Object Localisation using Deep Convolutional Neural Networking ¶

Members of Team

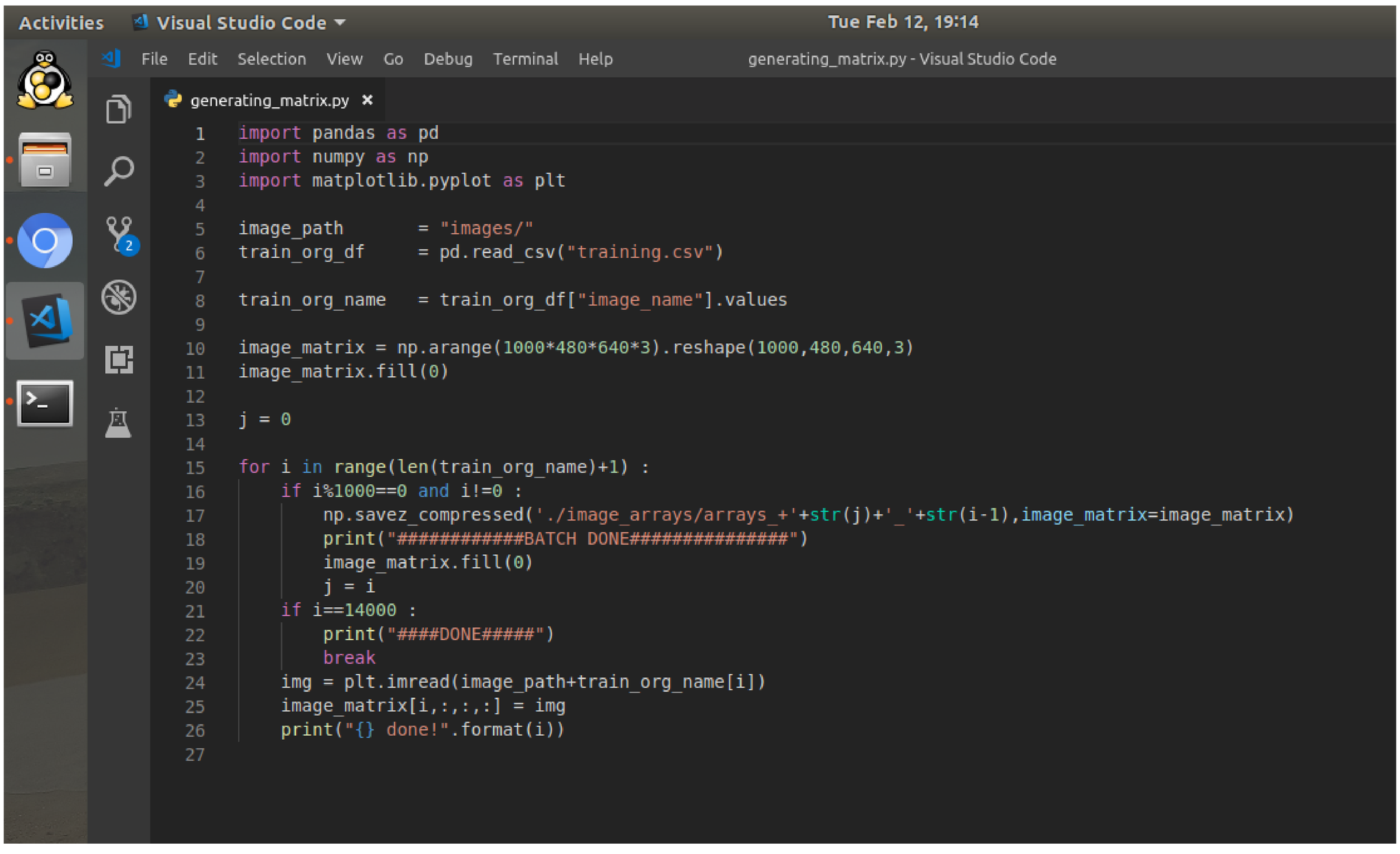
- Amet Vikra, 3rd Year, CS
- Ishaan Verma, 3rd Year, CS
- Dipak Agarwal, 3rd Year, CS

The project has been divided into several steps for clarity from downloading the images.zip folder to prediciting the values in test.csv

STEP 1

First we downloaded the images.zip file from dare2compete webpage.

