

PYIMAGESearch

[Click here to download the source code to this post](#)

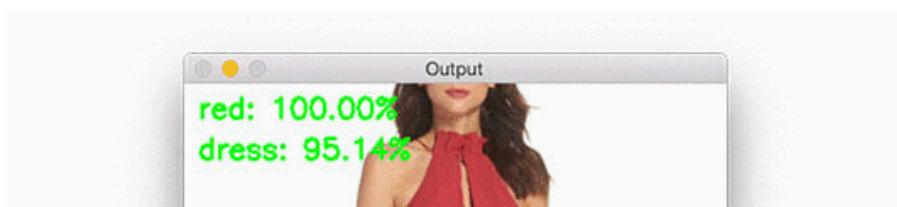
[DEEP LEARNING \(HTTPS://PYIMAGESEARCH.COM/CATEGORY/DEEP-LEARNING/\)](https://pyimagesearch.com/category/deep-learning/)

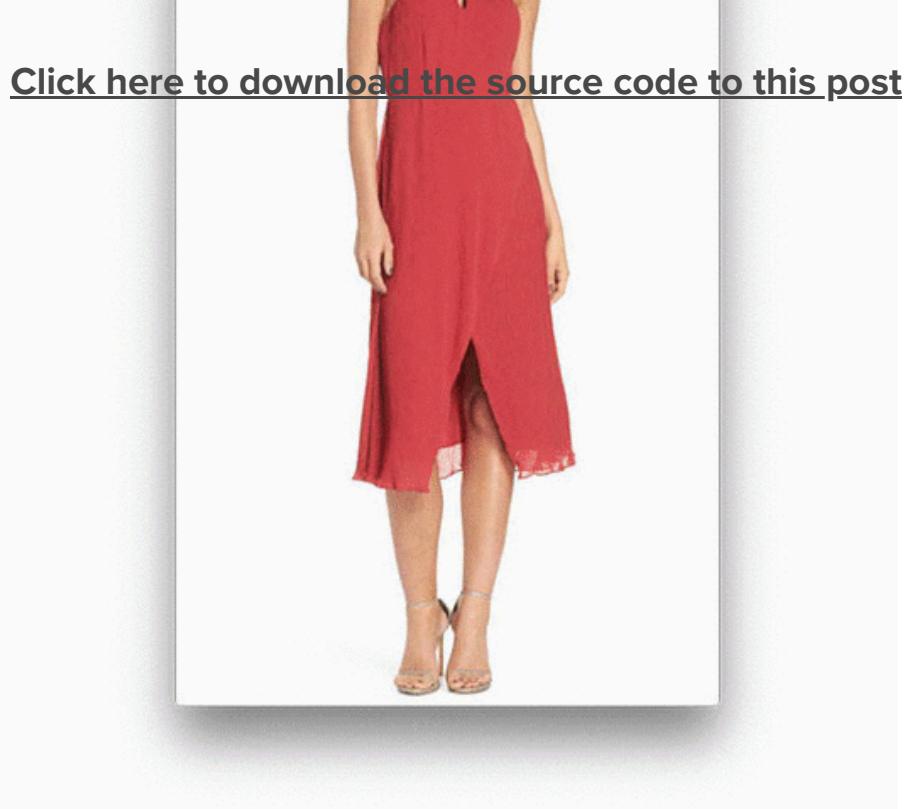
[KERAS AND TENSORFLOW \(HTTPS://PYIMAGESEARCH.COM/CATEGORY/KERAS-AND-TENSORFLOW/\)](https://pyimagesearch.com/category/keras-and-tensorflow/)

[TUTORIALS \(HTTPS://PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/\)](https://pyimagesearch.com/category/tutorials/)

Multi-label classification with Keras

by Adrian Rosebrock (<https://pyimagesearch.com/author/adrian/>) on May 7, 2018





[Click here to download the source code to this post](#)

Today's blog post on multi-label classification with Keras was inspired from an email I received last week from PyImageSearch reader, Switaj.

Switaj writes:

“

Hi Adrian, thanks for the PyImageSearch blog and sharing your knowledge each week.

I'm building an image fashion search engine and need help.

[Click here to download the source code to this post](#)

Using my app a user will upload a photo of clothing they like (ex. shirt, dress, pants, shoes) and my system will return similar items and include links for them to purchase the clothes online.

The problem is that I need to train a classifier to categorize the items into various classes:

- 1 **Clothing type:** Shirts, dresses, pants, shoes, etc.
- 2 **Color:** Red, blue, green, black, etc.
- 3 **Texture/appearance:** Cotton, wool, silk, tweed, etc.

I've trained **three separate CNNs** for each of the three categories and they work really well.

Is there a way to combine the three CNNs into a single network? Or at least train a single network to complete all three classification tasks?

I don't want to have to apply them individually in a cascade of if/else code that uses a different network depending on the output of a previous classification.

Thanks for your help.

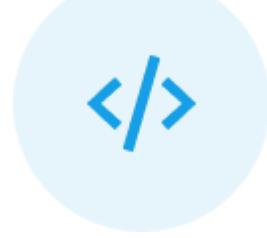
Switaj poses an excellent question:

Is it possible for a Keras deep neural network to return multiple predictions?

And if so, how is it done?

To learn how to perform multi-label classification with Keras, just keep reading.

[Click here to download the source code to this post](#)



Looking for the source code to this post?

JUMP RIGHT TO THE DOWNLOADS SECTION →

Multi-label classification with Keras

2020-06-12 Update: This blog post is now TensorFlow 2+ compatible!

Today's blog post on multi-label classification is broken into four parts.

In the first part, I'll discuss our multi-label classification dataset (and how you can build your own quickly).

From there we'll briefly discuss `SmallerVGGNet`, the Keras neural network architecture we'll be implementing and using for multi-label classification.

We'll then take our implementation of `SmallerVGGNet` and train it using our multi-label classification dataset.

Finally, we'll wrap up today's blog post by testing our network on example images and discuss when multi-label classification is appropriate, including a few caveats you need to look out for.

Our multi-label classification dataset





Figure 1: A montage of a multi-class deep learning dataset. We'll be using Keras to train a multi-label classifier to predict both the *color* and the *type* of clothing.

The dataset we'll be using in today's Keras multi-label classification tutorial is meant to mimic Switaj's question at the top of this post (although slightly simplified for the sake of the blog post).

Our dataset consists of **2,167** images across **six categories**, including:

- Black jeans (344 images)
- Blue dress (386 images)
- Blue jeans (356 images)
- Blue shirt (369 images)
- Red dress (380 images)
- Red shirt (332 images)

The goal of our Convolutional Neural network will be to predict both color and clothing type.

I created this dataset by following my previous tutorial on [**How to \(quickly\) build a**](#)

[deep learning image dataset \(<https://pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/>\)](https://pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/)

[Click here to download the source code to this post](#)

The entire process of downloading the images and manually removing irrelevant images for each of the six classes took approximately 30 minutes.

When trying to build your own deep learning image datasets, make sure you follow the tutorial linked above — it will give you a *huge* jumpstart on building your own datasets.

Configuring your development environment

To configure your system for this tutorial, I recommend following either of these tutorials:

- [How to install TensorFlow 2.0 on Ubuntu](https://pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-ubuntu/)
[\(https://pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-ubuntu/\)](https://pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-ubuntu/)
- [How to install TensorFlow 2.0 on macOS](https://pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-macos/)
[\(https://pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-macos/\)](https://pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-macos/)

Either tutorial will help you configure your system with all the necessary software for this blog post in a convenient Python virtual environment.

Please note that [PyImageSearch does not recommend or support Windows for CV/DL projects \(<https://pyimagesearch.com/faqs/single-faq/can-you-help-me-do-on-windows/>\)](https://pyimagesearch.com/faqs/single-faq/can-you-help-me-do-on-windows/).

