PYIMAGESEARCH

Click here to download the source code to this post

DEEP LEARNING (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/DEEP-LEARNING/)

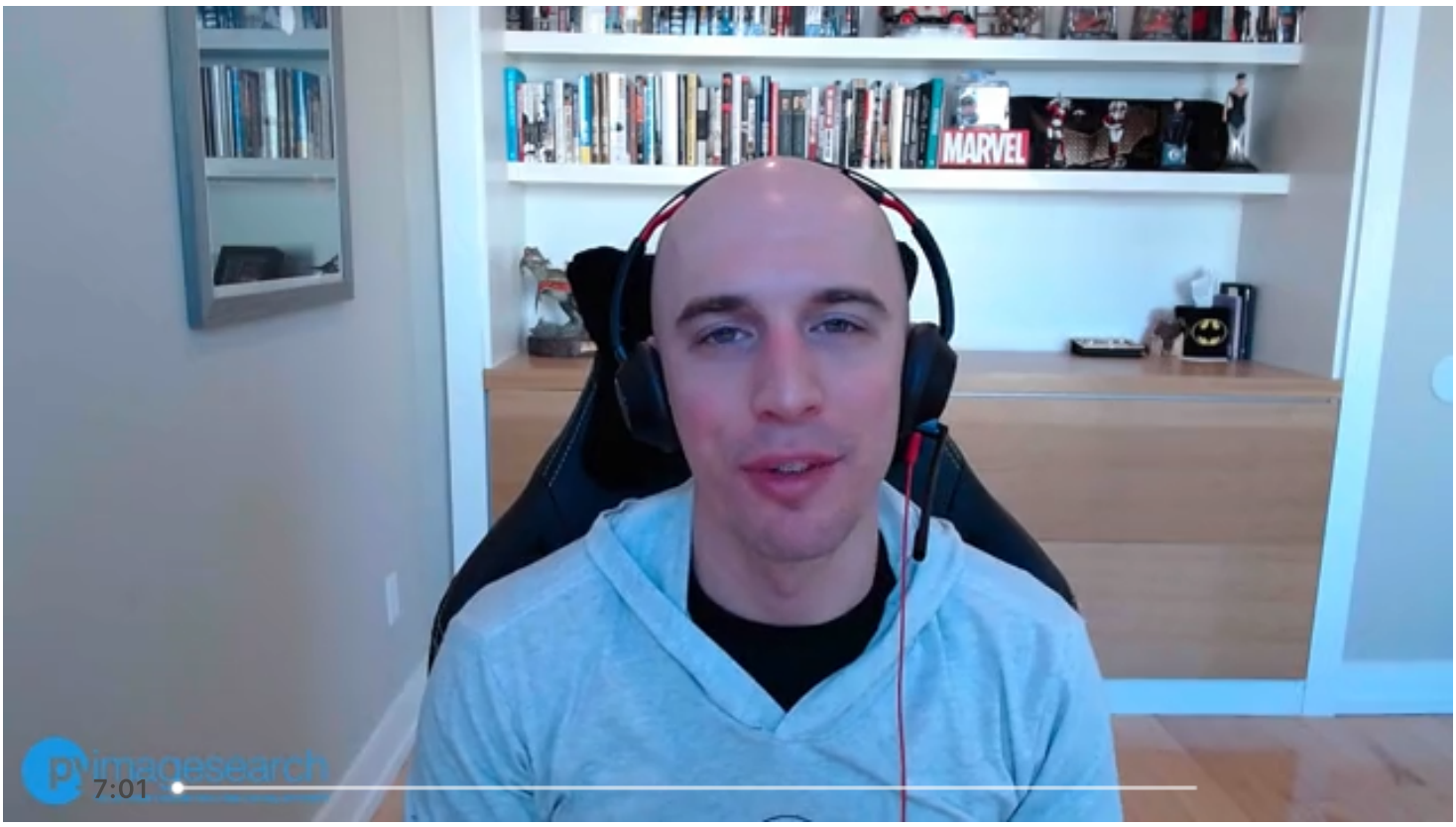
KERAS AND TENSORFLOW (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/KERAS-AND-TENSORFLOW/)

TUTORIALS (HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/)

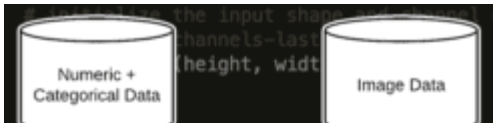
Keras: Multiple Inputs and Mixed Data

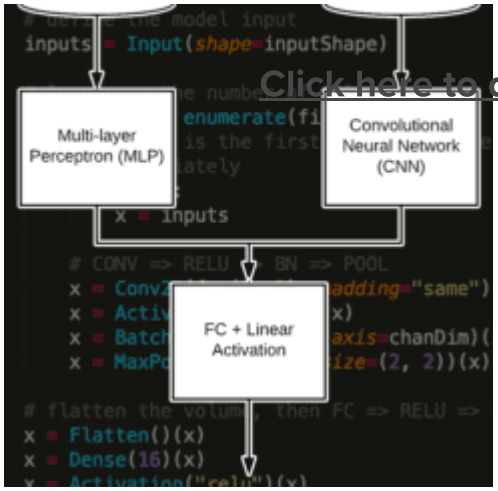
by **Adrian Rosebrock (https://www.pyimagesearch.com/author/adrian/)** *on* February 4, 2019

Last updated on July 8, 2021.



In this tutorial, you will learn how to use Keras for multi-input and mixed data.





Click here to download the source code to this post

(https://pyimagesearch.com/wp-content/uploads/2019/02/keras_multi_input_header.png)

You will learn how to define a Keras architecture capable of accepting multiple inputs, including numerical, categorical, and image data. We'll then train a single end-to-end network on this mixed data.

Today is the final installment in our three part series on Keras and regression:

1. **Basic regression with Keras (https://pyimagesearch.com/2019/01/21/regression-with-keras/)**
2. **Training a Keras CNN for regression prediction (https://pyimagesearch.com/2019/01/28/keras-regression-and-cnns/)**
3. Multiple inputs and mixed data with Keras (today's post)

In this series of posts, we've explored regression prediction in the context of *house price prediction*.

The house price dataset we are using includes not only ***numerical and categorical data,*** but ***image data as well*** — we call multiple types of data ***mixed data*** as our model needs to be capable of accepting our multiple inputs (that are not of the same type) and computing a prediction on these inputs.

In the remainder of this tutorial you will learn how to:

1. Define a Keras model capable of accepting multiple inputs, including numerical, categorical, and image data, **all at the same time**.
2. Train an end-to-end Keras model on the **mixed data inputs**.

3. Evaluate our model using the multi-inputs.

Click here to download the source code to this post

To learn more about multiple inputs and mixed data with Keras, *just keep reading!*

- **Update July 2021:** Added section on the problems associated with one-hot encoding categorical data and how learning embeddings can help resolve the problem, *especially* when working in a multi-input network scenario.

Looking for the source code to this post?

JUMP RIGHT TO THE DOWNLOADS SECTION →

Keras: Multiple Inputs and Mixed Data

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

In the first part of this tutorial, we will briefly review the concept of both **mixed data** and **how Keras can accept multiple inputs**.

From there we'll review our house prices dataset and the directory structure for this project.

Next, I'll show you how to:

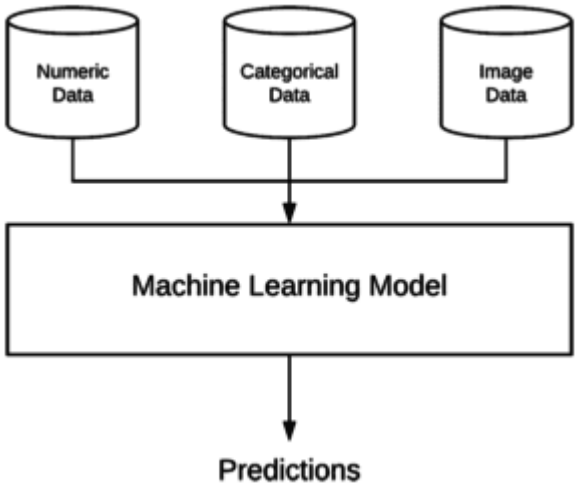
1. Load the numerical, categorical, and image data from disk.
2. Pre-process the data so we can train a network on it.
3. Prepare the mixed data so it can be applied to a multi-input Keras network.

Once our data has been prepared you'll learn **how to define and train a multi-input Keras model** that accepts multiple types of input data in a single end-to-end network.

Click here to download the source code to this post

Finally, we'll evaluate our multi-input and mixed data model on our testing set and compare the results to our previous posts in this series.

What is mixed data?



(https://pyimagesearch.com/wp-content/uploads/2019/02/keras_multi_input_mixed_data.png)

Figure 1: With the Keras' flexible deep learning framework, it is possible define a multi-input model that includes both CNN and MLP branches to handle mixed data.

In machine learning, mixed data refers to the concept of having multiple types of independent data.

For example, let's suppose we are machine learning engineers working at a hospital to develop a system capable of classifying the health of a patient.

We would have multiple types of input data for a given patient, including:

1. **Numeric/continuous values,** such as age, heart rate, blood pressure
2. **Categorical values,** including gender and ethnicity
3. **Image data,** such as any MRI, X-ray, etc.

All of these values constitute different data types; however, our machine learning model must be able to ingest this "mixed data" and make (accurate) predictions on it.

Click here to download the source code to this post

You will see the term "mixed data" in machine learning literature when working with multiple data modalities.

Developing machine learning systems capable of handling mixed data can be extremely challenging as each data type may require separate preprocessing steps, including scaling, normalization, and feature engineering.

Working with mixed data is still very much an open area of research and is often *heavily* dependent on the specific task/end goal.

We'll be working with mixed data in today's tutorial to help you get a feel for some of the challenges associated with it.

How can Keras accept multiple inputs?

(https://pyimagesearch.com/wp-content/uploads/2019/02/keras_multi_input_functional_api.jpg)

Figure 2: As opposed to its Sequential API, Keras' functional API allows for much more complex models. In this blog post we use the functional API to support our goal of creating a model with multiple inputs and mixed data for house price prediction.

Keras is able to handle multiple inputs (and even **multiple outputs (https://pyimagesearch.com/2018/06/04/keras-multiple-outputs-and-multiple-losses/)**) via its **functional API.**

Learn more about ***3 ways to create a Keras model with TensorFlow 2.0 (Sequential, Functional, and Model Subclassing)*** (https://www.pyimagesearch.com/2019/10/28/3-ways-to-create-a-keras-model-with-tensorflow-2-0-sequential-functional-and-model-subclassing/).

The functional API, as opposed to the sequential API (which you almost certainly have used before via the `Sequential` class), can be used to define much more complex models that are non-sequential, including:

- Multi-input models
- Multi-output models
- Models that are both multiple input and multiple output
- Directed acyclic graphs
- Models with shared layers

For example, we may define a simple sequential neural network as:


```
Keras: Multiple Inputs and Mixed Data
model = Sequential()
model.add(Dense(8, input_shape=(10,), activation="relu"))
model.add(Dense(4, activation="relu"))
model.add(Dense(1, activation="linear"))
```

This network is a simple feedforward neural without with 10 inputs, a first hidden layer with 8 nodes, a second hidden layer with 4 nodes, and a final output layer used for regression.

We can define the sample neural network using the functional API:


```
Keras: Multiple Inputs and Mixed Data
inputs = Input(shape=(10,))
x = Dense(8, activation="relu")(inputs)
x = Dense(4, activation="relu")(x)
x = Dense(1, activation="linear")(x)
model = Model(inputs, x)
```

Notice how we are no longer relying on the `Sequential` class.

