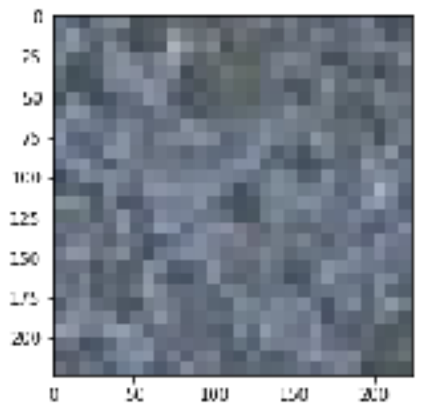
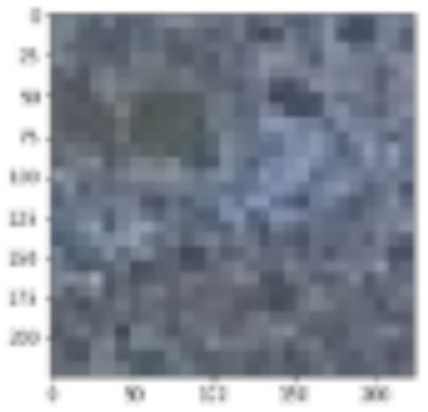
Geological Image Similarity

Objective: Create a machine learning model that returns the top K images that are most similar to an input image.

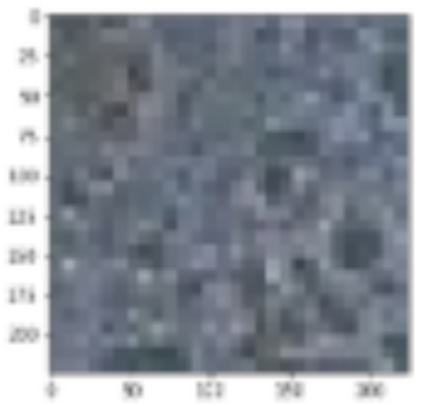
Example: Input Image



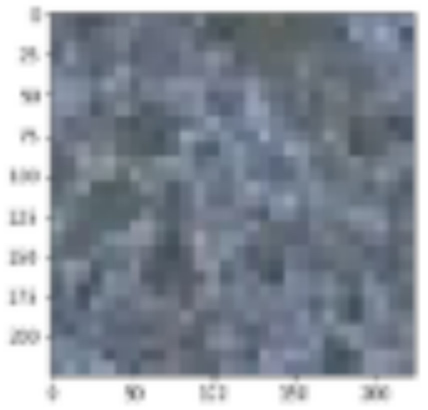
Output: K = 8 Similar Images + Similarity Score



