

به نام او



دانشگاه صنعتی شریف

دانشکده مهندسی برق

یادگیری عمیق

گزارش پروژه

دکتر فاطمی زاده

مهریار جعفری نوده - ۹۸۱۰۱۳۵۲

علیرضا سخایی راد - ۹۸۱۰۱۷۱۴

۹۸۱۰۱۳۵۲

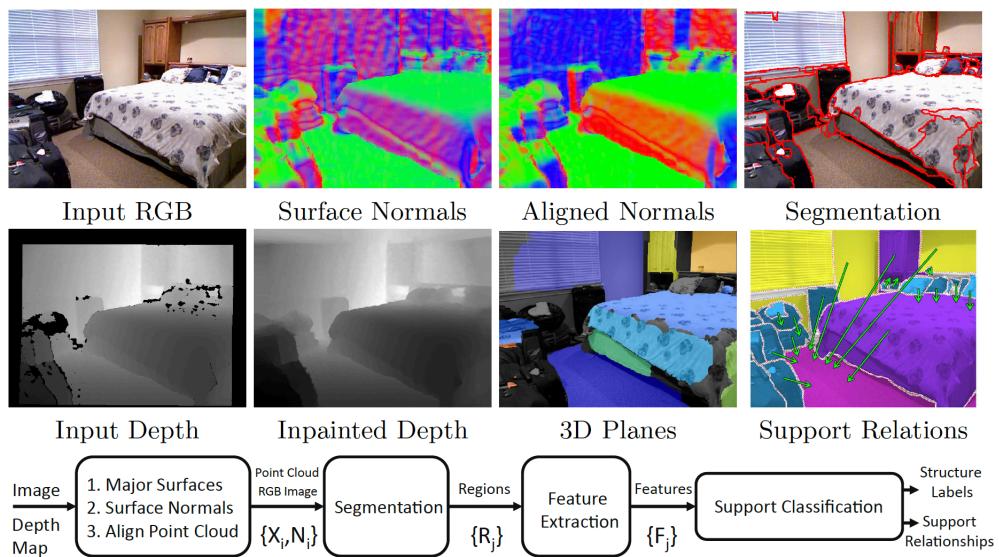
۱۵ بهمن ۱۴۰۰

فهرست مطالب

۱	۱ تحلیل مقاله دیتاست
۴	۲ تحلیل مقاله تشخیص شی و عمق همزمان
۶	۳ شبکه تشخیص اشیا
۶	۱.۳ ساختار و پیاده سازی
۸	۲.۳ مثال ها
۹	۳.۳ کدها
۹	۱.۳.۳ طراحی شبکه
۱۴	۲.۳.۳ استخراج باندینگ باکس ها
۱۶	۳.۳.۳ حذف اشتراک باندینگ باکس ها
۱۸	۴ شبکه تشخیص عمق
۱۸	۱.۴ ساختار کلی شبکه و بلوک های استفاده شده
۲۱	۲.۴ تشکیل مدل و تعیینتابع هزینه
۲۱	۱.۲.۴ اجزای مدل
۲۱	۲.۲.۴ تابع هزینه
۲۴	۳.۴ پایپ لاین دیتا
۲۶	۴.۴ نگاهی به دیتا
۲۷	۵.۴ ترین
۲۸	۶.۴ نمونه ای از نتایج
۲۹	۵ ترکیب مدل ها
۲۹	۱.۵ مثال ها

۱ تحلیل مقاله دیتاست

در ابتدا دربره دادگام مسیله صحبت میکنیم که شامل تعدادی عکس است که شامل اجект های مختلف میباشد که در فیلد های مختلف ورودی ، اطلاعاتی اعم از کلاس ها ، عمق های پیکسل ها و دادگان دیگری همچون اسم کلاس ها و غیره اورده شده است. همانطور که در مقاله دادگان مشاهده میشود ساختار زیر برای تشخیص عمق آنها و ماسک گذاری و یا سگمنتیشن برای عکس ها صورت گرفته است .



در ادامه برای سگمنت بندی عکس و برای تعیین تخصیص پیکسل ها به سطوح مختلف رابطه زیر تعریف شده و هدف مینیمایز کردن تابع هدف زیر است :

$$E(\mathbf{data}, \mathbf{y}) = \alpha_i \left[\sum_i f_{3d}(\mathbf{X}_i, y_i) + f_{norm}(\mathbf{N}_i, y_i) \right] + \sum_{i,j \in \mathcal{N}_8} f_{pair}(y_i, y_j, \mathbf{I}) \quad (2)$$

در نهایت همانطور که در ابتدا گفته شد فیلدهای مختلف ورودی به شکل زیر هستند.

- **accelData** – Nx4 matrix of accelerometer values indicated when each frame was taken. The columns contain the roll, yaw, pitch and tilt angle of the device.
- **depths** – HxWxN matrix of in-painted depth maps where H and W are the height and width, respectively and N is the number of images. The values of the depth elements are in meters.
- **images** – HxWx3xN matrix of RGB images where H and W are the height and width, respectively, and N is the number of images.
- **instances** – HxWxN matrix of instance maps. Use `get_instance_masks.m` in the Toolbox to recover masks for each object instance in a scene.
- **labels** – HxWxN matrix of object label masks where H and W are the height and width, respectively and N is the number of images. The labels range from 1..C where C is the total number of classes. If a pixel's label value is 0, then that pixel is 'unlabeled'.
- **names** – Cx1 cell array of the english names of each class.
- **namesToids** – map from english label names to class IDs (with C key-value pairs)
- **rawDepths** – HxWxN matrix of raw depth maps where H and W are the height and width, respectively, and N is the number of images. These depth maps capture the depth images after they have been projected onto the RGB image plane but before the missing depth values have been filled in. Additionally, the depth non-linearity from the Kinect device has been removed and the values of each depth image are in meters.
- **rawDepthFilenames** – Nx1 cell array of the filenames (in the Raw dataset) that were used for each of the depth images in the labeled dataset.
- **rawRgbFilenames** – Nx1 cell array of the filenames (in the Raw dataset) that were used for each of the RGB images in the labeled dataset.
- **scenes** – Nx1 cell array of the name of the scene from which each image was taken.
- **sceneTypes** – Nx1 cell array of the scene type from which each image was taken.

در ادامه کار ورودی هایی از قبیل عکس های رنگی، عمقها، کلاس ها و اسم کلاس ها را بعنوان اطلاعات مفید از روی ورودی بر میداریم و برای استفاده های بعدی ذخیره می کنیم.

۲ تحلیل مقاله تشخیص شی و عمق همزمان

در این مقاله ساختار زیر در ابتدای کار پیشنهاد شده است :

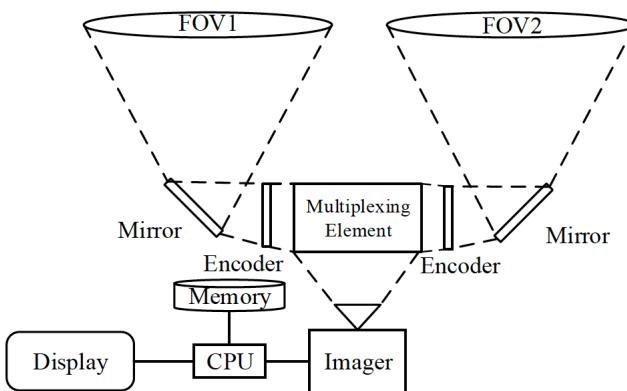


FIGURE 1. The architecture of the device for multiplexed imaging [1].

