's try another image:

```
Image classification with Keras and deep learning                    Shell
1 $ python test_network.py --model santa_not_santa.model \
2     --image examples/santa_02.png
```

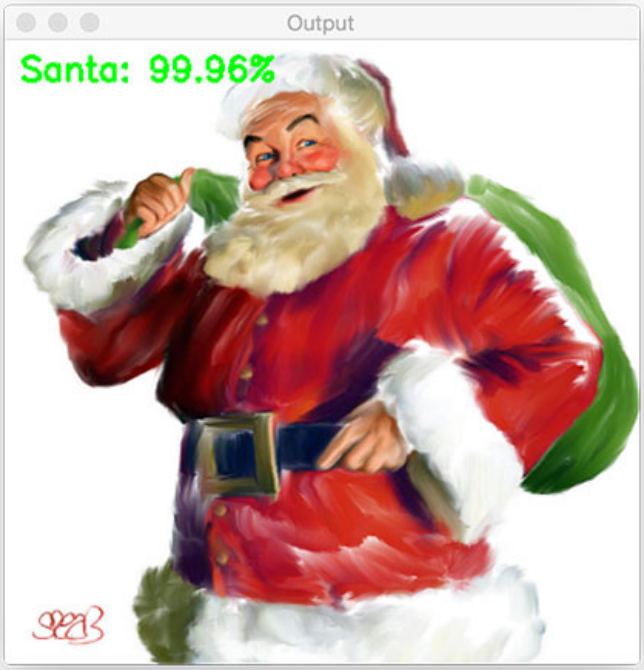


Figure 5: Using a Convolutional Neural Network, Keras, and Python to perform image classification.

Santa is correctly detected by the Not Santa detector and it looks like he's happy to be delivering some toys!

Now, let's perform image classification on an image that *does not* contain Santa:

Image classification with Keras and deep learning — Shell

```
$ python test_network.py --model santa_not_santa.model \
    --image examples/manhattan.png
```

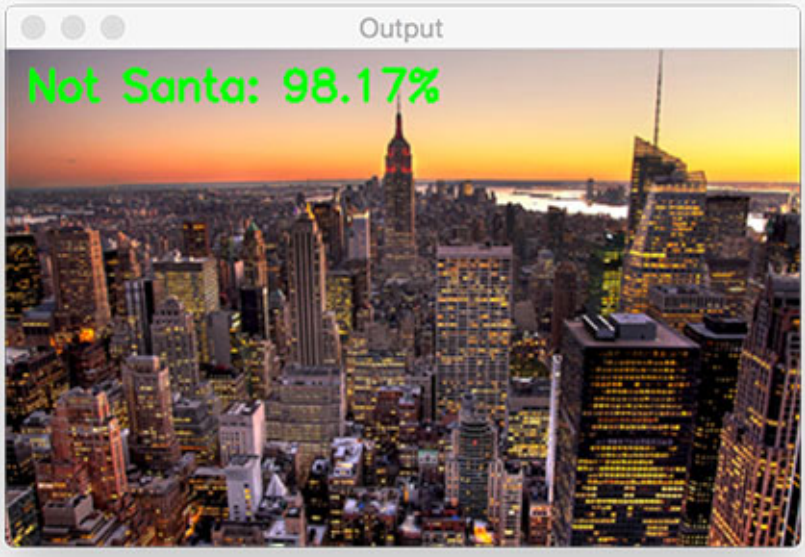


Figure 6: Image classification with deep learning.

It looks like it's too bright out for Santa to be flying through the sky and delivering presents in this part of the world yet (New York City) — he must still be in Europe at this time where night has fallen.

Speaking of the night and Christmas Eve, here is an image of a cold night sky:

Image classification with Keras and deep learning — Shell

```
$ python test_network.py --model santa_not_santa.model \
    --image examples/night_sky.png
```

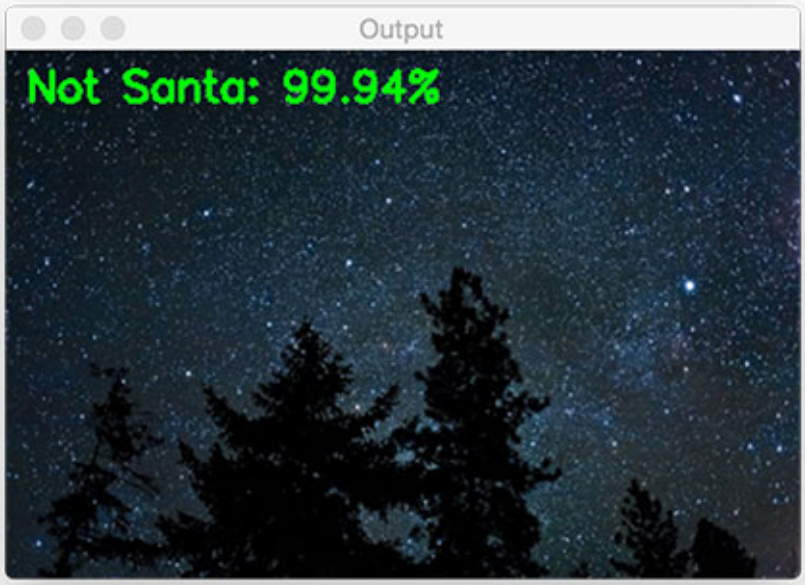


Figure 7: Santa isn't present in this part of the Christmas Eve sky, but he's out there somewhere.

But it must be too early for St. Nicholas. He's not in the above image either.

But don't worry!

As I'll show next week, we'll be able to detect him sneaking down the chimney and delivering presents with a Raspberry Pi.

Limitations of our deep learning image classification model

There are a number of limitations to our image classifier.

The first one is that the *28×28* pixel images are quite small (the LeNet architecture was originally designed to recognize handwritten digits, not objects in photos).

For some example images (where Santa is already small), resizing the input image down to *28×28* pixels effectively reduces Santa down to a tiny red/white blob that is only 2-3 pixels in size.

In these types of situations it's likely that our LeNet model is just predicting when there is a significant amount of red and white localized together in our input image (and likely green as well, as red, green, and white are Christmas colors).

State-of-the-art Convolutional Neural Networks normally accept images that are 200-300 pixels along their maximum dimension — these larger images would help us build a more robust Not Santa classifier. However, using larger resolution images would also require us to utilize a deeper network architecture, which in turn would mean that we need to gather additional training data and utilize a more computationally expensive training process.

This is certainly a possibility, but is also outside the scope of this blog post.

Therefore, If you want to improve our *Not Santa* app I would suggest you:

1. Gather additional training data (ideally, 5,000+ example "Santa" images).
2. Utilize higher resolution images during training. I imagine *64×64* pixels would produce higher accuracy. *128×128* pixels would likely be ideal (although I have not tried this).
3. Use a deeper network architecture during training.
4. Read through my book, *Deep Learning for Computer Vision with Python*, where I discuss training Convolutional Neural Networks on your own custom datasets in more detail.

Despite these limitations, I was *incredibly surprised* with how well the *Not Santa* app performed (as I'll discuss next week). I was expecting a decent number of false-positives but the network was surprisingly robust given how small it is.

Summary

In today's blog post you learned how to train the seminal LeNet architecture on a series of images containing "Santa" and "Not Santa", with our end goal being to build an app similar to HBO's Silicon Valley *Not Hotdog* application.

We were able to gather our "Santa" dataset (~460 images) by following our previous post on gathering deep learning images via Google Images.

The "Not Santa" dataset was created by sampling the UKBench dataset (where no images contain Santa).

We then evaluated our network on a series of testing images — in each case our *Not Santa* model correctly classified the input image.

In our next blog post, we'll deploy our trained Convolutional Neural Network to the Raspberry Pi to finish building our *Not Santa* app.

What now?

Now that you've learned how to train your first Convolutional Neural Network, I'm willing to bet that you're interested in:

- Mastering the fundamentals of machine learning and neural networks
- Studying deep learning in more detail
- Training your own Convolutional Neural Networks from scratch

If so, you'll want to take a look at my new book, *Deep Learning for Computer Vision with Python*.

Inside the book you'll find:

- Super-practical **walkthroughs**
- **Hands-on tutorials** (with lots of code)
- Detailed, thorough guides to help you **replicate state-of-the-art results** from seminal deep learning publications.

To learn more about my new book (and start your journey to deep learning mastery), *just click here*.

Otherwise, be sure to enter your email address in the form below to be notified when new deep learning posts are published here on PyImageSearch.

Downloads:

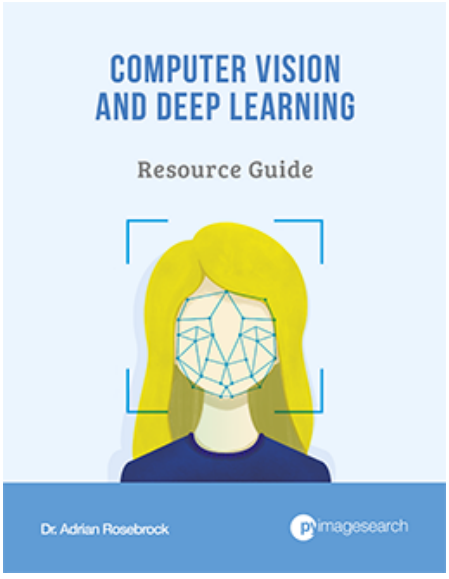
If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

Your email address

DOWNLOAD THE CODE!

Resource Guide (it's totally free).



Enter your email address below to get my **free 17-page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF**. Inside you'll find my hand-picked tutorials, books, courses, and Python libraries to help you master computer vision and deep learning!

Your email address

DOWNLOAD THE GUIDE!

convolutional neural network, **deep learning**, **image classification**, **keras**, **machine learning**, **neural nets**, **python**

‹ How to create a deep learning dataset using Google Images

Keras and deep learning on the Raspberry Pi ›

217 Responses to *Image classification with Keras and deep learning*

John Beale December 11, 2017 at 1:07 pm #

REPLY

Very clearly presented, as always! Looking forward to the next installment on deploying on R-Pi. The question I'm most interested in is what the tradeoff looks like between image resolution and processing time. For a given network architecture, is there some equation that can tell you, for a [x,y] pixel input, that it will take N FLOPs (or on given hardware, T seconds) to do the forward propagation through the network? I understand that there is a separate question of mAP scores versus input resolution.