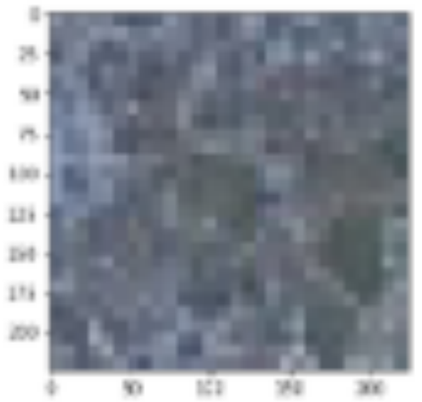
0.9857



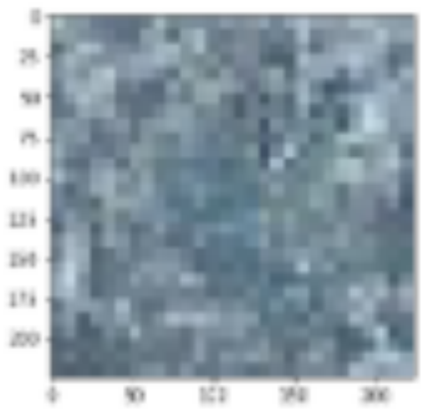
0.9816



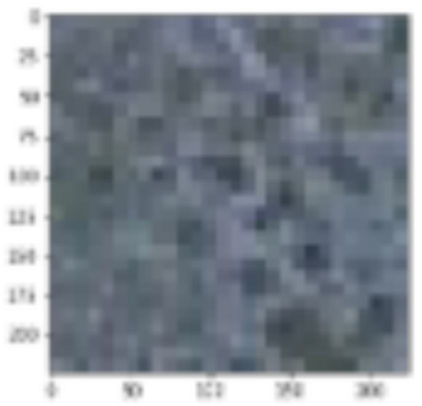
0.9844



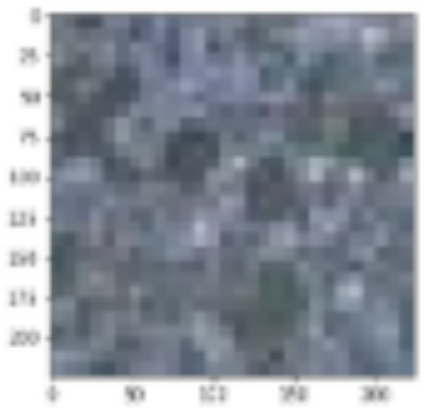
0.9795



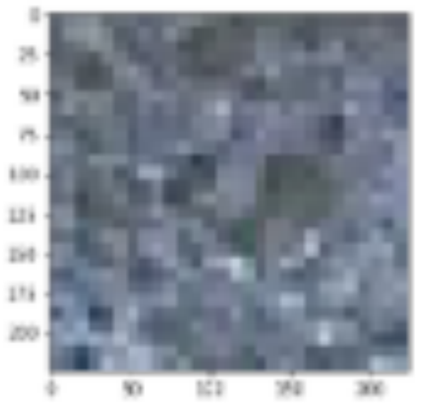
0.982



0.9779

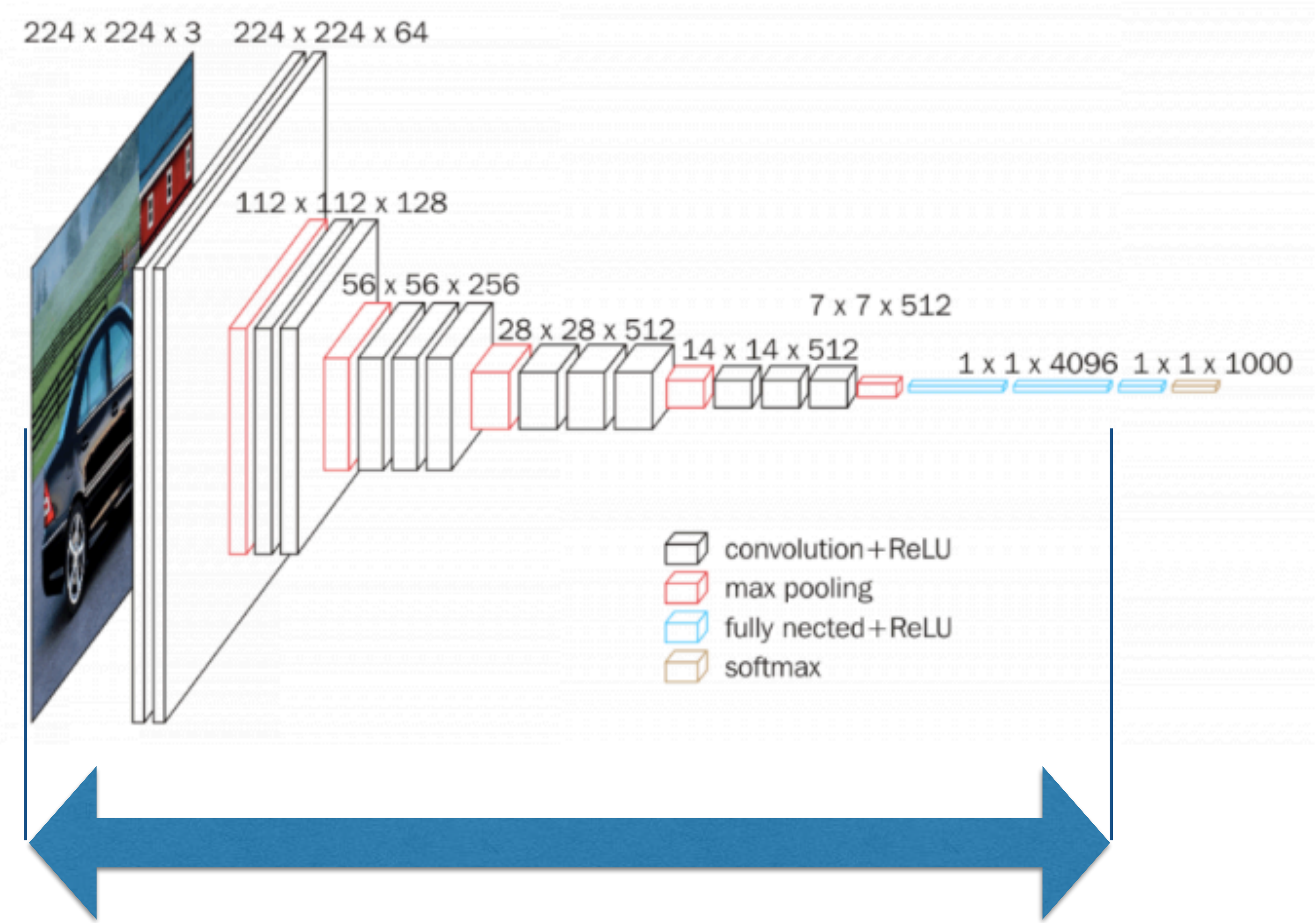


0.9819



