江南大学

课 程 设 计 报 告

题　　目：用于语义分割的全卷积网络

题 目（英文）：Fully Convolutional Networks for Semantic Segment

学　　号：1033190614、1033190502、1033190609

姓　　名：张梓琦，关乐平，刘鸿嘉

课　　程：人工智能

指 导 教 师：徐天阳

江 南 大 学

地址：无锡市蠡湖大道 1800 号

二〇二二 年 六 月

Catalogue

1 Introduction

1.1 Development History

All developments are long periods of technical accumulation, coupled with qualitative changes that occur when some external conditions are met. We have briefly summarised several periods of image segmentation.

Before 2000, when digital image processing we used methods based on several categories: threshold segmentation, region segmentation, edge segmentation, texture features, clustering, etc.

Between 2000 and 2010, the main methods were based on four categories: graph theory, clustering, classification, and a combination of clustering and classification.

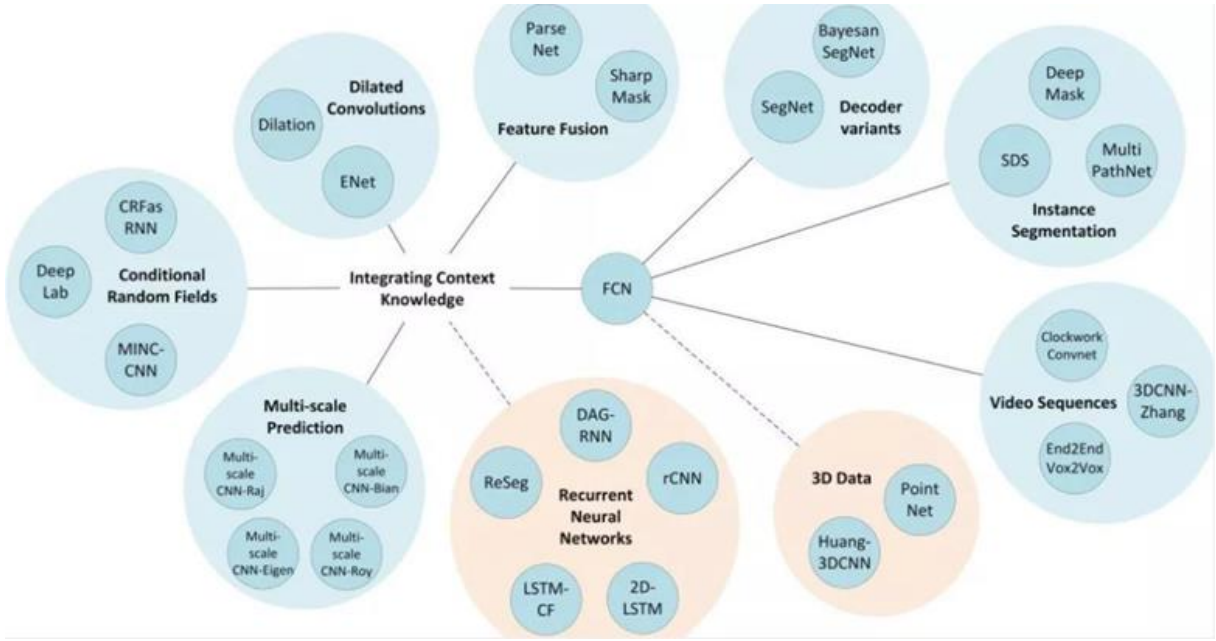
From 2010 to the present, the rise of neural network models and the development of deep learning has involved several main models

TABLE 2: Summary of semantic segmentation methods based on deep learning.

Name and Reference	Architecture	Targets							Source Code	Contribution(s)
		Accuracy	Efficiency	Training	Instance	Sequences	Multi-modal	3D		
Fully Convolutional Network [65]	VGG-16(FCN)	+	+	+	✗	✗	✗	✗	✓	Forerunner
SegNet [66]	VGG-16 + Decoder	+ + +	++	+	✗	✗	✗	✗	✓	Encoder-decoder
Bayesian SegNet [67]	SegNet	+ + +	+	+	✗	✗	✗	✗	✓	Uncertainty modeling
DeepLab [68] [69]	VGG-16/ResNet-101	+ + +	+	+	✗	✗	✗	✗	✓	Standalone CRF, atrous convolutions
MINC-CNN [43]	GoogLeNet(FCN)	+	+	+	✗	✗	✗	✗	✓	Patchwise CNN, Standalone CRF
CRFasRNN [70]	FCN-8s	+	++	+ + +	✗	✗	✗	✗	✓	CRF reformulated as RNN
Dilation [71]	VGG-16	+ + +	+	+	✗	✗	✗	✗	✓	Dilated convolutions
ENet [72]	ENet bottleneck	++	+ + +	+	✗	✗	✗	✗	✓	Bottleneck module for efficiency
Multi-scale-CNN-Raj [73]	VGG-16(FCN)	+ + +	+	+	✗	✗	✗	✗	✗	Multi-scale architecture
Multi-scale-CNN-Eigen [74]	Custom	+ + +	+	+	✗	✗	✗	✗	✓	Multi-scale sequential refinement
Multi-scale-CNN-Roy [75]	Multi-scale-CNN-Eigen	+ + +	+	+	✗	✗	++	✗	✗	Multi-scale coarse-to-fine refinement
Multi-scale-CNN-Bian [76]	FCN	++	+	++	✗	✗	✗	✗	✗	Independently trained multi-scale FCNs
ParseNet [77]	VGG-16	+ + +	+	+	✗	✗	✗	✗	✓	Global context feature fusion
ReSeg [78]	VGG-16 + ReNet	++	+	+	✗	✗	✗	✗	✓	Extension of ReNet to semantic segmentation
LSTM-CF [79]	Fast R-CNN + DeepMask	+ + +	+	+	✗	✗	✗	✗	✓	Fusion of contextual information from multiple sources
2D-LSTM [80]	MDRNN	++	++	+	✗	✗	✗	✗	✗	Image context modelling
rCNN [81]	MDRNN	+ + +	++	+	✗	✗	✗	✗	✓	Different input sizes, image context
DAG-RNN [82]	Elman network	+ + +	+	+	✗	✗	✗	✗	✓	Graph image structure for context modelling
SDS [10]	R-CNN + Box CNN	+ + +	+	+	++	✗	✗	✗	✓	Simultaneous detection and segmentation
DeepMask [83]	VGG-A	+ + +	+	+	++	✗	✗	✗	✓	Proposals generation for segmentation
SharpMask [84]	DeepMask	+ + +	+	+	+ + +	✗	✗	✗	✓	Top-down refinement module
MultiPathNet [85]	Fast R-CNN + DeepMask	+ + +	+	+	+ + +	✗	✗	✗	✓	Multi path information flow through network
Huang-3DCNN [86]	Own 3DCNN	+	+	+	✗	✗	✗	+ + +	✗	3DCNN for voxeli[illegible]
PointNet [87]	Own MLP-based	++	+	+	✗	✗	✗	+ + +	✓	Segmentation of uns[illegible]
Clockwork Convnet [88]	FCN	++	++	+	✗	+ + +	✗	✗	✓	Clockwork scheduling [illegible]
3DCNN-Zhang [88]	Own 3DCNN	++	+	+	✗	+ + +	✗	✗	✓	3D convolutions and graph cut for sequences
End2End Vox2Vox [89]	C3D	++	+	+	✗	+ + +	✗	✗	✗	3D convolutions/deconvolutions for sequences

1-A

By the end of 2017, we had differentiated hundreds of model structures. Of course, after examining the techniques and principles, we have discovered the characteristic that the most successful current deep learning techniques for image segmentation are based on a common pioneer: the FCN (Fully Convolutional Network).



1-B

1.2 Purpose of the study

For FCN, it means that all layers in the model network are convolutional layers, but the traditional CNN has several disadvantages: firstly, the storage overhead is large, the sliding window is large, each window needs storage space to hold features and discriminative categories, and using a fully connected structure, the last few layers are nearly exponentially storage incremental; secondly, the computational efficiency is low, a lot of repeated computations; finally, the sliding window size is relatively independent, and the use of full connectivity at the end can only

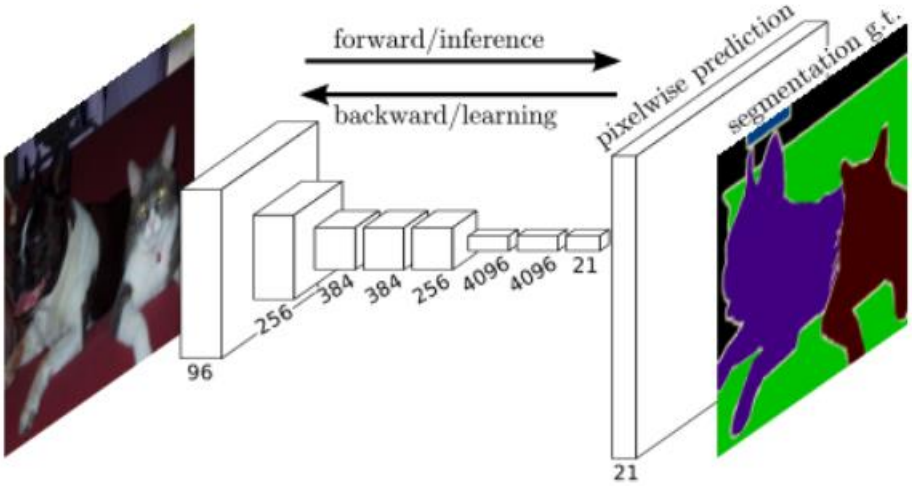
constrain local features. So to solve some of the problems of CNN, FCN converts the last three fully-connected layers in CNN into three convolutional layers.