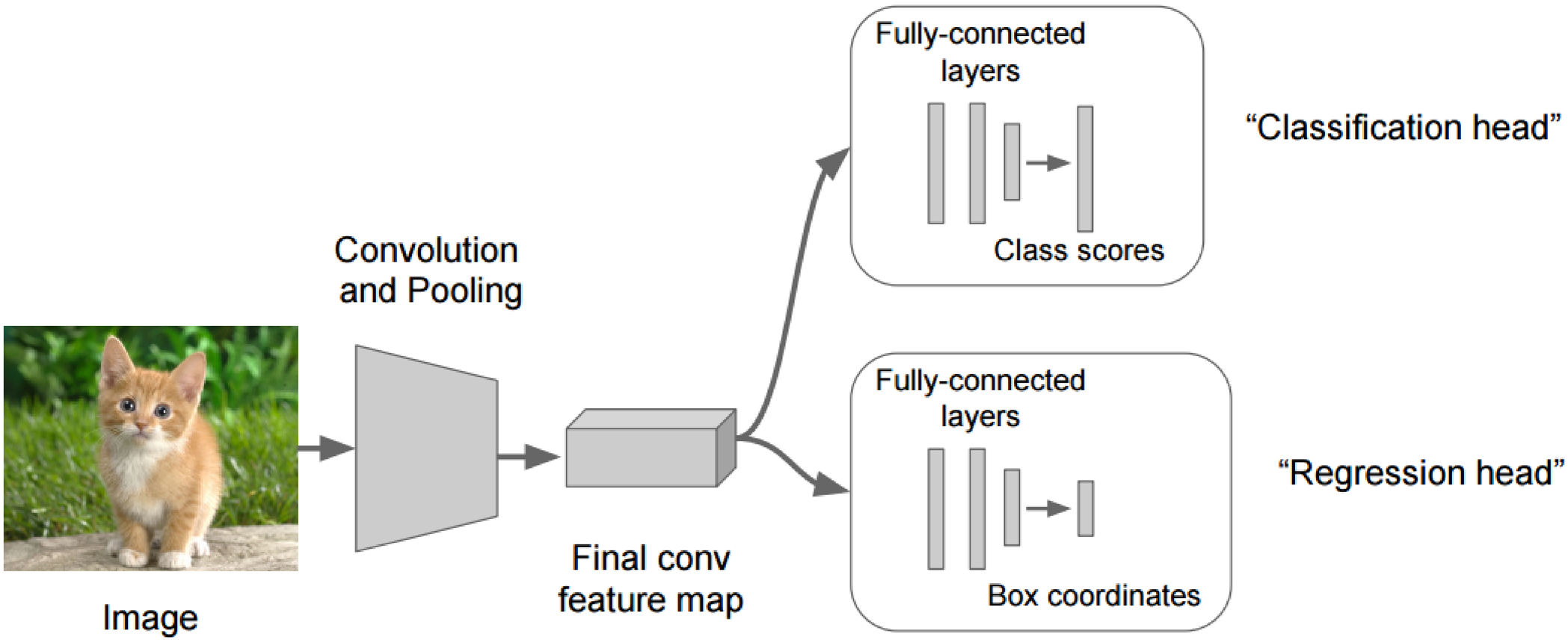
Since the 14000 training images in training.csv file cannot be loaded in python at same time, as numpy array of dimension (14000,480,640,3) cannot be formed on our hardware and it gives **MemoryException**. So we made batches of 1000 images of numpy array of dimension (1000,480,640,3) which was possible on our hardware using the code in "**generating_matrix.py**"



Using above code we created **14 .npz** files which was later used in training our model

STEP 2

Our main challange was to develop an appropriate CNN network that can capture **all the features** of an image of dimension (480,640,3), and will generate a **convolution feature map** of the image. This feature map will be then **flattened** and will be feeded into a **deep neural network regressor head** which will output an array of dimension (4,1) which will be our desired **bounding box coordinates**



