Table of Contents

[INTRODUCTION 2](#_Toc126523142)

[Machine Learning and Computer Vision in Medicine 3](#_Toc126523143)

[Our Research 4](#_Toc126523144)

[Article analysis results 8](#_Toc126523145)

[Conclusion in analysis part 12](#_Toc126523146)

[Model development 12](#_Toc126523147)

[Conclusion 28](#_Toc126523148)

[References 29](#_Toc126523149)

## INTRODUCTION

The METLAIN LLP company is engaged in the development of technological machines, installations and the design of its own equipment or the assembly of existing installations that they order.

More recently, the company opened a data department and is actively developing such departments as Data Engineers and Data Scientist.

In order to gain experience for the development of a neural network, I was given free time to study articles provided by another organization.

## Machine Learning and Computer Vision in Medicine

Machine learning is increasingly being used in the field of medicine to improve patient outcomes and enhance the delivery of healthcare services. Some examples of how machine learning is used in medicine include:

1. Diagnosis: Machine learning algorithms can be used to analyze medical images, such as X-rays and CT scans, to assist in the diagnosis of medical conditions.
2. Predictive modeling: Machine learning algorithms can be trained to analyze patient data, such as medical history and lab results, to predict the likelihood of a particular health outcome, such as the development of a disease.
3. Personalized medicine: Machine learning algorithms can be used to analyze individual patient data to inform personalized treatment plans and dosages, improving the effectiveness of treatments.
4. Clinical decision support: Machine learning algorithms can be integrated into electronic health record systems to provide real-time decision support for clinicians, helping them make informed treatment decisions.
5. Drug discovery: Machine learning algorithms can be used to analyze large datasets of genetic and molecular information to identify new targets for drug development.
6. Outcome prediction: Machine learning algorithms can be used to analyze patient data to predict outcomes, such as hospital readmission rates, allowing healthcare providers to proactively manage patient care and reduce costs.

These are just a few examples of how machine learning is being used in medicine to improve patient outcomes and enhance the delivery of healthcare services. While machine learning offers many benefits, it is important to carefully evaluate the accuracy and ethical implications of these systems to ensure that they are used responsibly.

In our case, while writing a dissertation, we used computer vision to process images of the mammary gland, which in some cases was affected by malignant tumors. That is, we were faced with the task of determining whether the gland is affected in general and the degree of its damage.

Recently, computer vision has been widely used both in medicine and in other industries. Consider the application of computer vision in medicine.

Computer vision is a field of study within computer science that focuses on enabling computers to interpret and understand visual information from the world. In the context of malignant tumor detection, computer vision techniques can be used to analyze medical images, such as X-rays, CT scans, and MRI scans, to identify and classify potential tumors. Some ways computer vision is used in malignant tumor detection include:

1. Image segmentation: Computer vision algorithms can be used to segment images, separating the tumors from the surrounding tissue, making it easier to identify and analyze the tumors.
2. Object detection: Computer vision algorithms can be trained to detect and identify potential tumors in medical images, marking them for further analysis.
3. Tumor classification: Computer vision algorithms can be used to classify tumors as either benign or malignant based on their appearance in medical images.
4. Tumor grading: Computer vision algorithms can be used to grade the severity of malignant tumors based on factors such as size, shape, and texture.
5. Image registration: Computer vision algorithms can be used to align multiple images of the same patient taken at different times, making it easier to track the growth and progression of a tumor over time.

By using computer vision techniques in malignant tumor detection, healthcare providers can improve the accuracy and efficiency of the diagnostic process, leading to better patient outcomes. However, it is important to note that computer vision is not a replacement for human expertise, and the results of computer vision systems should always be interpreted and validated by trained medical professionals.

## Our Research

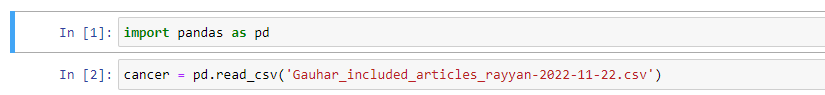
Our team was provided with data on breast cancer by an oncology center located in Almaty. Also, a representative of the cancer center provided a list of scientific articles that would help us during the research process. The list included 1848 article titles and was stored in a CSV file.

Изображение выглядит как текст

Автоматически созданное описание

Not all articles fit our thesis topic, so it was necessary to extract those articles that would suit us. A Python script was written that extracts the titles of articles from the general list by keywords.

First it was necessary to read data from a file, this was done using a library called pandas:



Then it was necessary to get information about the resulting dataset (column names, dataset size):

Изображение выглядит как текст

Автоматически созданное описание

To extract articles by keywords, the PandaSQL library was used, that is, we made queries to the dataset.



To begin with, we have selected articles that describe the problems associated with breast cancer.

Изображение выглядит как текст, снимок экрана, внутренний, несколько

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, внутренний, несколько

Автоматически созданное описание

We then extracted articles related to deep learning.

Изображение выглядит как текст, снимок экрана, внутренний

Автоматически созданное описание

Then we created a dataset and wrote it to a CSV file:



Изображение выглядит как текст

Автоматически созданное описание

Then we manually selected a list of 95 articles that were most suitable for us for further research.

Изображение выглядит как текст, газета, документ

Автоматически созданное описание

The result of the analysis was recorded by us in excel. We fixed the title of the article, url, the result of the analysis of the article and the method of Machine Learning used.