To see the power of Keras' function API consider the following code where we create a model that accepts multiple inputs:

Click here to download the source code to this post

→ **Launch Jupyter Notebook on Google Colab**

Keras: Multiple Inputs and Mixed Data

```
1. | # define two sets of inputs
2. | inputA = Input(shape=(32,))
3. | inputB = Input(shape=(128,))
4. |
5. | # the first branch operates on the first input
6. | x = Dense(8, activation="relu")(inputA)
7. | x = Dense(4, activation="relu")(x)
8. | x = Model(inputs=inputA, outputs=x)
9. |
10. | # the second branch opreates on the second input
11. | y = Dense(64, activation="relu")(inputB)
12. | y = Dense(32, activation="relu")(y)
13. | y = Dense(4, activation="relu")(y)
14. | y = Model(inputs=inputB, outputs=y)
15. |
16. | # combine the output of the two branches
17. | combined = concatenate([x.output, y.output])
18. |
19. | # apply a FC layer and then a regression prediction on the
20. | # combined outputs
21. | z = Dense(2, activation="relu")(combined)
22. | z = Dense(1, activation="linear")(z)
23. |
24. | # our model will accept the inputs of the two branches and
25. | # then output a single value
26. | model = Model(inputs=[x.input, y.input], outputs=z)
```

Here you can see we are defining two inputs to our Keras neural network:

1. `inputA` : 32-dim
2. `inputB` : 128-dim

Lines 21-23 define a simple `32-8-4` network using Keras' functional API.

Similarly, **Lines 26-29** define a `128-64-32-4` network.

We then *combine* the outputs of both the `x` and `y` on **Line 32**. The outputs of `x` and `y` are both 4-dim so once we concatenate them we have a 8-dim vector.

We then apply two more fully-connected layers on **Lines 36 and 37**. The first layer has 2 nodes

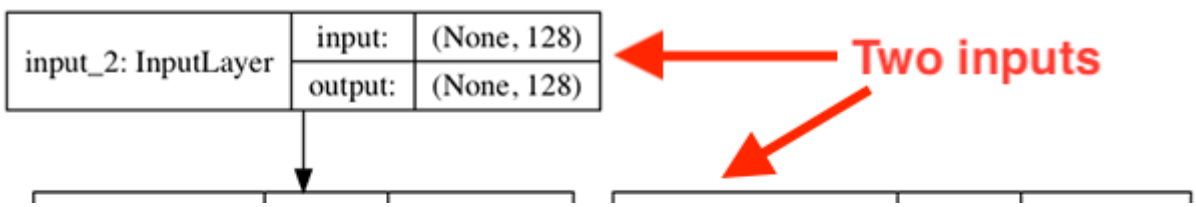
followed by a ReLU activation while the second layer has only a single node with a linear activation (i.e., our regression prediction).

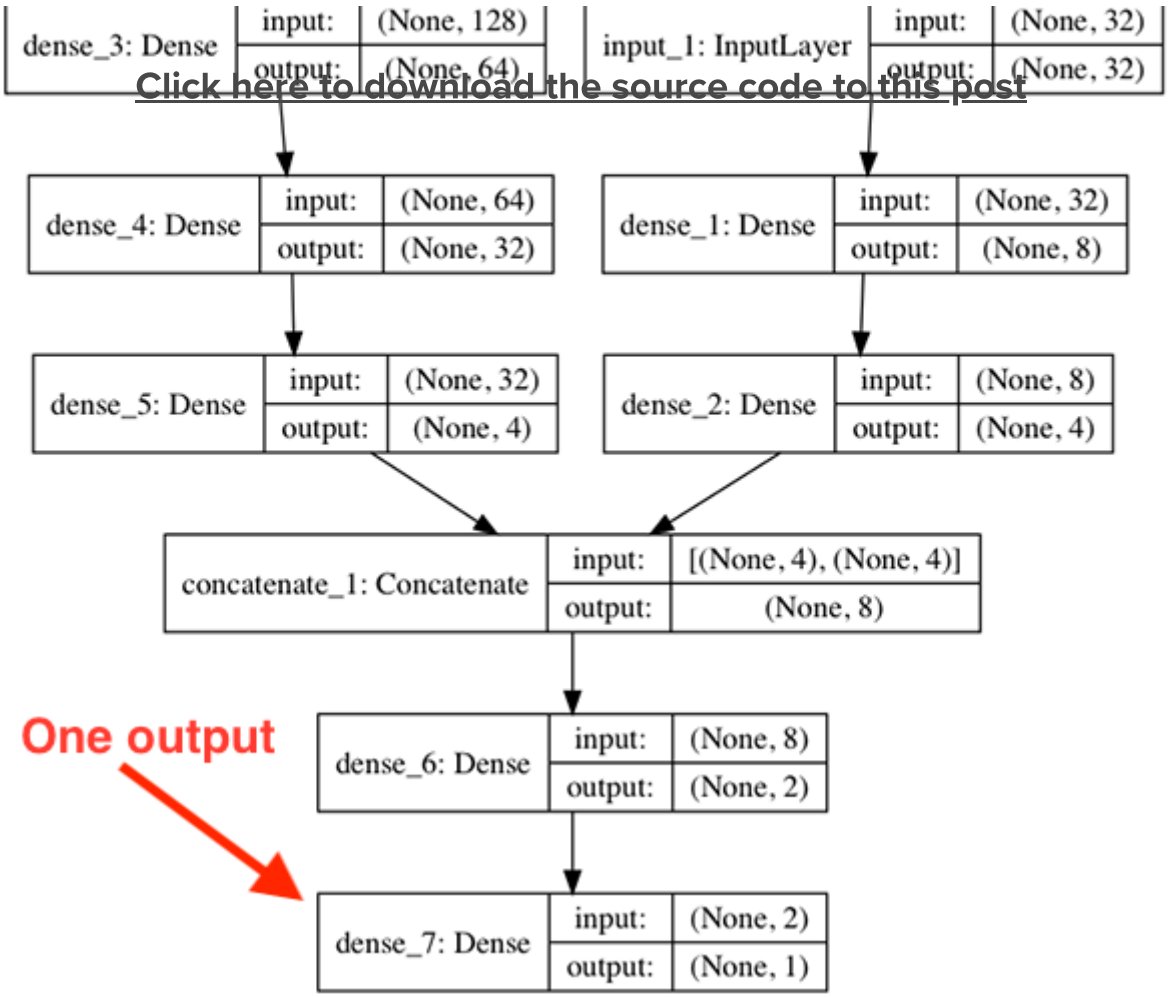
Click here to download the source code to this post

The final step to building the multi-input model is to define a `Model` object which:

1. Accepts our two `inputs`
2. Defines the `outputs` as the final set of FC layers (i.e., `z`).

If you were to use Keras to visualize the model architecture it would look like the following:





(https://pyimagesearch.com/wp-content/uploads/2019/02/keras_multi_input_arch.png)

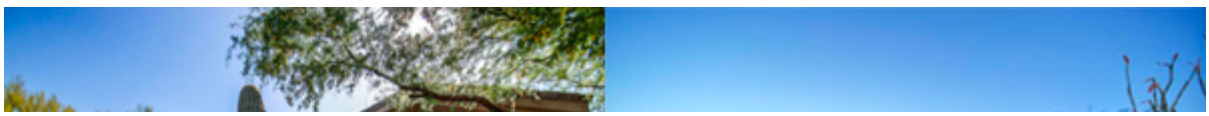
Figure 3: This model has two input branches that ultimately merge and produce one output. The Keras functional API allows for this type of architecture and others you can dream up.

Notice how our model has two distinct branches.

The first branch accepts our 128-d input while the second branch accepts the 32-d input. These branches operate independently of each other until they are concatenated. From there a single value is output from the network.

In the remainder of this tutorial, you will learn how to create multiple input networks using Keras.

The House Prices dataset



Click here to download the source code to this post

(https://pyimagesearch.com/wp-content/uploads/2019/01/keras_regression_dataset.png)

Figure 4: The House Prices dataset consists of both numerical/categorical data and image data. Using Keras, we'll build a model supporting the multiple inputs and mixed data types. The result will be a Keras regression model which predicts the price/value of houses.

In this series of posts, we have been using the House Prices dataset from Ahmed and Moustafa's 2016 paper, ***House price estimation from visual and textual features*** (https://github.com/emanhamed/Houses-dataset).

This dataset includes both **numerical/categorical data** along with **images data** for each of the 535 example houses in the dataset.

The numerical and categorical attributes include:

1. Number of bedrooms
2. Number of bathrooms
3. Area (i.e., square footage)
4. Zip code

A total of four images are provided for each house as well:

1. Bedroom
2. Bathroom
3. Kitchen
4. Frontal view of the house

Click here to download the source code to this post

In the first post in this series, you learned **how to train a Keras regression network (https://pyimagesearch.com/2019/01/21/regression-with-keras/)** on the numerical and categorical data.

Then, last week, you learned **how to perform regression with a Keras CNN (https://pyimagesearch.com/2019/01/28/keras-regression-and-cnns/)**.

Today we are going to work with multiple inputs and mixed data with Keras.

We are going to accept both the numerical/categorical data along with our image data to the network.

Two branches of a network will be defined to handle each type of data. The branches will then be combined at the end to obtain our final house price prediction.

In this manner, we will be able to leverage Keras to handle both multiple inputs and mixed data.

Obtaining the House Prices dataset

To grab the source code for today's post, use the ***"Downloads"*** section. Once you have the zip file, navigate to where you downloaded it, and extract it:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
$ cd path/to/zip
$ unzip keras-multi-input.zip
$ cd keras-multi-input
```

And from there you can download the House Prices dataset via:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
```

```
Keras: Multiple Inputs and Mixed Data
$ git clone https://github.com/emanhamed/Houses-dataset
```

Click here to download the source code to this post

The House Prices dataset should now be in the `keras-multi-input` directory which is the directory we are using for this project.

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

Either tutorial will help you configure you 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://www.pyimagesearch.com/faqs/single-faq/can-you-help-me-do-___-on-windows/)**.

Project structure

Let's take a look at how today's project is organized:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
$ tree --dirsfirst --filelimit 10
.
├── Houses-dataset
│   ├── Houses\ Dataset [2141 entries]
│   └── README.md
├── pyimagesearch
│   ├── __init__.py
│   ├── datasets.py
│   └── models.py
└── mixed_training.py

3 directories, 5 files
```

The Houses-dataset folder contains our House Prices dataset that we're working with for this series. When we're ready to run the `mixed_training.py` script we'll just need to provide a path as a

When we're ready to run the `mixed_training.py` script, you'll just need to provide a path as a **command line argument (https://pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/)** to the dataset (I'll show you exactly how this is done in the results section).

Click here to download the source code to this post

Today we'll be reviewing three Python scripts:

- `pyimagesearch/datasets.py` : Handles loading and preprocessing our numerical/categorical data as well as our image data. We previously reviewed this script over the past two weeks, but I'll be walking you through it again today.
- `pyimagesearch/models.py` : Contains our Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN). These components are the input branches to our multi-input, mixed data model. We reviewed this script last week and we'll briefly review it today as well.
- `mixed_training.py` : Our training script will use the `pyimagesearch` module convenience functions to load + split the data and concatenate the two branches to our network + add the head. It will then train and evaluate the model.

Loading the numerical and categorical data

```
>>> import pandas as pd
>>> cols = ["bedrooms", "bathrooms", "area", "zipcode", "price"]
>>> inputPath = "HousesInfo.txt"
>>> df = pd.read_csv(inputPath, sep=" ", header=None, names=cols)
>>> df.head()
   bedrooms  bathrooms  area  zipcode     price
0         4        4.0  4053    85255  869500.0
1         4        3.0  3343    36372  865200.0
2         3        4.0  3923    85266  889000.0
3         5        5.0  4022    85262  910000.0
4         3        4.0  4116    85266  971226.0
>>>
```

(https://pyimagesearch.com/wp-content/uploads/2019/01/keras_regression_pandas.png)

Figure 5: We use pandas, a Python package, to read CSV housing data.

We covered how to load the numerical and categorical data for the house prices dataset in our **Keras regression post (https://pyimagesearch.com/2019/01/28/keras-regression-and-cnns/)** but as a matter of completeness, we will review the code (in less detail) here today.

Be sure to refer to the **previous post (https://pyimagesearch.com/2019/01/28/keras-regression-and-cnns/)** if you want a detailed walkthrough of the code.

Click here to download the source code to this post

Open up the `datasets.py` file and insert the following code:

→ **Launch Jupyter Notebook on Google Colab**

Keras: Multiple Inputs and Mixed Data