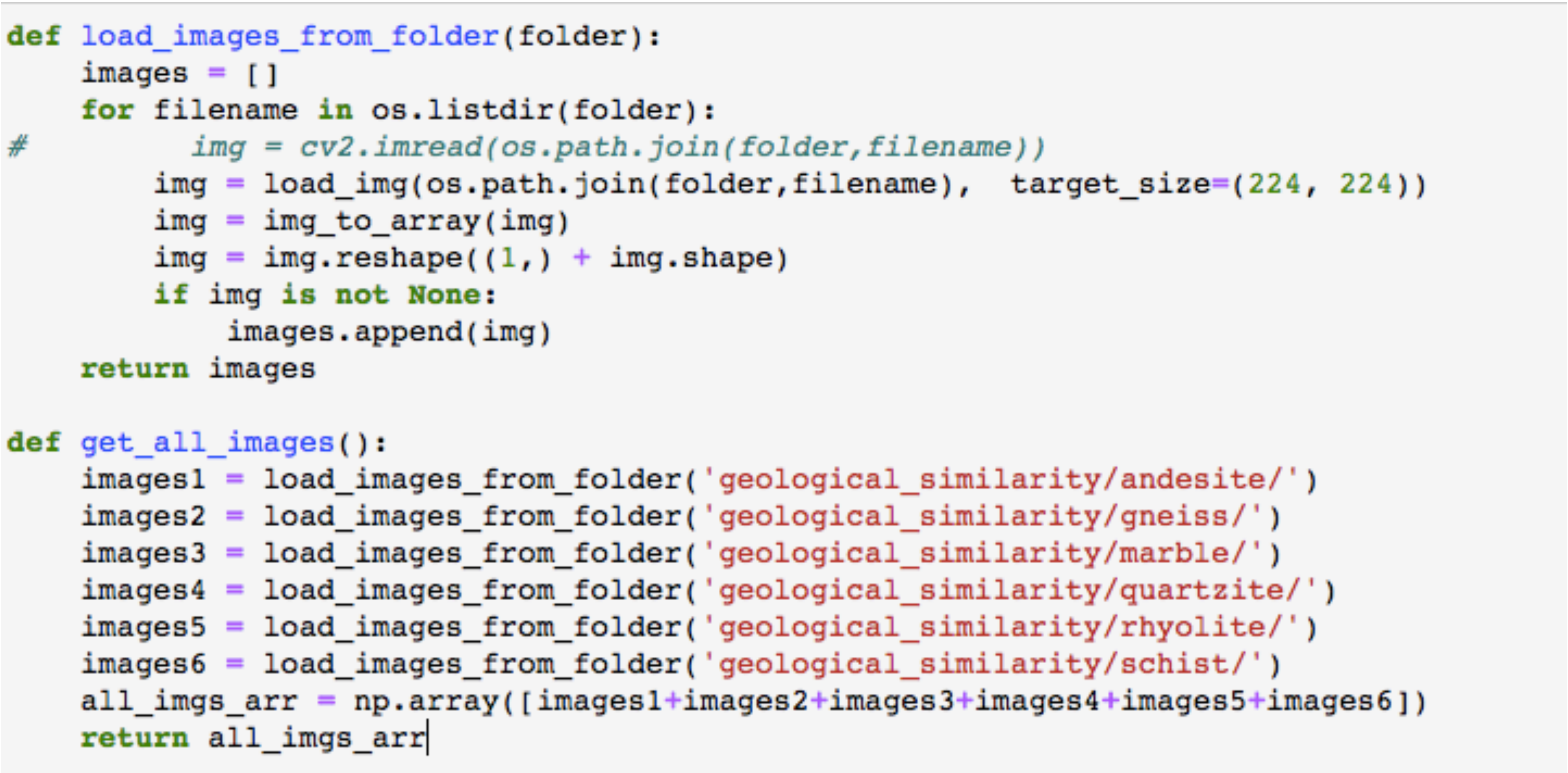
0.9767

Method: Using VGG16 by K. Simonyan and A. Zisserman



- Taking the geological images through the model and stopping one layer before the classification - shape of (4096,)
- After vectorizing the images to a (4096,) array then will look for similarities among the vectors (cosine similarity)

