

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Уральский федеральный университет  
имени первого Президента России Б.Н. Ельцина»  
Институт радиоэлектроники и информационных технологий – РТФ  
Школа профессионального и академического образования

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД ГЭК

Директор ШПиАО  
 Д.В. Денисов  
(подпись) (Ф.И.О.)  
« 03 » июня 2024 г.

## ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

RESEARCH OF LEFT VENTRICULAR SEGMENTATION ON TWO-  
DIMENSIONAL ULTRASOUND IMAGES BASED ON DIFFERENT DEEP  
LEARNING MODELS


Научный руководитель: Ронкин Михаил Владимирович  
к.т.н.

  
подпись

Нормоконтролер: Огуренко Егор Владимирович

  
подпись

Студент группы: РИМ-220901 Ли Баохэн

  
подпись

Екатеринбург  
2024

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования  
**«Уральский федеральный университет  
имени первого Президента России Б.Н. Ельцина»**

Институт радиоэлектроники и информационных технологий – РТФ  
Школа профессионального и академического образования  
Направление подготовки 09.04.02 Информационные системы и технологии  
Образовательная программа 09.04.02/33.09 Прикладной искусственный интеллект

### ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студент Ли Баохэн группы РИМ-220901

(фамилия, имя, отчество)

**1. Тема выпускной квалификационной работы**

Research of Left Ventricular Segmentation on Two-dimensional Ultrasound Images Based on Different Deep Learning Models

Утверждена распоряжением по институту от «4» декабря 2023 г. № 33.02-05/298

**2. Научный руководитель**

Ронкин Михаил Владимирович, доцент кафедры информационных технологий и систем управления, к.т.н.

(Ф.И.О., должность, ученая степень, ученое звание)


**3. Исходные данные к работе**

Датасет по теме магистерской диссертации


**4. Перечень демонстрационных материалов**

Презентация

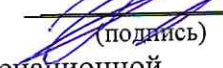
**5. Календарный план**

№ п/п	Наименование этапов выполнения работы	Срок выполнения этапов работы	Отметка о выполнении
1.	<i>1 раздел (глава)</i>	до 23.03.2024 г.	
2.	<i>2 раздел (глава)</i>	до 29.04.2024 г.	
3.	<i>3–4 раздел (глава)</i>	до 20.05.2024 г.	
4.	<i>ВКР в целом</i>	до 24.05.2024 г.	

Научный руководитель М.В. Ронкин  
Ф.И.О.

  
(подпись)

Студент задание принял к исполнению 12.02.24  
дата

  
(подпись)

**6. Допустить** ФИО к защите выпускной квалификационной работы в экзаменационной комиссии

Директор ШПиАО



Д.В. Денисов

## ABSTRACT

In recent years, the prevalence of cardiovascular diseases, as well as the mortality rate is increasing, which has seriously threatened human health, which requires doctors to diagnose cardiovascular diseases early to gain time for patients' later treatment, and the segmentation results of left ventricular ultrasound images can assist doctors in the diagnosis of cardiovascular diseases, but the left ventricular ultrasound images have the characteristics of strong noise, weak edges and complex tissue structure, which makes the image segmentation difficult, low efficiency and poor precision. One of the most important steps in estimating the health of the heart is the tracking and segmentation of the left ventricular (LV) endocardial border from EchoCG, which is used for measuring the ejection fraction and assessing the regional wall motion. The disadvantage of these methods is the necessity to apply image processing manually or in a semi-automatic mode, which requires special knowledge and skills. As a result, the issue of an automatic tracking and segmentation of the LV on EchoCG-images is an actual and practical problem. In my project, the ability of fully trained Deep Learning Models U-Net, U-Net++, MANet, LinkNet, FPN, PSPNet, PAN, DeepLabv3 and DeepLabv3+ to automatically identify the left ventricular region is explored. At the same time, in the U-Net, U-Net++, MANet, LinkNet, PSPNet, PAN, FPN, DeepLabv3 and DeepLabv3+ architectures, the encoder modules were then sequentially replaced with ResNet18, ResNet34, ResNet5, ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3, EfficientNet-b5, EfficientNet-b7 and MobileNetv2, and ImageNet was used as the pre-training weights; The addition of backbones to the model architecture leads to higher segmentation accuracy compared to the original model. Within the same model architecture, EfficientNet as the encoder achieves better segmentation results, with EfficientNet-b3 performing the best. Similarly, within the ResNet series, ResNet34 performs better. In the segmentation model of this experiment, DeepLabv3+ shows superior performance. This indicates that in the model architecture of this experiment, integrating ResNet34 and EfficientNet-b3 modules as encoders can effectively and feasibly automate the recognition of the endocardial boundary of the left ventricle in ultrasound images. Furthermore, data augmentation will also enhance the model's segmentation accuracy to a certain extent.

**Keywords:** Left ventricular segmentation, Deep Learning, Backbone, Data augmentation, U-Net, DeepLabv3, MANet, LinkNet, EfficientNet, FPN, PSPNet, PAN, ResNet, MobileNetv2

# CONTENTS

Abstract .....	3
Contents .....	4
Abbreviations .....	6
List of Tables .....	7
List of Figures .....	8
Introduction .....	9
Background and Importance of the Research .....	9
The Current State of Image Segmentation Research .....	12
Materials and methods .....	15
1.Dataset .....	15
2.The Evaluation Metrics .....	16
3.Image Preprocessing .....	17
3.1 Image grayscale .....	18
3.2 Image denoising .....	18
3.3 Image orientation cropping .....	19
3.4 Data Augmentation .....	19
3.5 Data exploration and initial visualization .....	20
3.6 Save and split the visualized dataset .....	21
4. Image Segmentation Models .....	22
4.1 Deep learning .....	22
4.2 Hyperparameters .....	24
4.3 Architectures .....	27
4.4 Encoder .....	36
4.5 Network Structure Design .....	39
Result .....	41

Discussion .....	49
Conclusion .....	50
References .....	52
Appendices .....	54
Appendix 1 .....	54
Appendix 2 .....	55

## ABBREVIATIONS

LV	Left Ventricular
RV	Right Ventricle
LA	Left Atrium
RA	Right Atrium
CNN	Convolutional Neural Network
CVD	Cardiovascular Disease
MRI	Machine Learning
US	Ultrasound
CAMUS	Cardiac Acquisitions for Multi-structure Ultrasound Segmentation
FPN	Feature Pyramid Network
PAN	Pyramid Attention Network
PSPNet	Pyramid Scene Parsing Network
MANet	Multi-scale Attention Network
ResNet	The Residual Neural Network

## LIST OF TABLES

Table 1: The results of U-Net with different backbones training in initial data and augmented data.....	43
Table 2: The results of U-Net++ with different backbones training in initial data and augmented data.....	43
Table 3: The results of MANet with different backbones training in initial data and augmented data.....	44
Table 4: The results of LinkNet with different backbones training in initial data and augmented data.....	44
Table 6: The results of PSPNet with different backbones training in initial data and augmented data.....	45
Table 7: The results of PAN with different backbones training in initial data and augmented data.....	46
Table 9: The results of DeepLabv3+ with different backbones training in initial data and augmented data.....	47
Table 10-1: The result of architectures with different backbones training in initial data and augmented data.....	47
Table 10-2: The result of architectures with different backbones training in initial data and augmented data.....	48

## LIST OF FIGURES

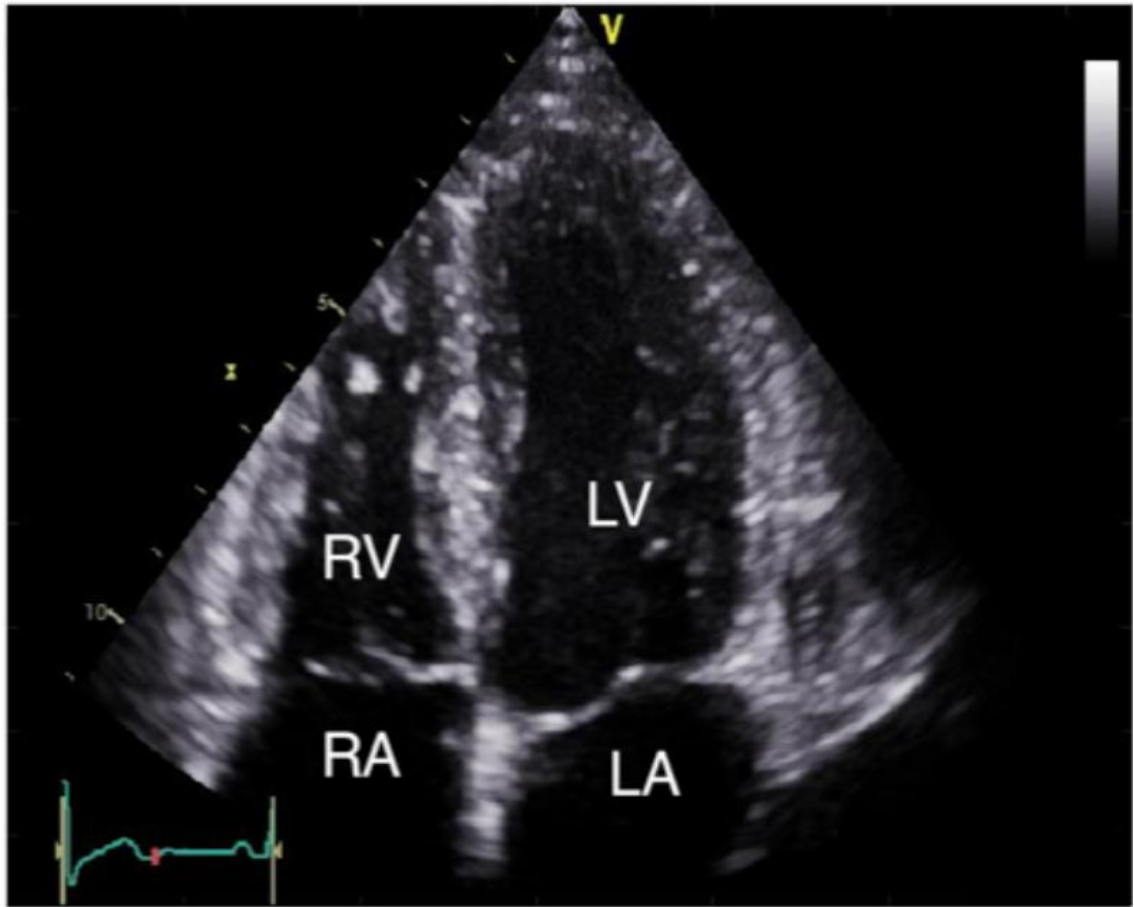
Fig.1	The image shows a 4-chamber view ultrasound image .....	10
Fig.2	An example of a left ventricle ultrasound image .....	11
Fig.3	Dataset sructure of one patient .....	15
Fig.4	Segmentation results comparison venn diagram .....	16
Fig.5	Patient001 original image visualization .....	20
Fig.6	Patient001 mask visualization .....	20
Fig.7	Random visualization samples 1-3 in the visualization test set .....	21
Fig.8	Random visualization samples 4-5 in the visualization test set .....	22
Fig.9	A basic artificial neural network .....	23
Fig.10	The figure shows the different activation functions used for the network .....	24
Fig.11	Learning rate effect .....	25
Fig.12	Overfitting analysis .....	26
Fig.13	Complete network and implemented dropout network .....	27
Fig.14	U-Net architecture .....	28
Fig.15	U-Net++architecture .....	29
Fig.16	The total architecture of MA-Net. ....	30
Fig.17	LinkNet Architecture. ....	31
Fig.18	Different pyramid structures: (d) shows the Feature Pyramid Network .....	32
Fig.19	Overview of our proposed PSPNet .....	33
Fig.20	Overview of the Pyramid Attention Network .....	34
Fig.21	Cascaded modules without and with atrous convolution. ....	35
Fig.22	Parallel modules with atrous convolution (ASPP),augmented with image-level features. ...	35
Fig.23	Our proposed DeepLabv3+ extends DeepLabv3 .....	36
Fig.24	Residual learning: a building block. ....	37
Fig.25	Architectures for ImageNet. ....	37
Fig.26	Model Scaling. ....	38
Fig.27	MobileNetv2 .....	39
Fig.28	The visualization of partial model segmentation results .....	42



# INTRODUCTION

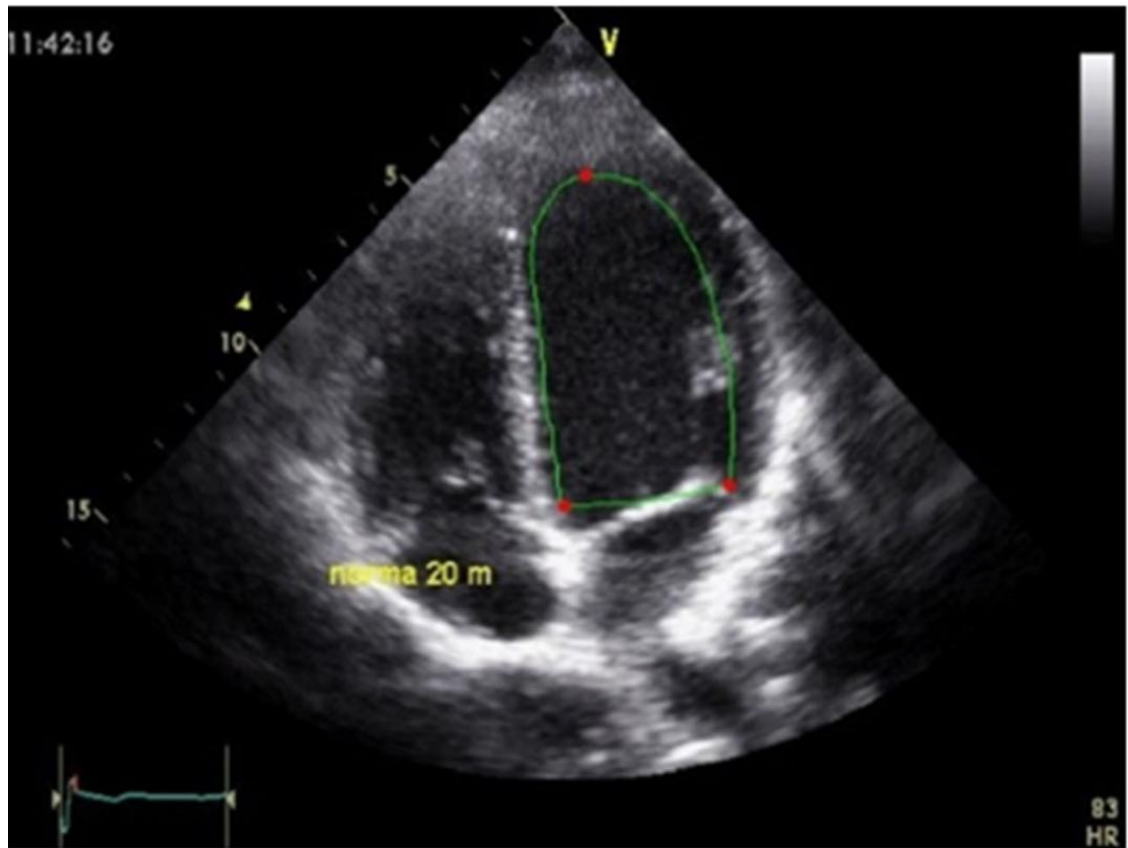
## Background and Importance of the Research

At present, cardiovascular disease (CVD)<sup>[1]</sup> has achieved certain results in prevention and treatment, but in general, the prevalence and mortality of the population is still on the rise and has become the number one threat to human health. The number of people suffering from cardiovascular diseases is about 300 million per year, of which hypertension accounts for 86.94% of the total number of people, other types of diseases account for 13.06%, including congenital heart disease (0.69%), rheumatic heart disease (0.86%), heart failure (1.55%), pulmonary heart disease (1.72%), coronary heart disease (3.79%), stroke (4.45%), etc. The mortality rate of cardiovascular diseases (accounting for about 40% of disease mortality) is also higher than that of other high-risk diseases such as tumors, and the mortality rate in poor rural areas is much higher than that in urban areas. The main reason why the mortality rate of cardiovascular diseases is so high is that cardiovascular diseases are difficult to be diagnosed. Generally, cardiovascular diseases have clinical manifestations later in the entire course of the disease, and it is difficult for doctors to detect the disease at the early stage of the disease, resulting in cardiovascular diseases not being treated in a timely manner. However, in the early stage of the disease process without any clinical manifestations or subhealthy state, the heart, blood vessels and their vascular blood flow and other parameters have already undergone lesions, if doctors can detect such lesions early and accurately distinguish the three states of healthy, subhealthy and diseased in a timely manner and prescribe the right medicine according to the disease level, it can make the patients in subhealthy easy to prevent and treat, and ultimately effectively reduce the cardiovascular disease. The prevalence and death rate of cardiovascular diseases can be effectively reduced. There are many cardiac imaging devices used to diagnose cardiovascular diseases at this stage, including ultrasound, computed tomography, coronary angiography, and cardiac MRI<sup>[2]</sup>. Compared to other medical imaging techniques, ultrasound images are often used to diagnose cardiovascular disease because they are non-invasive, free of ionizing radiation, relatively inexpensive, and simple to perform. Typically, we can see ultrasound images of the four chambers, as shown in **Figure 1**.



*Fig.1 The image shows a 4-chamber view ultrasound image where left ventricle (LV), right ventricle (RV), left atrium (LA) and right atrium (RA) are visible. A 2-chamber view only shows the LV and LA.*

The most important step in assessing cardiac health in ultrasound images is the measurement of ejection fraction and assessment of regional wall motion<sup>[3]</sup>, based on the LV endocardial border, which is usually obtained by cardiologists through manual mode or semi-automatic mode segmentation, where the expert defines the segmentation region through three designated base points (mitral and apical on the endocardium), as shown in **Figure 2**, with the three red dots being the three base points marked by the expert. The green curve refers to the endocardial border of the left ventricle manually segmented by the expert.