1.3 Significance of the study

Convolutional networks are driving advances in recognition technology. Convolutional networks are not only improving overall image classification, but are also making advances in local tasks with structured outputs. These advances include bounding box target detection, partial and critical point prediction, and local correspondence. From rough inference to fine inference, it is natural that the next step is to make predictions for each pixel.

Fully convolutional versions of existing networks can predict dense outputs from inputs of arbitrary size. Both learning and inference are performed over the entire image by intensive feed-forward computation and back propagation. Pixel prediction and learning in the network is achieved by subsampling pooling in the upsampling layer within the network.

FCN classifies images at the pixel level, thus solving the problem of semantic segmentation. Unlike the classical CNN, which uses a fully-connected layer after the convolution layer to obtain a fixed length feature vector for classification (fully connected layer + softmax output), the FCN can accept an input image of arbitrary size and uses a deconvolution layer to upsample the feature map of the last convolution layer to restore it to the same size as the input image, thus generating a prediction for each pixel. A prediction is made for each pixel, while preserving the spatial information in the original input image, and the final pixel-by-pixel classification is performed on the upsampled feature map.



1-C

2 Approach

2.1 Fully convolutional networks

The only difference between the fully connected and convolutional layers is that the neurons in the convolutional layer are connected to only one local region in the input data and that the neurons in the convolutional column share parameters. However, in both types of layers, the neurons compute a dot product, so their functional form is the same. It is therefore possible to interconvert the two.

For either convolutional layer, there exists a fully connected layer that implements the same forward propagation function as it does. The weight matrix is a huge matrix, with all but some specific blocks being zero. And in most of these blocks, the elements are equal.

Conversely, any fully-connected layer can be transformed into a convolutional layer. For example, a fully connected layer with K=4096 and an input data body of size 7∗7∗512 can be equivalently viewed as a convolutional layer with F=7,P=0,S=1,K=4096. In other words, the size of the filter is set to match the size of the input data body. Since there is only a single depth column covering and sliding over the input data body, the output will be 1∗1∗4096, which is the same as using the initial fully-connected layer.

Fully-connected layers into convolutional layers: Of the two transformations, converting fully-connected layers into convolutional layers is more useful in practice. Suppose the input to a convolutional neural network is

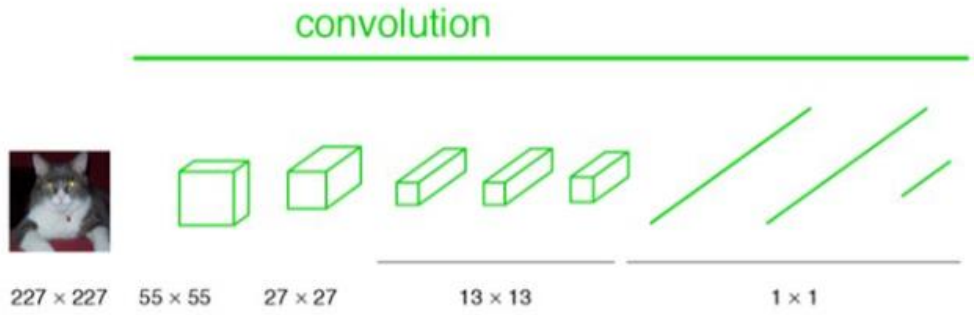
a 224x224x3 image, and a series of convolutional and downsampling layers transform the image data into a body of activation data of size 7x7x512. AlexNet uses two fully-connected layers of size 4096, and the last fully-connected layer with 1000 neurons is used to compute classification scores. We can transform any of these 3 fully-connected layers into a convolutional layer.

For the first fully-connected layer with a connection area of [7x7x512], let its filter size be F=7, so that the output data body is [1x1x4096].

For the second fully connected layer, make the filter size F=1 so that the output data body is [1x1x4096].

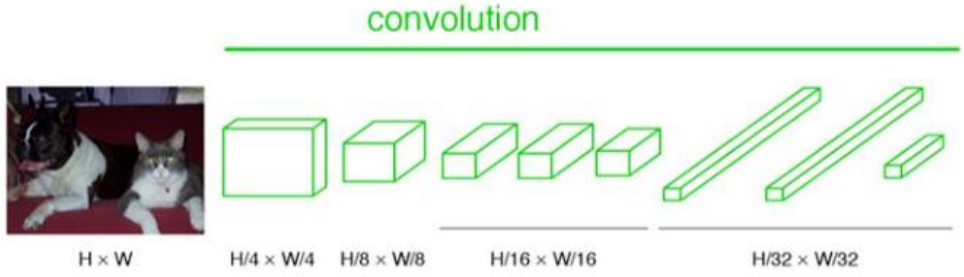
Do the same for the last fully-connected layer, making its F=1, so that the final output is [1x1x1000].

As shown below, the FCN converts the fully-connected layers in a traditional CNN into convolutional layers, corresponding to the CNN network FCN converts the last three fully-connected layers into three convolutional layers. In the traditional CNN structure, the first 5 layers are convolutional layers, layers 6 and 7 are each a 1D vector of length 4096, and layer 8 is a 1D vector of length 1000, corresponding to 1000 different classes of probabilities. fcn represents these 3 layers as convolutional layers, and the sizes of the convolutional kernels (number of channels, width, height) are (4096,1,1), (4096,1,1), (1000,1,1). There is no difference in the numbers, but convolution is a different concept and computation process from fully connected, using the weights and biases already trained in the previous CNN, but the difference is that the weights and biases have their own range and belong to a convolution kernel of their own. Therefore, all the layers in the FCN are convolutional layers, so it is called a full convolutional network.



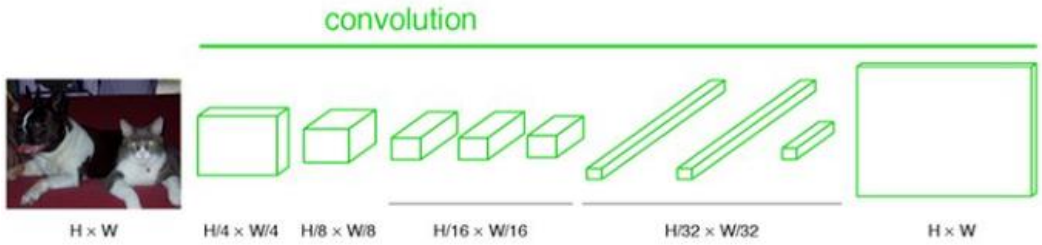
2-A

The input image size in the CNN is agreed to be fixed resize to 227x227, 55x55 after pooling in the first layer, 27x27 after pooling in the second layer, and 13*13 after pooling in the fifth layer. The FCN input image is H*W, which becomes 1/4 of the original image size after pooling in the first layer, 1/8 of the original image size in the second layer, 1/16 of the original image size in the fifth layer, and 1/32 of the original image size in the eighth layer (erratum: in fact the real code is 1/2 in the first layer, and so on).



2-B

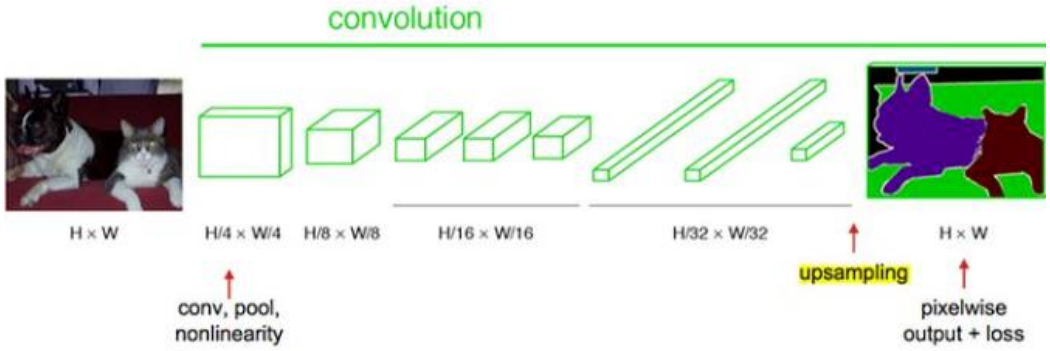
After multiple convolution and pooling, the image gets smaller and smaller, and the resolution gets lower and lower. When the image reaches H/32∗W/32, the image is the smallest layer, and the resulting map is called heatmap. The heatmap is our most important high-dimensional diagnostic map, and after getting the heatmap of high-dimensional features is the most important step and the last step to upsampling the original image, enlarging, enlarging, enlarging the image to the size of the original image.