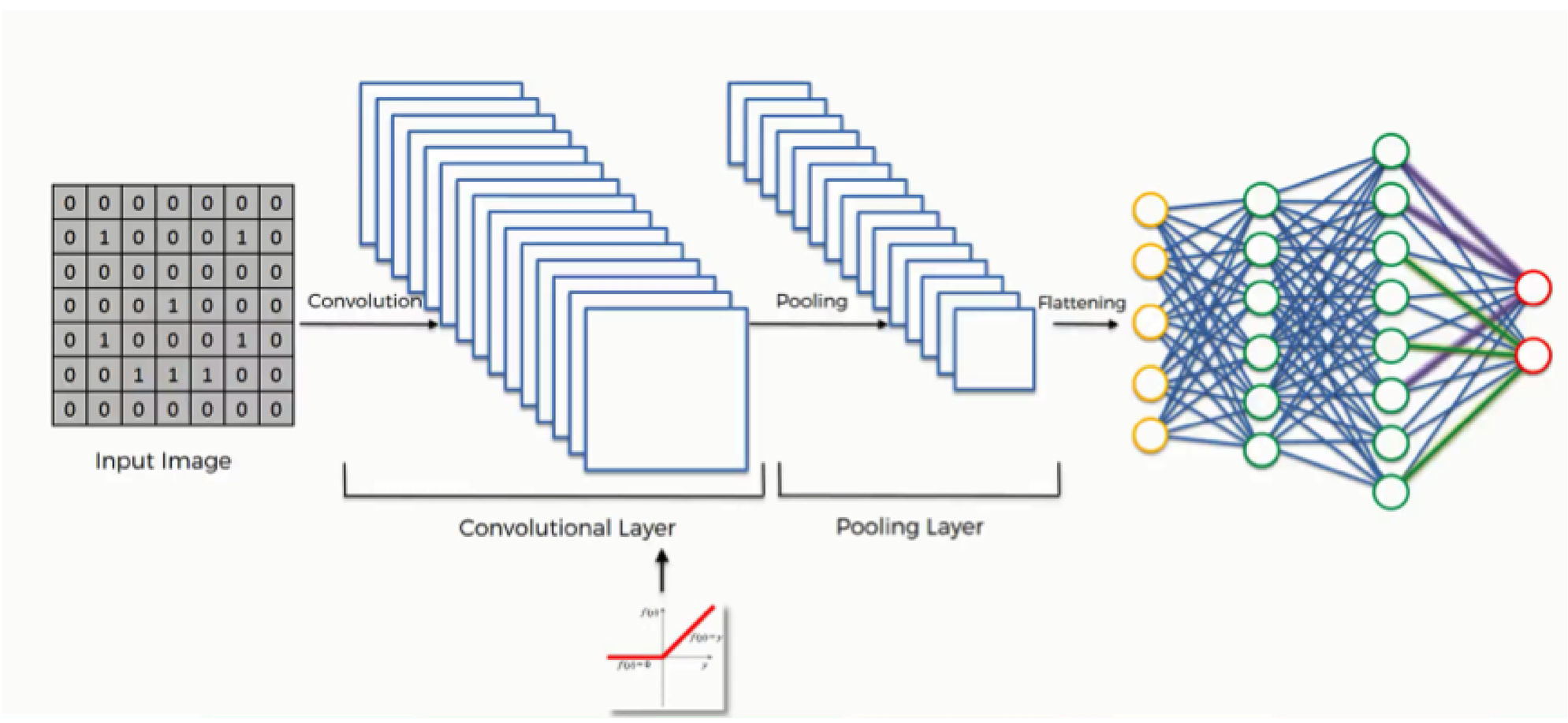
Since we dont have **class scores** so **classification head** can be ignored