*Fig.2 An example of a left ventricle ultrasound image. The three red dots denote the three cardinal points (the mitral valve annulus and the apex) calibrated by experts, and the green curve indicates the expert marking the left ventricular endocardium boundary*

Manual or semi-automatic segmentation<sup>[4]</sup> of left ventricular ultrasound images has the following problems: first, it is a tedious and time-consuming task, and can only be performed by professional clinicians. The segmentation process is easily influenced by subjective factors, resulting in large segmentation differences and inaccurate segmentation results; second, the repeatability of segmentation is poor, and it must be re-segmented for different cardiac patients. To address the above problems, it is important to study how to quickly and accurately obtain the endocardial border of left ventricular ultrasound images for research and application in related medical fields. Medical image segmentation is the basis of medical image processing and analysis, and the good or bad segmentation performance directly affects the quality of subsequent tasks, such as quantitative estimation of tissue or organ volume, localization of diseased tissues, three-dimensional reconstruction, and other tasks, which all depend on the quality of medical image segmentation, while influenced by the ultrasound image imaging principle and the complex heart structure, the following challenges exist in left ventricular ultrasound image segmentation: (1) due to the presence of tissue structures such as trabeculae or papillary muscles inside the blood pool of the ventricle,

their grayscale values are similar to those of the myocardium, which easily form false edges that do not belong to the endocardium; (2) the small difference in grayscale between the surrounding tissues of the epicardium (fat or lung, etc.) and the ventricle, resulting in difficulty in obtaining the exact location of the left ventricle<sup>[5]</sup> during segmentation; (3) the influence of the atria and the ventricle, which causes irregularities in the fundus slices of the left ventricle, and different batches of LV ultrasound images also differ in shape; (4) the resolution of ultrasound images is generally low, and the noise generated by the imaging equipment affects the image quality, which ultimately makes it difficult to segment the small-sized tissue of the ventricle at the apex of the heart.

### **The Current State of Image Segmentation Research**

More and more researchers are performing segmentation of medical images, and the segmentation of the left ventricle of the heart has been particularly studied. The traditional method for LV segmentation is to extract the contour edges of the image based on the image features by establishing an energy function. Segmentation algorithms are mainly classified into image-driven models, active contour models, active shape models, active appearance models, and atlas-based methods. The image-driven model is generally a general term for image edge-based, thresholding, region growing, and pixel classification. Since this method is particularly sensitive to noise and difficult to identify weak edges, it is not effective for US LV segmentation. For US small data sets, the segmentation results based on mapping are poor, especially the segmentation error is very large in images with large individual differences. Active contouring is widely used in contour detection or image segmentation, and is a series of algorithms derived from the Snake algorithm proposed by Kass et al. (1998)<sup>[6]</sup>. However, this algorithm has poor reliability and robustness due to the high computational cost, slow speed, and irregular parameter adjustment.

The method based on deep learning is outstanding in the field of computer vision, especially the Convolutional Neural Networks (CNN) can extract pixel-level image features. The segmentation network structure is simple and fast, but the error of weak edge recognition is large and the accuracy is low. In 2012, AlexNet<sup>[7]</sup> leveraged CNN architecture to achieve unprecedented recognition heights on the ILSVRC-2012 dataset, marking the heyday of convolutional neural networks. Many CNN-based network architectures have since surpassed human-level recognition accuracy on the ImageNet dataset. From 2016 to 2018, Chen et al. introduced segmentation network models such as DeepLabv1<sup>[8]</sup>, DeepLabv2<sup>[9]</sup>, DeepLabv3<sup>[10]</sup>, and DeepLabv3+<sup>[11]</sup>, which utilize dilated convolutions combined with pyramid spatial pooling for multiscale processing, followed by post-

processing with fully connected conditional random fields to achieve image segmentation. Building upon this, Zhao et al. proposed PSPNet<sup>[12]</sup>, which improves the ResNet<sup>[13]</sup> network structure using dilated convolutions and adds a pyramid pooling module to aggregate background information. In addition to the main branch, an auxiliary branch is added, enhancing the network's segmentation performance, leading to satisfactory segmentation results.

In 2015, Long et al. introduced Fully Convolutional Networks (FCN<sup>[14]</sup>), which first extended the encoder-decoder structure into an end-to-end network for semantic segmentation and employed skip connections (via addition operation) to improve upsampling coarseness, achieving good segmentation results. Addressing the challenge of training networks on small datasets, in 2015, Ronneberger et al. proposed U-Net<sup>[15]</sup>, where the decoder symmetrically mirrors the encoder structure with multiple consecutive upsampling layers. The skip connection mechanism is replaced with concatenation, enabling training on small datasets, making it commonly used in medical image segmentation. Many segmentation algorithms are built upon this network, with a notable example being U-Net++<sup>[16]</sup>, which introduces nested skip connection paths to improve information flow, further enhancing the accuracy and efficiency of semantic segmentation.

MANet<sup>[17]</sup> integrates multiple attention mechanisms to refine semantic segmentation boundaries, thereby improving segmentation result accuracy. In 2017, Chaurasia et al. introduced LinkNet<sup>[18]</sup>, with its lightweight structure and skip connections, performing exceptionally well in resource-constrained environments. Concurrently, Lin et al. proposed FPN<sup>[19]</sup>, which effectively enhances multiscale feature representation through the establishment of a feature pyramid network, thereby improving the performance of object detection and semantic segmentation. The introduction of FPN has significantly advanced the handling of different scales of feature representation, resulting in better performance in object detection and semantic segmentation tasks. The Pyramid Attention Network (PAN<sup>[20]</sup>) is used to explore the impact of global contextual information on semantic segmentation. It combines attention mechanisms and spatial pyramid structures to extract precise and dense features. The introduction of the Feature Pyramid Attention module is applied to high-level features, utilizing spatial pyramid attention structures combined with global contextual information to learn better feature representations. The introduction of the Global Attention Upsample module guides spatial information selection for low-level features at each decoder layer.

In 2015, He et al. introduced the ResNet<sup>[13]</sup> series, including ResNet18, ResNet34, ResNet101, among others. This series of models employ residual connections, enabling training of very deep

neural networks and achieving breakthrough performance in tasks like image classification. In 2018, Sandler et al. proposed MobileNetV2<sup>[21]</sup>, which introduced depthwise separable convolutions to achieve lightweight and efficient feature extraction in resource-constrained environments like mobile devices. In 2019, Tan et al. proposed the EfficientNet<sup>[22]</sup> series, including EfficientNet-b0 to EfficientNet-b7, achieving optimal performance in resource-constrained environments by simultaneously adjusting the network's width, depth, and resolution. This class of algorithms improves the coarseness of upsampling by jump connections. Experiments have shown that jump junction is effective in recovering detailed information of the target object and can perform effective segmentation even in the case of complex backgrounds or objects partially occluded, such as Mask-RCNN<sup>[23]</sup>.

Currently, natural image segmentation has reached a satisfactory level, but these models are still not well suited for the rigorous segmentation of medical images. Medical images generally have lower resolution than natural images, training data sets are generally smaller, and medical image segmentation accuracy requirements are much higher than natural images. Accurate segmentation masks may not be critical in natural images, but in medical images even image edge segmentation errors can affect the user's clinical experience.

In summary, various algorithms have achieved certain results in image segmentation tasks, but there are still three problems: 1) Although the segmentation methods based on depth learning have made significant progress compared with the traditional segmentation methods, they have lost some useful level details in the original image in the process of enhancing the receptive field; 2) It is insensitive to weak edges in US, resulting in large edge segmentation error; 3) Due to the low resolution, high noise and the influence of other tissues and organs on the segmentation results of US images, the segmentation accuracy is poor.

# MATERIALS AND METHODS

## 1.Dataset

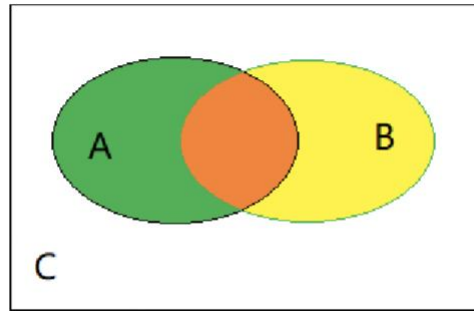
The CAMUS (Cardiac Acquisitions for Multi-structure Ultrasound Segmentation) challenge has been set out to improve segmentation of 2D echocardiographic images. The authors released a publicly available, 500 patient's dataset to allow for comparable results across heart segmentation research. The contents of a single patient's dataset are shown in **Figure 3**. After preprocessing, the dataset is obtained and divided into subsets, with 80% for the training set, 10% for the validation set, and 10% for the test set.



*Fig.3 Dataset structure of one patient*

## 2.The Evaluation Metrics

If you need to objectively evaluate the segmentation results of an image, you'll need to define evaluation metrics. For objective assessment, the selected metrics are Intersection over Union (IoU), Dice Similarity Coefficient (Dice), and Recall. To facilitate the explanation of the selected metrics, it's more intuitive to represent them graphically. Draw a Venn diagram comparing the segmentation results as shown in **Figure 4**. Set A represents the segmentation results obtained by the network automatically, while set B represents the expert manually segmented results (ground truth).



*Fig.4 Segmentation results comparison venn diagram*

IoU, short for Intersection over Union, refers to the ratio of the overlap area between the automatically segmented contour (predicted contour) and the expert manually segmented contour (ground truth contour) to the total area encompassed by both contours. It can be represented by Equation 2.1, where a higher value indicates better performance, with 1 indicating perfect overlap.

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \times 100\% \quad 2.1$$

The Dice coefficient is an evaluation metric for the similarity between the predicted contour and the ground truth contour. The formula for the Dice coefficient is shown in Equation 2.2. The Dice coefficient ranges from 0 to 1, with higher values indicating better segmentation performance.

$$Dice(A, B) = \frac{2 \cdot |A \cap B|}{|A| + |B|} \times 100\% \quad 2.2$$

The accuracy, for a given test dataset, is defined as the ratio of the number of samples correctly classified by the classifier to the total number of samples. It can be expressed by Equation 2.3, where TP denotes true positives (samples that are positive and predicted as positive), FP denotes false positives (samples that are negative but predicted as positive), TN denotes true negatives (samples that are negative and predicted as negative), and FN denotes false negatives (samples that are positive but predicted as negative).



$$\text{Acc} = \frac{TP + TN}{TP + FP + TN + FN} \quad 2.3$$

Recall, commonly used as a performance metric in information retrieval and machine learning, measures the proportion of actual positive samples that are successfully predicted as positive by the model. A higher recall indicates that the model can identify more true positive instances, meaning it can cover as many actual positive instances as possible. It can be expressed by Equation 2.4.

$$\text{Acc} = \frac{TP}{TP + FN} \quad 2.4$$

The "mIoU" appearing in this project represents the average value of the evaluation metric on the test set.

### 3. Image Preprocessing

Given the non-invasive and convenient nature of ultrasound imaging, echocardiography imaging is currently the most widely used clinical tool for the examination of cardiac diseases. The imaging principle is mainly based on the physical properties of ultrasound and the reflection characteristics of different tissues to ultrasound, and finally displays the ultrasound echo signal in the form of a graph. Different tissues have different acoustic impedance and attenuation coefficients for ultrasound. As ultrasound waves propagate through the body, different reflections of ultrasound waves are generated when passing through different tissues. Based on these reflected ultrasounds (ultrasound echoes), they are computationally converted to grayscale information on the image, which in turn displays the ultrasound image. Echocardiographic imaging is divided into 2D echocardiographic imaging and 3D.

2D echocardiography is performed by scanning along a series of scan beams within a sector, based on the physician's manual adjustment of the ultrasound probe position to obtain an ultrasound image of a fixed section. Clinically frequently used views include: short-axis views of the aortic root (including left atrium, right atrium, aorta, ascending aorta, and right ventricular outflow tract, based on which aortic valve-related disease is diagnosed); four-chamber views (including right and left ventricle, right and left atrium, based on which atrial septal lesions and chamber size and ventricular disease are diagnosed); and five-chamber views (including right and left (including right and left ventricle, left and right ventricle, left ventricular outflow tract, mainly based on the coordination of ventricular and atrial motion in this view); ventricular short-axis view (including left ventricle and right ventricle, mainly based on the diagnosis of ventricular lesions in this view); ventricular long-axis view (including left ventricle and left atrium, mainly to discriminate whether the pumping function of the left ventricle is normal in this view). Compared to 3D echocardiography, 2D

echocardiography has higher resolution and lower data dimensionality, and each view has its own specific disease diagnostic advantages.

### 3.1 Image grayscale

The process of converting a color image into a grayscale image by some algorithm in digital image processing is called image grayscale. The color image is made by superimposing three color components, R, G, and B. Each color component takes values in the range of 0 to 255, so the color image constitutes a color variation of more than 16 million kinds ( $255 \times 255 \times 255$ ). Grayscale image can be regarded as a special case of color image, which contains only one channel and its pixel values vary from 0 to 255, and the semantic information of the image is retained before and after image grayscale, so image grayscale can reduce the computation of segmentation network and improve the segmentation efficiency. There are many algorithms for image grayscale. According to the different importance of the three color components in the color image, the color image is weighted and averaged to find the grayscale image, and its formula is shown in Equation 3.1.

$$I(x, y) = a \cdot R(x, y) + b \cdot G(x, y) + c \cdot B(x, y) \quad 3.1$$

where  $I(x,y)$ ,  $R(x,y)$ ,  $G(x,y)$ , and  $B(x,y)$  represent the coordinates of the three channels (red, green, and blue) and their pixel values for grayscale and color images, respectively;  $a$ ,  $b$ , and  $c$  are the weights, and of the three color components, the human eye is generally the most sensitive to green and the least sensitive to blue.

In order to make the grayscale image available for display on the computer, the image is normalized to the maximum and minimum so that the grayscale value of the image ranges from 0 to 255, and the formula for the maximum and minimum normalization is shown in Equation 3.2.

$$I^* = \frac{I - \min}{\max - \min} \times 255 \quad 3.2$$

Where  $I$  represents the pixel value of a pixel point in the original image,  $\min$  and  $\max$  represent the minimum and maximum values of pixels in the original image, respectively, and  $I^*$  represents the maximum and minimum normalized pixel value.

### 3.2 Image denoising

As ultrasound images are affected by imaging equipment as well as lighting changes and other factors, they are easily interspersed with many types of noise, and these noise errors are transmitted cumulatively in the calculation, affecting the training of the classifier as well as the prediction, so it is necessary to denoise the images before they are input to the segmentation network. Noise can be generally divided into two categories: (1) additive noise: additive noise is characterized by the

image signal (ideal noiseless image) and noise signal are independent and uncorrelated, that is, the image with noise is superimposed from the image signal and noise signal, such as channel noise in the image transmission process; (2) multiplicative noise: multiplicative noise is characterized by the image signal and noise signal correlation, noise by modulation of the image signal itself.

There are many filtering algorithms used to eliminate noise, but the filtering operation blurs the image edges while eliminating noise, and the choice of denoising method will directly affect the final segmentation effect. For the impulse noise (salt-and-pepper noise) present in the left ventricular ultrasound image, the median filter is used for denoising, which belongs to a statistical ranking filter in nonlinear space. The response of the filter is based on the ranking of pixels in the image region (original pixels and their pixel neighbors) surrounded by the filter, and the median value of the pixels is substituted for the original pixel value, thus removing the impulse noise from the image.

### **3.3 Image orientation cropping**

To improve the segmentation speed and accuracy of the left ventricular ultrasound image, the traditional threshold segmentation method is used to locate the left ventricular contour in the ultrasound image and perform image cropping to reduce the recognition range of the subsequent neural network. The image is first binarized to distinguish the background, left ventricle, lung and fat, then morphological processing is used to remove the small interfering blocks and fill the closed area in the segmentation result, so that the adjacent areas in the image are connected together and the number of contours in the image is reduced; finally, the target contour is locked by a priori information to determine the area of the left ventricle in the ultrasound image, and the image is cropped according to the established target range to prevent the influence of other tissue structures (lung and fat) in the ultrasound image of the left ventricle on the segmentation result.

### **3.4 Data Augmentation**

Data augmentation is particularly useful when training deep learning models because such models often require a large amount of training data to achieve good performance, and sometimes collecting a sufficient amount of real data is difficult or expensive. By using data augmentation, it's possible to achieve effects similar to those of a larger dataset on a limited dataset, thus enhancing the model's generalization ability.

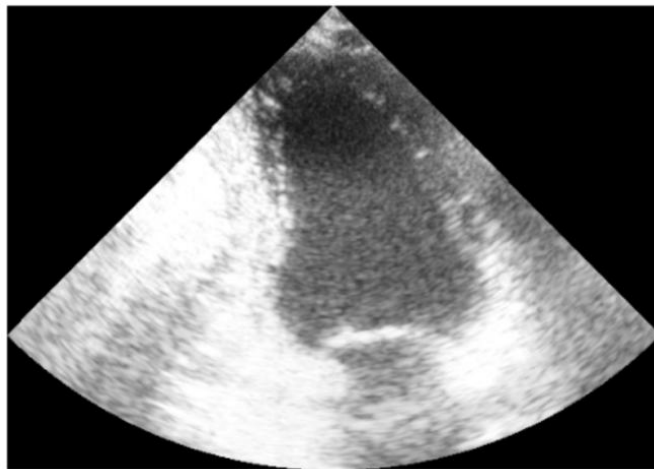
Common data augmentation operations include random rotation, scaling, translation, flipping, adding noise, and so on. By introducing these variations during the training process, the model can learn features under different transformations, thus better adapting to input data in various scenarios.

In order to improve the quality of segmentation, new models have been built on artificially augmented data. Augmented data have been obtained by using:

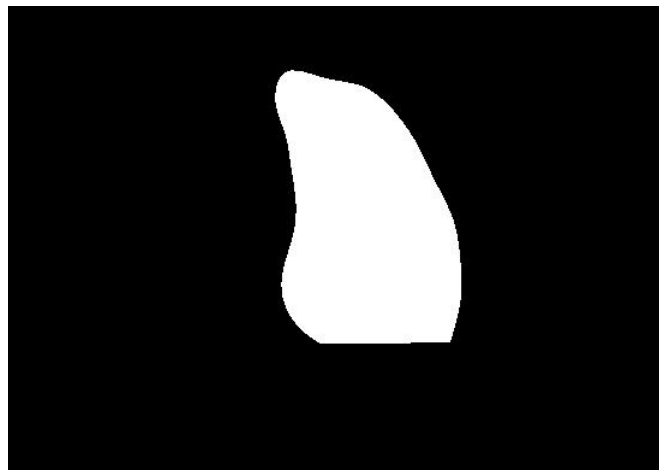
- 1) random vertical shifting of the image up or down up to 20% of the image height;
- 2) random horizontal shifting of the image left or right up to 20% of the image width;
- 3) random increase or decrease of the image in the range from 0.5 to 2 image sizes.

### 3.5 Data exploration and initial visualization

For the datasets used, the data was first explored and visualized for patient001 in the individual datasets. The 001\_2CH\_half\_sequence.nii.gz and 001\_2CH\_half\_sequence\_gt.nii.gz files were visualized in code. The original and masked visualizations for the patient001 dataset are shown in **Figure 5** and **Figure6**.



