

Flask Frame Work with Machine Learning Model

In this section we will be building a web application which is integrated to the model we built. An UI is provided for the uses where he has to enter the values for predictions. the enter values are given to the saved model and prediction is showcased on the UI.

We are using a machine learning Model which is built for predicting the compressed strength of the concrete and saved this file as “**strength.pkl.**” We have 8 independent variables and one dependent variable for this model

To build this you should know basics of “**HTML, CSS, Bootstrap, flask framework and python**”

Create a project folder which should contains

- An python fill called app.y
- Your machine learning alogorithm file (for example strength.py or strength.ipnby)
- Model file (for example strength.pkl or strength.h5)
- Templates folder which contains index.HTML file
- Static folder which contains css folder which contains styles.css

Name	Size	Type	Date Modified
static		File Folder	11/7/2019 12:21 PM
css		File Folder	11/7/2019 12:21 PM
style.css	5 KB	css File	8/29/2019 5:37 AM
templates		File Folder	11/7/2019 12:21 PM
index.html	1 KB	html File	8/29/2019 7:02 AM
app.py	984 bytes	py File	8/29/2019 7:29 AM
strength.pkl	642 bytes	pkl File	8/29/2019 7:16 AM
strength.py	1 KB	py File	8/29/2019 7:16 AM

Steps 1: building an Index. Html file

This is the basic HTML page for our Project. H1 tag is used to give heading to the project. As I have mentioned there are 8 input or 8 independent variables we have created 8 text input fields in the html page.

A button is used to send these values to the model files this functionality will be written in the python file app.py. the model predicts the value and is displayed on the `{{ prediction_text }}` filed

```
</head>

<body>
  <div class="login">
    <h1>Compressive Strength of Concrete</h1>

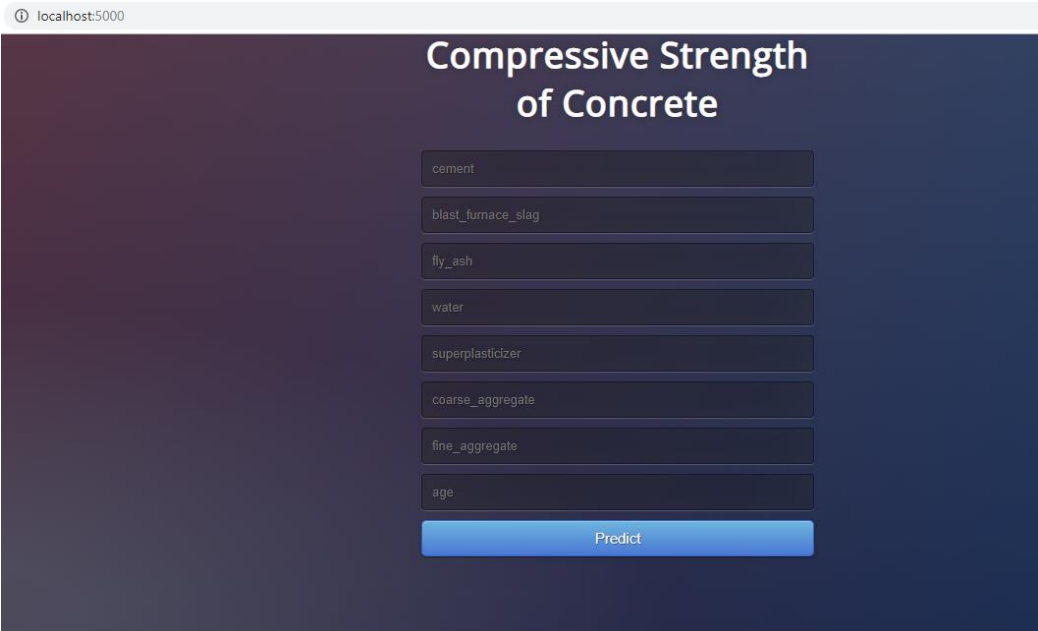
    <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('y_predict')}}"method="post">
      <input type="text" name="cement" placeholder="cement" required="required" />
      <input type="text" name="blast_furnace_slag" placeholder="blast_furnace_slag" required="required" />
      <input type="text" name="fly_ash" placeholder="fly_ash" required="required" />
      <input type="text" name="water" placeholder="water" required="required" />
      <input type="text" name="superplasticizer" placeholder="superplasticizer" required="required" />
      <input type="text" name="coarse_aggregate" placeholder="coarse_aggregate" required="required" />
      <input type="text" name="fine_aggregate" placeholder="fine_aggregate" required="required" />
      <input type="text" name="age" placeholder="age" required="required" />
      <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>

    </form>

    <br>
    <br>
    {{ prediction_text }}

  </div>
</body>
```

The built html page looks like



Step 2: Build python code

We will be using python for server side scripting. Let's see step by step process for writing backend code.