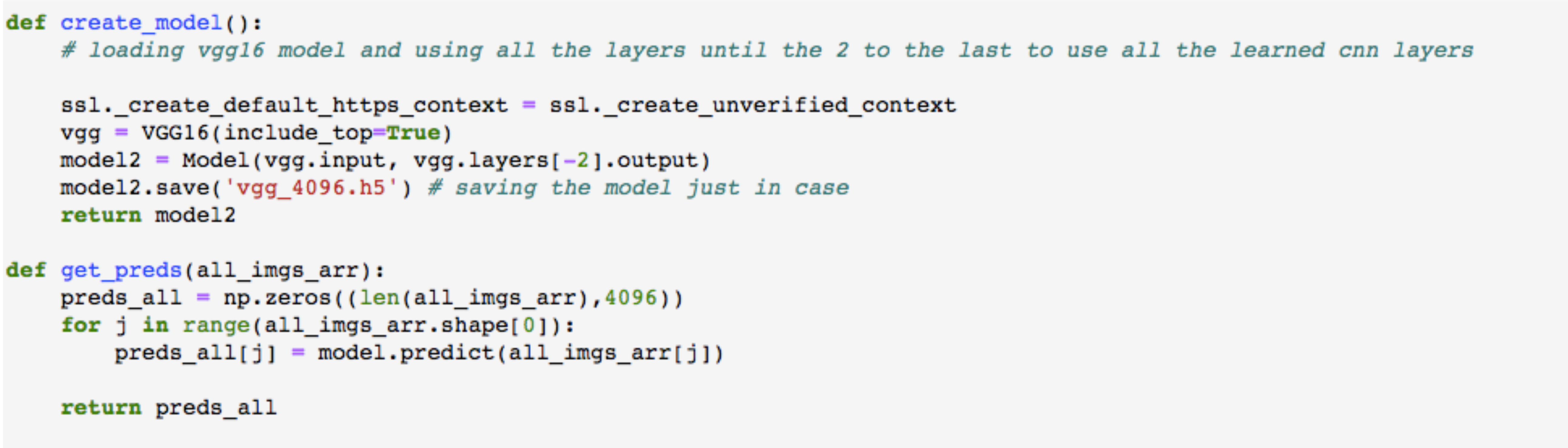
Step 1: Load, Read and save the data into Arrays



```python
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
#         img = cv2.imread(os.path.join(folder,filename))
        img = load_img(os.path.join(folder,filename),  target_size=(224, 224))
        img = img_to_array(img)
        img = img.reshape((1,) + img.shape)
        if img is not None:
            images.append(img)
    return images

def get_all_images():
    images1 = load_images_from_folder('geological_similarity/andesite/')
    images2 = load_images_from_folder('geological_similarity/gneiss/')
    images3 = load_images_from_folder('geological_similarity/marble/')
    images4 = load_images_from_folder('geological_similarity/quartzite/')
    images5 = load_images_from_folder('geological_similarity/rhyolite/')
    images6 = load_images_from_folder('geological_similarity/schist/')
    all_imgs_arr = np.array([images1+images2+images3+images4+images5+images6])
    return all_imgs_arr
```

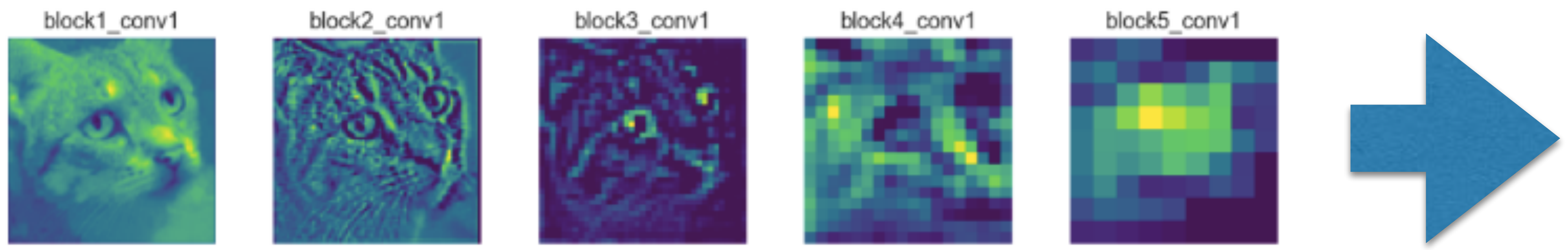
- Reading all the images in 6 different folders
- Reshaping the array to fit the input of VGG16 model.
- Saving it all on single array