*Fig.5 Patient001 original image visualization*

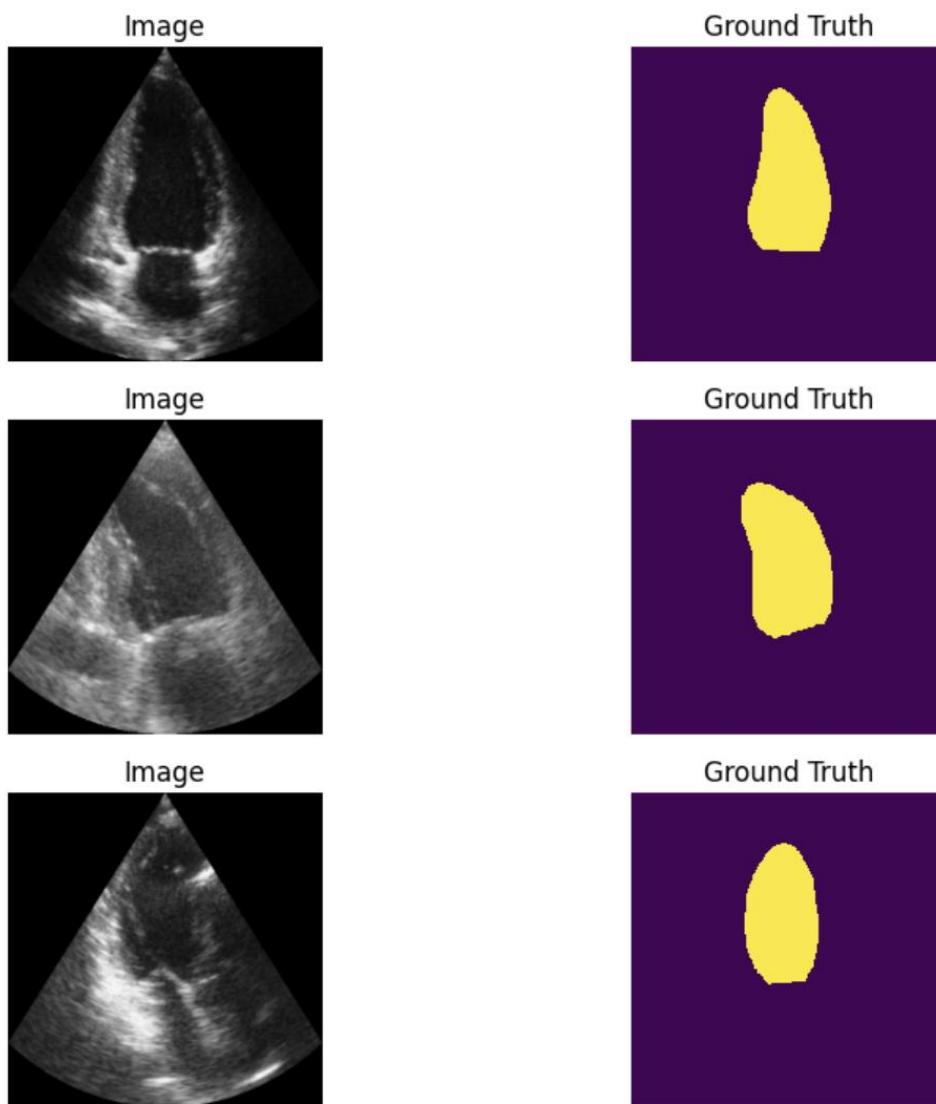


*Fig.6 Patient001 mask visualization*

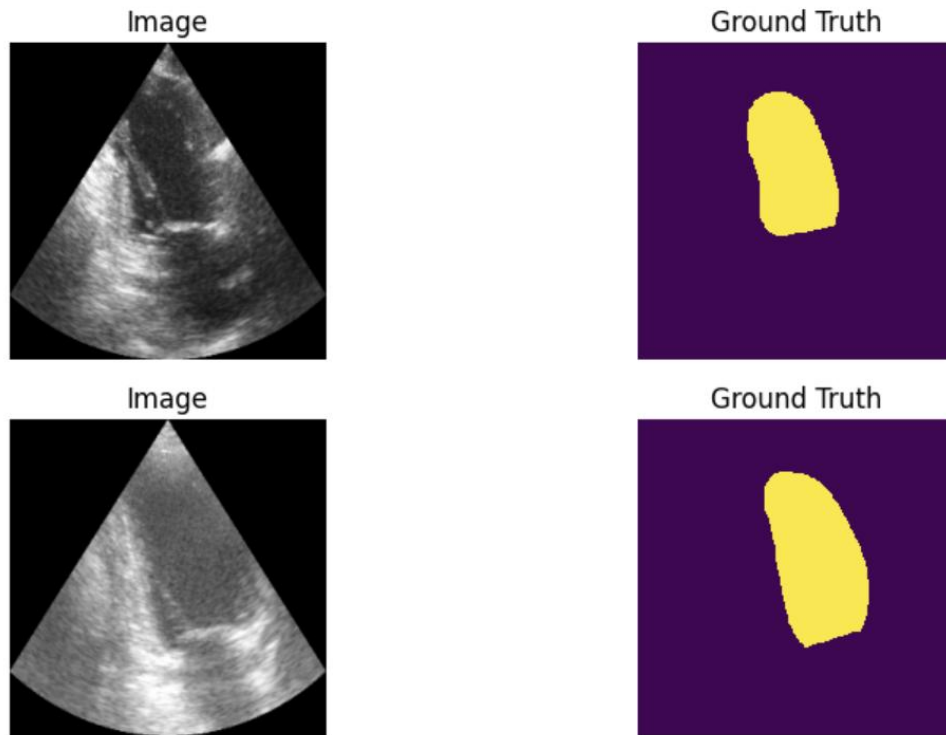
### 3.6 Save and split the visualized dataset

The dataset including 500 folders is visualized according to the visualization method used in the example given in section 2.3, and the original and masked images of the visualization are saved in the image and mask folders, respectively. The saved dataset is divided into a training set, a validation set, and a test set by code, with sizes of 400, 50, and 50. It is convenient for subsequent model training.

The above data was visualized using the visualize\_data function, and five samples from the test set were visualized randomly as shown in **Figure 7** and **Figure8** :



*Fig.7 Random visualization samples 1-3 in the visualization test set*



*Fig.8 Random visualization samples 4-5 in the visualization test set*

Of course, when preprocessing images, the method of narrowing down the image segmentation will vary depending on the dataset. Since different datasets correspond to different binarization thresholds when binarizing images, the image preprocessing methods can be difficult to generalize. In the follow-up work, more time and effort will be devoted to explore how to exploit the advantages of target detection networks to design networks that can detect and reduce the segmentation range of images in real time, with high robustness and higher accuracy.

#### **4. Image Segmentation Models**

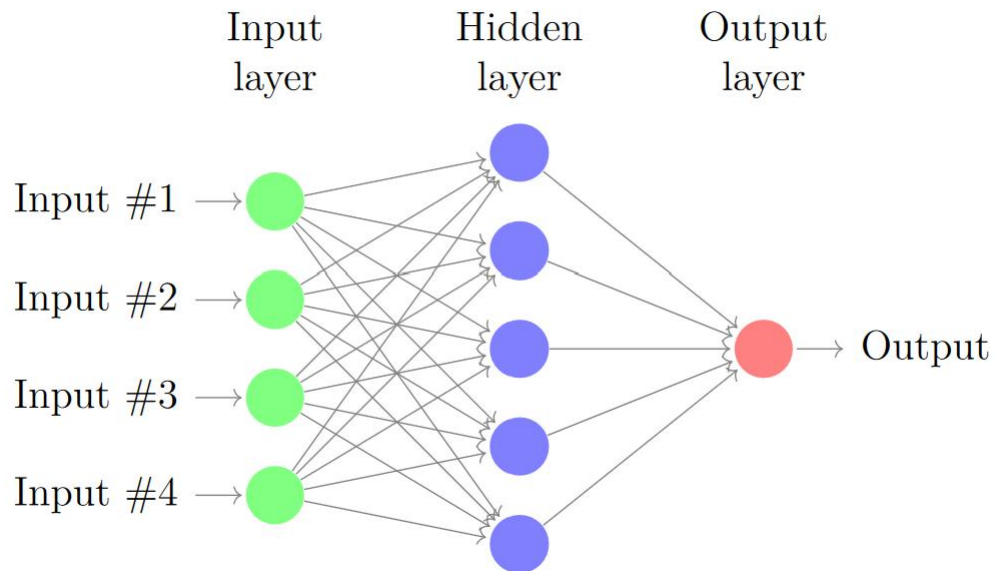
##### **4.1 Deep learning**

Deep learning is a research direction within the field of machine learning that aims to construct algorithmic models mimicking the neural networks of the human brain. It accomplishes automated learning and comprehension of complex data by using learning algorithms and hierarchical neuron structures. In deep learning, algorithmic models are trained on pre-provided datasets and then their performance is gradually optimized by varying network parameters and weights. Convolutional neural networks (CNNs) are a commonly used neural network structure in deep learning. They effectively identify patterns and features in image and video data through operations such as convolution and pooling and are widely applied in fields such as image recognition, object detection,

and natural language processing. Deep learning and convolutional neural networks are used as important technical tools in this thesis to handle difficult problems in pattern recognition and complex data processing, offering solid backing for future study and real-world implementations.

#### 4.1.1 Convolutional neural networks

An artificial neural network consists of three different types of layers; input layer, hidden layer and output layer, which can be seen in **Figure 9**. Each neuron from the input layer is connected to all or some of the neurons in the hidden layer<sup>[24]</sup>, but the neurons within the same layer are not connected to each other. A network can have multiple hidden layers. The convolutional neural network, which is the kind of artificial network used in this thesis, uses pixels from a small sample of an image known as a patch as its input neurons. The output has a value between 0 and 1, representing the probability of the patch belonging to the left ventricle. An entire image can be viewed as being repeatedly filtered at various scales when a convolutional neural network is applied to it.



*Fig.9 A basic artificial neural network*

#### 4.1.2 Loss function

To gauge a network's performance, one uses a loss function. The loss function represents the dissimilarity between the network output (i.e., the probability) and the true pixel labels, which was established with the help of the expert's annotation. Crossentropy is a loss function that is frequently used with convolutional neural networks. The network's cross-entropy is computed when weights and biases has been set and is given by comparing the predicted value given from the convolutional

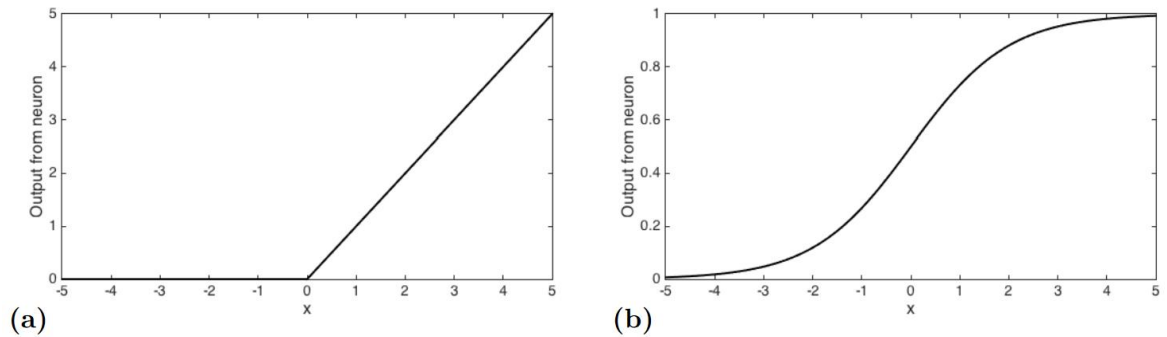
neural network with the corresponding correct label. The cross-entropy for one patch is given by( shown in Equation 4.1):

$$L(p, t) = - [t \log(p) + (1 - t) \log(1 - p)] \quad 4.1$$

where  $L$  is the loss value,  $p$  is the predicted value from the network (i.e. the probability of belonging to the left ventricle) and  $t$  is the target (i.e. the true label, 0 for background and 1 for left ventricle). To calculate the average error, the cross-entropies for the training data are added together and divided by the total training data size. The loss function is close to 0 when the network predicts the correct value, i.e., the true label value, which is an indication that it does not need to train more, and 1 when the output is incorrect.

### 4.1.3 Activation function

There are different activation functions; the ones used in this thesis are the rectified linear unit and the softmax function. All convolutional layers use the ReLU activation function, as defined by **Figure 10a**, whereas the output layer uses the softmax function, as defined by **Figure 10b**. In convolutional neural networks, these activation functions are frequently used.



*Fig.10 The figure shows the different activation functions used for the network. a) The ReLU function, which was used for all convolutional layers as activation function. b) The softmax function, which was used as activation function for the output layer.  $X$  represents the input for a neuron.*

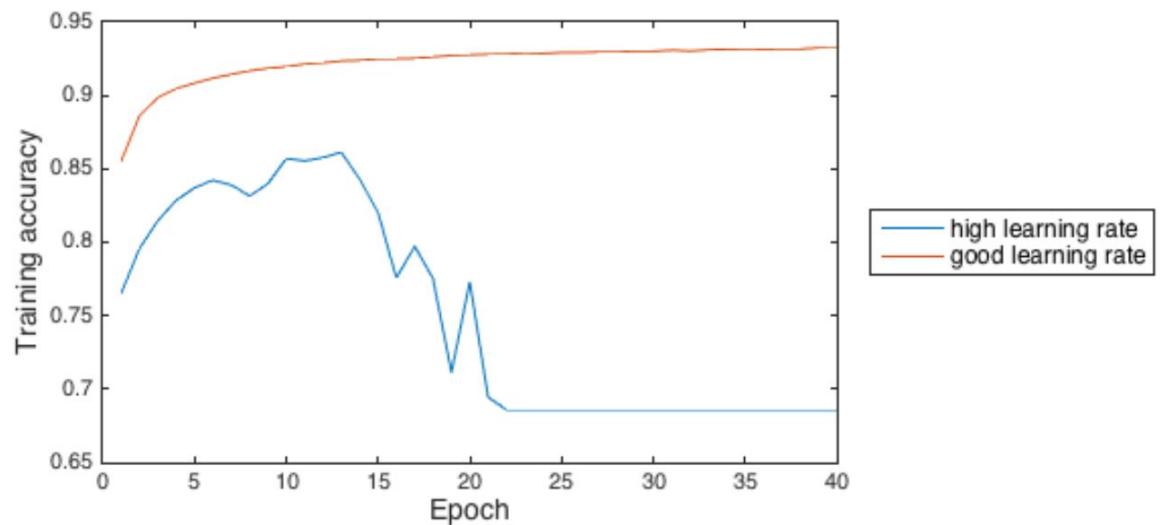
## 4.2 Hyperparameters

### 4.2.1 Learning rate

During CNN training, the learning rate controls how quickly the loss function is minimized and how well the training accuracy increases. Selecting an appropriate learning rate ensures that the CNN is presented with samples that are correctly learned by the learning algorithm, gradient descent. The learning rate should not be too small or too large. When the learning rate is too high, the loss function will first decrease but eventually stop decreasing or start to increase after a certain



number of epochs, indicating that the learning process has come to an end. A too small learning rate will make the learning process time-consuming because the weight updates in the CNN will be small, which results in a learning curve that decreases slowly. The opposite happens for training accuracy. As can be seen in **Figure 11**, a good learning rate will increase accuracy, while an excessively high learning rate will cause training accuracy to increase initially and then decrease.



*Fig.11 Learning rate effect. The learning rate decides on how large each step is when updating the weights and biases. When the minimum of the loss function is starting to approach, the step size can be decreased by implementing a learning decay making sure that the optimal weights are not missed because of a too large step size. The learning decay can be implemented in different ways, the learning rate can be updated after each epoch, depending on the value of the training accuracy or decreased based on the validation accuracy.*

#### 4.2.2 Patch size

An additional consideration is the input patch's size. If the patch size is too small, the information will not be sufficient for the CNN to determine what to classify the pixel as. If the patch is larger, more information will be given for CNN to learn from. In addition, the computational time is increased because more weights require regulation. It is possible that the pixels in the same patch will not correlate well with one another if the size is too big.

#### 4.2.3 Batch size

The number of patches examined before moving on in accordance with the gradient descent defines the batch size. A batch size greater than one can be beneficial while training a CNN with a large data set because it converges quicker when the step taken is based on more patches than one. Large batch sizes typically lead to an increase in learning rate, which speeds up the process of

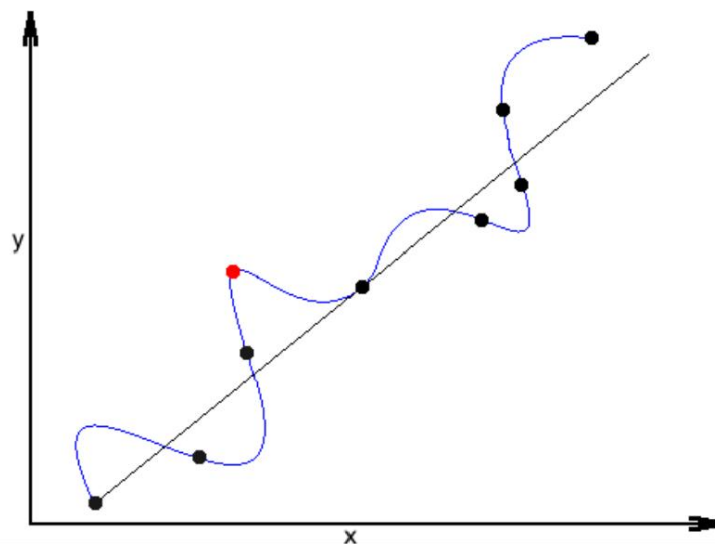
finding optimized weights and biases because a higher learning rate causes the gradient descent's step size to be larger.

#### 4.2.4 Epochs

To ensure that the sample order has no bearing on the weight optimization process, the batches in the training set will be randomized during one epoch. The CNN should learn the samples and not their order. The training will optimize the weights by working through the entire training set. Thereafter, the validation set is classified with the help of the new weights and biases, and the result of the validation depends on how well the CNN learned to recognize the samples it went through during the training phase. Because there is only classification using the current parameter settings and no training, the samples are not randomized during validation.