STEP 3

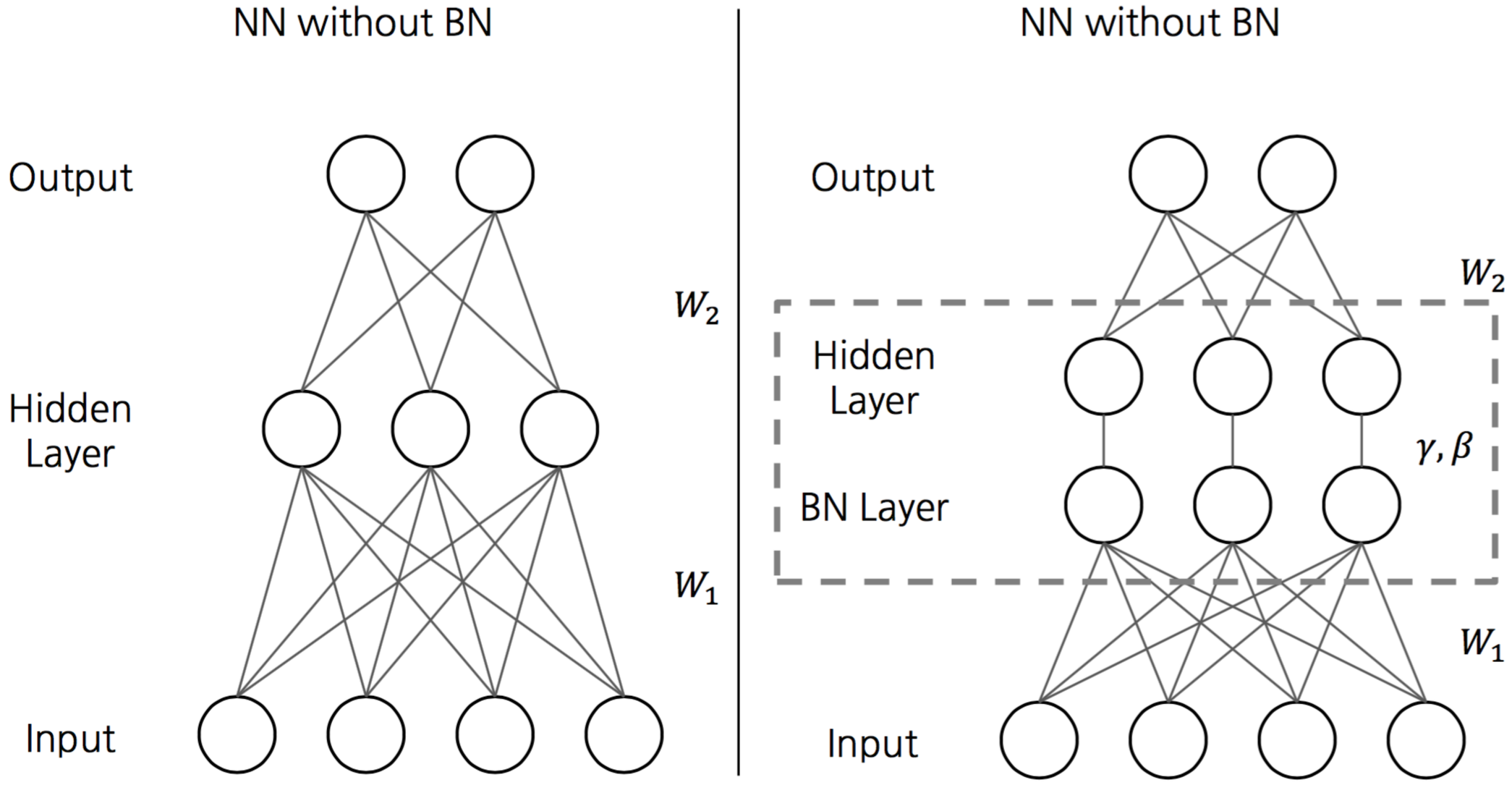
Our CNN model consist of **21 blocks** each consisting of **3 layers**

- A convolutional layer
- A Batch Normalization layer
- A Max Pooling layer

At each **convolution step**, a **sliding window** of size (3,3) is chosen
and number of **filters** is chosen in power of 2 starting from 32 and is increased
for 3-4 steps that is till it reaches 128 or 256 after that the window is decreased to
(1,1) and #filters is set to one step back that is either 64 or 128. The following
pattern is taken to **squeeze out** maximum possible information from the image
The inspiration of this pattern is taken from famous YOLO model which follows a similar
trend.