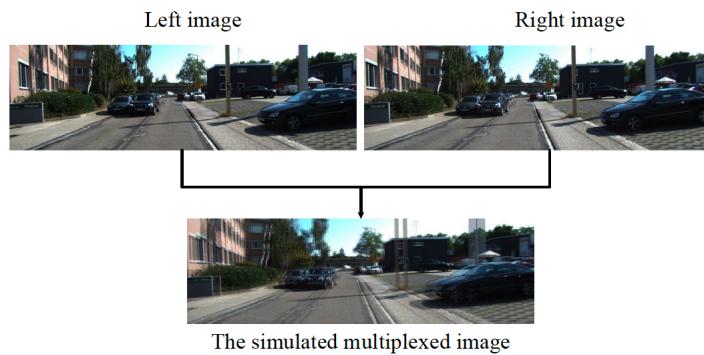
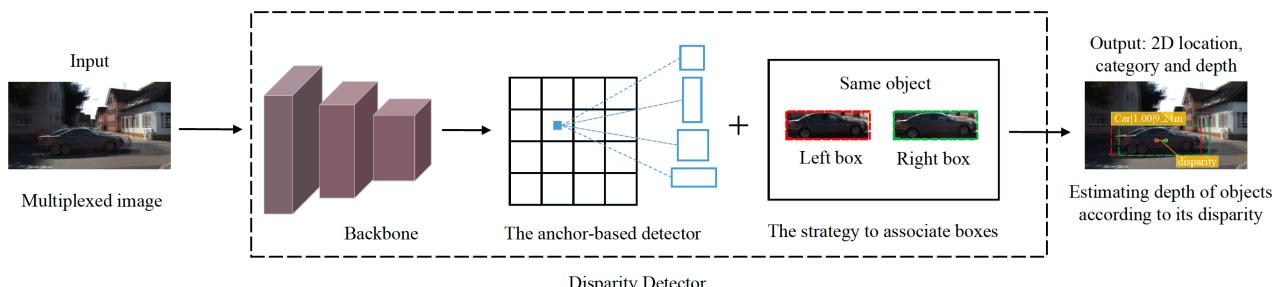


FIGURE 2. A stereo image (the image I_l from the left camera and the image I_r from the right camera) and the multiplexed image. In this paper, we use overlapped image $I = (I_l + I_r)/2$ to simulate the multiplexed image.

که گویی اطلاعات همزمان دو چشم را گرفته و ترکیبی از این دو توصیر را بصورت میانگین پیکس های آنها تشکیل میدهد .

در ادامه شبکه ساختار کلی زیر را پیشنهاد میدهد که متشکل از دو شبکه تشخیص عمق و شی در درون خود است که ما هر یک را جداگانه تشکیل خواهیم داد :



سپس شبکه جزئی تر میشود و ساختار کلی تکه‌ی تشخیص شی را شرح میدهد که در بخش های بعد ما آن را با استفاده از مارژول yolov3 پیاده سازی خواهیم کرد .

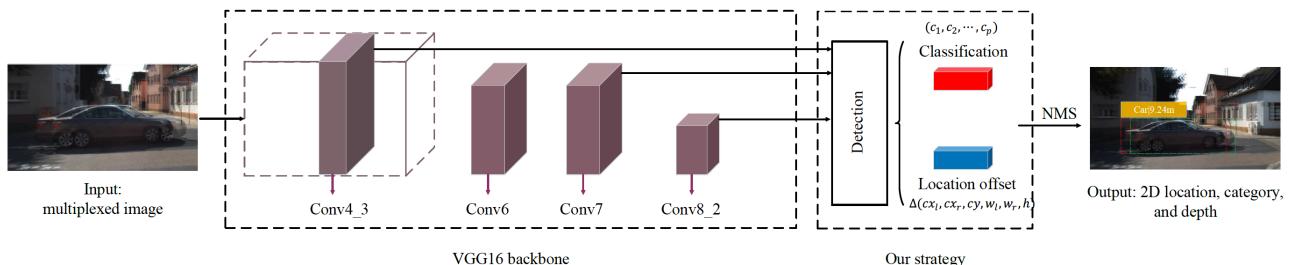


FIGURE 4. The detailed network architecture of the proposed Disparity Detector.

نهایتا شبکه پس از تشکیل دو مدل مختلف بیان میکند که پس از تشکیل باندینگ باکس ها برای تشخیص اشیا درون تصویر برای تشخیص عمق آنها با استفاده از شبکه تشخیص عمق پیکسل های عکس را پیدا کرده و با استفاده از فرمول های زیر میانگین یا مدین پیکسل هایی که درون باندینگ باکس قرار میگیرند را محاسبه میکیم

$$disp_{mean}^{object} = \frac{1}{0.4w \times 0.4h} \times \sum_{cx=0.2w}^{cx+0.2w} \sum_{cy=-0.2h}^{cy+0.2h} disp(i, j) \quad (3)$$

and

$$disp_{median}^{object} = median\{disp(i, j) | (i, j) \in BB\} \quad (4)$$

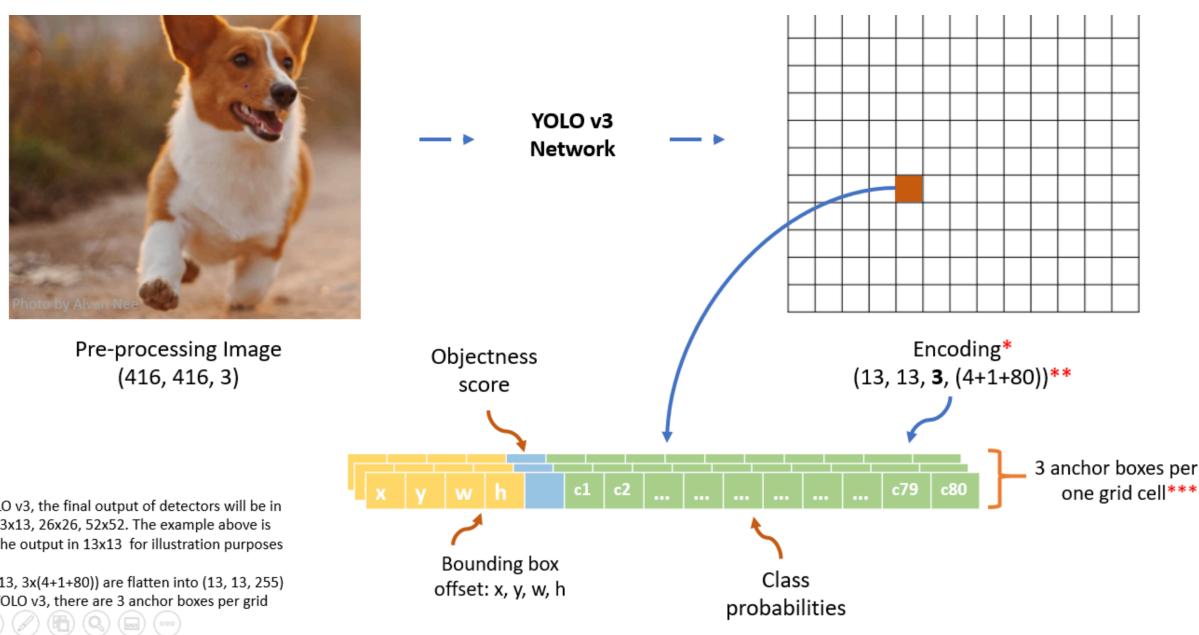
نکته مهم این است که در ساختار شبکه میانگین روی تمامی باندینگ باکس گرفته نمیشود بلکه روی قسمت های مرکزی تر گرفته میشود که حدود نصف کل باندیگ باکس را تشکیل میدهد.