Изображение выглядит как текст

Автоматически созданное описание

## Article analysis results

Article independent clinical validation of the automated ki67 scoring guideline from the international ki67 in breast cancer working group.

The study aimed to assess the reproducibility and prognostic potential of Ki67 scoring in breast cancer. Four pathologists built algorithms based on the guidelines of the International Ki67 in Breast Cancer Working Group (IKWG) using a digital image analysis platform. The algorithms were applied to a study cohort of 157 ER+ breast cancer patients with 15 years of follow-up. High interobserver reliability was achieved with an ICC of 0.938, and the hazard ratios showed independent prognostic potential. The results suggest the IKWG DIA instructions have potential for clinical use, but a prospective study is needed to fully assess its utility.

Multi-Parametric MRI-Based Radiomics Models for Predicting Molecular Subtype and Androgen Receptor Expression in Breast Cancer.

This study analyzed 162 patients with breast cancer and aimed to predict molecular subtype and androgen receptor (AR) expression using multi-parametric MRI-based radiomics and machine learning techniques. The study found that there was a difference in menopausal status, ER, PR, HER2, and Ki-67 index between low-AR and high-AR groups, and there was also a difference in AR expression and Ki-67 among the three molecular subtypes. The Multilayer Perceptron (MLP) method showed the best performance in predicting AR expression and molecular subtype, with an accuracy of 85.8% and 72.5%, respectively. The results suggest that multi-parametric MRI-based radiomics and machine learning can be a promising non-invasive method for predicting breast cancer molecular subtype and AR expression.

Evaluating the Accuracy of Breast Cancer and Molecular Subtype Diagnosis by Ultrasound Image Deep Learning Model.

In this study, a deep learning model (DLM) was developed to diagnose breast tumors using ultrasound images. The DLM was trained with 2,822 images, tested on 707 images and validated on 210 images. The results showed that the DLM had higher accuracy, sensitivity and specificity compared to the Breast Imaging Reporting and Data System (BI-RADS) categorization used in clinics, with a diagnostic accuracy of 92.86% and reduced unnecessary biopsies by 67.86%. The DLM also had good prediction ability for molecular subtypes of breast cancer, such as triple-negative, HER2 (+), and HR (+). The study concluded that the DLM has great potential in reducing unnecessary biopsies and has specific clinical application value in predicting molecular subtypes of breast cancer.

A Computational Tumor-Infiltrating Lymphocyte Assessment Method Comparable with Visual Reporting Guidelines for Triple-Negative Breast Cancer.

The conclusion of this text is that a deep learning-based computational Tumor-infiltrating lymphocytes (CTA) assessment method was established and compared with the standard visual TILs assessment (VTA) in a cohort of Asian and Caucasian patients with triple-negative breast cancer. The results showed a moderate to good correlation between the manual and automatic TIL scores, and the aTIL score was found to be prognostic of disease-free survival. The study suggests that a workflow combining both VTA and CTA may be useful for pathologists in evaluating risk and making decisions for TNBC patients.

Space curvature-inspired nanoplasmonic sensor for breast cancer extracellular vesicle fingerprinting and machine learning classification.

This text is about a new method for characterizing extracellular vesicles (EVs) for use as a liquid biopsy target in cancer research. The method involves using a novel surface-enhanced Raman spectroscopy substrate to fingerprint and classify three subtypes of breast cancer EVs. The findings show that this platform is capable of effectively classifying the different subtypes and has the potential to be used in clinical applications to improve human health.

Deep Learning-Based Prediction Model for Breast Cancer Recurrence Using Adjuvant Breast Cancer Cohort in Tertiary Cancer Center Registry.

The study aimed to determine the probability of breast cancer (BC) recurrence in real time by developing a BC prognosis model. A machine learning model, Weibull Time To Event Recurrent Neural Network (WTTE-RNN), was applied to data collected from 13,117 patients. The model considered 32 features related to BC recurrence and was trained on 60% of the data, with 20% each for validation and testing. The results showed that the deep learning-based model outperformed three other machine learning-based models, with a C-index of 0.92 for the training data set and 0.89 for the validation and test data sets. The area under the curve (AUC) values were 0.90 at 2-year point, 0.91 at 5-year point, and 0.91 at 7-year point. The model showed great results in terms of pathologic characteristics, with a median absolute error (MAE) and weighted mean absolute error (wMAE) as low as 3.5%.

Tumour Stroma Ratio Assessment Using Digital Image Analysis Predicts Survival in Triple Negative and Luminal Breast Cancer.

This study aimed to investigate the relationship between tumour stroma ratio (TSR) and prognosis in breast cancer patients. A cohort of 647 patients with 403 luminal and 244 triple negative breast cancer (TNBC) were analyzed using QuPath software and digital H&E images of tissue microarrays (TMAs). The results showed that low TSR (high stroma) in TNBC was associated with poor overall survival (OS) and breast cancer specific survival (BCSS), while low TSR in luminal tumours was associated with a favorable prognosis for OS but not for BCSS. The study also highlights the inverse relationship between TSR and stromal percentage, and the significance of tumour subtype stromal phenotype in further refining these findings. The TNBC cut-point of TSR was validated in the whole TNBC cohort.