2-C

Each layer of data in a convnet is a three-dimensional array of size h × w × d, where h and w are spatial dimensions, and d is the feature or channel dimension. The first layer is the image, with pixel size h × w, and d color channels. Locations in higher layers correspond to the locations in the image they are path-connected to, which are called their receptive fields.

The final output is 1000 heatmap images that have been upsampled to the original size. In order to classify each pixel by predicting the label into the final image that has been semantically segmented, there is a small trick here, which is to finally classify the pixel by finding the maximum numerical description (probability) of its position in the 1000 images, pixel by pixel, as the pixel. . This results in an image that has already been classified, as shown below with the dog and cat on the right.



2-D

Convnets are built on translation invariance. Their basic components (convolution, pooling, and acti-vation functions) operate on local input regions, and depend only on relative spatial coordinates.

$$y_{ij} = f_{ks}(\{x_{si+\delta i, sj+\delta j}\}_{0 \le \delta i, \delta j \le k})$$

2-E

2.2 Translation invariance

In Euclidean geometry, translation is a geometric change that represents moving every point in an image or a space the same distance in the same direction. For example, for an image classification task, the target in the image should give the same result regardless of where it is moved in the image, this is the translation invariance in a convolutional neural network.

Translational invariance means that the system produces exactly the same response (output), regardless of how its input has been translated. Translational equivariance means that the system works the same at different locations, but its response changes as the target location changes.

2.3 Receptive Field

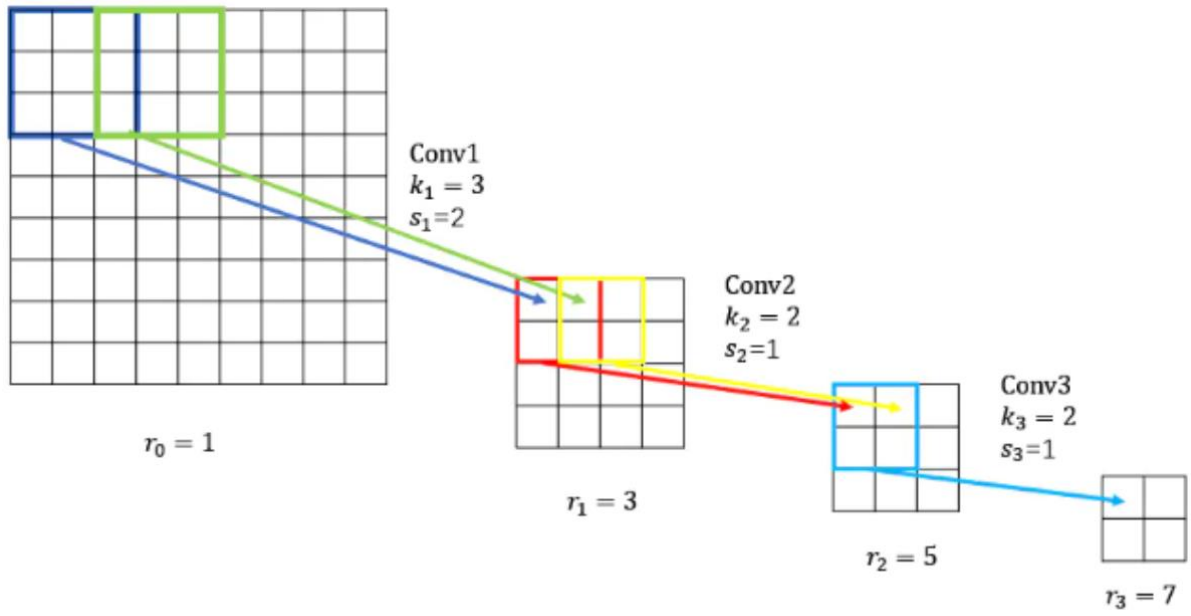
2.3.1 Receptive Field Introduction

In a convolutional neural network, Receptive Field is defined as the size of the area mapped on the original image for each pixel on the featuremap output from each layer of the convolutional neural network, where the original image is the input image to the network, which is pre-processed (e.g. resize, warp, crop).

The reason why neurons cannot perceive all the information of the original image is that convolutional and pooling

layers are commonly used in convolutional neural networks, and are locally connected between layers.

A larger value of the neuron's receptive field means that it has access to a larger range of the original image, which means that it may contain more global, higher-level semantic features; conversely, a smaller value means that the features it contains tend to be more local and detailed. The value of the perceptual field can therefore be used to roughly determine the level of abstraction of each layer.



2-F

2.3.2 Receptive Field Calculation

The perceptive field size of a deep convolutional layer is related to the filter size and step size of all the layers before it, and these two parameters are involved in convolutional and pooling layers. We use kn, sn, and r n to denote the kernel_size, stride, and receptive_field of the nth layer, respectively, so that for a convolutional neural network, the receptive field is calculated as follows:

$r_0 = 1, r_1 = k_1$

$$r_n = r_{n-1} * k_n - (k_n - 1) * (r_{n-1} - \prod_{i=1}^{n-1} s_i) \qquad n >= 2$$

2-G

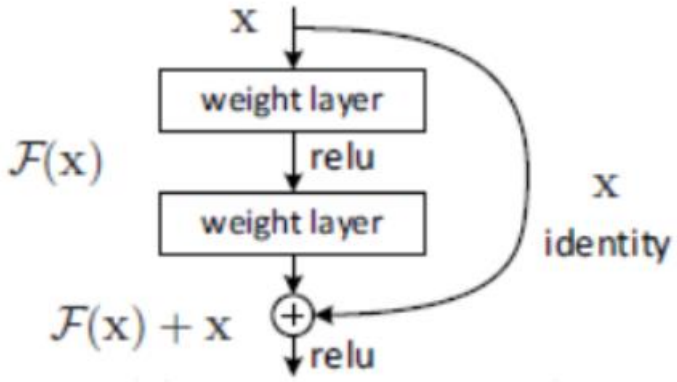
A larger value for the receptive field of a neuron means that it has access to a larger range of original images, which means that it may contain more global, higher semantic features; conversely, a smaller value means that the features it contains tend to be more local and detailed. The value of the perceptual field can therefore be used to roughly determine the level of abstraction of each layer

The effective receptive field is often smaller than the theoretical receptive field, because the input layer uses the edge points less often than the middle points, and therefore makes a different contribution, so that after multiple layers of convolution, the input layer's contribution to the feature map points is distributed in the shape of a Gaussian distribution

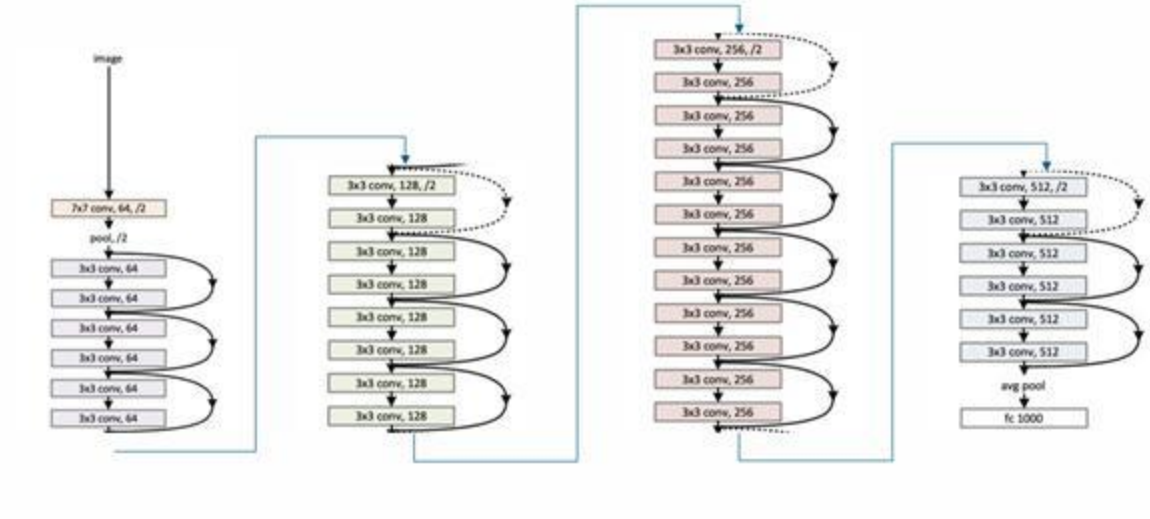
2.4 ResNet

He Kaiming proposed the residual neural network, or Reset, in 2015 and won the classification competition at ILSVRC-2015.