۳ شبکه تشخیص اشیا

۱.۳ ساختار و پیاده سازی

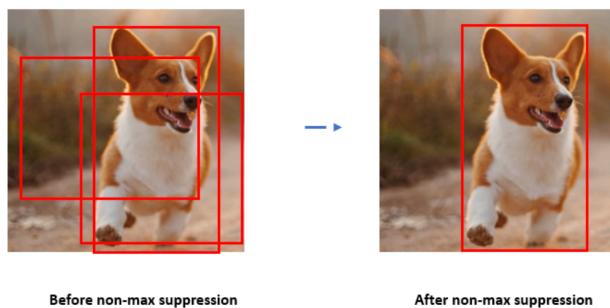
در این بخش از شبکه *yoloV3* استفاده شده است که ابتدا توضیح کوتاهی از مقاله میدهیم و پیاده سازی را شرح میدهیم. در این ساختار شبکه عمیقی بصورت زیر داریم : که در ۳ جا از این شبکه خروجی میگیریم گه به ترتیب 13×13 ، 26×26 ، 52×52 میباشدند ، هر یک از این گرید ها به ازای هر پیکسل خود اطلاعاتی مانند زیر دارند که مثالی برای گرید با کمترین سایز است :

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	32	1×1	
	64	3×3	128×128
	Residual		
2x	128	$3 \times 3 / 2$	64×64
	64	1×1	
	128	3×3	
8x	Residual		64×64
	256	$3 \times 3 / 2$	32×32
	128	1×1	
8x	256	3×3	
	Residual		32×32
	512	$3 \times 3 / 2$	16×16
8x	256	1×1	
	512	3×3	
	Residual		16×16
4x	Convolutional	$3 \times 3 / 2$	8×8
	512	1×1	
	1024	$3 \times 3 / 2$	
4x	Convolutional	1×1	
	1024	3×3	
	Residual		8×8
Avgpool		Global	
Connected		1000	
Softmax			



مشاهده میکنید هر پیکسل ۳ بردار ویژگی یا باندینگ باکس دارد که برای هر باندینگ باکس ۸۰ اندیس برای احتمال کلاس ها، یک اندیس برای احتمال شی بودن و چهار اندیس برای نشان دادن مکان باندینگ باکس وجود دارد، که البته این ساختار درجا از خروجی شبکه دارک نت خارج نمیشود و تکه کد ۲.۳.۳ ساختاری برای استخراج این بردارها را نمایش میدهد، و همچنین ۱.۳.۲ ساختار شبکه ساخته شده را نشان میدهد.

و در نهایت از باندینگ باکس هایی که با هم اورلپ دارند تنها یکی را برミداریم : که با تکه کد ۳.۳.۳ انعام شده

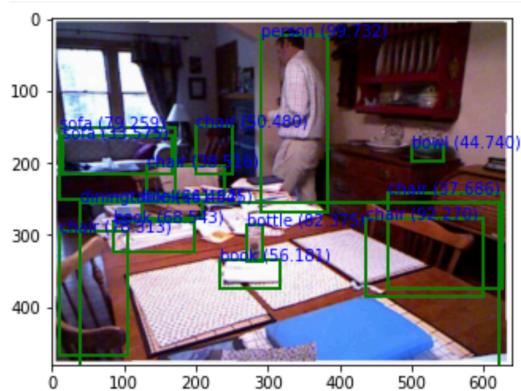


است..

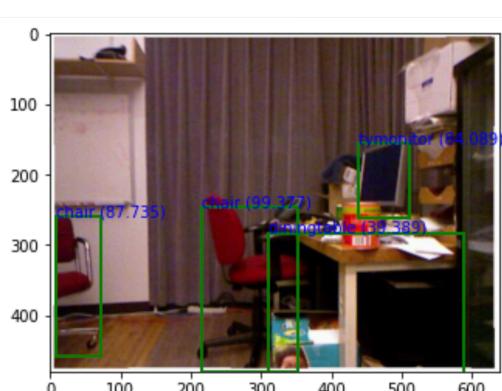
نتیجه نهایی برای عکس شامل باندینگ باکسها، اسم کلاس ها و در صر اطمینان از آن کلاس هاست، دقیق شود که شبکه را مجددا روی همه ۱۰۰۰ کلاس موجود روی دیتاست ترین نکردیم و وزن های ترین شده روی دیتاست های دیفالت خودش را روی آن لود کردیم تا اشیای غیربدیهی دیتکت شوند نه چیزهایی مثل دیوار، زمین، و غیره .

در ادامه مثال هایی از عکس های دیتکت شده توسط شبکه و کلاس های آنها را مشاهده میکنید، همچنین چون در دیتاست اصلی فیلدی که اندیس را به اسم ربط بددهد خالی گذاشته شده بود ، در نهایت برای محاسبه فرمول خطای را روی عمق ها در نظر میگیریم و برای بخش دیتکشن هم تعداد دیتکشن درست روی اشیای اصلی در عکس را بعنوان یک ضریب در خطای عمق در نظر میگیریم.