Adrian Rosebrock December 11, 2017 at 4:33 pm #

REPLY

It's not an exact computation because you need to consider there are many parameters to consider. There is the speed of the physical hardware itself. Then you have the libraries used to accelerate learning. On top of that is the actual network architecture. Is the network fully convolutional? Or do you have FC layers in there? All of these choices have an impact and really the best way to benchmark is with system timings. Quite an exhaustive one can be found here.

Ayesha shakeel December 11, 2017 at 3:40 pm #

REPLY

Hy Adrian, hope you're having a great time. Can you please give me a Christmas gift by helping me resolve this issue? i would be very grateful
The issue is: I am following your tutorial to install open CV 3 and python 2.7 on my raspberry pi3. here's the link to the tutorial https://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/
I have followed all the steps and i get the outputs at each step described by you but when i come to the step of compiling cv, using make -j4(i have tried simple make also), i get this error "fatal error : stdlib.h >> no such file or directory found".
i have checked and i have std library in the path /usr/include/ c++/6/stdlib.h, still why does it give this error. please please help me resolve it, my project's deadline is approaching and i need to have open CV installed for that. Thank you!
regards
Ayesha

Adrian Rosebrock December 11, 2017 at 4:35 pm #

REPLY

This sounds like an issue with the precompiled headers. Delete your "build" directory, re-create it, and re-run CMake, but this time include the following flag:

```
-D ENABLE_PRECOMPILED_HEADERS=OFF
```

The compile might take slightly longer but it should resolve the issue.

sam December 11, 2017 at 11:02 pm #

REPLY

what would be the change for it to do image classification on 4 classes?

Adrian Rosebrock December 12, 2017 at 9:07 am #

REPLY

You would use categorical cross-entropy as your loss function and you would change `classes=4` in the LeNet instantiation. If you're just getting started with deep learning, please take a look at my new book, Deep Learning for Computer Vision with Python. This book will help you learn the fundamentals of deep learning for computer vision applications. Be sure to take a look!

RomRoc December 12, 2017 at 4:22 am #

REPLY

Hello Adrian, as always an incredibly useful post. You should know that I started learning opencv + computer vision + deep learning from 2 months, and your blog was the starting point of my study.

It could be nice if next year you will make a post for object detection using deep learning.

Thanks for your work, and have a great Christmas holiday!

Adrian Rosebrock December 12, 2017 at 8:59 am #

REPLY

It's great you have been enjoying the PyImageSearch blog! Congratulations on your computer vision + deep learning journey.

I actually have a blog post on deep learning object detection. I'm covering how to train your own custom deep learning object detectors inside Deep Learning for Computer Vision with Python.

RomRoc December 15, 2017 at 7:27 am #

REPLY

Amazing post, this is the only one post I found in Internet that describes properly opencv functionality for deep learning object detection. I have to say even opencv official documentation is not very clear as your post.

So, semantic segmentation using deep learning and opencv could be a nice post in your blog for next year 🙂

Bye

Adrian Rosebrock December 15, 2017 at 8:14 am #

REPLY

I'm glad you found the blog post useful! I really appreciate the kind words as well, thank you. I will consider doing semantic segmentation as a blog post in the future as well.

QV December 12, 2017 at 1:35 pm #

REPLY

HI Adrian, I come across your post to find some info that relate to my current project, but the most impression I am left with is your emotional Christmas story. Like you, I also had lot of struggle growing up in my family. But Christmas is always a wonderful time. And it is very compelling that you find a way to utilize technology to express your feeling and your story. Thank you for sharing with us!

Adrian Rosebrock December 13, 2017 at 1:39 pm #

REPLY

Thank you for the comment! I don't normally share personal information on the PyImageSearch blog but the past month it's felt appropriate. Best wishes to you, your family, and have a Merry Christmas and Happy Holidays.

Jeff December 12, 2017 at 3:04 pm #

REPLY

Hello Adrian,

I am trying to use your code above but unfortunately I keep getting error.
Where do I have to write the path to the images folder where the Santa images are located. And where do I write the path for the NOT Santa images?

John December 21, 2017 at 8:40 pm #

REPLY

Hi Adrian,

Unfortunately, I'm having the same issue as well. You say that "The –dataset switch should point to the directory containing the images we will be training our image classifier on (i.e., the "Santa" and "Not Santa" images)…" But where do I specify that?

I've tried specifying it (replacing "images" with the path to santa images) in the following line, but it doesnt seem to work.

$ python train_network.py –dataset images –model santa_not_santa.model

Could you please help?

Thanks and Merry Christmas

I've tried

John December 21, 2017 at 9:00 pm #

REPLY

Specifically, I'm wondering about lines 9-15 of train_network.py and how I specify the path to the dataset on any of those lines. I've tried a few things, but i keep getting these errors.

Using TensorFlow backend.
…

usage: train_network.py [-h] -d DATASET -m MODEL [-p PLOT]
train_network.py: error: the following arguments are required: -d/–dataset, -m/–model

Could you please provide an example code with pathways? Any help would be appreciated. Thanks

Adrian Rosebrock December 22, 2017 at 6:42 am #

REPLY

Hi John — first, please make sure you use the “Downloads” section of this blog post to download the source code. From there unzip the archive and use your terminal to navigate to where the unzipped code is. You do not need to modify any code. Use your terminal to execute the code as is done in the blog post. If you’re new to command line arguments please give this tutorial a read.

John December 22, 2017 at 10:15 am #

Hi Adrian,
Thanks for your response. I’ve downloaded the data. but I keep getting errors when I try to run the following line in the terminal:

python train_network.py –dataset images –model santa_not_santa.model

File “train_network.py”, line 9, in
from keras.preprocessing.image import ImageDataGenerator
ModuleNotFoundError: No module named ‘keras’

It’s strange, because thus far, I don’t have any issues importing keras and running python scripts with it. More generally, I’m wondering how to create the santa_not_santa.model as well (I might have missed it, but it doesn’t appear to be in the blog post).

If you could clarify the issue for me, that would be fantastic!

Thanks again,

Adrian Rosebrock December 26, 2017 at 4:41 pm #

Running the following command will generate the santa_not_santa.model file:

```
$ python train_network.py --dataset images --model
santa_not_santa.model
```

Since that is the script causing the error your model is not trained and saved to disk.

As for the Keras import issue, that is very strange. You mentioned being able to import and use Keras. Were you using a Python virtual environment? Unfortunately without physical access to your machine I’m not sure what the particular error is.

Cassandra December 21, 2017 at 10:19 pm #

REPLY

I'm having the same issue as well. not sure where to specify the file path for the images. Any help would be appreciated

REPLY

Adrian Rosebrock December 22, 2017 at 8:33 am #

Hi Cassandra — Be sure to use the "Downloads" section of the blog post to download the model and data. You'll need to use the same commands I do in the blog post. For a review of command line arguments, please see this tutorial.

REPLY

Jeff December 12, 2017 at 3:07 pm #

Sorry Adrian,

I forgot to mention train_network.py returns..

ModuleNotFoundError: No module named 'pyimagesearch'

REPLY

Yuri December 12, 2017 at 4:03 pm #

This is an excellent post and systematically submitted information. In the framework of this network, is it possible to obtain information about the coordinates of the object, so that it is possible to construct a rectangle that allocates it?

REPLY

Adrian Rosebrock December 13, 2017 at 1:38 pm #

With LeNet, no, you cannot obtain the (x, y)-coordinates of the object. You would need to train either a YOLO, SSD, or R-CNN network. Pre-trained ones can be found here. If you want to train them from scratch please take a look at my book, Deep Learning for Computer Vision with Python where I discuss the process in detail.

REPLY

Bharath Kumar December 12, 2017 at 10:33 pm #

Hey your the go to tutorials for computer vision..why dont you teach on youtube? Just curious.!!

REPLY

Adrian Rosebrock December 13, 2017 at 1:37 pm #

I've considered it! Maybe in the future I will 🙂

REPLY

Alice December 13, 2017 at 1:53 am #

I find Computer Vision, Deep Learning, Python,.. are so difficult stuffs but I did not give up because your posts and instructions make me feel like one day I can make a little program run. However, after I haven't had any success after many times of trying but as I said I won't give up. I wish you a Merry Christmas and a Happy New Year approaching in the next few weeks.

Adrian Rosebrock December 13, 2017 at 1:36 pm #

REPLY

Thank you so much for the comment Alice, comments like these really put a smile on my face 🙂 Keep trying and keep working at it. What particular problem are you having trying to get your script to run?

Alice December 13, 2017 at 10:35 pm #

REPLY

I got very popular problem and I saw many people got on StackoverFlow:

"Error: '::hypot' has not been declared" in cmath

Adrian Rosebrock December 15, 2017 at 8:28 am #

REPLY

Unfortunately I have not encountered that error before. What library is throwing that error?

Alice December 17, 2017 at 9:48 pm #

Well, I solved it and now the program is running well. I am wondering of making it an Android app when the input is taken from phone's camera, the output in real-time shows santa and not-santa, it is like your demo with object-recognition. Please suggest me some tutorials I should follow. Thanks

Jeff December 13, 2017 at 2:09 am #

REPLY

Hello Adrian,

How do I get the following library:

from pyimagesearch.lenet import LeNet

Adrian Rosebrock December 13, 2017 at 1:36 pm #

REPLY

Hi Jeff, please make sure use the "Downloads" section of this blog post. Once you download the code you'll find the necessary Python files and project structure.

Peter December 13, 2017 at 6:12 am #

REPLY

Hi Adrian, good stuff. I don't seem to have imutils, as in

from imutils import paths

Is this from an earlier post or do I have to conda it?

Peter December 13, 2017 at 8:47 am #

REPLY

No worries Sheila, I found it.