After every convolution layer a **batch normalization layer** is placed to
counter the internal covariate shift in networks it leads to possible higher learning
rates and helps in training the model. The batch normalization layer is also necessary
because **image feature pool** generated after every convolution layer contains lot
of values (an avg of ~614400) so the values the needs to be in check so that they are not
too off from the desired result



At the end a **max pooling layer** is placed to reduced to height and width of the image while retaining the internal information in the image

And finally at the end a **22nd** convolutional layer which generates the desired feature map of dimension (15,20,425).
Now in this **feature map** of size (15,20) each pixel gives us 425 values which corresponds to a **region in image**. So all this 15 x 20 , 425 values is feeded to a **DNN regressor** which contains **3 layers** (including input). This DNN regressor trains the different regions of images and given the label it tries to configure the parameters so that DNN regressor gives the boundin
g
box coordinates of image.

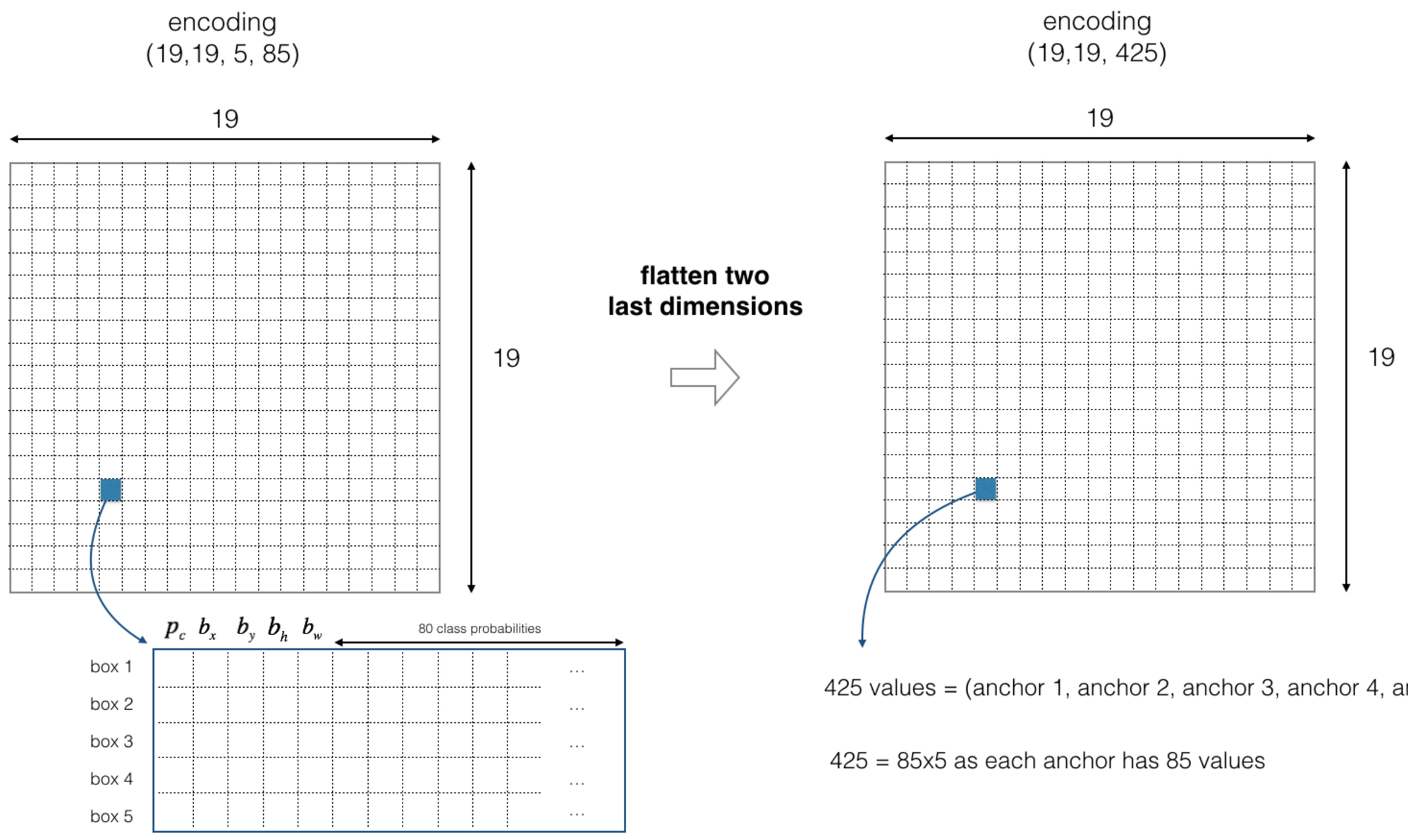


image courtesy:"Coursera/Deeplearning.ai"
NOTE: Here the dimension of our image is (15,20)

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, 480, 640, 3)	0
conv2d_1 (Conv2D)	(None, 480, 640, 32)	896
bn_1 (BatchNormalization)	(None, 480, 640, 32)	128
leaky_relu_1 (LeakyReLU)	(None, 480, 640, 32)	0
max_pooling_1 (MaxPooling2D)	(None, 240, 320, 32)	0
conv2d_2 (Conv2D)	(None, 240, 320, 64)	18496