#### 4.2.5 Overfitting

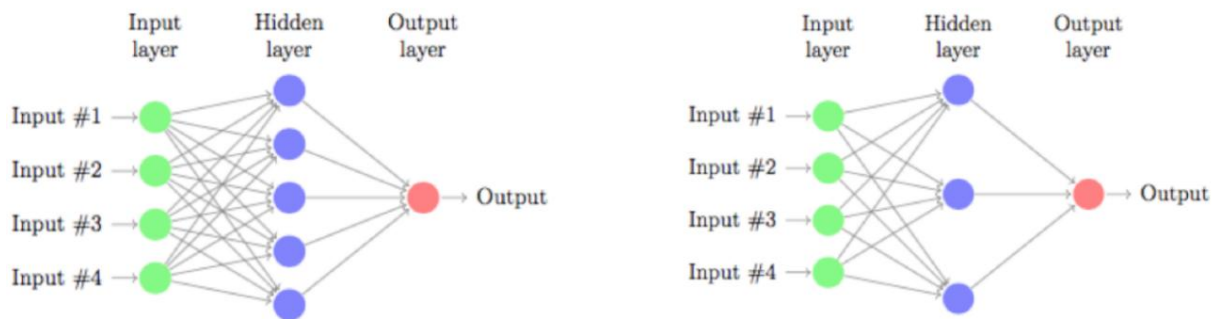
Convolutional neural networks have the ability to learn patterns between the input and the output with the help of a few convolutional layers. However, it is possible for the CNN to learn the data too well, in which case it will identify more noise than true patterns. This can be seen in **Figure 12**. This phenomenon is called overfitting. When training a convolutional neural network, overfitting is a common issue that frequently arises when the size of the training set is restricted. A sign of overfitting in the CNN is when the validation accuracy is lower than the training accuracy. The degree to which the CNN is overfitting increases with the difference between the training and validation accuracy.



*Fig.12 Overfitting analysis. The model represented by the blue line in the figure has learnt from the data points so well that it fits to points which does not exist in the data, for example the red dot. The black line is an example where data is assumed to contain noise, the model has less overfitting.*

## 4.2.6 Dropout

The dropout technique is one way to reduce the amount of overfitting in a CNN. A hidden neuron in the CNN can be temporarily removed with a certain probability when dropout is used. The connections that go to the neuron and from the neuron are also temporarily deactivated. A dropout rate of 50% in a hidden layer means that the probability of each independent neuron being temporarily removed during the epoch is 50%. To prevent neurons from becoming overly dependent on one another, applying a dropout to a CNN can be thought of as training on only a portion of the CNN; this is shown in **Figure 13**. A stronger connection is formed between the neurons if every neuron contains the same type of information. In the end, dropout can be seen as combining different convolutional neural networks with different information. For every new epoch, all deactivated neurons are reset, allowing for the deactivation of additional neurons.



*Fig.13 Complete network and implemented dropout network. The left figure represents the complete network and the network on the right show the same network when dropout has been implemented, which has temporarily removed 2 neurons from the hidden layer.*

## 4.3 Architectures

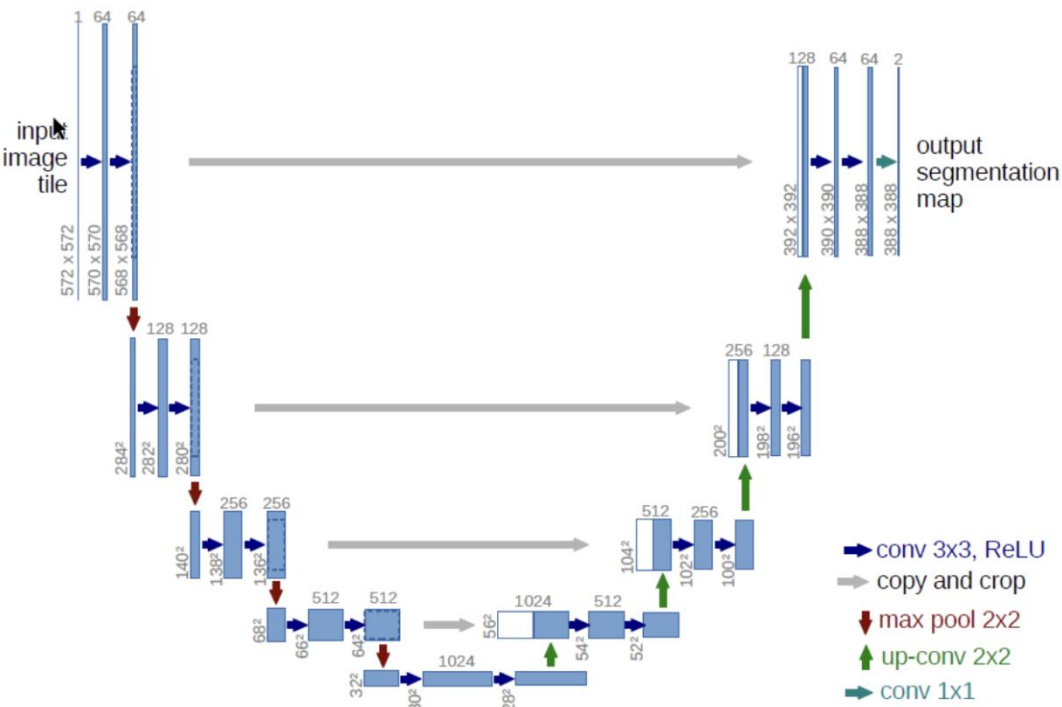
### 4.3.1 U-Net

U-Net is a deep learning model used for image segmentation, proposed by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015. The name "U-Net" stems from the shape of its network architecture, which resembles a "U" due to its encoder-decoder structure. The inspiration for U-Net came from addressing the challenges of data imbalance and preserving fine details in medical image segmentation tasks. Its network structure consists of two parts: the encoder and the decoder. The encoder consists of stacked convolutional and pooling layers, which progressively reduce the spatial dimensions of the input image while extracting feature information. These convolution operations capture features at different scales and semantics using convolutional kernels. The number of

channels in the feature maps is increased at each step, enhancing the model's learning capacity. The decoder is responsible for upsampling the output of the encoder (via transpose convolutions or upsampling operations) and connecting it with the corresponding encoder levels. These connections, known as skip connections, help propagate low-level detail information to the decoder, yielding more accurate segmentation results. Additionally, the decoder gradually recovers the resolution through stacked convolutional and upsampling layers. U-Net employs a loss function called the cross-entropy loss, which differs from traditional convolutional neural networks. This loss function measures the difference between the model's predictions and the true labels, allowing the network parameters to be updated accordingly, leading to predictions that closely align with the true segmentation.

U-Net architecture is concise yet effective, making it suitable for various image segmentation tasks such as medical image segmentation, object detection, and semantic segmentation. It has achieved remarkable performance in numerous competitions and real-world applications, becoming a significant benchmark model in the field of image segmentation. The specific structure is shown in

**Figure 14:**



*Fig.14 U-Net architecture*

### 4.3.2 U-Net++

U-Net++ introduces more skip connections and a multi-scale feature aggregation mechanism on the classic U-Net model for medical image segmentation. This architecture is a deeply supervised encoder-decoder network, where the encoder and decoder subnetworks are interconnected through a series of nested dense skip pathways. Skip connections allow the model to pass information between layers at different depths, improving the understanding of image structure and semantic information. The multi-scale feature aggregation mechanism merges feature maps at different scales, enhancing the model's perception of various details in the image. Additionally, U-Net++ incorporates an attention mechanism, enabling the model to adaptively focus on important regions in the image. These improvements have significantly boosted performance across various domains, making U-Net++ a crucial choice for image segmentation tasks. The specific structure is shown in **Figure 15**.

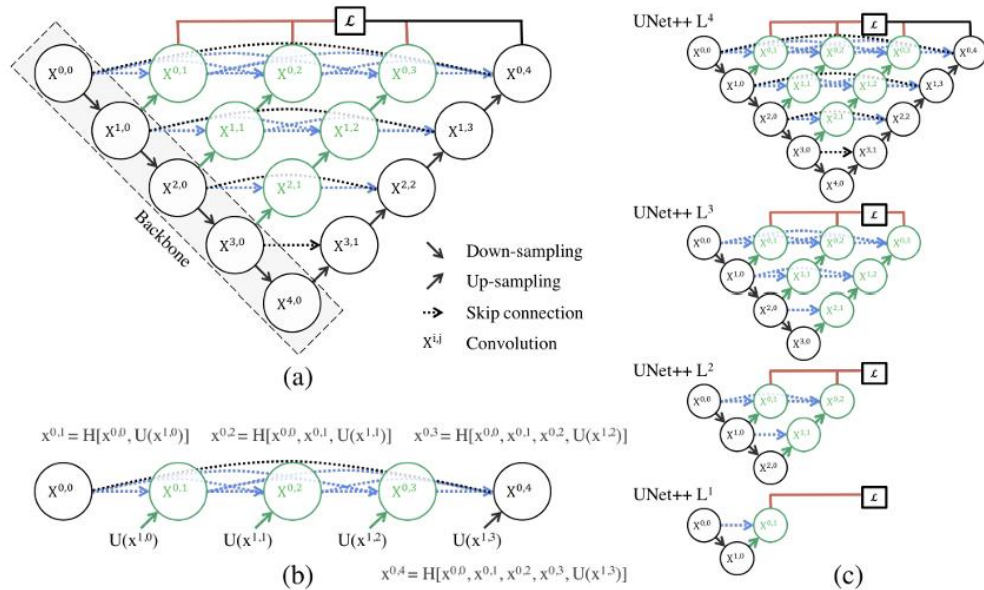


Fig.15 U-Net++architecture

(a) U-Net++ consists of an encoder and decoder that are connected through a series of nested dense convolutional blocks. The main idea behind U-Net++ is to bridge the semantic gap between the feature maps of the encoder and decoder prior to fusion. For example, the semantic gap between  $(X^{0,0}, X^{1,3})$  is bridged using a dense convolution block with three convolution layers. In the graphical abstract, black indicates the original U-Net, green and blue show dense convolution blocks on the skip pathways, and red indicates deep supervision. Red, green, and blue components distinguish U-Net++ from U-Net. (b) Detailed analysis of the first skip pathway of U-Net++. (c) U-Net++ can be pruned at inference time, if trained with deep supervision.

### 4.3.3 MANet

MANet, short for "Multi-scale Attention Network," is a network architecture used for image processing tasks. This structure introduces multi-scale attention mechanisms to weight image features at different scales, enhancing the model's ability to focus on important information. This design allows the model to better adapt to features at various scales, leading to significant performance improvements in tasks such as object detection and image segmentation. MANet consists primarily of Position Attention Blocks (PAB) and Multi-scale Fusion Attention Blocks (MFAB).

**Position Attention Blocks (PAB):** This block focuses on globally capturing spatial dependencies between pixels in the image by introducing a position attention mechanism. It helps the network better understand spatial structures and relationships in the image, thereby improving perception of important information.

**Multi-scale Fusion Attention Blocks (MFAB):** This block functions by fusing multi-scale semantic features to capture channel dependencies across any feature maps. By integrating semantic information at different scales, MFAB provides more comprehensive contextual information, enabling the model to better understand the content of the image and handle features at different scales more flexibly and accurately. The specific structure is shown in **Figure 16**.

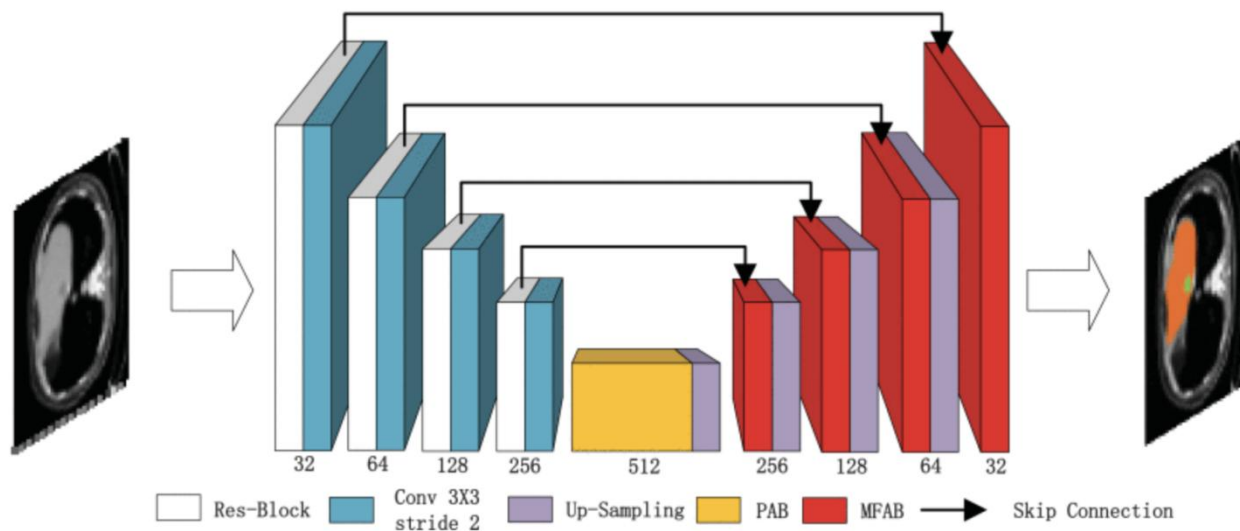


Fig.16 The total architecture of MA-Net.

### 4.3.4 LinkNet

LinkNet is a fully convolutional neural network designed for image semantic segmentation tasks, consisting of an encoder and a decoder connected via skip connections. The encoder extracts features from the input image, which are utilized by the decoder to generate precise segmentation masks. Skip connections allow the decoder to access early layers of the network, preserving spatial details necessary for accurate segmentation. In LinkNet, skip connections enable the decoder to access early layers of the encoder, aiding in maintaining the required spatial details for precise segmentation, thus enhancing segmentation performance. LinkNet uses a "sum" operation to fuse decoder blocks and skip connections, ensuring effective integration of features at different scales. Additionally, employing a fully convolutional neural network architecture enables LinkNet to handle input images of arbitrary sizes while preserving resolution, thus exhibiting good adaptability when processing images of varying dimensions. Relative to other complex semantic segmentation models, LinkNet features a relatively lightweight design with lower computational and memory requirements, making it suitable for deployment and usage in resource-constrained environments. Due to its relatively simple structure, LinkNet is typically easy to train and fine-tune, allowing for rapid implementation and demonstrating good generalization performance. The specific structure is shown in **Figure 17**.

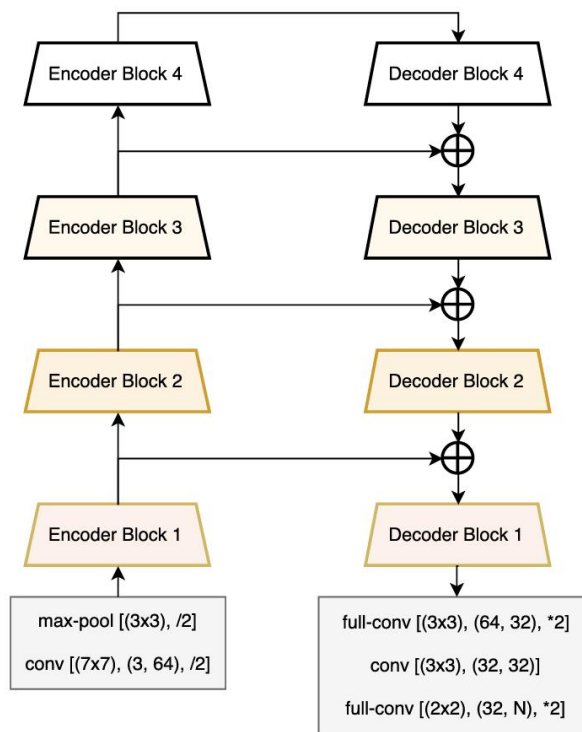


Fig.17 LinkNet Architecture.

### 4.3.5 FPN

FPN stands for "Feature Pyramid Network," which is a deep learning architecture used for multi-scale object detection and semantic segmentation tasks in computer vision. FPN was introduced by Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie in a paper published in 2017. The main idea behind FPN is to create a feature pyramid that combines features from different scales and levels of abstraction to improve the performance of object detection and segmentation algorithms. Traditional convolutional neural networks (CNNs) face challenges in detecting objects of various sizes, as they typically operate on fixed-size inputs and may lose fine-grained details or fail to capture larger context.

To address this issue, FPN constructs a pyramid of multi-scale feature maps, with each pyramid level corresponding to a different input scale. Lower pyramid levels have higher resolutions, making them suitable for detecting small objects and details. Higher pyramid levels have lower resolutions, making them suitable for detecting larger objects and capturing global context. These feature maps are then integrated and propagated through a top-down pathway and lateral connections to enable cross-scale object detection and segmentation. A diagram of the different pyramid structures is shown in **Figure 18**:

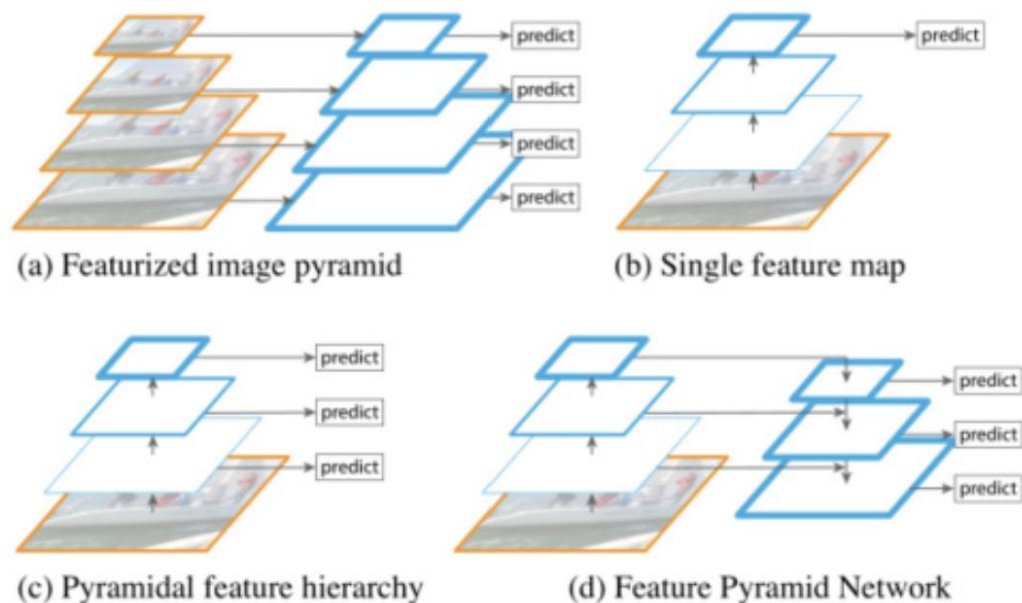


Fig.18 Different pyramid structures: (d) shows the Feature Pyramid Network

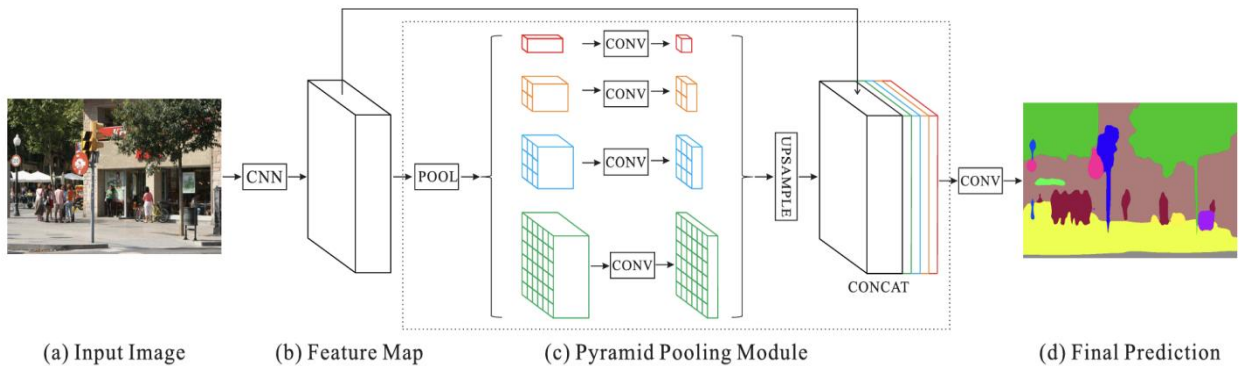


### 4.3.6 PSPNet

PSPNet (Pyramid Scene Parsing Network) is a fully convolutional neural network designed for image semantic segmentation, specifically targeting multi-class segmentation tasks on high-resolution images. It consists of an encoder responsible for extracting features from input images and a spatial pyramid decoder built on top of the encoder. The spatial pyramid decoder utilizes a pyramid structure to decode features, covering information across different scales. This architecture enhances the network's understanding of semantic information in images, thereby improving segmentation accuracy and robustness, particularly for high-resolution images.