```
1. # import the necessary packages
2. from sklearn.preprocessing import LabelBinarizer
3. from sklearn.preprocessing import MinMaxScaler
4. import pandas as pd
5. import numpy as np
6. import glob
7. import cv2
8. import os
9.
10. def load_house_attributes(inputPath):
11.     # initialize the list of column names in the CSV file and then
12.     # load it using Pandas
13.     cols = ["bedrooms", "bathrooms", "area", "zipcode", "price"]
14.     df = pd.read_csv(inputPath, sep=" ", header=None, names=cols)
15.
16.     # determine (1) the unique zip codes and (2) the number of data
17.     # points with each zip code
18.     zipcodes = df["zipcode"].value_counts().keys().tolist()
19.     counts = df["zipcode"].value_counts().tolist()
20.
21.     # loop over each of the unique zip codes and their corresponding
22.     # count
23.     for (zipcode, count) in zip(zipcodes, counts):
24.         # the zip code counts for our housing dataset is *extremely*
25.         # unbalanced (some only having 1 or 2 houses per zip code)
26.         # so let's sanitize our data by removing any houses with less
27.         # than 25 houses per zip code
28.         if count < 25:
29.             idxs = df[df["zipcode"] == zipcode].index
30.             df.drop(idxs, inplace=True)
31.
32.     # return the data frame
33.     return df
```

Our imports are handled on **Lines 2-8**.

From there we define the `load_house_attributes` function on **Lines 10-33**. This function reads the numerical/categorical data from the House Prices dataset in the form of a CSV file via Pandas' `pd.read_csv` on **Lines 13 and 14**.

The data is filtered to accommodate an imbalance. Some zipcodes only are represented by 1 or 2 houses, therefore we just go ahead and `drop` (**Lines 23-30**) any records where there are fewer than `25` houses from the zipcode. The result is a more accurate model later on.

Now let's define the [illegible] function

Now let's define the `process_house_attributes` function:

Click here to download the source code to this post
→ **Launch Jupyter Notebook on Google Colab**

Keras: Multiple Inputs and Mixed Data

```
35. def process_house_attributes(df, train, test):
36.     # initialize the column names of the continuous data
37.     continuous = ["bedrooms", "bathrooms", "area"]
38.
39.     # performin min-max scaling each continuous feature column to
40.     # the range [0, 1]
41.     cs = MinMaxScaler()
42.     trainContinuous = cs.fit_transform(train[continuous])
43.     testContinuous = cs.transform(test[continuous])
44.
45.     # one-hot encode the zip code categorical data (by definition of
46.     # one-hot encoding, all output features are now in the range [0, 1])
47.     zipBinarizer = LabelBinarizer().fit(df["zipcode"])
48.     trainCategorical = zipBinarizer.transform(train["zipcode"])
49.     testCategorical = zipBinarizer.transform(test["zipcode"])
50.
51.     # construct our training and testing data points by concatenating
52.     # the categorical features with the continuous features
53.     trainX = np.hstack([trainCategorical, trainContinuous])
54.     testX = np.hstack([testCategorical, testContinuous])
55.
56.     # return the concatenated training and testing data
57.     return (trainX, testX)
```

This function applies **min-max scaling** to the *continuous features* via scikit-learn's `MinMaxScaler` (**Lines 41-43**).

Then, **one-hot encoding** for the *categorical features* is computed, this time via scikit-learn's `LabelBinarizer` (**Lines 47-49**).

The *continuous* and *categorical* features are then **concatenated** and returned (**Lines 53-57**).

Be sure to refer to the previous posts in this series for more details on the two functions we reviewed in this section:

1. ***Regression with Keras*** **(https://pyimagesearch.com/2019/01/21/regression-with-keras/)**
2. ***Keras, Regression, and CNNs*** **(https://pyimagesearch.com/2019/01/28/keras-regression-and-cnns/)**

Loading the image dataset

(https://pyimagesearch.com/wp-content/uploads/2019/01/keras_regression_cnns_houses.jpg)

Figure 6: One branch of our model accepts a single image — a montage of four images from the home. Using the montage combined with the numerical/categorial data input to another branch, our model then uses regression to predict the value of the home with the Keras framework.

The next step is to define a helper function to load our input images. Again, open up the `datasets.py` file and insert the following code:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
59. def load_house_images(df, inputPath):
60.     # initialize our images array (i.e., the house images themselves)
61.     images = []
62.
63.     # loop over the indexes of the houses
64.     for i in df.index.values:
65.         # find the four images for the house and sort the file paths,
66.         # ensuring the four are always in the *same order*
67.         basePath = os.path.sep.join([inputPath, "{}_*".format(i + 1)])
68.         housePaths = sorted(list(glob.glob(basePath)))
```

The `load_house_images` function has three goals:

1. Load all photos from the House Prices dataset. Recall that we have **four photos** per house (**Figure 6**).
2. Generate a **single montage image** from the four photos. The montage will always be arranged as you see in the figure.
3. Append all of these home montages to a list/array and return to the calling function.

Beginning on **Line 59**, we define the function which accepts a Pandas dataframe and dataset `inputPath`.

From there, we proceed to:

- Initialize the `images` list (**Line 61**). We'll be populating this list with all of the montage images that we build.
- Loop over houses in our data frame (**Line 64**). Inside the loop, we:
  - Grab the paths to the four photos for the current house (**Lines 67 and 68**).

Let's keep making progress in the loop:

→ **Launch Jupyter Notebook on Google Colab**

Keras: Multiple Inputs and Mixed Data

```
70. |         # initialize our list of input images along with the output image
71. |         # after *combining* the four input images
72. |         inputImages = []
73. |         outputImage = np.zeros((64, 64, 3), dtype="uint8")
74. |
75. |         # loop over the input house paths
76. |         for housePath in housePaths:
77. |             # load the input image, resize it to be 32 32, and then
78. |             # update the list of input images
79. |             image = cv2.imread(housePath)
80. |             image = cv2.resize(image, (32, 32))
81. |             inputImages.append(image)
82. |
83. |         # tile the four input images in the output image such the first
84. |         # image goes in the top-right corner, the second image in the
85. |         # top-left corner, the third image in the bottom-right corner,
86. |         # and the final image in the bottom-left corner
87. |         outputImage[0:32, 0:32] = inputImages[0]
88. |         outputImage[0:32, 32:64] = inputImages[1]
89. |         outputImage[32:64, 32:64] = inputImages[2]
90. |         outputImage[32:64, 0:32] = inputImages[3]
```

```
91.
92.         # add the tiled image to our set of images the network will be
93.         # trained on
94.         images.append(outputImage)
95.
96.     # return our set of images
97.     return np.array(images)
```

Click here to download the source code to this post

The code so far has accomplished the first goal discussed above (grabbing the four house images per house). Let's wrap up the `load_house_images` function:

- Still inside the loop, we:
  - Perform initializations (**Lines 72 and 73**). Our `inputImages` will be in list form containing four photos of each record. Our `outputImage` will be the montage of the photos (like **Figure 6**).
  - Loop over 4 photos (**Line 76**):
    - Load, resize, and append each photo to `inputImages` (**Lines 79-81**).
  - Create the tiling (a montage) for the four house images (**Lines 87-90**) with:
    - The bathroom image in the *top-left*.
    - The bedroom image in the *top-right*.
    - The frontal view in the *bottom-right*.
    - The kitchen in the *bottom-left*.
  - Append the tiling/montage `outputImage` to `images` (**Line 94**).
- Jumping out of the loop, we `return` all the `images` in the form of a NumPy array (**Line 97**).

We'll have as many `images` as there are records we're training with (remember, we dropped a few of them in the `process_house_attributes` function).

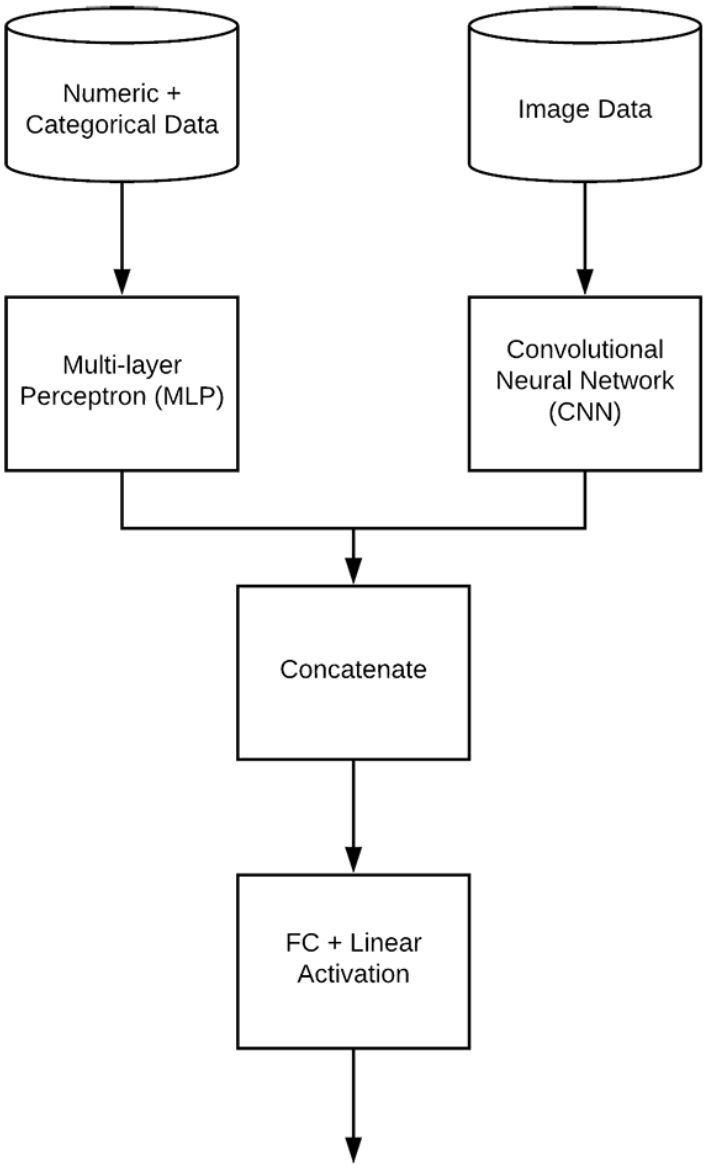
Each of our tiled `images` will look like **Figure 6** (without the overlaid text of course). You can see the four photos therein have been arranged in a montage (I've used larger image dimensions so we can better visualize what the code is doing). Just as our numerical and categorical attributes represent the house, these four photos (tiled into a single image) will represent the visual aesthetics

of the house.

Click here to download the source code to this post

If you need to review this process in further detail, be sure to refer to **last week's post (https://pyimagesearch.com/2019/01/28/keras-regression-and-cnns/)**.

Defining our Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN)



(https://pyimagesearch.com/wp-content/uploads/2019/02/keras_multi_input_design.png)

Figure 7: Our Keras multi-input + mixed data model has one

Figure 7: Our Keras multi-input + mixed data model has one branch that accepts the numerical/categorical data *(left)* and another branch that accepts image data in the form a 4-photo montage *(right)*.

Click here to download the source code to this post

As you've gathered thus far, we've had to massage our data carefully using multiple libraries: Pandas, scikit-learn, OpenCV, and NumPy.

We've organized and pre-processed the two modalities of our dataset at this point via `datasets.py`:

- Numeric and categorical data
- Image data

The skills we've used in order to accomplish this have been developed through experience + practice, machine learning best practices, and behind the scenes of this blog post, a little bit of debugging. Please don't overlook what we've discussed so far using our data massaging skills as it is key to the rest of our project's success.