Multi-Class Classification of Breast Cancer Using 6B-Net with Deep Feature Fusion and Selection Method.

This study proposed a novel deep learning-based solution for multi-class breast cancer classification from histopathology images. A deep CNN model named 6B-Net was developed for the classification of four and eight classes of breast cancer, with a feature fusion and selection mechanism. The method was evaluated using two large public datasets, achieving a multi-class average accuracy of 94.20% and 90.10% for four and eight classes respectively. The results showed the highest multi-class average accuracy in breast cancer classification, making the proposed method useful for early and accurate diagnosis of breast cancer.

NuCLS: A scalable crowdsourcing approach and dataset for nucleus classification and segmentation in breast cancer.

This article describes how to use the Mask R-CNN model. This article describes a novel collaborative framework for engaging crowds of medical students and pathologists to produce quality labels for cell nuclei. We used this approach to produce the NuCLS dataset, containing >220,000 annotations of cell nuclei in breast cancers. This article presents data and analysis results for single and multi-rater annotations from both non-experts and pathologists. NuCLS is a large-scale multi-class dataset generated by engaging crowds of medical students and pathologists. NuCLS is sourced from the same images as the Breast Cancer Semantic Segmentation (BCSS) dataset.

A Novel Multistage Transfer Learning for Ultrasound Breast Cancer Image Classification

This article describes several machine learning models and compares their performance.

The cancer cell lines [33,34] for this experiment were cultured for seven days, and bright-field images were acquired every day using an inverted fluorescent microscopeИзображение выглядит как стол

Автоматически созданное описание

Construction and Validation of a Prognostic Risk Model for Triple-Negative Breast Cancer Based on Autophagy-Related Genes

A total of 222 autophagy-related genes were downloaded from The Human Autophagy Database. The RNA-sequencing data and corresponding clinical data of TNBC were obtained from The Cancer Genome Atlas (TCGA) database. Based on these data, a risk prediction model was built.

Immune Effective Score as a Predictor of Response to Neoadjuvant Trastuzumab Therapy and a Prognostic Indicator for HER2-Positive Breast Cancer

Single nucleotide variation (SNV) and copy number variation (CNV) data of TCGABRCA were obtained from the GDC Data Portal.

Lymph Node Metastasis Status in Breast Carcinoma Can Be Predicted via Image Analysis of Tumor Histology

An algorithm for predicting the state of the lymph node has been developed. The dataset contained data from 101 patients. Using digital image processing, patient images were classified.

Digital Microscopy Assessment of Angiogenesis in Different Breast Cancer Compartments

Whole slide digital examination and region of interest (ROI) analysis are a valuable tool in scoring angiogenesis markers and disclosing their prognostic capacity. Were analyzed 50 samples of carcinoma of patients between 37 and 70 years old (mean age 57), diagnosed with breast invasive carcinoma, NST (invasive ductal carcinoma, NOS).

Reverse phase protein array based tumor profiling identifies a biomarker signature for risk classification of hormone receptor-positive breast cancer

Tumor samples were classified using three algorithms: SVM, RF and PAM (prediction analysis for microarrays). Tumor specimens (discovery set, n = 109) from patients diagnosed with primary invasive breast carcinoma were collected at the time of surgery between 2008 and 2010 at the Department of Gynecology and Obstetrics/National Center for Tumor Diseases, Heidelberg. They implemented the software in the R programming language and made it available through the bootfs R-package.

Reverse-Phase Protein Array for Prediction of Patients at Low Risk of Developing Bone Metastasis From Breast Cancer

Tumor samples were obtained from 169 patients with primary invasive breast carcinoma who underwent surgery. The patients were categorized by whether they developed breast cancer bone metastasis (BCBM) during follow-up. Clinical characteristics and protein expression by RPPA were compared and verified by leave-one-out cross-validation. Lymph node status (p = .023) and expression level of 22 proteins by RPPA were significantly correlated with BCBM in logistic regression analysis. These variables were used to build a logistic regression model. After filtering the variables through a stepwise algorithm, the final model, consisting of 8 proteins and lymph node status, had sensitivity of 30.0%, specificity of 90.5%, positive predictive value of 30.0%, and negative predictive value of 90.5% in the cross-validation.

## Conclusion in analysis part

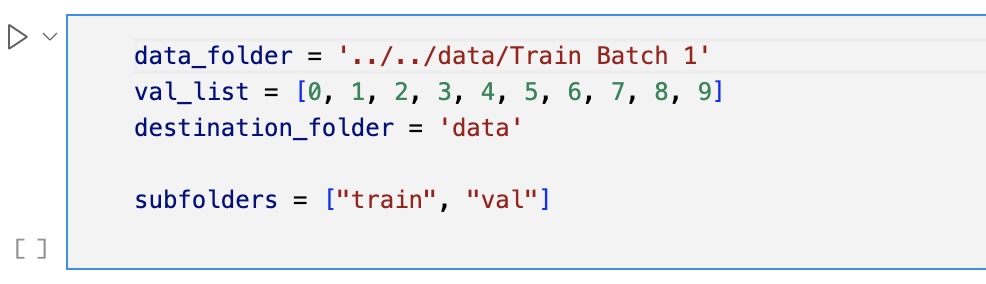
As a result of the analysis, the main aspects were swept up and described in an Excel file, it took a lot of effort from each of the team to analyze. The analysis gave us an understanding in which direction we need to prescribe a neural network model.