However, PSPNet is not proficient in handling the recognition and segmentation of small objects. This limitation arises because its spatial pyramid structure does not prioritize retaining fine-grained spatial details but rather focuses on encompassing semantic information across scales. Consequently, in scenarios requiring precise identification and segmentation of small objects, PSPNet may exhibit subpar performance, with its generated masks potentially lacking in pixel-level accuracy.

The specific structure is shown in **Figure 19**.

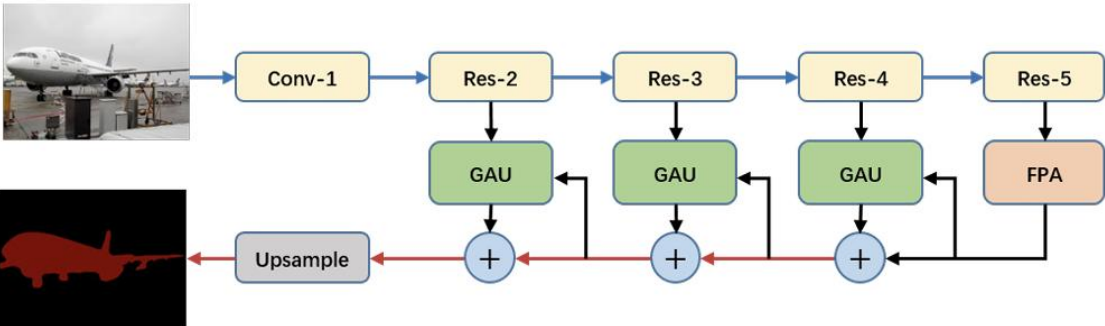


*Fig.19 Overview of our proposed PSPNet. Given an input image (a), we first use CNN to get the feature map of the last convolutional layer (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representation is fed into a convolution layer to get the final per-pixel prediction (d).*

### 4.3.7 PAN

The PAN (Pyramid Attention Network) model is a deep learning architecture for object detection tasks, which combines the Feature Pyramid Network (FPN) and the Global Attention Upsample (GAU) module. The Feature Pyramid Attention (FPA) module utilizes a pyramid structure to provide pixel-level attention information, enlarging the receptive field to enhance feature representation. The GAU module utilizes high-level feature maps to guide pixel-level localization of low-level features for global perception and upsampling, integrating features across different scales

to improve detection performance. PAN combines the feature pyramid network and attention mechanism to improve object detection accuracy across different scales. Using a pyramid structure provides pixel-level attention information, expanding the receptive field to enhance feature representation. The global attention mechanism and upsampling module aid in global perception and feature integration, enhancing detection performance. However, training and tuning parameters may require longer time and larger computational resources. Further optimization and parameter tuning may be needed for specific scenarios or datasets to achieve better performance. For very small objects or highly complex scenes, further improvements may be necessary to enhance detection performance. The structure is shown in **Figure 20**.



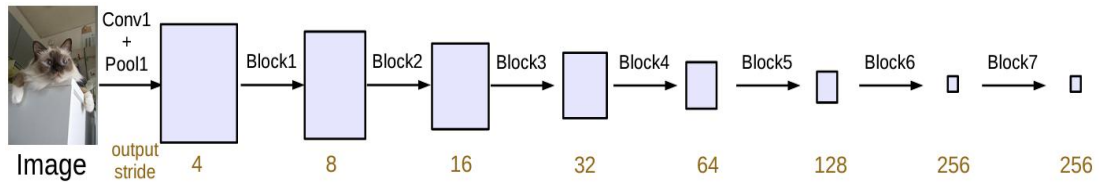
*Fig.20 Overview of the Pyramid Attention Network. We use ResNet-101 to extract dense features. Then we perform FPA and GAU to extract precise pixel prediction and localization details. The blue and red lines represent the downsample and upsample operators respectively.*

### 4.3.8 DeepLabv3

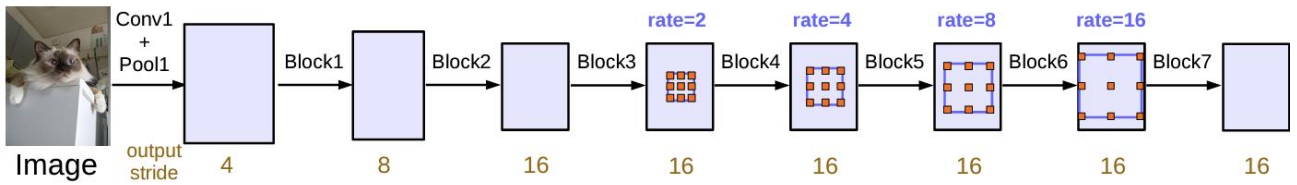
DeepLabv3 builds upon DeepLabv1 and DeepLabv2, focusing on atrous convolution, Fully-connected Conditional Random Field (CRF), and ASPP. It reconsiders the use of atrous convolution and enhances the ASPP module. It introduces a module for segmenting objects at multiple scales, designs serial and parallel atrous convolution modules, and employs various atrous rates to capture multi-scale contextual information. By using atrous convolution, it prevents issues with low resolution. Concatenating different dilation rates of atrous convolutions or running parallel atrous convolutions (as in DeepLabv2's ASPP) gathers more contextual information. The reevaluation of atrous convolution allows for obtaining a larger receptive field within the cascade module and spatial pyramid pooling framework, enabling the acquisition of multi-scale information. Composed of atrous convolutions with different sampling rates and BN layers, we attempt to lay out modules in a cascade or parallel manner. Using a large sampling rate of  $3 \times 3$  atrous convolutions, due to

limitations near image boundaries where long-range information cannot be captured, they may degrade into  $1 \times 1$  convolutions, proposing the fusion of image-level features into the ASPP module.

The model architectures are shown in **Figure 21,22**:



(a) Going deeper without atrous convolution



(b) Going deeper without atrous convolution. Atrous convolution with  $rate > 1$  is applied after block when  $output\_stride = 16$ .

Fig.21 Cascaded modules without and with atrous convolution.

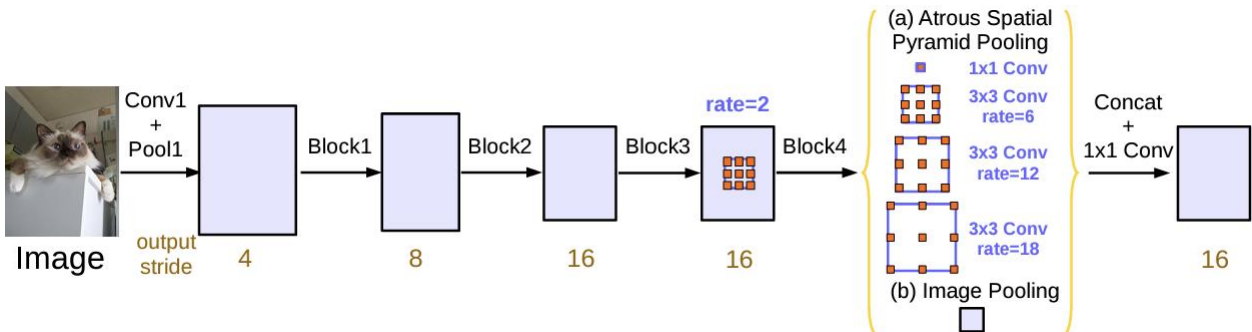
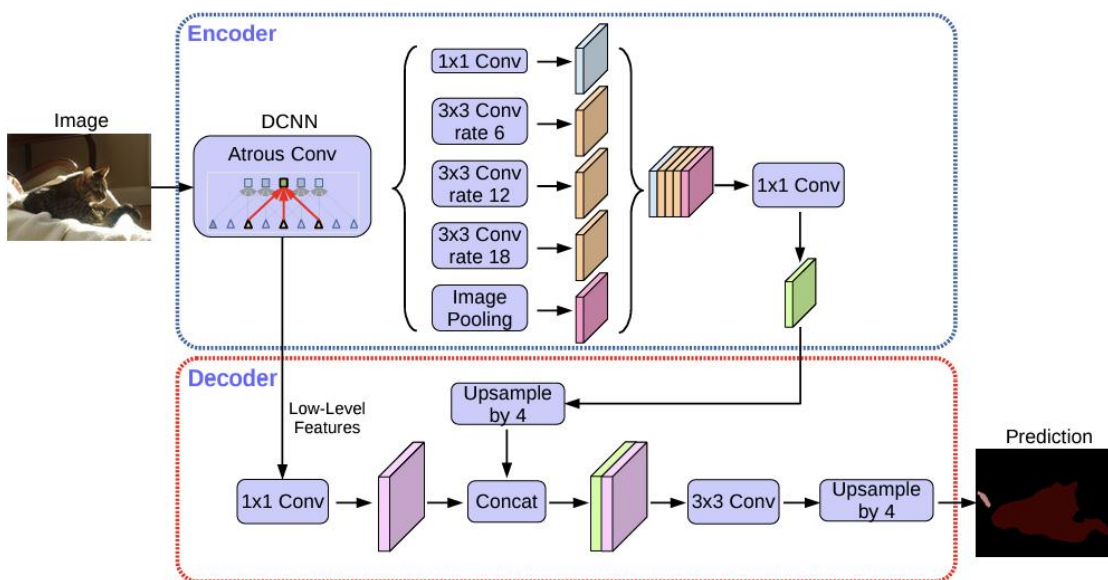


Fig.22 Parallel modules with atrous convolution (ASPP), augmented with image-level features.

### 4.3.9 DeepLabv3+

DeepLabv3+ is a deep learning semantic segmentation model designed to perform semantic labeling for every pixel in an image. It is an upgraded version of the DeepLab series models, incorporating technologies such as atrous convolution and Spatial Pyramid Pooling (SPP) to improve the accuracy and efficiency of semantic segmentation. The main body of the encoder in the DeepLabv3+ model consists of a DCNN with atrous convolutions, using common classification networks such as ResNet, followed by the Atrous Spatial Pyramid Pooling module to introduce multiscale information. Compared to DeepLabv3, v3+ introduces an improved decoder module, also known as the Xception module, to further enhance segmentation accuracy. It incorporates a Context Encoding module for extracting richer contextual information to improve segmentation

performance. Additionally, depthwise separable convolutions are employed instead of standard convolutions to reduce the number of parameters and enhance computational efficiency. The structure is illustrated in **Figure 23**.



*Fig.23 Our proposed DeepLabv3+ extends DeepLabv3 by employing a encoder-decoder structure. The encoder module encodes multi-scale contextual information by applying atrous convolution at multiple scales, while the simple yet effective decoder module refines the segmentation results along object boundaries.*

## 4.4 Encoder

### 4.4.1 ResNet

The Residual Neural Network (ResNet) is a type of deep learning model where each layer of the network does not directly learn the expected output, but instead learns the residual relationship with the input. By introducing residual blocks, ResNet addresses the issues of gradient vanishing and exploding that occur during training deep neural networks, enabling the training of deeper network models. The core idea of ResNet is to use skip connections to pass the input signal (identity mapping) directly to subsequent layers, preserving the original information, thus making the training of neural networks more efficient and stable.

ResNet achieves this by adding skip connections that skip certain network layers to perform identity mapping and then add this with the output of the network layers. This network operates similarly to a highway neural network, using large positive biases on weights to "gate" the information flow. This design allows deep learning models with tens or hundreds of layers to be easier to train, maintaining or even improving accuracy when increasing the depth of the model. ResNet introduces

two types of mappings: identity mapping, represented by the curve labeled with  $x$  on the right side, and residual mapping, where the residual refers to the part  $F(x)$ . The final output is the sum of  $F(x)$  and  $x$ . The implementation of  $F(x) + x$  can be achieved through a feedforward neural network with "shortcut connections," which skip one or more layers. The "weight layer" in the diagram refers to the convolution operation. If the network has already reached an optimum and continues to deepen, the residual mapping will approach zero, leaving only the identity mapping. In theory, the network will then consistently remain in an optimal state, and the network's performance will not decrease with increasing depth. The structure is illustrated in **Figure 24**. ResNets with five different layers are shown in **Figure 25**.

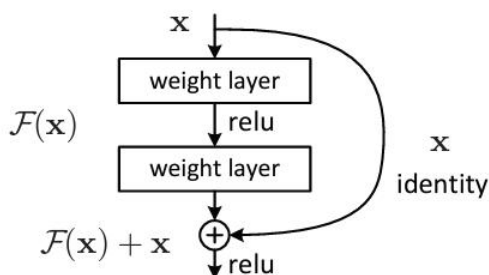


Fig.24 Residual learning: a building block.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

Fig.25 Architectures for ImageNet. Building blocks are shown in brackets, with the numbers of blocks stacked. Downsampling is performed by conv31, conv41, and conv51 with a stride of 2.

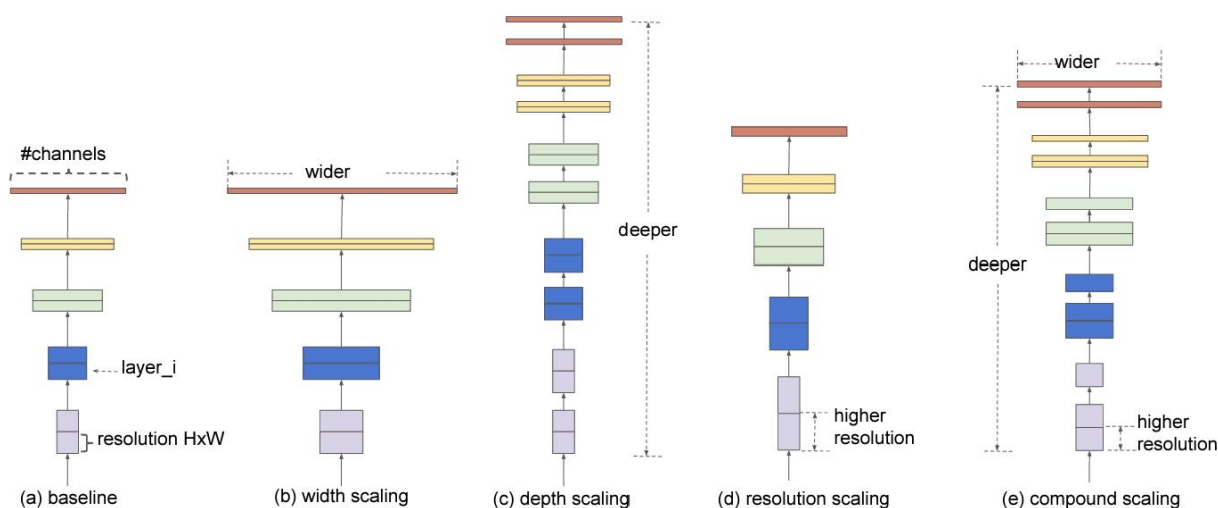
## 4.4.2 EfficientNet

EfficientNet is an efficient neural network architecture based on a compound scaling method that adjusts network depth, width, and image resolution to optimize model performance. EfficientNet achieves higher accuracy and computational efficiency compared to traditional neural networks while keeping the number of model parameters relatively low. Its core idea is to improve performance by comprehensively optimizing network depth, width, and image resolution without increasing complexity.

Key features of EfficientNet include:

1. Compound Scaling: EfficientNet scales network depth, width, and resolution in a compound manner to enhance model performance.
2. Mobile-Size Models: EfficientNet includes different model versions, such as EfficientNet-B0 to EfficientNet-B7, suitable for various computational resources and task requirements.
3. Efficient Building Blocks: EfficientNet employs lightweight network building blocks like depthwise separable convolutions and the Swish activation function to enhance computational efficiency.

EfficientNet is an efficient neural network architecture that optimizes network depth, width, and resolution through effective allocation of computational resources, leading to improved model performance and reduced computational costs. The structure is shown in **Figure 26**.



*Fig.26 Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.*

### 4.4.3 MobileNetv2

MobileNetv2 is a lightweight neural network architecture designed to provide efficient computational performance in resource-constrained environments such as mobile and embedded devices. MobileNetv2 combines the design of depthwise separable convolutions and linear bottlenecks to optimize network structure and parameter configuration, achieving high accuracy while keeping the model lightweight. It is well-suited for deployment and applications in resource-constrained scenarios.

The key features of MobileNetv2 include:

- 1) Depthwise Separable Convolution: MobileNetv2 employs a depthwise separable convolution structure that decomposes traditional convolution operations into depthwise convolutions and pointwise convolutions, reducing computational cost and parameter count.
- 2) Linear Bottleneck: The introduction of linear bottleneck structures helps maintain the dimension information of feature maps, enhancing the model's representational capacity.
- 3) Inverted Residuals: MobileNetv2 utilizes an inverted residual structure that enhances feature propagation and capture through feature up/downsampling and residual connections.
- 4) Faster training and inference speed: Due to the design of the network structure and optimized configuration, MobileNetv2 typically exhibits faster speeds during training and inference. The structure is shown in **Figure 27**.