Multi-label classification project structure

Go ahead and visit the “**Downloads**” section of this blog post to grab the code + files. Once you’ve extracted the zip file, you’ll be presented with the following

directory structure:

[**Click here to download the source code to this post**](#)

→ [**Launch Jupyter Notebook on Google Colab**](#)

Multi-label classification with Keras

```

1. |   └── classify.py
2. |   └── dataset
3. |       |   └── black_jeans [344 entries]
4. |       |   └── blue_dress [386 entries]
5. |       |   └── blue_jeans [356 entries]
6. |       |   └── blue_shirt [369 entries]
7. |       |   └── red_dress [380 entries]
8. |       |   └── red_shirt [332 entries]
9. |   └── examples
10. |       |   └── example_01.jpg
11. |       |   └── example_02.jpg
12. |       |   └── example_03.jpg
13. |       |   └── example_04.jpg
14. |       |   └── example_05.jpg
15. |       |   └── example_06.jpg
16. |       |   └── example_07.jpg
17. |   └── fashion.model
18. |   └── mlb.pickle
19. |   └── plot.png
20. |   └── pyimagesearch
21. |       |   └── __init__.py
22. |       |   └── smallervggnet.py
23. |   └── search_bing_api.py
24. |   └── train.py

```

In the root of the zip, you're presented with **6 files** and **3 directories**.

The important files we're working with (in approximate order of appearance in this article) include:

- 1 `search_bing_api.py` : This script enables us to [**quickly build our deep learning image dataset \(<https://pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/>\)**](#). You do not need to run this script as the dataset of images has been included in the zip archive. I'm simply including this script as a matter of completeness.
- 2 `train.py` : Once we've acquired the data, we'll use the `train.py` script to train our classifier.
- 3 `fashion.model` : Our `train.py` script will serialize our Keras model to disk. We will use this model later in the `classify.py` script.
- 4 `mlb.pickle` : A scikit-learn MultiLabelBinarizer pickle file created by

`train.py` — this file holds our class names in a convenient serialized data structure. [**Click here to download the source code to this post**](#)

- 5 `plot.png` : The training script will generate a `plot.png` image file. If you're training on your own dataset, you'll want to check this file for accuracy/loss and overfitting.
- 6 `classify.py` : In order to test our classifier, I've written `classify.py`. You should always test your classifier locally before deploying the model elsewhere (such as to an [iPhone deep learning app](#) (<https://pyimagesearch.com/2018/04/23/running-keras-models-on-ios-with-coreml/>) or to a [Raspberry Pi deep learning project](#) (<https://pyimagesearch.com/2018/04/30/a-fun-hands-on-deep-learning-project-for-beginners-students-and-hobbyists/>)).

The three directories in today's project are:

- 1 `dataset` : This directory holds our dataset of images. Each class class has its own respective subdirectory. We do this to (1) keep our dataset organized and (2) make it easy to extract the class label name from a given image path.
- 2 `pyimagesearch` : This is our module containing our Keras neural network. Because this is a module, it contains a properly formatted `__init__.py`. The other file, `smallervggnet.py` contains the code to assemble the neural network itself.
- 3 `examples` : Seven example images are present in this directory. We'll use `classify.py` to perform multi-label classification with Keras on each of the example images.

If this seems a lot, don't worry! We'll be reviewing the files in the approximate order in which I've presented them.

Our Keras network architecture for multi-label classification

[Click here to download the source code to this post](#)

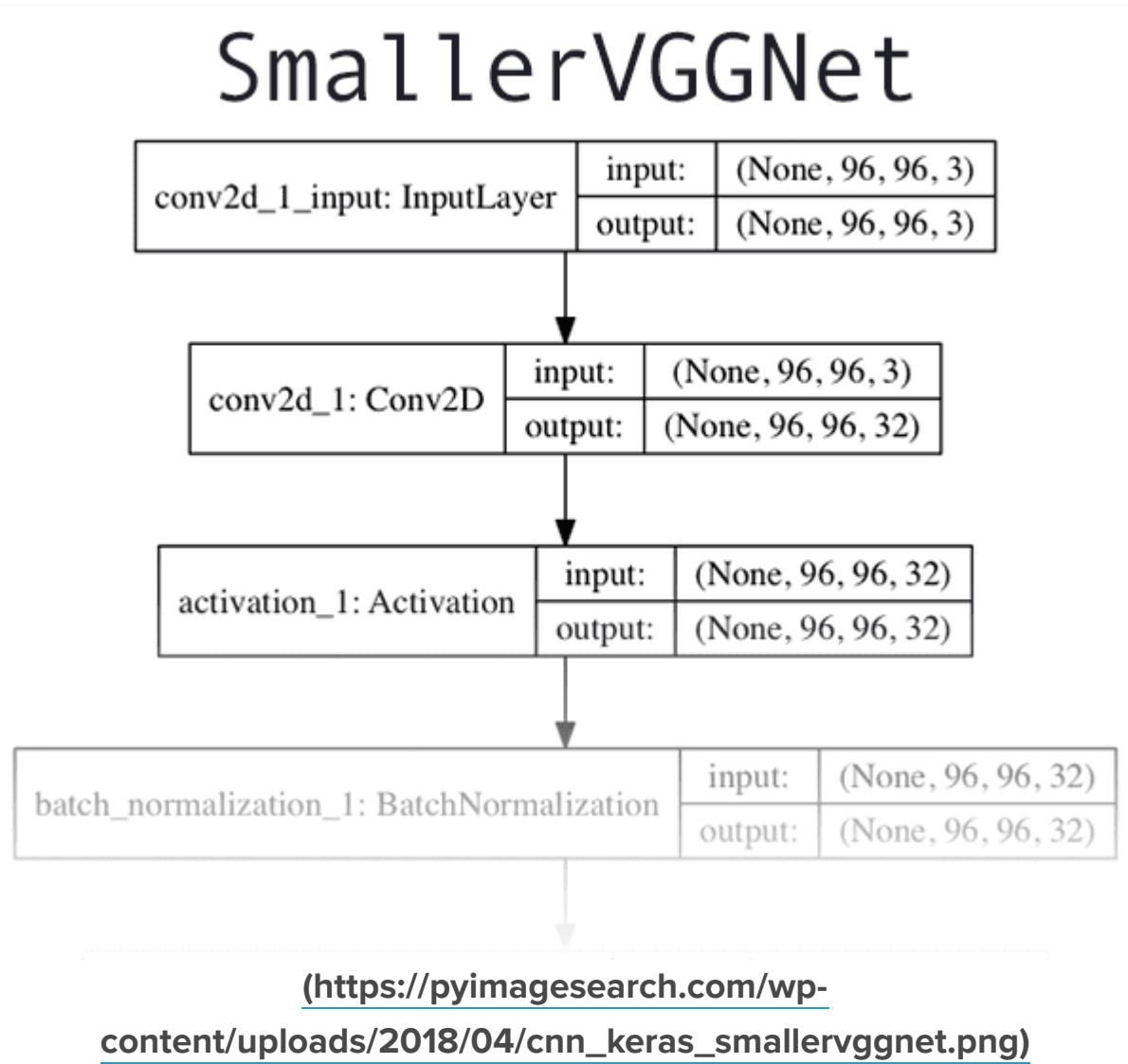


Figure 2: A VGGNet-like network that I've dubbed “SmallerVGGNet” will be used for training a multi-label deep learning classifier with Keras.

The CNN architecture we are using for this tutorial is `SmallerVGGNet`, a simplified version of its big brother, `VGGNet`. The `VGGNet` model was first introduced by Simonyan and Zisserman in their 2014 paper, [Very Deep Convolutional Networks for Large Scale Image Recognition](#) (<https://arxiv.org/pdf/1409.1556.pdf>).