## Model development

Data preprocessing

As a rule, to create a quality model it does not matter whether it is a classical machine learning model, or a complex artificial intelligence based on the use of a neural network you need data. The main problem is reading the data, in our example we use NIFTI medical images, which are MRI scans. Since the image data is quite heavy and this makes it difficult to learn, it is necessary to pre-process the data.

It is assumed that the MRI images are in the source folder, which is defined as '.../.../data/Train Batch 1':



The script begins by creating the destination folder and its subfolders (train and val) if they do not already exist:



First, you need to provide information about the get\_patient() function:



The patient number is obtained by splitting the file name with '-' and '.'.

Next, we load the file using the nibabel library. Convert the file to a NumPy array using the get\_data() method. Trims and standardizes the data if it is a volume file (indicated by the file name beginning with vol). Selects only tum as a positive label if it is a segmentation file (indicated by a filename beginning with seg). Transposes the data so that the z-axis (slices) is the first dimension. Traverses all slices and saves each slice in a new location (train or val) using the np.save method.

This script is written in Python and uses the NumPy and Nibabel libraries to perform its tasks.



The idea exactly why you should use the NumPy format is a widely used and powerful library for numerical computing in Python. It provides efficient and convenient ways to manipulate and process multidimensional arrays, which makes it well suited for image processing tasks.

In this case, MRI images are loaded as NIFTI files and then converted to NumPy arrays for processing and storage in a more efficient and convenient format for further use in machine learning or computer vision tasks.

Conversion to NumPy arrays also allows images to be processed and manipulated using various NumPy functions, such as transpose, cut and slice, before being saved as individual slices to the destination folder.

Data loading

The idea behind this file and class is to implement custom datasets for use in the PyTorch machine learning project. The dataset is used for the liver segmentation task and consists of 2D CT scans of the liver and its lesions. The data files in the dataset directory are sorted by type (first character of file name), patient ID, and item number.

There are auxiliary functions present, namely load\_file and load\_file\_context is similar, for context load\_file\_context differs from load\_file it loads the previous and next slices of the sample.

Let's dwell on the auxiliaries, starting with load\_file.

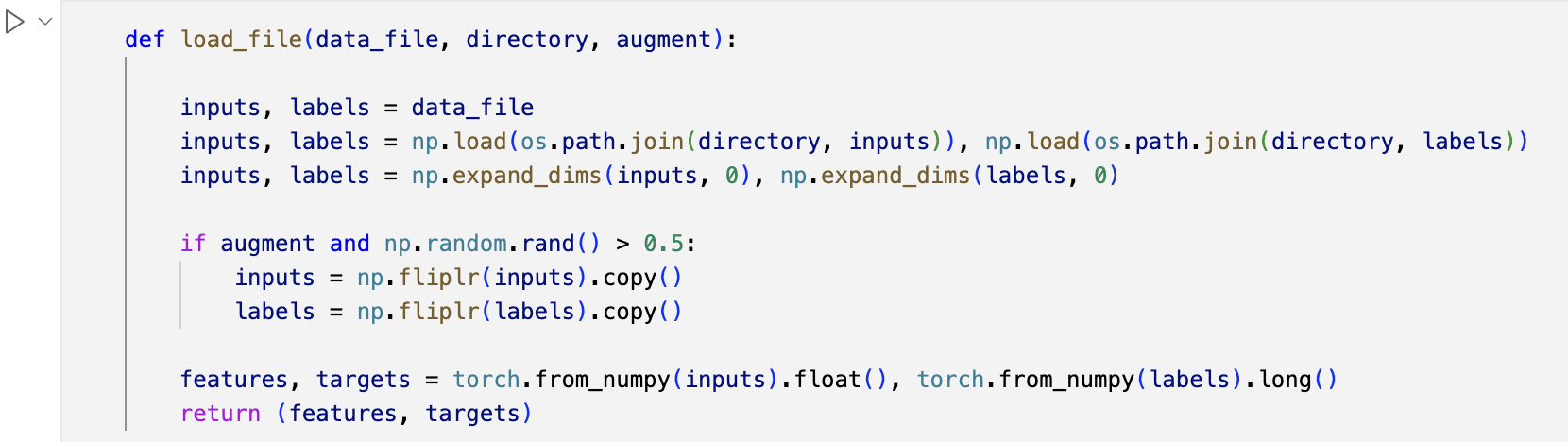
The load\_file method is used to load a data file and perform a data addition if one is specified. The method takes three arguments:

* data\_file: a tuple containing two elements - input data file name and labels.
* directory: the path to the directory where the data files are stored.
* augment: a boolean value indicating whether the data should be augmented or not.

The method first loads the inputs and labels data files from the specified directory with np.load, and then adds an additional dimension to each of them with np.expand\_dims.

If augment is set to True, the method checks if the random value generated by np.random.rand() is 0.5. If the condition is true, the inputs are flipped with np.fliplr and the labels are also flipped accordingly.

The inputs and labels are then converted into PyTorch tensors using torch.from\_numpy and returned as a tuple.