Adrian Rosebrock December 13, 2017 at 1:35 pm #

REPLY

Hi Peter — congrats on figuring it out. I just want to make sure if other readers have the same question they know they can find imutils on GithHub and/or install it via pip:

```
$ pip install imutils
```

AsafOron December 13, 2017 at 10:56 am #

REPLY

Very well presented and easy to follow, wonderful !

Can one utilize this same model for object detection? that is you have a big image say 500×500 with multiple santas in it and you need to identify the various santas and put a bounding box around each and provide a santa count. i believe it can be done by sliding a 28×28 window on the big image and run it through the model but it seems very inefficient not to mention that santas in the images may vary in size. is there a better way ?

Adrian Rosebrock December 13, 2017 at 1:40 pm #

REPLY

Please see my reply to Yuri above where I discuss this question. You can use sliding windows, image pyramids, and non-maxima suppression but you would be better off training a network using an object detection framework.

Subash Gandyer December 14, 2017 at 7:06 am #

REPLY

model.add(Dense(500))

Why is it 500 and not 5000 or any other number? How did you arrive at this?

Adrian Rosebrock December 15, 2017 at 8:23 am #

REPLY

We are following the exact implementation of the LeNet architecture. Please see the post for more details.

menokey December 18, 2017 at 10:03 pm #

REPLY

Hello Adrain ,
Why are we appending all images into one array as in
data.append(image)

menokey December 18, 2017 at 10:59 pm #

REPLY

For the directory structure of pyimagesearch ,what is networks folder and why do we need another letnet.py inside

Adrian Rosebrock December 19, 2017 at 4:16 pm #

REPLY

Please use the “Downloads” section of this blog post to download the code + director structure so you can compare yours to mine. This will help you understand the proper directory structure.

stoiclemon December 21, 2017 at 6:55 pm #

REPLY

Do I have to install sklearn.model separately? can’t seem to find it anywherein the Downloads folder.

Adrian Rosebrock December 22, 2017 at 6:45 am #

REPLY

Yes, you need to install scikit-learn via:

```
$ pip install scikit-learn
```

Chandra December 23, 2017 at 3:34 am #

REPLY

Hi,

Thank you for providing this tutorial. I have a simple question.

You said in line 22-25, you do scaling by dividing your image with 255. I believe that because you expect the images input have many colors. But how if the input is black and white photo or roentgen photography? Does it need to be scaled? How to scale it?

Please advise

Adrian Rosebrock December 26, 2017 at 4:34 pm #

REPLY

The scaling is done to scale the pixel values from [0, 255] down to [0, 1]. This is done to give the neural network less “freedom” in values and enables the network learn faster.

kaisar khatak December 26, 2017 at 1:46 am #

REPLY

Cool post. I think you already identified the issue with the size of the images and network. The LeNet model is just predicting when there is a significant amount of red and white localized together in the input image. If you feed the program any images/frames with a lot of red and/or white, the program will generate false positives.

You have identified some solutions as well:
Gather additional training data
Utilize higher resolution images during training. I imagine 64×64 pixels would produce higher accuracy. 128×128 pixels would likely be ideal (although I have not tried this).
Use a deeper network architecture during training.

Maybe, try using YOLO/SSD for object localization???

BTW, I used the SNOW app (ios/android) and Santa Claus face filter for testing….

video:
https://drive.google.com/file/d/14AjetH-vRosXSoymbz7wnv-iOcTXyuYe/view?usp=sharing

image:
https://drive.google.com/file/d/1PXdtA-a1utL12Uy265-qsiOTR8b1phhL/view?usp=sharing

Happy Holidays!

Adrian Rosebrock December 26, 2017 at 3:52 pm #

REPLY

Thanks for sharing, Kaisar! Yes, you’re absolutely right — the model will report false positives when there is a significant amount of red/white. YOLO or SSD could be used for localization/detection, but that would again require much larger input resolution and ideally more training data.

Abder-Rahman Ali December 28, 2017 at 8:31 am #

REPLY

Thanks so much for this nice post. The issue is that the program is classifying all the images in the “exmaples” directory as “not santa” with 100%.

The plot also looks like this (which is weird): https://www.dropbox.com/s/24q26wvf0ljihdd/fig.png?dl=1

This is the command I used to train the network:

$ python train_network.py –dataset /full-path/image-classification-keras/images/santa –dataset /full-path/image-classification-keras/images/not_santa –model /full-path/image-classification-keras/santa.model –plot /full-path/image-classification-keras/

Any ideas where I might be having some mistakes?

Thanks.

Adrian Rosebrock December 28, 2017 at 2:08 pm #

REPLY

Please take a look at the example command used to execute the script:

```
$ python train_network.py --dataset images --model santa_not_santa.model
```

The “images” directory should contain two sub-directories: “santa” and “not_santa”. Your command does not reflect this. Use the “Downloads” section of the blog post to compare your directory structure to mine.

Abder-Rahman Ali December 28, 2017 at 2:48 pm #

REPLY

Thanks so much Adrian. It works now 🙂 I just get the following warning:

libpng warning: iCCP: known incorrect sRGB profile

I downloaded the code from the link you send through email, and not sure how the “examples” folder came in.

Abder-Rahman Ali December 30, 2017 at 10:21 pm #

REPLY

Hello Adrian, when I downloaded the code, I noticed that the “examples” directory is within the “images” directory. Shouldn’t it be separate? Thanks.

Adrian Rosebrock December 31, 2017 at 10:01 am #

REPLY

Great catch! I added the “examples” directory after I had trained the model. The “examples” directory should be moved up a level. I’ll get the blog post + code updated.

Adrian Rosebrock January 7, 2018 at 9:03 am #

REPLY

Just a quick update: I have updated the code + downloads for this post to move the “examples” up one directory.

Mohammed January 3, 2018 at 3:32 am #

REPLY

I am a new in this area and i want to ask about extract features, so my question is how to decide the best number of epochs that i stop train and get a vectors of features for every image in dataset ?

Adrian Rosebrock January 3, 2018 at 12:54 pm #

REPLY

Hey Mohammed — typically we only perform feature extraction on a pre-trained network. We wouldn't typically train a network and use it only for feature extraction. Could you elaborate a bit more on your project and what your end goal is?

judson antu January 5, 2018 at 8:50 am #

REPLY

hey Adrian,
how good would be this method for detecting rotten and good apples or in that case any fruit. will the only CPU method be enough to train for such a level of accuracy?

and what about resizing the image to more than a 28×28 pixel array, like maybe 56×56 array?

Adrian Rosebrock January 5, 2018 at 1:24 pm #

REPLY

It's hard to say without seeing a dataset first. Your first step should be to collect the dataset and *then* decide on a model and input spatial dimensions.

Judson antu January 5, 2018 at 9:42 pm #

REPLY

Okay, so in my case, the classification would be done in a controlled environment. Like the fruits would be passing on a conveyer belt. In that case , would we need diversity in images?

Adrian Rosebrock January 8, 2018 at 2:57 pm #

REPLY

If it's a controlled environment you can reduce the amount of images you would need, but I would still suggest 500 images (ideally 1,000) per object class that you want to recognize. If your conveyor belt is already up and running put a camera on it and start gathering images. You can then later label them. This would be the fastest method to get up and running.

Andy January 8, 2018 at 12:18 pm #

REPLY

Adrian,

Thank you for a great tutorial.

Question – what does the "not santa" dataset really need to represent for this to work effectively for other types of problems?

For example, if our "not santa" dataset does not contain many images of things like strawberries, watermelons, etc – could it mistakenly classify those as santa (red, green, white, etc.)?

Adrian Rosebrock January 8, 2018 at 2:32 pm #

REPLY

The architecture used in this example is pretty limited at only 28×28 pixels. State-of-the-art neural networks accept images that are typically 200-300 pixels along their largest dimension. Ensuring your images are that large and using a network such as ResNet, VGGNet, SqueezeNet, or another current architecture is a good starting point.

From there you need to gather data, ideally 1,000 images per object class that you want to recognize.

Jim Walker January 16, 2018 at 3:43 pm #

REPLY

Adrian:
Thanks for the project. A problem I am having is this error: If on CPU, do you have a BLAS library installed Theano can link against? On the CPU we do not support float16.

I looked up BLAS libraries but didn't get very far…What does it mean and how can I correct it?
Thanks for your help.

Adrian Rosebrock January 17, 2018 at 10:19 am #

REPLY

BLAS is a linear algebra optimization library. Are you using Keras with a TensorFlow or Theano backend? And which operating system?

Jim Walker January 17, 2018 at 1:04 pm #

REPLY

Theano backend with Windows 10

Adrian Rosebrock January 18, 2018 at 8:58 am #

REPLY

It sounds like you need to install BLAS on your Windows system then reinstall Theano and Keras. I haven't used Windows in a good many years and I do not support Windows here on the PyImageSearch blog. In general I recommend using a Unix-based operating system such as Ubuntu or macOS for deep learning. Using Windows will likely lead to trouble. Additionally, consider using the TensorFlow backend as Theano is no longer being developed.

Akbar H January 16, 2018 at 7:23 pm #

REPLY

(notSanta, santa) = model.predict(image)[0]

is this label notSanta and santa, same as 0 and 1 ?

thanks.

Adrian Rosebrock January 17, 2018 at 10:17 am #

REPLY

`notSanta` is the *probability* of the "not santa" label while `santa` is the *probability* of the "santa" label. The probability can be in the range [0, 1.0].

isra60 January 17, 2018 at 2:34 am #

REPLY

Hi Adrian.

I'm really interested in this tutorial and I want to learn to my own purposes

Have you ever tried to train with thermal or infrarred images?? Any hints of how to do this??
Maybe this is not possible as this models and detectors are only color reliable or maybe we can train them in other way..

As for visible images we have PASCAL VOC 2012 in order to benchmark our models do you know a benchmark for thermal images?

Thank you

Adrian Rosebrock January 17, 2018 at 10:10 am #

REPLY

I have not trained to train a network on thermal or infrared images but the process would be the same. Ensure Keras and/or OpenCV can load them, apply labels to them, and train. That would at least give you a reasonable benchmark to improve upon.