❖ Importing Libraries

- ❖ Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of current module (`__name__`) as argument

Pickle library to load the model file

```
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

app = Flask(__name__)
model = pickle.load(open('strength.pkl', 'rb'))
```

❖ Routing to the html Page

Here we will be using declared constructor to route to the html page which we have created earlier.

```
@app.route('/')
def home():
    return render_template('index.html')
```

In the above example, '/' URL is bound with index.html function. Hence, when the home page of web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method

❖ Showcasing prediction on UI

```
@app.route('/y_predict', methods=['POST'])
def y_predict():
    """
    For rendering results on HTML GUI
    """
    x_test = [[int(x) for x in request.form.values()]]

    prediction = model.predict(x_test)
    print(prediction)
    output=prediction[0][0]
    return render_template('index.html',
        prediction_text=
        'Compressive Strength of Concrete kg/m^3 {}'.format(output))
```

Here we are routing our app to Y_predict() function. This function retrieves all the values from the html page using Post request. A variable X test is defined to store all the element in two dimensional arrays. This two dimensional array is passed to model.predictfunction(). This functions returns the prediction. And this prediction value is rendered to the text that we have mentioned in the nindex.html page earlier.

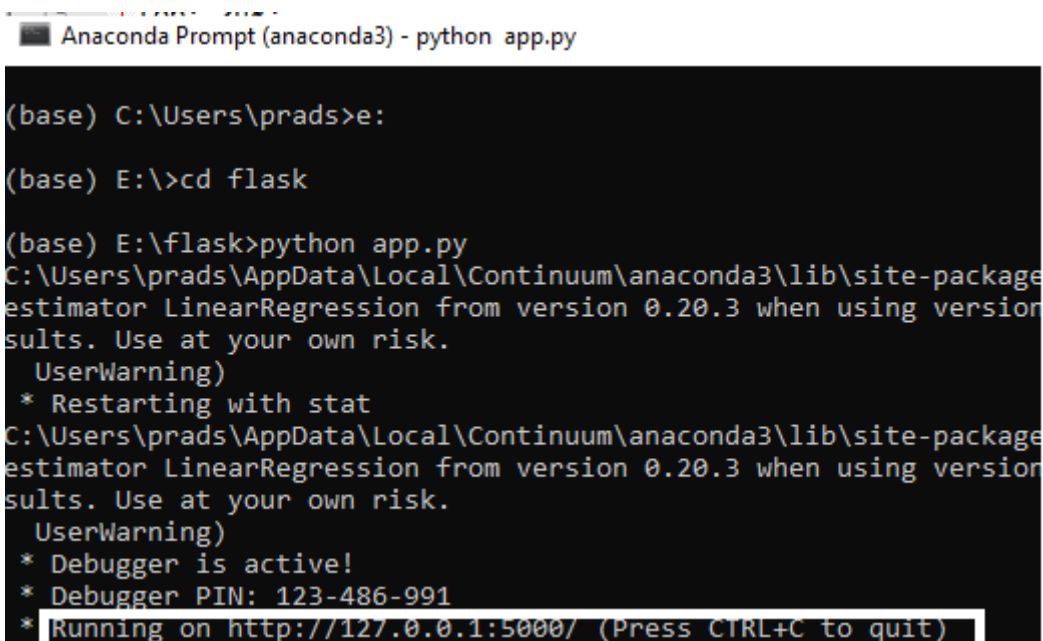
❖ Main Function

This is used to run the application in local host

```
if __name__ == "__main__":  
    app.run(debug=True)
```