The load\_file\_context function takes five input arguments:

* data\_files: a list of tuples, each containing an input file name and a corresponding label.
* idx: an integer representing the index of the average fragment in the list.
* context: an integer representing the number of fragments loaded before and after the middle fragment.
* directory: the path to the directory where the input data and labels are stored.
* augment: a Boolean value that specifies whether the inputs should be randomly flipped horizontally or not.

The function loads the middle slice, labeled idx, and its corresponding label by calling np.load on the file paths stored in the data\_files list. The loaded inputs and labels are then expanded along the first axis to add batch dimensionality.

If augment is True and the result of np.random.rand() is greater than 0.5, the inputs are flipped horizontally with np.fliplr, and the labels are flipped accordingly.

The function then loads the slices before the middle slice and the slices after the middle slice, looping through the indices between idx-context and idx+context+1. If the slice belongs to another patient or is out of bounds, the middle slice is taken instead. If augment is True, the loaded inputs are flipped horizontally.

Finally, the function combines all slices along the first axis to form a context and converts the inputs and labels into torch.tensors using torch.from\_numpy. The function returns a tuple of inputs (traits) and labels (targets).



Moving on to the implementation of the LiverDataSet class

The class has the following methods:

* \_\_init\_\_: initializes the dataset. The directory argument specifies the directory in which the data files are stored. The augment argument specifies whether the data should be augmented. The context argument is used to load the context of the slice (previous and next slices).
* \_\_getitem\_\_: returns one data sample from the dataset, either with or without context. If the context is greater than zero, the load\_file\_context function is called. Otherwise the load\_file function is called.
* \_\_len\_\_: returns the number of samples in the dataset.
* getWeights: calculates and returns the weights of the samples. It calculates positive and negative samples and returns weights according to their proportions.
* getPatients: returns a dictionary with patient IDs as keys and lists of sample indices belonging to each patient as values.

Networks

class UNetSmall(nn.Module):

def \_\_init\_\_(self, dice=False):

super(UNetSmall, self).\_\_init\_\_()

self.conv1\_input = nn.Conv2d(1, 64/2, 3, padding=1)

self.conv1 = nn.Conv2d(64/2, 64/2, 3, padding=1)

self.conv2\_input = nn.Conv2d(64/2, 128/2, 3, padding=1)

self.conv2 = nn.Conv2d(128/2, 128/2, 3, padding=1)

self.conv3\_input = nn.Conv2d(128/2, 256/2, 3, padding=1)

self.conv3 = nn.Conv2d(256/2, 256/2, 3, padding=1)

self.conv4\_input = nn.Conv2d(256/2, 512/2, 3, padding=1)

self.conv4 = nn.Conv2d(512/2, 512/2, 3, padding=1)

self.conv7\_up = nn.ConvTranspose2d(512/2, 256/2, 2, 2)

self.conv7\_input = nn.Conv2d(512/2, 256/2, 3, padding=1)

self.conv7 = nn.Conv2d(256/2, 256/2, 3, padding=1)

self.conv8\_up = nn.ConvTranspose2d(256/2, 128/2, 2, 2)

self.conv8\_input = nn.Conv2d(256/2, 128/2, 3, padding=1)

self.conv8 = nn.Conv2d(128/2, 128/2, 3, padding=1)

self.conv9\_up = nn.ConvTranspose2d(128/2, 64/2, 2, 2)

self.conv9\_input = nn.Conv2d(128/2, 64/2, 3, padding=1)

self.conv9 = nn.Conv2d(64/2, 64/2, 3, padding=1)

self.conv9\_output = nn.Conv2d(64/2, 2, 1)

if dice:

self.final = F.softmax

else:

self.final = F.log\_softmax

def switch(self, dice):

if dice:

self.final = F.softmax

else:

self.final = F.log\_softmax

def forward(self, x):

layer1 = F.relu(self.conv1\_input(x))

layer1 = F.relu(self.conv1(layer1))

layer2 = F.max\_pool2d(layer1, 2)

layer2 = F.relu(self.conv2\_input(layer2))

layer2 = F.relu(self.conv2(layer2))

layer3 = F.max\_pool2d(layer2, 2)

layer3 = F.relu(self.conv3\_input(layer3))

layer3 = F.relu(self.conv3(layer3))

layer4 = F.max\_pool2d(layer3, 2)

layer4 = F.relu(self.conv4\_input(layer4))

layer4 = F.relu(self.conv4(layer4))

layer7 = F.relu(self.conv7\_up(layer4))

layer7 = torch.cat((layer3, layer7), 1)

layer7 = F.relu(self.conv7\_input(layer7))

layer7 = F.relu(self.conv7(layer7))

layer8 = F.relu(self.conv8\_up(layer7))

layer8 = torch.cat((layer2, layer8), 1)

layer8 = F.relu(self.conv8\_input(layer8))

layer8 = F.relu(self.conv8(layer8))

layer9 = F.relu(self.conv9\_up(layer8))

layer9 = torch.cat((layer1, layer9), 1)

layer9 = F.relu(self.conv9\_input(layer9))

layer9 = F.relu(self.conv9(layer9))

layer9 = self.final(self.conv9\_output(layer9))

return layer9

We write two networks for our model is U-Net and V-Net. Let’s begin from U-Net:



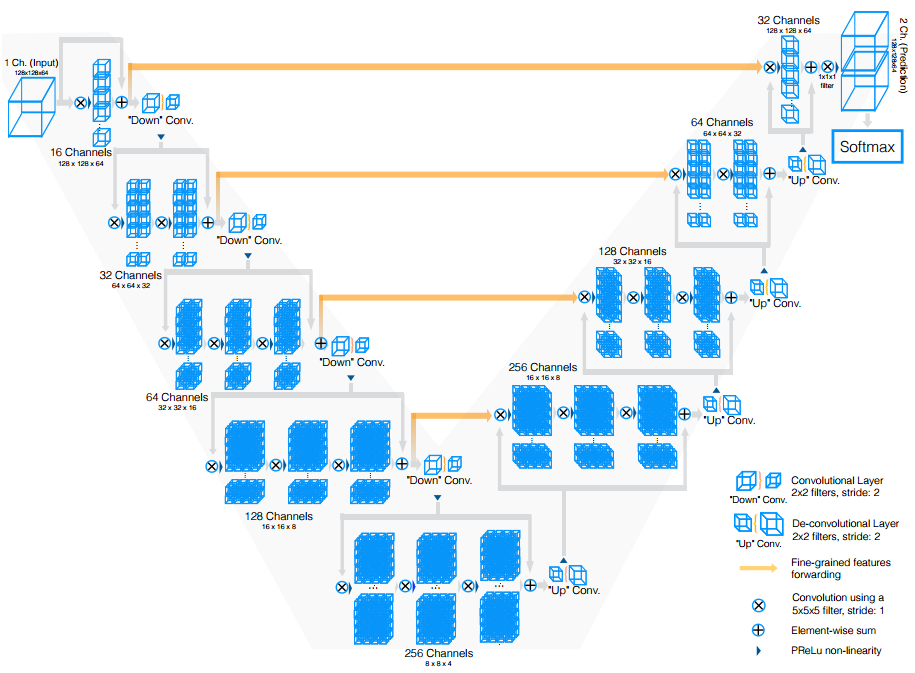
This is an implementation of the U-Net architecture in PyTorch, a deep learning framework. The U-Net is a convolutional neural network designed for semantic segmentation tasks.

The code defines a UNet class that extends the nn.Module class from PyTorch. It has the following components:

* conv1\_input to conv5: five sets of 2D convolutional layers with rectified linear unit (ReLU) activation function.
* conv6\_up to conv9\_output: nine sets of 2D transposed convolutional layers and 2D convolutional layers with ReLU activation function.
* final: the final activation function to be used, either the softmax or log softmax function, specified by the dice argument.

The switch method allows the final activation function to be changed during runtime.

The forward method defines the forward pass of the U-Net. It passes the input tensor x through multiple sets of convolutional and transposed convolutional layers, with ReLU activation functions, max pooling layers, and concatenation of features from the downsampling and upsampling paths. The final output is passed through the final activation function.



This code implements a 3D Convolutional Neural Network (VNet\_Xtra) using PyTorch's nn module. The network consists of multiple convolutional layers and batch normalization layers, followed by down-sampling and up-sampling layers to perform spatial resolution adaptation. Dropout layers are also added for regularization if dropout is set to True. The network takes an input of shape (1 + context \* 2, X, Y) where X and Y represent spatial dimensions, and context determines the number of additional channels in the input data. The output is a segmentation map of the same shape as the input.

Train code

model\_name = '25D'

augment = True

dropout = True

dice = True

context = 2

lr = 1e-2

batch\_size = 10

num\_samples = 1000

low\_lr\_epoch = 80

epochs = 100

train\_folder = 'data/train'

val\_folder = 'data/val'

print model\_name

print "augment="+str(augment)+" dropout="+str(dropout)

print str(epochs) + " epochs - lr: " + str(lr) + " - batch size: " + str(batch\_size)

cuda = torch.cuda.is\_available()

loss\_weight = torch.FloatTensor([0.01, 0.99])

if cuda: loss\_weight = loss\_weight.cuda()

criterion = nn.NLLLoss2d(weight=loss\_weight)

net = networks.VNet\_Xtra(dice=dice, dropout=dropout, context=context)

if cuda: net = torch.nn.DataParallel(net, device\_ids=list(range(torch.cuda.device\_count()))).cuda()

optimizer = optim.Adam(net.parameters(), lr=lr)

train = LiverDataSet(directory=train\_folder, augment=augment, context=context)

train\_sampler = torch.utils.data.sampler.WeightedRandomSampler(weights=train.getWeights(), num\_samples=num\_samples)

train\_data = torch.utils.data.DataLoader(train, batch\_size=batch\_size, shuffle=True, sampler=train\_sampler, num\_workers=2)

val = LiverDataSet(directory=val\_folder, context=context)

val\_data\_list = []

patients = val.getPatients()

for key in patients.keys():

samples = patients[key]

val\_sampler = torch.utils.data.sampler.SubsetRandomSampler(samples)

val\_data = torch.utils.data.DataLoader(val, batch\_size=batch\_size, shuffle=False, sampler=val\_sampler, num\_workers=2)

val\_data\_list.append(val\_data)

print 'Start training...'