Let's shift gears and discuss our multi-input and mixed data network that we'll build with Keras' functional API.

In order to build our multi-input network we will need two branches:

- The first branch will be a **simple Multi-layer Perceptron (MLP)** designed to handle the **categorical/numerical inputs**.
- The second branch will be a **Convolutional Neural Network** to operate over the **image data**.
- These **branches** will then be **concatenated** together to form the final **multi-input Keras model**.

We'll handle building the final concatenated multi-input model in the next section — our current task is to define the two branches.

Open up the `models.py` file and insert the following code:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
# import the necessary packages
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 Dropout
8. | from tensorflow.keras.layers import Dense
9. | from tensorflow.keras.layers import Flatten
10. | from tensorflow.keras.layers import Input
11. | from tensorflow.keras.models import Model
12. |
13. | def create_mlp(dim, regress=False):
14. |    # define our MLP network
15. |    model = Sequential()
16. |    model.add(Dense(8, input_dim=dim, activation="relu"))
17. |    model.add(Dense(4, activation="relu"))
18. |
19. |    # check to see if the regression node should be added
20. |    if regress:
21. |        model.add(Dense(1, activation="linear"))
22. |
23. |    # return our model
24. |    return model
```


Lines 2-11 handle our Keras imports. You'll see each of the imported functions/classes going forward in this script.

Our **categorical/numerical data** will be processed by a simple Multi-layer Perceptron (MLP).

The MLP is defined by `create_mlp` on **Lines 13-24**.

Discussed in detail in the **first post in this series (https://pyimagesearch.com/2019/01/21/regression-with-keras/)**, the MLP relies on the Keras `Sequential` API. Our MLP is quite simple having:

- A fully connected (`Dense`) input layer with ReLU `activation` (**Line 16**).
- A fully-connected hidden layer, also with ReLU `activation` (**Line 17**).
- And finally, an **optional** regression output with linear activation (**Lines 20 and 21**).

While we used the regression output of the MLP in the first post, it **will not be used** in this multi-input, mixed data network. As you'll soon see, we'll be setting `regress=False` explicitly even though it is the default as well. **Regression will actually be performed later on the head of the *entire* multi-input, mixed data network** (the bottom of **Figure 7**).

The MLP ***branch*** is returned on **Line 24**.

Referring back to **Figure 7**, we've now built the *top-left* branch of our network.

Let's now define the *top-right* branch of our network, a CNN:

Click here to download the source code to this post

→ **Launch Jupyter Notebook on Google Colab**

Keras: Multiple Inputs and Mixed Data

```
26. def create_cnn(width, height, depth, filters=(16, 32, 64), regress=False):
27.     # initialize the input shape and channel dimension, assuming
28.     # TensorFlow/channels-last ordering
29.     inputShape = (height, width, depth)
30.     chanDim = -1
31.
32.     # define the model input
33.     inputs = Input(shape=inputShape)
34.
35.     # loop over the number of filters
36.     for (i, f) in enumerate(filters):
37.         # if this is the first CONV layer then set the input
38.         # appropriately
39.         if i == 0:
40.             x = inputs
41.
42.         # CONV => RELU => BN => POOL
43.         x = Conv2D(f, (3, 3), padding="same")(x)
44.         x = Activation("relu")(x)
45.         x = BatchNormalization(axis=chanDim)(x)
46.         x = MaxPooling2D(pool_size=(2, 2))(x)
```

The `create_cnn` function handles the **image data** and accepts five parameters:

- `width` : The width of the input images in pixels.
- `height` : How many pixels tall the input images are.
- `depth` : The number of channels in our input images. For RGB color images, it is three.
- `filters` : A tuple of progressively larger filters so that our network can learn more discriminate features.
- `regress` : A boolean indicating whether or not a fully-connected linear activation layer will be appended to the CNN for regression purposes.

The `inputShape` of our network is defined on **Line 29**. It assumes "channels last" ordering for the TensorFlow backend.

The `Input` to the model is defined via the `inputShape` on (**Line 33**).

From there we begin looping over the filters and create a set of `CONV => RELU > BN => POOL` layers. Each iteration of the loop appends these layers. Be sure to check out Chapter 11 from the *Starter Bundle* of ***Deep Learning for Computer Vision with Python (https://pyimagesearch.com/deep-learning-computer-vision-python-book/)*** for more information on these layer types if you are unfamiliar.

Let's finish building the CNN branch of our network:


```
Keras: Multiple Inputs and Mixed Data
48. |     # flatten the volume, then FC => RELU => BN => DROPOUT
49. |     x = Flatten()(x)
50. |     x = Dense(16)(x)
51. |     x = Activation("relu")(x)
52. |     x = BatchNormalization(axis=chanDim)(x)
53. |     x = Dropout(0.5)(x)
54. |
55. |     # apply another FC layer, this one to match the number of nodes
56. |     # coming out of the MLP
57. |     x = Dense(4)(x)
58. |     x = Activation("relu")(x)
59. |
60. |     # check to see if the regression node should be added
61. |     if regress:
62. |         x = Dense(1, activation="linear")(x)
63. |
64. |     # construct the CNN
65. |     model = Model(inputs, x)
66. |
67. |     # return the CNN
68. |     return model
```

We `Flatten` the next layer (**Line 49**) and then add a fully-connected layer with `BatchNormalization` and `Dropout` (**Lines 50-53**).

Another fully-connected layer is applied to match the four nodes coming out of the multi-layer perceptron (**Lines 57 and 58**). Matching the number of nodes is not a requirement but it does help balance the branches.

On **Lines 61 and 62**, a check is made to see if the regression node should be appended; it is then added in accordingly. **Again, we will not be conducting regression at the end of this branch either.** Regression will be performed on the head of the multi-input, mixed data network (the very bottom of **Figure 7**).

Finally, the model is constructed from our `inputs` and all the layers we've assembled together, `x` (**Line 65**).

Click here to download the source code to this post

We can then `return` the CNN ***branch*** to the calling function (**Line 68**).

Now that we've defined ***both branches* of the multi-input Keras model**, let's learn how we can combine them!

Multiple inputs with Keras

We are now ready to build our final Keras model capable of handling both multiple inputs and mixed data. This is where the ***branches* come together** and ultimately where the "magic" happens. Training will also happen in this script.

Create a new file named `mixed_training.py`, open it up, and insert the following code:

→ **Launch Jupyter Notebook on Google Colab**

Keras: Multiple Inputs and Mixed Data

```
1. # import the necessary packages
2. from pyimagesearch import datasets
3. from pyimagesearch import models
4. from sklearn.model_selection import train_test_split
5. from tensorflow.keras.layers import Dense
6. from tensorflow.keras.models import Model
7. from tensorflow.keras.optimizers import Adam
8. from tensorflow.keras.layers import concatenate
9. import numpy as np
10. import argparse
11. import locale
12. import os
13.
14. # construct the argument parser and parse the arguments
15. ap = argparse.ArgumentParser()
16. ap.add_argument("-d", "--dataset", type=str, required=True,
17.    help="path to input dataset of house images")
18. args = vars(ap.parse_args())
```

Our imports and command line arguments are handled first.

Notable imports include:

- `datasets`: Our three convenience functions for loading/processing the CSV data and loading/pre-processing the house photos from the Houses Dataset.

- `models` : Our MLP and CNN input branches which will serve as our multi-input, mixed data.

Click here to download the source code to this post

- `train_test_split` : A scikit-learn function to construct our training/testing data splits.
- `concatenate` : A special Keras function which will accept multiple inputs.
- `argparse` : Handles parsing **command line arguments (https://pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/)**.

We have one command line argument to parse on **Lines 15-18**, `--dataset`, which is the path to where you downloaded the House Prices dataset.

Let's load our numerical/categorical data and image data:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
20. | # construct the path to the input .txt file that contains information
21. | # on each house in the dataset and then load the dataset
22. | print("[INFO] loading house attributes...")
23. | inputPath = os.path.sep.join([args["dataset"], "HousesInfo.txt"])
24. | df = datasets.load_house_attributes(inputPath)
25. |
26. | # load the house images and then scale the pixel intensities to the
27. | # range [0, 1]
28. | print("[INFO] loading house images...")
29. | images = datasets.load_house_images(df, args["dataset"])
30. | images = images / 255.0
```

Here we've loaded the House Prices dataset as a Pandas dataframe (**Lines 23 and 24**).

Then we've loaded our `images` and scaled them to the range *[0, 1]* (**Lines 29-30**).

Be sure to review the `load_house_attributes` and `load_house_images` functions above if you need a reminder on what these functions are doing under the hood.

Now that our data is loaded, we're going to construct our training/testing splits, scale the prices, and process the house attributes:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
32. | # partition the data into training and testing splits using 75% of
33. | # the data for training and the remaining 25% for testing
```

```
34. | print("[INFO] processing data...")
35. | split = train_test_split(df, images, test_size=0.25, random_state=42)
36. | (trainAttrX, testAttrX, trainImagesX, testImagesX) = split
37. |
38. | # find the largest house price in the training set and use it to
39. | # scale our house prices to the range [0, 1] (will lead to better
40. | # training and convergence)
41. | maxPrice = trainAttrX["price"].max()
42. | trainY = trainAttrX["price"] / maxPrice
43. | testY = testAttrX["price"] / maxPrice
44. |
45. | # process the house attributes data by performing min-max scaling
46. | # on continuous features, one-hot encoding on categorical features,
47. | # and then finally concatenating them together
48. | (trainAttrX, testAttrX) = datasets.process_house_attributes(df,
49. |    trainAttrX, testAttrX)
```

Click here to download the source code to this post

Our training and testing splits are constructed on **Lines 35 and 36**. We've allocated 75% of our data for training and 25% of our data for testing.

From there, we find the `maxPrice` from the training set (**Line 41**) and scale the training and testing data accordingly (**Lines 42 and 43**). Having the pricing data in the range *[0, 1]* leads to better training and convergence.

Finally, we go ahead and process our house attributes by performing min-max scaling on continuous features and one-hot encoding on categorical features. The `process_house_attributes` function handles these actions and concatenates the continuous and categorical features together, returning the results (**Lines 48 and 49**).

Ready for some magic?

Okay, I lied. There isn't actually any "magic" going on in this next code block! But we will `concatenate` the branches of our network and finish our multi-input Keras network:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
51. | # create the MLP and CNN models
52. | mlp = models.create_mlp(trainAttrX.shape[1], regress=False)
53. | cnn = models.create_cnn(64, 64, 3, regress=False)
54. |
55. | # create the input to our final set of layers as the *output* of both
56. | # the MLP and CNN
57. | combinedInput = concatenate([mlp.output, cnn.output])
58. |
59. | # our final FC layer head will have two dense layers, the final one
60. | # being our regression head
61. | x = Dense(4, activation="relu")(combinedInput)
62. | x = Dense(1, activation="linear")(x)
63. |
```

```
64. | # our final model will accept categorical/numerical data on the MLP
65. | # input and images on the CNN input, outputting a single value (the
66. | # predicted price of the house)
67. | model = Model(inputs=[mlp.input, cnn.input], outputs=x)
```

Click here to download the source code to this post

Handling multiple inputs with Keras is quite easy when you've organized your code and models.

On **Lines 52 and 53**, we create our `mlp` and `cnn` models. Notice that `regress=False` — our regression head comes later on **Line 62**.

We'll then `concatenate` the `mlp.output` and `cnn.output` as shown on **Line 57**. I'm calling this our `combinedInput` because it is the input to the rest of the network (from **Figure 3** this is `concatenate_1` where the two branches come together).

The `combinedInput` to the final layers in the network is based on the output of both the MLP and CNN branches' `8-4-1` FC layers (since each of the 2 branches outputs a 4-dim FC layer and then we concatenate them to create an 8-dim vector).

We tack on a fully connected layer with four neurons to the `combinedInput` (**Line 61**).

Then we add our `"linear"` `activation` **regression** head (**Line 62**), ***the output of which is the predicted price.***

Our `Model` is defined using the `inputs` of both branches as our multi-input and the final set of layers `x` as the `output` (**Line 67**).