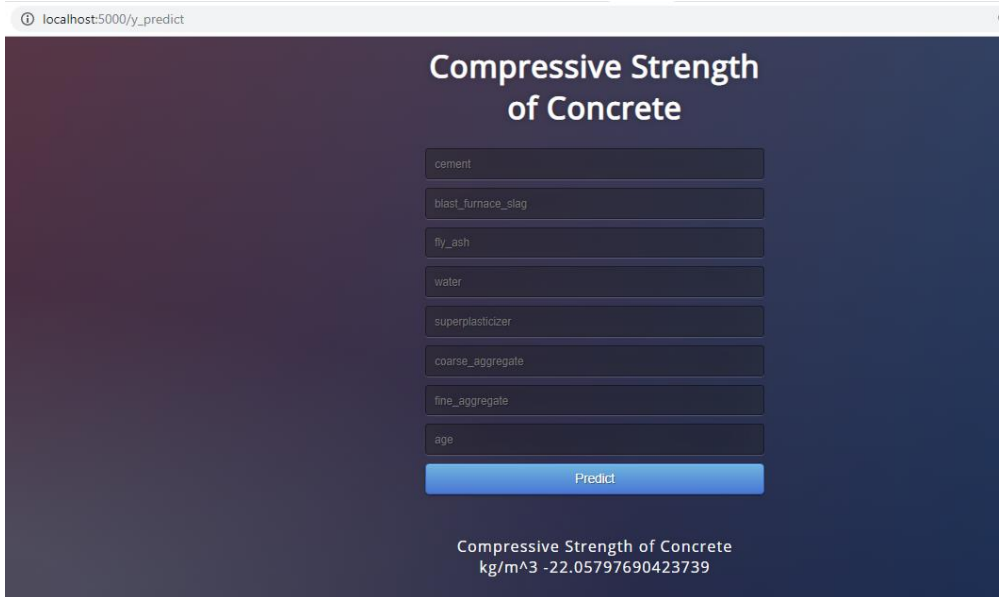
Step 3: How to Run The app

- Open anaconda prompt from start menu
- Navigate to the folder where your app.py resides
- Now type “python app.py” command
- It will show the local host where your app is running.
- Navigate to the localhost where you can view your web page
- Enter the values and see the prediction on web page



```
Anaconda Prompt (anaconda3) - python app.py  
  
(base) C:\Users\prads>e:  
  
(base) E:\>cd flask  
  
(base) E:\flask>python app.py  
C:\Users\prads\AppData\Local\Continuum\anaconda3\lib\site-packages  
estimator LinearRegression from version 0.20.3 when using version  
sults. Use at your own risk.  
  UserWarning)  
  * Restarting with stat  
C:\Users\prads\AppData\Local\Continuum\anaconda3\lib\site-packages  
estimator LinearRegression from version 0.20.3 when using version  
sults. Use at your own risk.  
  UserWarning)  
  * Debugger is active!  
  * Debugger PIN: 123-486-991  
  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Your output will look like:



localhost:5000/y_predict

Compressive Strength of Concrete

cement

blast_furnace_slag

fly_ash

water

superplasticizer

coarse_aggregate

fine_aggregate

age

Predict

Compressive Strength of Concrete
kg/m³ -22.05797690423739

Code files at:

<https://thesmartbridge.com/documents/spsaimldocs/flask.rar>

Step 1: Gathering the data

The process of gathering data depends on the type of project we desire to make. If we desire to make a project on image classification, our data needs to be in a particular format (for example you are working on a project for animal classification then you have to collect the images of all the categories /classes of animals like dog, horse cat etc.

Follow the below steps to create your data set for image classification of Animals

- Create a folder and name it as dataset anywhere in your desktop

- Go to the dataset folder and create another two subfolders and name them as trainset and testset.
- Go to the trainset folder and create folders with respect to the categories you wish to classify. (for example if you would like to classify cats, dogs and horses then create three sub folders in the train set and name them as cat, dog, horse).
- Go to testset folder and create three sub folders and name them as cat dog and horse.
- Now collect the images of cats make sure the count of images you collect should be not less than 100. paste 70% of the cat images collected in the cat subfolder which lies in dataset/trainset/cat. Paste remaining 30% in the cat subfolder lies in dataset/test set/cat.
- Repeat the above step for dog and horse as well

For reference look at this link:

https://drive.google.com/drive/folders/1726m6n_rOAFvMRt_M-oKUC1GK3itnN2T?usp=sharing

Note: In the reference data set link i have taken 7 images of each class in trainset and 3 images for each class in testset folders a total of 10 images for each class. But you please consider a total of 100 images per each class

Step2: Defining Model architecture: For image classification we use Convolution neural network

This is a very crucial step in our deep learning model building process. We have to define how our model will look and that requires

- Importing the libraries
- Initializing the model
- Adding CNN (Convolution Neural Network) Layers
- Adding Dense layers

❖ Importing the Libraries:

```
#import keras Libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten

C:\Users\HP\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

❖ Initializing the model:

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define a linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.

```
: model=Sequential()
```

❖ Adding a CNN Layers:

We will be adding three layers for CNN

- Convolution layer
- Pooling layer
- Flattening layer

Please refer to the below link for the description of these three layers

<https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add>


```
In [3]: model.add(Conv2D(32,3,3,input_shape=(64,64,3),activation='relu'))#1st parameter =no of features detectors 2nd& 3rd =Size of featur
#4th input image size,5 th parameter is channel for color=3 gray scale=1,6 th to avoid negative pixels we use activation function
```

WARNING:tensorflow:From C:\Users\HP\Anaconda3\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
 Instructions for updating:
 Colocations handled automatically by placer.

C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (3, 3), input_shape=(64, 64, 3...), activation="relu")`
 """Entry point for launching an IPython kernel.

Step4:Add Pooling Layer

```
In [4]: model.add(MaxPooling2D(pool_size=(2,2)))#1parameter=size of pooling matrix
```

Step5: Add flatten layer ¶

```
In [5]: model.add(Flatten())
```

❖ Adding Dense Layers:

Please refer to the link for the description:

<https://skymind.ai/wiki/neural-network>

Step6: Ann starts so need to add dense layers

```
In [6]: model.add(Dense(output_dim=128,activation='relu',init='random_uniform'))
```

C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(activation="relu", units=128, kernel_initializer="random_uniform")`
 """Entry point for launching an IPython kernel.

```
In [7]: model.add(Dense(output_dim=1,activation='sigmoid',init='random_uniform'))
```

C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid", units=1, kernel_initializer="random_uniform")`
 """Entry point for launching an IPython kernel.

Once you are done with initializing all the layers you have to pre-process the images.

Step 3: Loading and pre-processing the data:

Data is gold as far as deep learning models are concerned. Your image classification model has a far better chance of performing well if you have a

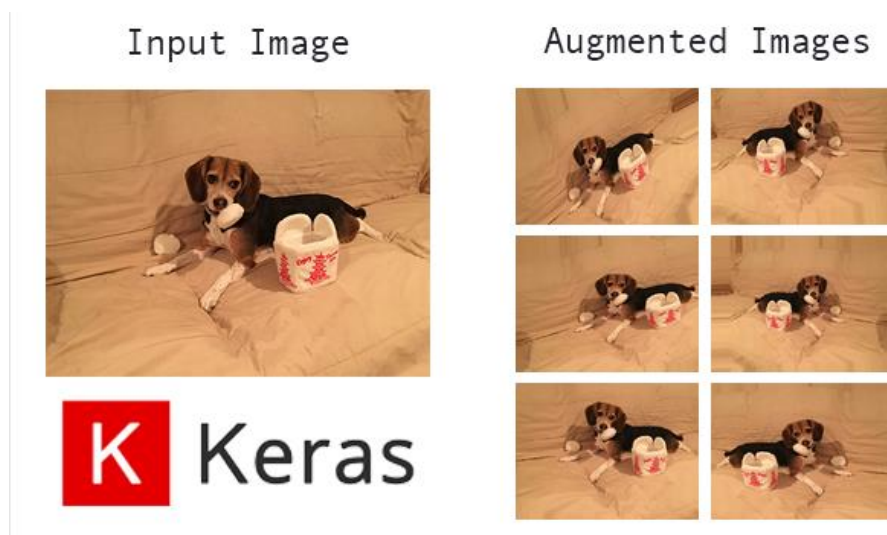
good amount of images in the training set. Also, the shape of the data varies according to the architecture/framework that we use.