for epoch in range(epochs):

running\_loss = 0.0

if epoch == low\_lr\_epoch:

for param\_group in optimizer.param\_groups:

lr = lr / 10

param\_group['lr'] = lr

net.train()

for i, data in enumerate(train\_data):

inputs, labels = data

if cuda: inputs, labels = inputs.cuda(), labels.cuda()

inputs, labels = Variable(inputs), Variable(labels)

outputs = net(inputs)

if dice:

outputs = outputs[:,1,:,:].unsqueeze(dim=1)

loss = dice\_loss(outputs, labels)

else:

labels = labels.squeeze(dim=1)

loss = criterion(outputs, labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

running\_loss += loss.data[0]

if dice:

print(' [epoch %d] - train dice loss: %.3f' % (epoch + 1, running\_loss/(i+1)))

else:

print(' [epoch %d] - train cross-entropy loss: %.3f' % (epoch + 1, running\_loss/(i+1)))

net.eval()

all\_dice = []

all\_accuracy = []

if (epoch+1)%10 != 0: continue

for val\_data in val\_data\_list:

accuracy = 0.0

intersect = 0.0

union = 0.0

for i, data in enumerate(val\_data):

inputs, labels = data

if cuda: inputs, labels = inputs.cuda(), labels.cuda()

inputs, labels = Variable(inputs, volatile=True), Variable(labels, volatile=True)

outputs = net(inputs)

if not dice: outputs = outputs.exp()

outputs = outputs[:, 1, :, :].unsqueeze(dim=1).round()

outputs, labels = outputs.data.cpu().numpy(), labels.data.cpu().numpy()

accuracy += (outputs == labels).sum() / float(outputs.size)

intersect += (outputs+labels==2).sum()

union += np.sum(outputs) + np.sum(labels)

all\_accuracy.append(accuracy / float(i+1))

all\_dice.append(1 - (2 \* intersect + 1e-5) / (union + 1e-5))

print(' val dice loss: %.9f - val accuracy: %.8f' % (np.mean(all\_dice), np.mean(all\_accuracy)))

This script trains a 3D medical image segmentation model. The specific architecture used is the "VNet\_Xtra" network with the option to use either Dice loss or cross-entropy loss. The script is written in Python and uses the PyTorch deep learning library.

The script starts by importing necessary libraries and defining the hyperparameters for the training process:

* model\_name: name of the model
* augment: Boolean indicating whether data augmentation should be performed or not
* dropout: Boolean indicating whether dropout should be used in the model
* dice: Boolean indicating whether Dice loss should be used as the loss function or cross-entropy loss
* context: integer indicating how many slices of context (2.5D) should be used
* lr: learning rate
* batch\_size: size of each training batch
* num\_samples: number of samples per epoch
* low\_lr\_epoch: epoch where the learning rate should be lowered
* epochs: total number of training epochs

The script then sets the path to the training and validation datasets and initializes the GPU.

Next, the script sets up the loss function. If cross-entropy loss is used, a weighting of positive vs negative pixels and a NLL loss layer is defined.

The script then initializes the VNet\_Xtra network and an Adam optimizer.

Data loaders for the training and validation datasets are then created. The training data loader uses a WeightedRandomSampler to sample data, while the validation data loader samples data per patient.

Finally, the script enters a training loop for the specified number of epochs. In each epoch, the learning rate is reduced if the current epoch is the low\_lr\_epoch. The network is then set to train mode and the training data is passed through the network to calculate the loss. The gradients are computed and the weights are updated with the Adam optimizer. The training loss is printed at the end of each epoch.

Every 10 epochs, the network is set to evaluation mode and the validation data is passed through the network. The Dice coefficient and accuracy are calculated and saved, but not printed in the code provided.

The training process continues until all specified epochs are completed.

Inference

In machine learning, inference refers to the process of using a trained model to make predictions on new, unseen data. This is often the final step in the machine learning process, where the model is used to generate predictions for real-world data. The goal of inference is to estimate the unknown parameters of the model based on the input data, and to use these estimates to make predictions. Inference requires that the model has been trained on a large dataset of labeled examples, and that the parameters of the model have been optimized to minimize the error between the model's predictions and the actual outcomes.

cuda = torch.cuda.is\_available()

net = torch.load("model\_"+model\_name+".pht")

if cuda: net = torch.nn.DataParallel(net, device\_ids=list(range(torch.cuda.device\_count()))).cuda()

net.eval() # inference mode

for file\_name in files:

data = nib.load(os.path.join(test\_folder, file\_name))

input\_aff = data.affine

data = data.get\_data()

data = np.clip(data, -200, 200) / 400.0 + 0.5

data = np.transpose(data, (2, 0, 1))

output = np.zeros((len(data), 512, 512))

for i in range(len(data)):

slices\_input = []

z = i - context

slices\_input.append(np.expand\_dims(data[i], 0))

while z <= i + context:

if z == i:

# middle slice is already appended

pass

elif z < 0:

# append first slice if z falls outside of data bounds

slices\_input.append(np.expand\_dims(data[0], 0))

elif z >= len(data):

# append last slice if z falls outside of data bounds

slices\_input.append(np.expand\_dims(data[len(data)-1], 0))

else:

# append slice z

slices\_input.append(np.expand\_dims(data[z], 0))

z += 1

inputs = np.expand\_dims(np.concatenate(slices\_input, 0), 0)

inputs = Variable(torch.from\_numpy(inputs).float(), volatile=True)

if cuda: inputs = inputs.cuda()

outputs = net(inputs)

outputs = outputs[0, 1, :, :].round()

outputs = outputs.data.cpu().numpy()

output[i, :, :] = outputs \* 2

output = np.transpose(output, (1, 2, 0)).astype(np.uint8)

output = nib.Nifti1Image(output, affine=input\_aff)

new\_file\_name = "test-segmentation-" + file\_name.split("-")[-1]

print new\_file\_name

This is a script for applying a trained neural network to a set of medical image data and saving the output as Nifti files.

