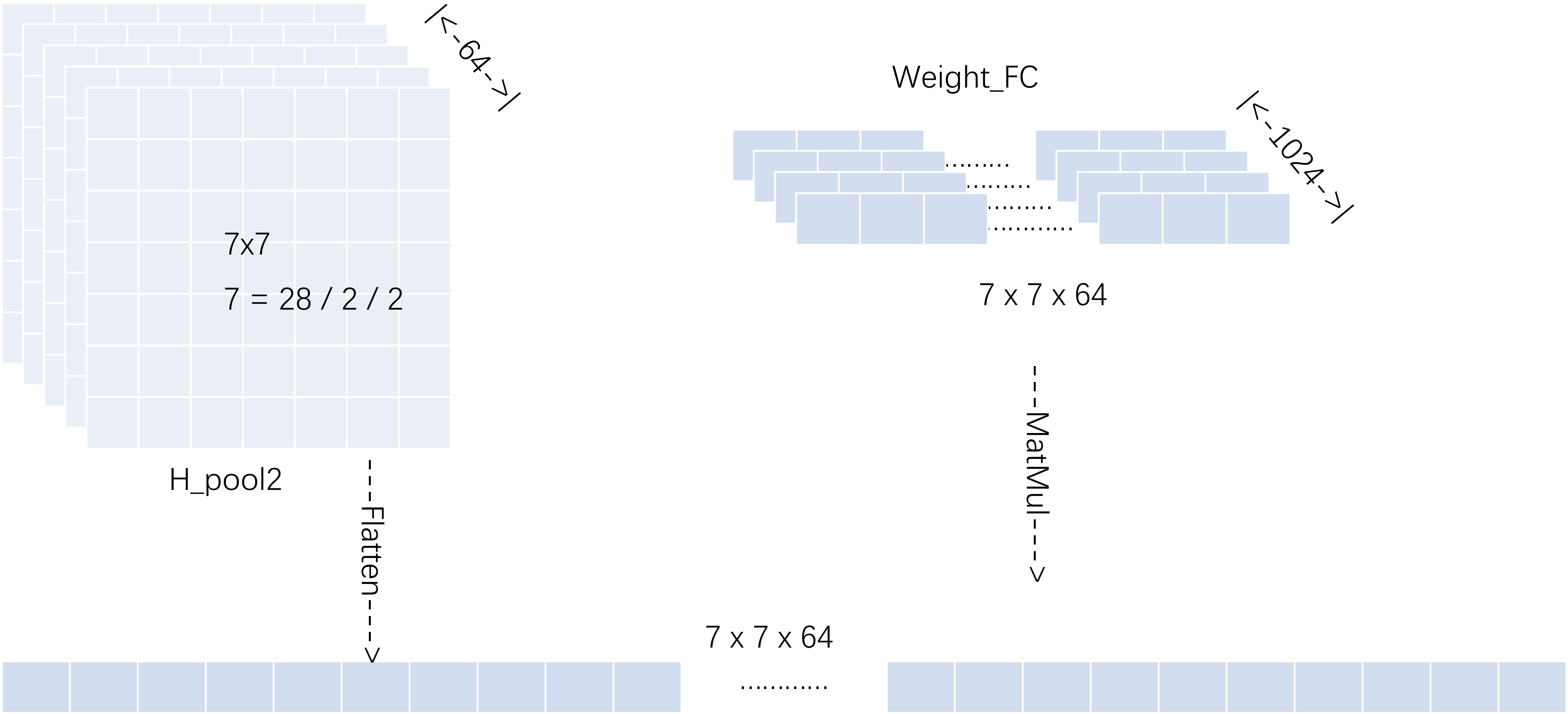
Final Project

zhihaozh@brandeis.edu

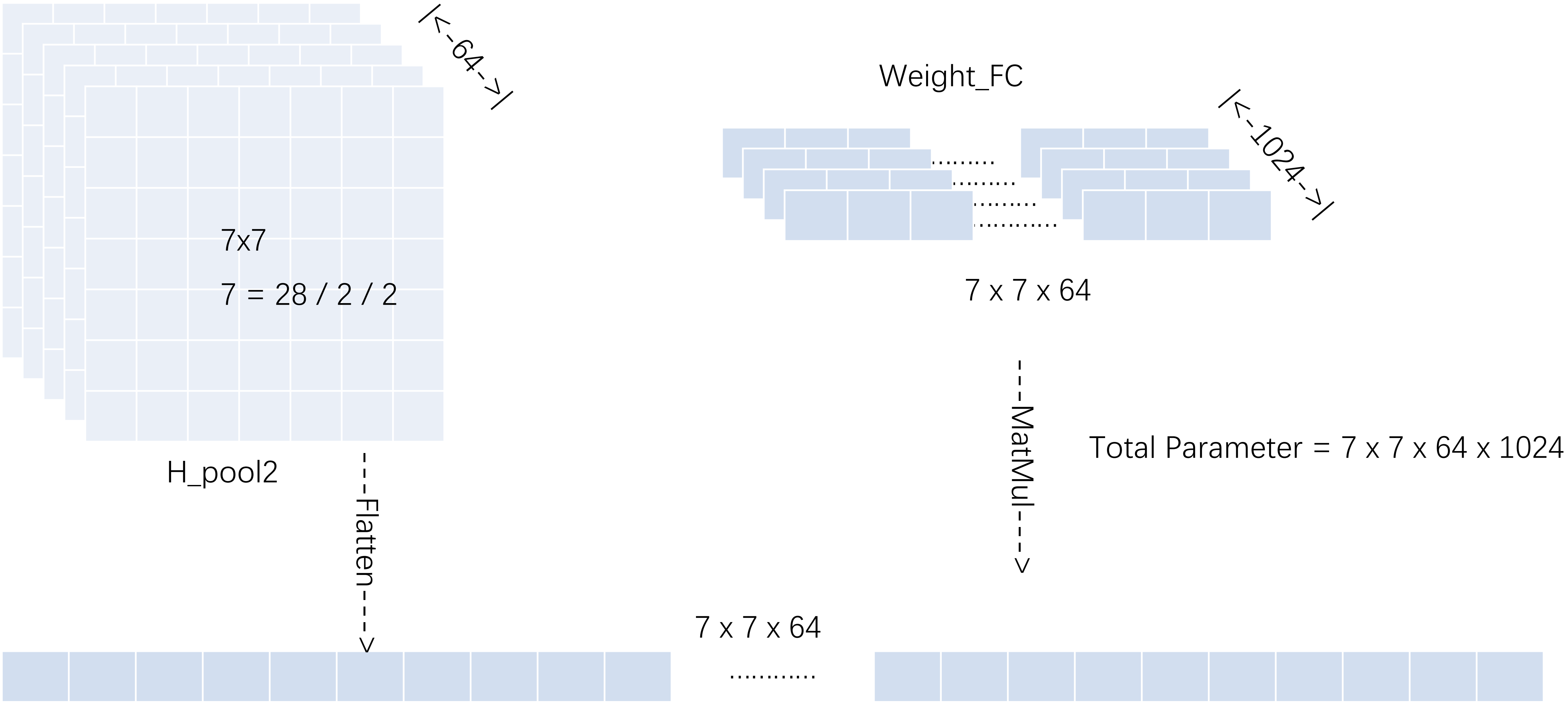
Convolute Arbitrary Size of Image

- X = tf.placeholder(tf.float32, shape=[none,28,28,1])
- Y_ = tf.placeholder(tf.float32,shape=[none,10])
- …
- H_flat = tf.reshape(h_pool2, [-1, 7*7*64])
- H_fc1 = tf.mul(…)
- …

