



한양대학교

Sub-typing Cancer Cells Using Machine Learning

Saira Tahsin

Jimenez Saucedo Paz Carolina

Isai Carreto Martinez

Hanyang University

Dept. of Computer Software Engineering

2020.11.30



<붙임 1>

SW프로젝트 요약서

프로젝트 기간	2020.01.01 – 2020.11.27
프로젝트 팀원	지메네스 소스도 파스 카롤리나(컴퓨터소프트웨어학부 4), 사이라 타신(컴퓨터소프트웨어학부 4) 이사이 까레또 마르띠네스 (컴퓨터소프트웨어학부 4),
지도교수	백은옥
프로젝트 멘토	백제현
프로젝트 명	생체 데이터 클러스터링을 통한 암의 서브타이핑



프로젝트 내용

During the cancer subtyping with biometric data classification, we deduced the intensity of tumors. Human architecture is independently constructed by different patterns of genes and proteins. Hence, determined patients with the same cancer, also have divergences in their cancer structure or subtype. Our goal is to select the right peptides on our data, create visual samples to finally classify them as different types of cancers. Since we had limited data to work with, we used one-shot learning algorithm which tries to mimic human intelligence in that we are able to instinctively weigh the similarities from just one sample of a given object and use it to differentiate new samples. It is a relatively new field in supervised learning.

In this project, we used a Siamese Neural Network (a type of one-shot learning) that exerts a distinctive structure to rank resemblance between inputs. Applying a convolutional architecture, we managed to achieve reasonable results. The model is capable of learning generic image features practical for predictions about class distributions with hardly any available sample from these new distributions. And it is trained using standard optimization methods on pairs sampled from the source data. The validation model learns to recognize input pairs related to the possibility that they fit in to a similar or different class. The pairing with the highest score according to validation network is assigned the highest probability for the one-shot task. If the features stored by the model are adequate



to approve or refuse the likeness of images from one set of types, then they must be ample for other types, as the model has already been exposed to different type of tissues to stimulate deviation amongst the identified features.

We divided our dataset into 2 main categories. They are images\_background (used to train) and images\_evaluation (used to evaluate).

Once the learning curve flattened out, we used the weights which got the best validation 20-way accuracy for testing. Our network averaged ~50% accuracy for tasks from the evaluation set, compared to 93% in the original paper. The original paper works with grayscale alphabets from 50 different languages. Probably the difference is because we didn't implement many of the performance enhancing tricks from the original paper, like layer wise learning rates/momentum, data augmentation with distortions, Bayesian hyperparameter optimization and we also probably trained for less epochs. We are not too worried about this because this project was more about implementing one-shot learning in general, than squeezing the last few % performance out of a classifier.

We were curious to see how accuracy varied over different values of "N" in N way one shot learning, so we plotted it, with comparisons to k nearest neighbors, random guessing and training set performance.



기대효과

및

개선방향

This project can be seen as a demonstration on how one-shot learning can be used to significantly lower the amounts of data required to make meaningful predictions in fields like medical imaging, drug discovery applications. In biomedical domains, such as digital pathology, the amount of annotated samples is very reduced and imbalanced. On the other hand, models trained using supervised methods need to be retrained when a new class (with its samples) is introduced. This process is computationally expensive and can be difficult when one does not have a lot of examples for the same class. One shot learning is one of solution to this issue.

This is a practical pilot for multiple cancer's clinical treatments. Progressive profile technologies for tissue have gathered profuse sets of data. Set up on these sort of expression data, numerous algorithmic methods have been brought forward to predict cancer subtypes. It is essential to study how to improve the integration of sundry profiles of data for a more solid clinical treatment of cancer.



<붙임 2>

SW프로젝트 결과보고서

프로젝트명	생체 데이터 클러스터링을 통한 암의 서브타이핑
프로젝트 요약	<p>Some patients with the same cancer, also have dissimilarities in their cancer structure. These differences are interpreted as a subtype of cancer and can be catalogued by the composition of genomes and proteins. Mainly, the formation of proteins is deduced through a pattern of protein fragments known as peptides. In this project, we will make a visual representation of these patterns as a 3D map. Consequently, at the hand of machine learning algorithms implementation, subtypes of cancer will be identified and classified.</p> <p>Our project was done through a one-shot classification process. Thus, we only used one training sample for each class. The network learns a similarity function, which takes two images as an input and expresses how similar they are. For every pair of input image, the model generates a similarity between 0 and 1. Basically, the same patient is compared to different patients out of which only one of them matches the original one.</p>



	<p>We got an accuracy of 50% for the model. To see how accuracy varied over different values of “N” in N way one shot learning, so we plotted it, with comparisons to k nearest neighbors, random guessing and training set performance and Siamese clearly performed the best.</p>
<p>프로젝트 기간</p>	<p>2020.01.01 – 2020.11.27</p>
<p>산출물</p>	<p>졸업 작품 ( O ),      졸업 논문 ( )  (해당 내용은 동그라미 표시)</p>
<p>성과물</p>	<p>특허 (      ),    논문(      ),    SW(   O   )  (해당 내용은 동그라미 표시)</p>



학과	학번	학년	이름	연락처
컴퓨터소프트웨어	2017053772	4	지메네스 소스도 파스 카롤리나	<a href="mailto:Pazjimenez96@gmail.com">Pazjimenez96@gmail.com</a> 010-6635-9016
컴퓨터소프트웨어	2016056617	4	이사이 까레토 마르띠네스	<a href="mailto:isaiicatmat@gmail.com">isaiicatmat@gmail.com</a> 010-6566-9707
컴퓨터소프트웨어	2017053736	4	사이라 타신	<a href="mailto:preotasherpa@gmail.com">preotasherpa@gmail.com</a> 010-9730-7084





## 목 차

### 1. 프로젝트 개요

#### 1.1 프로젝트 목적 및 배경

#### 1.2 프로젝트 최종 목표

### 2. 프로젝트 내용

### 3. 프로젝트의 기술적 내용

### 4. 프로젝트의 역할 분담

#### 4.1 개별 임무 분담

#### 4.2 개발 일정

### 5. 결론 및 기대효과



## 1. 프로젝트 개요

### 1.1 산학프로젝트 목적 및 동기

Because the composition of the genome and protein is different from person to person, the cause and the appearance may be different even for the same cancer. This can be regarded as a "subtype" of cancer, which can be classified through the composition of genomes and proteins.

In general, the composition of proteins is inferred through the pattern of peptides, which are fragments of proteins. In this project, we will image the pattern of peptides as a 3D map, and by implementing machine learning algorithms, subtypes of cancer will be identified and classified.

In order to accomplish the project definition, we are going to apply all the knowledge acquired these past 4 years in computer software engineering department, specifically, we will focus on machine learning: an application of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

Machine learning is taking a bigger part in our health and well-being on a daily basis, and it is already being used for faster patient diagnosis. Even the prevention of illness in the first place has been aided by predicting the potential health problems one may be susceptible to, based on age, socio-economic status, genetic history, etc. The use of programs to analyze and cross-reference symptoms against databases containing millions of other cases and illnesses has led to faster diagnoses of illness and disease, saving lives through quicker treatment and decreasing the time a patient spends in the health system. Hospitals are currently using AI algorithms to detect tumors more accurately in radiology scans and analyze different moles for skin cancer, and machine learning is being adapted to accelerate research toward a cure for cancer.

In other words, the necessity of this project relies on the discovery of cancer subtypes which is useful for guiding clinical treatment of multiple cancers. Progressive profile technologies



for tissue have accumulated diverse types of data. Based on these types of expression data, various computational methods have been proposed to predict cancer subtypes. It is crucial to study how to better integrate these multiple profiles of data for better clinical treatment of cancer.

## 1.2 산학프로젝트 목표

Several patients can possess the same type of cancer. However, each person has different patterns of genes/proteins, which implies that the same cancer can have different patterns as well, and this is called a cancer subtype. In other words, the aim of this project is to find cancer subtypes from each patient through the analysis of peptides (small fragments of protein). To achieve this, we will image the pattern of peptides as a 3D map. The images produced will be used as a dataset to feed our deep learning model named Siamese neural networks for one-shot learning. This model is going to use a combination of convolutional neural networks for image classification and cluster techniques to detect and classify all the subtypes of cancer.



## 2. 산학프로젝트 내용

The prerequisites needed to achieve the specific goals of this project involve having knowledge of concepts like statistics, linear algebra, calculus, probability, bioinformatics and programming languages. It is also required to have a strong background in machine learning, deep learning, and convolutional neural networks. Statistics contain tools that can be used to get some outcome from the data. There is descriptive statistics which is used to transform raw data into some important information. Also, inferential statistics can be used to get important information from a sample of data instead of using a complete dataset. Linear algebra deals with vectors, matrices, and linear transformations. It is very important in machine learning as it can be used to transform and perform operations on the dataset. Also, Calculus is an important field in mathematics, and it plays an integral role in many machine learning algorithms. Data sets having multiple features are used to build machine learning models where integrations and differentiations are a must. Probability helps predict the likelihood of the occurrences, it helps us to reason the situation may or may not happen again. For machine learning, probability is a foundation.

Furthermore, to achieve the main goal of this project, having a background on deep learning is also a requirement. This is a computer science branch that studies the design of algorithms that can learn. Deep learning is a subfield of machine learning that is inspired by artificial neural networks, which in turn are inspired by biological neural networks.

Also, it is essential to know programming languages like Python, which is the programming language that will be used for this project, in order to implement the whole Machine Learning process. Moreover, Python provide in-built libraries that make it very easy to implement Machine Learning algorithms.

Another necessary field is Bioinformatics. Bioinformatics is a subdiscipline of biology and



computer science concerned with the acquisition, storage, analysis, and dissemination of biological data, most often DNA and amino acid sequences. Bioinformatics uses computer programs for a variety of applications, including determining gene and protein functions, establishing evolutionary relationships, and predicting the three-dimensional shapes of proteins. We had the help of our TA (Teacher's assistant) and mentor Mr. Baek Je Hyun, Director of the R&D Center for Clinical Mass Spectrometry, Seegene Medical Foundation, to guide us with this matter.

To conclude, been familiar with the basics of machine learning, bioinformatics, deep learning and having some experience on using Convolutional Neural Networks with Python and Keras are the main requirements for the completion of this project.

Deep Convolutional Neural Networks have become the state-of-the-art methods for image classification tasks. However, one of the biggest limitations is that they require lots of labelled data. In many applications, collecting this many data are sometimes not feasible. In addition, the process of learning good features for machine learning applications can be very computationally expensive and may prove difficult in cases where little data is available. Since our project is based on a relatively small dataset, the model needs to do the learning just with a couple of labeled images instead of huge number of images. One Shot Learning aims to solve this problem.