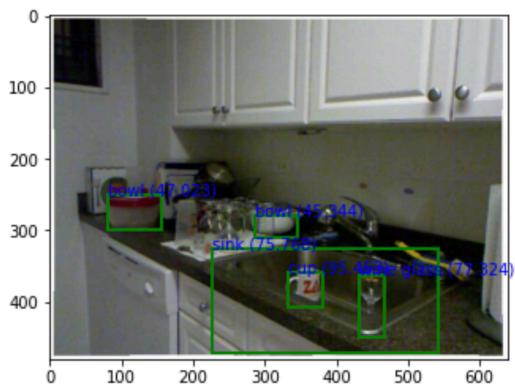
مثال ها ۲.۳



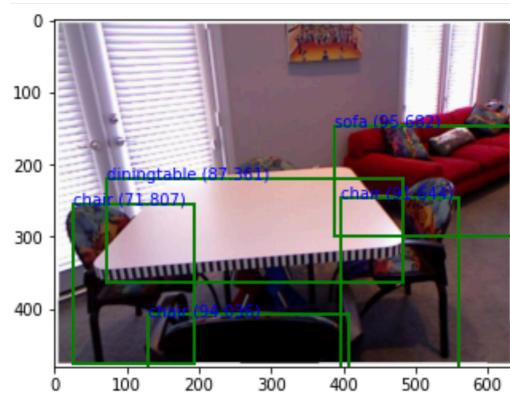
(ب) مثال دو



(آ) مثال یک



(د) مثال ۴



(ج) مثال سه

۳.۳ کدها

۱.۳.۳ طراحی شبکه

```

1 def _conv_block(inp, convs, skip=True):
2     x = inp
3     count = 0
4     for conv in convs:
5         if count == (len(convs) - 2) and skip:
6             skip_connection = x
7         count += 1
8         if conv['stride'] > 1: x = ZeroPadding2D(((1,0),(1,0)))(x)
9             # peculiar padding as darknet prefer left and top
10            x = Conv2D(conv['filter'],
11                        conv['kernel'],
12                        strides=conv['stride'],
13                        padding='valid' if conv['stride'] > 1 else 'same'
14                        ', # peculiar padding as darknet prefer left
15                        and top
16                        name='conv_' + str(conv['layer_idx']),
17                        use_bias=False if conv['bnorm'] else True)(x)
18
19 if conv['bnorm']: x = BatchNormalization(epsilon=0.001,
20                                         name='bnorm_' + str(conv['layer_idx']))(x)
21
22 if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' +
23                                 str(conv['layer_idx']))(x)
24
25 return add([skip_connection, x]) if skip else x

```

```

19 def make_yolov3_model():
20     input_image = Input(shape=(None, None, 3))
21     # Layer 0 => 4
22     x = _conv_block(input_image, [{ 'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 0 },
23                               { 'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1 },
24                               { 'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2 },
25                               { 'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3 }

```

```

3}])

# Layer 5 => 8
x = _conv_block(x, [{"filter": 128, "kernel": 3, "stride": 2, "bnorm": True, "leaky": True, "layer_idx": 5},
                     {"filter": 64, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 6},
                     {"filter": 128, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 7}])

# Layer 9 => 11
x = _conv_block(x, [{"filter": 64, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 9},
                     {"filter": 128, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 10}])

# Layer 12 => 15
x = _conv_block(x, [{"filter": 256, "kernel": 3, "stride": 2, "bnorm": True, "leaky": True, "layer_idx": 12},
                     {"filter": 128, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 13},
                     {"filter": 256, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 14}])

# Layer 16 => 36
for i in range(7):
    x = _conv_block(x, [{"filter": 128, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 16+i*3},
                         {"filter": 256, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 17+i*3}])

skip_36 = x

# Layer 37 => 40
x = _conv_block(x, [{"filter": 512, "kernel": 3, "stride": 2, "bnorm": True, "leaky": True, "layer_idx": 37},
                     {"filter": 256, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 38},
                     {"filter": 512, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 39}])

```

```

46     # Layer 41 => 61
47     for i in range(7):
48         x = _conv_block(x, [{"filter": 256, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 41+i*3}, {"filter": 512, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 42+i*3}])
49
50     skip_61 = x
51     # Layer 62 => 65
52     x = _conv_block(x, [{"filter": 1024, "kernel": 3, "stride": 2, "bnorm": True, "leaky": True, "layer_idx": 62}, {"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 63}, {"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 64}])
53
54     # Layer 66 => 74
55     for i in range(3):
56         x = _conv_block(x, [{"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 66+i*3}, {"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 67+i*3}])
57
58     # Layer 75 => 79
59     x = _conv_block(x, [{"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 75}, {"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 76}, {"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 77}, {"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 78}, {"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 79}], skip=False)
60
61     # Layer 80 => 82
62     yolo_82 = _conv_block(x, [{"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 80}],
63
64
65
66

```

```

67                                     {'filter': 255, 'kernel': 1, 'stride':
68                                         ': 1, 'bnorm': False, 'leaky':
69                                         False, 'layer_idx': 81}], skip=
70                                         False)

71 # Layer 83 => 86
72 x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1,
73                         'bnorm': True, 'leaky': True, 'layer_idx': 84}], skip=False)
74 x = UpSampling2D(2)(x)
75 x = concatenate([x, skip_61])
76 # Layer 87 => 91
77 x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1,
78                         'bnorm': True, 'leaky': True, 'layer_idx': 87},
79                             {'filter': 512, 'kernel': 3, 'stride': 1,
80                             'bnorm': True, 'leaky': True, 'layer_idx':
81                             : 88},
82                             {'filter': 256, 'kernel': 1, 'stride': 1,
83                             'bnorm': True, 'leaky': True, 'layer_idx':
84                             : 89},
85                             {'filter': 512, 'kernel': 3, 'stride': 1,
86                             'bnorm': True, 'leaky': True, 'layer_idx':
87                             : 90},
88                             {'filter': 256, 'kernel': 1, 'stride': 1,
89                             'bnorm': True, 'leaky': True, 'layer_idx':
90                             : 91}], skip=False)

91 # Layer 92 => 94
92 yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride':
93                         : 1, 'bnorm': True, 'leaky': True, 'layer_idx': 92},
94                             {'filter': 255, 'kernel': 1, 'stride':
95                         : 1, 'bnorm': False, 'leaky':
96                         False, 'layer_idx': 93}], skip=
97                         False)

98 # Layer 95 => 98
99 x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1,
100                         'bnorm': True, 'leaky': True, 'layer_idx': 96}], skip=False)
101 )
102 x = UpSampling2D(2)(x)
103 x = concatenate([x, skip_36])
104 # Layer 99 => 106
105 yolo_106 = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride':
106                         : 1, 'bnorm': True, 'leaky': True, 'layer_idx': 99},
107                             {'filter': 256, 'kernel': 3, 'stride':
108                         : 1, 'bnorm': True, 'leaky':
```

```

    True, 'layer_idx': 100},
88  {'filter': 128, 'kernel': 1, 'stride':
     ': 1, 'bnorm': True, 'leaky':
      True, 'layer_idx': 101},
89  {'filter': 256, 'kernel': 3, 'stride':
     ': 1, 'bnorm': True, 'leaky':
      True, 'layer_idx': 102},
90  {'filter': 128, 'kernel': 1, 'stride':
     ': 1, 'bnorm': True, 'leaky':
      True, 'layer_idx': 103},
91  {'filter': 256, 'kernel': 3, 'stride':
     ': 1, 'bnorm': True, 'leaky':
      True, 'layer_idx': 104},
92  {'filter': 255, 'kernel': 1, 'stride':
     ': 1, 'bnorm': False, 'leaky':
      False, 'layer_idx': 105}], skip=
      False)

93 model = Model(input_image, [yolo_82, yolo_94, yolo_106])
94 return model

95
96
97
98 yolov3 = make_yolov3_model()
99
100 # load the weights
101 weight_reader = WeightReader('/content/drive/My Drive/yolov3.
      weights')
102
103 # set the weights
104 weight_reader.load_weights(yolov3)
105
106 # save the model to file
107 yolov3.save('model.h5')