ResNet can effectively eliminate the gradient dispersion or gradient explosion problem caused by the increase in the number of convolutional layers.The core idea of ResNet is that the network output is divided into 2 parts identity mapping, residual mapping, i.e. y = x + F(x), which is illustrated as follows.



2-H



2-I

2.5 Transposed Convolution

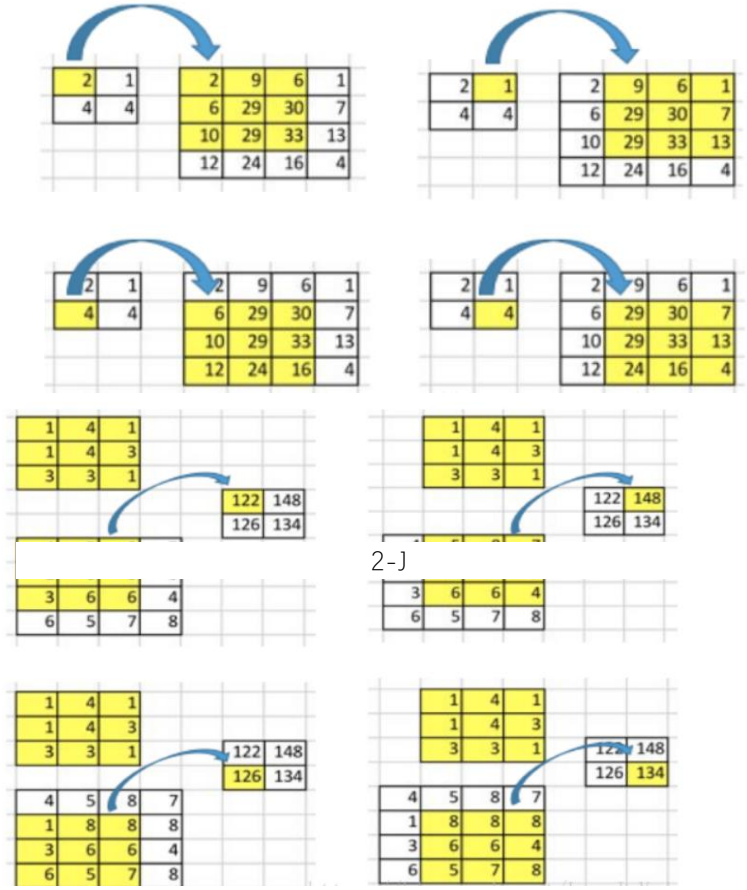
2.5.1 Upsample

In the field of deep learning applied to computer vision, as the input image is extracted by convolutional neural network (CNN), the size of the output often becomes smaller, and sometimes we need to restore the image to its original size for further computation, this operation of mapping the image from small resolution to large resolution by expanding the image size is called upsampling.

2.5.2 Transposed Convolution

Deconvolution can be understood as the inverse of convolution. Deconvolution does not recover the loss of values caused by the convolution operation, it simply transforms the steps in the convolution process in reverse once, and can therefore be called transposed convolution. The parameters can be learned in end-to-end training using deconvolution. The formula for deconvolution is: output = (input -1)×stride + k - 2p

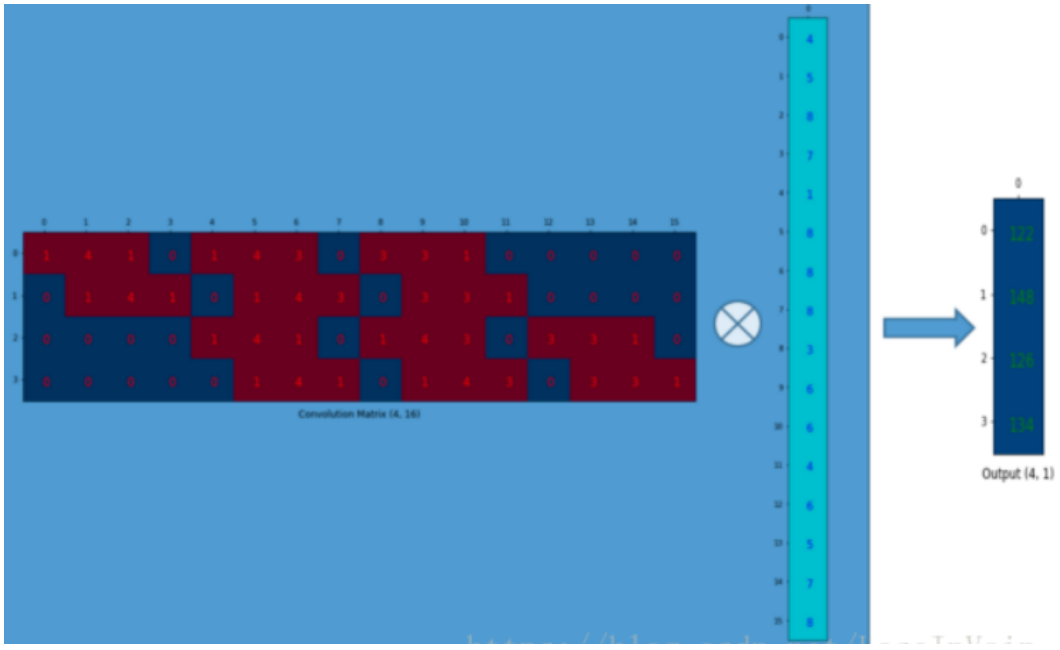
A convolution, for example, takes a large matrix and passes it through a convolution kernel to call it a small matrix; a convolution operation is a many-to-one mapping relationship. A deconvolution operation is a one-to-many mapping relationship. A deconvolution is the reversal of a convolution operation, converting a small matrix into a large one.



2-J

2-K

We want to go from 4(2 × 2) to 16(4 × 4), so we use a 16 × 4 matrix, but one more thing to note is that we are trying to maintain a 1 to 9 mapping relationship.



2-L

Suppose we transpose that this convolution matrix C (4 × 16) becomes C_T (16 × 4). We can matrix multiply the C^T and column vectors (4 × 1) to produce a 16 × 1 output matrix. This transpose matrix maps exactly one element to nine elements.

2.6 Jump structure

The 32-pixel step size of the final prediction layer limits the scale of detail in the upsampled output. FCN therefore solves this problem by adding a treaty structure, which fuses the final prediction layer with a lower layer with a finer step size. This turns the linear topology into a DAG, with edges jumping from lower to higher layers. Because they see fewer pixels, finer scale predictions should require fewer layers, so it makes sense to output from a shallower network. Combining fine and coarse layers allows the model to make local predictions that fit the global structure.

A jump structure is added between layers to fuse coarse, semantic and local appearance information. This jump structure is learned end-to-end to optimise the semantic and spatial accuracy of the output.

3 Experimental Evaluation

3.1 Experimental conditions

For the selection of the dataset, we first used the base dataset from VOC2012, which was divided into 21 categories, namely 'background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car', 'cat', 'chair', ' cow', 'diningtable', 'dog', 'horse', 'motorbike', 'person', 'potted plant', 'sheep', 'sofa', 'train', 'tv/monitor'. In addition to this, we tested this on our own photos around us and observed the segmentation effect.

3.2 Experimental procedure

Reading data from a dataset:

```
voc_root = 'data\\VOCdevkit\\VOC2012'
print("voc_root: \n", os.listdir(voc_root), "\n")
print("JPEGImages: \n", os.listdir(voc_root+'/JPEGImages')[:10], "\n")
print("SegmentationClass: \n", os.listdir(voc_root+'/SegmentationClass')[:10], "\n")

def read_images(root = voc_root, train = True):
 # 从数据集中读取数据
# train == True 读取训练集
# train == False 读取测试集
 txt_fname = root + '/ImageSets/Segmentation/' + ('train.txt' if train else 'val.txt
')
with open(txt_fname, 'r') as f:
images = f.read().split()
data = [os.path.join(root, 'JPEGImages', i + '.jpg') for i in images]
label = [os.path.join(root, 'SegmentationClass', i + '.png') for i in images]
return data, label
```

Cutting image:

```
def crop_image(data, label, height, width):
    # 切割图象
```

2-G

```
    '''''
    data is PIL.Image object
    label is PIL.Image object
    '''
    box = (0, 0, width, height)
    data = data.crop(box)
    label = label.crop(box)
    return data, label
```

Define labels and colours for each category

```
# 定义数据集每个类别的标签
classes = ['background', 'aeroplane', 'bicycle', 'bird', 'boat',
           'bottle', 'bus', 'car', 'cat', 'chair', 'cow', 'diningtable',
           'dog', 'horse', 'motorbike', 'person', 'potted plant',
           'sheep', 'sofa', 'train', 'tv/monitor']

# 定义数据集每个类别的显示颜色(RGB)
colormap = [[0, 0, 0], [128, 0, 0], [0, 128, 0], [128, 128, 0], [0, 0, 128],
            [128, 0, 128], [0, 128, 128], [128, 128, 128], [64, 0, 0], [192, 0, 0],
            [64, 128, 0], [192, 128, 0], [64, 0, 128], [192, 0, 128],
            [64, 128, 128], [192, 128, 128], [0, 64, 0], [128, 64, 0],
            [0, 192, 0], [128, 192, 0], [0, 64, 128]]

cm2lbl = np.zeros(256 ** 3)
for i, cm in enumerate(colormap):
    cm2lbl[(cm[0] * 256 + cm[1]) * 256 + cm[2]] = i
```