### **Convolutional Neural networks (CNN)**

CNN is a type of deep learning model for processing data that has a grid pattern, such as images, which is inspired by the organization of animal visual cortex [13, 14] and designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns. CNN is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers. The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized type of

linear operation. In digital images, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers (Fig. 2), and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex. The process of optimizing parameters such as kernels is called training, which is performed so as to minimize the difference between outputs and ground truth labels through an optimization algorithm called backpropagation and gradient descent, among others.

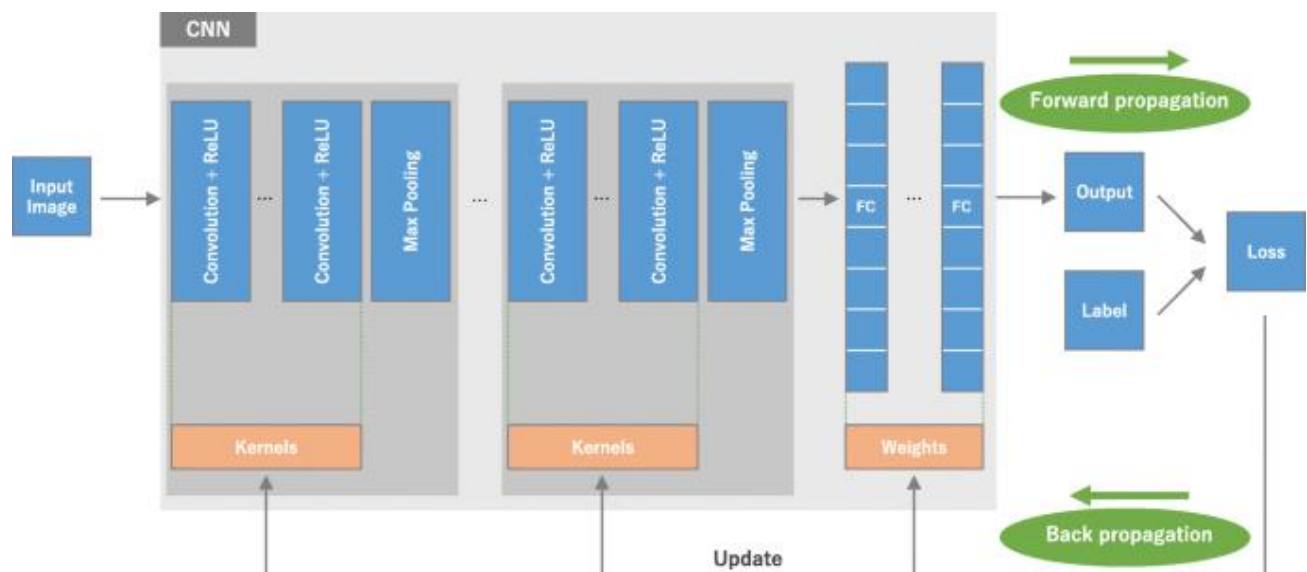


Figure 1.

In Fig.1 we have an overview of a convolutional neural network (CNN) architecture and the training process. A CNN is composed of a stacking of several building blocks: convolution layers, pooling layers (e.g., max pooling), and fully connected (FC) layers. A model's performance under particular kernels and weights is calculated with a loss function through forward propagation on a training dataset, and learnable parameters, i.e., kernels and weights, are updated according to the loss value through backpropagation with gradient descent optimization algorithm. ReLU, rectified linear unit.

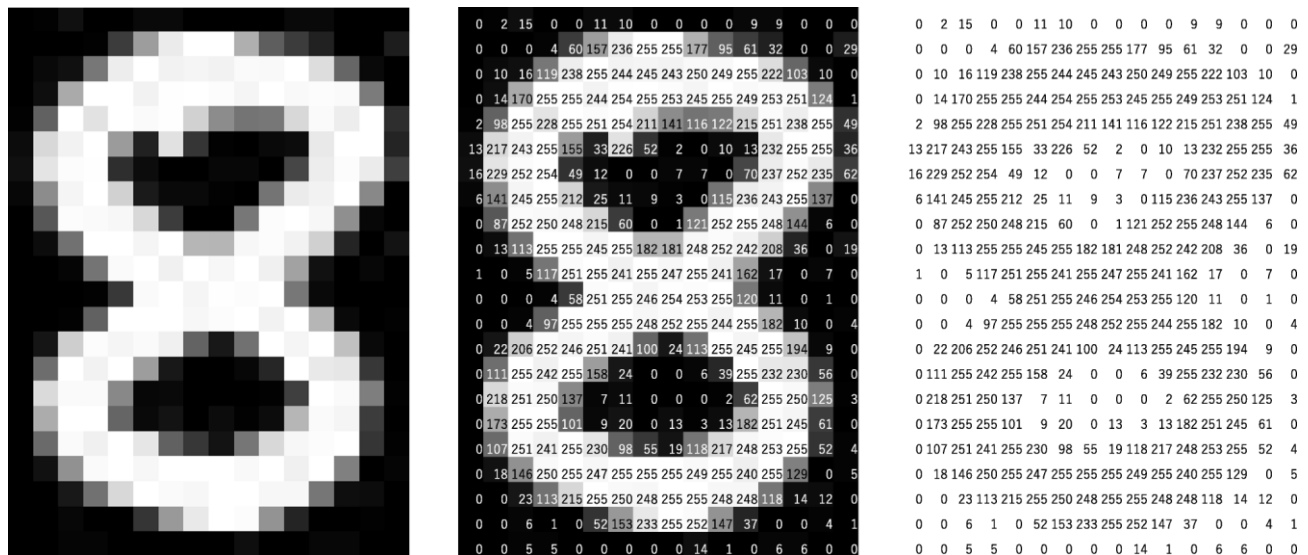


Figure 2.

A computer sees an image as an array of numbers. The matrix on the right contains numbers between 0 and 255, each of which corresponds to the pixel brightness in the left image. Both are overlaid in the middle image.

## Building blocks of CNN architecture

The CNN architecture includes several building blocks, such as convolution layers, pooling layers, and fully connected layers. A typical architecture consists of repetitions of a stack of several convolution layers and a pooling layer, followed by one or more fully connected layers. The step where input data are transformed into output through these layers is called forward propagation (Fig. 1). Although convolution and pooling operations described in this section are for 2D-CNN, similar operations can also be performed for three-dimensional (3D)-CNN.

- **Convolution layer**

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e.,

convolution operation and activation function.

- **Convolution**

Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a tensor. An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map (Fig. 3 a–c). This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors; different kernels can, thus, be considered as different feature extractors (Fig. 4). Two key hyperparameters that define the convolution operation are size and number of kernels. The former is typically  $3 \times 3$ , but sometimes  $5 \times 5$  or  $7 \times 7$ . The latter is arbitrary and determines the depth of output feature maps.

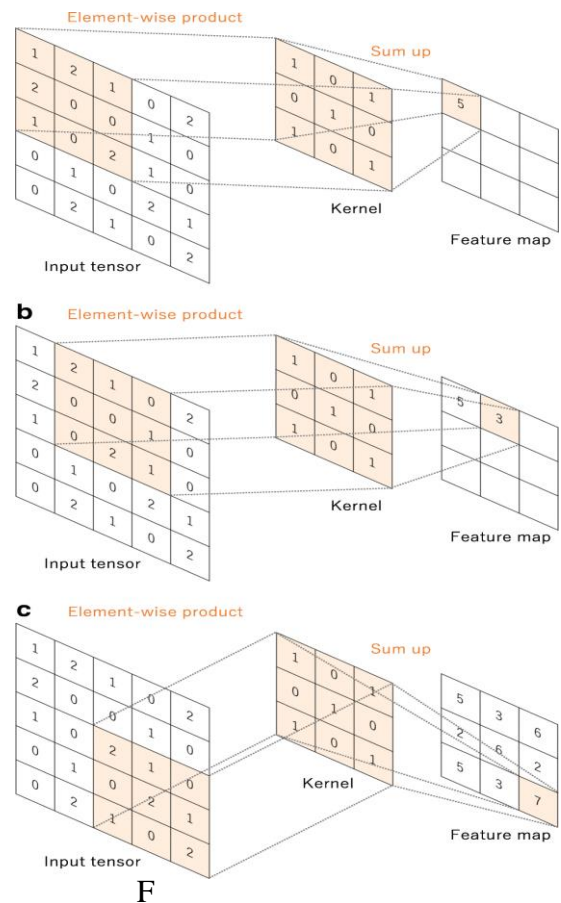


Figure 3.



In Fig. 3 a–c shows an example of convolution operation with a kernel size of  $3 \times 3$ , no padding, and a stride of 1. A kernel is applied across the input tensor, and an element-wise product between each element of the kernel and the input tensor is calculated at each location and summed

to obtain the output value in the corresponding position of the output tensor, called a feature map. Fig. 4 shows examples of how kernels in convolution layers extract features from an input tensor. Multiple kernels work as different feature extractors, such as a horizontal edge detector (top), a vertical edge detector (middle), and an outline detector (bottom). Note that the left image is an input, those in the middle are kernels, and those in the right are output feature maps

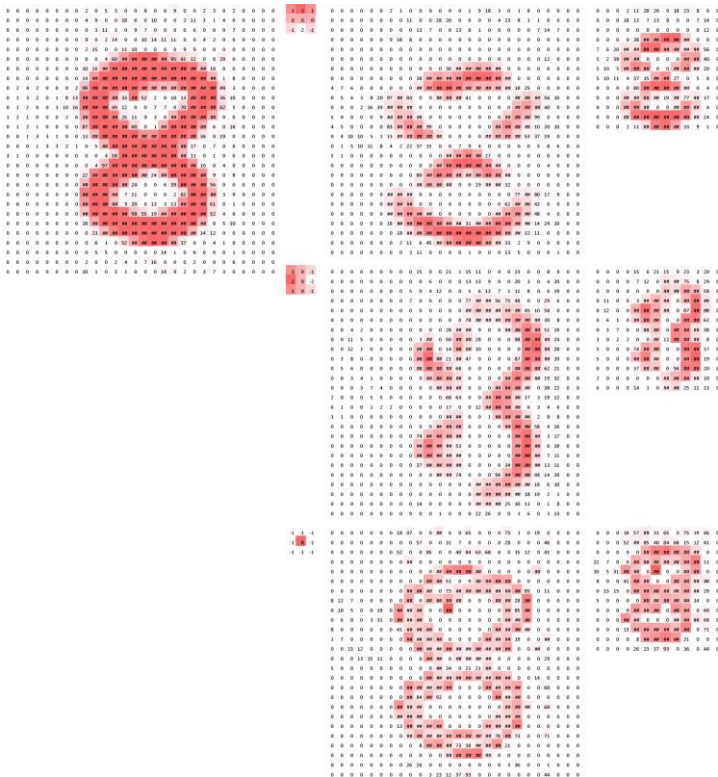


Figure 4

The convolution operation described above does not allow the center of each kernel to overlap the outermost element of the input tensor and reduces the height and width of the output feature map compared to the input tensor. Padding, typically zero padding, is a technique to address this issue, where rows and columns of zeros are added on each side of the input tensor, so as to fit the center of a kernel on the outermost element and keep the same in-plane dimension through the convolution operation (Fig. 4). Modern CNN architectures usually employ zero padding to retain in-plane dimensions in order to apply more layers. Without zero padding, each successive feature map would get smaller after the convolution operation.