Let's go ahead and compile, train, and evaluate our newly formed `model`:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
69. | # compile the model using mean absolute percentage error as our loss,
70. | # implying that we seek to minimize the absolute percentage difference
71. | # between our price *predictions* and the *actual prices*
72. | opt = Adam(lr=1e-3, decay=1e-3 / 200)
73. | model.compile(loss="mean_absolute_percentage_error", optimizer=opt)
74. |
75. | # train the model
76. | print("[INFO] training model...")
77. | model.fit(
78. |     x=[trainAttrX, trainImagesX], y=trainY,
79. |     validation_data=([testAttrX, testImagesX], testY),
80. |     epochs=200, batch_size=8)
81. |
82. | # make predictions on the testing data
83. | print("[INFO] predicting house prices...")
84. | preds = model.predict([testAttrX, testImagesX])
```

Our `model` is compiled with `"mean_absolute_percentage_error"` loss and an `Adam` optimizer with learning rate `decay` (**Lines 72 and 73**).

Click here to download the source code to this post

Training is kicked off on **Lines 77-80**. This is known as fitting the model (and is also where all the weights are tuned by the process known as backpropagation).

Calling `model.predict` on our testing data (**Line 84**) allows us to grab predictions for evaluating our model. Let's perform evaluation now:

→ **Launch Jupyter Notebook on Google Colab**

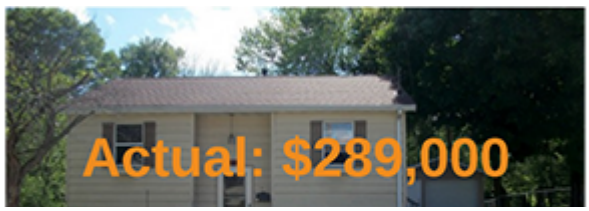
Keras: Multiple Inputs and Mixed Data

```
 86. | # compute the difference between the *predicted* house prices and the
 87. | # *actual* house prices, then compute the percentage difference and
 88. | # the absolute percentage difference
 89. | diff = preds.flatten() - testY
 90. | percentDiff = (diff / testY) * 100
 91. | absPercentDiff = np.abs(percentDiff)
 92. |
 93. | # compute the mean and standard deviation of the absolute percentage
 94. | # difference
 95. | mean = np.mean(absPercentDiff)
 96. | std = np.std(absPercentDiff)
 97. |
 98. | # finally, show some statistics on our model
 99. | locale.setlocale(locale.LC_ALL, "en_US.UTF-8")
100. | print("[INFO] avg. house price: {}, std house price: {}".format(
101. |    locale.currency(df["price"].mean(), grouping=True),
102. |    locale.currency(df["price"].std(), grouping=True)))
103. | print("[INFO] mean: {:.2f}%, std: {:.2f}%".format(mean, std))
```

To evaluate our model, we have computed absolute percentage difference (**Lines 89-91**) and used it to derive our final metrics (**Lines 95 and 96**).

These metrics (price mean, price standard deviation, and mean + standard deviation of the absolute percentage difference) are printed to the terminal with proper currency locale formatting (**Lines 100-103**).

Multi-input and mixed data results



(https://pyimagesearch.com/wp-content/uploads/2019/01/keras_regression_predicted.png)

Figure 8: Real estate price prediction is a difficult task, but our Keras multi-input + mixed input regression model yields relatively good results on our limited House Prices dataset.

Finally, we are ready to train our multi-input network on our mixed data!

Make sure you have:

1. Configured your dev environment according to the **first tutorial in this series (https://pyimagesearch.com/2019/01/21/regression-with-keras/)**.
2. Used the ***"Downloads"*** section of this tutorial to download the source code.
3. Downloaded the house prices dataset using the instructions in the *"Obtaining the House Prices dataset"* section above.

From there, open up a terminal and execute the following command to kick off training the network:

→ **Launch Jupyter Notebook on Google Colab**

```
Keras: Multiple Inputs and Mixed Data
$ python mixed_training.py --dataset Houses-dataset/Houses\ Dataset/
[INFO] loading house attributes...
[INFO] loading house images...
[INFO] processing data...
[INFO] training model...
Epoch 1/200
34/34 [==============================] - 0s 10ms/step - loss: 972.0082 - val_loss: 137.5819
Epoch 2/200
34/34 [==============================] - 0s 4ms/step - loss: 708.1639 - val_loss: 873.5765
Epoch 3/200
34/34 [==============================] - 0s 5ms/step - loss: 551.8876 - val_loss: 1078.9347
Epoch 4/200
34/34 [==============================] - 0s 3ms/step - loss: 347.1892 - val_loss: 888.7679
Epoch 5/200
34/34 [==============================] - 0s 4ms/step - loss: 258.7427 - val_loss: 986.9370
Epoch 6/200
34/34 [==============================] - 0s 3ms/step - loss: 217.5041 - val_loss: 665.0192
Epoch 7/200
34/34 [==============================] - 0s 3ms/step - loss: 175.1175 - val_loss: 435.5834
Epoch 8/200
34/34 [==============================] - 0s 5ms/step - loss: 156.7351 - val_loss: 465.2547
Epoch 9/200
34/34 [==============================] - 0s 4ms/step - loss: 133.5550 - val_loss: 718.9653
```