As a matter of completeness we are going to implement `SmallerVGGNet` in this guide; however, I'm going to defer any lengthy explanation of the architecture/code to my [previous post](#) (<https://pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-tutorial/>) — please refer to it if you have any questions.

[CONVOLUTIONAL-NEURAL-NETWORKS-CH11/1](#) — please refer to it if you have any questions on the architecture or are simply looking for more detail. If you're looking to design your own models, you'll want to pick up a copy of my book, [Deep Learning for Computer Vision with Python](#) (<https://pyimagesearch.com/deep-learning-computer-vision-python-book/>).

Ensure you've used the “**Downloads**” section at the bottom of this blog post to grab the source code + example images. From there, open up the `smallervggnet.py` file in the `pyimagesearch` module to follow along:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```
1. # import the necessary packages
2. from tensorflow.keras.models import Sequential
3. from tensorflow.keras.layers import BatchNormalization
4. from tensorflow.keras.layers import Conv2D
5. from tensorflow.keras.layers import MaxPooling2D
6. from tensorflow.keras.layers import Activation
7. from tensorflow.keras.layers import Flatten
8. from tensorflow.keras.layers import Dropout
9. from tensorflow.keras.layers import Dense
10. from tensorflow.keras import backend as K
```

On **Lines 2-10**, we import the relevant Keras modules and from there, we create our `SmallerVGGNet` class:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```
12. class SmallerVGGNet:
13.     @staticmethod
14.     def build(width, height, depth, classes, finalAct="softmax"):
15.         # initialize the model along with the input shape to be
16.         # "channels last" and the channels dimension itself
17.         model = Sequential()
18.         inputShape = (height, width, depth)
19.         chanDim = -1
20.
21.         # if we are using "channels first", update the input shape
22.         # and channels dimension
23.         if K.image_data_format() == "channels_first":
24.             inputShape = (depth, height, width)
25.             chanDim = 1
```

Our class is defined on **Line 12**. We then define the `build` function on **Line 14**, responsible for assembling the convolutional neural network.

The `build` method requires four parameters — `width` , `height` , `depth` , and

classes . The depth specifies the number of channels in an input image, and classes is the number (integer) of categories/classes (not the class labels themselves). We'll use these parameters in our training script to instantiate the model with a $96 \times 96 \times 3$ input volume.

The optional argument, finalAct (with a default value of "softmax") will be utilized **at the end** of the network architecture. Changing this value from softmax to sigmoid will enable us to perform multi-label classification with Keras.

Keep in mind that this behavior is **different** than our original implementation of SmallerVGGNet in our previous post — we are adding it here so we can control whether we are performing simple classification or multi-class classification.

From there, we enter the body of build , initializing the model (**Line 17**) and defaulting to "channels_last" architecture on **Lines 18 and 19** (with a convenient switch for backends that support "channels_first" architecture on **Lines 23-25**).

Let's build the first CONV => RELU => POOL block:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

27. |     # CONV => RELU => POOL
28. |     model.add(Conv2D(32, (3, 3), padding="same",
29. |                     input_shape=inputShape))
30. |     model.add(Activation("relu"))
31. |     model.add(BatchNormalization(axis=chanDim))
32. |     model.add(MaxPooling2D(pool_size=(3, 3)))
33. |     model.add(Dropout(0.25))

```

Our CONV layer has 32 filters with a 3×3 kernel and RELU activation (Rectified Linear Unit). We apply batch normalization, max pooling, and 25% dropout.

Dropout is the process of randomly disconnecting nodes from the *current* layer to the next layer. This process of random disconnects naturally helps the network to reduce overfitting as no one single node in the layer will be responsible for predicting a

certain class, object, edge, or corner.

From there we have two sets of (CONV => RELU) * 2 => POOL blocks:

→ [Launch Jupyter Notebook on Google Colab](#)

[Click here to download the source code to this post](#)

```
Multi-label classification with Keras
35.     # (CONV => RELU) * 2 => POOL
36.     model.add(Conv2D(64, (3, 3), padding="same"))
37.     model.add(Activation("relu"))
38.     model.add(BatchNormalization(axis=chanDim))
39.     model.add(Conv2D(64, (3, 3), padding="same"))
40.     model.add(Activation("relu"))
41.     model.add(BatchNormalization(axis=chanDim))
42.     model.add(MaxPooling2D(pool_size=(2, 2)))
43.     model.add(Dropout(0.25))
44.
45.     # (CONV => RELU) * 2 => POOL
46.     model.add(Conv2D(128, (3, 3), padding="same"))
47.     model.add(Activation("relu"))
48.     model.add(BatchNormalization(axis=chanDim))
49.     model.add(Conv2D(128, (3, 3), padding="same"))
50.     model.add(Activation("relu"))
51.     model.add(BatchNormalization(axis=chanDim))
52.     model.add(MaxPooling2D(pool_size=(2, 2)))
53.     model.add(Dropout(0.25))
```

Notice the numbers of filters, kernels, and pool sizes in this code block which work together to progressively reduce the spatial size but increase depth.

These blocks are followed by our only set of `FC => RELU` layers:

→ [Launch Jupyter Notebook on Google Colab](#)

```
Multi-label classification with Keras
55.     # first (and only) set of FC => RELU layers
56.     model.add(Flatten())
57.     model.add(Dense(1024))
58.     model.add(Activation("relu"))
59.     model.add(BatchNormalization())
60.     model.add(Dropout(0.5))
61.
62.     # softmax classifier
63.     model.add(Dense(classes))
64.     model.add(Activation(finalAct))
65.
66.     # return the constructed network architecture
67.     return model
```

Fully connected layers are placed at the end of the network (specified by `Dense` on **Lines 57 and 63**).

Line 64 is **important** for our multi-label classification — `finalAct` dictates whether we'll use "softmax" activation for single-label classification or "sigmoid" activation **in the case of today's multi-label classification**. Refer to **Line 14** of this

script, `smallervggnet.py` and `Line 25` of `train.py`.

[Click here to download the source code to this post](#)

Implementing our Keras model for multi-label classification

Now that we have implemented `SmallerVGGNet`, let's create `train.py`, the script we will use to train our Keras network for multi-label classification.

I urge you to review the [previous post](#)

(<https://pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/>) upon which today's `train.py` script is based. In fact, you may want to view them on your screen side-by-side to see the difference and read full explanations. Today's review will be succinct in comparison.

Open up `train.py` and insert the following code:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

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 tensorflow.keras.preprocessing.image import ImageDataGenerator
7. | from tensorflow.keras.optimizers import Adam
8. | from tensorflow.keras.preprocessing.image import img_to_array
9. | from sklearn.preprocessing import MultiLabelBinarizer
10. | from sklearn.model_selection import train_test_split
11. | from pyimagesearch.smallervggnet import SmallerVGGNet
12. | import matplotlib.pyplot as plt
13. | from imutils import paths
14. | import tensorflow as tf
15. | import numpy as np
16. | import argparse
17. | import random
18. | import pickle
19. | import cv2
20. | import os

```

On **Lines 2-20** we import the packages and modules required for this script. **Line 3** specifies a matplotlib backend so that we can save our plot figure in the background.