Fill the labels with the subscript information of the corresponding category according to the RGB values

```
def image2label(im):
    # 将标签按照 RGB 值填入对应类别的下标信息
    data = np.array(im, dtype="int32")
    idx = (data[:, :, 0] * 256 + data[:, :, 1]) * 256 + data[:, :, 2]
    # print(np.shape(cm2lbl), np.shape(idx))
    # print(cm2lbl[0])
    return np.array(cm2lbl[idx], dtype="int32")
```

Cropping data to h*w size

```
def image_transforms(data, label, height, width):
    # 将数据裁切为 h*w 大小
    data, label = crop_image(data, label, height, width)
    # 将数据转换成 tensor，并且做标准化处理
    im_tfs = tfs.Compose([
        # 将 PIL Image 或 numpy.ndarray 转换为 tensor，并除 255 归一化到[0,1]之间
        tfs.ToTensor(),
        # 标准化处理-->转换为标准正太分布，使模型更容易收敛
        tfs.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225])
    ])
    data = im_tfs(data)
    label = image2label(label)
    label = torch.from_numpy(label)
    return data, label
```

Data pre-processing, filtered images

```
class VOCSegDataset(Dataset):
    # 数据预处理

    # 构造函数
    def __init__(self, train, height, width, transforms):
        self.height = height
        self.width = width
        self.fnum = 0  # 用来记录被过滤的图片数
        self.transforms = transforms
        data_list, label_list = read_images(train=train)
        # 过滤不符合规则的图片
```

```
        self.data_list = self._filter(data_list)
        self.label_list = self._filter(label_list)
        if (train == True):
            print("训练集：加载了 " + str(len(self.data_list)) + " 张图片和标签" + ",过滤了" + str(self.fnum) + "张图片")
        else:
            print("测试集：加载了 " + str(len(self.data_list)) + " 张图片和标签" + ",过滤了" + str(self.fnum) + "张图片")

    # 过滤掉长小于 height 和宽小于 width 的图片
    def _filter(self, images):
        img = []
        for im in images:
            if (Image.open(im).size[1] >= self.height and
                    Image.open(im).size[0] >= self.width):
                img.append(im)
            else:
                self.fnum += 1
        return img

    # 重载 getitem 函数，使类可以迭代
    def __getitem__(self, idx):
        data = self.data_list[idx]
        label = self.label_list[idx]
        data = Image.open(data)
        label = Image.open(label).convert('RGB')
        data, label = self.transforms(data, label, self.height, self.width)
        return data, label

    def __len__(self):
        return len(self.data_list)
```

Load the dataset and output the number of filtered images and the number loaded

```
height = 224
width = 224

voc_train = VOCSegDataset(True, height, width, image_transforms)
voc_test = VOCSegDataset(False, height, width, image_transforms)
train_data = DataLoader(voc_train, batch_size=8, shuffle=True)
valid_data = DataLoader(voc_test, batch_size=8)
```

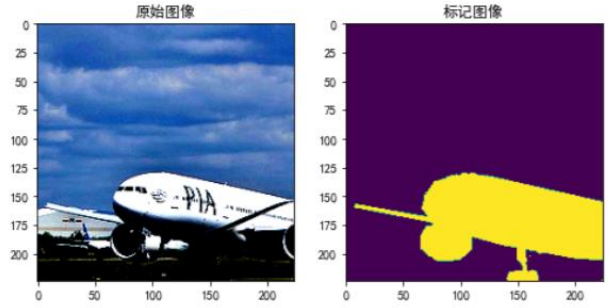
```
训练集：加载了 1456 张图片和标签,过滤了16张图片
测试集：加载了 1436 张图片和标签,过滤了26张图片
```

Display of data sets

```
x = np.array(voc_train[9][0]).transpose(1,2,0)
y = np.array(voc_train[9][1])
print(np.shape(y))
fig, axs = plt.subplots(1, 2, figsize = (8, 8))
axs[0].imshow(x)
axs[0].set_title("原始图像")
axs[1].imshow(y)
axs[1].set_title("标记图像")
```

```
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
(224, 224)
```



3-A

Constructing the kernel: Here we use a bilinear interpolation method to construct the kernel, calculating each output y_{ij} from the four nearest inputs through a linear mapping that depends only on the relative positions of the input and output units

```
def bilinear_kernel(in_channels, out_channels, kernel_size):
    '''''
    return a bilinear filter tensor
    '''
    factor = (kernel_size + 1) // 2
    if kernel_size % 2 == 1:
        center = factor - 1
    else:
        center = factor - 0.5
    og = np.ogrid[:kernel_size, :kernel_size]
    # 生成横向和纵向的 0-kernel_size、步长为一的两个二维数组
    filt = (1 - abs(og[0] - center) / factor) * (1 - abs(og[1] - center) / factor)
    weight = np.zeros((in_channels, out_channels, kernel_size, kernel_size), dtype='
float32')
    weight[range(in_channels), range(out_channels), :, :] = filt
    return torch.from_numpy(weight)
```

When constructing fcn, we use the pre-trained resnet34 instead of vgg

```
model_urls = {
    'resnet18': 'https://download.pytorch.org/models/resnet18-5c106cde.pth',
    'resnet34': 'https://download.pytorch.org/models/resnet34-333f7ec4.pth',
    'resnet50': 'https://download.pytorch.org/models/resnet50-19c8e357.pth',
    'resnet101': 'https://download.pytorch.org/models/resnet101-5d3b4d8f.pth',
    'resnet152': 'https://download.pytorch.org/models/resnet152-b121ed2d.pth',
}

# 使用预训练的 resnet 34（34 层残差网络）代替论文中的 vgg 实现 fcn
model_root = "./model/resnet34-333f7ec4.pth"
pretrained_net = models.resnet34(pretrained=False)
pre = torch.load(model_root)
pretrained_net.load_state_dict(pre)
# 分类的总数
num_classes = len(classes)
```

Construction of fully convolutional neural network (FCN) models

```
class fcn(nn.Module):
    def __init__(self, num_classes):
        super(fcn, self).__init__()

```

```
        # 第一段，通道数为 128，输出特征图尺寸为 28*28
        # conv1, conv2_x, conv3_x
        self.stage1 = nn.Sequential(*list(pretrained_net.children())[:-4])
        # 第二段，通道数为 256，输出特征图尺寸为 14*14
        # conv4_x
        self.stage2 = list(pretrained_net.children())[-4]
        # 第三段，通道数为 512，输出特征图尺寸为 7*7
        # conv5_x
        self.stage3 = list(pretrained_net.children())[-3]

        # 三个 kernel 为 1*1 的卷积操作，各个通道信息融合
        self.scores1 = nn.Conv2d(512, num_classes, 1)
        self.scores2 = nn.Conv2d(256, num_classes, 1)
        self.scores3 = nn.Conv2d(128, num_classes, 1)

        # 反卷积，将特征图尺寸放大八倍
        self.upsample_8x = nn.ConvTranspose2d(num_classes, num_classes, kernel_size=16, stride=8, padding=4, bias=False)
        self.upsample_8x.weight.data = bilinear_kernel(num_classes, num_classes, 16)


        # 反卷积，将特征图尺寸放大两倍
        self.upsample_2x_1 = nn.ConvTranspose2d(num_classes, num_classes, kernel_size=4, stride=2, padding=1, bias=False)
        self.upsample_2x_1.weight.data = bilinear_kernel(num_classes, num_classes, 4)
        self.upsample_2x_2 = nn.ConvTranspose2d(num_classes, num_classes, kernel_size=4, stride=2, padding=1, bias=False)
        self.upsample_2x_2.weight.data = bilinear_kernel(num_classes, num_classes, 4)

    def forward(self, x):
        x = self.stage1(x)
        s1 = x  # 224/8 = 28

        x = self.stage2(x)
        s2 = x  # 224/16 = 14

        x = self.stage3(x)
        s3 = x  # 224/32 = 7

        # 将各通道信息融合
        s3 = self.scores1(s3)
        # 上采样 放大二倍
        s3 = self.upsample_2x_1(s3)

        s2 = self.scores2(s2)
        # 将二三层训练特征合成 14*14
        s2 = s2 + s3
        s2 = self.upsample_2x_2(s2)

        # 28*28
        s1 = self.scores3(s1)
        # 将一二三层训练特征合成 28*28
        s = s1 + s2
        # 将 s 放大八倍，变为原图像尺寸 224*224
        s = self.upsample_8x(s)

        # 返回特征图
        return s
```

FCN network structure.

```
print(net)
```

```
(stage1): Sequential(
  (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

3-B