Convolute Arbitrary Size of Image



Convolute Arbitrary Size of Image



|<-64->|

7x7

7 = 28 / 2 / 2

H_pool2

---Flatten--->

Weight_FC

|<-1024->|

7 x 7 x 64

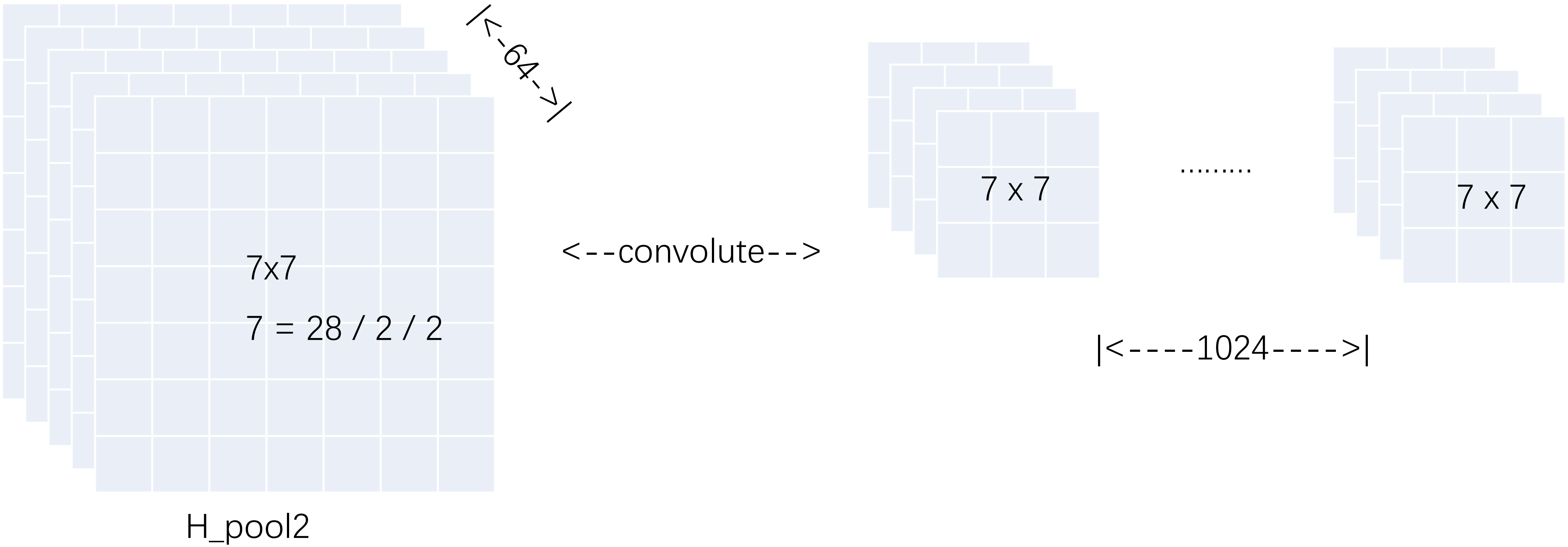
---MatMul--->

Total Parameter = 7 x 7 x 64 x 1024

7 x 7 x 64

....