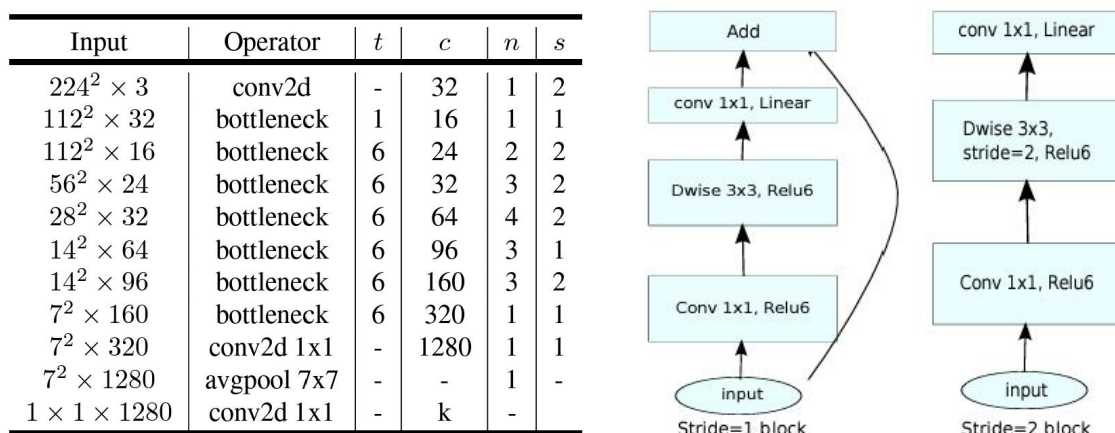


Fig.27 MobileNetv2

### 4.5 Network Structure Design

For the design of the model structure, the encoders in the architectures U-Net,U-Net++, MANet, Linknet, PSPNet, PAN, FPN, DeepLabv3 and DeepLabv3+ are replaced by Resnet18, Resnet34,

Resnet5,Resnet101, Efficientnet-b0, Efficientnet-b1, Efficientnet-b3,Efficientnet-b5,Efficientnet-b7and MobileNetv2, respectively, and ImageNet is used as the pre-trained weights. The model structures are as follows:

U-Net, U-Net (Encoder=ResnNet18, ResNet34, ResNet50, ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3, EfficientNet-b5, EfficientNet-b7and MobileNetv2),

U-Net++, U-Net++(Encoder=ResnNet18, ResNet34, ResNet50, ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3, EfficientNet-b5, EfficientNet-b7and MobileNetv2),

MANet, MANet(Encoder=ResnNet18, ResNet34, ResNet50, ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3, EfficientNet-b5, EfficientNet-b7and MobileNetv2),

LinkNet, LinkNet(Encoder=ResnNet18, ResNet34, ResNet50, ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3, EfficientNet-b5, EfficientNet-b7and MobileNetv2),

PSPNet, PSPNet(Encoder=ResnNet18, ResNet34, ResNet50, ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3, EfficientNet-b5, EfficientNet-b7and MobileNetv2),

PAN, PAN(Encoder=ResnNet18, ResNet34, ResNet50, ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3, EfficientNet-b5, EfficientNet-b7and MobileNetv2),

FPN, FPN(Encoder=ResnNet18, ResNet34, ResNet50, ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3, EfficientNet-b5, EfficientNet-b7and MobileNetv2),

DeepLabv3, DeepLabv3(Encoder=ResNet18, ResNet34, ResNet5,ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3,EfficientNet-b5,EfficientNet-b7and MobileNetv2) ,

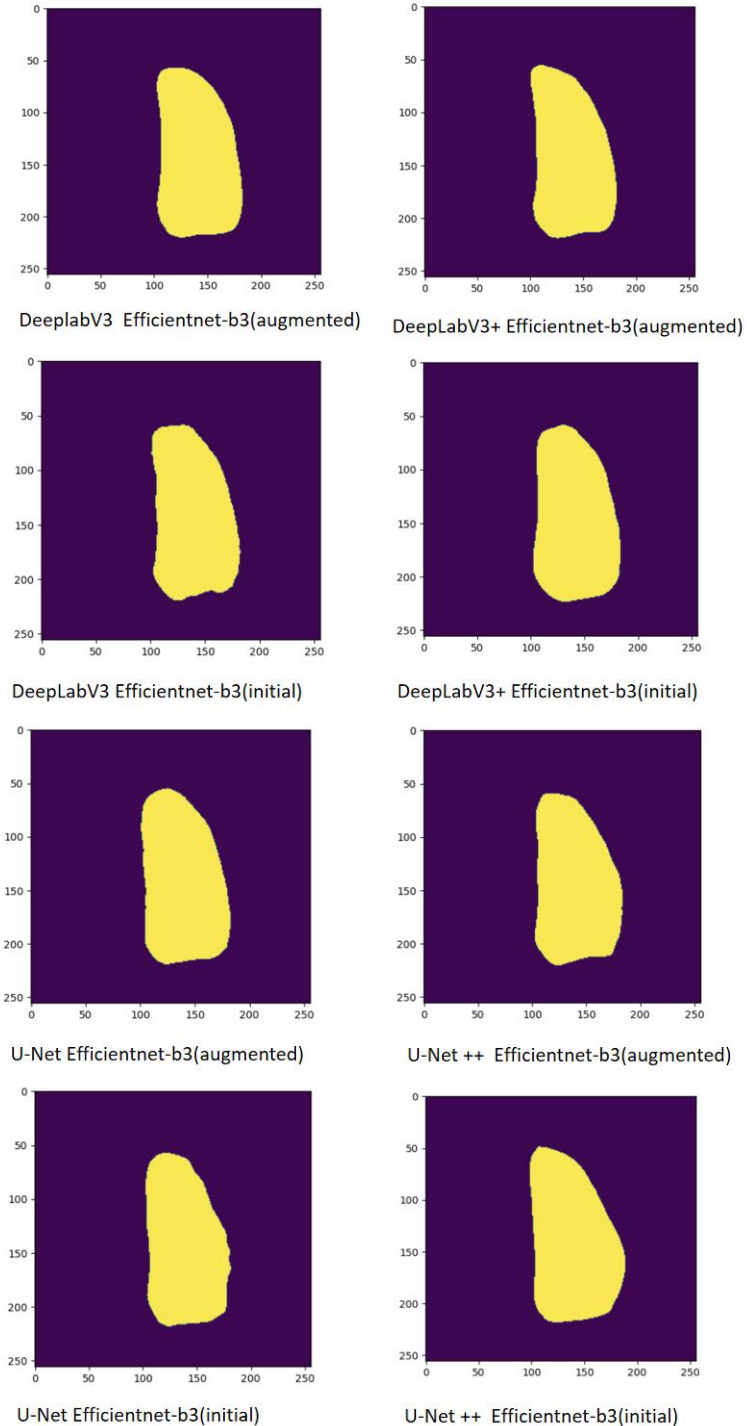
DeepLabv3+,DeepLabv3+(Encoder=ResNet18,ResNet34, ResNet5,ResNet101, EfficientNet-b0, EfficientNet-b1, EfficientNet-b3,EfficientNet-b5,EfficientNet-b7and MobileNetv2).

Next, model training is performed for each of the above model structures with the previously processed initial dataset and augmented dataset.



# RESULT

For the model proposed above, the training of the model has been carried out, the visualization of partial model segmentation results is shown in **Figure 28** and the data related to the experimental results are in **Table 1-10** as follows:



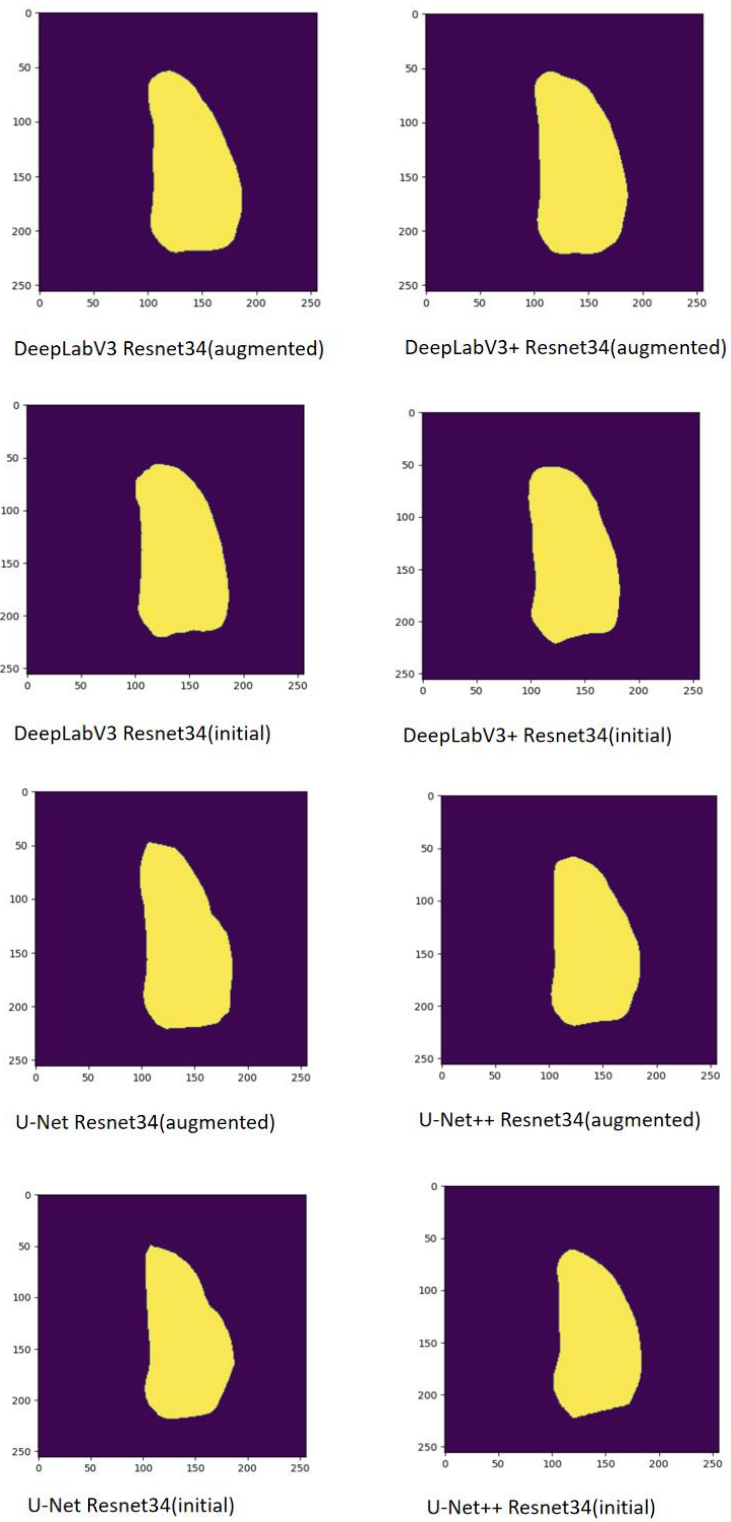


Fig.28 The visualization of partial model segmentation results

Table 1: The results of **U-Net** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>U-Net</b>		0.821	0.831	0.850	0.873	0.910	0.920
<b>U-Net</b>	ResNet 18	0.842	0.852	0.870	0.881	0.914	0.925
<b>U-Net</b>	ResNet 34	0.854	0.860	0.880	0.911	0.920	0.933
<b>U-Net</b>	ResNet 50	0.852	0.865	0.872	0.883	0.929	0.938
<b>U-Net</b>	ResNet 101	0.850	0.863	0.869	0.880	0.921	0.937
<b>U-Net</b>	MobileNetv2	0.832	0.835	0.851	0.875	0.919	0.931
<b>U-Net</b>	EfficientNet-b0	0.843	0.854	0.887	0.901	0.920	0.939
<b>U-Net</b>	EfficientNet-b1	0.851	0.862	0.884	0.900	0.928	0.940
<b>U-Net</b>	EfficientNet-b3	0.855	0.866	0.887	0.913	0.924	0.946
<b>U-Net</b>	EfficientNet-b5	0.848	0.856	0.871	0.894	0.923	0.932
<b>U-Net</b>	EfficientNet-b7	0.832	0.840	0.861	0.897	0.929	0.934

Table 2: The results of **U-Net++** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>U-Net++</b>		0.837	0.845	0.860	0.883	0.918	0.927
<b>U-Net++</b>	ResNet 18	0.861	0.873	0.882	0.895	0.919	0.928
<b>U-Net++</b>	ResNet 34	0.875	0.889	0.885	0.907	0.926	0.940
<b>U-Net++</b>	ResNet 50	0.867	0.883	0.880	0.901	0.920	0.935
<b>U-Net++</b>	ResNet 101	0.867	0.878	0.883	0.912	0.919	0.934
<b>U-Net++</b>	MobileNetv2	0.852	0.875	0.875	0.910	0.920	0.937
<b>U-Net++</b>	EfficientNet-b0	0.850	0.878	0.887	0.912	0.922	0.940
<b>U-Net++</b>	EfficientNet-b1	0.869	0.887	0.889	0.910	0.927	0.932
<b>U-Net++</b>	EfficientNet-b3	0.880	0.899	0.901	0.917	0.930	0.958
<b>U-Net++</b>	EfficientNet-b5	0.882	0.902	0.896	0.910	0.921	0.945
<b>U-Net++</b>	EfficientNet-b7	0.865	0.887	0.887	0.901	0.920	0.939

Table 3: The results of **MANet** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>MANet</b>		0.781	0.799	0.821	0.844	0.879	0.906
<b>MANet</b>	ResNet 18	0.802	0.822	0.847	0.865	0.912	0.933
<b>MANet</b>	ResNet 34	0.820	0.844	0.870	0.888	0.922	0.944
<b>MANet</b>	ResNet 50	0.803	0.821	0.855	0.877	0.910	0.922
<b>MANet</b>	ResNet 101	0.799	0.812	0.844	0.867	0.915	0.926
<b>MANet</b>	MobileNetv2	0.821	0.845	0.872	0.880	0.903	0.930
<b>MANet</b>	EfficientNet-b0	0.820	0.844	0.869	0.886	0.914	0.936
<b>MANet</b>	EfficientNet-b1	0.833	0.853	0.877	0.894	0.923	0.938
<b>MANet</b>	EfficientNet-b3	0.851	0.879	0.899	0.902	0.938	0.951
<b>MANet</b>	EfficientNet-b5	0.843	0.865	0.875	0.876	0.889	0.914
<b>MANet</b>	EfficientNet-b7	0.823	0.843	0.845	0.845	0.865	0.901

Table 4: The results of **LinkNet** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>LinkNet</b>		0.753	0.785	0.803	0.831	0.872	0.899
<b>LinkNet</b>	ResNet 18	0.771	0.803	0.832	0.853	0.893	0.921
<b>LinkNet</b>	ResNet 34	0.797	0.821	0.850	0.871	0.922	0.942
<b>LinkNet</b>	ResNet 50	0.789	0.818	0.835	0.843	0.912	0.930
<b>LinkNet</b>	ResNet 101	0.773	0.812	0.832	0.842	0.905	0.924
<b>LinkNet</b>	MobileNetv2	0.783	0.796	0.821	0.843	0.911	0.926
<b>LinkNet</b>	EfficientNet-b0	0.776	0.795	0.821	0.840	0.926	0.927
<b>LinkNet</b>	EfficientNet-b1	0.780	0.801	0.831	0.855	0.937	0.940
<b>LinkNet</b>	EfficientNet-b3	0.812	0.832	0.865	0.889	0.932	0.952
<b>LinkNet</b>	EfficientNet-b5	0.801	0.821	0.847	0.858	0.920	0.939
<b>LinkNet</b>	EfficientNet-b7	0.789	0.810	0.830	0.840	0.892	0.921

Table 5: The results of **FPN** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>FPN</b>		0.791	0.802	0.820	0.840	0.873	0.890
<b>FPN</b>	ResNet 18	0.810	0.833	0.830	0.840	0.887	0.912
<b>FPN</b>	ResNet 34	0.849	0.860	0.869	0.879	0.920	0.933
<b>FPN</b>	ResNet 50	0.840	0.853	0.866	0.876	0.912	0.922
<b>FPN</b>	ResNet 101	0.841	0.842	0.855	0.866	0.901	0.913
<b>FPN</b>	MobileNetv2	0.839	0.850	0.874	0.874	0.923	0.938
<b>FPN</b>	EfficientNet-b0	0.847	0.867	0.873	0.880	0.920	0.933
<b>FPN</b>	EfficientNet-b1	0.857	0.879	0.880	0.899	0.931	0.945
<b>FPN</b>	EfficientNet-b3	0.872	0.886	0.878	0.890	0.933	0.949
<b>FPN</b>	EfficientNet-b5	0.843	0.877	0.867	0.887	0.923	0.937
<b>FPN</b>	EfficientNet-b7	0.845	0.876	0.856	0.877	0.914	0.930

Table 6: The results of **PSPNet** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>PSPNet</b>		0.830	0.850	0.850	0.870	0.899	0.901
<b>PSPNet</b>	ResNet 18	0.845	0.878	0.887	0.891	0.912	0.914
<b>PSPNet</b>	ResNet 34	0.862	0.899	0.892	0.910	0.921	0.949
<b>PSPNet</b>	ResNet 50	0.850	0.876	0.876	0.903	0.919	0.938
<b>PSPNet</b>	ResNet 101	0.840	0.860	0.883	0.900	0.913	0.935
<b>PSPNet</b>	MobileNetv2	0.833	0.855	0.875	0.885	0.907	0.929
<b>PSPNet</b>	EfficientNet-b0	0.831	0.855	0.861	0.877	0.911	0.930
<b>PSPNet</b>	EfficientNet-b1	0.840	0.866	0.878	0.890	0.919	0.925
<b>PSPNet</b>	EfficientNet-b3	0.870	0.896	0.901	0.913	0.927	0.951
<b>PSPNet</b>	EfficientNet-b5	0.855	0.867	0.875	0.893	0.922	0.939
<b>PSPNet</b>	EfficientNet-b7	0.849	0.859	0.857	0.877	0.917	0.935