Dii88 January 21, 2018 at 1:16 am #

REPLY

I cannot make it run at Windows 10, can you show me a guide or tutorial to run this sample code in windows 10 environment?

Adrian Rosebrock January 22, 2018 at 6:25 pm #

REPLY

Hey there, what is the error you are getting when trying to run the code on Windows?

Leo January 23, 2018 at 8:43 am #

REPLY

Hi, Adrian! Very nice article! 🙂

However, I have a question. I tried to apply the same NN to detect whether the image contains a road or not, but it couldn't detect the non-road images in any case.
– I tested on images with size 250×250 (but later changed to 50×50)
– Whenever I try to classify the new image with non-road, it always return the result that says that it's road (which is incorrect). Even when I provided a completely black image with no content, it said that it was road with 55% accuracy.

I think I need to pre-process the images (e.g.: convert to grayscale, detect edges and fed the images only with detected edges to NN). What do you think about this?

Adrian Rosebrock January 23, 2018 at 1:59 pm #

REPLY

It sounds like your network is heavily overfitting or is incorrectly thinking every input image is a “road”. I would double-check the labels from the Python script to start. You should not need any other image processing operations. A CNN should easily be able to determine road vs. non-road. I believe there is an issue with the labeling or there is severe overfitting. Checking the plot of accuracy vs. loss will help diagnose the overfitting.

Leo January 24, 2018 at 3:58 am #

REPLY

Yeah, you are right.
I had a bug, I forgot to normalize the RGB images to the range [0; 1] in the validation script. 😀
Thank you for the hints!

Adrian Rosebrock January 24, 2018 at 4:59 pm #

REPLY

Awesome, congrats on resolving the issue 🙂

Vlad January 25, 2018 at 6:54 am #

REPLY

Hi. What to do with such an error? I can not understand the reason

(cv) dntwrd@dntwrd-900X3G:/media/dntwrd/for_ubuntu$ python test_network.py –model santa_not_santa.model –image images/examples/santa_01.png
Using TensorFlow backend.
Traceback (most recent call last):
File “test_network.py”, line 23, in
orig = image.copy()
AttributeError: ‘NoneType’ object has no attribute ‘copy’

Adrian Rosebrock January 25, 2018 at 3:46 pm #

REPLY

Hey Vlad — it looks like `cv2.imread` is returning “None”, implying that the image could not be read from disk. Double-check your paths to the input image. The “examples” directory is actually one level up from images, so update your switch to be `--image examples/santa_01.png` and you should be all set.

Mat January 29, 2018 at 4:53 am #

REPLY

It works ! What a thrill for a newbie like me… It still condiders a kinder or the swiss flag with a santa because of the rezising of the images. But that doesn't matter at this stage. To me, such exemples are a perfect starting point to deep learning

Adrian Rosebrock January 30, 2018 at 10:16 am #

REPLY

Congrats on getting up and running with your first Convolutional Neural Network, Mat! 🙂

Oliver January 29, 2018 at 7:22 am #

REPLY

I think I'm missing something here, it seems like the only file in the download section is the "Resource Guide" and that one doesn't contain any code or santa pictures.

Oliver January 29, 2018 at 10:50 am #

REPLY

Actually, nevermind. Got it all to work on my own, awesome tutorial, thank you a lot, Adrian!

Now I'll have to see how I modify it to work with categorical_crossentropy instead of binary.

Adrian Rosebrock January 30, 2018 at 10:15 am #

REPLY

Congrats on resolving the issue Olivier. For what it's worth, there is a section that says "Downloads" directory above the "Summary" section.

Swapping in "categorical_crossentropy" is a single line change, just add more than two classes to the "images" directory and you'll be all set.

Judson Antu February 1, 2018 at 2:16 am #

REPLY

hey adrian,
i wanted to use this code as a separate thread in my raspberry, but i get this error when i use tensorflow in another thread.

ValueError: Tensor Tensor("activation_4?softmax:0",shape=(?,2),dtype=float32)is not an element of this graph,)

from my searching i got to know that, python and threading don't go well because of GIL. and also tensorflow graph must be adjusted for that, but i cant grasp enough of the tensoflow documentation.

avital's comment her https://github.com/keras-team/keras/issues/2397 led me to the tensorflow documentation here
https://www.tensorflow.org/api_docs/python/tf/get_default_graph

how do you explain this with eference to your code here. i Hope you will shed some light into my brain

and by the way, your deep learning book is helping me a lot!!Thankyou

Adrian Rosebrock February 3, 2018 at 11:13 am #

REPLY

Hey Judson, I'm glad you're enjoying Deep Learning for Computer Vision with Python!

As for your question, can you explain more about the threading process? I did some threading experiments with OpenCV + deep learning in this blog post. The post uses OpenCV's "dnn" module but the threading should be extendible to Keras + TensorFlow. At the very least that will give you some boilerplate code to go on.

Hayley February 6, 2018 at 8:44 pm #

REPLY

Hi Adrian, your post really helps me a lot on my thesis. I tried to use the model on my own data set, the data set only have 60 images, after training, both train_loss and val_loss stabled at about 0.75. and the accuracy is lower than loss. I've tried to change the batch size, change the parameter in lenet, but the result is same. Would you please help me with that? Thanks a lot.

Adrian Rosebrock February 8, 2018 at 8:39 am #

REPLY

The accuracy and loss are not the same. A loss function is optimizing over your loss landscape. The accuracy is the prediction accuracy on your training and/or testing set. I think the problem is that you may not have enough training data. Try gathering more training images to start. From there you might want to try a deeper network architecture provided you have more training images.

Shayan February 7, 2018 at 1:47 am #

REPLY

Great explanation Adrain! Do you know how we could expand this model to sequence of image frames classification, like you would use in playing a video of a person speaking one word and then classifying that word. It would be great if you could help me on that and get in touch!

Adrian Rosebrock February 8, 2018 at 8:35 am #

REPLY

Hey Shayan — I'm a bit confused about your question. You mentioned using video and then classifying the word the person is saying. Are you using the audio to recognize the word or are you trying to use the persons lips?

Rijul February 12, 2018 at 1:15 am #

REPLY

Hey Adrian I tried training the network using images gathered from google and ukbench.
Use case : detect whether a person is smoking in an image or not .
I got 0 acc and 7.9 loss changed learning rate(to 1e-4) got 50 loss 50 acc hence every nonsmoker in

image is 50% confidence and smoker in image also 50% confidence. Can you help me out or another place where I can contact u . Thankyou.

[link] https://imgur.com/1RWo4FH

Adrian Rosebrock February 12, 2018 at 6:18 pm #

REPLY

Are you using the network included with this blog post or are you using a custom network? I would expect the network to obtain at least some accuracy, even if it's poor. 0% classification accuracy sounds like you may need to check your class labels as there might be an issue when you parsed them from the image paths.

Rijul February 13, 2018 at 3:25 pm #

REPLY

Yes I am indeed using the same network as mentioned here. I rechecked the code but can u see once.

[code image link] https://imgur.com/xN93Njf

Thankyou

Adrian Rosebrock February 18, 2018 at 10:17 am #

REPLY

Your code looks correct. How many images do you have per class? I would suggest trying another network architecture.

Tarak February 13, 2018 at 1:25 pm #

REPLY

The code worked! Thanks!
I'm trying to use this code to build a CoreML file to use it on my iOS but after using coremltools.convert, I'm getting model input as a MultiArray instead of an image input unlike other models that apple has released.

coreml = coremltools.converters.keras.convert('/home/tarak/Documents/keras_image_classifier/peacocknopeacock.model', image_input_names='camera', is_bgr= True, output_names='Output_names')

WHat is going wrong??

Adrian Rosebrock February 18, 2018 at 10:19 am #

REPLY

Hey Tarak — I'm not sure what the error is here but I'll be doing a CoreML tutorial here on the PyImageSearch blog in the next couple of weeks. I'll be sure to include detailed instructions on how to export the model for use in CoreML.

sido February 15, 2018 at 6:21 am #

REPLY

err
..
from pyimagesearch.lenet import LeNet
ImportError: No module named ‘pyimagesearch’
pliz help me!!

Adrian Rosebrock February 18, 2018 at 10:00 am #

REPLY

Make sure you use the “Downloads” section of this blog post to download the source code and example images. The downloads contains the “pyimagesearch” module.

Vishwa July 23, 2018 at 4:02 am #

REPLY

After downloading it, what should I do?

Because I downloaded it and I ran the “from pyimagesearch.lenet import LeNet” code again and I still get an error.

Adrian Rosebrock July 25, 2018 at 8:18 am #

REPLY

Make sure you download the code, unzip it, change directory to the unzipped directory, and then execute your script via the command line from that directory.

Enes March 1, 2018 at 3:51 am #

REPLY

Hi I want to ask a question about loss function. While you are training your model, you are using binary cross-entropy as loss function. But your network has two output.

model = LeNet.build(width=28, height=28, depth=3, classes=2)
# softmax classifier
model.add(Dense(classes))
model.add(Activation(“softmax”))

But when I examined examples people used one output while they are using binary cross-entropy as loss function. But you have 2 output.

Is there any problem there about outputs and loss functions.

Thanks.

Adrian Rosebrock March 2, 2018 at 10:47 am #

REPLY

You use binary cross-entropy for two class classification (hence the term binary, one/off, two classes). You use categorical cross-entropy for > 2 classes.

Other loss functions exist as well, but those are the ones primarily used for classification.

Ryan Chase March 1, 2018 at 4:12 pm #

REPLY

Hey Adrian – where do you specify that you want to exclude the final fully-connected layer of LeNet in favor of adding a fully connected layer for our Santa/not-santa binary classes? It seems that when the model is built this is not explicitly specified in the code (unless it's specified somewhere in `pyimagesearch.lenet import LeNet`. In that case, wouldn't you want flexibility to specify how much of the convolutional base you want to keep in-tact? All I see is:

model = LeNet.build(width=28, height=28, depth=3, classes=2)

…whereas in François' github script, 5.3-using-a-pretrained-convnet.ipynb (https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.3-using-a-pretrained-convnet.ipynb)
you actually specify include_top=False so that we can connect the finally fully-connected layer to our classes:

conv_base = VGG16(weights='imagenet',
include_top=False,
input_shape=(150, 150, 3))

Can you clear this up for me? Thanks!