I'll be making the assumption that you have Keras, scikit-learn, matplotlib, imutils

and OpenCV installed at this point. Be sure to refer to the “[Configuring your development environment](#)” section above. [Click here to download the source code to this post](#)

Now that (a) your environment is ready, and (b) you’ve imported packages, let’s parse command line arguments:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

22. # construct the argument parse and parse the arguments
23. ap = argparse.ArgumentParser()
24. ap.add_argument("-d", "--dataset", required=True,
25.     help="path to input dataset (i.e., directory of images)")
26. ap.add_argument("-m", "--model", required=True,
27.     help="path to output model")
28. ap.add_argument("-l", "--labelbin", required=True,
29.     help="path to output label binarizer")
30. ap.add_argument("-p", "--plot", type=str, default="plot.png",
31.     help="path to output accuracy/loss plot")
32. args = vars(ap.parse_args())

```

Command line arguments to a script are like parameters to a function — if you don’t understand this analogy then you need to read up on [command line arguments](#) (<https://pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/>).

We’re working with four command line arguments (**Lines 23-32**) today:

- 1 --dataset : The path to our dataset.
- 2 --model : The path to our output serialized Keras model.
- 3 --labelbin : The path to our output multi-label binarizer object.
- 4 --plot : The path to our output plot of training loss and accuracy.

Be sure to refer to the [previous post](#)

(<https://pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/>) as needed for explanations of these arguments.

Let's move on to initializing some important variables that play critical roles in our training process:[Click here to download the source code to this post](#)

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```
34. # initialize the number of epochs to train for, initial learning rate,
35. # batch size, and image dimensions
36. EPOCHS = 30
37. INIT_LR = 1e-3
38. BS = 32
39. IMAGE_DIMS = (96, 96, 3)
40.
41. # disable eager execution
42. tf.compat.v1.disable_eager_execution()
```

These variables on **Lines 36-39** define that:

- Our network will train for 75 `EPOCHS` in order to learn patterns by incremental improvements via backpropagation.
- We're establishing an initial learning rate of `1e-3` (the default value for the Adam optimizer).
- The batch size is `32`. You should adjust this value depending on your GPU capability if you're using a GPU but I found a batch size of `32` works well for this project.
- As stated above, our images are `96 x 96` and contain `3` channels.

Additional information on hyperparameters is provided in the [previous post](#) (<https://pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/>).

Line 42 disables [TensorFlow's Eager Execution mode](#) (<https://www.tensorflow.org/guide/eager>). We found that this was necessary during our **2020-06-12 Update** to achieve the same accuracy as on the original publication date of this article.

From there, the next two code blocks handle loading and preprocessing our training data:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

44. # grab the image paths and randomly shuffle them
45. print("[INFO] loading images...")
46. imagePaths = sorted(list(paths.list_images(args["dataset"])))
47. random.seed(42)
48. random.shuffle(imagePaths)
49.
50. # initialize the data and labels
51. data = []
52. labels = []

```

Here we are grabbing the `imagePaths` and shuffling them randomly, followed by initializing `data` and `labels` lists.

Next, we're going to loop over the `imagePaths`, preprocess the image data, and extract multi-class-labels.

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

54. # loop over the input images
55. for imagePath in imagePaths:
56.     # load the image, pre-process it, and store it in the data list
57.     image = cv2.imread(imagePath)
58.     image = cv2.resize(image, (IMAGE_DIMS[1], IMAGE_DIMS[0]))
59.     image = img_to_array(image)
60.     data.append(image)
61.
62.     # extract set of class labels from the image path and update the
63.     # labels list
64.     l = label = imagePath.split(os.path.sep)[-2].split("_")
65.     labels.append(l)

```

First, we load each image into memory (**Line 57**). Then, we perform preprocessing (an important step of the deep learning pipeline) on **Lines 58 and 59**. We append the image to `data` (**Line 60**).

Lines 64 and 65 handle splitting the image path into multiple labels for our multi-label classification task. After **Line 64** is executed, a 2-element list is created and is then appended to the labels list on **Line 65**. Here's an example broken down in the terminal so you can see what's going on during the multi-label parsing:

→ [Launch Jupyter Notebook on Google Colab](#)

```
Multi-label classification with Keras
1. | $ python
2. | >>> import Click here to download the source code to this post
3. | >>> labels = []
4. | >>> imagePath = "dataset/red_dress/long_dress_from_macys_red.png"
5. | >>> l = label = imagePath.split(os.path.sep)[-2].split("_")
6. | >>> l
7. | ['red', 'dress']
8. | >>> labels.append(l)
9. |
10. | >>> imagePath = "dataset/blue_jeans/stylish_blue_jeans_from_your_favorite_store.png"
11. | >>> l = label = imagePath.split(os.path.sep)[-2].split("_")
12. | >>> labels.append(l)
13. |
14. | >>> imagePath = "dataset/red_shirt/red_shirt_from_target.png"
15. | >>> l = label = imagePath.split(os.path.sep)[-2].split("_")
16. | >>> labels.append(l)
17. |
18. | >>> labels
19. | [['red', 'dress'], ['blue', 'jeans'], ['red', 'shirt']]
```

As you can see, the `labels` list is a “list of lists” — each element of `labels` is a 2-element list. The two labels for each list is constructed based on the file path of the input image.

We’re not quite done with preprocessing:

→ [Launch Jupyter Notebook on Google Colab](#)

```
Multi-label classification with Keras
67. | # scale the raw pixel intensities to the range [0, 1]
68. | data = np.array(data, dtype="float") / 255.0
69. | labels = np.array(labels)
70. | print("[INFO] data matrix: {} images {:.2f}MB".format(
71. |     len(imagePaths), data nbytes / (1024 * 1000.0)))
```

Our `data` list contains images stored as NumPy arrays. In a single line of code, we convert the list to a NumPy array and scale the pixel intensities to the range `[0, 1]`.

We also convert labels to a NumPy array as well.

From there, let’s binarize the labels — the below block is ***critical*** for this week’s multi-class classification concept:

→ [Launch Jupyter Notebook on Google Colab](#)

```
Multi-label classification with Keras
73. | # binarize the labels using scikit-learn's special multi-label
74. | # binarizer implementation
```

```

75. | print("[INFO] class labels:")
76. | mlb = MultiLabelBinarizer()
77. | labels = mlb.fit_transform(class_labels)
78. |
79. | # loop over each of the possible class labels and show them
80. | for (i, label) in enumerate(mlb.classes_):
81. |     print("{}: {}".format(i + 1, label))

```

In order to binarize our labels for multi-class classification, we need to utilize the scikit-learn library's [MultiLabelBinarizer](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.html) (<http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.html>) class. You *cannot* use the standard `LabelBinarizer` class for multi-class classification. **Lines 76 and 77** fit and transform our human-readable labels into a vector that encodes which class(es) are present in the image.

Here's an example showing how `MultiLabelBinarizer` transforms a tuple of `("red", "dress")` to a vector with six total categories:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

1. | $ python
2. | >>> from sklearn.preprocessing import MultiLabelBinarizer
3. | >>> labels = [
4. | ...     ("blue", "jeans"),
5. | ...     ("blue", "dress"),
6. | ...     ("red", "dress"),
7. | ...     ("red", "shirt"),
8. | ...     ("blue", "shirt"),
9. | ...     ("black", "jeans")
10. | ...
11. | >>> mlb = MultiLabelBinarizer()
12. | >>> mlb.fit(labels)
13. | MultiLabelBinarizer(classes=None, sparse_output=False)
14. | >>> mlb.classes_
15. | array(['black', 'blue', 'dress', 'jeans', 'red', 'shirt'], dtype=object)
16. | >>> mlb.transform([("red", "dress")])
17. | array([[0, 0, 1, 0, 1, 0]])

```

One-hot encoding transforms categorical labels from a single integer to a vector. The same concept applies to **Lines 16 and 17** except this is a case of two-hot encoding.

Notice how on **Line 17** of the Python shell (not to be confused with the code blocks for `train.py`) two categorical labels are “hot” (represented by a “1” in the array), indicating the presence of each label. In this case “dress” and “red” are hot in the array. **Lines 16 and 17** All other labels have a value of “0”

array (Lines 14-17). All other labels have a value of 0.

Let's construct the training and testing splits as well as initialize the data augmenter:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

83. # partition the data into training and testing splits using 80% of
84. # the data for training and the remaining 20% for testing
85. (trainX, testX, trainY, testY) = train_test_split(data,
86.         labels, test_size=0.2, random_state=42)
87.
88. # construct the image generator for data augmentation
89. aug = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
90.     height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
91.     horizontal_flip=True, fill_mode="nearest")
```

Splitting the data for training and testing is common in machine learning practice — I've allocated 80% of the images for training data and 20% for testing data. This is handled by scikit-learn on **Lines 85 and 86**.