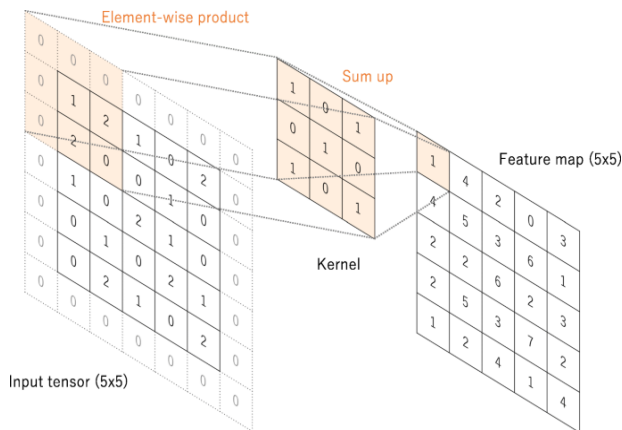


Figure 5.

A convolution operation with zero padding so as to retain in-plane dimensions. Note that an input dimension of  $5 \times 5$  is kept in the output feature map. In this example, a kernel size and a stride are set as  $3 \times 3$  and 1, respectively

The distance between two successive kernel positions is called a stride, which also defines the convolution operation. The common choice of a stride is 1; however, a stride larger than 1 is sometimes used in order to achieve downsampling of the feature maps. An alternative technique to perform downsampling is a pooling operation, as described below.

The key feature of a convolution operation is weight sharing: kernels are shared across all the image positions. Weight sharing creates the following characteristics of convolution operations: (1) letting the local feature patterns extracted by kernels translation be invariant as kernels travel across all the image positions and detect learned local patterns, (2) learning spatial hierarchies of feature patterns by downsampling in conjunction with a pooling operation, resulting in capturing an increasingly larger field of view, and (3) increasing model efficiency by reducing the number of parameters to learn in comparison with fully connected neural networks.

- **Nonlinear activation function**

The outputs of a linear operation such as convolution are then passed through a nonlinear activation function. Although smooth nonlinear functions, such as sigmoid or hyperbolic tangent (tanh) function, were used previously because they are mathematical representations of a biological neuron behavior, the most common nonlinear activation function used presently is the rectified linear unit (ReLU), which simply computes the function:  $f(x) = \max(0, x)$  (Fig. 6)

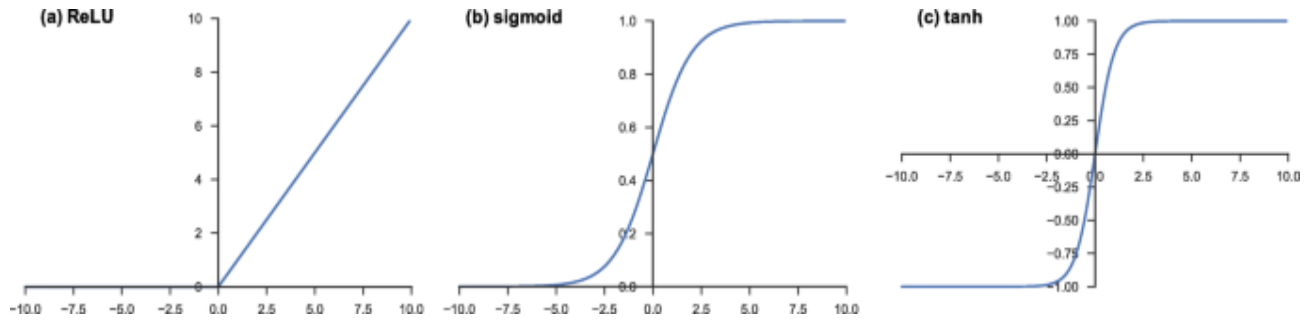


Figure 6.

- **Pooling layer**

A pooling layer provides a typical downsampling operation which reduces the in-plane dimensionality of the feature maps in order to introduce a translation invariance to small shifts and distortions, and decrease the number of subsequent learnable parameters. It is of note that there is no learnable parameter in any of the pooling layers, whereas filter size, stride, and padding are hyperparameters in pooling operations, similar to convolution operations.

- **Max pooling**

The most popular form of pooling operation is max pooling, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other values (Fig. 6). A max pooling with a filter of size  $2 \times 2$  with a stride of 2 is commonly used in practice. This downsamples the in-plane dimension of feature maps by a factor of 2. Unlike height and width, the depth dimension of feature maps remains unchanged.



In fig. 7, image a shows an example of max pooling operation with a filter size of  $2 \times 2$ , no padding, and a stride of 2, which extracts  $2 \times 2$  patches from the input tensors, outputs the maximum value in each patch, and discards all the other values, resulting in downsampling the in-plane dimension of an input tensor by a factor of 2. b Examples of the max pooling operation on the same images in Fig. 3b. Note that images in the upper row are downsampled by a factor of 2, from  $26 \times 26$  to  $13 \times 13$ .

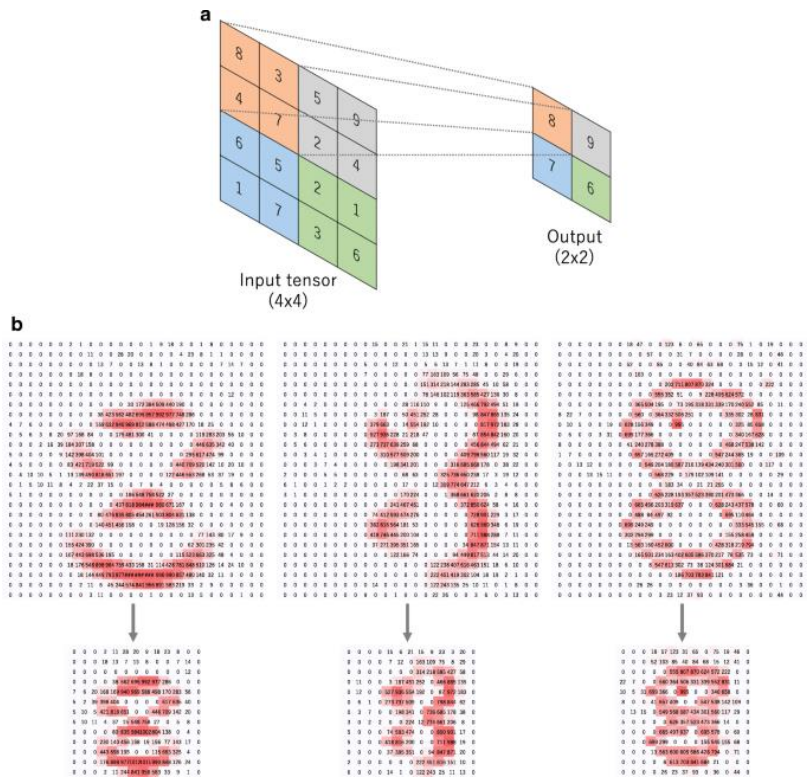


Figure 7.



- **Fully connected layer**

The output feature maps of the final convolution or pooling layer is typically flattened, i.e., transformed into a one-dimensional (1D) array of numbers (or vector), and connected to one or more fully connected layers, also known as dense layers, in which every input is connected to every output by a learnable weight. Once the features extracted by the convolution layers and downsampled by the pooling layers are created, they are mapped by a subset of fully connected layers to the final outputs of the network, such as the probabilities for each class in classification tasks. The final fully connected layer typically has the same number of output nodes as the number of classes. Each fully connected layer is followed by a nonlinear function, such as ReLU, as described above.

- **Last layer activation function**

The activation function applied to the last fully connected layer is usually different from the others. An appropriate activation function needs to be selected according to each task. An activation function applied to the multiclass classification task is a softmax function which normalizes output real values from the last fully connected layer to target class probabilities, where each value ranges between 0 and 1 and all values sum to 1.

- **Training a network**

Training a network is a process of finding kernels in convolution layers and weights in fully connected layers which minimize differences between output predictions and given ground truth labels on a training dataset. Backpropagation algorithm is the method commonly used for training neural networks where loss function and gradient descent optimization algorithm play essential roles. A model performance under particular kernels and weights is calculated by a loss function through forward propagation on a training dataset, and learnable parameters, namely kernels and weights, are updated according to the loss value through an optimization algorithm called



backpropagation and gradient descent, among others (Fig. 1).

- **Loss function**

A loss function, also referred to as a cost function, measures the compatibility between output predictions of the network through forward propagation and given ground truth labels. Commonly used loss function for multiclass classification is cross entropy, whereas mean squared error is typically applied to regression to continuous values. A type of loss function is one of the hyperparameters and needs to be determined according to the given tasks.

- **Gradient descent**

Gradient descent is commonly used as an optimization algorithm that iteratively updates the learnable parameters, i.e., kernels and weights, of the network so as to minimize the loss. The gradient of the loss function provides us the direction in which the function has the steepest rate of increase, and each learnable parameter is updated in the negative direction of the gradient with an arbitrary step size determined based on a hyperparameter called learning rate (Fig. 8). The gradient is, mathematically, a partial derivative of the loss with respect to each learnable parameter, and a single update of a parameter is formulated as follows:

$$w := w - \alpha * \frac{\partial L}{\partial w}$$

where  $w$  stands for each learnable parameter,  $\alpha$  stands for a learning rate, and  $L$  stands for a loss function. It is of note that, in practice, a learning rate is one of the most important hyperparameters to be set before the training starts. In practice, for reasons such as memory limitations, the gradients of the loss function with regard to the parameters are computed by using a subset of the training dataset called mini-batch, and applied to the parameter updates. This method is called mini-batch gradient descent, also frequently referred to as stochastic gradient descent (SGD), and a mini-batch size is also a hyperparameter. In addition, many improvements on the



gradient descent algorithm have been proposed and widely used, such as SGD with momentum, RMSprop, and Adam.