Convolute Arbitrary Size of Image



|<-64->|

7x7

7 = 28 / 2 / 2

H_pool2

<--convolute-->

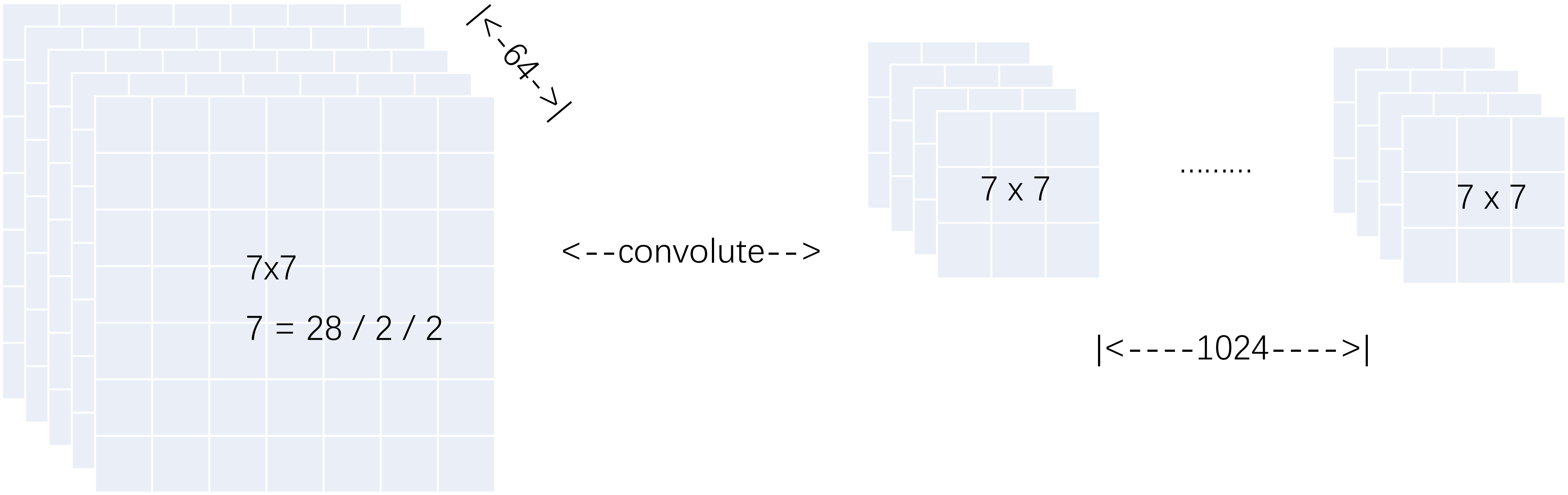
7 x 7

....

7 x 7

|<----1024---->|

Convolute Arbitrary Size of Image



|<-64->|

7x7

7 = 28 / 2 / 2

<--convolute-->

7 x 7

....

7 x 7

|<----1024---->|

H_pool2

Total Parameter = 7 x 7 x 64 x 1024