Hence, the critical data pre-processing step (the eternally important step in any project). I highly recommend going through the “basics of image processing using Python

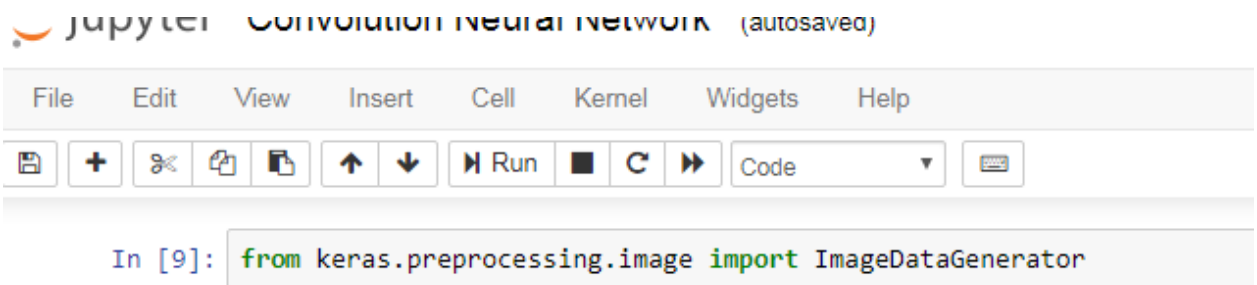
we use Keras’ ImageDataGenerator class to perform data augmentation. i.e, we are using some kind of parameters to process our collected data. The word “augment” means to make something “greater” or “increase” something (in this case, data), the Keras ImageDataGenerator class actually works by:

- ✓ Accepting a batch of images used for training.
- ✓ Taking this batch and applying a series of random transformations to each image in the batch (including random rotation, resizing, shearing, etc.).
- ✓ Replacing the original batch with the new, randomly transformed batch.
- ✓ Training the CNN on this randomly transformed batch (i.e., the original data itself is not used for training).

Note: The ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data.



❖ Import the library



```
In [9]: from keras.preprocessing.image import ImageDataGenerator
```

❖ Define the parameters /arguments for ImageDataGenerator class

```
: train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
```

Note: The ImageDataGenerator transforms each image in the batch by a series of random translations, these translations are based on the arguments

❖ Applying ImageDataGenerator functionality to trainset and testset

```
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

x_train = train_datagen.flow_from_directory(r'E:\dataset\training_set',target_size=(64,64),batch_size=32,class_mode='categorical')
x_test = train_datagen.flow_from_directory(r'E:\dataset\test_set',target_size=(64,64),batch_size=32,class_mode='categorical')

Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

Step 4: Configuring the learning process:

With both the training data defined and model defined, it's time configure the learning process. This is accomplished with a call to the compile() method of the Sequential model class. Compilation requires 3 arguments: an optimizer, a loss function, and a list of metrics.

In our example, set up as a multi-class classification problem, we will use the Adam optimizer, the categorical cross entropy loss function, and include solely the accuracy metric.

```
In [8]: model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

Step 5: Train the model

At this point we have training data and a fully configured neural network to train with said data. All that is left is to pass the data to the model for the training process to commence, a process which is completed by iterating on the training data. Training begins by calling the fit() method.

```
: model.fit_generator(x_train,samples_per_epoch = 8000,epochs=25,validation_data=x_test,nb_val_samples=2000)#
```

Note: This steps takes few minutes based on the epochs (no : of time you would like to train the machine with the given data set) you give .

Step 6: Save The Model:

Your model is to be saved for the future purpose. This saved model ac also be integrated with android application or web application in order to predict something

```
In [18]: model.save('mymodel.h5')# this will save the weights,for keras h5 is extension
```

Step 7: Prediction:

The last and final step is to make use of Saved model to do predictions. We use load model class to load the model. We use imread() class from opencv library to read an image and give it to the model to predict the result. Before giving the original image to predict the class, we have to pre-process that image and apply predictions to get accurate results

```
In [8]: from keras.models import load_model
import numpy as np
import cv2
model = load_model('mymodel.h5')
```

```
In [9]: model.compile(optimizer='adam', loss='binary_crossentropy')
```

```
In [10]: from skimage.transform import resize
def detect(frame):
    try:
        img = resize(frame, (64, 64))
        img = np.expand_dims(img, axis=0)
        if np.max(img) > 1:
            img = img / 255.0
        prediction = model.predict(img)
        print(prediction)
        prediction_class = model.predict_classes(img)
        print(prediction_class)
    except AttributeError:
        print("shape not found")
```

```
In [12]: frame = cv2.imread("cat.jpg")
data = detect(frame)
```