Gradient descent is an optimization algorithm that iteratively updates the learnable parameters so as to minimize the loss, which measures the distance between an output prediction and a ground truth label. The gradient of the loss function provides the direction in which the function has the steepest rate of increase, and all parameters are updated in the negative direction of the gradient with a step size determined based on a learning rate.

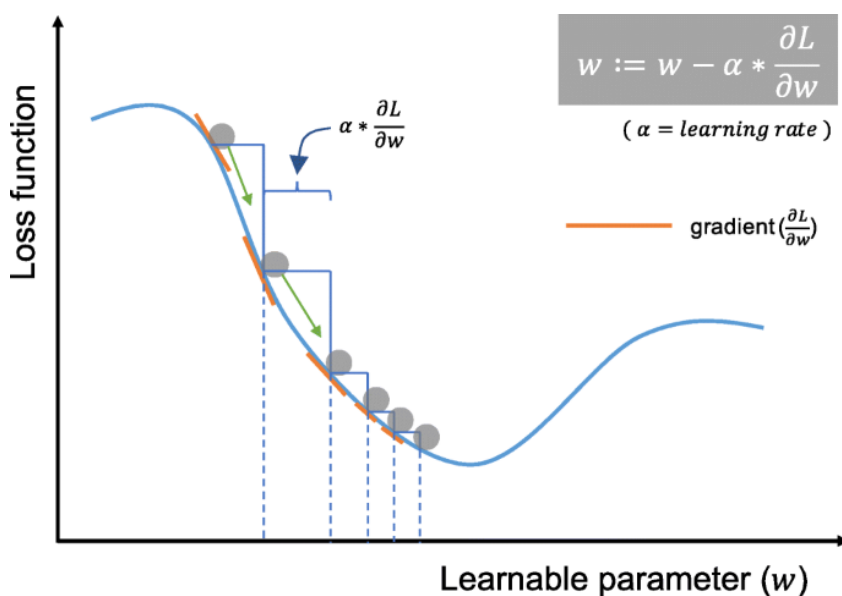


Figure 8.



## Siamese neural networks for one shot learning

Currently most deep learning models need generally thousands of labeled samples per class. Data acquisition for most tasks is very expensive. The possibility to have models that could learn from one or a few samples is a lot more interesting than having the need of acquiring and labeling thousands of samples. One could argue that a young child can learn a lot of concepts without needing a large number of examples. This is where one-shot learning appears: the task of classifying with only having access of one example of each possible class in each test task. This ability of learning from little data is very interesting and could be used in many machine learning problems.

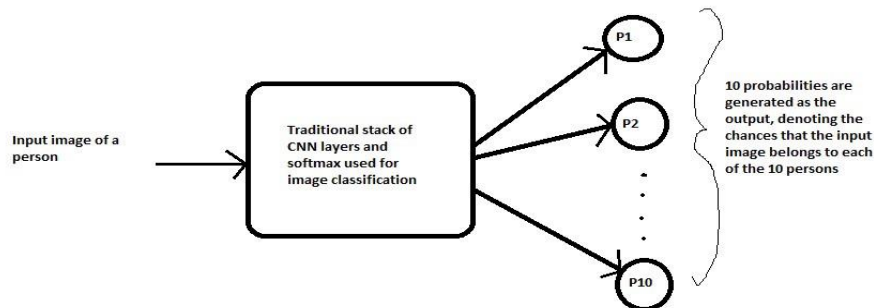
- **Classification vs. One shot learning.**

In case of standard classification, the input image is fed into a series of layers, and finally at the output we generate a probability distribution over all the classes (typically using a Softmax). For example, if we are trying to classify an image as cat or dog or horse or elephant, then for every input image, we generate 4 probabilities, indicating the probability of the image belonging to each of the 4 classes. Two important points must be noticed here. First, during the training process, we require a large number of images for each of the class (cats, dogs, horses and elephants). Second, if the network is trained only on the above 4 classes of images, then we cannot expect to test it on any other class, example “zebra”. If we want our model to classify the images of zebra as well, then we need to first get a lot of zebra images and then we must re-train the model again. There are applications wherein we neither have enough data for each class and the total number classes is huge as well as dynamically changing. Thus, the cost of data collection and periodical re-training is too high.

On the other hand, in a one shot classification, we require only one training example for each class. Hence the name One Shot.



Assume that we want to build face recognition system for a small organization with only 10 employees (small numbers keep things simple). Using a traditional classification approach, we might come up with a system that looks as below:



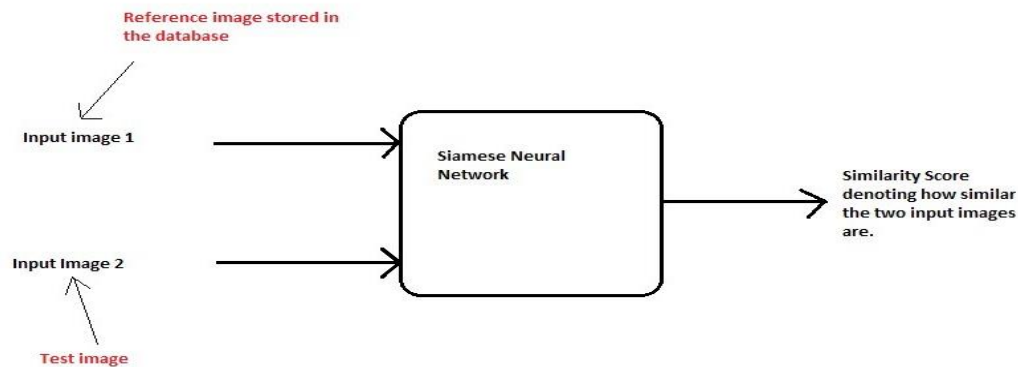
*Figure 9. Standard classification using CNN*

### **Problems:**

a) To train such a system, we first require a lot of different images of each of the 10 persons in the organization which might not be feasible. (Imagine if you are doing this for an organization with thousands of employees).

b) What if a new person joins or leaves the organization? You need to take the pain of collecting data again and re-train the entire model again. This is practically not possible especially for large organizations where recruitment and attrition is happening almost every week.

Now let's understand how we approach this problem using one shot classification which helps to solve both of the above issues:



*Figure 10. One Shot Classification*

Instead of directly classifying an input (test) image to one of the 10 people in the organization, this network instead takes an extra reference image of the person as input and will produce a similarity score denoting the chances that the two input images belong to the same person. Typically the similarity score is squished between 0 and 1 using a sigmoid function; wherein 0 denotes no similarity and 1 denotes full similarity. Any number between 0 and 1 is interpreted accordingly.

Notice that this network is not learning to classify an image directly to any of the output classes. Rather, it is learning a similarity function, which takes two images as input and expresses how similar they are.

- **Siamese neural network architecture**

The term Siamese means twins. The two Convolutional Neural Networks shown above are not different networks but are two copies of the same network, hence the name Siamese Networks. Basically they share the same parameters. The two input images ( $x_1$  and  $x_2$ ) are passed through the ConvNet to generate a fixed length feature vector for each ( $h(x_1)$  and  $h(x_2)$ ).

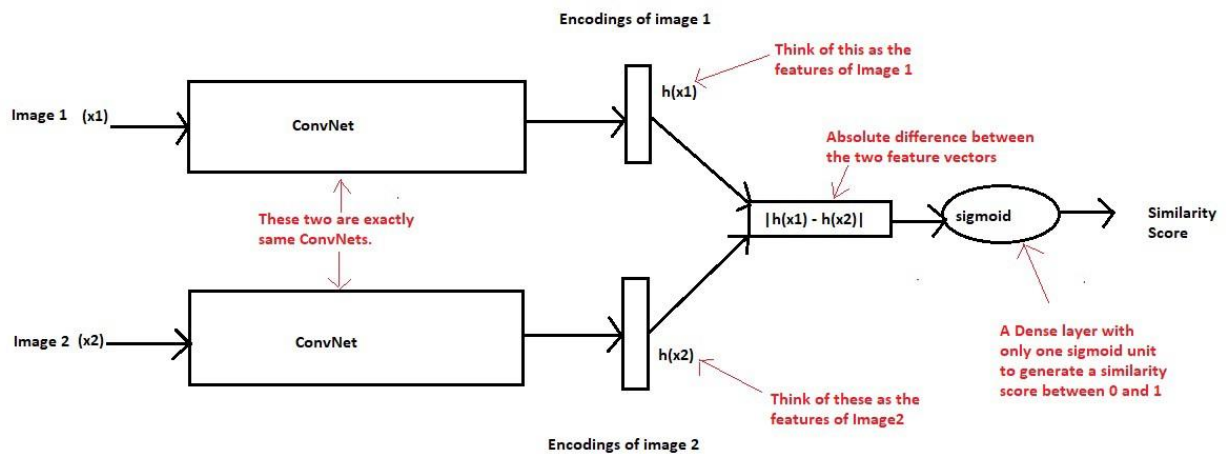


Figure 11. Siamese neural network architecture

Assuming the neural network model is trained properly, we can make the following hypothesis: If the two input images belong to the same character, then their feature vectors must also be similar, while if the two input images belong to the different characters, then their feature vectors will also be different. Thus, the element-wise absolute difference between the two feature vectors must be very different in both the above cases. And hence the similarity score generated by the output sigmoid layer must also be different in these two cases. This is the central idea behind the Siamese Networks.

### 3. 산학프로젝트의 기술적 내용

We started our project by understanding and studying bioinformatics. Bioinformatics is a field of computational science that has to do with the analysis of sequences of biological molecules. [It] usually refers to genes, DNA, RNA, or protein, and is particularly useful in comparing genes and other sequences in proteins. One can think about bioinformatics as essentially the linguistics part of genetics. That is, the linguistics people are looking at patterns in language, and that is what bioinformatics does--looking for patterns within sequences of DNA or protein.

The first step was to learn how protein detection works. After reading Steen, Hanno & Mann, Matthias. (2004). The ABC's (and XYZ's) of peptide sequencing. Nature reviews. Molecular cell biology. 5. 699-711. 10.1038/nrm1468, gave us a better understanding on how protein detection works on a molecular level (Fig: 12). Moreover, the peptides were divided into to labels, red and blue colors, were red represents a tumor tissue and blue a normal tissue.