Convolute Arbitrary Size of Image

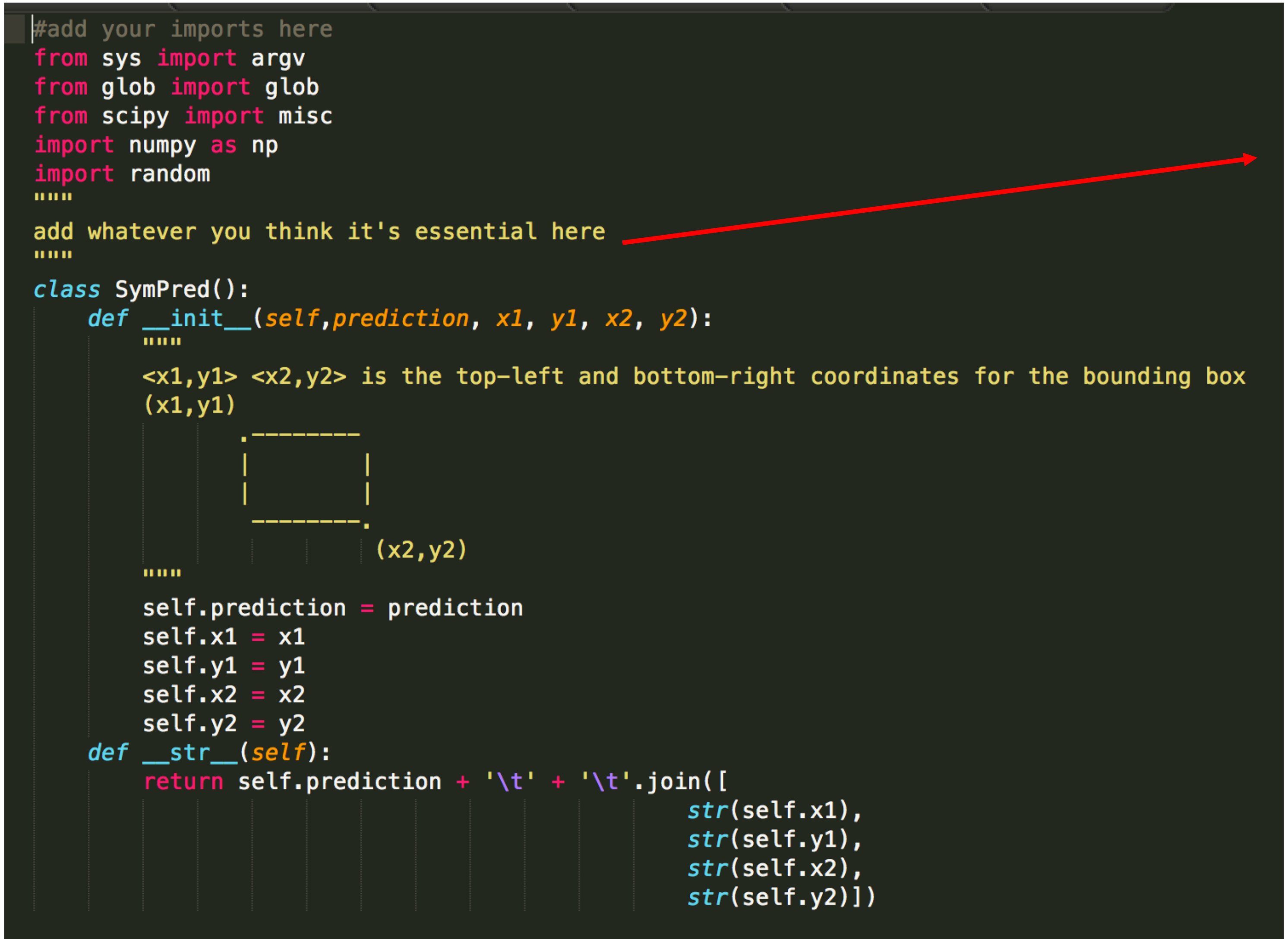
- If isTrain
  - …
  - Padding = VALID
- Else:
  - X = tf.placeholder(⋯, shape=[1,none, none, 1])
  - Padding = SAME
- …
- W_fc = weight_variable([7,7,64,1024])
- H_fc = conv2d(⋯) → Shape = 1,1,1,1
- …
- H_readout = reshape(⋯,[-1,10])

