```
model_path = ('/content/drive/MyDrive/Project/Train_Yolo/runs/detect/train/weights/last.pt')

# Load a model
model = YOLO(model_path)  # load a custom model
```

Cài đặt mô hình: model_path = (‘*model_path’*)

model_path: đường dẫn đến file model đã train (last.pt/best.pt)

```
threshold = 0.5
def pred(img_path):
  frame = Image.open(img_path)

  frame1 = np.asarray(frame)
  results = model(frame1)[0]
  # img1 = ImageDraw.Draw(frame)

  for result in results.boxes.data.tolist():
    row = []
    column = []
    x1, y1, x2, y2, score, class_id = result

    if score > threshold:
        cv2.rectangle(frame1, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 4)
        cv2.putText(frame1, results.names[int(class_id)].upper() + ": " + str(round(score, 2)), (int(x1), int(y1 - 10)),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3, cv2.LINE_AA)
        row.append(int(x1))
        row.append(int(x2))
        column.append(int(y1))
        column.append(int(y2))

  plt.imshow(frame1)
  return row, column
```

Hàm dự đoán với đầu vào *img_path* là đường dẫn đến hình ảnh muốn tìm kiếm và sẽ in hình kết quả với vùng chọn, nếu tỉ lệ dự đoán >= threshold(0,5)

```
def crop_save(img_path,row,end_column):
  img = Image.open(img_path)
  img = img.crop((row[0],column[0],row[1],column[1]))
  img.save('crop.jpg')
```

Hàm cắt vùng chọn từ kết quả thu được và lưu lại hình ảnh với tên: ‘crop.jpg’

```
def canny_filter(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    edge = canny(gray)
    edge = edge.astype(np.int16)
    edge = np.repeat(edge[..., np.newaxis], 3, -1)
    return edge
```

Hàm làm nổi các vùng có điểm góc, cạnh trong ảnh

```python
class FeatureExtractor:
    def __init__(self):
        # Use MobileNet V2 as the architecture and ImageNet for the weight
        self.model = models.mobilenet_v3_small(pretrained=True)
        # Remove the last layer (output layer)
        self.model.classifier = nn.Sequential(*list(self.model.classifier.children())[:-1])
        self.model.eval()
        # Define image transforms
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])

    def extract(self, img):
        # Convert the image color space
        img = cv2.imread(img)
        img = canny_filter(img)
        # Apply image transforms
        img = torch.FloatTensor(img).permute(2, 1, 0).unsqueeze(0)
        # Add batch dimension
        # Extract Features
        device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
        net = self.model.to(device)
        img = img.to(device)
        with torch.no_grad():
            feature = net(img).flatten()
            feature = feature.to('cpu')
            feature = feature.numpy()
        img = img.detach()
        torch.cuda.empty_cache()
        return feature / np.linalg.norm(feature)
```

Lớp chứa hàm rút trích đặc trưng ảnh, ảnh đầu vào lúc này đã đi qua lọc làm nổi các vùng có điểm góc, cạnh và mô hình được sử dụng để rút trích là MobileNetV3(small)

```python
extract_feat = FeatureExtractor()
def ReadData(path):
  features_dict = {}
  for sector in os.listdir(path):
    path_sector = os.path.join(path, sector)
    # if os.path.exists('/content/drive/MyDrive/features/'+ sector + '.pkl') is False:
    tmp = tqdm(os.listdir(path_sector))
    for brand in tmp:
      path_brand = os.path.join(path_sector, brand)
      tmp.set_description(path_brand)
      for filename in os.listdir(path_brand):
        direct = os.path.join(path_brand, filename)
        feature = extract_feat.extract(direct)
        features_dict[direct] = feature
      # with open('drive/MyDrive/features/'+ sector + '.pkl',"wb") as file:
      #   pickle.dump(features_dict,file)

  return features_dict
```