Step 2: Predicting with VGG16



```
def create_model():
    # loading vgg16 model and using all the layers until the 2 to the last to use all the learned cnn layers

    ssl._create_default_https_context = ssl._create_unverified_context
    vgg = VGG16(include_top=True)
    model2 = Model(vgg.input, vgg.layers[-2].output)
    model2.save('vgg_4096.h5') # saving the model just in case
    return model2

def get_preds(all_imgs_arr):
    preds_all = np.zeros((len(all_imgs_arr),4096))
    for j in range(all_imgs_arr.shape[0]):
        preds_all[j] = model.predict(all_imgs_arr[j])

    return preds_all
```

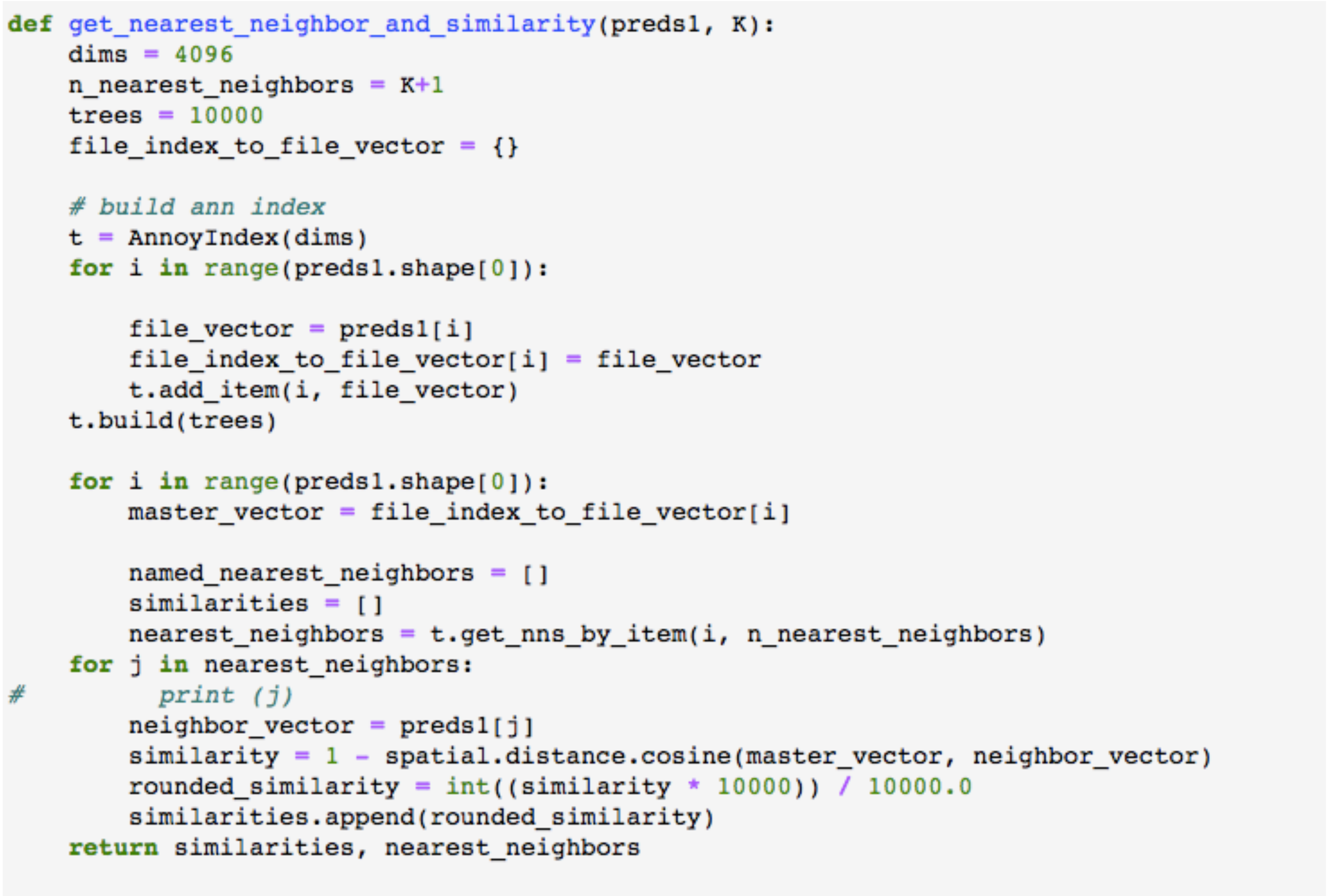
- Loading the VGG16 model with Keras
- Getting the layer before the classification from VGG16
- Getting predictions from the image array to get the (4096,) shaped output