## How to detect protein?

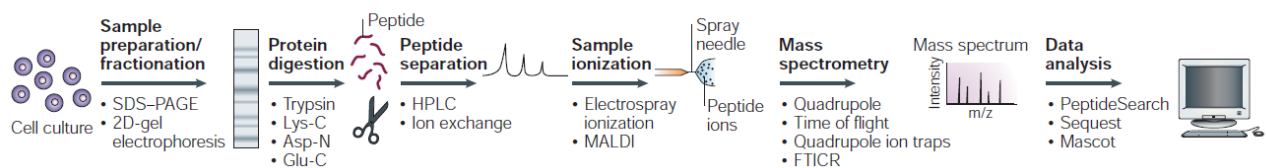
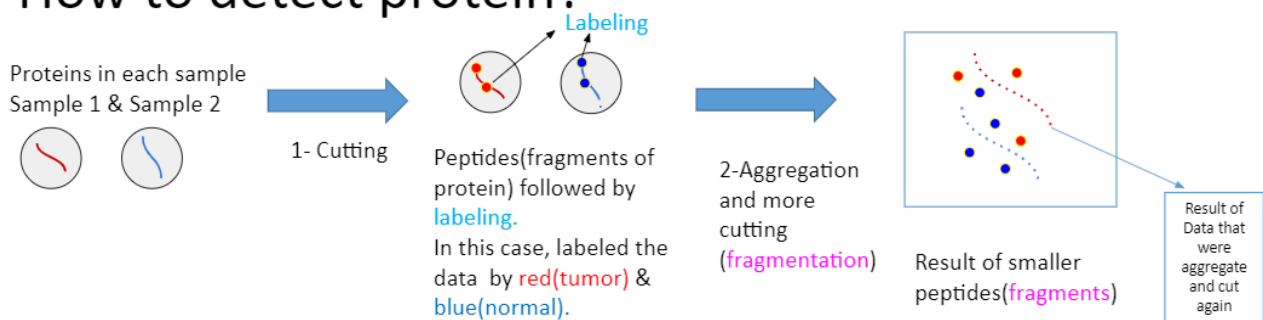
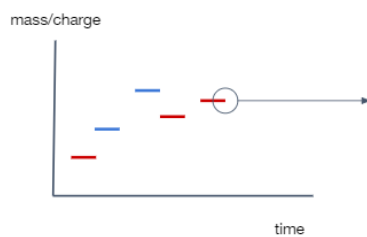


Fig 12a: How protein detection works

## How to detect protein?



After all the process done before, the Information that can be get from the peptides to analyze the data.

- Mass/charge of a **peptide**
- Detection time
- Mass/charge of the **fragments**
- Mass/charge of the **labels**

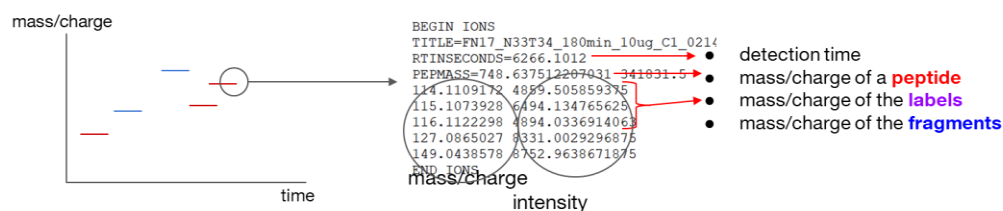
Fig 12b: How to detect protein. (Analysis)

### Data Description and Manipulation:

After analyzing data description, we moved on to the data manipulation and data visualization part of the project.

We were given MGF (mascot generic format) files. A Mascot data file is a plain text (ASCII) file containing peak list information and, optionally, search parameters. In the Mascot generic format, (MGF), each MS/MS dataset is a list of pairs of mass and intensity values, delimited by BEGIN IONS and END IONS statements. (Fig 13)

### Mascot Generic Format (MGF)



Labeling Mass	1-50 patient	51-80 patient
114.110680	Normal tissue for the <b>first</b> sample	Normal tissue for the <b>first</b> sample
115.107715	Tumor tissue for the <b>first</b> sample	Tumor tissue for the <b>first</b> sample
116.111070	Normal tissue for the <b>first</b> sample	Normal tissue for the <b>second</b> sample
117.114424	Tumor tissue for the <b>first</b> sample	Tumor tissue for the <b>second</b> sample

Fig 13: Mascot generic format

While working with the labeled data, after consulting with the professor, the allowable experimental error was set at 20 ppm.

PPM stands for parts for million. However, regarding to experimental error in mass spectrometry, ppm error means that “relative error to  $m/z$ ”. For example, 20 ppm error of the first labeling, 114.110680, is below:

$$(114.110680 \times 20 \times 10^{-6}) = 0.0022822136$$

Thus, the expected  $m/z$  of the first labeling (Fig 13) is (114.1083977864, 114.1129622136).

## Experimental Error



*Fig 13: Experimental Error*

Next step was to convert given MGF files into a specific format for better usability. The main objective to transform to this specific format was that data would be easier to work with (Fig 14).



### Convert MGF files into below format

```
BEGIN IONS
TITLE=FN17_N33T34_180min_10ug_C1_0214
RTINSECONDS=6266.1012
PEPMASS=748.637512207031 341831.5
114.1109172 4859.505859375
115.1073928 6494.134765625
116.1122298 4894.0336914063
127.0865027 8331.0029296875
149.0438578 8752.9638671875
END IONS

BEGIN IONS
TITLE=FN17_N33T34_180min_10ug_C1_0214:
RTINSECONDS=3092.43552
PEPMASS=630.743713378906 3669340.25
102.0553441 400538.4375
102.0611752 5298.041015625
103.0584364 4060.9057617188
112.0877126 6909.6235351563
112.159806 3252.7487792969
113.071436 17864.154296875
114.0554513 283779.15625
115.058226 7222.8466796875
115.0863344 4528.4658203125
116.0707225 16951.6640625
127.0865355 4800.4897460938
129.0653956 17705.59375
130.0503897 20927.279296875
131.0808696 11960.17578125
END IONS
```



PepMass	PepInt	Detection Time	Label114	Label115	Label116	Label117
748.637512207031	341831.5	6266.1012	4859.505859375	6494.134765625	4894.0336914063	0
630.743713378906	3669340.25	3092.43552	0	0	0	0

•Using 20 ppm of allowable experimental error

•Using the labeling table.



Labeling Mass
114.110680
115.107715
116.111070
117.114424

Fig 14: Convert MGF files into a specific format

### Curation and Quantification: isotope

Different existence of neutrons in chemical elements is called an isotope (the mass unit is 1.00235 Da).

For example, hydrogen does not have a neutron in general; however, some hydrogens have neutrons such as deuterium and tritium. Therefore, their mass are different each other by 1.00235 Da. For example, a mass difference between hydrogen and deuterium is 1.00235 (just a mass of neutron). A mass difference between hydrogen and tritium is 2.00470 (two neutrons) (Fig 15).

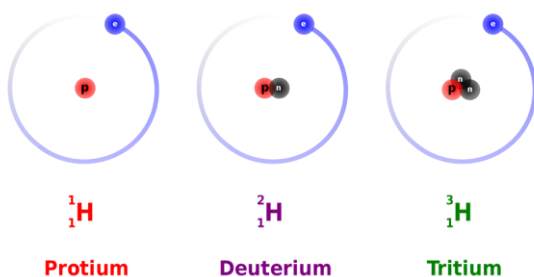
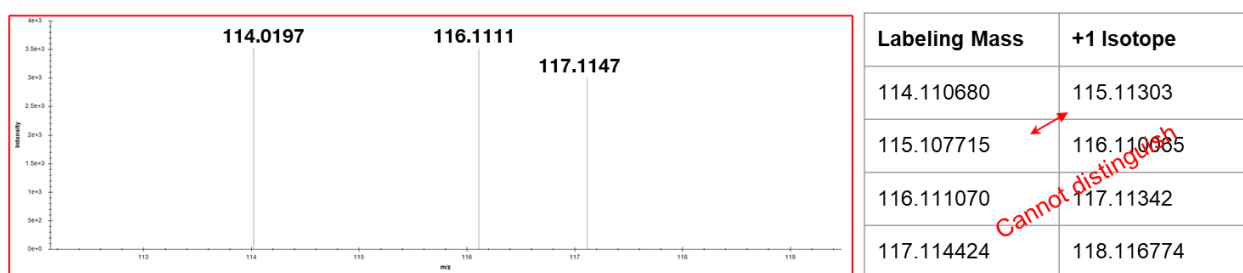


Fig 15: Hydrogen isotopes

Same principle works on each labeling mass. Because of isotope, there is a possibility of overlapping of each labeling peak. As can be seen in fig 16, labeling mass of 115.107715 and isotope of labeling mass of 114.110680, which is 115.11303 are overlapping. This type of overlapping leads to error in labeling. A correction table provided by the paper named i-tracker: For quantitative proteomics using iTRAQ™, Shandforth, I, P et al was used (Fig16).

## Curation and Quantification: i-Tracker



Because of isotope, each labeling peak can be overlapped each other. This result distorts the real values of labeling peaks. In this reason, a vendor provides correction table such like that:

Reagent	% of -2	% of -1	% of +1	% of +2
ITRAQ™ Reagent 114	0.0	1.0	5.9	0.2
ITRAQ™ Reagent 115	0.0	2.0	5.6	0.1
ITRAQ™ Reagent 116	0.0	3.0	4.5	0.1
ITRAQ™ Reagent 117	0.1	4.0	3.5	0.1

Based on the table, we can correct the distortion and quantify labeling peaks using i-Tracker algorithm.

*i-Tracker: For quantitative proteomics using iTRAQ™. Shadforth, I. P et al., 2005 BMC Genomics.*

Fig 16: Curation and Quantification: i-tracker

## Curation and Quantification: visualization

After implementing i-tracker algorithm to our dataset, the next step was to visualize it. This is the data visualization code (fig 18) that was used to save .png files of 24 MFG files (of 1 patient) as graphs. The graph reads as X axis as RT Retention time, Y axis as MZ (fig 17).