```
  (5): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
)
```

3-C

```
  (0): BasicBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (2): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (3): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (4): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (5): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

3-D

```
(stage3): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (2): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
(scores1): Conv2d(512, 21, kernel_size=(1, 1), stride=(1, 1))
(scores2): Conv2d(256, 21, kernel_size=(1, 1), stride=(1, 1))
(scores3): Conv2d(128, 21, kernel_size=(1, 1), stride=(1, 1))
(upsample_8x): ConvTranspose2d(21, 21, kernel_size=(16, 16), stride=(8, 8), padding=(4, 4), bias=False)
(upsample_2x_1): ConvTranspose2d(21, 21, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
(upsample_2x_2): ConvTranspose2d(21, 21, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
```

3-E

Model evaluation indicators, calculation of confusion matrix and determination of accuracy.

```python
# 计算混淆矩阵
def _fast_hist(label_true, label_pred, n_class):
    # mask 在和 label_true 相对应的索引的位置上填入 true 或者 false
    # label_true[mask]会把 mask 中索引为 true 的元素输出
    mask = (label_true >= 0) & (label_true < n_class)
    hist = np.bincount(
        n_class * label_true[mask].astype(int) +
        label_pred[mask], minlength=n_class ** 2).reshape(n_class, n_class)
    return hist


"""
label_trues 正确的标签值
label_preds 模型输出的标签值
n_class 数据集中的分类数
"""
def label_accuracy_score(label_trues, label_preds, n_class):
    """Returns accuracy score evaluation result.
      - overall accuracy
      - mean accuracy
      - mean IU
      - fwavacc
    """
    hist = np.zeros((n_class, n_class))
    # 通过迭代器将一个个数据进行计算
    for lt, lp in zip(label_trues, label_preds):
        hist += _fast_hist(lt.flatten(), lp.flatten(), n_class)

    # np.diag(a)假如 a 是一个二维矩阵，那么会输出矩阵的对角线元素
    # np.sum()可以计算出所有元素的和。如果 axis=1，则表示按行相加
    acc = np.diag(hist).sum() / hist.sum()
    # np.diag 以一维数组的形式返回方阵的对角线
    acc_cls = np.diag(hist) / hist.sum(axis=1)
    # nanmean 会自动忽略 nan 的元素求平均
    acc_cls = np.nanmean(acc_cls)
    iu = np.diag(hist) / (hist.sum(axis=1) + hist.sum(axis=0) - np.diag(hist))
    mean_iu = np.nanmean(iu)
    freq = hist.sum(axis=1) / hist.sum()
    fwavacc = (freq[freq > 0] * iu[freq > 0]).sum()

```

```
    return acc, acc_cls, mean_iu, fwavacc
```

Defining hyperparameters and saving training data.

```
net = fcn(num_classes)
PATH = "./model/fcn-resnet34.pth"
net.load_state_dict(torch.load(PATH))
if torch.cuda.is_available():
    net = net.cuda()

# 训练时的数据
train_loss = []
train_acc = []
train_acc_cls = []
train_mean_iu = []
train_fwavacc = []

# 验证时的数据
eval_loss = []
eval_acc = []
eval_acc_cls = []
eval_mean_iu = []
eval_fwavacc = []
```

Conducting network training.

```
# 损失
criterion = nn.NLLLoss()

# 加速器 sgd
Eta = 1e-2
basic_optim = torch.optim.SGD(net.parameters(), lr=Eta, weight_decay=1e-4)
optimizer = basic_optim

# 网络训练
EPOCHES = 60

exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

Model training.

```
for e in range(EPOCHES):

    _train_loss = 0  # 记录一轮训练总损失
    _train_acc = 0
    _train_acc_cls = 0
    _train_mean_iu = 0
    _train_fwavacc = 0
    exp_lr_scheduler.step()
    prev_time = datetime.now()
    net = net.train()
    for img_data, img_label in train_data:
        if torch.cuda.is_available():
            im = Variable(img_data).cuda()
            label = torch.tensor(img_label, dtype=torch.int64)
            label = Variable(label).cuda()
        else:
            im = Variable(img_data)
            label = torch.tensor(img_label, dtype=torch.int64)
            label = Variable(label)

        # 前向传播
        out = net(im)
```

```
        out = F.log_softmax(out, dim=1)
        loss = criterion(out, label)

        # 反向传播
        # 梯度清零
        optimizer.zero_grad()
        loss.backward()
        # 更新
        optimizer.step()
        _train_loss += loss.item()

        # label_pred 输出的是 21*224*224 的向量，对于每一个点都有 21 个分类的概率
        # 我们取概率值最大的那个下标作为模型预测的标签，然后计算各种评价指标
        label_pred = out.max(dim=1)[1].data.cpu().numpy()
        label_true = label.data.cpu().numpy()
        # label_pred: (8, 224, 224) label_true: (8, 224, 224)
        for lbt, lbp in zip(label_true, label_pred):
            # lbt: (224, 224) lbp: (224, 224)
            acc, acc_cls, mean_iu, fwavacc = label_accuracy_score(lbt, lbp, num_clas
ses)
            _train_acc += acc
            _train_acc_cls += acc_cls
            _train_mean_iu += mean_iu
            _train_fwavacc += fwavacc

    # 记录当前轮的数据
    train_loss.append(_train_loss / len(train_data))
    train_acc.append(_train_acc / len(voc_train))
    train_acc_cls.append(_train_acc_cls)
    train_mean_iu.append(_train_mean_iu / len(voc_train))
    train_fwavacc.append(_train_fwavacc)

    net = net.eval()

    _eval_loss = 0
    _eval_acc = 0
    _eval_acc_cls = 0
    _eval_mean_iu = 0
    _eval_fwavacc = 0

    for img_data, img_label in valid_data:
        if torch.cuda.is_available():
            im = Variable(img_data).cuda()
            label = torch.tensor(img_label, dtype=torch.int64)
            label = Variable(label).cuda()
        else:
            im = Variable(img_data)
            label = torch.tensor(img_label, dtype=torch.int64)
            label = Variable(label)

        # forward
        out = net(im)
        # 对结果进行归一化
        out = F.log_softmax(out, dim=1)
        loss = criterion(out, label)
        _eval_loss += loss.item()

        label_pred = out.max(dim=1)[1].data.cpu().numpy()
        label_true = label.data.cpu().numpy()
        for lbt, lbp in zip(label_true, label_pred):
            acc, acc_cls, mean_iu, fwavacc = label_accuracy_score(lbt, lbp, num_clas
ses)
            _eval_acc += acc
            _eval_acc_cls += acc_cls
            _eval_mean_iu += mean_iu
```

```
            _eval_fwavacc += fwavacc

    # 记录当前轮的数据
    eval_loss.append(_eval_loss / len(valid_data))
    eval_acc.append(_eval_acc / len(voc_test))
    eval_acc_cls.append(_eval_acc_cls)
    eval_mean_iu.append(_eval_mean_iu / len(voc_test))
    eval_fwavacc.append(_eval_fwavacc)

    # 打印当前轮训练的结果
    cur_time = datetime.now()
    h, remainder = divmod((cur_time - prev_time).seconds, 3600)
    # divmod() 函数返回当参数 1 除以参数 2 时包含商和余数的元组。
    m, s = divmod(remainder, 60)
     epoch_str = ('Epoch: {}, Train Loss: {:.5f}, Train Acc: {:.5f}, Train Mean IU: {:.5f}, '
                  'Valid Loss: {:.5f}, Valid Acc: {:.5f}, Valid Mean IU: {:.5f} '.format(
         e, _train_loss / len(train_data), _train_acc / len(voc_train), _train_mean_iu / len(voc_train),
            _eval_loss / len(valid_data), _eval_acc / len(voc_test), _eval_mean_iu / len(voc_test)))
    time_str = 'Time: {:.0f}:{:.0f}:{:.0f}'.format(h, m, s)
    print(epoch_str + time_str)
```

Training process.