Ryan Chase March 1, 2018 at 4:43 pm #

REPLY

…I cracked open lenet.py. It appears that we're not even using the pre-trained weights from the LeNet model – only the LeNet model architecture?

Adrian Rosebrock March 2, 2018 at 10:40 am #

REPLY

We are using the LeNet architecture and then we train the LeNet model, generating the LeNet weights. This model is not pre-trained. It sounds like you have a lot of interest in studying deep learning which is fantastic, but I would recommend that you work through Deep Learning for Computer Vision with Python to better help you understand the fundamentals of CNNs, how to train them, and how to modify their architectures.

Ryan March 2, 2018 at 3:07 pm #

REPLY

Right, I was expecting that the script would use transfer learning and load the weights from ImageNet or something (as in the below snippet). In this case though it looks like you were able to just use the LeNet architecture and train weights from scratch.

```
from keras.applications import VGG16
```

```
conv_base = VGG16(weights='imagenet',
include_top=False,
input_shape=(150, 150, 3))
```

Adrian Rosebrock March 3, 2018 at 7:51 am #

I think you may have a misunderstanding of the LeNet architecture. LeNet was one of the first CNNs. It accepts very small input images of 28×28 pixels. It was never trained on the ImageNet dataset. It would have performed extremely poorly.

Adrian Rosebrock March 2, 2018 at 10:38 am #

REPLY

If you wanted to remove the fully-connected layers you would need to remove Lines 33-39; however, keep in mind that this post isn't like the VGG, ResNet, etc. pre-trained architectures in Keras which automatically handle include_top for you. You would need to implement this functionality yourself.

Ryan March 1, 2018 at 8:18 pm #

REPLY

For the line,

```
(notSanta, santa) = model.predict(image)[0]
```

…how do you know the order of the probabilities? How could you detect this order in the case of mutli-class classification problem? I'm using the Keras ImageDataGenerator which automatically creates a one-hot encoded label matrix based on the directories where the images are stored – separated into their specific class directory. The issue is that I don't know how to get the actual labels that the columns of the one_hot encoded matrix correspond to. Thank you!

Adrian Rosebrock March 2, 2018 at 10:36 am #

REPLY

I would explicitly impose an order by using scikit-learns LabelEncoder class. This class will ensure order and allow you to transform integers to labels and vice versa.

Ryan March 2, 2018 at 3:01 pm #

REPLY

Okay, thanks. So Keras doesn't have anything built-in to identify which column of the output probabilities corresponds to which class? I find that very odd…

Also, per your suggestions, do you have an example of someone marrying a pre-trained model in Keras where the fully-connected output is connected to the multiple classes they care about classifying along with the scikit-learn LabelEncoder that you could point me to?

Since the Keras ImageDataGenererator creates the label arrays automatically based on the directory structure, I'm just curious how I would ensure that the two frameworks (Keras and scikit-

learn) are able to work together for this purpose.

Thanks! Huge fan!

Adrian Rosebrock March 3, 2018 at 7:50 am #

REPLY

Keras takes the same approach as scikit-learn — it's not the responsibility of the .predict method to explicitly impose the order. The .predict methods accept an image (or feature vector) as input and a set of labels or probabilities is returned. We then use a separate class (such as LabelEncoder or LabelBinarizer), if necessary, to transform the returned values into labels.

I do not have any examples here on the PyImageSearch blog (yet) that use scikit-learn's LabelEncoder/LabelBinarizer, but I do have numerous examples of this in my deep learning book. In my next Keras-based deep learning post I'll try to cover it if I remember.

The gist in pseudocode would be:

```python
labels = parse labels from file paths
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
preds = model.predict(image)
idxs = np.argsort(preds)[::-1]
le.inverse_transform(idxs[0])
```

Ryan March 2, 2018 at 5:11 pm #

REPLY

Ah, I also found the class_indices attribute in Keras' flow_from_directory(directory):

classes: optional list of class subdirectories (e.g. ['dogs', 'cats']). Default: None. If not provided, the list of classes will be automatically inferred from the subdirectory names/structure under directory, where each subdirectory will be treated as a different class (and the order of the classes, which will map to the label indices, will be alphanumeric). The dictionary containing the mapping from class names to class indices can be obtained via the attribute class_indices.

Source: https://keras.io/preprocessing/image/

I think this is what I was looking for, though would we interested to see an example marrying keras with scikit-learn's LabelEncoder class if that's available.

Adrian Rosebrock March 3, 2018 at 7:52 am #

REPLY

See my previous comment. Also, I sent you an email. Check it when you get a chance. Thanks.

Masoud March 8, 2018 at 3:08 am #

REPLY

Hi Adrian,

Thank you very much for such an amazing and informative post. I was wondering how I can get a confusion matrix if I iterate this model in a loop for a larger number of test data. Is there any built in function in keras which can help me get it?
Thanks again

Adrian Rosebrock March 9, 2018 at 9:19 am #

REPLY

I would create a Keras callback that at the end of each epoch calls the confusion_matrix function of scikit-learn and logs the result to disk.

DHEERAJ March 9, 2018 at 6:07 am #

REPLY

Hello Adrian,This is very useful post for beginners.

As you mentioned that using Lenet,we cant recognize objects with good accuracy.Then using the same code If i change the architecture name in the starting of the code (Lenet to Alexnet) and add some more (convolutional layers) and dataset having images of size 299*299 with 100 classes,will it work and classify with good accuracy?

Please mention if there are some more changes required .

Adrian Rosebrock March 9, 2018 at 8:46 am #

REPLY

It's not exactly that simple, but you have the right idea in mind. The larger your network, the more parameters you introduce. The more parameters you introduce, the more data you'll likely need to train your network. If you want to implement AlexNet you should follow the original publication or follow along with the code inside my book, Deep Learning for Computer Vision with Python. This book will help you implement popular CNNs and even create your own architectures. I hope that helps point you in the right direction!

amina March 10, 2018 at 12:22 pm #

REPLY

please I want to do multi_class classification any help ? I am new to keras !

Arnold March 15, 2018 at 4:08 pm #

REPLY

Hello Adrian, thank you for the blog post.
I have a question I would like to ask. I have a classification problem that basically from a plain image with text I want to classify them by certain features. These features include things like if the text is bold or not, lower or upper case, colors, etc,

Do you think using Lenet would be a good approach for this?
We are using input size of 170x120x3

We are adding a empty border for each image in order to fit the aspect ratio, for avoiding resizing distortion on the text.

And the last things is that our classes are not well balanced so we decided to add a “miscellaneous” class that basically means all non common classes that we can’t classify by itself because of the sample size, but this approach is not working properly and is confusing the main classes, any suggestions? Thank you.

Adrian Rosebrock March 19, 2018 at 5:45 pm #

REPLY

I think LeNet would be a good starting point. I’m not sure it will be the most optimal architecture but it should be more than sufficient for a starting point.

Also, keep in mind that LeNet requires images to be 28x28x1. If you are using 170x120x3 you’ll need to modify the input shape of the architecture to include the channels dimension. Additionally, you’ll need to squash the input image down to 28×28 and ignore aspect ratio.

If you require a 170×120 input, or similar, you’ll need to code your own implementation to handle this.

Osama Almas March 21, 2018 at 7:02 pm #

REPLY

Hey Adrian,

I’m trying to use this same architecture to predict if an image is happy or sad, I’m using the JAFFE dataset which has images in tiff format. Thing is, in the fit function in my code throws this error “ValueError: (‘Input data in `NumpyArrayIterator` should have rank 4. You passed an array with shape’, (0,))” I think I’m missing something easy here since the requirement to santa not santa architecture is almost same just the pictures are in grayscale and different format.
Can you help me out what this?

Adrian Rosebrock March 22, 2018 at 9:41 am #

REPLY

Hey Osama — I haven’t worked with the JAFFE dataset so I can’t really comment on what the exact nature of the error is. That said, based on your error it seems like your iterator is not returning a valid image. You should insert some debugging statements to narrow down on the issue further. For what it’s worth, I cover emotion recognition, including how to train your own CNN to perform emotion recognition, inside my book, Deep Learning for Computer Vision with Python. It would be a great starting point for your project.

Osama Almas March 27, 2018 at 3:47 pm #

REPLY

I have fixed the error , It was because of the .tiff format, as u suggested wasn’t returning a valid image, i changed all the image formats to jpg(just changed the extension of each tiff image) then my network detected the images, but even though they are grayscale images my network detected that it had 3 depth, so i converted those images to grayscale using cv2, now everything works fine. hope this helps anyone in hte future.
Thanks brother for your reply.

Adrian Rosebrock March 30, 2018 at 7:29 am #

REPLY

Awesome, I'm glad that worked! Congrats on resolving the issue 🙂

ANKUR June 1, 2018 at 5:56 pm #

REPLY

hi osama…i got the same error
my images were in jpg format from the beginning….what shall i do???

Anshuman March 29, 2018 at 3:46 am #

REPLY

I am facing the same problem with a dataset I created myself using google image search. Were you able to fix the problem?

Karlos March 22, 2018 at 8:55 am #

REPLY

My black cat is so so similar to Schipperke puppies. I have used several DNN architectures but they do not work (as a binary problem, multiclass etc). Also, I have used a lot of images. Do you think that my classification problem could be similar to the problem "chihuahuas-muffins"

Adrian Rosebrock March 22, 2018 at 9:32 am #

REPLY

Potentially, but it's hard to say without seeing your example images. Did you use the network architecture (LeNet) from this blog post? Or did you use a different one?

Travis Prosser March 22, 2018 at 1:46 pm #

REPLY

Great tutorial! But I'm wondering: why must the network be trained on, and modelled to predict "not Santa" specifically, instead of just training to recognize, and report a Santa when the certainty is above some chosen threshold?