Table 7: The results of **PAN** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>PAN</b>		0.800	0.822	0.850	0.860	0.892	0.899
<b>PAN</b>	ResNet 18	0.810	0.833	0.855	0.869	0.899	0.912
<b>PAN</b>	ResNet 34	0.846	0.877	0.870	0.875	0.930	0.940
<b>PAN</b>	ResNet 50	0.821	0.844	0.855	0.860	0.910	0.934
<b>PAN</b>	ResNet 101	0.820	0.844	0.856	0.863	0.920	0.921
<b>PAN</b>	MobileNetv2	0.842	0.866	0.853	0.870	0.918	0.920
<b>PAN</b>	EfficientNet-b0	0.846	0.870	0.882	0.890	0.920	0.923
<b>PAN</b>	EfficientNet-b1	0.850	0.873	0.885	0.893	0.923	0.939
<b>PAN</b>	EfficientNet-b3	0.852	0.899	0.886	0.902	0.935	0.949
<b>PAN</b>	EfficientNet-b5	0.820	0.850	0.889	0.900	0.922	0.933
<b>PAN</b>	EfficientNet-b7	0.830	0.856	0.885	0.893	0.910	0.925

Table 8: The results of **DeepLabv3** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	MRecall (initial data)	MRecall (augmented data)
<b>DeepLabV3</b>		0.841	0.851	0.872	0.891	0.922	0.941
<b>DeepLabV3</b>	ResNet 18	0.851	0.870	0.881	0.894	0.930	0.948
<b>DeepLabV3</b>	ResNet 34	0.871	0.892	0.891	0.911	0.932	0.958
<b>DeepLabV3</b>	ResNet 50	0.870	0.883	0.890	0.908	0.931	0.952
<b>DeepLabV3</b>	ResNet 101	0.867	0.876	0.875	0.902	0.932	0.953
<b>DeepLabV3</b>	MobileNetv2	0.870	0.876	0.879	0.891	0.921	0.945
<b>DeepLabV3</b>	EfficientNet-b0	0.850	0.871	0.879	0.890	0.925	0.934
<b>DeepLabV3</b>	EfficientNet-b1	0.856	0.885	0.890	0.912	0.935	0.945
<b>DeepLabV3</b>	EfficientNet-b3	0.878	0.899	0.891	0.916	0.938	0.962
<b>DeepLabV3</b>	EfficientNet-b5	0.877	0.899	0.876	0.917	0.929	0.948
<b>DeepLabV3</b>	EfficientNet-b7	0.870	0.889	0.875	0.913	0.929	0.945

Table 9: The results of **DeepLabv3+** with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>DeepLabV3+</b>		0.861	0.872	0.881	0.893	0.923	0.945
<b>DeepLabV3+</b>	ResNet 18	0.875	0.886	0.898	0.903	0.935	0.950
<b>DeepLabV3+</b>	ResNet 34	0.889	0.901	0.895	0.917	0.947	0.965
<b>DeepLabV3+</b>	ResNet 50	0.879	0.884	0.892	0.916	0.944	0.962
<b>DeepLabV3+</b>	ResNet 101	0.870	0.879	0.882	0.907	0.936	0.957
<b>DeepLabV3+</b>	MobileNetv2	0.874	0.880	0.882	0.895	0.929	0.949
<b>DeepLabV3+</b>	EfficientNet-b0	0.860	0.883	0.890	0.918	0.927	0.948
<b>DeepLabV3+</b>	EfficientNet-b1	0.869	0.885	0.899	0.920	0.938	0.948
<b>DeepLabV3+</b>	EfficientNet-b3	0.892	0.913	0.906	0.928	0.940	0.969
<b>DeepLabV3+</b>	EfficientNet-b5	0.880	0.903	0.908	0.920	0.936	0.952
<b>DeepLabV3+</b>	EfficientNet-b7	0.875	0.894	0.908	0.917	0.930	0.950

Table10-1: The result of architectures with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
<b>U-Net</b>		0.821	0.831	0.850	0.873	0.910	0.920
<b>U-Net</b>	EfficientNet-b3	0.855	0.866	0.887	0.913	0.924	0.946
<b>U-Net++</b>	EfficientNet-b3	0.880	0.899	0.901	0.912	0.930	0.958
<b>MAnet</b>	EfficientNet-b3	0.851	0.879	0.899	0.902	0.938	0.951
<b>Linknet</b>	EfficientNet-b3	0.812	0.832	0.865	0.889	0.932	0.952
<b>FPN</b>	EfficientNet-b3	0.872	0.886	0.878	0.890	0.933	0.949
<b>PSPNet</b>	EfficientNet-b3	0.870	0.896	0.901	0.913	0.927	0.951
<b>PAN</b>	EfficientNet-b3	0.852	0.899	0.886	0.902	0.935	0.949
<b>DeepLabV3</b>	EfficientNet-b3	0.878	0.899	0.891	0.916	0.938	0.962
<b>DeepLabV3+</b>	EfficientNet-b3	0.892	0.913	0.906	0.928	0.940	0.969

Table10-2: The result of architectures with different backbones training in initial data and augmented data.

Architecture	Encoder	Accuracy (initial data)	Accuracy (augmented data)	MIoU (initial data)	MIoU (augmented data)	Recall (initial data)	Recall (augmented data)
U-Net		0.821	0.831	0.850	0.873	0.910	0.920
U-Net	ResNet 34	0.854	0.860	0.880	0.911	0.920	0.933
U-Net++	ResNet 34	0.875	0.889	0.901	0.925	0.946	0.960
MAnet	ResNet 34	0.820	0.844	0.870	0.888	0.922	0.944
Linknet	ResNet 34	0.797	0.821	0.850	0.871	0.922	0.942
PSPNet	ResNet 34	0.862	0.899	0.892	0.910	0.921	0.949
FPN	ResNet 34	0.849	0.860	0.869	0.879	0.920	0.933
PAN	ResNet 34	0.846	0.877	0.870	0.875	0.930	0.940
DeepLabV3	ResNet 34	0.881	0.892	0.891	0.911	0.942	0.962
DeepLabV3+	ResNet 34	0.889	0.901	0.895	0.917	0.947	0.965

By comparing the results from Table 1 to Table 10, the following conclusions can be drawn:

- 1) In Tables 1-10, for models with the same architecture, models with replaced backbones show slightly better precision and intersection over union (IoU) compared to the original unadjusted models.
- 2) In Tables 1-0, within models of the same architecture and backbone structure, models with data augmentation perform better than those without data augmentation.
- 3) Among the model architectures in Tables 1-9, Deeplabv3+ exhibits better overall performance compared to other model structures.
- 4) Among models with the same backbones within the same family, U-Net++ outperforms U-Net, and DeepLabv3+ performs better than DeepLabv3.
- 5) Within the same model structures and backbone series (ResNet 18, 34, 50, 101), in training on this dataset, ResNet 34 shows slightly better results. The training time is longer for ResNet 50 and ResNet 101 models.
- 6) Within the same model structures and backbone series (EfficientNet-b0, b1, b3, b5, b7), in training on this dataset, EfficientNet-b3 shows better results. The training time is shorter for EfficientNet-b3 models. EfficientNet-b5 and EfficientNet-b7 models take longer to train.



## DISCUSSION

Although some research work has been done on the left ventricular ultrasound image segmentation method based on deep learning such as U-Net, U-Net++, MANet, LinkNet, FPN, PSPNet, PAN , DeepLabv3and DeepLabv3+ with other encoder blocks. Through the above work, the following problems in medical image segmentation are mainly solved: (1) the image morphology operation is used to narrow the image segmentation range, and the influence of other tissues around the segmentation target on the segmentation accuracy is solved; (2) the neural network structure with high neuron utilization rate is designed for the problems of low medical image subscale rate, high noise and small data set, which solves the problem that the network is difficult to accurately identify the image (3) the network has strong generalization ability and can automatically learn to adapt to the perceptual field size of the segmentation target according to the training set, which is also applicable to other segmentation tasks.

When training the models, need to pay attention to the version of the Pytorch installation environment and library and package versions ; otherwise, the code is prone to problems. From the results of this study, it can be observed that data augmentation can enhance the segmentation accuracy of the model. Adjusting the backbones in the model can improve the segmentation model's accuracy to some extent, but different backbones have varying impacts on different structural models. Backbones of the same series have diverse effects on segmentation results for the same architecture. Moreover, different series within the same backbones, such as ResNet 18, 34, 50, 101, and EfficientNet-b0, b1, b3, b5, b7, exhibit different segmentation accuracy in various architectures, and the training time of the models also differs. In this experiment, EfficientNet demonstrated good overall performance, with EfficientNet-b3 being the best. Additionally, model training is influenced by the dataset, and for this dataset, the experimental results indicated that DeepLabv3+ achieved better segmentation results. Although there has been some improvement in the segmentation accuracy, the difference is not very big. Therefore, for this dataset in future experiments, it is necessary to compare more encoder modules and try out more segmentation models to select a combination of model architecture and backbones that offer better segmentation results. Additionally, expanding the current dataset and enhancing the overall training data may lead to better results.

## CONCLUSION

The research of left ventricular ultrasound image segmentation is mainly used to assist doctors in diagnosing cardiovascular diseases. On the basis of reading a lot of literature, U-Net, U-Net++, MANet, LinkNet, FPN, PSPNet, PAN, DeepLabv3 and DeepLabv3+ models designed for LV ultrasound segmentation were trained, and the corresponding experimental results were obtained by replacing the encoder blocks in the models with ResNet 18, 34, 50, 101, EfficientNet-b0, b1, b3, b5, b7 and MobileNetv2.

The research work and main results of this report are as follows: (1) To address the problems of low resolution, strong noise, weak edges, and small data set of ultrasound images, the images are preprocessed, including image graying, filtering, size normalization, and data set expansion, and to address the problems that the surrounding tissues (fat or lung, etc.) of the ventricular epicardium of ultrasound images can interfere with the recognition of the segmentation algorithm, the images are thresholded for segmentation, and the target segmentation range of ultrasound images is narrowed by combining the a priori information that the ventricle is located in the upper right part of the image to locate the target region. (2) In order to improve the segmentation accuracy of the segmentation neural network based on the coding and decoding framework, ResNet 18, 34, 50, 101, EfficientNet-b0, b1, b3, b5, b7 and MobileNetv2 coding modules are added to the existing model of segmentation. In order to reduce the loss of image information during the convolutional pooling process of the segmentation network, the segmentation algorithm first combines the advantages of the coding module to further improve the segmentation accuracy of the network.

The experiments show that the same architecture with replaced backbones show slightly better precision and intersection over union (IoU) compared to the original unadjusted models. Within models of the same architecture and backbone structure, models with data augmentation perform better than those without data augmentation. Among the model architectures, DeepLabv3+ exhibits better overall performance compared to other model structures. Among models with the same backbones within the same family, U-Net++ outperforms U-Net, and DeepLabv3+ performs better than DeepLabv3. Within the same model structures and backbone series (ResNet 18, 34, 50, 101), in training on this dataset, ResNet 34 shows slightly better results. The training time is longer for ResNet 50 and ResNet 101 models. Within the same model structures and backbone series (EfficientNet-b0, b1, b3, b5, b7), in training on this dataset, EfficientNet-b3 shows better results. The training time is shorter for EfficientNet-b3 models. EfficientNet-b5 and EfficientNet-b7 models

take longer to train. Using different model architectures with various backbones and data augmentation has improved the segmentation accuracy of the model to some extent.

There are still aspects that need improvement: (1) when preprocessing images, the method of narrowing down the image segmentation varies depending on the dataset, which is due to the difficulty of generalizing the image preprocessing method in this paper because different datasets correspond to different thresholds when binarizing images. (2) Due to the relatively small differences in the comparison of experimental results, additional metrics need to be added for evaluation. (3) the data set is too small, resulting in an algorithm that is more sensitive to the segmentation contour. (4) It is hoped that after this study, a corresponding segmentation system can be designed to assist in segmentation. Through this, it is possible to enhance the application of the segmentation model and system in practical use. For example, human-computer interaction functions can be incorporated into the system. The ideal system is one that will make dynamic adjustments to the segmentation strategy according to the specific situation, and should also be able to make empirical modifications to the segmentation results at a later stage, to apply it to the actual need for continued development, continuous improvement, and maturity.

The focus of future research work on the above issues is: (1) Try out more segmentation model architectures, combine them with different backbones, and then compare and validate them. (2) Explore algorithms that can automatically identify segmentation areas. In future work, we will integrate the advantages of object detection or attention mechanisms with "Segment Anything" to design a network that can real-time detect and narrow down the segmentation scope of images, with strong robustness and high accuracy; (3) Expand the dataset for further comparison and validation of algorithms.

## REFERENCES

1. D'Andrea A., Sperlongano S., Pacileo M., et al. New ultrasound technologies for ischemic heart disease assessment and monitoring in cardiac rehabilitation // *Journal of Clinical Medicine*. 2020. Vol. 9. No. 10. P. 3131.
2. Plewes D. B., Kucharczyk W. Physics of MRI: a primer // *Journal of magnetic resonance imaging*. 2012. Vol. 35. No. 5. P. 1038–1054.
3. Jang J., Ahn C. Y., Jeon K., et al. A reconstruction method of blood flow velocity in left ventricle using color flow ultrasound // *Computational and mathematical methods in medicine*. 2015.
4. Wu S., Yu S., Zhuang L., et al. Automatic segmentation of ultrasound tomography image // *BioMed research international*. 2017.
5. Ma Y., Wang L., Ma Y., Dong M., du Shiqiang, Sun X. An SPCNN-GVF-based approach for the automatic segmentation of left ventricle in cardiac cine MR images // *Int J Comput Assist Radiol Surg*. 2012. Vol. 11. No. 11. P. 1951 - 1964.
6. Kass M., Witkin A., Terzopoulos D. Snakes: Active contour models // *Int J Comput Vision*. 1988. Vol. 1. No. 4. P. 321 - 331.
7. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks // *Adv Neural Inform Process Syst*. 2012. Vol. 25.
8. Chen L.-C., Papandreou G., et al. Deeplab: Semantic image segmentation with deep convolutional nets and fully connected CRFs // *arXiv preprint arXiv:1412.7062*. 2014. P. 1–9.
9. Chen L.-C., Papandreou G., et al. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs // *IEEE Trans Pattern Anal Mach Intell*. 2017. Vol. 40, No. 4. P. 834–848.
10. Chen L. C., Papandreou G., Schroff F., et al. Rethinking atrous convolution for semantic image segmentation // *arXiv preprint arXiv:1706.05587*. 2017. P. 1–9.
11. Chen L. C., Zhu Y., Papandreou G., et al. Encoder-decoder with atrous separable convolution for semantic image segmentation // *European Conference on Computer Vision*. 2018. P. 801–818.
12. Zhao H., Shi J., Qi X., et al. Pyramid scene parsing network // *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017. P. 2881 - 2890.
13. He K., Zhang X., Ren S., et al. Deep residual learning for image recognition // *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016. P. 770 - 778.
14. Long J., Shelhamer E., Darrell T. Fully convolutional networks for semantic segmentation // *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015. P. 3431 - 3440.
15. Ronneberger O., Fischer P., Brox T. U-net: Convolutional networks for biomedical image segmentation // *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention*. 2015. Vol. 18. P. 234 - 241.

16. Zhou Z., Siddiquee M. M. R., Tajbakhsh N., et al. Unet++: A nested u-net architecture for medical image segmentation // *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, 2018. P. 3-11.
17. Fan T., Wang G., Li Y., et al. Ma-net: A multi-scale attention network for liver and tumor segmentation // *IEEE Access*. 2020. Vol. 8. P. 179656-179665.
18. Chaurasia A., Culurciello E. Linknet: Exploiting encoder representations for efficient semantic segmentation // *2017 IEEE Visual Communications and Image Processing (VCIP)*. 2017. P. 1-4.
19. Lin T. Y., Dollár P., Girshick R., et al. Feature pyramid networks for object detection // *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017. P. 2117-2125.
20. Li H., Xiong P., An J., et al. Pyramid attention network for semantic segmentation // *arXiv preprint arXiv:1805.10180*. 2018.
21. Howard A., Zhmoginov A., Chen L. C., et al. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection, and segmentation // *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. P. 4510-4520.
22. Tan M., Le Q. EfficientNet: Rethinking model scaling for convolutional neural networks // *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR. 2019. P. 6105-6114.
23. He K., Gkioxari G., Dollár P., et al. Mask R-CNN // *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017. P. 2961-2969.
24. Wu S., Yu S., Zhuang L., et al. Automatic segmentation of ultrasound tomography image // *BioMed Research International*. 2017.

## APPENDICES

Appendix 1:

The academic papers published during Master period:

1. **Effects of equine-assisted activities and therapies for individuals with autism spectrum disorder: systematic review and meta-analysis.** International journal of environmental research and public health, 20(3), 2630.
2. **"The impact of digital environment on children's cognitive development and mental health"** has been accepted by the journal "*Current Psychology*".
3. **"The Opportunities and Challenges of Artificial Intelligence in Allergy and Immunology"** has been submitted to the journal "*Computers in Biology and Medicine*".

## Appendix 2:

### Project code:

#### Part 1:

```
1.from google.colab import drive
2.drive.mount('/content/drive')
3.import nibabel as nib
4.import numpy as np
5.import matplotlib.pyplot as plt
6.import os
7.folder_path = '/content/drive/MyDrive/CAMUS/CAMUS_public/database_nifti/patient0001/'
8.file_name= 'patient0012_2CH_half_sequence.nii.gz'
9.file_name = 'patient0001_2CH_half_sequence.nii.gz'
10.if os.path.exists(folder_path+file_name):
11.    print('find the file')
12.else:
13.    print('cant find the file ')
14.img = nib.load(folder_path+file_name).get_fdata()
15.print(img.shape)
16.plt.imshow(img[:, :,2], cmap='gray')
17.rotated_img = np.rot90(img[:, :,2], k=-1)
18.plt.imshow(rotated_img, cmap='gray')
19.flipped_img = np.flip(rotated_img, axis=1)
20.plt.imshow(flipped_img, cmap='gray')
21.plt.axis('off')
22.plt.show()
23.mask_img = nib.load(folder_path + 'patient0001_2CH_half_sequence_gt.nii.gz').get_fdata()
24.rotated_img = np.rot90(mask_img[:, :, 2])
25.flipped_img = np.flipud(rotated_img)
26.plt.imshow(flipped_img, cmap='viridis')
27.plt.axis('off')
28.plt.show()
29.flipped_img[flipped_img != 1]=0
30.plt.imshow(flipped_img)
31.plt.imsave("test.png",flipped_img, cmap='gray')
32.from PIL import Image
33.img2 = np.array(Image.open('test.png'))
34.plt.imshow(img2, 'gray')
35.from google.colab import drive
36.drive.mount('/content/drive')
37.import os
38.import nibabel as nib
39.import numpy as np
40.import matplotlib.pyplot as plt
41.# Define the paths for NIfTI files and target folders
42.folder_path = '/content/drive/MyDrive/CAMUS/CAMUS_public/database_nifti'
43.save_folder1 = '/content/drive/MyDrive/CAMUS/CAMUS_public/database_nifti/mask'
44.save_folder2 = '/content/drive/MyDrive/CAMUS/CAMUS_public/database_nifti/image'
45.# Ensure the mask and image folders exist, create them if they don't
46.if not os.path.exists(save_folder1):
47.    os.makedirs(save_folder1)
48.if not os.path.exists(save_folder2):
49.    os.makedirs(save_folder2)
50.# Iterate through each patient's MRI images
51.for i in range(1, 501):
52.    folder_name = f"patient{i:04d}"
53.    image_file = os.path.join(folder_path, folder_name, f"{folder_name}_2CH_half_sequence.nii.
    gz")
```

```

54.mask_file = os.path.join(folder_path, folder_name, f"{folder_name}_2CH_half_sequence_gt.nii.gz
   ")
55. # Read and process the mask image
56. mask_img = nib.load(mask_file).get_fdata()
57. rotated_mask_img = np.rot90(mask_img[:, :, 2])
58. flipped_mask_img = np.flipud(rotated_mask_img)
59. # Binarize the mask: set all non-1 pixel values to 0
60. binary_mask = flipped_mask_img.copy()
61. binary_mask[binary_mask != 1] = 0
62. # Save the processed binary mask to save_folder1
63. plt.imsave(os.path.join(save_folder1, f"img{i}.png"), binary_mask, cmap='gray')
64. # Read and process the original image
65. original_img = nib.load(image_file).get_fdata()
66. rotated_original_img = np.rot90(original_img[:, :, 2])
67. flipped_original_img = np.flipud(rotated_original_img)
68. # Save the original image to save_folder2
69. plt.imsave(os.path.join(save_folder2, f"img{i}.png"), flipped_original_img, cmap='gray')
70.# Output completion message to the terminal
71.print("Processing completed. Binary masks and original images have been saved to the specified
   folders.")