```
s: 0.49830, Valid Acc: 0.89420, Valid Mean IU: 0.57202 Time: 0:0:42
Epoch: 27, Train Loss: 0.04224, Train Acc: 0.98345, Train Mean IU: 0.89445, Valid Los
s: 0.49892, Valid Acc: 0.89381, Valid Mean IU: 0.57287 Time: 0:0:42
Epoch: 28, Train Loss: 0.04185, Train Acc: 0.98349, Train Mean IU: 0.89513, Valid Los
s: 0.50622, Valid Acc: 0.89322, Valid Mean IU: 0.57142 Time: 0:0:49
Epoch: 29, Train Loss: 0.04206, Train Acc: 0.98348, Train Mean IU: 0.89375, Valid Los
s: 0.50191, Valid Acc: 0.89426, Valid Mean IU: 0.57371 Time: 0:0:48
Epoch: 30, Train Loss: 0.04165, Train Acc: 0.98360, Train Mean IU: 0.89621, Valid Los
s: 0.49704, Valid Acc: 0.89346, Valid Mean IU: 0.56481 Time: 0:0:49
Epoch: 31, Train Loss: 0.04201, Train Acc: 0.98344, Train Mean IU: 0.89478, Valid Los
s: 0.49330, Valid Acc: 0.89418, Valid Mean IU: 0.57040 Time: 0:0:48
Epoch: 32, Train Loss: 0.04198, Train Acc: 0.98345, Train Mean IU: 0.89484, Valid Los
s: 0.49907, Valid Acc: 0.89426, Valid Mean IU: 0.57120 Time: 0:0:48
Epoch: 33, Train Loss: 0.04201, Train Acc: 0.98351, Train Mean IU: 0.89314, Valid Los
s: 0.49317, Valid Acc: 0.89445, Valid Mean IU: 0.57292 Time: 0:0:47
Epoch: 34, Train Loss: 0.04163, Train Acc: 0.98364, Train Mean IU: 0.89520, Valid Los
s: 0.49903, Valid Acc: 0.89384, Valid Mean IU: 0.57186 Time: 0:0:48
Epoch: 35, Train Loss: 0.04166, Train Acc: 0.98360, Train Mean IU: 0.89532, Valid Los
s: 0.49857, Valid Acc: 0.89341, Valid Mean IU: 0.56861 Time: 0:0:48
```

3-F

Saving model training results and plotting curves.

```
PATH = "./model/fcn-resnet34.pth"
torch.save(net.state_dict(), PATH)
plt.plot(np.array(train_loss))
plt.title("Train Loss")
plt.figure()
plt.plot(np.array(train_acc))
plt.title("Pixel Accuracy")
plt.figure()
plt.plot(np.array(train_mean_iu))
plt.title("Mean IU")
```

Test training results.

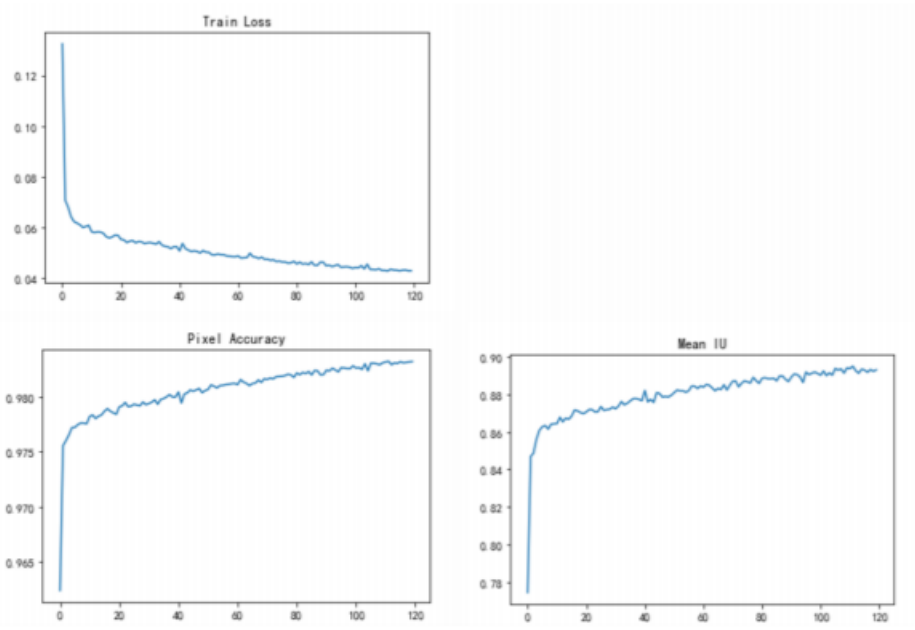
```
print("Eval Pixel Accuracy: ", np.array(eval_acc)[-1])
print("Eval Mean IU: ", np.array(eval_mean_iu)[-1])
```

```
Eval Pixel Accuracy:  0.8944751938358038
Eval Mean IU:  0.5766601620395866
```

3-G

3.3 Visualisation analysis

Plotting training data evaluation curves.



3-H

The accuracy rate shows that the 89.4% accuracy of our model is closer to the data in the paper.

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	90.3	75.9	62.7	83.2

3-I

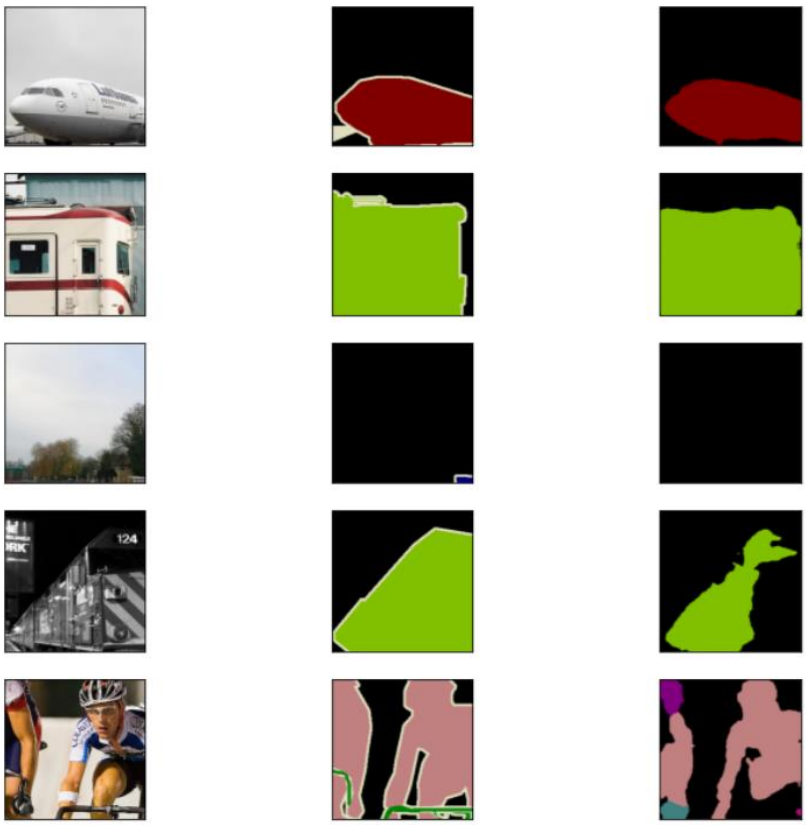
Visualisation of the test set prediction results.

```
# 加载模型
net = fcn(num_classes)
PATH = "./model/fcn-resnet34.pth"
net.load_state_dict(torch.load(PATH))
if torch.cuda.is_available():
    net = net.cuda()
cm = np.array(colormap).astype('uint8')
size = 224
num_image = 10
def predict(img, label):  # 预测结果
    img = Variable(img.unsqueeze(0)).cuda()
    out = net(img)
    pred = out.max(1)[1].squeeze().cpu().data.numpy()
    # 将 pred 的分类值，转换成各个分类对应的 RGB 值
```

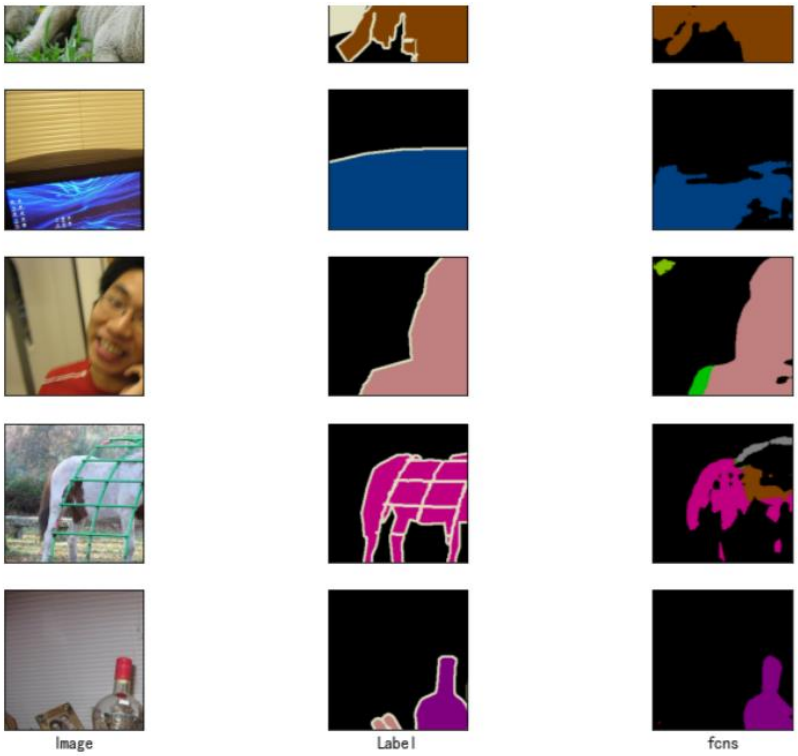
```
    pred = cm[pred]
    # 将 numpy 转换成 PIL 对象
    pred = Image.fromarray(pred)
    label = cm[label.numpy()]
    return pred, label
_, figs = plt.subplots(num_image, 3, figsize=(12, 22))
for i in range(num_image):
    img_data, img_label = voc_test[i]
    pred, label = predict(img_data, img_label)
    img_data = Image.open(voc_test.data_list[i])
    img_label = Image.open(voc_test.label_list[i]).convert("RGB")
    img_data, img_label = crop_image(img_data, img_label, 224, 224)
    figs[i, 0].imshow(img_data)  # 原始图片
    figs[i, 0].axes.get_xaxis().set_visible(False)  # 去掉 x 轴
    figs[i, 0].axes.get_yaxis().set_visible(False)  # 去掉 y 轴
    figs[i, 1].imshow(img_label)  # 标签
    figs[i, 1].axes.get_xaxis().set_visible(False)  # 去掉 x 轴
    figs[i, 1].axes.get_yaxis().set_visible(False)  # 去掉 y 轴
    figs[i, 2].imshow(pred)  # 模型输出结果
    figs[i, 2].axes.get_xaxis().set_visible(False)  # 去掉 x 轴
    figs[i, 2].axes.get_yaxis().set_visible(False)  # 去掉 y 轴