```

۲.۳.۳ استخراج باندینگ باکس ها

```

1 class BoundBox:
2     def __init__(self, xmin, ymin, xmax, ymax, objness=None,
3                  classes=None):
4         self.xmin = xmin
5         self.ymin = ymin
6         self.xmax = xmax
7         selfymax = ymax
8         self.objness = objness
9         self.classes = classes
10        self.label = -1
11        self.score = -1
12
13    def get_label(self):
14        if self.label == -1:
15            self.label = np.argmax(self.classes)
16
17    return self.label
18
19    def get_score(self):
20        if self.score == -1:
21            self.score = self.classes[self.get_label()]
22
23    return self.get_score
24
25    def _sigmoid(x):
26        return 1. / (1. + np.exp(-x))
27
28    def decode_netout(netout, anchors, obj_thresh, net_h, net_w):
29        grid_h, grid_w = netout.shape[:2]
30        nb_box = 3
31        netout = netout.reshape((grid_h, grid_w, nb_box, -1))
32        nb_class = netout.shape[-1] - 5
33        boxes = []
34        netout[..., :2] = _sigmoid(netout[..., :2])
35        netout[..., 4:] = _sigmoid(netout[..., 4:])
36        netout[..., 5:] = netout[..., 4][..., np.newaxis] * netout[...,
37                                         5:]
38        netout[..., 5:] *= netout[..., 5:] > obj_thresh

```

```

38
39     for i in range(grid_h * grid_w):
40         row = i / grid_w
41         col = i % grid_w
42         for b in range(nb_box):
43             # 4th element is objectness score
44             objectness = netout[int(row)][int(col)][b][4]
45             if(objectness.all() <= obj_thresh):
46                 continue
47             # first 4 elements are x, y, w, and h
48             x, y, w, h = netout[int(row)][int(col)][b][:4]
49             x = (col + x) / grid_w # center position, unit: image
50                         width
51             y = (row + y) / grid_h # center position, unit: image
52                         height
53             w = anchors[2 * b + 0] * np.exp(w) / net_w # unit:
54                         image width
55             h = anchors[2 * b + 1] * np.exp(h) / net_h # unit:
56                         image height
57             # last elements are class probabilities
58             classes = netout[int(row)][col][b][5:]
59             box = BoundBox(x - w / 2, y - h / 2, x + w / 2,
60                             y + h / 2, objectness, classes)
61             boxes.append(box)
62
63     return boxes

```

۳.۳.۳ حذف اشتراک باندینگ باکس ها

```

1 def _interval_overlap(interval_a, interval_b):
2     x1, x2 = interval_a
3     x3, x4 = interval_b
4     if x3 < x1:
5         if x4 < x1:
6             return 0
7         else:
8             return min(x2, x4) - x1
9     else:
10        if x2 < x3:
11            return 0
12        else:
13            return min(x2, x4) - x3
14
15
16 def bbox_iou(box1, box2):
17     intersect_w = _interval_overlap(
18         [box1.xmin, box1xmax], [box2.xmin, box2xmax])
19     intersect_h = _interval_overlap(
20         [box1.ymin, box1ymax], [box2.ymin, box2ymax])
21     intersect = intersect_w * intersect_h
22     w1, h1 = box1xmax - box1xmin, box1ymax - box1ymin
23     w2, h2 = box2xmax - box2xmin, box2ymax - box2ymin
24     union = w1 * h1 + w2 * h2 - intersect
25     return float(intersect) / union
26
27
28 def do_nms(boxes, nms_thresh):
29     if len(boxes) > 0:
30         nb_class = len(boxes[0].classes)
31     else:
32         return
33     for c in range(nb_class):
34         sorted_indices = np.argsort([-box.classes[c] for box in
35             boxes])
36         for i in range(len(sorted_indices)):
37             index_i = sorted_indices[i]
38             if boxes[index_i].classes[c] == 0:
39                 continue

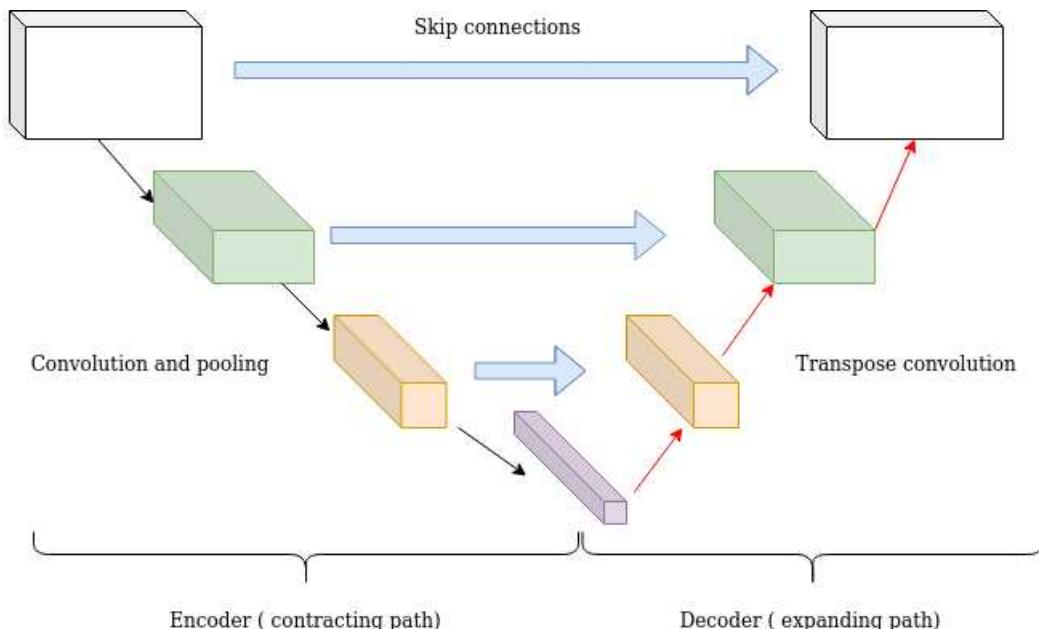
```

```
39         for j in range(i + 1, len(sorted_indices)):  
40             index_j = sorted_indices[j]  
41             if bbox_iou(boxes[index_i], boxes[index_j]) >=  
42                 nms_thresh:  
                         boxes[index_j].classes[c] = 0
```

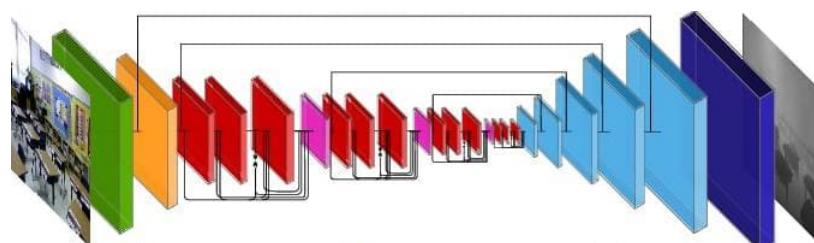
۴ شبکه تشخیص عمق

۱.۴ ساختار کلی شبکه و بلوک‌های استفاده شده

ورودی و خروجی این شبکه هر دو به صورت تصویر می‌باشند، پس نیاز است که ابتدا در ورودی فیچر اکسترکشن انعام شود و سپس به کمک لایه‌های Deconvolution و یا Upsample کد ایجاد شده به یک عکس تبدیل شود. برای اینکه اطلاعات موجود در رزولوشن‌های مختلف از دست نرود، از اسکیپ کانکشن‌های متقابن استفاده شده است و ساختار کلی شبکه به صورت زیر خواهد بود:



در ساختار استفاده شده سه نوع لایه مختلف استفاده شده است که کد هریک جلوتر آورده می‌شود. لایه‌های مربوط به انکودر، لایه bottleneck و لایه‌های مربوط به دیکودر که تصویر کد شده را تبدیل به depth map می‌کنند. از هر کدام از لایه‌های انکودر و دیکودر ۴ تا پشت سر هم استفاده شده است (که تعداد فیلترهای مورداستفاده در هر یک به صورت قائمی از دو است که ترتیب‌های آن در کد آمده است). خروجی هر یک از ۴ قسمت انکودر به خروجی هر کدام از قسمت‌های دیکودر با یک skip connection متصل است و این اتصال به صورت concatenation می‌باشد. این گونه فلوی اطلاعات بهتر انعام می‌شود و دیتا‌بی به علت طولانی شدن شبکه از دست نمی‌رود. ساختار با جزئیات بیشتر به شکل زیر می‌باشد:



در قسمت‌های انکودر از لایه‌های کانولوشنی با کرنل به اندازه ۳ و همچنین بچ نرمالیزیشن و پولینگ استفاده شده است. همچنین تابع فعالساز مورداستفاده تابع ReLU می‌باشد. در لایه bottleneck از لایه‌های کانولوشنی با کرنل ۳ در ۳ و پدینگ مشابه استفاده شده است و تابع فعالساز Leaky ReLU با پارامتر دو دهم می‌باشد. در قسمت دیکودر نیز از لایه‌های کانولوشنی با کرنل‌های ۳ در ۳ و پدینگ مشابه، و همچنین بچ نرمالیزیشن استفاده شده است. در انتهای هر بلوک نیز یک لایه برای انعام عمل concatenation قرار داده می‌شود که به بلوک متناظر از انکودر متصل است.

کدهای مربوط به هر کدام از بلوک به صورت زیر است:

```

1
2 class DownscaleBlock(layers.Layer):
3
4     def __init__(self, filters, kernel_size=(3, 3), padding="same", strides=1, **kwargs):
5
6         super().__init__(**kwargs)
7         self.convA = layers.Conv2D(filters, kernel_size, strides, padding)
8         self.convB = layers.Conv2D(filters, kernel_size, strides, padding)
9         self.reluA = layers.LeakyReLU(alpha=0.2)
10        self.reluB = layers.LeakyReLU(alpha=0.2)
11        self.bn2a = tf.keras.layers.BatchNormalization()
12        self.bn2b = tf.keras.layers.BatchNormalization()
13
14        self.pool = layers.MaxPool2D((2, 2), (2, 2))
15
16    def call(self, input_tensor):
17        d = self.convA(input_tensor)
18        x = self.bn2a(d)
19        x = self.reluA(x)
20
21        x = self.convB(x)
22        x = self.bn2b(x)
23        x = self.reluB(x)
24
25        x += d
26        p = self.pool(x)
27        return x, p
28
29
30
31 class UpscaleBlock(layers.Layer):
32
33     def __init__(self, filters, kernel_size=(3, 3), padding="same", strides=1, **kwargs):
34
35         super().__init__(**kwargs)
36

```

```

37         self.us = layers.UpSampling2D((2, 2))
38         self.convA = layers.Conv2D(filters, kernel_size, strides,
39                                     padding)
40         self.convB = layers.Conv2D(filters, kernel_size, strides,
41                                     padding)
42         self.reluA = layers.LeakyReLU(alpha=0.2)
43         self.reluB = layers.LeakyReLU(alpha=0.2)
44         self.bn2a = tf.keras.layers.BatchNormalization()
45         self.bn2b = tf.keras.layers.BatchNormalization()
46         self.conc = layers.concatenate()
47
48     def call(self, x, skip):
49         x = self.us(x)
50         concat = self.conc([x, skip])
51         x = self.convA(concat)
52         x = self.bn2a(x)
53         x = self.reluA(x)
54
55         x = self.convB(x)
56         x = self.bn2b(x)
57         x = self.reluB(x)
58
59
60     class BottleneckBlock(layers.Layer):
61
62         def __init__(self, filters, kernel_size=(3, 3), padding="same", strides=1, **kwargs):
63             super().__init__(**kwargs)
64             self.convA = layers.Conv2D(filters, kernel_size, strides,
65                                     padding)
66             self.convB = layers.Conv2D(filters, kernel_size, strides,
67                                     padding)
68             self.reluA = layers.LeakyReLU(alpha=0.2)
69             self.reluB = layers.LeakyReLU(alpha=0.2)
70
71         def call(self, x):
72             x = self.convA(x)
73             x = self.reluA(x)

```

```

74     x = self.convB(x)
75     x = self.reluB(x)
76     return x

```

۲.۴ تشكیل مدل و تعیین تابع هزینه

۱.۲.۴ اجزای مدل

همانطور که توضیح داده شد، برای تشكیل مدل از ۴ لایه داون‌سمپل، یک لایه باتل‌نک و ۴ لایه آپ‌سمپل استفاده شده است که تعداد فیلترها در هر بلوک از انکودر و دیکودر متقارن است و جلوتر در کد آورده می‌شود.

۲.۲.۴ تابع هزینه

برای انعام ترین، سه هزینه مختلف درنظر گرفته شده است که کد مربوط به هریک در تابع محاسبه هزینه در کلاسی که مدل ما یک شی از آن است، آورده شده است.
 ترم اول تابع هزینه، ضریب شباهت ساختاری (SSIM) می‌باشد که مهم‌ترین هزینه برای بهبود عملکرد مدل می‌باشد. برای محاسبه آن از یکی از توابع آماده کتابخانه تنسورفلو برای پردازش تصویر استفاده شده است.
 دومین ترم، یک ترم L1 است که میانگین اختلاف پیکسل به پیکسل خروجی واقعی و خروجی مدل است. در اینجا نیز مجدداً از توابع تنسورفلو برای میانگین‌گیری استفاده می‌شود.
 با توجه به نوع خروجی، یک تابع هزینه برای افزایش smoothness تصویر خروجی قرار داده شده است. وزن‌های هرکدام از این هزینه‌ها به شرح زیر هستند:

```

self.ssim_loss_weight = 0.85
self.l1_loss_weight = 0.1
self.edge_loss_weight = 0.9

```

برای اینکه بتوانیم این مدل را به خوبی پیاده سازی کنیم، نیاز است که یک کلاس بسازیم که از بخش مدل کراس ارث ببرد و سپس ویژگی‌های موردنظرمان (تابع هزینه و ساختار و اسکیپ کانکشن‌ها) را در آن پیاده‌سازی کنیم. کد زیر، کلاسی است که مدل از آن ارث خواهد برد:

```

1 class DepthEstimationModel(tf.keras.Model):
2
3     def __init__(self):
4         super().__init__()
5         self.ssim_loss_weight = 0.85
6         self.l1_loss_weight = 0.1
7         self.edge_loss_weight = 0.9
8         self.loss_metric = tf.keras.metrics.Mean(name="loss")
9         f = [16, 32, 64, 128, 256]
10        #####
11        self.downscale_blocks = [
12            DownscaleBlock(f[0]),
13            DownscaleBlock(f[1]),