Our data augmenter object is initialized on **Lines 89-91**. Data augmentation is a best practice and a most-likely a “must” if you are working with less than 1,000 images per class.

Next, let's build the model and initialize the Adam optimizer:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

93. # initialize the model using a sigmoid activation as the final layer
94. # in the network so we can perform multi-label classification
95. print("[INFO] compiling model...")
96. model = SmallerVGGNet.build(
97.     width=IMAGE_DIMS[1], height=IMAGE_DIMS[0],
98.     depth=IMAGE_DIMS[2], classes=len(mlb.classes_),
99.     finalAct="sigmoid")
100.
101. # initialize the optimizer (SGD is sufficient)
102. opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
```

On **Lines 96-99** we build our `SmallerVGGNet` model, noting the `finalAct="sigmoid"` parameter indicating that we'll be performing **multi-label classification**.

From there, we'll compile the model and kick off training (this could take a while depending on your hardware). [Click here to download the source code to this post](#)

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

104. # compile the model using binary cross-entropy rather than
105. # categorical cross-entropy -- this may seem counterintuitive for
106. # multi-label classification, but keep in mind that the goal here
107. # is to treat each output label as an independent Bernoulli
108. # distribution
109. model.compile(loss="binary_crossentropy", optimizer=opt,
110.     metrics=["accuracy"])
111.
112. # train the network
113. print("[INFO] training network...")
114. H = model.fit(
115.     x=aug.flow(trainX, trainY, batch_size=BS),
116.     validation_data=(testX, testY),
117.     steps_per_epoch=len(trainX) // BS,
118.     epochs=EPOCHS, verbose=1)

```

2020-06-12 Update: Formerly, TensorFlow/Keras required use of a method called `.fit_generator` in order to accomplish data augmentation. Now, the `.fit` method can handle data augmentation as well, making for more-consistent code. This also applies to the migration from `.predict_generator` to `.predict`. Be sure to check out my articles about [**fit and fit_generator**](#)

(https://pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit_generator-a-hands-on-tutorial/) as well as [**data augmentation**](#) (<https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/>).

On **Lines 109 and 110** we compile the model using **binary cross-entropy** rather than **categorical cross-entropy**.

This may seem counterintuitive for multi-label classification; however, the goal is to treat each output label as an *independent Bernoulli distribution* and we want to *penalize each output node independently*.

From there we launch the training process with our data augmentation generator (**Lines 114-118**).

After training is complete we can save our model and label binarizer to disk:

→ [Launch Jupyter Notebook on Google Colab](#) code to this post

Multi-label classification with Keras

```
120. | # save the model to disk
121. | print("[INFO] serializing network...")
122. | model.save(args["model"], save_format="h5")
123.
124. | # save the multi-label binarizer to disk
125. | print("[INFO] serializing label binarizer...")
126. | f = open(args["labelbin"], "wb")
127. | f.write(pickle.dumps(mlb))
128. | f.close()
```

2020-06-12 Update: Note that for TensorFlow 2.0+ we recommend explicitly setting the `save_format="h5"` (HDF5 format).

From there, we plot accuracy and loss:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```
130. | # plot the training loss and accuracy
131. | plt.style.use("ggplot")
132. | plt.figure()
133. | N = EPOCHS
134. | plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
135. | plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
136. | plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
137. | plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
138. | plt.title("Training Loss and Accuracy")
139. | plt.xlabel("Epoch #")
140. | plt.ylabel("Loss/Accuracy")
141. | plt.legend(loc="upper left")
142. | plt.savefig(args["plot"])
```

2020-06-12 Update: In order for this plotting snippet to be TensorFlow 2+ compatible the `H.history` dictionary keys are updated to fully spell out “accuracy” sans “acc” (i.e., `H.history["val_accuracy"]` and `H.history["accuracy"]`). It is semi-confusing that “val” is not spelled out as “validation”; we have to learn to love and live with the API and always remember that it is a work in progress that many developers around the world contribute to.

Accuracy + loss for training and validation is plotted on **Lines 131-141**. The plot is saved as an image file on **Line 142**.

In my opinion, the training plot is just as important as the model itself. I typically go

through a few iterations of training and viewing the plot before I'm satisfied to share with you on the blog. [Click here to download the source code to this post](#)

I like to save plots to disk during this iterative process for a couple reasons: (1) I'm on a headless server and don't want to rely on X-forwarding, and (2) I don't want to forget to save the plot (even if I am using X-forwarding or if I'm on a machine with a graphical desktop).

Recall that we changed the matplotlib backend on **Line 3** of the script up above to facilitate saving to disk.

Training a Keras network for multi-label classification

Don't forget to use the "**Downloads**" section of this post to download the code, dataset, and pre-trained model (just in case you don't want to train the model yourself).

If you want to train the model yourself, open a terminal. From there, navigate to the project directory, and execute the following command:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```
1. | $ python train.py --dataset dataset --model fashion.model \
2. |   --labelbin mlb.pickle
3. | Using TensorFlow backend.
4. | [INFO] loading images...
5. | [INFO] data matrix: 2165 images (467.64MB)
6. | [INFO] class labels:
7. |   1. black
8. |   2. blue
9. |   3. dress
10. |  4. jeans
11. |  5. red
12. |  6. shirt
13. | [INFO] compiling model...
14. | [INFO] training network...
15. | Epoch 1/30
16. | 54/54 [=====] - 2s 35ms/step - loss: 0.3184 - accuracy: 0.8774 -
|   val_loss: 1.1824 - val_accuracy: 0.6251
17. | Epoch 2/30
18. | 54/54 [=====] - 2s 37ms/step - loss: 0.1881 - accuracy: 0.9427 -
|   val_loss: 1.4268 - val_accuracy: 0.6255
19. | Epoch 3/30
20. | 54/54 [=====] - 2s 38ms/step - loss: 0.1551 - accuracy: 0.9471 -
|   val_loss: 1.0533 - val_accuracy: 0.6305
21 |
```

```

...
22. Epoch 28/30
23. 54/54 [=====] - 2s 41ms/step - loss: 0.0656 - accuracy: 0.9763 -
val_loss: 0.0916 - val_accuracy: 0.9715
24. Epoch 29/30
25. 54/54 [=====] - 2s 40ms/step - loss: 0.0801 - accuracy: 0.9751 -
val_loss: 0.0500 - val_accuracy: 0.9823
26. Epoch 30/30
27. 54/54 [=====] - 2s 37ms/step - loss: 0.0636 - accuracy: 0.9770 -
val_loss: 0.0500 - val_accuracy: 0.9823
28. [INFO] serializing network...
29. [INFO] serializing label binarizer...

```

As you can see, we trained the network for 30 epochs, achieving:

- **97.70%** multi-label classification accuracy on the training set
- **98.23%** multi-label classification accuracy on the testing set

The training plot is shown in **Figure 3**:



(https://pyimagesearch.com/wp-content/uploads/2018/05/keras_multi_label_plot.png)

Figure 3: Our Keras deep learning multi-label classification accuracy/loss graph on the training and validation data

and validation data.

~~Click here to download the source code to this post~~ Applying Keras multi-label classification to new images

Now that our multi-label classification Keras model is trained, let's apply it to images *outside* of our testing set.

This script is quite similar to the `classify.py` script in my [previous post](#) (<https://pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/>) — be sure to look out for the multi-label differences.