bn_2 (BatchNormalization)	(None, 240, 320, 64)	256
leaky_relu_2 (LeakyReLU)	(None, 240, 320, 64)	0
max_pooling_2 (MaxPooling2D)	(None, 120, 160, 64)	0
conv2d_3 (Conv2D)	(None, 120, 160, 128)	73856
bn_3 (BatchNormalization)	(None, 120, 160, 128)	512
leaky_relu_3 (LeakyReLU)	(None, 120, 160, 128)	0
conv2d_4 (Conv2D)	(None, 120, 160, 64)	8256
bn_4 (BatchNormalization)	(None, 120, 160, 64)	256
leaky_relu_4 (LeakyReLU)	(None, 120, 160, 64)	0
conv2d_5 (Conv2D)	(None, 120, 160, 128)	73856
bn_5 (BatchNormalization)	(None, 120, 160, 128)	512
leaky_relu_5 (LeakyReLU)	(None, 120, 160, 128)	0
max_pooling_3 (MaxPooling2D)	(None, 60, 80, 128)	0
conv2d_6 (Conv2D)	(None, 60, 80, 256)	295168
bn_6 (BatchNormalization)	(None, 60, 80, 256)	1024
leaky_relu_6 (LeakyReLU)	(None, 60, 80, 256)	0
conv2d_7 (Conv2D)	(None, 60, 80, 128)	32896
bn_7 (BatchNormalization)	(None, 60, 80, 128)	512
leaky_relu_7 (LeakyReLU)	(None, 60, 80, 128)	0
conv2d_8 (Conv2D)	(None, 60, 80, 256)	295168
bn_8 (BatchNormalization)	(None, 60, 80, 256)	1024
leaky_relu_8 (LeakyReLU)	(None, 60, 80, 256)	0
max_pooling_4 (MaxPooling2D)	(None, 30, 40, 256)	0
conv2d_9 (Conv2D)	(None, 30, 40, 512)	1180160
bn_9 (BatchNormalization)	(None, 30, 40, 512)	2048
leaky_relu_9 (LeakyReLU)	(None, 30, 40, 512)	0
conv2d_10 (Conv2D)	(None, 30, 40, 256)	131328
bn_10 (BatchNormalization)	(None, 30, 40, 256)	1024
leaky_relu_10 (LeakyReLU)	(None, 30, 40, 256)	0
conv2d_11 (Conv2D)	(None, 30, 40, 512)	1180160
bn_11 (BatchNormalization)	(None, 30, 40, 512)	2048
leaky_relu_11 (LeakyReLU)	(None, 30, 40, 512)	0