The code uses the nibabel library for handling Nifti file formats, torch for loading and running the neural network, and numpy for processing the data.

The script starts by defining some variables:

* model\_name: the name of the saved model that will be loaded
* context: the number of slices before and after the current slice to use as input to the network
* result\_folder: the directory where the output Nifti files will be saved
* test\_folder: the directory where the input test data is stored

The script then creates the result\_folder if it doesn't exist yet. The files variable is then set to a list of filenames in test\_folder that start with "test".

The network is loaded using torch.load("model\_"+model\_name+".pht"). If CUDA is available, the network is moved to GPU using torch.nn.DataParallel. The network is then set to evaluation mode with net.eval().

For each file in the files list, the script loads the data using nib.load and converts it to a numpy array. The data is then normalized and transposed so that the slices are the first dimension. An empty numpy array output is created to store the network's predictions.

The script then loops through the slices, appending a list of slices (the current slice plus the context number of slices before and after) to slices\_input. This list of slices is then concatenated and passed as input to the network. The network's output is rounded and saved to the output array.

Finally, the output array is transposed back to its original orientation and saved as a Nifti file with nib.save. The new filename is constructed by replacing the original filename's prefix with "test-segmentation-".

## Conclusion

At the end of the practical part, it remains to determine the number of epochs for our neural network in order to best train it. At the moment the results are not quite good, but we need to improve it in the future. The practical part was quite an interesting iteration with the team, the development went smoothly, everyone contributed a lot.

## References

Article independent clinical validation of the automated ki67 scoring guideline from the international ki67 in breast cancer working group: <https://www.mdpi.com/2218-273X/11/11/1612/pdf>

Multi-Parametric MRI-Based Radiomics Models for Predicting Molecular Subtype and Androgen Receptor Expression in Breast Cancer: <http://www.frontiersin.org/Oncology/about>

Evaluating the Accuracy of Breast Cancer and Molecular Subtype Diagnosis by Ultrasound Image Deep Learning Model: <https://www.frontiersin.org/articles/10.3389/fonc.2021.623506/full>

A Computational Tumor-Infiltrating Lymphocyte Assessment Method Comparable with Visual Reporting Guidelines for Triple-Negative Breast Cancer: <https://www.thelancet.com/action/showPdf?pii=S2352-3964%2821%2900285-1>

Space curvature-inspired nanoplasmonic sensor for breast cancer extracellular vesicle fingerprinting and machine learning classification: <https://opg.optica.org/directpdfaccess/5941d077-59f1-41de-84b3f6dd6fa79f45_451756/boe-12-7-3965.pdf?da=1&id=451756&seq=0&mobile=no>

Deep Learning-Based Prediction Model for Breast Cancer Recurrence Using Adjuvant Breast Cancer Cohort in Tertiary Cancer Center Registry: <https://www.frontiersin.org/articles/10.3389/fonc.2021.596364/full>

Tumour Stroma Ratio Assessment Using Digital Image Analysis Predicts Survival in Triple Negative and Luminal Breast Cancer: <https://www.mdpi.com/2072-6694/12/12/3749/pdf>

Multi-Class Classification of Breast Cancer Using 6B-Net with Deep Feature Fusion and Selection Method: <https://www.mdpi.com/2075-4426/12/5/683/pdf>

A scalable crowdsourcing approach and dataset for nucleus classification and segmentation in breast cancer: <https://academic.oup.com/gigascience/article/doi/10.1093/gigascience/giac037/6586817?searchresult=1>

A Novel Multistage Transfer Learning for Ultrasound Breast Cancer Image Classification: <https://www.mdpi.com/2075-4418/12/1/135/pdf>

Construction and Validation of a Prognostic Risk Model for Triple-Negative Breast Cancer Based on Autophagy-Related Genes: <https://www.frontiersin.org/articles/10.3389/fonc.2022.829045/full>

Immune Effective Score as a Predictor of Response to Neoadjuvant Trastuzumab Therapy and a Prognostic Indicator for HER2-Positive Breast Cancer: <https://www.mdpi.com/1718-7729/29/1/26/pdf>

Lymph Node Metastasis Status in Breast Carcinoma Can Be Predicted via Image Analysis of Tumor Histology : <https://aqch.com/toc/auto_article_process.php?year=2015&page=273&id=23355&sn=0>

Digital Microscopy Assessment of Angiogenesis in Different Breast Cancer Compartments: <https://www.hindawi.com/journals/bmri/2013/286902/>

Reverse phase protein array based tumor profiling identifies a biomarker signature for risk classification of hormone receptor-positive breast cancer: <https://www.sciencedirect.com/science/article/pii/S2212963414000023?via%3Dihub>

Reverse-Phase Protein Array for Prediction of Patients at Low Risk of Developing Bone Metastasis From Breast Cancer: <https://academic.oup.com/oncolo/article/19/9/909/6399551>