If you were trying to distinguish between pictures of two different classes, (say it was trained with pictures of a stapler and pictures of a calculator), but then presented a picture of Santa (or some other unknown item), wouldn't you only want it to ID the object if it was at least X% certain, rather than reporting the picture of Santa as one of those two classes? Perhaps if neither class is matched with greater than 30% certainty, you would label the picture as "unknown". Or is there some reason that in that case you'd need to train the network on three classes total (stapler, calculator, other)? Wouldn't that make the performance more dependent on the randomness of your training data for the "other" class, even if you had great training data for the two classes you're interested in?

I suppose in short, I'm asking if you're trying to identify N classes of objects, do you inherently need another catch-all class to capture the "none of the above" category? And therefore need N+1 sets of

training data (the last of which, being a random sample of arbitrary images)?

Adrian Rosebrock March 27, 2018 at 6:47 am #

REPLY

Image classifiers do one thing: classify images. Given a set of classes an image classifier will assign a label to it with some probability. You do not need a “catch all” class for all image classification tasks. The problem you run into is that if you do not use a “catch all” class you could easily misclassify data points as false positives. In this case, yes, I could train the network on just “santa” but its accuracy would not be as good. Whether or not you do this really depends on your project and dataset.

Vinay March 27, 2018 at 12:46 am #

REPLY

Hello Adrian,

Thanks for the very good post. However while running the training module, i get the below error –

File “C:/Study Assignments/train.py”, line 60, in
image = cv2.resize(image, (28, 28))

error: (-215) ssize.width > 0 && ssize.height > 0 in function cv::resize

Can you please help me with some pointers/directions.

Thanks a lot,

Adrian Rosebrock March 27, 2018 at 6:10 am #

REPLY

It sounds like the path to the directory containing the input images is incorrect. Make sure you double-check your input path and that you are correctly using command line arguments.

Akshaya Balamurugan April 5, 2018 at 1:24 pm #

REPLY

Hi Adrian,

Thank you for your great post here. This is quite helpful for beginners like us to start off with DL and Keras. I was trying to follow through the code to do a prediction on 6 classes using categorical cross entropy. Say, there’s a beaker filled with water and another class with an empty beaker with 500 images each after some shear manipulations. After running a certain number of epochs (Accuracy checked), when a test is made on it, the prediction is right only if the image is given very similar to the training set.

Unfortunately, if there’s a hand holding that beaker (empty or filled), the prediction is not coming up right. Any thoughts on what could be done in this case ? Should the training data has to be very diverse including the hands covered in the image too to have a generalized model?

Adrian Rosebrock April 6, 2018 at 8:50 am #

REPLY

Your training set should *absolutely* be more diverse and include images that more closely resemble what the network will see during prediction time. Keep in mind that CNNs, while powerful, are not magic. If you do not train them on images that resemble what they will see when deployed, they will not work well.

Akbar Hidayatuloh April 5, 2018 at 8:36 pm #

REPLY

(notSanta, santa) = model.predict(image)[0]

why notSanta is first? is this because when we extracting the class label we use 1 to define class santa and 0 to class notSanta? how about if i want to classify more than 2 class?

Thank you.

Adrian Rosebrock April 6, 2018 at 8:49 am #

REPLY

We are able to do “(notSanta, santa)” because we know the ordering ahead of time. If you wanted to classify more than two classes I would recommend using scikit-learn’s LabelEncoder/LabelBinarizer. I’ll have a blog post coming out in the next two weeks that will discuss exactly how to do this, so keep an eye on the PyImageSearch blog.

fachrul April 8, 2018 at 1:40 am #

REPLY

hi adrian, your post is very great.
but, i have tried run train_network.py then problem arised.
“libpng warning: iCCP: known incorrect sRGB profile”

Adrian Rosebrock April 10, 2018 at 12:33 pm #

REPLY

That’s just a warning from the “libpng” library saying that it does not have a supplied color profile. The warning can be safely ignored. It will not affect the execution of the code.

Davis Smith April 9, 2018 at 1:02 pm #

REPLY

Hey Adrian,

I’m following the tutorial, but caught a snag. I have set up an Anaconda virtual environment for Python 2.7 per your suggestion. However, when trying to install Tensor flow on my machine (Windows 10), apparently it only supports Python 3.5 and 3.6. I was assuming we needed to use Python 2.7 on the machine as well as the Raspberry Pi, but perhaps this was a wrong assumption?

Thanks

Adrian Rosebrock April 10, 2018 at 12:04 pm #

REPLY

You can use Python 3 to train your model; however, if you intend on using your Raspberry Pi + TensorFlow to deploy your model you need Python 2.7.

Dinesh Kumar April 17, 2018 at 11:15 am #

REPLY

during running of test_network.py…. I got this

AttributeError: 'NoneType' object has no attribute 'copy'

how do I solve this

Adrian Rosebrock April 18, 2018 at 3:05 pm #

REPLY

Your path to your input images is invalid and "cv2.imread" is returning "None". Double-check your input paths and read this post on NoneType errors.

Shubhanker Goyal April 18, 2018 at 11:12 am #

REPLY

what changes we have to do in the following code if we have to train it on multiple classes?

Adrian Rosebrock April 18, 2018 at 2:45 pm #

REPLY

The only change you need to make to the code is change `binary_crossentropy` to `categorical_crossentropy` when compiling the model. That's literally it.

Gagan April 23, 2018 at 8:19 am #

REPLY

Hello, Adrian Rosebrock.
You just awesomely did this, Thank you so much for the project I learn a lot from this and finally able to made by my own.
Now, I want to link this project with my android app in which I successfully did my java part but I'm facing the problem in the optimization of the model for the mobile user.
I used this link, MobileNet and Inception V3 model for the for the optimization but still on the first step. I need your help to cross the ladder.
https://codelabs.developers.google.com/codelabs/tensorflow-for-poets-2-tflite/#0
Thank you again for this valuable knowledge.

Adrian Rosebrock April 25, 2018 at 5:59 am #

REPLY

Thanks for sharing, Gagan. I just published a tutorial on Keras + iOS and I'd like to do a tutorial for Android as well. I'll be sure to take a look at your post for inspiration!

GAGAN April 26, 2018 at 4:21 am #

REPLY

Thanks for replying, Rosebrock. Again a very knowledgeable tutorial.
Will be so grateful for android too.

Sachin Dalvi April 23, 2018 at 11:45 am #

REPLY

Hello Adrian,

How can I only save the architecture of the Model as ".h5" or ".json" and not the complete trained model ?

thanks

Adrian Rosebrock April 25, 2018 at 5:56 am #

REPLY

Take a look at the Keras docs. The `model.to_json()` function will return the model architecture as a string which you can then save to disk.

Fred Macdo April 26, 2018 at 12:06 pm #

REPLY

Do all the images need to be in the same file, can they be in a folder and then separated in subsequent folders?

Adrian Rosebrock April 28, 2018 at 6:13 am #

REPLY

For each class, create a directory for that class. All images for that class should be stored in that same directory without nested subdirectories. To see the example dataset and project structure, be sure to use the "Downloads" section of this blog post.

Peter April 26, 2018 at 2:26 pm #

REPLY

Hi Adrian, thank you for a great tutorial.
I'm running into a problem, when I copy your code and run it with your santa_not_santa.model it will always give me the correct answer, but if I train it again using:
python train_network.py –dataset images –model santa_not_santa.model
(I just want to reiterate that I'm using exactly your code)
I always get Santa from 55-96% certainty when I test then manhattan image:
python test_network.py –model santa_not_santa.model –image examples\manhattan.png

Adrian Rosebrock April 28, 2018 at 6:12 am #

REPLY

That is indeed strange behavior. I'm not sure why that may happen. Which version of Keras are you running? And which backend?

Laurent Weingart April 29, 2018 at 6:17 am #

REPLY

HI Adrian, my comment did not pass through moderation so I'll try to put it again with less parts looking like code…

I must say that your tutorials are among the best I've seen on the web.
I have my first CNN to train with images and some 10 classes or so for an artistic project, which led me to this tutorial, but reading it I would have a few questions if you don't mind.
On the last group of layers that you add in this example, in line 34 of lenet.py you add a Dense layer with 500 nodes for a fully connected layer. Could you please tell us how you ended up with this number ? Is it 2x5x50 from the previous layer ?
Then you mention that the spatial dimension for the LeNet is 28×28. Would you mind elaborating on this value ?

Well, from the moment of my first attempt to place a comment and now, I realised that the LeNet was an already existing architecture, so that is where it comes from, but I don't see where it is defined or why it would not work with a different image size ?

I'm sorry for my questions but it's just that I'm trying to understand how these numbers are defined. 🙂
Also, could you please point me in the direction of some instructions or documentation on how I should build my CNN architecture, like the number of layers and functions ?

Well with this last question, once again between yesterday ad today I downloaded you ToC and few free chapters from your book and I think I will find enough there to get something working not too badly.

Thank you again for you contributions to the web, cheers.

Adrian Rosebrock May 3, 2018 at 10:21 am #

REPLY

1. We flatten our spatial volume into a list via the "Flatten()" class. From there we connect 500 fully-connected codes.

2. The LeNet architectures assume all input images are 28x28x1.

3. It can work with different image sizes but you would need to be careful. If you're working with larger images you may need to add more layers in the network.

If you would like to learn more about deep learning + computer vision then I would highly recommend Deep Learning for Computer Vision with Python (as you have already done). My book addresses all of your questions and more. I guarantee it will help you learn how to apply deep learning to your own projects. Be sure to check it out!

Peter April 30, 2018 at 10:19 am #

REPLY

Thanks for your quick response Adrian, I use the following:
TensorFlow: v.1.7.0
Keras: 2.1.6

Can you remember what version and backend you used for this tutorial so I can make a comparison? Here are the printouts of konsole:
https://ibb.co/jkD4ZH
https://ibb.co/nFdMEH

I'm using tensorflow-gpu, CUDA v9.0, cuDNN v7.0.5 and run everything on windows. Not sure if those are normal values in the epoch, any help will greatly help me to move forward.
Many thanks Adrian.