conv2d_12 (Conv2D)	(None, 30, 40, 256)	131328
bn_12 (BatchNormalization)	(None, 30, 40, 256)	1024
leaky_relu_12 (LeakyReLU)	(None, 30, 40, 256)	0
conv2d_13 (Conv2D)	(None, 30, 40, 512)	1180160
bn_13 (BatchNormalization)	(None, 30, 40, 512)	2048
leaky_relu_13 (LeakyReLU)	(None, 30, 40, 512)	0
max_pooling_5 (MaxPooling2D)	(None, 15, 20, 512)	0
conv2d_14 (Conv2D)	(None, 15, 20, 1024)	4719616
bn_14 (BatchNormalization)	(None, 15, 20, 1024)	4096
leaky_relu_14 (LeakyReLU)	(None, 15, 20, 1024)	0
conv2d_15 (Conv2D)	(None, 15, 20, 512)	524800
bn_15 (BatchNormalization)	(None, 15, 20, 512)	2048
leaky_relu_15 (LeakyReLU)	(None, 15, 20, 512)	0
conv2d_16 (Conv2D)	(None, 15, 20, 1024)	4719616
bn_16 (BatchNormalization)	(None, 15, 20, 1024)	4096
leaky_relu_16 (LeakyReLU)	(None, 15, 20, 1024)	0
conv2d_17 (Conv2D)	(None, 15, 20, 512)	524800
bn_17 (BatchNormalization)	(None, 15, 20, 512)	2048
leaky_relu_17 (LeakyReLU)	(None, 15, 20, 512)	0
conv2d_18 (Conv2D)	(None, 15, 20, 1024)	4719616
bn_18 (BatchNormalization)	(None, 15, 20, 1024)	4096
leaky_relu_18 (LeakyReLU)	(None, 15, 20, 1024)	0
conv2d_19 (Conv2D)	(None, 15, 20, 1024)	9438208
bn_19 (BatchNormalization)	(None, 15, 20, 1024)	4096
leaky_relu_19 (LeakyReLU)	(None, 15, 20, 1024)	0
conv2d_20 (Conv2D)	(None, 15, 20, 1024)	9438208
bn_20 (BatchNormalization)	(None, 15, 20, 1024)	4096
leaky_relu_20 (LeakyReLU)	(None, 15, 20, 1024)	0
conv2d_21 (Conv2D)	(None, 15, 20, 1024)	9438208
bn_21 (BatchNormalization)	(None, 15, 20, 1024)	4096
leaky_relu_21 (LeakyReLU)	(None, 15, 20, 1024)	0
conv2d_22 (Conv2D)	(None, 15, 20, 425)	435625
flatten_1 (Flatten)	(None, 127500)	0
fc_1 (Dense)	(None, 64)	8160064
fc_2 (Dense)	(None, 64)	4160
fc_3 (Dense)	(None, 4)	260

Total params: 56,765,997
Trainable params: 56,745,453
Non-trainable params: 20,544

STEP 4

Now for training our model, we did it in the following way :

In [16]:
```
#### Load fold10 -- arrays_9000_9999 #####
loaded = np.load('./images_arrays/arrays_9000_9999.npz')
X = loaded['image_matrix']
y = train_org.drop(['image_name'],axis=1)[:1000].values
X_train,y_train,X_test,y_test = shuffle_batch(X,y,SEED)
```