```
Epoch 10/200
34/34 [==============================] - 0s 3ms/step - loss: 115.4481 - val_loss: 880.0882
...
Epoch 191/200
34/34 [==============================] - 0s 4ms/step - loss: 23.4761 - val_loss: 23.4792
Epoch 192/200
34/34 [==============================] - 0s 5ms/step - loss: 21.5748 - val_loss: 22.8284
Epoch 193/200
34/34 [==============================] - 0s 3ms/step - loss: 21.7873 - val_loss: 23.2362
Epoch 194/200
34/34 [==============================] - 0s 6ms/step - loss: 22.2006 - val_loss: 24.4601
Epoch 195/200
34/34 [==============================] - 0s 3ms/step - loss: 22.1863 - val_loss: 23.8873
Epoch 196/200
34/34 [==============================] - 0s 4ms/step - loss: 23.6857 - val_loss: 1149.7415
Epoch 197/200
34/34 [==============================] - 0s 4ms/step - loss: 23.0267 - val_loss: 86.4044
Epoch 198/200
34/34 [==============================] - 0s 4ms/step - loss: 22.7724 - val_loss: 29.4979
Epoch 199/200
34/34 [==============================] - 0s 3ms/step - loss: 23.1597 - val_loss: 23.2382
Epoch 200/200
34/34 [==============================] - 0s 3ms/step - loss: 21.9746 - val_loss: 27.5241
[INFO] predicting house prices...
[INFO] avg. house price: $533,388.27, std house price: $493,403.08
[INFO] mean: 27.52%, std: 22.19%
```

Click here to download the source code to this post

Our mean absolute percentage error starts off very high but continues to fall throughout the training process.

By the end of training, we are obtaining of **27.52%** mean absolute percentage error on our testing set, implying that, on average, our network will be ~26-27% off in its house price predictions.

Let's compare this result to our previous two posts in the series:

1. Using *just* an MLP on the numerical/categorical data: **22.71%**
2. Using *just* a CNN on the image data: **56.91%**

As you can see, working with mixed data by:

1. Combining our numerical/categorical data along with image data

   And training a multi-input model on the mixed data...

2. **...has led to a model that performs well, *but not even as great* as the simpler MLP method!**

Note: *If you run the experiment enough times, you may achieve results as good as* `[INFO] mean: 19.79%, std: 17.93%` *due to the stochastic nature of weight initialization.*

Using embeddings to improve model architecture and accuracy

Before we could even train our multi-input network, we first needed to preprocess our data, including:

1. Applying min-max scaling to our continuous features
2. Applying one-hot encoding to our categorical features

However, there are two primary issues one-hot encoding our categorical values:

1. **High dimensionality:** If there are many unique categories then the output transformed one-hot encoded vector can become unmanageable.
2. **No concept of "similar" categories:** After applying one-hot encoding there is no guarantee that "similar" categories are placed closer together in the *N*–dimensional space than non-similar ones.

For example, let's say we are trying to encode categories of fruits, including:

1. Granny Smith apples
2. Honeycrisp apples
3. Bananas

Intuition tells us that in an *N*-dimensional space the Granny Smith apples and Honeycrisp apples should live closer together than the bananas; however, *one-hot encoding makes no such guarantee!*

Luckily, we can overcome this problem by learning "embeddings" using our neural network. Don't get me wrong, learning embeddings is *much* harder than simply using one-hot encodings; however, the accuracy of the model can jump significantly, and you won't have to deal with the two issues mentioned above.

Here are some resources to help you get started with embeddings and deep learning:

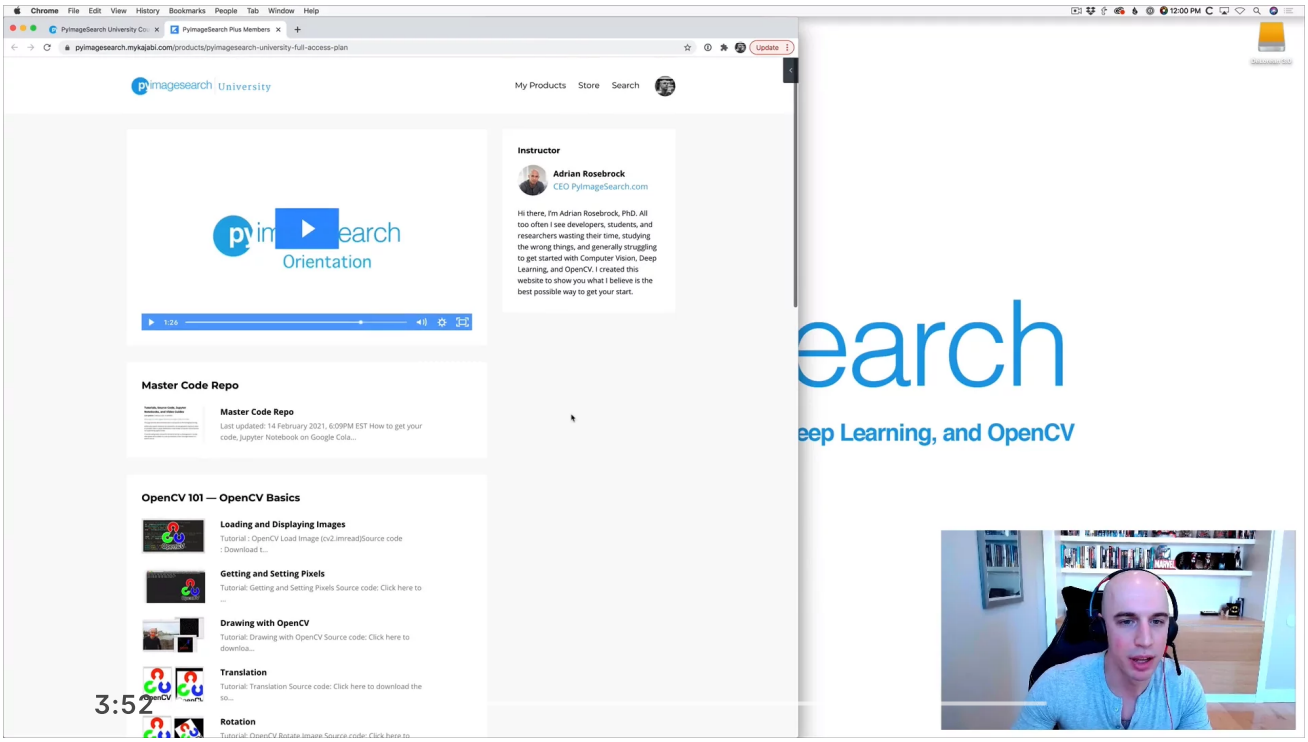
- **Google Developers guide (https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture)**

- Understanding Embedding Layer in Keras (https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce)

Click here to download the source code to this post

- Deep Learning for Tabular Data using PyTorch (https://towardsdatascience.com/deep-learning-for-tabular-data-using-pytorch-1807f2858320)

What's next? I recommend PyImageSearch University (https://www.pyimagesearch.com/pyimagesearch-university/?utm_source=blogPost&utm_medium=bottomBanner&utm_campaign=What%27s%20next%3F%20I%20recommend).



Course information:

25 total classes • 37h 19m video • Last updated: 7/2021

★★★★★ 4.84 (128 Ratings) • 10,597 Students Enrolled

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

Click here to download the source code to this post

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.

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 PyImageSearch 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 PyImageSearch University you'll find:

✓ **25 courses** on essential computer vision, deep learning, and OpenCV topics

✓ 25 Certificates of Completion

✓ **37h 19m** on-demand video

✓ **Brand new courses released *every month***, 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* 400+ tutorials** on PyImageSearch

✓ **Easy one-click downloads** for code, datasets, pre-trained models, etc.

✓ Access on mobile, laptop, desktop, etc.

Summary

In this tutorial, you learned how to define a Keras network capable of accepting multiple inputs.

You learned how to work with mixed data using Keras as well.

To accomplish these goals we defined a multiple input neural network capable of accepting:

- Numerical data
- Categorical data
- Image data

The numerical data was min-max scaled to the range *[0, 1]* prior to training. Our categorical data was one-hot encoded (also ensuring the resulting integer vectors were in the range *[0, 1]*).

The numerical and categorical data were then concatenated into a single feature vector to form the first input to the Keras network.

Our image data was also scaled to the range *[0, 1]* — this data served as the second input to the Keras network.

One branch of the model included strictly fully-connected layers (for the concatenated numerical and categorical data) while the second branch of the multi-input model was essentially a small Convolutional Neural Network.

The outputs of both branches were combined and a single output (the regression prediction) was defined.

Click here to download the source code to this post

In this manner, we were able to train our multiple input network end-to-end, resulting in accuracy almost as good as just one of the inputs alone.

I hope you enjoyed today's blog post — if you ever need to work with multiple inputs and mixed data in your own projects definitely consider using the code covered in this tutorial as a template.

From there you can modify the code to your own needs.

To download the source code, and be notified when future tutorials 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!

Click here to download the source code to this post

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.

Previous Article:

macOS Mojave: Install TensorFlow and Keras for Deep Learning

(https://www.pyimagesearch.com/2019/01/30/macos-mojave-install-tensorflow-and-keras-for-deep-learning/)

Next Article:

Fashion MNIST with Keras and Deep Learning

(https://www.pyimagesearch.com/2019/02/11/fashion-mnist-with-keras-and-deep-learning/)

87 responses to: Keras: Multiple Inputs and Mixed Data

Rich (http://www.sugeyone.com/)

February 4, 2019 at 11:39 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499512)

This is a great example of fusion, thank you Adrian!

Do you have an idea of how you would use pre-trained weights, with the features already mapped, of something trained on VGG or ImageNet in the CNN branch? Where would something like that fit in the process?

Click here to download the source code to this post

Adrian Rosebrock

February 4, 2019 at 11:51 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499515)

Do you already know how to perform fine-tuning? If not, make sure you read up on the topic. **Deep Learning for Computer Vision with Python (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)** covers fine-tuning and transfer learning in detail.

For a project such as this one you would remove perform the standard fine-tuning process for a CNN but this time have a separate branch for your categorical and numeric value inputs. From there you can train your network.

Hendrick (http://hendrickrick@icloud.com)

October 28, 2019 at 5:17 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-567758)

Hi Adrian . I am your big fans. How to create a dataset for this tutorial

---

Adrian Rosebrock

November 7, 2019 at 10:38 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-570777)

Hey Hendrick — could you be a bit more specific? What type of dataset are you trying to create?

Khine

March 7, 2019 at 11:40 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-505425)

Dear Rich,

For your case, you would need to write custom function to load pre-trained weight first. And, you can use resulted matrix in input layer.

Click here to download the source code to this post

riyaz

July 8, 2019 at 7:17 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-524882)

saving images to numpy array will require so much space what is another option to save it ?
if we want to train a network on different images of same class , how do i use image generator during training to pass train directory of two diff images?

Adrian Rosebrock

July 10, 2019 at 9:48 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-525189)

Hey Riyaz — **Deep Learning for Computer Vision with Python (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)** teaches you how to work with large image datasets, including how to write your own custom data generators. I would suggest you start there.

Arslan

February 4, 2019 at 11:53 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499516)

You are really inspiration for me to always learn new concepts in machine learning. Let me know if you have any plans for Speech Recognition/Voice related projects?
Thanks again for the great post.

Adrian Rosebrock

February 4, 2019 at 12:26 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499520)

Thanks Arslan 😐 I primarily cover computer vision here so I don't currently have any plans for

Thanks Arslan 😉 I primarily cover computer vision here so I don't currently have any plans for speech recognition tutorials but I do think it's an interesting topic. Thanks for the suggestion but to be honest it's probably unlikely that I'll cover it here.

Click here to download the source code to this post

Laxmi

February 5, 2019 at 1:28 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499643)

Thanks Adrian for the great tutorial as always! Being a newbie in deep learning I have been following your post rapidly these days with resourceful information. I am not sure if I can ask this question here; my apology in advance if not. I studied Francis´s keras book mentioning about the usages of Sequential and functional APIs; also seen in your posts and here as well. But I have noticed some examples like this **https://towardsdatascience.com/how-to-train-your-model-dramatically-faster-9ad063f0f718 (https://towardsdatascience.com/how-to-train-your-model-dramatically-faster-9ad063f0f718)** have used Sequential model also for the model architecture from Inception too. Is it possible to use Sequential model instead for functional API for Inception, ResNet or Xception too like this medium post?

Adrian Rosebrock

February 5, 2019 at 9:16 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499721)

Any Sequential API network can be implemented with a Functional API. The reverse is not true.

Laxmi

February 6, 2019 at 8:11 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500103)

Thank you for the response. So, we can not use Sequential Model for inception, resnet or other deeper CNN then, is it?

Adrian Rosebrock

February 5, 2019 at 8:32 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500262)

Click here to download the source code to this post

Any model that is non-sequential, such as Inception or ResNet, cannot be implemented using the Sequential class. You would need the Functional API instead.

Yash Rathod

February 5, 2019 at 8:20 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499707)

Hey Adrian, you got some wonderful stuff here. I follow each and every article you post and certainly I love each of them. Also its due to you I've gained so much interest in computer vision and machine learning.

Adrian Rosebrock

February 5, 2019 at 9:10 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499713)

Thanks for being a reader Yash, I'm glad you're enjoying the tutorials!

Henry

February 5, 2019 at 4:15 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499824)

For the maxPrice, would it be better if we use maxPrice = df["price"].max()

This will guarantee the scaled trainY and testY values are within the range [0, 1].

maxPrice = trainAttrX["price"].max() will work most of time but it is possible the real maxPrice is in testAttrX["price"], not in trainAttrX["price"]. The outcome may not be significant but I feel maxPrice

= df["price"].max() is more logical.

Click here to download the source code to this post

Adrian Rosebrock

February 7, 2019 at 7:21 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500289)

Technically you are right, that would absolutely guarantee the target values are in the range [0, 1].

However, that is incorrect in the context of running a deep learning experiment.

We are not allowed to use our test set to determine any information on the training process. We can only use the training set to determine any values required for normalization, scaling, preprocessing, etc.

Henrik T.

March 21, 2019 at 7:53 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-508300)

Also, for future predictions of unknown “samples”, you might encounter samples with even higher price. So it will be impossible to ensure that future samples stay within the [0, 1] range (unless you set a very high “maxPrice” – but then you no longer span training data from [0, 1] which is sub-optimal).

Juan E. Tapiero

February 5, 2019 at 11:23 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499887)

Hi Adrian, I have been happily following this project (and all the cool stuff you post). I just have a small question... why do you think I obtain different mean and standard deviation results without changing anything at all on the code? Thanks!

Adrian Rosebrock

February 7, 2019 at 7:18 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500284)

The weights in the neural network are randomly initialized. Neural networks are stochastic algorithms. You will get slightly different results each time and since we're using a very small dataset a poor initialization of weights may lead to worse results.

Denis Brion

February 8, 2019 at 9:25 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500451)

Thanks Juan for asking a question I was asking myself and Adrian for answering.