Adrian Rosebrock May 3, 2018 at 10:11 am #

REPLY

I believe I used TF 1.5 for this. It was on a Unix machine with my Titan X GPU. I cannot recall the CUDA version offhand (I'm traveling right now and don't have SSH access to the machine). Your network output looks good so it's clearly training and learning but I'm not sure why it would always be predicting "sandta" for all test images.

Peter May 4, 2018 at 10:49 am #

REPLY

Hi Adrian, I have trained myself a model that works for the problem i'm testing, and I want to thank you for this tutorial, it helped tremendously to get myself going.
Yeah, I'm not sure what is wrong, but in the example folder, the model that I trained from your code it did not have problem with night_sky image, only manhattan.
I look forward to dive into more tutorials you have and reading your book to learn more, thank you for your time!

Adrian Rosebrock May 9, 2018 at 10:31 am #

REPLY

Congrats on training your image classification model, Peter! Great job 🙂

usup suparma May 8, 2018 at 12:31 pm #

REPLY

the same thing i experienced like peter. the data I input are all santa although the image I input is not santa. but the results remain santa with accuracy above 95%

mohamed mayhoub May 4, 2018 at 1:42 pm #

REPLY

thank you very much, I tried the code but the fatal error: the following arguments are required: -d/–dataset, -m/–model
I create the directory (dataset, model, and plot) but don't solution, please help me

Adrian Rosebrock May 9, 2018 at 10:29 am #

REPLY

Your error can be resolved by reading this post on command line arguments.

roxana May 5, 2018 at 9:15 am #

REPLY

hi dear Adrian
why your val_accuracy is higher than training accuracy?in most of example i saw that training acc always higher than val_acc ,my cnn network have this problem,are you think its ok?or something may be wrong?

Adrian Rosebrock May 9, 2018 at 10:25 am #

REPLY

No, it's a common misconception that training accuracy is *always* higher than the validation accuracy. It may be the cause that training accuracy is normally higher than validation accuracy, but keep in mind that both are just proxies. Depending on the amount of training data you have, your reguarlization techniques, your data augmentation, etc. it is possible for validation accuracy to be higher.

Arun May 7, 2018 at 7:20 am #

REPLY

Hi Adrian…

This tutorial has been really useful for me and I learnt a lot of new stuff. Also I watched this tutorial of yours

https://www.pyimagesearch.com/2017/09/18/real-time-object-detection-with-deep-learning-and-opencv/

where you taught how to detect real time objects.
Now, it would be really useful for me if I am able to include my own feature class and training set and utilise it in real time object detection.

Is it possible to create my own class and use it in a recording video??

Can you please help me?? I am new to deep learning and I am really interested in it…

Adrian Rosebrock May 9, 2018 at 10:05 am #

REPLY

Hi Arun — building your own custom object detectors is a bit of an advanced topic. If you're new to deep learning I would suggest working through Deep Learning for Computer Vision with Python. I have included chapters on training your own object detectors as well. The book will help you go from deep learning beginner to deep learning practitioner quickly.

ImranKhan May 10, 2018 at 11:49 am #

REPLY

please make tutorial on deep neural network credit card fraud.

Adrian Rosebrock May 14, 2018 at 6:52 am #

REPLY

I don't have any datasets for credit card fraud. The PyImageSearch blog is also primarily computer vision-based. I don't have any plans to extend to other types of pattern recognition. Perhaps in the future, but not right now.

lee May 10, 2018 at 3:45 pm #

REPLY

Hi Adrian, I face this problem when I try to compile the code as you mention:

"TypeError: softmax() got an unexpected keyword argument 'axis'"

any idea how to solve this?
thanks for your help

Adrian Rosebrock May 14, 2018 at 12:13 pm #

REPLY

Which version of Keras and TensorFlow are you using?

Suke May 11, 2018 at 1:11 pm #

REPLY

Hi,back to you again. Question is: do you think Generative Adversarial Networks (GAN) can be used as a classifier too? For example, GAN is used to classify/determine objects in PASCAL VOC dataset?

Thank you.

Adrian Rosebrock May 11, 2018 at 2:18 pm #

REPLY

I think the larger question is what your end goal is? Is there a particular reason you're trying to use GANs here?

Hossain May 12, 2018 at 8:05 am #

REPLY

Hi Adrian, Thanks a lot for your great article. I downloaded code and dataset. When I test I get Santa around 55% for the Manhattan picture. What could be the reason?

Adrian Rosebrock May 14, 2018 at 12:04 pm #

REPLY

Hey Hossain — try retraining your network and see if you get the same result. Given our small image dataset a poor random weight initialization could be the cause.

Hossain June 9, 2018 at 9:25 am #

REPLY

Yes. That makes sense. Thanks, Adrian.

Thakur Rohit May 22, 2018 at 10:42 pm #

REPLY

Hello Adrian,
I am trying to use my own data set for training purpose. I have 1200 images of one class and same of other and all of them are grayscale images. When i changed the training script input channel from 3 to 1 i got this error:

ValueError: Error when checking input: expected conv2d_1_input to have shape (None, 28, 28, 1) but got array with shape (831, 28, 28, 3)

Can you explain what i am doing wrong?
Thanks in advance.

Adrian Rosebrock May 23, 2018 at 7:11 am #

REPLY

It looks like your images are being loaded as RGB arrays even though they are grayscale. Make sure you explicity convert your images to grayscale during preprocessing:

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Thakur Rohit May 23, 2018 at 10:26 pm #

REPLY

Hello Adrian,
I explicitly converted the images to grayscale but getting the same error when changing depth from 3 to 1 in training script. Is it possible that something else is causing this error? Can you suggest a way to resolve this?

Adrian Rosebrock May 24, 2018 at 6:37 am #

REPLY

I'm still convinced the error is related to your grayscale conversion. You should debug this further by examining the shape of the NumPy array that you are passing into your model for training. There may be some sort of logic bug in the code where you perform the grayscale conversion but the grayscale image is not added to your "images" list.

Sourabh Mane May 28, 2018 at 1:10 am #

REPLY

Hi Adrian,
Great tutorial!! I have some questions for you,

1.Is it necessary to keep nos of images same for both the classes while training and does it affect the accuracy??

2.I have trained 2 classes with 2000 plus images for both classes and , accuracy i m getting is 0.73 , so what should i do to get accuracy 1??

3.Can LeNet be used to train 10,000 plus images each for classes??

4.Does increasing the nos of epochs affects accuracy??

Adrian Rosebrock May 28, 2018 at 9:31 am #

REPLY

1. Ideally you should have a balanced dataset but if you do not you should consider computing the class weight for each classes.

2. It's very unlikely that you'll obtain 100% accuracy and in most situations, not desirable as pure 100% accuracy likely indicates overfitting. There are many methods you can to boost your accuracy. I cover my best practices and techniques to increase accuracy inside Deep Learning for Computer vision with Python.

3. Yes, LeNet was originally used with the entire MNIST dataset which included 60,000 training examples and 10,000 testing samples.

4. Sometimes yes, sometimes no. It would depend on the other hyperparameters you are using.

Sourabh Mane May 29, 2018 at 1:33 am #

REPLY

Thanks for the reply.

Sourabh Mane May 28, 2018 at 1:12 am #

REPLY

Error has been solved.It was occurring due to corrupted images which was present in the dataset.

Noor May 29, 2018 at 11:32 pm #

REPLY

Hi, one of the best tutorial out there. I have a question, what if i want to detect "Santa" "not Santa" in real time via webcam with label and score? How can i implement it with your code. Thanks in advanced

Adrian Rosebrock May 31, 2018 at 5:12 am #

REPLY

Be sure to see the followup to this post where I do implement this method in real-time. You can find the post here.

David Hill May 31, 2018 at 11:34 pm #

REPLY

Adrian, you are the best resource anywhere for this stuff, I can't thank you enough. I got your sample running perfectly. Then I deleted the model that came with the download and re-built it from scratch using the training script. My result is much degraded, not nearly as accurate as yours (the Manhattan image scored 90% Santa). To train the supplied model did you simply use many more epocs than 25 and possibly a lower learning rate, or did you also use more images to train your model? I'm trying to figure out how to get results like yours for a different target image. Also fyi I think you made a face detector here, I'm adding faces to my negative set, teddy bears score very high Santa rating. Thanks!

Adrian Rosebrock June 5, 2018 at 8:29 am #

REPLY

Hey David, I used the exact same data, network architecture, and training parameters as discussed in this blog post — nothing else was different. I'm not entirely sure why the Manhattan image scored 90% Santa but keep in mind that this is just an example image. If your accuracy and loss matches mine that is what you should be concerned about. Don't let yourself "overfit" to a single testing image by trying to tune your results to be identical to be 100% identical mine.

ANKUR June 2, 2018 at 4:59 pm #

REPLY

unable to run argparser part
can you give an alternative??

Adrian Rosebrock June 5, 2018 at 8:01 am #

REPLY

If you're new to Python command line arguments that's okay but you should read this post before giving up.

Dalia abo bakr June 4, 2018 at 7:23 am #

REPLY

Hi Adrian
thanks alot for this tutorials
I tried this code but i get an error
error: the following arguments are required: -i/–image
So how can i solve it?

Adrian Rosebrock June 5, 2018 at 7:51 am #

REPLY

Hey Dalia — make sure you read this blog post on command line arguments to solve your error.

Siladittya Manna June 5, 2018 at 4:05 am #

REPLY

How will the neural net behave if we provide very less number of images not belonging to “Santa”?
Suppose if the ratio of “Santa” to “Not-Santa” images is 9:1

Is it possible that the Neural net will show biased behaviour?

Adrian Rosebrock June 5, 2018 at 7:10 am #

REPLY

Try it and see! 😉 What you are referring to is class imbalances which can be a very real problem if your classes are not reasonably balanced. You can compute the class weight and weight underrepresented classes higher in an attempt to reweight them.

Sourabh Mane June 5, 2018 at 5:18 am #

REPLY

Hi Adrian,
Can we train this network on video clips?
If yes,then how can i implement it??