# 在最后一行图片下面添加标题
figs[num_image - 1, 0].set_title("Image", y=-0.2)
figs[num_image - 1, 1].set_title("Label", y=-0.2)
figs[num_image - 1, 2].set_title("fcns", y=-0.2)
```

```
Text(0.5, -0.2, 'fcns')
```



3-J

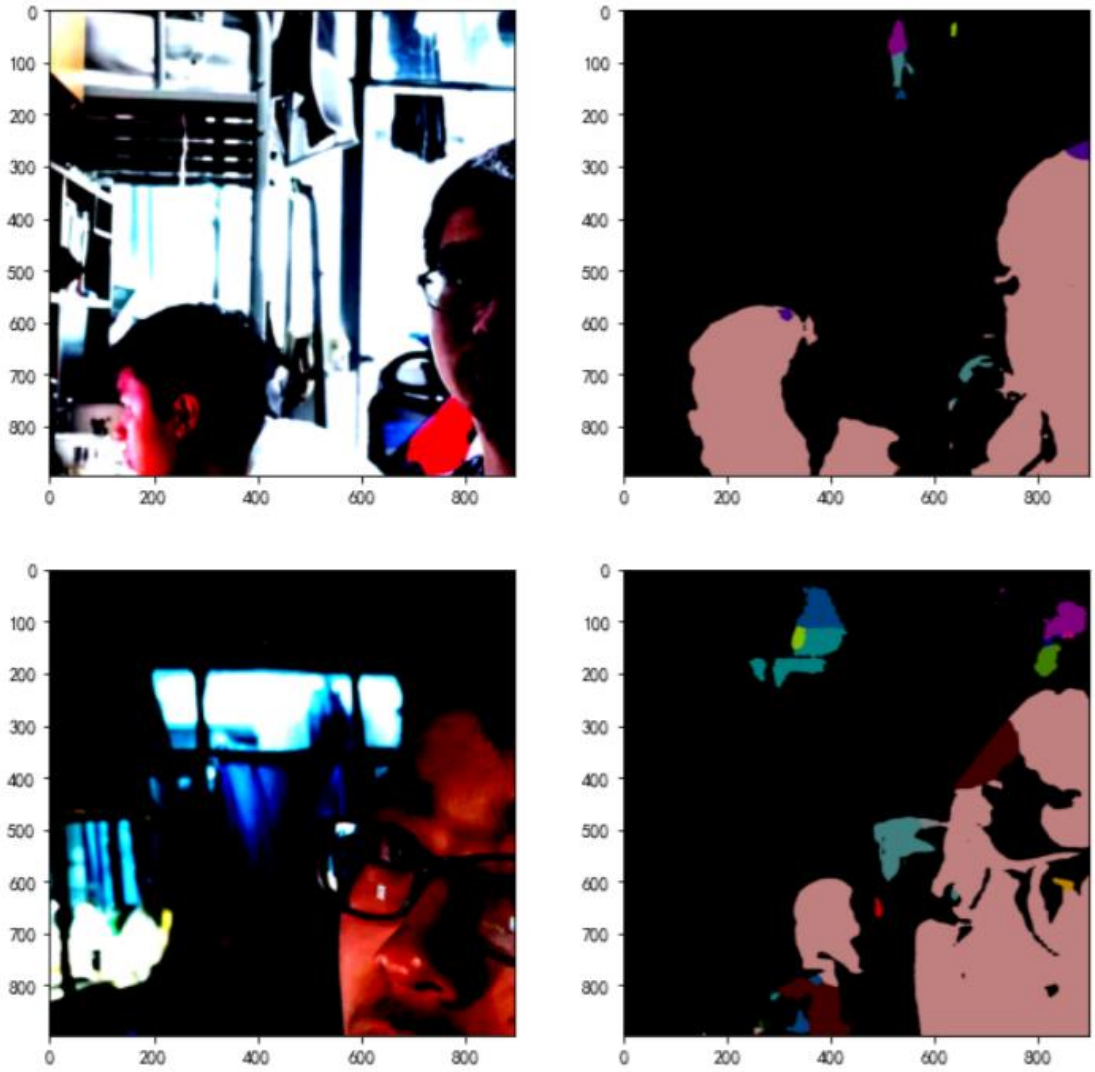


3-K

Testing of model segmentation using everyday photographs.

```
MY_DATA_DIR = '.\\data'
print(os.listdir(MY_DATA_DIR+'\\Mydata')[:10])
Is = os.listdir(MY_DATA_DIR+'\\Mydata')
data = [os.path.join(MY_DATA_DIR, 'Mydata', i) for i in Is]
height = 896
width = 896
def My_Image(data, height, width):
    img = []
    for im in data:
        if (Image.open(im).size[1] >= height and Image.open(im).size[0] >= width):
            img.append(im)
    img = [Image.open(i) for i in img]
    box = (0, 0, width, height)
    img = [i.crop(box) for i in img]
    im_tfs = tfs.Compose([
            tfs.ToTensor(),
            tfs.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225])
    ])
    img = [im_tfs(i) for i in img]
    return img
def predict_(img):
    img = Variable(img.unsqueeze(0)).cuda()
    out = net(img)
    pred = out.max(1)[1].squeeze().cpu().data.numpy()
    pred = cm[pred]
    pred = Image.fromarray(pred)
    return pred
_, figs = plt.subplots(len(data), 2, figsize=(10, 10))
for i in range(len(data)):
    pred = predict_(img[i])
    figs[i, 0].imshow(img[i].data.numpy().transpose(1,2,0))
```

```
34.     figs[i, 1].imshow(pred)
```



3-L

3.4 Experimental experience

Semantic segmentation, both textual and image semantic segmentation, has been a remarkable achievement in the field of artificial intelligence in the last decade, not only in many areas of industry, but also in our understanding of the extraction process and the intrinsic composition of semantic information. In real life we recognise an 'object' as if we are naturally able to extract information about its boundaries and, in the process of further understanding, to clearly distinguish the different details of the subject to which it belongs. The success of machine learning has led us to reflect on this phenomenon and to realise that it is possible to attach this phenomenon (behaviour) to a machine through the language of mathematics, and that semantic segmentation of images at pixel level is undoubtedly an important part of what can be achieved in the future with general or strong artificial intelligence.
FCN, the pioneer of pixel-level semantic segmentation of images, has pioneered the introduction of deconvolution into the traditional convolutional neural network-full-connected neural network model of image segmentation processing, making it possible not only to achieve unlimited scale of the input image, but also to give new ideas for semantic edge processing of images. We have learnt about the fundamentals of image semantic segmentation, the effect of perceptual field on network structure, residual networks and other related knowledge, and become proficient in the use of frameworks such as pytorch, which provides a basis for further study. The pixel accuracy predicted by the model in this project was maintained at around 89.4%, and we collected some photos in our daily life, and the test can see that the model has better semantic segmentation results in a variety of environments.

However, the FCN-based semantic segmentation is still unable to achieve fine detail in dark areas and edge details. Future improvements are expected in terms of more detailed tuning of the network and further work on semantic segmentation algorithms such as u-net and segnet.

References

[1] Jonathan Long, Evan Shelhamer, Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. CVPR2015 paper. 2015 3431-3440.
[2] 李梦怡，朱定局．基于全卷积网络的图像语义分割方法综述．华南师范大学计算机学院．2019
[3] 张建，基于深度学习的图像语义分割方法．电子科技大学．2018.
[4] 章琳,袁非牛,张文睿,曾夏玲．全卷积神经网络研究综述．计算机工程与应用．2020(01)
[5] 刘健,袁谦,吴广,喻晓．卷积神经网络综述．计算机时代．2018(11)

Group division of labour

张梓琦: Theoretical studies, Paper preparation(1/3)
关乐平: Data set collection and testing,PPT Production,(1/3)
刘鸿嘉: Course defence, Code implementation(1/3)