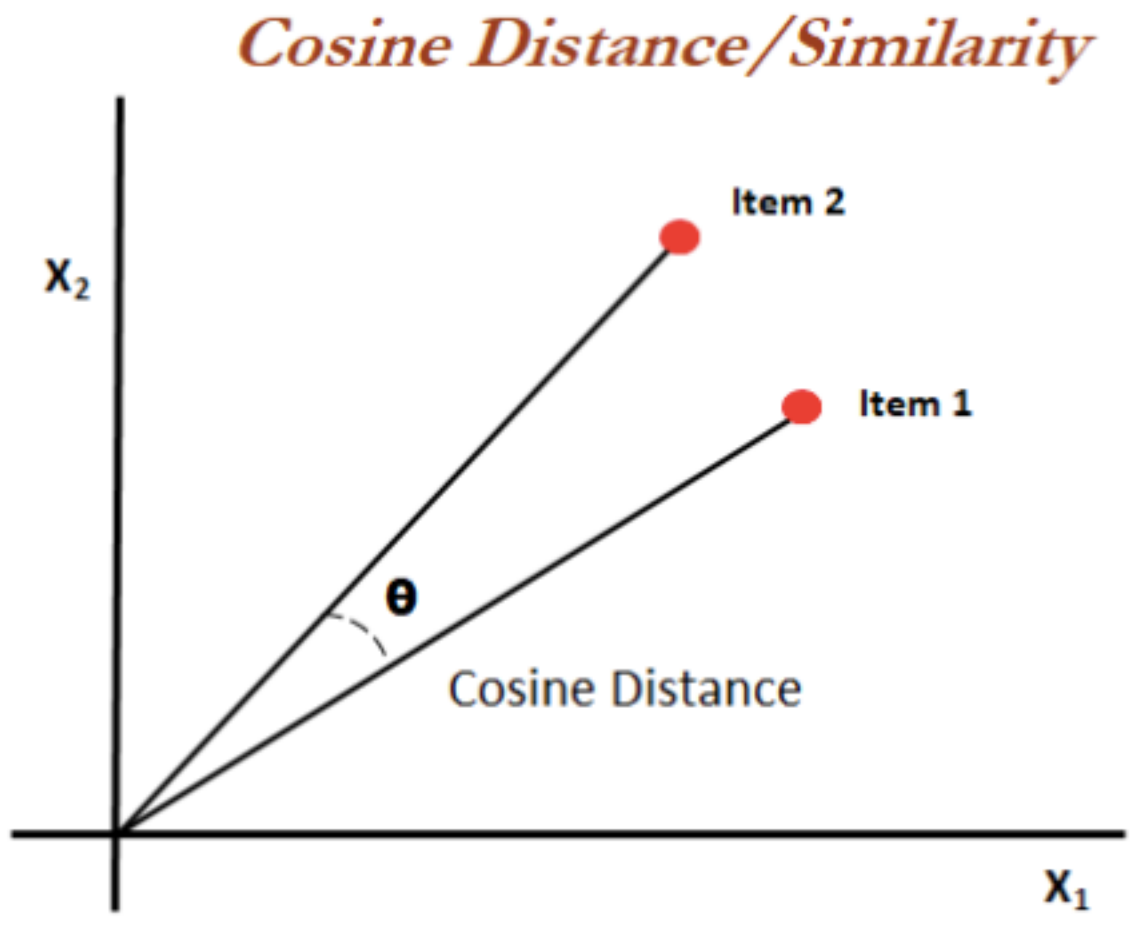
Fully Connected Layer

Only one of the many feature maps of each block of CNN in VGG16

Step 3: Image Similarity (Cosine Similarity)