Adrian Rosebrock June 5, 2018 at 7:09 am #

REPLY

Can you clarify what you mean by “train this network on video clips”? The network discussed here is only for images or single frames. Depending on your project you may be able to train your network on single frames and then apply it to a video frame-by-frame.

Sourabh Mane June 6, 2018 at 3:29 am #

REPLY

I mean to say instead of jpg images can we pass .mp4 files, is it possible or i have to extract each frames from videos and prepare image dataset out of video and then train??

Sourabh Mane June 6, 2018 at 7:26 am #

REPLY

I have downloaded the ROSE-Youtu Face Liveness Detection Database (ROSE-Youtu) consists of 4225 videos with 25 subjects in total (3350 videos with 20 subjects publically available with 5.45GB in size).

I want to use this to differentiate between Real Faces & Fake Faces i.e for Face Recognition Anti-spoof.

So I’m using your (https://www.pyimagesearch.com/2017/12/11/image-classification-with-keras-and-deep-learning/) program for the same. In one folder(real) I’m dumping images of people and in another folder(fake) I’m dumping sketches/photos of Profile Pics(using different smartphone camera’s)/ .mp4 files from ROSE db.

I am unable to train this as real & fake. The .mp4 files aren’t been considered under Training, its skipped.

How do i make use of this data to create a model for Face Anti-spoof ?? Please help

REPLY

Adrian Rosebrock June 7, 2018 at 3:11 pm #

Keep in mind that a video file cannot be read via `cv2.imread`. You would need to either:

1. Use `cv2.VideoCapture` to load each frame from the video
2. Or use a tool like FFMPEG to extract all frames from the video file

REPLY

mukesh bhuriya June 5, 2018 at 1:20 pm #

orig = image.copy()
AttributeError: 'NoneType' object has no attribute 'copy'

sir i am getting this error.
can i get solution i am using pycharm

REPLY

Adrian Rosebrock June 7, 2018 at 3:17 pm #

Double-check your command line arguments. The path you supplied to the `--image` switch does not exist.

REPLY

Andrew July 5, 2018 at 4:59 pm #

For a classifier like this, is there a rule of thumb for how we should construct the "Not Santa" image set? For example, I downloaded a bunch of images of dolphins for my "Not Santa" set and when I ran the classifier on something like a dog, it was pretty sure that it was santa…likely due to the green in the image or the fact that there was no water or silver. So how should I be thinking about collecting images for the negative case? Simply get as diverse an image set as possible?

REPLY

Adrian Rosebrock July 10, 2018 at 8:58 am #

You need to:

1. Consider where the model will be deployed and what "negative" samples will look like
2. Devise a class that includes such negative examples
3. Train your model on that class

There's no "engineering" rule here, it's just taking the time to consider how and where this model will be used.

REPLY

Andrew July 12, 2018 at 6:18 pm #

Ok, that makes sense. So it's really all about context of the problem you are solving. I started thinking about something else as well. If you are constructing a "not santa" image set and use disproportionally more images for that class to try and cover more things that aren't santa, do you have any issues with your classifier thinking ground truth is that santa is say 10% if only 10% of my images are santa?

Thank you again for your response and this post!

Adrian Rosebrock July 13, 2018 at 4:54 am #

REPLY

If you have a very large class imbalance like the one you suggested you should apply weight balancing to help correct the problem. Inside Keras you can compute the "class weight" and then weigh the samples such that they are equal during the updating phase. I discuss this technique, including other fundamentals of deep learning inside Deep Learning for Computer Vision with Python. Be sure to take a look, I believe it will help you quite a bit with your studies.

Krishna Raj July 9, 2018 at 9:47 am #

REPLY

Hello Adrian,

Great tutorial.
A problem I ran into is, when I ran the test on the manhattan skyline example with the model I trained on my pc, it shows a false positive with 87% santa. I used the same downloaded module.
I tried varying the epoch numbers and even then, the problem persists. Could you please take a guess at what is happening?
Also, my percentage rates are not the same as yours for the examples either.

However, the santa_or_not_santa.model that you provided with the downloads, ie, the pre trained one, works fine.
The one i tried to train on mine using your same code and commands as stated above gives me this error.

(I am not using tensorflow-gpu, just the CPU)
Thanks

Adrian Rosebrock July 10, 2018 at 8:24 am #

REPLY

Keep in mind that this is a small model trained on a small dataset. Random weight initializations can make a big difference in the output model. Secondly, your differences between TensorFlow and Keras may also be causing slight differences. NNs are stochastic algorithms and your results will never be 100% identical to mine (but ideally within some tolerance).

Tux August 3, 2018 at 9:55 am #

REPLY

Hi Adrian and thanks a lot !

Would this method actually work text, such as detecting chinese characters ?

Thanks !

Adrian Rosebrock August 7, 2018 at 7:05 am #

REPLY

You would need to train a model to detect/recognize Chinese characters, but yes, using this method it could be possible. Keep in mind you may need to tune the model and hyperparameters though.

Tsuichi August 17, 2018 at 2:57 am #

REPLY

thanks for this great tutorial

1. what kinds of LeNet you using ?
2. i'm get error or warning because "T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2" So how can i solve it?

sorry for this question, I hope you'll reply.

thanks

Adrian Rosebrock August 17, 2018 at 7:12 am #

REPLY

1. The LeNet implementation is covered in the post. You can see how I implemented LeNet. Make sure you read the post.
2. This isn't an error, it's a warning, and it's just a message from TensorFlow saying how you may be able to improve performance. Ignore it and proceed with the script.

Tsuichi August 17, 2018 at 6:41 pm #

REPLY

thanks for your reply adrian

1. after I read it, you using the Seminal LeNet Architectur, right? but i don't know it because i'm only know LeNet-1, LeNet-4, and LeNet-5. what's the Seminal LeNet Architectur ?

thanks.

Adrian Rosebrock August 22, 2018 at 10:20 am #

REPLY

Typically, we refer to "LeNet-5" as the seminal LeNet architecture.

Aniket August 23, 2018 at 3:52 am #

REPLY

Hi Adrian,

I really appreciate your effort with the code explanations. where as I have 1 question I would like you to answer for me.

1. Here “label = imagePath.split(os.path.sep)[-2]”, can you tell me what -2 represents. Is it the no. of classes or something else. I am using 9 classes in my case. so to label the 9 cases I have created labels for each but just to understand better, what does that -2 represent.

Adrian Rosebrock August 24, 2018 at 8:43 am #

REPLY

No, the “-2” is an array index value. Refer to this thread.

Vishal August 27, 2018 at 8:17 am #

REPLY

if i change the image to fruits will this code work to differentiate between the two fruits?

Adrian Rosebrock August 30, 2018 at 9:17 am #

REPLY

The code will run and execute for any images you provide it. However, whether or not it “works” and produces the results you expect is heavily dependent on your training data.

Trackbacks/Pingbacks

Keras and deep learning on the Raspberry Pi - PyImageSearch - December 18, 2017
[…] week, we learned how to train a Convolutional Neural Network using Keras to determine if Santa was in an input […]

Leave a Reply

Name (required)

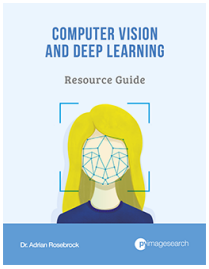
Email (will not be published) (required)

Website

SUBMIT COMMENT

Search...

Resource Guide (it's totally free).



Get your **FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF.** Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

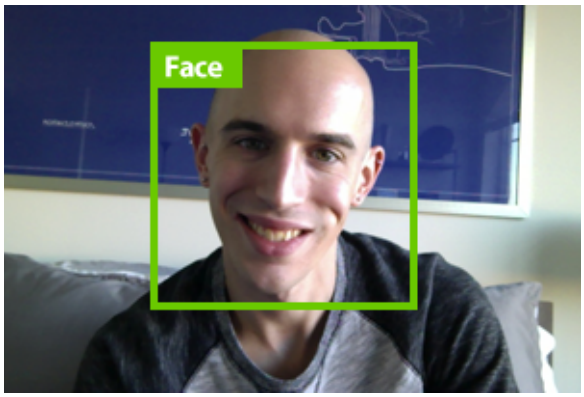
Download for Free!

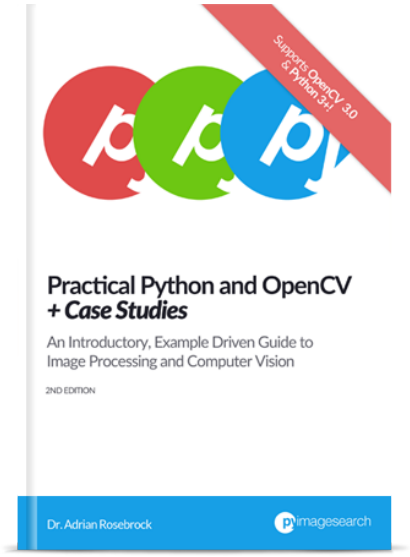
Deep Learning for Computer Vision with Python Book — OUT NOW!

You're interested in deep learning and computer vision, *but you don't know how to get started.* Let me help. **My new book will teach you all you need to know about deep learning.**

CLICK HERE TO MASTER DEEP LEARNING

You can detect faces in images & video.





Want to learn computer vision & OpenCV? I can teach you in a **single weekend**. I know. It sounds crazy, but it's no joke. My new book is your **guaranteed, quick-start guide** to becoming an OpenCV Ninja. So why not give it a try? Click here to become a computer vision ninja.

CLICK HERE TO BECOME AN OPENCV NINJA

Subscribe via RSS

Never miss a post! Subscribe to the PyImageSearch RSS Feed and keep up to date with my image search engine tutorials, tips, and tricks

POPULAR

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3
APRIL 18, 2016

Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi
SEPTEMBER 4, 2017

Install OpenCV and Python on your Raspberry Pi 2 and B+
FEBRUARY 23, 2015

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox
JUNE 1, 2015

Ubuntu 16.04: How to install OpenCV
OCTOBER 24, 2016

How to install OpenCV 3 on Raspbian Jessie
OCTOBER 26, 2015

Basic motion detection and tracking with Python and OpenCV
MAY 25, 2015

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.