I hesitated between poor random initialisation (is there a way to make a fixed random seed with Keras: for debugging purposes, it would be more comfortable being able to reproduce a bug) or overflow/numerical accidents: as I use a RPi , I was unsure of the quality of Keras (and I noticed that it used 3 processors, with learning times 10 times slower than on Adrian PC...) and it gives very few time to try to fix / adapt things.

Stalin Amirtharaj K

February 6, 2019 at 7:01 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-499957)

Another great article , Adrian ! your article's serves as reference for all of my learning & POCs. Thanks a lot 🙂

Adrian Rosebrock

February 7, 2019 at 7:08 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500276)

Thanks so much Stalin, I'm glad you're enjoying the tutorials!

Click here to download the source code to this post

Yingjie

February 7, 2019 at 10:50 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500380)

Thank you for your explanation! This article helps me a lot! However, I'm still confused about implementing multiple inputs and multiple outputs case. In this article, different inputs have the same size, how does Keras deal with inputs with different size and multiple outputs? Especially when I'd like to iteratively sample a mini-batch from dataset A and then sample a mini-batch from dataset B. Thanks!

Xu Zhang

March 13, 2019 at 2:45 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-506513)

@Yingjie & @ Adrian Rosebrock,
In this post, there are two datasets with the same sample numbers. If some samples don't have images, what should we do? Does Keras API for the multi-input model have to keep the same numbers of samples for different input? Many thanks

---

Adrian Rosebrock

March 13, 2019 at 3:04 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-506517)

There is an entire body of literature that governs "missing values" in machine learning. You should spend some time researching it and looking into it.

Fadwa Fawzy

February 9, 2019 at 3:12 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500554)

Hello Adrian,

Click here to download the source code to this post

Thank you so much for this awesome tutorial.

I have a question though, why did you choose #bedRs and #BathRs to be continuous features not categorical.

I thought continuous feature means it can accepts real values like the area, but a house can't have 2.5 bathrooms :D.

Adrian Rosebrock

February 10, 2019 at 6:49 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500675)

Actually a house can have 2.5 bathrooms. The ".5" is normally referred to as a "half bath", such as a toilet + sink but no shower/bath.

Fadwa Fawzy

February 9, 2019 at 4:30 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500560)

Another question. What if the number of images per entry is not consistent?. For example, the number of images per house entry ranges from 0 to 10.

Adrian Rosebrock

February 10, 2019 at 6:49 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-500676)

See the other comments on this post where I address the same question.

Kent

February 11, 2019 at 1:27 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-501014)

Click here to download the source code to this post

Hi Adrian,
Sorry for late post. Thank you for the tutorial. Your efforts are appreciated. Sometimes intermediate results can be pulled as an embedding, to my understanding. If a joint embedding was the goal, could that be the output of the concatenation function ? Could this embedding be used in a similarity calculation ? Thank you for patience with my question. I am still trying to understand nuances. Kent

---

Adrian Rosebrock

February 14, 2019 at 1:29 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-501335)

You are correct, am embedding can be used here as well.

Marco

February 14, 2019 at 3:58 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-501409)

Did you get a chance to try this with individual images that are larger than 32×32? To me this feels like it cannot capture much information about the house.

Adrian Rosebrock

February 15, 2019 at 6:18 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-501530)

You can modify the architecture and try experiments with larger images if you would like. The problem is that if you increase the spatial dimensions of the images you will need to deepen your network as well. That raises a big problem as our dataset itself is so small. Realistically we would need a larger dataset to obtain higher accuracy.

Click here to download the source code to this post

Ravi

March 6, 2019 at 5:41 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-505220)

Hi Adrian,

Thanks for your tutorial. please suggest on how can i get through “from pyimagesearch import models” as i could find the respective packages. Many thanks ..

Adrian Rosebrock

March 8, 2019 at 5:43 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-505584)

You need to use the “Downloads” section of the post to download the “pyimagesearch” module.

Kyoosik Kim

March 11, 2019 at 7:54 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-506186)

Thank you so much for your work! Inspired by the tutorial, I have been doing some NLP project. I wish you could give me some advice if I can ask. There are two types of inputs; text and numerical. The texts are converted into Tfidf vectors in 1000 length and the numerical feature is simply one column added onto each vector. Now, concatenating them creates a new set of features of 1001 size. Each row should be sparse since 1000 of them are Tfidf vector values. My question is whether or not the only numerical feature may be overpowered by the Tfidf vector values. If so, how can I put more weight on the numerical feature?

Adrian Rosebrock

March 13, 2019 at 3:24 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-506540)

I don't do much work in NLP. You should reach out to Jason over at **Machine Learning Mastery. (https://machinelearningmastery.com/)** He knows a bit more about NLP than I do and would likely be able to give you a more detailed answer.

Henrik T.

March 21, 2019 at 9:26 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-508310)

Hi Adrian,
Excellent tutorial (the series)! Previously I have been searching "high and low" for good introductions to CNN-regression, without really finding any. And the introduction to the Keras functional API and example of mixed input is equally great.
I have two questions that I hope you can address:
1) With a mixed input setup, is it possible to use augmentation on the CNN branch (I know it doesn't make sense with the Houses example), or will that break the "alignment" of the two branches?
2) Again, in relation to the mixed input, I am trying to get my head straight on the following: does the merging of the MPL impact the learning in CNN branch? What I mean is: assuming IDENTICAL initialization of the weights in the "pure CNN" regression example and "mixed input CNN" regression example will the weights end up differently due to the concatenated MPL? Or does the CNN branch act as an "independent feature extractor"?

Adrian Rosebrock

March 22, 2019 at 8:38 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-508510)

Thanks Henrik, I'm glad you enjoyed the guide!

1. You can certainly apply data augmentation to the CNN branch. I would encourage it, actually. You can either augment each image individually or simply augment the entire montage. I would test both.

2. The two branches are independent until they merge. They don't share any intermediary layers until they are concatenated.

Click here to download the source code to this post

ashish

March 27, 2019 at 4:52 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-509358)

Hi, thanks for the great tutorial. I have a doubt , what will happen if we set the regress= true for individual network (i.e. cnn and mlp). Is it okay to use regress= True for both of the network. I have tried the same and getting good results for my application

Peter Yu

April 14, 2019 at 5:04 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-512787)

Hi Adrian,

Many thanks for your excellent and detailed tutorial, which exactly solves the puzzles I am now coming across.

I hope to be allowed for a further question. What if I use the ImageDataGenerator to preprocess trainImagesX, how do I get to combine the preprocessed trainImagesX and another categorical or numeric attribute trainAttrX using model.fit_generator?

Adrian Rosebrock

April 18, 2019 at 7:28 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-513472)

I would suggest reading **this tutorial (https://www.pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit_generator-a-hands-on-tutorial/)** to learn how to create your own custom data generator for Keras.

Click here to download the source code to this post

Ismail

April 18, 2019 at 3:49 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-513386)

Excellent tutorial Adrian. Please keep this up.

Adrian Rosebrock

April 18, 2019 at 6:25 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-513404)

Thanks Ismail!

Ulf Wallgren

May 4, 2019 at 5:10 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-515943)

Hi Adrian
Exelent tutorial!
Have you any plans for transforming this or any old tutoriai to Thensorflow 2.0 alpha0?

Adrian Rosebrock

May 8, 2019 at 1:28 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-516606)

I'm not sure what you mean. This code will work using TensorFlow 2.0. Either use TF 2 as a backend to Keras or use the "tf.keras" implementation directly.

sara

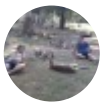
May 7, 2019 at 10:21 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-516396)

Click here to download the source code to this post

Dear Adrian:

I really appreciate your useful tutorials, Just simple question. If I have multiple EEG time-series file from different patients, could I concatenate all of them then feed the data to the NN?

Thanks…



Daniel

May 17, 2019 at 6:11 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-518106)

Hey Adrian, many thanks for this very detailed analysis, unfortunately you don't find enough of this awesome approaches. However, I have a short question about your analysis, why is the last step in which the dataframe testY is subtracted by preds carried out?

diff = preds.flatten() – testY

I did the analysis once and the values in preds[1].flatten() and without .flatten() are always constant, is that right?
Only by subtracting the values change (as in your analysis), is this common when working with multiple inputs, or why are the values constant?

Adrian Rosebrock

May 23, 2019 at 10:06 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-518942)

Both to the "preds" and "testY" are vectors of the predicted values and the ground-truth values, respectively. We take the element-wise difference using that line of code.

Nishu **Click here to download the source code to this post**

June 5, 2019 at 9:27 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-520870)

Hello Adrain,

Loved your Tutorial!!! I got some new ideas just reading through your tutorial!! Thank you very much! Please post such tutorials more!!

I have a question. Presently I have a data set that contains a label for two images simultaneously. Basically if two images are related, label is 1 and 0 if not. I don't know how to put these two images in Keras model. I have converted these two images into array. But I am stuck now!!

Description about data set:
Total rows: 50000
each row contain two images in one column and label in another column

Samuel Howell (http://ArtfulHome.com)

August 15, 2019 at 2:06 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-538602)

Hello Adrain,

Awesome tutorial! This is the only thing I could find online to show me how to use image and non image data together, very awesome!

But now that I have done the tutorial, I am confused about how to use the model to evaluate new data.

For example, if I have 4 images for a house, and all the data except for the price, how would I use my model to predict a price for it? I have my model saved and I know how to load it, I'm just not sure how to correctly plug in my data....

Adrian Rosebrock Click here to download the source code to this post

August 16, 2019 at 5:27 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-539181)

Thanks for the kind words, Samuel. I'm glad you enjoyed the post. To make predictions you use the "model.predict" method. Form your input vector in the same way as we do in the guide (concatenate the categorical values, etc.)

Mark Ryan

August 18, 2019 at 6:04 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-539924)

Great, clear article. Thanks very much for sharing it.

I am very interested in two aspects of getting multi-input Keras models into production:
1. creating a pipeline to encapsulate the data preparation steps so that you can apply the pipeline to apply the model to new data values. I've looked at wrapping the keras model so it can be used with a scikit learn pipeline, but I haven't seen any examples of this working for mult-input Keras models.

2. deploying multi-input Keras models. I've attempted to deploy a multi-input Keras model with AWS Sagemaker, but there seem to be some showstopper issues with the needed libraries that expect single input for Keras models.

Any tips on the above two aspects of putting multi-input Keras models into production? Have you seen any end-to-end examples of multi-input Keras models that incorporate a pipeline for data prep as well as deployment? THANKS!!

j

September 22, 2019 at 11:07 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-553565)

Good job, a great contributor to the advancement of knowledge

Click here to download the source code to this post

Adrian Rosebrock

September 25, 2019 at 10:41 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-554265)

Thanks so much! 🙂

quzhou

October 7, 2019 at 7:43 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-560965)

Hi Adrian,
Thank you so much for the tutorial. How do you make sure that the which house attribute (line24) and house image (line29) corresponding to the same house?

Thanks,
Yu

Adrian Rosebrock

October 10, 2019 at 10:19 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-563652)

Both functions access our Pandas' DataFrame and loop over the valid entries sequentially, thus ensuring that the entries correspond to the same house.

bit_scientist

January 14, 2020 at 4:01 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-649090)

I was about to ask a similar question and looked through the comments as you recommended.

I have both image and text data-sets of patients. Image data-set is in a folder (not in Pandas' Dataframe) and text data-set is in .xlsx file which combines numerical and categorical. How should I create respective models for them so that I can later concatenate safely? Images' name and column in text data contain ID values for me to keep the consistency between the two.

Now question, do you think it is a good way to convert both images and text data into Pandas' Dataframe separetely like you did and proceed?

Thanks for all your contribution to the field.