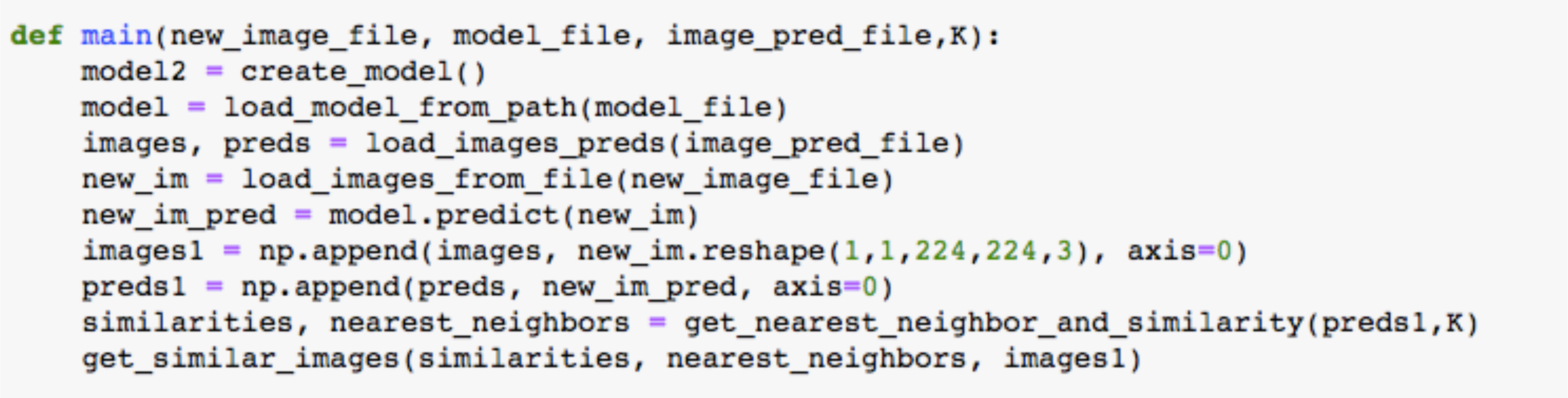
```python
def get_nearest_neighbor_and_similarity(preds1, K):
    dims = 4096
    n_nearest_neighbors = K+1
    trees = 10000
    file_index_to_file_vector = {}

    # build ann index
    t = AnnoyIndex(dims)
    for i in range(preds1.shape[0]):

        file_vector = preds1[i]
        file_index_to_file_vector[i] = file_vector
        t.add_item(i, file_vector)
    t.build(trees)

    for i in range(preds1.shape[0]):
        master_vector = file_index_to_file_vector[i]

        named_nearest_neighbors = []
        similarities = []
        nearest_neighbors = t.get_nns_by_item(i, n_nearest_neighbors)
    for j in nearest_neighbors:
#         print (j)
        neighbor_vector = preds1[j]
        similarity = 1 - spatial.distance.cosine(master_vector, neighbor_vector)
        rounded_similarity = int((similarity * 10000)) / 10000.0
        similarities.append(rounded_similarity)
    return similarities, nearest_neighbors
```



- AnnoyIndex get_nns_by_item(i, n_nearest_neighbor) calculates the K nearest neighbors for the input image based on Cosine Similarity

- Comparing the model prediction vectors and finding the most similar ones.