```

## Part 2:

```

1.U-net with Pytorch
2.from google.colab import drive
3.drive.mount('/content/drive')
4.Import torch
5.import random
6.import numpy as np
7.import matplotlib.pyplot as plt
8.import os
9.import copy
10.import time
11.import torch
12.import torch.nn as nn
13.import torch.nn.functional as F
14.import torch.optim as optim
15.import torch.utils.data as data
16.import torchvision
17.from torchvision import transforms, datasets
18.import torchvision.transforms as T
19.from torchsummary import summary
20.from sklearn.metrics import confusion_matrix
21.from sklearn.metrics import ConfusionMatrixDisplay
22.from tqdm.notebook import tqdm, trange
23.import pandas as pd
24.try:
25.    from albumentations.pytorch import ToTensorV2
26.except:
27.# In the last version up for 22.05.2022 the function ToTensorV2 will not work!
28.    !pip install albumentations==0.4.6
29.# try:
30.#     import filetype
31.# except:
32.#     !pip install filetype
33.#     import filetype
34.import albumentations
35.from albumentations.pytorch import ToTensorV2
36.import albumentations as A
37.from albumentations.pytorch import ToTensorV2
38.from torch.utils.data import Dataset, DataLoader
39.def torch_settings():

```



```

40.torch_version = ".".join(torch.__version__.split(".")[0:2])
41.print('torch version:',torch_version)
42.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
43.print('Using device:', device)
44.dtype = torch.float32
45.    if device.type == 'cuda':
46.        cuda_version = torch.__version__.split("+")[-1]
47.        print("cuda: ", cuda_version)
48.
49.        torch.set_default_tensor_type(torch.cuda.FloatTensor)
50.        print('Cuda is available:',torch.cuda.is_available())
51.
52.        n_devices = torch.cuda.device_count()
53.        print('number of devices: %d'%(n_devices))
54.
55.        for cnt_device in range(n_devices):
56.            print(torch.cuda.get_device_name(cnt_device))
57.            print('Memory Usage:')
58.            print('Allocated:', round(torch.cuda.memory_allocated(cnt_device)/1024**3,1), 'GB')
59.
60.            print('Cached:  ', round(torch.cuda.memory_reserved(cnt_device)/1024**3,1), 'GB')
61.
62.    torch.set_default_dtype(dtype) # float32
63.    print('default data type:',dtype)
64.    num_workers=os.cpu_count()
65.    print ('available number of workers:',num_workers)
66.
67.    return device, dtype, num_workers
68.#-----
69.def torch_seed(seed = 42, deterministic = True):
70.    random.seed(seed) # random and transforms
71.    np.random.seed(seed) #numpy
72.    torch.manual_seed(seed) #cpu
73.    torch.cuda.manual_seed(seed) #gpu
74.    torch.backends.cudnn.deterministic=deterministic #cudnn
75.Dataset
76.N_TRAIN = 400
77.N_VAL = 50
78.N_TEST = 50
79.STEP = 1
80.dataset_path = '/content/drive/MyDrive/CAMUS/CAMUS_public'
81.images_directory = os.path.join(dataset_path, "image")
82.masks_directory = os.path.join(dataset_path, "mask")
83.
84.images_filenames = os.listdir(images_directory)
85.
86.# for filtering only image files!!
87.images_filenames = [i for i in images_filenames
88.                    if os.path.splitext(i)[-1]
89.                    in ('.png', '.jpg', '.jpeg', '.tiff', '.bmp', '.gif')]
90.
91.os.path.splitext('/path/to/somefile.ext')
92.
93.random.shuffle(images_filenames)
94.
95.train_images_filenames = images_filenames[:N_TRAIN:STEP]
96.val_images_filenames = images_filenames[N_TRAIN:N_TRAIN+N_VAL:STEP]
97.test_images_filenames = images_filenames[N_TRAIN+N_VAL:N_TRAIN+N_VAL+N_TEST:STEP]
98.
99.print(len(train_images_filenames), len(val_images_filenames), len(test_images_filenames))
100.For uploading data let's also create our dataset class as inheritance of Dataset form PyTorch.
101.
102.from torch.utils.data import Dataset

```

```

100.class CAMUSDataset(Dataset):
101.     def __init__(self, images_filenames, images_directory, masks_directory, masks_filenames=None,
102.                  transforms=None, device = device):
103.         self.images_filenames = images_filenames
104.         self.images_directory = images_directory
105.         self.masks_directory = masks_directory
106.         # if masks_filenames != None:
107.         #     self.masks_filenames = masks_filenames
108.         # else:
109.         #     masks_filenames = images_filenames
110.         self.transforms = transforms
111.
112.     def __len__(self):
113.         return len(self.images_filenames)
114.
115.     def preprocess_mask(self, mask):
116.         mask = np.asarray(mask).astype(np.float32)
117.         mask = mask / 255
118.         # mask[mask == 2.0] = 0.0
119.         # mask[(mask == 1.0) | (mask == 3.0)] = 1.0
120.         return mask
121.
122.     def __getitem__(self, idx):
123.
124.         image_filename = self.images_filenames[idx]
125.         # masks_filename = self.masks_filenames[idx]
126.         path_ = os.path.join(images_directory, image_filename)
127.         image = np.asarray(Image.open(path_).convert("RGB"))
128.
129.         path_ = os.path.join(masks_directory, image_filename)
130.         mask = Image.open(path_)
131.         mask = self.preprocess_mask(mask)
132.
133.         if self.transforms is not None:
134.             transformed = self.transforms(image=image, mask=mask)
135.             image = transformed["image"]
136.             mask = transformed["mask"]
137.
138.         return image[:3,:,:].to(device), mask[:, :, 0].to(device)
139.
140.mean = [0.485, 0.456, 0.406]
141.std = [0.229, 0.224, 0.225]
142.
143.train_transform = A.Compose([A.Resize(280, 280),
144.                              A.RandomCrop(256,256),
145.                              A.ShiftScaleRotate(0.2,0.2, 30),
146.                              A.RGBShift(25,25,25),
147.                              A.RandomBrightnessContrast(0.3,0.3),
148.                              A.Normalize(mean, std),
149.                              ToTensorV2(),])
150.
151.test_transform = A.Compose([A.Resize(256, 256),
152.                              A.Normalize(mean, std),
153.                              ToTensorV2()])
154.
155.train_dataset = CAMUSDataset(train_images_filenames, images_directory, masks_directory, tran
sforms=train_transform,)
156.
157.val_dataset = CAMUSDataset(val_images_filenames, images_directory, masks_directory, trans
forms=test_transform,)
158.

```

```

159.test_dataset = CAMUSDataset(test_images_filenames, images_directory, masks_directory, trans
    forms=test_transform,)
160.
161.print(f"Train size: {len(train_dataset)}")
162.print(f"Valid size: {len(val_dataset)}")
163.print(f"Test size: {len(test_dataset)}")
#So we can visualize the data using our visualize_data function
164.from PIL import Image
165.def visualize_dataset(dataset, samples=5, predicted_masks = None, mean = mean, std = std):
166.    dataset = copy.deepcopy(dataset)
167.    cols = 3 if predicted_masks else 2
168.
169.    figure, ax = plt.subplots(nrows=samples, ncols=cols, figsize=(10, 12))
170.
171.    for i in range(samples):
172.        image, mask = dataset[i]
173.
174.        image = image.data.cpu().numpy().transpose((1,2,0))
175.        image = np.clip(image*std + mean,0,1)
176.
177.        mask = np.clip(mask.data.cpu().numpy(),0,1)
178.
179.        ax[i, 0].imshow(image)
180.        ax[i, 0].set_title("Image"); ax[i, 0].set_axis_off()
181.        ax[i, 1].imshow(mask, interpolation="nearest")
182.        ax[i, 1].set_title("Ground Truth"); ax[i, 1].set_axis_off()
183.        if predicted_masks:
184.            predicted_mask = predicted_masks[i]
185.            ax[i, 2].imshow(predicted_mask, interpolation="nearest")
186.            ax[i, 2].set_title("Predicted Mask"); ax[i, 2].set_axis_off()
187.
188.    plt.tight_layout()
189.    plt.show()
190.visualize_dataset(test_dataset, samples=5)
191.
#Dataloaders
192.BATCH = 4
193.if torch.cuda.is_available():
194.    kwarg = {'generator':torch.Generator(device='cuda'), 'pin_memory':True}
195.else:
196.    kwarg = {'num_workers':0}
197.
198.train_loader = DataLoader(
199.    train_dataset,
200.    batch_size=BATCH,
201.    shuffle=True,
202.    **kwarg)
203.
204.val_loader = DataLoader(
205.    val_dataset,
206.    batch_size=BATCH,
207.    shuffle=False,
208.    **kwarg)
209.
210.test_loader = DataLoader(
211.    test_dataset,
212.    batch_size=BATCH,
213.    shuffle=False,
214.    **kwarg)
215.SIZE = 32
216.def conv_block(in_channels, out_channels, mid_channels = None):
217.    if mid_channels is None:

```

```

218.         mid_channels = out_channels
219.
220.     return nn.Sequential(
221.         nn.Conv2d(in_channels, mid_channels, 3, padding=1, bias = False),
222.         nn.BatchNorm2d(mid_channels),
223.         nn.ReLU(inplace=True),
224.         nn.Conv2d(mid_channels, out_channels, 3, padding=1, bias = False),
225.         nn.BatchNorm2d(mid_channels),
226.         nn.ReLU(inplace=True))
227.
228. class UNet(nn.Module):
229.
230.     def __init__(self, n_channels = 1, n_class=1):
231.         super().__init__()
232.
233.         self.down1 = conv_block(n_channels, SIZE)
234.         self.down2 = conv_block(SIZE, SIZE*2)
235.         self.down3 = conv_block(SIZE*2, SIZE*4)
236.         self.down4 = conv_block(SIZE*4, SIZE*8)
237.
238.         self.maxpool = nn.MaxPool2d(2)
239.
240.         self.upsample = nn.Upsample(scale_factor=2,
241.                                     mode='bilinear',
242.                                     align_corners=True)
243.
244.
245.         self.up3 = conv_block(SIZE*4 + SIZE*8, SIZE*4)
246.         self.up2 = conv_block(SIZE*2 + SIZE*4, SIZE*2)
247.         self.up1 = conv_block(SIZE*2 + SIZE, SIZE)
248.
249.         self.out = nn.Conv2d(SIZE, n_class, 3, padding=1)
250.
251.     #-----
252.     def forward(self, x):
253.
254.         #ENCODER
255.         conv1 = self.down1(x)
256.         x = self.maxpool(conv1)
257.
258.         conv2 = self.down2(x)
259.         x = self.maxpool(conv2)
260.
261.         conv3 = self.down3(x)
262.         x = self.maxpool(conv3)
263.
264.         x = self.down4(x)
265.
266.         #DECODER
267.         x = self.upsample(x)
268.         x = torch.cat([x, conv3], dim=1)
269.
270.         x = self.up3(x)
271.         x = self.upsample(x)
272.         x = torch.cat([x, conv2], dim=1)
273.
274.         x = self.up2(x)
275.         x = self.upsample(x)
276.         x = torch.cat([x, conv1], dim=1)
277.
278.         x = self.up1(x)
279.
280.         out = self.out(x)

```

```

281.
282.     return out
283.
284. For implementation we create dice_loss and class of new loss DiceBCELoss.
285. def dice_loss(pred, target, smooth=1):
286.     #flatten label and prediction tensors
287.
288.     pred     = pred.view(-1)
289.     target   = target.view(-1)
290.
291.     intersection = (pred * target).sum()
292.     loss = 1 - (2.*intersection + smooth)/(pred.sum() + target.sum() + smooth)
293.
294.     return loss
295.
296. class DiceBCELoss(nn.Module):
297.     def __init__(self, weight=1):
298.         super().__init__()
299.         self.weight = weight
300.
301.     def forward(self, pred, target, smooth=1):
302.
303.     #comment out if your model contains a sigmoid or equivalent activation layer
304.         pred = torch.sigmoid(pred)
305.
306.         dice = dice_loss(pred, target, smooth=1)
307.
308.         bce = F.binary_cross_entropy(pred.squeeze(), target.squeeze(), reduction='mean')
309.         dice_bce = self.weight*bce + dice*(1-self.weight)
310.
311.         return dice_bce
312. LR = 0.001
313. model = model.to(device)
314. criterion = DiceBCELoss(weight = 0.5)
315. criterion = criterion.to(device)
316. optimizer = torch.optim.Adam(model.parameters(), lr=LR)
317. describe accuracy as dice coefficient
318. dice coefficient=2·(A∩B)|A|+|B|→accuracy
319. def accuracy(y_pred, y):
320.     return 1-dice_loss(torch.sigmoid(y_pred), y, smooth=1)
321. #-----
322. def train(model, dataloader, optimizer, criterion, metrics, device):
323.     epoch_loss = 0
324.     epoch_acc  = 0
325.     model.train()
326.     for (x, y) in tqdm(dataloader, desc="Training", leave=False):
327.         x = x.to(device)
328.         y = y.to(device)
329.
330.         optimizer.zero_grad()
331.         y_pred = model.forward(x)
332.
333.         loss = criterion(y_pred, y)
334.         acc  = metrics( y_pred, y)
335.
336.         loss.backward()
337.         optimizer.step()
338.
339.         epoch_loss += loss.item()
340.         epoch_acc  += acc.item()
341.
342.     return epoch_loss / len(dataloader), epoch_acc / len(dataloader)

```

```

343. #-----
344. def evaluate(model, dataloader, criterion, metrics, device):
345.
346.     epoch_loss = 0
347.     epoch_acc = 0
348.
349.     model.eval()
350.
351.     with torch.no_grad():
352.
353.         for (x, y) in tqdm(dataloader, desc="Evaluating", leave=False):
354.
355.             x = x.to(device)
356.             y = y.to(device)
357.
358.             y_pred = model.forward(x)
359.
360.             loss = criterion(y_pred, y)
361.             acc = metrics(y_pred, y)
362.
363.             epoch_loss += loss.item()
364.             epoch_acc += acc.item()
365.
366.     return epoch_loss / len(dataloader), epoch_acc / len(dataloader)
367. #-----
368.
369. def epoch_time(start_time, end_time):
370.     elapsed_time = end_time - start_time
371.     elapsed_mins = int(elapsed_time / 60)
372.     elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
373.     return elapsed_mins, elapsed_secs
374.
#weights init
375. def init_weights(m):
376.
377.     if isinstance(m, nn.Conv2d):
378.         nn.init.kaiming_normal_(m.weight)
379.
380.     elif isinstance(m, nn.Linear):
381.         nn.init.xavier_uniform_(m.weight)
382.
383.     if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
384.         if m.bias != None:
385.             m.bias.data.fill_(0)
386.
387. model.apply(init_weights);
388.
389. EPOCHS = 20
390. train_loss = torch.zeros(EPOCHS)
391. valid_loss = torch.zeros(EPOCHS)
392. train_acc = torch.zeros(EPOCHS)
393. valid_acc = torch.zeros(EPOCHS)
394. best_valid_loss = float('inf')
395. best_epoch = 0
396. for epoch in trange(EPOCHS, desc="Epochs"):
397.     start_time = time.monotonic()
398.     train_loss[epoch], train_acc[epoch] = train(model,
399.                                                train_loader,
400.                                                optimizer,
401.                                                criterion,
402.                                                accuracy,
403.                                                device)

```

```

404.
405.     valid_loss[epoch], valid_acc[epoch] = evaluate(model,
406.                                                    val_loader,
407.                                                    criterion,
408.                                                    accuracy,
409.                                                    device)
410.
411.     if valid_loss[epoch] < best_valid_loss:
412.         best_valid_loss = valid_loss[epoch]
413.         best_epoch = epoch
414.         torch.save(model.state_dict(), 'best_model.pt')
415.
416.     epoch_mins, epoch_secs = epoch_time(start_time, time.monotonic())
417.     if epoch%2 == 1:     # print every 2 epochs:
418.         print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
419.         print(f'¥tTrain Loss: {train_loss[epoch]:.3f} | Train Acc: {train_acc[epoch]*100:.2f}
420.             %')
421.         print(f'¥t Val. Loss: {valid_loss[epoch]:.3f} | Val. Acc: {valid_acc[epoch]*100:.2f}
422.             %')
421.
#check the outputs
422.x,y = next(iter(val_loader))
423.x = x.to(device)
424.y = y.to(device)
425.y_pred = torch.sigmoid(model.forward(x))
426.plt.imshow(y_pred[1,0, :, :].detach().cpu().numpy()); plt.show()
427.plt.imshow(y[1, :, :].detach().cpu().numpy()); plt.show()

428.threshold = 0.5
429.y_pred[y_pred>=threshold]=1
430.y_pred[y_pred<threshold]=0
431.plt.imshow(y_pred[1,0, :, :].detach().cpu().numpy()); plt.show()

```