Click here to download the source code to this post

Adrian Rosebrock

January 16, 2020 at 10:31 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-656557)

It's hard to say without knowing more details about your project along with your current experience level with deep learning. **Feel free to send me a note (https://www.pyimagesearch.com/contact/)** and we can continue the conversation over email.

Шахбоз Кодиров

October 21, 2019 at 3:09 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-565702)

Dear Mr. Adrian Rosebrock! My name is Shahboz Qodirov. I am a post-graduate student of the South Ural State University, the High School of Electronics and Computer Science. I'm attached to the department of Informational and Measuring Equipment.
My field of research is a computer science and information processing. The topic of my thesis is "Development of artificial neural network for predicting drill pipe sticking".
Dear Mr. Adrian Rosebrock, I read your work (Keras: Multiple Inputs and Mixed Data), it's awesome work! I haven't seen such work on the Internet, you are genius! I really want to reproduce your work, but it doesn't work out for me, since I use Jupyter notebook. I downloaded codes for this work from your (post) site, and really want to repeat your work. I want to deal with your approach.
Dear Adrian Rosebrock, please, send me the codes of your work suitable for jupyter notebook, I will be very grateful to you.
Yours sincerely, Shahboz Qodirov

Yours sincerely, Shahboz Qodirov.

Click here to download the source code to this post

Arjun

October 24, 2019 at 7:44 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-566813)

Hi adrian,
I have been a fan of your work for the last few months.
I am currently working on a project which is music generation from lyrics. It is an interesting one but seems a bit too much to handle for a beginner like me. This basically deals with matching the syllables of lyrics to the musical notes and making a rnn model. I am doing this project based on a paperwork presented on this same project. Bit it deals with two sequences as input. I am attaching a link of the paper presented and I would like some insight from you on this project.
https://arxiv.org/pdf/1809.04318.pdf (https://arxiv.org/pdf/1809.04318.pdf)
Please do help me with your valuable insight.
Thank you

Adrian Rosebrock

October 25, 2019 at 10:16 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-567103)

That sounds like a very neat project Arjun; however, I ask that you be respectful of my time and **read this entry in my FAQ. (https://www.pyimagesearch.com/faqs/single-faq/can-you-explain-this-paper-or-code-not-from-pyimagesearch-to-me)** While it's my pleasure to help you on your journey to understanding computer vision and deep learning, I do not dissect papers/code on a 1:1 basis. It's just too time consuming on my part — I hope you understand.

Abeer

November 7, 2019 at 5:16 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-570941)

I love this tutorial. it is exactly what is need. Really thank you for ur hard work!!

Click here to download the source code to this post

Adrian Rosebrock

November 14, 2019 at 9:37 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-573259)

Thanks Abeer! 🙂

Edward

November 15, 2019 at 9:51 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-574134)

Hi Adrian,

Have you tried multiple inputs (numerical + categorical and image) with data augmentation for the image?

Furong

December 3, 2019 at 3:22 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-588399)

Hi, I am interested to know what kind of data augmentation you mean? Thanks

cham

November 18, 2019 at 12:09 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-574819)

Dear Mr.Adrian Rosebrock.
I love this tutorial and other two week's tutorials.
I read three part series on Keras and regression.
I really appreciate your useful and detail explanation!

However,I have one question.
How can I create a regression model which predicts n-dimensional vector, not just scalar like house price.

Click here to download the source code to this post

For example, I want a model to predict 3-dimentional vector (x,y,z) as below.

input: image & some atrributes.
output: (x,y,z)

How can I create such a model based on “keras-multi-input” ?

P.S.

Of course, I will use (x,y,z) training data instead of house price.

Adrian Rosebrock

November 21, 2019 at 9:15 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-576831)

(x, y, z) data is just a standard integer/floating point to a neural network. You could just pass the vector in to the network as a flattened list.

Nicolas Morales

November 21, 2019 at 12:33 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-577303)

Thank you for this tutorial! It has been very informative. Wouldn’t you want to use the third dimension in your montage, on lines 87 to 90?

Defneh

December 2, 2019 at 11:36 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-

mixed-data/#comment-587946)

Click here to download the source code to this post

Hi,
Adraian you are amazing !!! Thank you for this tutorial, I am new in machine learning. I dont know if my question make sense but is it possible to use this method for image classification.For classifying the health of a patient (have the disease or not) as you mentioned in the tutorial . what are the general changes that I have to make to this code ?
Thank you again

Adrian Rosebrock

December 5, 2019 at 10:32 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-589873)

Yes, you can use this method for any neural network that accepts multi-modal data. Exactly what changes are required depends on your dataset.

Ernie

December 9, 2019 at 9:49 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-590601)

Looking over this, it appears to be a great tutorial.
But, Python does not manage memory very well and is not good at using GPU's either.
Have you ported this to C or C++ ?
Or has anyone else done this? I would love a copy of that.
Thanks for the great tutorial !

Bay Huang

April 6, 2020 at 6:33 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-770804)

Yes, that's the big problem I found. I found TACC v100 GPU doesn't support this kind of multiple inputs model that I need eagerly

aravali **Click here to download the source code to this post**

December 20, 2019 at 6:56 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-604767)

Is it possible to Fuse image and Time series ECG signal in LSTM model.
I want to use time series ECG signal along with images of patients.
I am unable to use them together in LSTM.
Please help me if it is possible.

Cristian Arteaga (http://arteagac.github.io)

January 12, 2020 at 2:56 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-644775)

Dear Adrian. Thank you so much for this great post. It's awesome to find such high quality and complete explanations with code like the ones you provide. I hope you can help me with a quick question about the difference in training regimes for each data type and how it affects when I concatenate both NN models. I am facing two main challenges.
First, in my problem, the MLP for numerical data converges after 150 epochs and the CNN for image data converges after 13 epochs. Any advice on how to deal with such difference (of required epochs) when I concatenate both the MLP and CNN? Can the Adam decay help to this?
Second, the loss values have different ranges or scale. For the numerical MLP the range is (0,8) and for the image CNN the range is (0,2). Any advice on how to deal with this difference in ranges of loss values when I concatenate both NN?

I think my two challenges are related. I suspect the different training regimes for each NN is giving me bad results when I concatenate them (bad metrics). I would appreciate any advice you can give me. Thanks in advance!

kesavan

February 3, 2020 at 2:26 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-695677)

Hello Adrian,
I have multiple view of a single object in a single image (for example, bottle – top side, front view of a car in a single image) and I have many images for a single data type(example, car -100 images , bottle – 100 images.. P.S. each images have minimum 3 views).. what kind of deep learning approach should I use? (I only have multiple image data and no numerical or categorical data).
Thanks in advance....

Click here to download the source code to this post

Adrian Rosebrock

February 5, 2020 at 2:05 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-703801)

What is the end goal of the project? To recognize the exact object in the image?

Liam

February 3, 2020 at 10:52 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-696540)

Excellent tutorial! I have a question (newbie in DL)

I want to do something similar: a CNN trained with an image + numerical data + class type, and the output should be the numerical data + class type. The numerical data is a k-vector (with probabilities), and the class type just binary classification. According my understandood, the output should be different for my k-vector and for my binary classification (categorical cross entropy and binary cross entropy). Can I create the same architecture with different outputs? I saw the example, and the “branches” are at the beginning not at the end. Other info: my probability vector sum 1 and should be sum 1 at the end as well.

Thanks again for your amazed tutorials!

Ketut Adi

February 21, 2020 at 9:01 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-749957)

Thanks for your Amazed tutorial Adrian, very help full to learn

Click here to download the source code to this post

Adrian Rosebrock

February 27, 2020 at 9:25 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-759325)

You are welcome, Ketut! I'm glad you found it helpful.

Pedro Junqueira (http://petodata.space/)

March 12, 2020 at 2:02 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-766102)

Hi Adrian, Thanks for the post. I am trying to tackle a problem which is similar to this.
It will have some numeric and categorical data as well as images.
It is to predict mineral occurrences on a particular geo location. Imagine that I can have multiple images of a particular area of interest as layers (map layers).
This can be satellite images (raster) or geophysics survey that on top of each other will characterize as a signature feature for a mineral occurrence.
I am thinking on how to create a dataset feature to represent this.
I thought of instead of tiling it as on the example above. Is it not better to stack it in multiple layers? And instead of using a RGB for each image I combine in a gray scale ? so 1 channel for each image stacked? Is it too bizare to gave a image like 500x500x20? Like 20 channels?

The goal is to detect (predict) in the image this signature then set targets and if I know where in the image this is I know the geo location because those TIF files are geo referenced.

Regards

Pedro

Adrian Rosebrock

March 19, 2020 at 10:04 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-766941)

That sounds like an interesting project. Could you send me an email and share some example images of what you're working with? That would help me recommend a way for you to get started.

Click here to download the source code to this post

Athulya

March 23, 2020 at 11:53 am (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-767386)

Hi Adrian! Thankyou for the amazing tutorial.
I am working on a skin lesion classification project where i am providing skin lesions to a CNN and categorical data(age, gender, localization of the mole) to a MLP. My dataset has 10k images. since the example in the tutorial is for a regression problem, I would like to know how i can combine the two models for a classification problem.

Adrian Rosebrock

March 25, 2020 at 1:36 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-767650)

You can combine the method covered in this tutorial **with the one covered here. (https://www.pyimagesearch.com/2018/06/04/keras-multiple-outputs-and-multiple-losses/)**

Alex

March 23, 2020 at 8:49 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-767421)

You're just a f***in genius. Thank you!

Adrian Rosebrock

March 25, 2020 at 1:35 pm (https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/#comment-767648)

You're welcome. **Click here to download the source code to this post**

Comment section

Hey, Adrian Rosebrock here, author and creator of PyImageSearch. While I love hearing from readers, a couple years ago I made the tough decision to no longer offer 1:1 help over blog post comments.

At the time I was receiving 200+ emails per day and another 100+ blog post comments. I simply did not have the time to moderate and respond to them all, and the sheer volume of requests was taking a toll on me.

Instead, my goal is to *do the most good* for the computer vision, deep learning, and OpenCV community at large by focusing my time on authoring high-quality blog posts, tutorials, and books/courses.

If you need help learning computer vision and deep learning, I suggest you refer to my full catalog of books and courses (https://www.pyimagesearch.com/books-and-courses/) — they have helped tens of thousands of developers, students, and researchers *just like yourself* learn Computer Vision, Deep Learning, and OpenCV.

Click here to browse my full catalog. (https://www.pyimagesearch.com/books-and-courses/)

Click here to download the source code to this post

Similar articles

IMAGE PROCESSING TUTORIALS

Thresholding: Simple Image Segmentation using OpenCV

September 8, 2014

(https://www.pyimagesearch.com/2014/09/08/thresholding-simple-image-segmentation-using-opencv/)

DEEP LEARNING FACE APPLICATIONS KERAS AND TENSORFLOW MEDICAL COMPUTER VISION OBJECT DETECTION TUTORIALS

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning

May 4, 2020
(https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/)

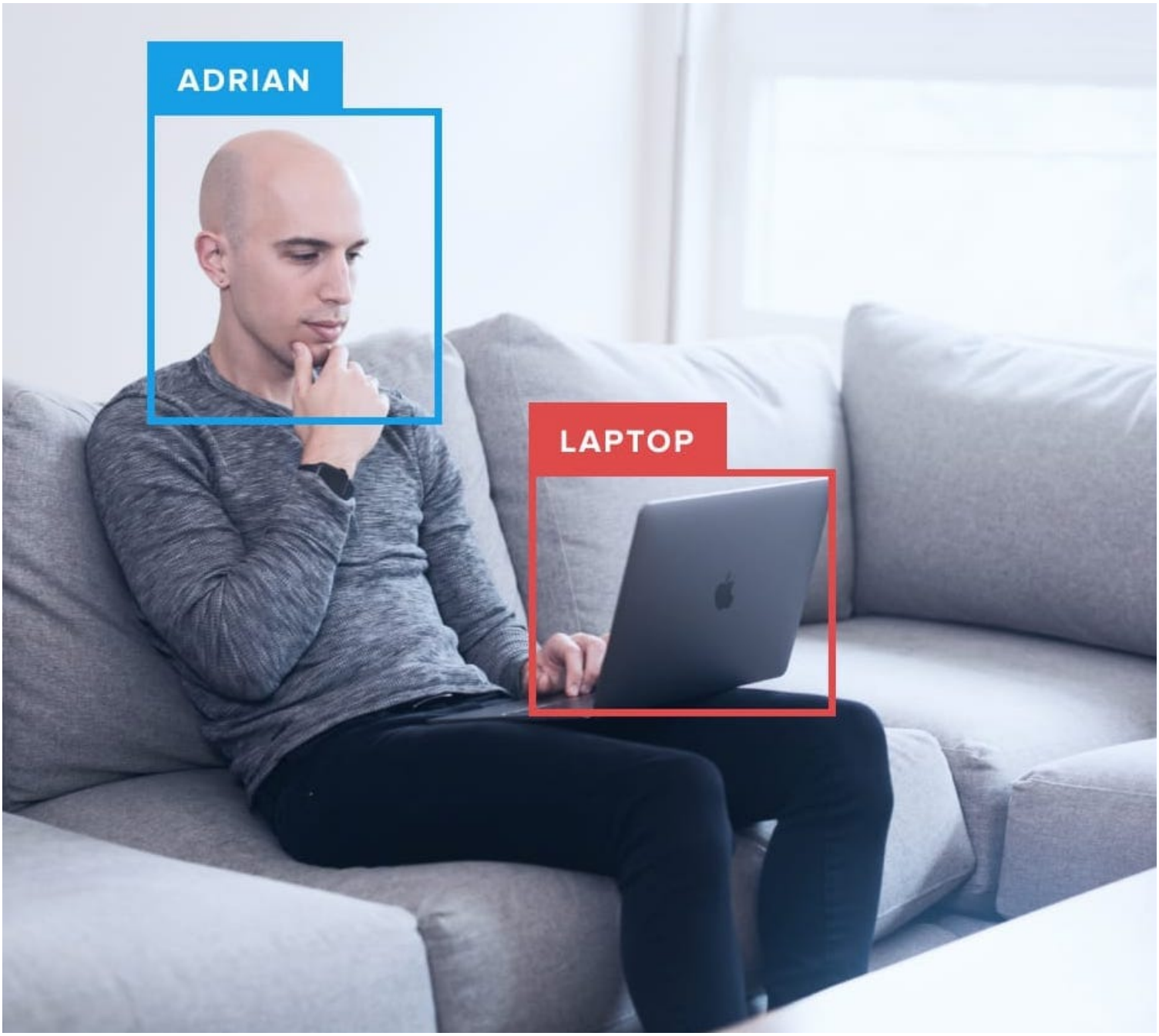
DEEP LEARNING OPTICAL CHARACTER RECOGNITION (OCR) TUTORIALS

Getting started with EasyOCR for Optical Character Recognition

September 14, 2020
(https://www.pyimagesearch.com/2020/09/14/getting-started-with-easyocr-for-optical-character-recognition/)

character recognition/)

Click here to download the source code to this post



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

opencv/)

Deep Learning for Computer Vision with Python (https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/)

PyImageSearch Gurus Course (https://www.pyimagesearch.com/pyimagesearch-gurus/)

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

Click here to download the source code to this post

About (https://www.pyimagesearch.com/about/)

FAQ (https://www.pyimagesearch.com/faqs/)

Blog (https://www.pyimagesearch.com/topics/)

Contact (https://www.pyimagesearch.com/contact/)

Privacy Policy (https://www.pyimagesearch.com/privacy-policy/)

(https://www.facebook.com/pyimagesearch) (https://twitter.com/PyImageSearch) (http://www.linkedin.com/pub/adrian-rosebrock/2a/873/59b) (https://www.youtube.com/channel/UCoQK7OVcIVy-nV4m-SMCk_Q/videos)