Create Your own Training Batches

- Batch Shape of Mnist
- Train_x = [batch_size, 784]
- …reshape…

- Train_x = [batch_size, image_x, image_y,1]
- Train_y = [batch_size, label_num]

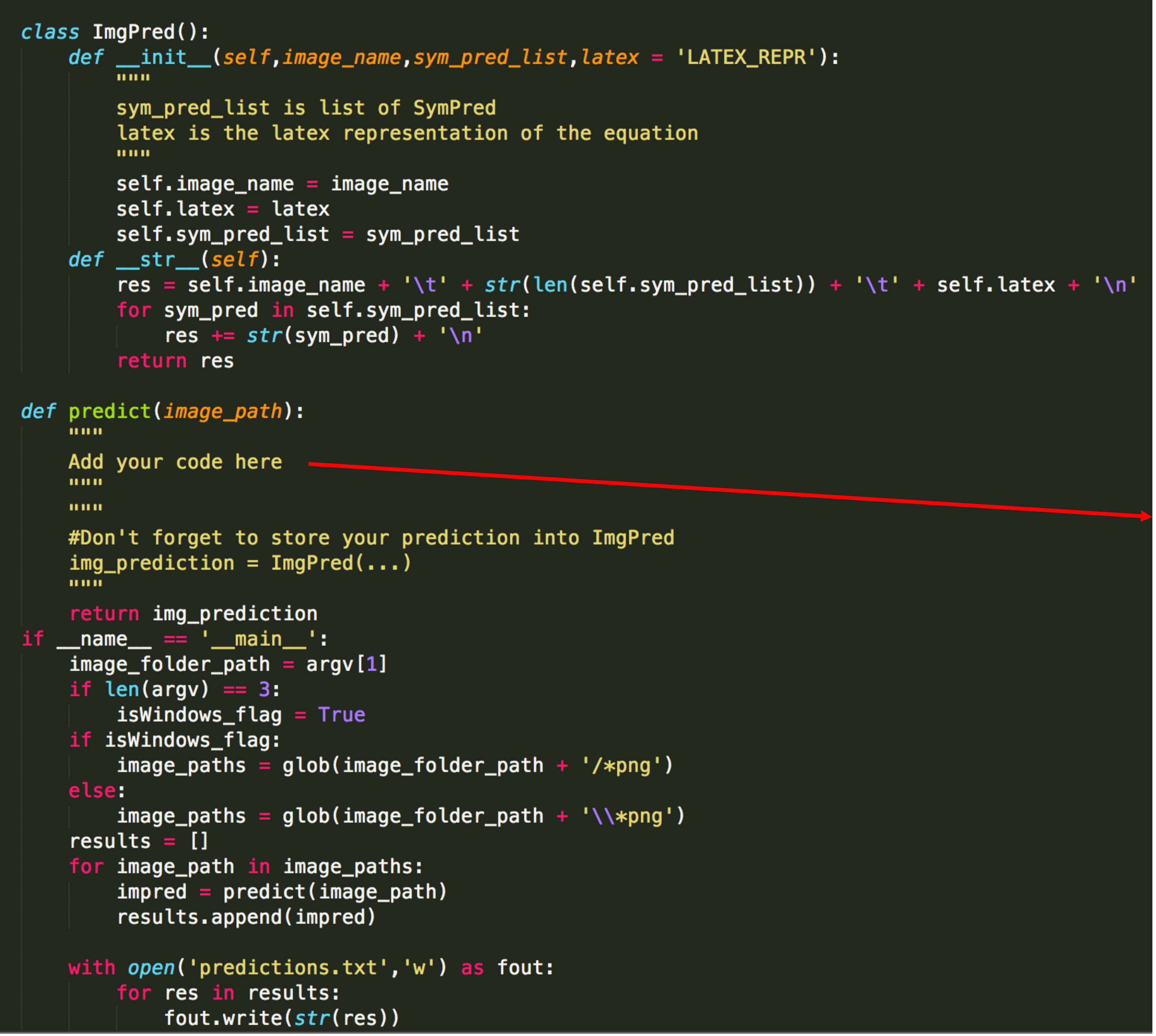
How to use Predict.py



```python
#add your imports here
from sys import argv
from glob import glob
from scipy import misc
import numpy as np
import random
"""
add whatever you think it's essential here
"""
class SymPred():
    def __init__(self,prediction, x1, y1, x2, y2):
        """
        <x1,y1> <x2,y2> is the top-left and bottom-right coordinates for the bounding box
        (x1,y1)
               .---------
               |        |
               |        |
                ---------.
                          (x2,y2)
        """
        self.prediction = prediction
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
    def __str__(self):
        return self.prediction + '\t' + '\t'.join([
                                                str(self.x1),
                                                str(self.y1),
                                                str(self.x2),
                                                str(self.y2)])
```

Add definitions of your Neural Network
And other essential functions

Or you can import those use ‘import’



```
class ImgPred():
    def __init__(self,image_name,sym_pred_list,latex = 'LATEX_REPR'):
        """
        sym_pred_list is list of SymPred
        latex is the latex representation of the equation
        """
        self.image_name = image_name
        self.latex = latex
        self.sym_pred_list = sym_pred_list
    def __str__(self):
        res = self.image_name + '\t' + str(len(self.sym_pred_list)) + '\t' + self.latex + '\n'
        for sym_pred in self.sym_pred_list:
            res += str(sym_pred) + '\n'
        return res

def predict(image_path):
    """
    Add your code here
    """
    """
    #Don't forget to store your prediction into ImgPred
    img_prediction = ImgPred(...)
    """
    return img_prediction
if __name__ == '__main__':
    image_folder_path = argv[1]
    if len(argv) == 3:
        isWindows_flag = True
    if isWindows_flag:
        image_paths = glob(image_folder_path + '/*png')
    else:
        image_paths = glob(image_folder_path + '\\*png')
    results = []
    for image_path in image_paths:
        impred = predict(image_path)
        results.append(impred)

    with open('predictions.txt','w') as fout:
        for res in results:
            fout.write(str(res))
```

Call your own prediction function here
And store your annotation result in
SymPred and ImgPred