```

```

14             DownscaleBlock(f[2]),
15             DownscaleBlock(f[3]),
16         ]
17 #####
18     self.bottle_neck_block = BottleNeckBlock(f[4])
19 #####
20     self.upscale_blocks = [
21         UpscaleBlock(f[3]),
22         UpscaleBlock(f[2]),
23         UpscaleBlock(f[1]),
24         UpscaleBlock(f[0]),
25     ]
26 #####
27     self.conv_layer = layers.Conv2D(1, (1, 1), padding="same",
28                                     activation="tanh")
29
30     def calculate_loss(self, target, pred):
31         # Edges
32         dy_true, dx_true = tf.image.image_gradients(target)
33         dy_pred, dx_pred = tf.image.image_gradients(pred)
34         weights_x = tf.exp(tf.reduce_mean(tf.abs(dx_true)))
35         weights_y = tf.exp(tf.reduce_mean(tf.abs(dy_true)))
36
37         # Depth smoothness
38         smoothness_x = dx_pred * weights_x
39         smoothness_y = dy_pred * weights_y
40
41         depth_smoothness_loss = tf.reduce_mean(abs(smoothness_x)) +
42             tf.reduce_mean(
43                 abs(smoothness_y))
44
45         # Structural similarity (SSIM) index
46         ssim_loss = tf.reduce_mean(
47             1
48             - tf.image.ssim(
49                 target, pred, max_val=WIDTH, filter_size=7, k1=0.01
50                 ** 2, k2=0.03 ** 2
51             )
52         )
53         # Point-wise depth
54         l1_loss = tf.reduce_mean(tf.abs(target - pred))

```

```

53
54     loss = (
55         (self.ssim_loss_weight * ssim_loss)
56         + (self.l1_loss_weight * l1_loss)
57         + (self.edge_loss_weight * depth_smoothness_loss)
58     )
59
60     return loss
61
62     @property
63     def metrics(self):
64         return [self.loss_metric]
65
66     def train_step(self, batch_data):
67         input, target = batch_data
68         with tf.GradientTape() as tape:
69             pred = self(input, training=True)
70             loss = self.calculate_loss(target, pred)
71
72         gradients = tape.gradient(loss, self.trainable_variables)
73         self.optimizer.apply_gradients(zip(gradients, self.
74             trainable_variables))
75         self.loss_metric.update_state(loss)
76         return {
77             "loss": self.loss_metric.result(),
78         }
79
80     def test_step(self, batch_data):
81         input, target = batch_data
82
83         pred = self(input, training=False)
84         loss = self.calculate_loss(target, pred)
85
86         self.loss_metric.update_state(loss)
87         return {
88             "loss": self.loss_metric.result(),
89         }
90
91     def call(self, x):
92         c1, p1 = self.downscale_blocks[0](x)
93         c2, p2 = self.downscale_blocks[1](p1)
94         c3, p3 = self.downscale_blocks[2](p2)

```

```

94     c4, p4 = self.downscale_blocks[3](p3)
95
96     bn = self.bottle_neck_block(p4)
97
98     u1 = self.upscale_blocks[0](bn, c4)
99     u2 = self.upscale_blocks[1](u1, c3)
100    u3 = self.upscale_blocks[2](u2, c2)
101    u4 = self.upscale_blocks[3](u3, c1)
102
103    return self.conv_layer(u4)

```

۳.۴ پایپ لاین دیتا

در آخرین گام پیش از ترین، نیاز است که دیتا را به صورت جنریتور به تابع فیت بدهیم. برای اینکار، یک کلاس طراحی می‌کنیم که از `tf.keras.utils.Sequence` ارث می‌برد. با پیاده‌سازی دو متده (که توضیح کافی در [این لینک](#) داده شده است) به راحتی یک کلاس برای ساخت جنریتور طراحی می‌کنیم و یک شی ساخته شده از آن را به مدل می‌دهیم. در این کلاس می‌توانیم هرگونه عملیات پیش‌پردازشی را انجام دهیم که در اینجا صرفاً برای ترین بهتر مدل یک عمل داون‌سملپل (نصف کردن ابعاد هر تصویر ورودی و خروجی) انجام می‌شود. کد این کلاس به شرح زیر است:

```

1 class DataGenerator(tf.keras.utils.Sequence):
2     def __init__(self, images, maps, batch_size=6, dim=(HEIGHT,
3                 WIDTH), n_channels=3, shuffle=True):
4
5         self.images = images
6         self.maps = maps
7         self.indices = list(range(images.shape[0]))
8         self.dim = dim
9         self.n_channels = n_channels
10        self.batch_size = batch_size
11        self.shuffle = shuffle
12        self.on_epoch_end()
13
14    def __len__():
15        return int(np.ceil(len(self.indices) / self.batch_size))
16
17    def __getitem__(self, index): # Generate one batch of data
18        if (index + 1) * self.batch_size > len(self.indices):
19            self.batch_size = len(self.indices) - index * self.
20                batch_size

```

```

20     # Generate indices of the batch
21     index = self.indices[index * self.batch_size: (index + 1) *
22                           self.batch_size]
23
24     # Find list of IDs
25     batch = [self.indices[k] for k in index]
26     x, y = self.data_generation(batch)
27
28     return x, y
29
30
31     # Updates indexes after each epoch
32
33     self.index = np.arange(len(self.indices))
34     if self.shuffle == True:
35         np.random.shuffle(self.index)
36
37     def load(self, image, depth_map):
38
39         # Loading the images
40
41         image_ = image
42         image_ = cv2.resize(image_, (WIDTH, HEIGHT))
43         image_ = tf.image.convert_image_dtype(image_, tf.float32)
44
45         # Loading the maps
46
47         depth_map = depth_map.squeeze()
48         depth_map = cv2.resize(depth_map, (WIDTH, HEIGHT))
49         depth_map = np.expand_dims(depth_map, axis=2)
50         depth_map = tf.image.convert_image_dtype(depth_map, tf.
51                                               float32)
52
53         return image_, depth_map
54
55     def data_generation(self, batch):
56
57         x = np.empty((self.batch_size, *self.dim, self.n_channels))
58         y = np.empty((self.batch_size, *self.dim, 1))
59         for i, batch_id in enumerate(batch):
60             x[i,], y[i,] = self.load(

```

```

60         self.images[batch_id] ,
61         self.maps[batch_id] ,
62     )
63     return x , y