Hàm đọc bộ dữ liệu và tiến hành rút trích đặc trưng

```
path = ("/content/drive/MyDrive/dataset/datasetcopy/trainandtest/train")
features = ReadData(path)
```

path = (‘*dataset_path’*)

dataset_path: đường dẫn đến thư mục dataset **(./data_extract)**

```
#Cluster the features
n_clusters=20

kmeans = KMeans(n_clusters = n_clusters, random_state=0)
kmeans.fit(list(features.values()))
labels = kmeans.labels_
centers = kmeans.cluster_centers_
label = {}
label["features"] = features
label["label"] = labels
label["centers"] = centers
```

Thuật toán KMeans để gom cụm các ảnh trong bộ dữ liệu có vector đặc trưng giống nhau và lấy nhãn.

```
#Load vector features labeled by KMeans to search
if os.path.exists('drive/MyDrive/Project/new_clusters.pkl') is True:
  with open("/content/drive/MyDrive/Project/new_clusters.pkl","rb") as file:
    label = pickle.load(file)
else:
#Save first time running system not have file .pkl
  with open("drive/MyDrive/Project/new_clusters.pkl","wb") as file:
    pickle.dump(label,file)
```

Để tiết kiệm thời gian, trọng số các vector đặc trưng đã được gom cụm khi qua thuật toán KMeans sẽ được lưu sẵn tại file ‘new_cluster.pkl’, chỉ cần chạy để load lại các vector đã lưu. Nếu như chạy hệ thống từ ban đầu, chưa có file lưu sẽ lưu lại sau khi qua thuật toán KMeans

```
input_dir = "/content/crop.jpg"
top_k = 20

input_fea = extract_feat.extract(input_dir)
cos = torch.nn.CosineSimilarity(dim=0, eps=1e-6)
#Predict the label of input
max = 0
for x in range(len(label["centers"])):
  output = cos(torch.Tensor(label["centers"][x]), torch.Tensor(input_fea))
  output = float(output)
  if output > max:
    predict_label = x
    max = output
#Take all path in label to compare
indices = [i for i, x in enumerate(label["label"]) if x == predict_label]
feature_key = list(label["features"].keys())
result = {}
for i in indices:
  vector_fea = label["features"][feature_key[i]]
  output = cos(torch.Tensor(vector_fea), torch.Tensor(input_fea))
  output = output.detach() #for gpu
  result[feature_key[i]] = float(output)

#Sort similarity and take k_top
result = dict(sorted(result.items(), key=lambda item: item[1], reverse = True))
result = {list(result.keys())[x]: list(result.values())[x] for x in range(top_k)}
```

- Nhập đường dẫn đến ảnh đã cắt sau khi qua phần detect và nhập top_k ảnh giống với ảnh đầu vào.
- Sau đó tiến hành dự đoán nhãn cho ảnh đầu vào bằng cách so sanh trọng số similarity giữa ảnh đầu với các vector đặc trưng trung tâm của từng nhãn để xác định nhãn cho ảnh đầu vào và tiến hành so sánh độ giống nhau của ảnh với các ảnh trong bộ dữ liệu có cùng nhãn bằng thuật toán CosineSimilarity của thư viên torch
- Khi đã thu được độ giống nhau giữa ảnh đầu vào với các ảnh trong bộ dữ liệu, sắp xếp kết quả thu được từ cao đến thấp và tiến hành lấy top_k ảnh

```
if os.path.isdir('drive/MyDrive/Result_Img') is True and len(os.listdir('drive/MyDrive/Result_Img')) != 0:
  for filename in os.listdir("drive/MyDrive/Result_Img"):
    file = os.path.join("drive/MyDrive/Result_Img", filename)
    os.remove(file)
elif(os.path.isdir('drive/MyDrive/Result_Img') is False):
  os.mkdir('drive/MyDrive/Result_Img')
for x in result:
  print(x+": ", result[x])
  img = Image.open(x)
  tmp = x.split('/')[-2] + '_' + x.split('/')[-1]
  res_path = 'drive/MyDrive/Result_Img/' + tmp
  img.save(res_path)
```

- Kiểm tra xem thư mục kết quả đã tồn tại hay chưa,và tạo nếu chưa có hoặc làm trống thư mục nếu thư mục không trống.
- Cuối cùng, in ra độ giống nhau đối với các ảnh trong bộ dữ liệu và lưu ảnh trong thư mục kết quả.s