Process

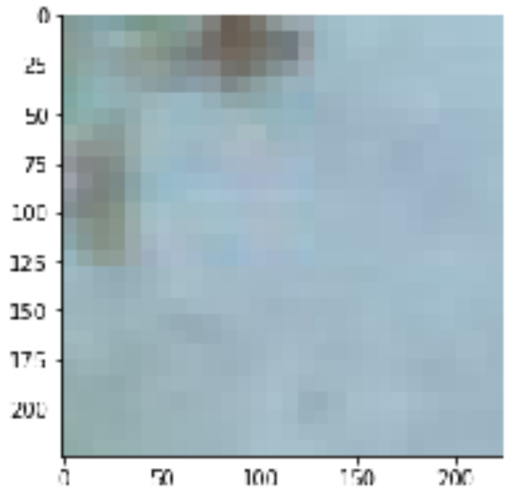


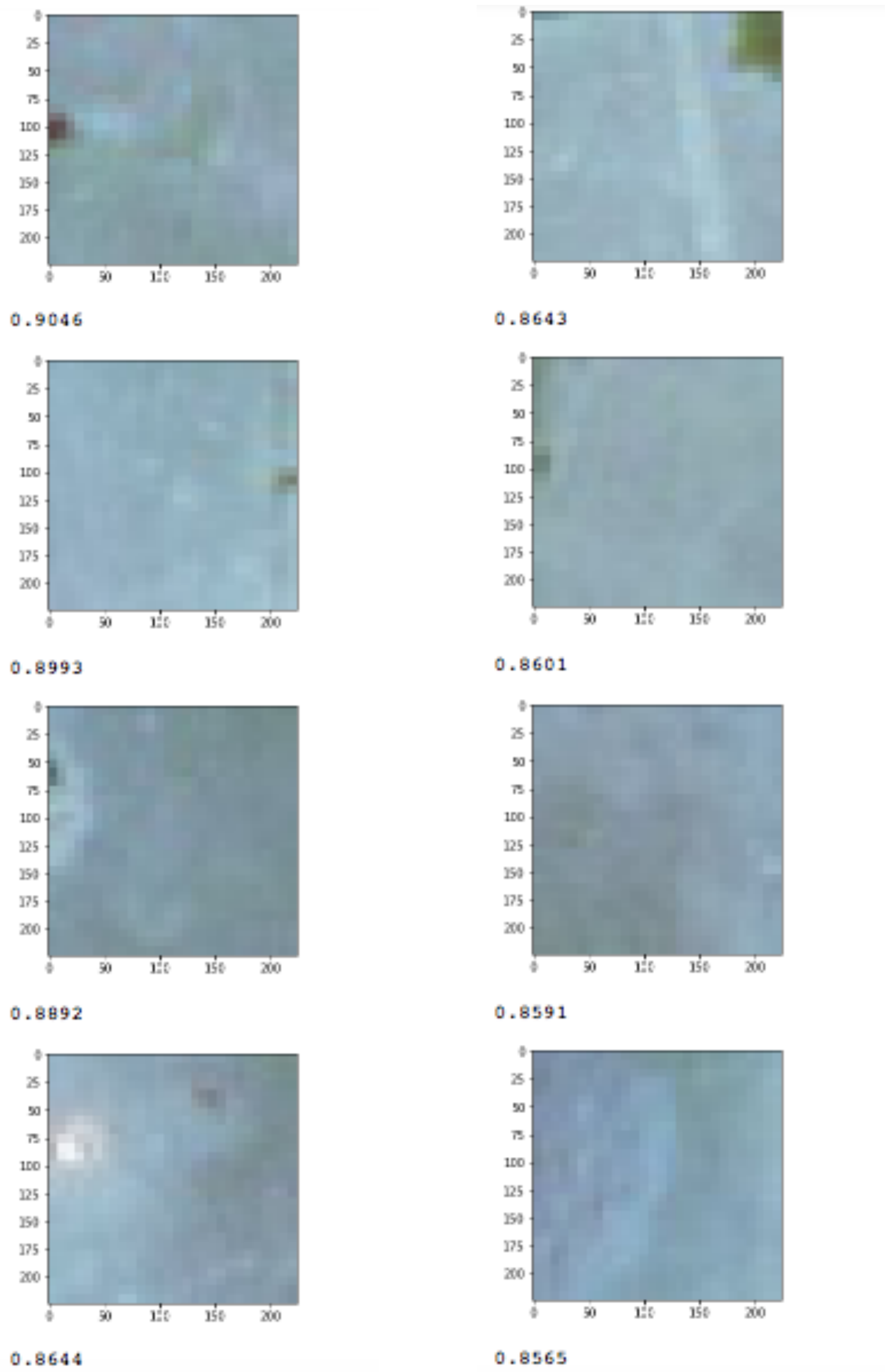
```python
def main(new_image_file, model_file, image_pred_file,K):
    model2 = create_model()
    model = load_model_from_path(model_file)
    images, preds = load_images_preds(image_pred_file)
    new_im = load_images_from_file(new_image_file)
    new_im_pred = model.predict(new_im)
    images1 = np.append(images, new_im.reshape(1,1,224,224,3), axis=0)
    preds1 = np.append(preds, new_im_pred, axis=0)
    similarities, nearest_neighbors = get_nearest_neighbor_and_similarity(preds1,K)
    get_similar_images(similarities, nearest_neighbors, images1)
```

- Step 1: creating the model
- Step 2: loading the existing images and their corresponding predictions
- Step 3: Appending the new image and it’s corresponding prediction to the 2 arrays of images and predictions.
- Step 4: Calculating the most similar image prediction to the input image
- Step 5: Visualizing the similar images

Results

Example: Input Image





Output: K =8 Similar Images + Similarity Score

Next Steps

- Improving load and reading images and predictions
  - Distributed computing
  - Cloud computing

- Classifying images before the finding similar images
  - By fine tuning the VGG16 to the 6 given classes

- Exploring different similarity algorithms
  - Siamese Networks
  - K Nearest Neighbors