When you're ready, open create a new file in the project directory named `classify.py` and insert the following code (or follow along with the file included with the “**Downloads**”):

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

1. # import the necessary packages
2. from tensorflow.keras.preprocessing.image import img_to_array
3. from tensorflow.keras.models import load_model
4. import numpy as np
5. import argparse
6. import imutils
7. import pickle
8. import cv2
9. import os
10.
11. # construct the argument parse and parse the arguments
12. ap = argparse.ArgumentParser()
13. ap.add_argument("-m", "--model", required=True,
14.     help="path to trained model model")
15. ap.add_argument("-l", "--labelbin", required=True,
16.     help="path to label binarizer")
17. ap.add_argument("-i", "--image", required=True,
18.     help="path to input image")
19. args = vars(ap.parse_args())

```

On **Lines 2-9** we import the necessary packages for this script. Notably, we'll be using Keras and OpenCV in this script.

Then we proceed to parse our three required command line arguments on **Lines 12-19**.

From there, we load and preprocess the input image:

→ [Launch Jupyter Notebook on Google Colab](#) code to this post

Multi-label classification with Keras

```

21. | # load the image
22. | image = cv2.imread(args["image"])
23. | output = imutils.resize(image, width=400)
24.
25. | # pre-process the image for classification
26. | image = cv2.resize(image, (96, 96))
27. | image = image.astype("float") / 255.0
28. | image = img_to_array(image)
29. | image = np.expand_dims(image, axis=0)
```

We take care to preprocess the image in the *same manner* as we preprocessed our training data.

Next, let's load the model + multi-label binarizer and classify the image:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

31. | # load the trained convolutional neural network and the multi-label
32. | # binarizer
33. | print("[INFO] loading network...")
34. | model = load_model(args["model"])
35. | mlb = pickle.loads(open(args["labelbin"], "rb").read())
36.
37. | # classify the input image then find the indexes of the two class
38. | # labels with the *largest* probability
39. | print("[INFO] classifying image...")
40. | proba = model.predict(image)[0]
41. | idxs = np.argsort(proba)[-1:-2]
```

We load the `model` and multi-label binarizer from disk into memory on **Lines 34 and 35**.

From there we classify the (preprocessed) input `image` (**Line 40**) and extract the top two class labels indices (**Line 41**) by:

- Sorting the array indexes by their associated probability in descending order
- Grabbing the first two class label indices which are thus the top-2 predictions from our network

You can modify this code to return more class labels if you wish. I would also suggest thresholding the probabilities and only returning labels with > N% confidence. [Click here to download the source code to this post](#)

From there, we'll prepare the class labels + associated confidence values for overlay on the output image:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```

43. # loop over the indexes of the high confidence class labels
44. for (i, j) in enumerate(idxs):
45.     # build the label and draw the label on the image
46.     label = "{}: {:.2f}%".format(mlb.classes_[j], proba[j] * 100)
47.     cv2.putText(output, label, (10, (i * 30) + 25),
48.                 cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
49.
50.     # show the probabilities for each of the individual labels
51.     for (label, p) in zip(mlb.classes_, proba):
52.         print("{}: {:.2f}%".format(label, p * 100))
53.
54.     # show the output image
55.     cv2.imshow("Output", output)
56.     cv2.waitKey(0)
```

The loop on **Lines 44-48** draws the top two multi-label predictions and corresponding confidence values on the `output` image.

Similarly, the loop on **Lines 51 and 52** prints the all the predictions in the terminal. This is useful for debugging purposes.

Finally, we show the `output` image on the screen (**Lines 55 and 56**).

Keras multi-label classification results

Let's put `classify.py` to work using command line arguments. You do not need to modify the code discussed above in order to pass new images through the CNN.

Simply use the [command line arguments](#)

[\(https://pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/\)](https://pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/) in your terminal as is shown below.

Let's try an image of a red dress — notice the three command line arguments that are processed at runtime:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label Classification with Keras

[Click here to download the source code to this post](#)

```
1. $ python classify.py --model fashion.model --labelbin mlb.pickle \
2.   --image examples/example_01.jpg
3. Using TensorFlow backend.
4. [INFO] loading network...
5. [INFO] classifying image...
6. black: 0.00%
7. blue: 3.58%
8. dress: 95.14%
9. jeans: 0.00%
10. red: 100.00%
11. shirt: 64.02%
```



(https://pyimagesearch.com/wp-content/uploads/2018/04/keras_multi_label_output_01.png)

Figure 4: The image of a red dress has correctly been classified as “red” and “dress” by our Keras multi-label classification deep learning script.

Success! Notice how the two classes (“red” and “dress”) are marked with high confidence.

Now let’s try a blue dress:

→ [Launch chapter notebook on Colab](#) or [Get the code to this post](#)

Multi-label classification with Keras

```
1. | $ python classify.py --model fashion.model --labelbin mlb.pickle \
2. |   --image examples/example_02.jpg
3. | Using TensorFlow backend.
4. | [INFO] loading network...
5. | [INFO] classifying image...
6. | black: 0.03%
7. | blue: 99.98%
8. | dress: 98.50%
9. | jeans: 0.23%
10. | red: 0.00%
11. | shirt: 0.74%
```



(https://pyimagesearch.com/wp-content/uploads/2018/04/keras_multi_label_output_02.png)

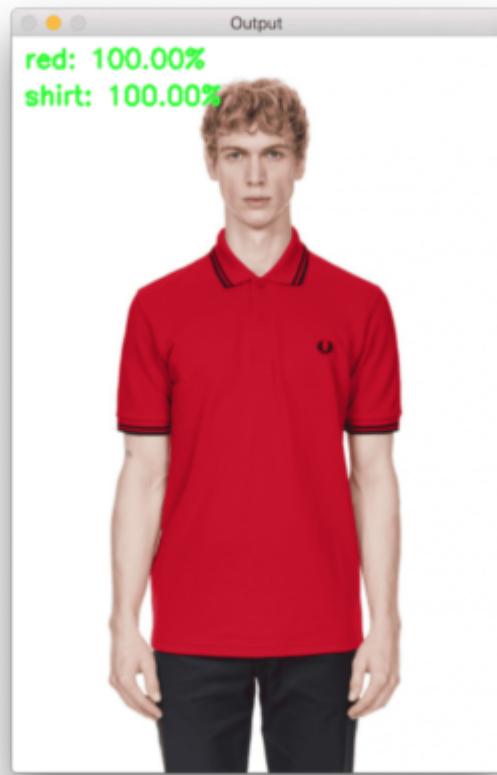
Figure 5: The “blue” and “dress” class labels are correctly applied in our second test of our Keras multi-label image classification project.

A blue dress was no contest for our classifier. We’re off to a good start, so let’s try an image of a red shirt:

→ [Launch Jupyter Notebook on Google Colab](#)
[Click here to download the source code to this post](#)

Multi-label classification with Keras

```
1. $ python classify.py --model fashion.model --labelbin mlb.pickle \
2.   --image examples/example_03.jpg
3. Using TensorFlow backend.
4. [INFO] loading network...
5. [INFO] classifying image...
6. black: 0.00%
7. blue: 0.69%
8. dress: 0.00%
9. jeans: 0.00%
10. red: 100.00%
11. shirt: 100.00%
```



(https://pyimagesearch.com/wp-content/uploads/2018/04/keras_multi_label_output_03.png)

Figure 6: With 100% confidence, our deep learning multi-label classification script has correctly classified this red shirt.

The red shirt result is promising.

How about a blue shirt?

→ [Click here to download the source code to this post](#)

Multi-label classification with Keras