```

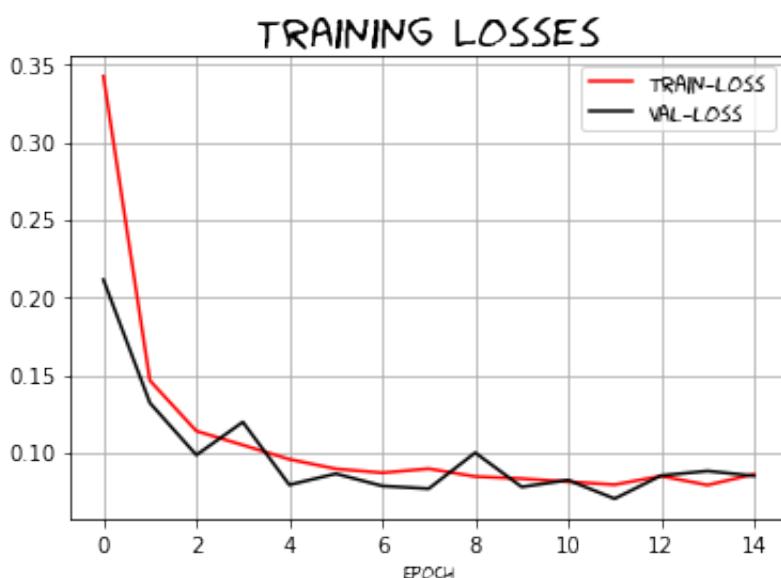
۴.۴ نگاهی به دیتا

نهایتاً پیش از اخمام فرآیند ترین، به کمک تابعی که برای نمایش دیتا (فاز ترین و تست) نوشته شده است، نمونه‌هایی از ورودی و خروجی موردنظر را مشاهده می‌کنیم:



۵.۴ ترین

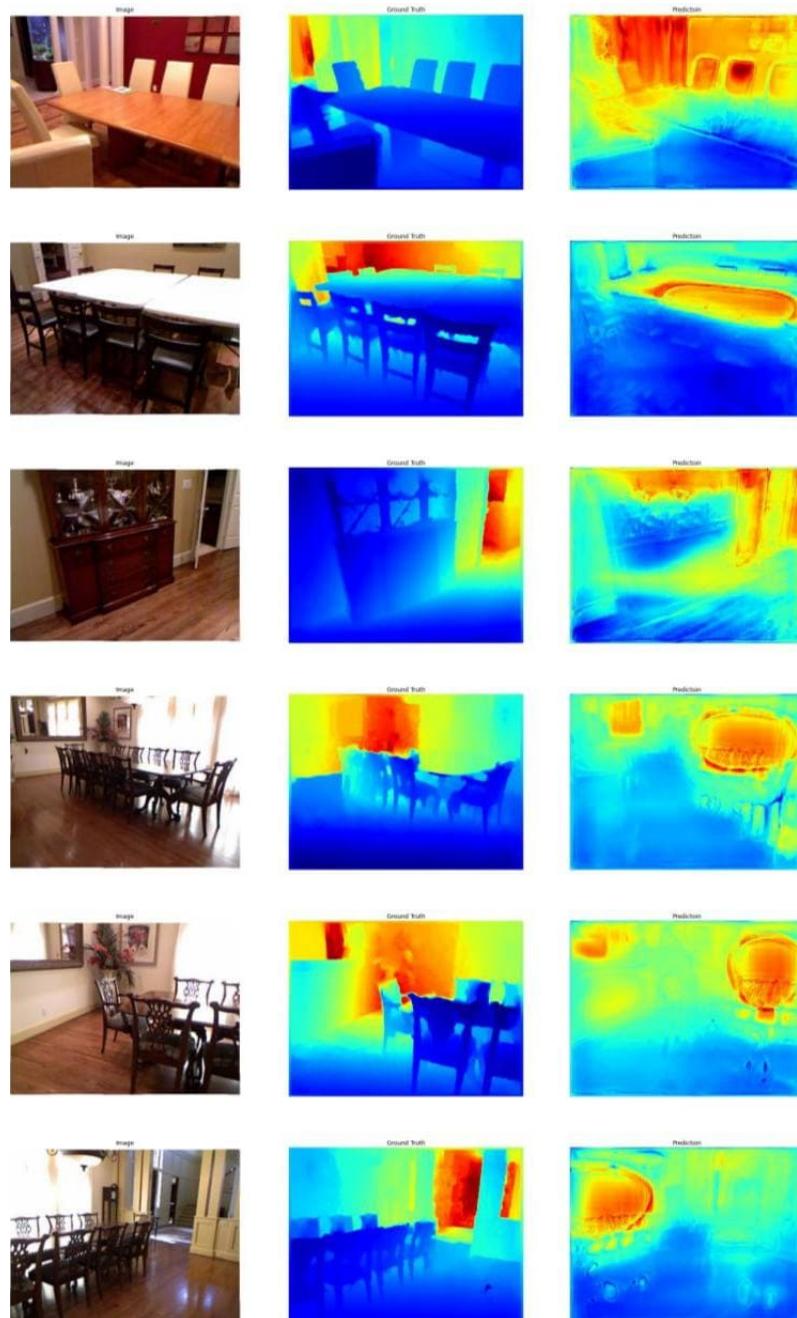
ترین را با ۱۵ ایپاک و بچهای ۳۲ تایی انعام می‌دهیم. نمودارهای مربوط به تابع هزینه به شرح زیر است:



لازم به ذکر است که از ۱۴۴۹ نمونه، ۱۴۰۰ تا برای ترین و سایر برای ولیدیشن در نظر گرفته شده اند.

۶.۴ نمونه‌ای از نتایج

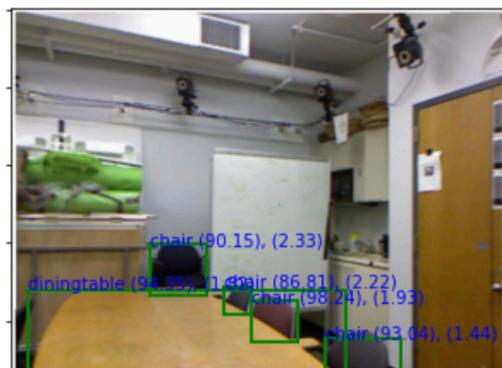
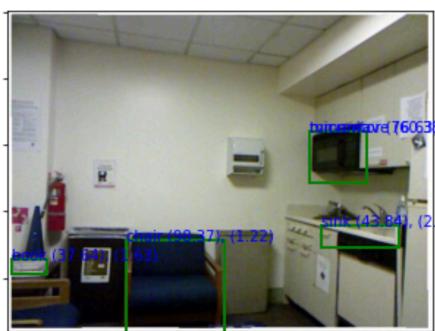
برای ۶ تا از دادگان ولیدیشن که مدل ندیده است، ورودی، خروجی ایده‌آل و خروجی مدل به شرح زیر است که دقت نسبتاً خوبی را نشان می‌دهد:



۵ ترکیب مدل ها

در نهایت با قرار دادن مدل تشخیص اشیا، تشخیص شی های داخل عکس، تشخیص باتدینگ باکس ها، رد کردن هکس از مدل تشخیص عمیق و بدست آوردن عمق پیکسل ها، طبق گفته مقاله روی پیکسل های داخل باتدینگ باکس ها میانگین گرفته و عنوان عدد عمق کنار اسم ابجکت، دقت تشخیص، قرار میدهیم که مثال های آن به شرح زیر است :

۱.۵ مثال ها



در پایان نیز برای بدسن آوردن خطأ، آن را بصورت کشی از اشیا داخل عکس که نشان داده شده اند ضربدر خطای میانگین مربعات پیکسل های عمق داخل باتدینگ باکس ها قرار میدهیم که روی تمام دادگان میانگین گرفته و به عدد زیر میرسیم:

```
[ 58 ] np.nanmean(np.array(error))
```

1.188327756524086

که نشاندهنده دقت بسیار خوبی میباشد.