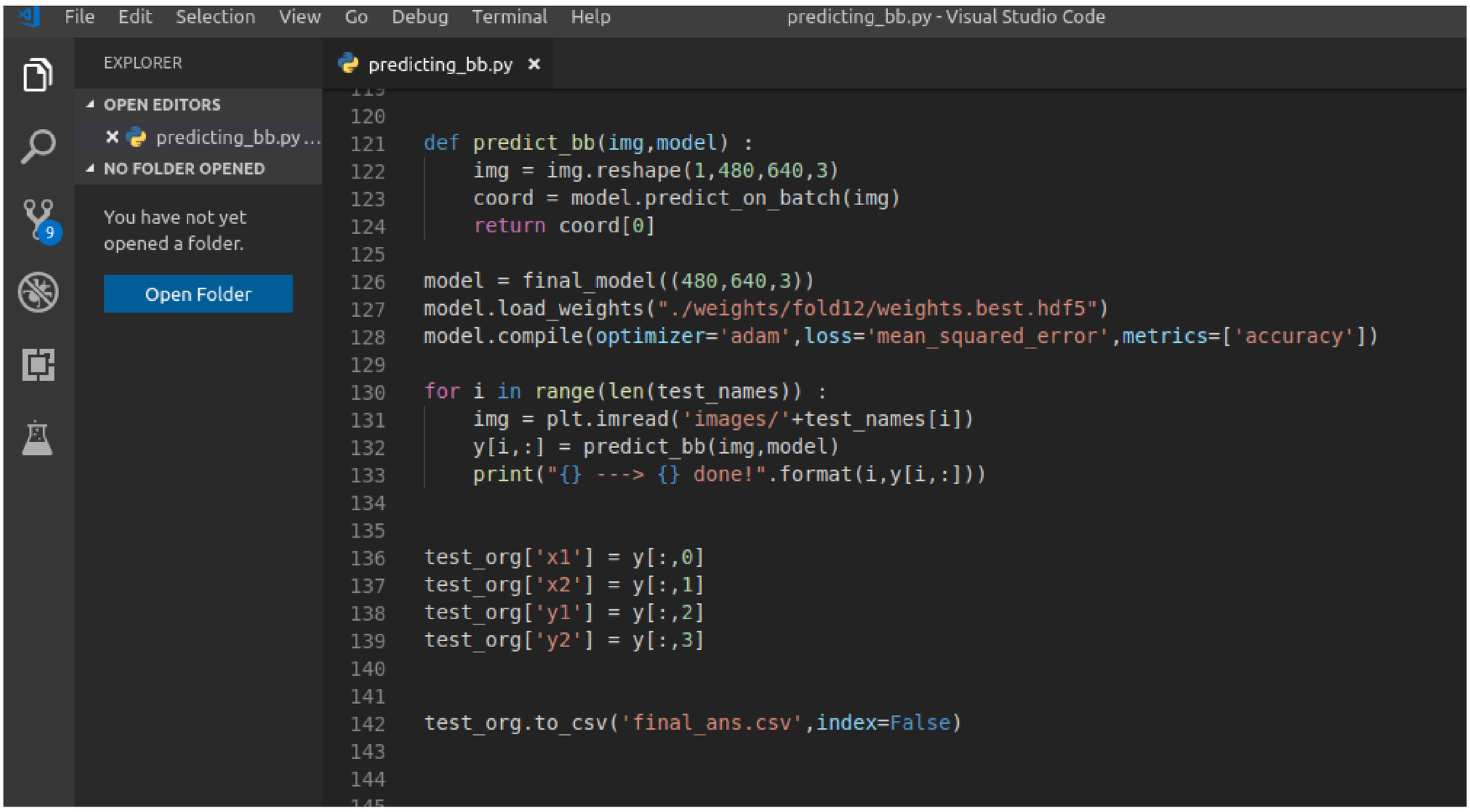
In [17]:
```
model = final_model((480,640,3))
model.load_weights("./weights/fold9/weights.best.hdf5")
model.compile(optimizer='adam',loss='mean_squared_error',metrics=['accuracy'])
filepath="./weights/fold10/weights.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
```

In [18]:
```
model.fit(X_train,y_train,validation_split=0.2, epochs=10, batch_size=8, callbacks=callbacks_list, verbose=0)
```

The numpy arrays are loaded from .npz file which are already created then the model, alredy defined abo
ve,
is instantiated and the **weights** from the previous fold is loaded beacuse as already discussed all numpy
image arrays cant be loaded so they are trained in batches of 1000 and after training a batch of 1000 t
he
the weights for the best validation set is saved for the use of next fold

STEP 5

Now for predicting the values, we did it in the following way :



The **best weights** from all the folds is choosen and used to predict the unknown values and the result is exported as .csv file

THATS ALL

THANK YOU!