```
1. | $ python classify.py --model fashion.model --labelbin mlb.pickle \
2. |   --image examples/example_04.jpg
3. | Using TensorFlow backend.
4. | [INFO] loading network...
5. | [INFO] classifying image...
6. | black: 0.00%
7. | blue: 99.99%
8. | dress: 22.59%
9. | jeans: 0.08%
10. | red: 0.00%
11. | shirt: 82.82%
```



(<https://pyimagesearch.com/wp->

[content/uploads/2018/04/keras_multi_label_o utput_04.png](https://pyimagesearch.com/2018/04/keras_multi_label_o utput_04.png))

Figure 7: Deep learning + multi-label + Keras classification of a blue shirt is correctly calculated

blue shirt is correctly classified.

Our model is very ~~confident~~ that it has encountered a shirt. That being said, this is still a correct multi-label classification!

Let's see if we can fool our multi-label classifier with blue jeans:

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```
1. | $ python classify.py --model fashion.model --labelbin mlb.pickle \
2. |   --image examples/example_05.jpg
3. | Using TensorFlow backend.
4. | [INFO] loading network...
5. | [INFO] classifying image...
6. | black: 0.00%
7. | blue: 100.00%
8. | dress: 0.01%
9. | jeans: 99.99%
10. | red: 0.00%
11. | shirt: 0.00%
```



(https://pyimagesearch.com/wp-content/uploads/2018/04/keras_multi_label_output_05.png)

Figure 8: This deep learning multi-label classification result proves that blue jeans can be correctly classified as both “blue” and “jeans”.

Click here to download the source code to this post

Let's try black jeans:

→ **Launch Jupyter Notebook on Google Colab**

Multi-label classification with Keras

```

1. | $ python classify.py --model fashion.model --labelbin mlb.pickle \
2. |   --image examples/example_06.jpg
3. | Using TensorFlow backend.
4. | [INFO] loading network...
5. | [INFO] classifying image...
6. | black: 100.00%
7. | blue: 0.00%
8. | dress: 0.01%
9. | jeans: 100.00%
10. | red: 0.00%
11. | shirt: 0.00%
```



([\[content/uploads/2018/04/keras_multi_label_output_06.png\]\(content/uploads/2018/04/keras_multi_label_output_06.png\)\)](https://pyimagesearch.com/wp-</p>
</div>
<div data-bbox=)

Figure 9: Both labels, “jeans” and “black” are correct in this Keras multi-label classification deep learning

This Keras multi-label classification deep learning experiment.

Click here to download the source code to this post

I can't be 100% sure that these are denim jeans (they look more like leggings/jeggings to me), but our multi-label classifier is!

Let's try a final example of a black dress (`example_07.jpg`). While our network has learned to predict “*black jeans*” and “*blue jeans*” along with both “*blue dress*” and “*red dress*”, can it be used to classify a “*black dress*”?

→ [Launch Jupyter Notebook on Google Colab](#)

Multi-label classification with Keras

```
1. | $ python classify.py --model fashion.model --labelbin mlb.pickle \
2. |   --image examples/example_07.jpg
3. | Using TensorFlow backend.
4. | [INFO] loading network...
5. | [INFO] classifying image...
6. | black: 91.28%
7. | blue: 7.70%
8. | dress: 5.48%
9. | jeans: 71.87%
10. | red: 0.00%
11. | shirt: 5.92%
```





[\(https://pyimagesearch.com/wp-content/uploads/2018/04/keras_multi_label_output_07.png\)](https://pyimagesearch.com/wp-content/uploads/2018/04/keras_multi_label_output_07.png)

Figure 10: What happened here? Our multi-class labels are incorrect. Color is marked as “black” but the classifier had a higher confidence that this was an image of “jeans” than a “dress”. The reason is that our neural network never saw this combination in its training data. See the “Summary” below for further explanation.

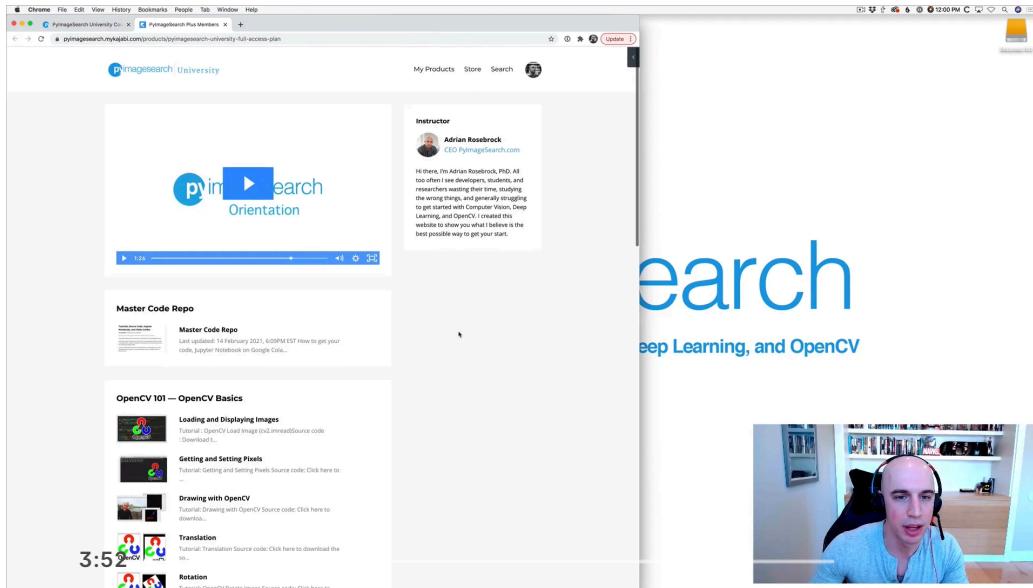
Oh no — a blunder! Our classifier is reporting that the model is wearing black jeans when she is actually wearing a black dress.

What happened here?

Why are our multi-class predictions incorrect? To find out why, review the summary below.

What's next? We recommend [PyImageSearch University](#)

[\(https://pyimagesearch.com/pyimagesearch-university/?
Click here to download the source code to this post\)](https://pyimagesearch.com/pyimagesearch-university/?utm_source=blogPost&utm_medium=bottomBanner&utm_campaign=What%27s%20next%3F%20I%20recommend).)



Course information:

84 total classes • 114+ hours of on-demand code walkthrough videos • Last updated: February 2024

★★★★★ 4.84 (128 Ratings) • 16,000+ Students Enrolled

I strongly believe that if you had the right teacher you could *master* computer vision and deep learning.

Do you think learning computer vision and deep learning has to be time-consuming, overwhelming, and complicated? Or has to involve complex mathematics and equations? Or requires a degree in computer science?

That's not the case.

[Click here to download the source code to this post](#)

All you need to master computer vision and deep learning is for someone to explain things to you in *simple, intuitive* terms. *And that's exactly what I do.* My mission is to change education and how complex Artificial Intelligence topics are taught.

If you're serious about learning computer vision, your next stop should be Pylmagesearch University, the most comprehensive computer vision, deep learning, and OpenCV course online today. Here you'll learn how to *successfully* and *confidently* apply computer vision to your work, research, and projects. Join me in computer vision mastery.

Inside Pylmagesearch University you'll find:

- ✓ **86 courses** on essential computer vision, deep learning, and OpenCV topics
- ✓ **86 Certificates** of Completion
- ✓ **115+ hours** of on-demand video
- ✓ **Brand new courses released regularly**, ensuring you can keep up with state-of-the-art techniques
- ✓ **Pre-configured Jupyter Notebooks in Google Colab**
- ✓ Run all code examples in your web browser — works on Windows, macOS, and Linux (no dev environment configuration required!)
- ✓ Access to **centralized code repos** for **all 540+ tutorials** on Pylmagesearch
- ✓ **Easy one-click downloads** for code, datasets, pre-trained models, etc.
- ✓ **Access** on mobile, laptop, desktop, etc.

**CLICK HERE TO JOIN PYIMAGESearch UNIVERSITY
([HTTPS://PYIMAGESearch.COM/PYIMAGESearchUNIVERSITY/](https://PYIMAGESearch.COM/PYIMAGESearchUNIVERSITY/))**

UTM_SOURCE=BLOGPOST&utm_medium=bottombanner&utm_campaign=WHAT%27S%20NEXT%3F%20I%20RECOMMEND)

Summary

In today's blog post you learned how to perform multi-label classification with Keras.

Performing multi-label classification with Keras is straightforward and includes two primary steps:

- 1 Replace the *softmax activation* at the end of your network with a *sigmoid activation*
- 2 Swap out *categorical cross-entropy* for *binary cross-entropy* for your loss function

From there you can train your network as you normally would.

The end result of applying the process above is a multi-class classifier.

You can use your Keras multi-class classifier to predict *multiple labels* with just a *single* forward pass.

However, there is a difficulty you need to consider:

You need training data for each *combination* of categories you would like to predict.

Just like a neural network cannot predict classes it was never trained on, your neural network cannot predict multiple class labels for combinations it has never seen. The reason for this behavior is due to activations of neurons inside the network.

If your network is trained on examples of both (1) black pants and (2) red shirts and now you want to predict “red pants” (where there are no “red pants” images in your dataset), the neurons responsible for detecting “red” and “pants” will fire, but since the network has never seen this combination of data/activations before once they reach the fully-connected layers, your output predictions will very likely be incorrect (i.e., you may encounter “red” or “pants” but very unlikely both).

Again, **your network cannot correctly make predictions on data it was never trained on** (and you shouldn’t expect it to either). Keep this caveat in mind when training your own Keras networks for multi-label classification.

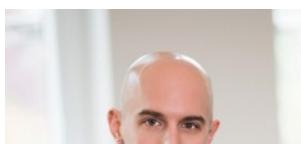
I hope you enjoyed this post!

To be notified when future posts are published here on PyImageSearch, just enter your email address in the form below!



Download the Source Code and FREE 17-page Resource Guide

Enter your email address below to get a .zip of the code and 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!



About the Author

Hi there, I'm Adrian Rosebrock, PhD. All too often I see



developers, students, and researchers wasting their time, studying the wrong things and generally struggling to get started with Computer Vision, Deep Learning, and OpenCV. I created this website to show you what I believe is the best possible way to get your start.

Click here to download the source code to this post

Similar articles

OPENCV TUTORIALS TUTORIALS

Splitting and Merging Channels with OpenCV

January 23, 2021

(<https://pyimagesearch.com/2021/01/23/splitting-and-merging-channels-with-opencv/>) →

DEEP LEARNING KERAS AND TENSORFLOW TUTORIALS

Traffic Sign Classification with Keras and Deep Learning

November 4, 2019

(<https://pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep-learning/>) →

OPTICAL CHARACTER RECOGNITION (OCR) TUTORIALS

What is Optical Character Recognition (OCR)?

August 9, 2022 [Click here to download the source code to this post](#)

(<https://pyimagesearch.com/2021/08/09/what-is-optical-character-recognition-ocr/>)



You can learn Computer Vision, Deep Learning, and OpenCV.

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

Topics

Deep Learning

(<https://pyimagesearch.com/category/deep-learning-2/>).

Dlib Library

(<https://pyimagesearch.com/category/dlib/>).

Embedded/IoT and Computer Vision

(<https://pyimagesearch.com/category/embedded/>).

Machine Learning and Computer Vision

(<https://pyimagesearch.com/category/machine-learning-2/>).

Medical Computer Vision

(<https://pyimagesearch.com/category/medical/>).

Optical Character Recognition (OCR)

(<https://pyimagesearch.com/category/optical-character-recognition-ocr/>).

[Face Applications](https://pyimagesearch.com/category/faces/)

(<https://pyimagesearch.com/category/faces/>)

Click here to download the source code to this post

[Image Processing](https://pyimagesearch.com/category/image-processing/)

(<https://pyimagesearch.com/category/image-processing/>)

[Interviews](https://pyimagesearch.com/category/interviews/)

(<https://pyimagesearch.com/category/interviews/>)

[Keras](https://pyimagesearch.com/category/keras/)

(<https://pyimagesearch.com/category/keras/>)

[OpenCV Install Guides](https://pyimagesearch.com/opencv-tutorials-resources-guides/)

(<https://pyimagesearch.com/opencv-tutorials-resources-guides/>)

[Object Detection](https://pyimagesearch.com/category/object-detection/)

(<https://pyimagesearch.com/category/object-detection/>)

[Object Tracking](https://pyimagesearch.com/category/object-tracking/)

(<https://pyimagesearch.com/category/object-tracking/>)

[OpenCV Tutorials](https://pyimagesearch.com/category/opencv/)

(<https://pyimagesearch.com/category/opencv/>)

[Raspberry Pi](https://pyimagesearch.com/category/raspberry-pi/)

(<https://pyimagesearch.com/category/raspberry-pi/>)

Books & Courses[PyImageSearch University](https://pyimagesearch.com/pyimagesearch-university/)

(<https://pyimagesearch.com/pyimagesearch-university/>)

[FREE CV, DL, and OpenCV Crash Course](https://pyimagesearch.com/free-opencv-computer-vision-deep-learning-crash-course/)

(<https://pyimagesearch.com/free-opencv-computer-vision-deep-learning-crash-course/>)

[Practical Python and OpenCV](https://pyimagesearch.com/practical-python-opencv/)

(<https://pyimagesearch.com/practical-python-opencv/>)

[Deep Learning for Computer Vision with Python](https://pyimagesearch.com/deep-learning-computer-vision-python-book/) (<https://pyimagesearch.com/deep-learning-computer-vision-python-book/>)[PyImageSearch Gurus Course](https://pyimagesearch.com/pyimagesearch-gurus/)

(<https://pyimagesearch.com/pyimagesearch-gurus/>)

[Raspberry Pi for Computer Vision](https://pyimagesearch.com/raspberry-pi-for-computer-vision/)

(<https://pyimagesearch.com/raspberry-pi-for-computer-vision/>)

PyImageSearch[Affiliates](https://pyimagesearch.com/affiliates/)

(<https://pyimagesearch.com/affiliates/>)

[Get Started](https://pyimagesearch.com/start-here/) (<https://pyimagesearch.com/start-here/>)[About](https://pyimagesearch.com/about/) (<https://pyimagesearch.com/about/>)[Consulting](https://pyimagesearch.com/consulting-2/)

(<https://pyimagesearch.com/consulting-2/>)

[Coaching](https://pyimagesearch.com/consult-adrian/) (<https://pyimagesearch.com/consult-adrian/>)[FAQ](https://pyimagesearch.com/faqs/) (<https://pyimagesearch.com/faqs/>)[YouTube](https://pyimagesearch.com/youtube/)

(<https://pyimagesearch.com/youtube/>)

[Blog](https://pyimagesearch.com/topics/) (<https://pyimagesearch.com/topics/>)[Contact](https://pyimagesearch.com/contact/) (<https://pyimagesearch.com/contact/>)[Privacy Policy](https://pyimagesearch.com/privacy-policy/)

(<https://pyimagesearch.com/privacy-policy/>)

[Click here to download the source code to this post](#)

 (<https://www.facebook.com/pyimagesearch>) 

 (<https://twitter.com/PylImageSearch>) 

 (<https://www.linkedin.com/company/pyimagesearch/>) 

[\(\[https://www.youtube.com/channel/UCoQK7OVcIVy-nV4m-SMCk_Q/videos\]\(https://www.youtube.com/channel/UCoQK7OVcIVy-nV4m-SMCk_Q/videos\)\)](https://www.youtube.com/channel/UCoQK7OVcIVy-nV4m-SMCk_Q/videos)

© 2024 PylImageSearch (<https://pyimagesearch.com>). All Rights Reserved.

 [Privacy and cookie settings](#)

Managed by Google. Complies with IAB TCF. CMP ID: 300