## Curation and Quantification: visualization

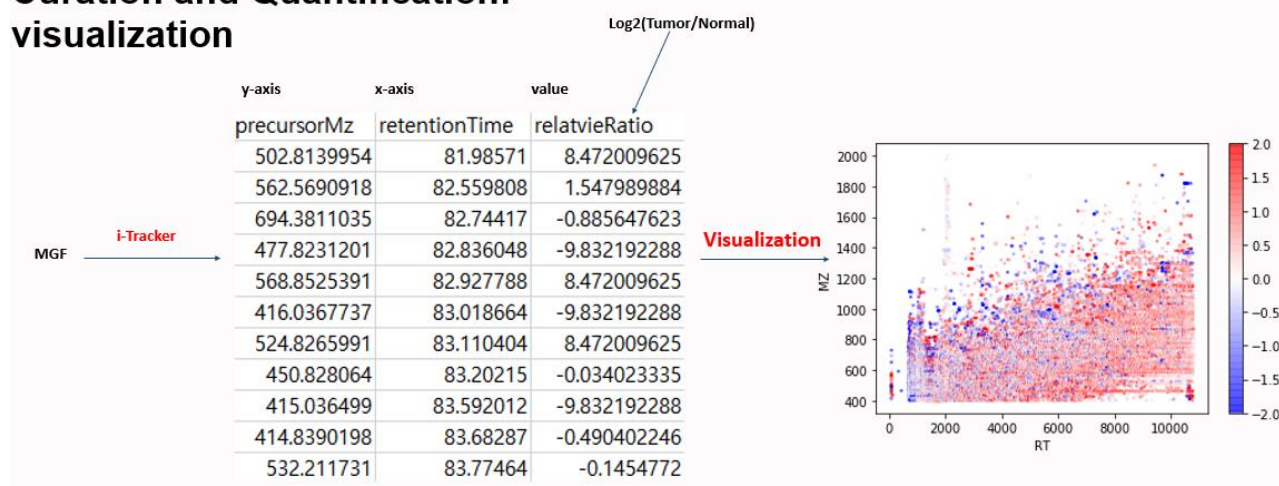


Fig17 : Visualtization

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import os
4
5 def getQuantTable(filename):
6     file = open(filename, 'r')
7     quantLines = []
8     header = file.readline().split()
9
10    ##readfile
11    while True:
12        line = file.readline()
13        if not line: break
14        if len(line) == 0: continue
15
16        floatData= [float(data)for data in line.split()]
17        quantLines.append(floatData)
18
19    file.close()
20
21    ##fileInfo
22    print(len(quantLines), "by" , len(quantLines[0]))
23
24    ##transpose
25    quantLines = list(zip(*quantLines))
26
27    return quantLines
28
29 def showPattern(from_, to_ ,quantTable, figure):
30     ##range setting
31     #plt.xlabel("RT")
32     #plt.ylabel("MZ")
33     plt.scatter(quantTable[1][from_:to_], quantTable[0][from_:to_], c= quantTable[2][from_:to_], s = 1, alpha=0.5,cmap=plt.cm.bwr,vmin=-2,vmax=2)
34     #plt.colorbar()
35     plt.xticks([])
36

```

Fig 18: python code used to implement i-tracker visualization



Our project code was written in python3.5. Some of the python libraries used for image processing are `matplotlib.pyplot`, `scipy.misc`, `numpy`. The Siamese network was implemented using Keras. The model has been trained using tensorflow backend in Keras. We loaded the images into tensors and later use these tensors to provide data in batches to the model. Once we load the train and test images, we save the tensors on the disk in a pickle file, so that we can utilize them later directly without having to load the images again.

Ideally, the model should have been trained on Cloud with a P4000 GPU. Training it on CPU will lead to OOM ERROR. Out of memory (OOM) is an often undesired state of computer operation where no additional memory can be allocated for use by programs or the operating system. Such a system will be unable to load any additional programs, and since many programs may load additional data into memory during execution, these will cease to function correctly. This usually occurs because all available memory, including disk swap space, has been allocated. However, we trained our model on the server with limited GPU.

## **Introduction and application method of one-shot learning**

A shot is nothing more than a single example available for training, so in N-shot learning, we have N examples for training. With the term “few-shot learning”, the “few” usually lies between zero and five, meaning that training a model with zero examples is known as zero-shot learning, one example is one-shot learning, and so on. All of these variants are trying to solve the same problem with differing levels of training material.

In one-shot learning, we only have a single example of each class. Now the task is to classify any test image to a class using that constraint. There are many different architectures developed to achieve this goal, such as Siamese Neural Networks, which brought about major progress and led to exceptional results, and then matching networks, which also helped us make great leaps in this field.

There are two ways of implementing one shot learning. Bayesian Decision Method and Siamese network method. Unlike Bayesian Method, Siamese DOES NOT rely on a predetermined model

framework. This is the basis of our CNN model.

We implemented this paper by Koch et al: <https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf>. Given the above intuition let's look at the picture of the architecture with finer details taken from the research paper itself:

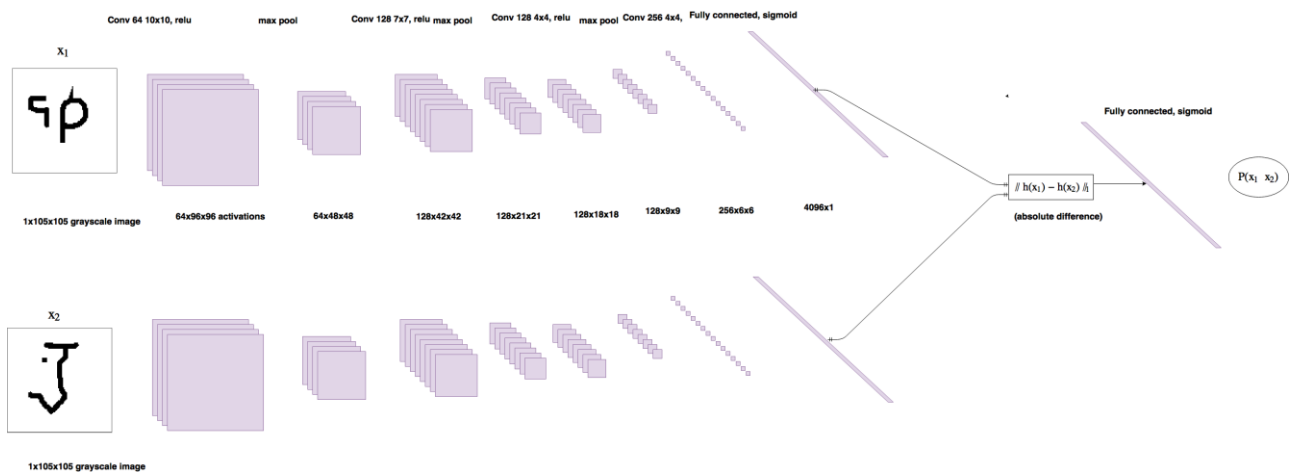


Fig 19: Siamese Neural Network Architecture

Koch et al uses a convolutional Siamese network to classify pairs of omniglot images, so the twin networks are both convolutional neural nets (CNNs). The twins each have the following architecture: convolution with 64 10x10 filters, relu -> max pool -> convolution with 128 7x7 filters, relu -> max pool -> convolution with 128 4x4 filters, relu -> max pool -> convolution with 256 4x4 filters. The twin networks reduce their inputs down to smaller and smaller 3d tensors, finally there is a fully connected layer with 4096 units. The absolute difference between the two vectors is used as input to a linear classifier. All up, the network has 38,951,745 parameters - 96% of which belong to the fully connected layer (fig19).

We had to change this paper's architecture to fit our RGBA Dataset. The referred architecture works with grayscale image meaning works with 1 channel. Our dataset images have 4 channels.



We had to alter the channels according to our needs.

The output is squashed into  $[0,1]$  with a sigmoid function to make it a probability. We use the target  $t=1$  when the images have the same class and  $t=0$  for a different class. It's trained with logistic regression. This means the loss function should be binary cross entropy between the predictions and targets. There is also a L2 weight decay term in the loss to encourage the network to learn smaller/less noisy weights and possibly improve generalization:

$$L(x_1, x_2, t) = t \cdot \log(p(x_1 \circ x_2)) + (1 - t) \cdot \log(1 - p(x_1 \circ x_2)) + \lambda \cdot \|w\|_2$$

When it does a one-shot task, the Siamese net simply classifies the test image as whatever image in the support set it thinks is most similar to the test image:

$$C(\hat{x}, S) = \underset{c}{\operatorname{argmax}} P(\hat{x} \circ x_c), x_c \in S$$

This uses an argmax unlike nearest neighbor which uses an argmin, because a metric like L2 is higher the more “different” the examples are, but this models outputs  $p(x_1 \circ x_2)$ , so we want the highest. E

### Effective dataset size in pairwise training

One interesting thing we noticed about training on pairs is that there are quadratically many possible pairs of images to train the model on, making it hard to overfit. Say we have  $C$  examples each of  $E$  classes. Since there are  $C \cdot E$  images total, the total number of possible pairs is given by:

$$N_{pairs} = \binom{C \cdot E}{2} = \frac{(C \cdot E)!}{2!(C \cdot E - 2)!}$$

In the original paper the omniglot dataset was used. For omniglot with its 20 examples of 964 training classes, this leads to 185,849,560 possible pairs, which suppose a huge amount. However, the Siamese network needs examples of both same and different class pairs. There are  $E$  examples per class, so there will be  $(E^2)$  pairs for every class, which means there are:



$$N_{same} = \binom{E}{2} \cdot C$$

possible pairs with the same class - 183,160 pairs for omniglot. Even though 183,160 example pairs is plenty, it's only a thousandth of the possible pairs, and the number of same-class pairs increases quadratically with E but only linearly with C. This is important because the Siamese network should be given a 1:1 ratio of same-class and different-class pairs to train on - perhaps it implies that pairwise training is easier on datasets with lots of examples per class.

For our dataset, we had 24 images for 12 training class. The possible pairs are 41,328, which is significantly small than the paper's dataset.

## Dataset

The dataset was arranged as followings: The total dataset was divided into 2 main categories. The images\_background (used for training) and images\_evaluation (used for evaluation). We had type-1, type-2, type-3, type-4, in total 4 types of cancers cells to work with and in total there were 17 patients. After getting the labeled data of each patients, they were put into their respective types. The train-evaluation set was divided (60%-40%). Meaning type-1, type-2, type-3 were put into images\_background and type-4 was put into images\_evaluation. For testing purpose, the types were interchanged.

## The Code:

### Mapping the problem to binary classification task

Let's understand how we can map this problem into a supervised learning task where our dataset contains pairs of (Xi, Yi) where 'Xi' is the input and 'Yi' is the output.

Recall that the input to our system will be a pair of images and the output will be a similarity score between 0 and 1.

Xi = Pair of images

Yi = 1 ; if both images contain the same characteristics

Yi = 0; if both images contain different characteristic. We will use the below generator



function to generate data in batches during the training of the network. Let's have a better understanding by visualizing the dataset below:







Xi		Yi
Image 1	Image 2	Similarity Score
		S1 = 1
		S2 = 0
		S3 = 0

Fig 20: Similarity Score

Thus we need to create pairs of images along with the target variable, as shown above, to be fed as input to the Siamese Network. Note that even though images from type-1 are shown above (fig20), but in practice we will generate pairs randomly from all the types in the training data.

The code to generate these pairs and targets is shown below:

```
def get_batch(batch_size,s="train"):

    """Create batch of n pairs, half same class, half different class"""

    if s == 'train':

        X = Xtrain

        categories = train_classes

    else:

        X = Xval

        categories = val_classes

    print(X.shape)

    n_classes, n_examples, h, w, ch = X.shape

    # randomly sample several classes to use in the batch
```



```
categories = rng.choice(n_classes,size=(batch_size,),replace=True)
```

```
# initialize 2 empty arrays for the input image batch
```

```
pairs=[np.zeros((batch_size, h, w,ch)) for i in range(2)]
```

```
# initialize vector for the targets
```

```
targets=np.zeros((batch_size,))
```

```
# make one half of it '1's, so 2nd half of batch has same class
```

```
targets[batch_size//2:] = 1
```

```
for i in range(batch_size):
```

```
    category = categories[i]
```

```
    idx_1 = rng.randint(0, n_examples)
```

```
    pairs[0][i,:,:,:] = X[category, idx_1].reshape(h, w, ch)
```

```
    idx_2 = rng.randint(0, n_examples)
```

```
# pick images of same class for 1st half, different for 2nd
```

```
if i >= batch_size // 2:
```

```
    category_2 = category
```

```
else:
```

```
    # add a random number to the category modulo n classes to ensure 2nd image has a  
different category
```

```
    category_2 = (category + rng.randint(1,n_classes)) % n_classes
```



```
pairs[1][i,:,:,:] = X[category_2,idx_2].reshape(h, w,ch)
```

```
return pairs, targets
```

We need to call the above function by passing the batch\_size and it will return “batch\_size” number of image pairs along with their target variables.

We will use the below generator function to generate data in batches during the training of the network.

```
222 def generate(batch_size, s="train"):
223     while True:
224         pairs, targets = get_batch(batch_size,s)
225         yield (pairs, targets)
226
```

## Model Architecture and Training

Here is the model definition. We only define the twin network’s architecture once as a Sequential() model and then call it with respect to each of two input layers, this way the same parameters are used for both inputs. Then merge them together with absolute distance, add an output layer, and compile the model with binary cross entropy loss.

```
100 def initialize_weights(shape, dtype=None):
101     """
102     The paper, http://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf
103     suggests to initialize CNN layer weights with mean as 0.0 and standard deviation of 0.01
104     """
105     return np.random.normal(loc = 0.0, scale = 1e-2, size = shape)
106
107 def initialize_bias(shape, dtype=None):
108     """
109     The paper, http://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf
110     suggests to initialize CNN layer bias with mean as 0.5 and standard deviation of 0.01
111     """
112     return np.random.normal(loc = 0.5, scale = 1e-2, size = shape)
```





```
114 def get_siamese_model(input_shape):
115     """
116     Model architecture based on the one provided in: http://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf
117     """
118
119     # Define the tensors for the two input images
120     left_input = Input(input_shape)
121     right_input = Input(input_shape)
122
123     # Convolutional Neural Network
124     model = Sequential()
125     model.add(Conv2D(64, (10,10), activation='relu', input_shape=input_shape,
126                     kernel_initializer=initialize_weights, kernel_regularizer=l2(2e-4)))
127     model.add(MaxPooling2D())
128     model.add(Conv2D(128, (7,7), activation='relu',
129                     kernel_initializer=initialize_weights,
130                     bias_initializer=initialize_bias, kernel_regularizer=l2(2e-4)))
131     model.add(MaxPooling2D())
132     model.add(Conv2D(128, (4,4), activation='relu', kernel_initializer=initialize_weights,
133                     bias_initializer=initialize_bias, kernel_regularizer=l2(2e-4)))
134     model.add(MaxPooling2D())
135     model.add(Conv2D(256, (4,4), activation='relu', kernel_initializer=initialize_weights,
136                     bias_initializer=initialize_bias, kernel_regularizer=l2(2e-4)))
137     model.add(Flatten())
138     #We had to change 4096-> 10 for limitation purposes
139     model.add(Dense(4096, activation='sigmoid',
140                     kernel_regularizer=l2(1e-3),
141                     kernel_initializer=initialize_weights,bias_initializer=initialize_bias))
```

```
143     # Generate the encodings (feature vectors) for the two images
144     encoded_l = model(left_input)
145     encoded_r = model(right_input)
146
147     # Add a customized layer to compute the absolute difference between the encodings
148     L1_layer = Lambda(lambda tensors:K.abs(tensors[0] - tensors[1]))
149     L1_distance = L1_layer([encoded_l, encoded_r])
150
151     # Add a dense layer with a sigmoid unit to generate the similarity score
152     prediction = Dense(1,activation='sigmoid',bias_initializer=initialize_bias)(L1_distance)
153
154     # Connect the inputs with the outputs
155     siamese_net = Model(inputs=[left_input,right_input],outputs=prediction)
156
157     # return the model
158     return siamese_net
159
160 model = get_siamese_model((480, 640, 4))
161 model.summary()
```

```
optimizer = Adam(lr = 0.00006)
```

```
model.compile(loss="binary_crossentropy",optimizer=optimizer)
```

The model was compiled using the adam optimizer and binary cross entropy loss function as shown above. Learning rate was kept low as it was found that with high learning rate, the model took a lot of time to converge. However, these parameters can well be tuned further to improve the



present settings.

The original paper used layerwise learning rates and momentum – we skipped this because it; was kind of messy to implement in keras and the hyperparameters aren't the most important part of the paper. Koch et al adds examples to the dataset by distorting the images and runs experiments with a fixed training set of up to 150,000 pairs. Since that won't fit in our computer's memory, we decided to just randomly sample pairs. Loading image pairs was probably the hardest part of this to implement. Since there were 24 examples for every class, the data was reshaped into  $N\_classes \times 24 \times 480 \times 640 \times 4$  arrays, to make it easier to index by category.

Moving on to training loop:

```
273 evaluate_every = 200 # interval for evaluating on one-shot tasks
274 batch_size = 2
275 n_iter = 4000 # No. of training iterations
276 N_way = 1 # how many classes for testing one-shot tasks
277 n_val = 250 # how many one-shot tasks to validate on
278 best = -1
279
280 model_path = './weights/'
281
282 print("Starting training process!")
283 print("-----")
284 t_start = time.time()
285 for i in range(1, n_iter+1):
286     (inputs, targets) = get_batch(batch_size)
287     loss = model.train_on_batch(inputs, targets)
288     if i % evaluate_every == 0:
289         print("\n ----- \n")
290         print("Time for {0} iterations: {1} mins".format(i, (time.time()-t_start)/60.0))
291         print("Train Loss: {0}".format(loss))
292         val_acc = test_oneshot(model, N_way, n_val, verbose=True)
293         model.save_weights(os.path.join(model_path, 'weights.{0}.h5'.format(i)))
294         if val_acc >= best:
295             print("Current best: {0}, previous best: {1}".format(val_acc, best))
296             best = val_acc
297
298 model.load_weights(os.path.join(model_path, "weights.1.h5"))
```

The model was trained for 4000 iterations with batch size of 2.

After every 200 iterations, model validation was done using 20-way one shot learning and the accuracy was calculated over 250 trials. This concept is explained in the next section.

## Validating the Model

Moving onto validate and test our model. For every pair of input images, our model generates a similarity score between 0 and 1. But just looking at the score it's difficult to ascertain whether the model is really able to recognize similar characteristics and distinguish dissimilar ones. A simpler way of looking into the model is N-way one shot learning. An example of 3-way one shot learning (fig 21).


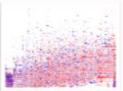
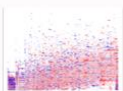



Xi		Yi
Image 1	Image 2	Similarity Score
		S1 = 1
		S2 = 0
		S3 = 0

Fig 21: 3 way one shot learning

Basically the same image is compared to 3 different images out of which only one of them matches the original image. Let's say by doing the above 3 comparisons we get 3 similarity scores S1, S2, S3 and as shown. Now if the model is trained properly, we expect that S1 is the maximum of all the 3 similarity scores because the first pair of images is the only one where we have two same images.

Thus if S1 happens to be the maximum score, we treat this as a correct prediction otherwise we consider this as an incorrect prediction. Repeating this procedure 'k' times, we can calculate the percentage of correct predictions as follows:

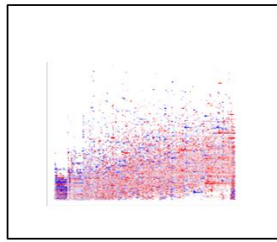
$$\text{percent\_correct} = (100 * n\_correct) / k$$

where k => total no. of trials and n\_correct => no. of correct predictions out of k trials.



Similarly, Below is how a 25-way one shot learning would look like:

Test Image



The similarity Score for this comparison is expected to be the highest

Support Set

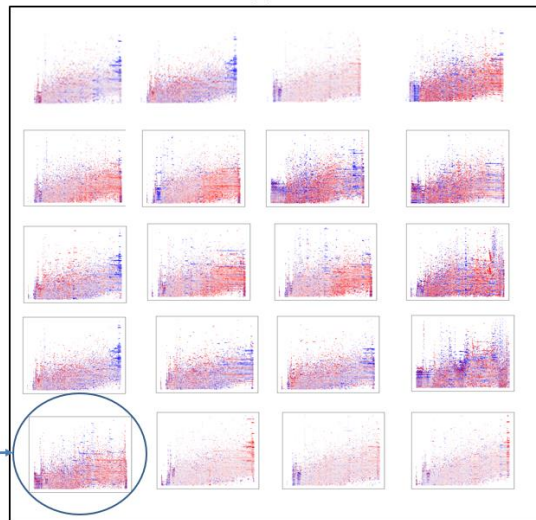


Fig 22: Support Set

It's quite obvious that smaller values of 'N' will lead to more correct predictions and larger values of 'N' will lead to relatively less correct predictions when repeated multiple times. In practice we will generate the test image and the support set randomly from all the types of the test/validation dataset images (fig22).



The code to generate the test image along with the support set is as follows:

```
227 def make_oneshot_task(N,k, s="val", language=None):
228     """Create pairs of test image, support set for testing N way one-shot learning. """
229     if s == 'train':
230         X = Xtrain
231         categories = train_classes
232     else:
233         X = Xval
234         categories = val_classes
235         n_classes, n_examples, h, w, ch = X.shape
236
237         indices = rng.randint(0, n_examples,size=(N,))
238         if language is not None: # if language is specified, select characters for that language
239             low, high = categories[language]
240             if N > high - low:
241                 raise ValueError("This language ({} has less than {} letters".format(language, N))
242             categories = rng.choice(range(low,high),size=(N,),replace=False)
243
244         else: # if no language specified just pick a bunch of random letters
245             categories = rng.choice(range(n_classes),size=(N,),replace=True)
246         true_category = categories[0]
247         ex1, ex2 = rng.choice(n_examples,replace=True,size=(2,))
248         test_image = np.asarray([X[true_category,ex1,:,:]*N].reshape(N, h, w,ch)
249         support_set = X[categories,indices,:,:)
250         support_set[0,:,:) = X[true_category,ex2]
251         support_set = support_set.reshape(N, h, w,ch)
252         targets = np.zeros((N,))
253         targets[0] = 1
254         targets, test_image, support_set = shuffle(targets, test_image, support_set)
255         pairs = [test_image,support_set]
256         return pairs, targets
```

```
258 def test_oneshot(model, N, k, s = "val", verbose = 0):
259     """Test average N way oneshot learning accuracy of a siamese neural net over k one-shot tasks"""
260     n_correct = 0
261     if verbose:
262         print("Evaluating model on {} random {} way one-shot learning tasks ... \n".format(k,N))
263     for i in range(k):
264         inputs, targets = make_oneshot_task(N,s,k)
265         probs = model.predict(inputs)
266         if np.argmax(probs) == np.argmax(targets):
267             n_correct+=1
268     percent_correct = (100.0 * n_correct / k)
269     if verbose:
270         print("Got an average of {}% {} way one-shot learning accuracy \n".format(percent_correct,N))
271     return percent_correct
```

Here, we monitor one-shot tasks validation accuracy to test performance, rather than loss on the validation set.

## Test Results and Inference



The N-way testing was done for  $N = 1, 3, 5, \dots, 19$ . For each N-way testing, 50 trials were performed and the average accuracy was computed over these 50 trials. To get the best accuracy possible, we tried with different scenarios where the value to parameters were changed for analyzing every possible outcome.

evaluate\_every = 200 to 400 # interval for evaluating on one-shot tasks

batch\_size = 1 to 2

n\_iter = 2000 to 20,000 # No. of training iterations

N\_way = 1 # how many classes for testing one-shot tasks

n\_val = 250 # how many one-shot tasks to validate on

trials = 50, 80, 100

## **Base Line 1 — Nearest Neighbor Model**

It is always a good practice to create simple baseline models and compare their results with complex model you are trying to build. Our first baseline model is the Nearest Neighbor approach.

If you are familiar with K-Nearest Neighbor algorithm, then this is just the same thing.

As discussed above, in an N-way one shot learning, we compare a test image with N different images and select that image which has highest similarity with the test image as the prediction.

This is intuitively similar to the KNN with  $K=1$ .

## **Base Line 2 — Random Model**

Creating a random model which makes prediction at random is a very common technique to make sure that the model we created is at least better than a model which makes completely random predictions.



Below is the comparison between 4 models:

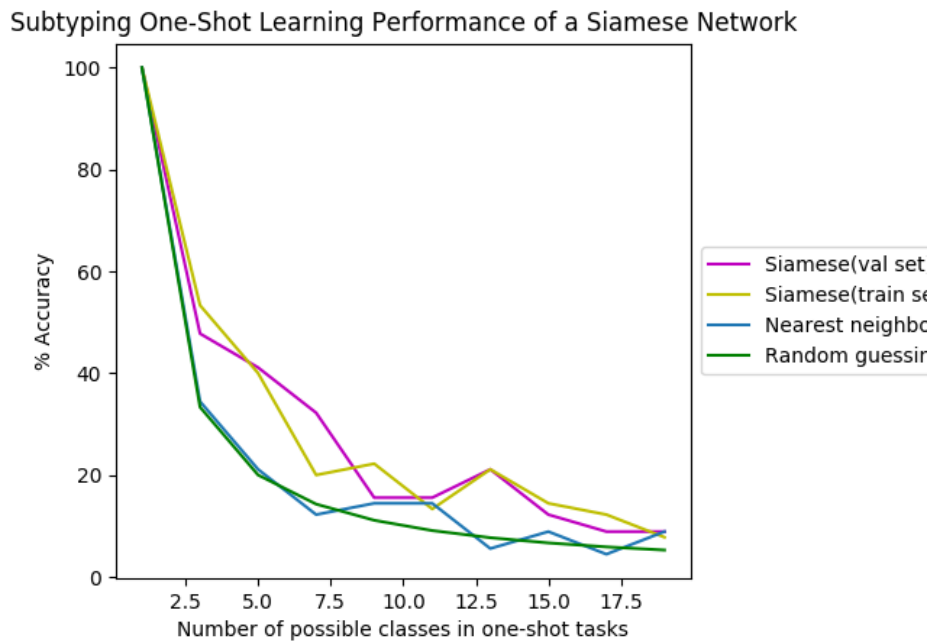


Fig 23: Performance of a Siamese network

## Description of the difficulties and the process of resolution

### Merging RGBA images:

Our main approach was to merge all the images belonging to one patient. However, it proved to be a difficult thing to do. We tried different approaches i.e. guided filtering fusion algorithm, Difference extractor: to calculate data loss, blending technique. Unfortunately, we were faced with unusable data because of massive data loss. Our second idea was to work with sub-samples from each image from a specific area. To calculate how much overlapping we were having, we went back to raw data. We worked with the .csv files and calculated the normal gaussian curve over multiple grid sizes. This gave an idea on how much overlapping we were having and how much data we were losing.

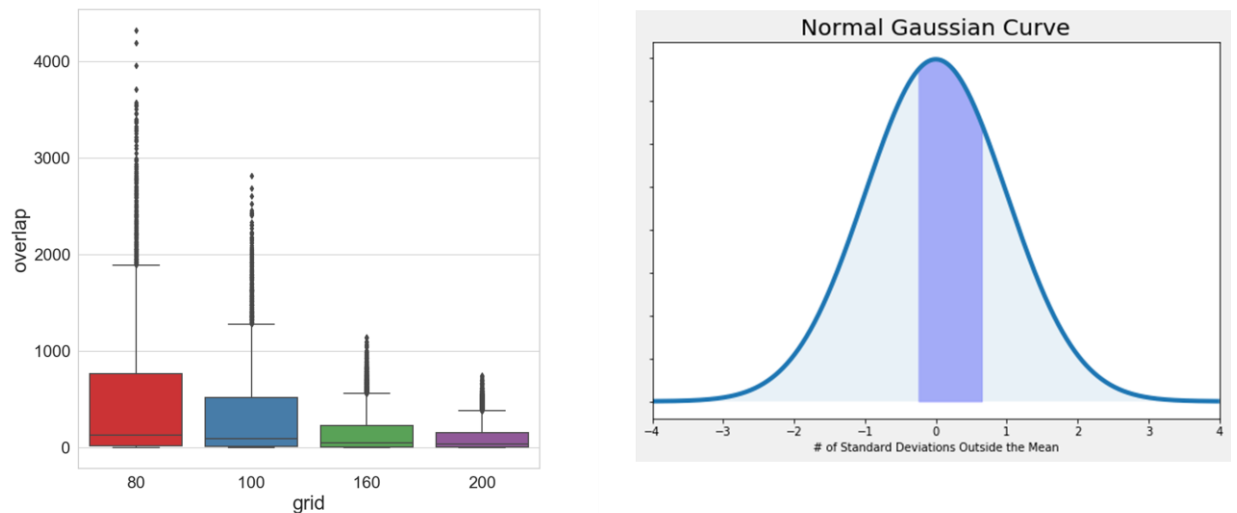


Fig 24: Gaussian Curve

### Channel Transformation:

One of the first difficulties we had to overcome was the transformation of the image channels. As it can be seen from the referred paper, the Siamese twin network architecture is designed for greyscale images a.k.a images with channel 1 only. However, our datasets were composed of 4 channels a.k.a RGBA images. This will also lead to another obstacle, since working with RGBA images leads to the usage of a significant amount of memory.

### CPU and GPU limitation:

When the code was first ran in our laptops, we got OOM error which means it lacked the memory capacity. So the proposed solution was to use cloud computing platforms like google colab. However unfortunately, google colab had the similar issue. It ran out of GPU and timeout. Our final solution was to ask our TA to give us a server/cloud with at least PA4000 GPU capacity.

We were provided a server from the lab. However, the server still had the following computational error: cannot create a tensor proto whose content is larger than 2GB. Our solution was to change the 4096 units to 10 units. In the paper architecture, Twin networks reduce their inputs down to smaller and smaller 3d tensors, finally there is a fully connected layer with 4096





units. We were finally able to run the code on our dataset and receive a result. An accuracy of between 40%~100% was obtained, which is not really the expected accuracy but since the architecture was compromised, the accuracy was compromised as well.

### **Batch Size:**

Calculating the batch size was a difficult task. The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. When all training samples are used to create one batch, the learning algorithm is called batch gradient descent. When the batch is the size of one sample, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

- **Batch Gradient Descent.** Batch Size = Size of Training Set
- **Stochastic Gradient Descent.** Batch Size = 1
- **Mini-Batch Gradient Descent.**  $1 < \text{Batch Size} < \text{Size of Training Set}$

In the case of mini-batch gradient descent, popular batch sizes include 32, 64, and 128 samples.

We were unable to run batch 32 as done in the paper, because of hardware limitation. We had to keep our batch size small in between 1 and 2.

### **Training and Evaluation classes:**

In the paper, 30 classes were trained in the model and 20 classes were used for evaluation (60%-40%). In our case, we had only 4 classes, so we split it into 3 classes in images\_train and 1 class in images\_evaluation.



#### 4. 산학프로젝트의 역할 분담

##### 4.1 개별 임무 분담

번호	학과	학번	학년	이름	담당업무
1	컴퓨터소프트웨어	2017053736	4	Saira Tahsin 사이라 타신	Theory, Presentation, Testing and Troubleshooting, Write reports
2	컴퓨터소프트웨어	2017053772	4	Paz 지메네스 소스도 파스 카롤리나	Testing and Troubleshooting, presentation, Write reports
3	컴퓨터소프트웨어	2016056617	4	Isai 이사이 까레또 마르띠네스	Implementation, Testing and Troubleshooting, Write reports



[illegible]



## 5. 결론 및 기대효과

Regardless of the difficulties we went through, we had an enormous support from our mentor, and a positive mindset from each member of the team. As expressed before, we steered the helm in directions where there was no tangible solution for the problems we tried to solve. At times, we were about to reach a point where resolutions were obtained, but they were not reliable. Therefore, we had to develop a problem-solving mindset where we defined the problem in detail, then we generated alternative results where the last of them was to go a step back and change our approach to go forward with the project. We ended up evaluating and choosing an alternative to finally implement and follow up with the optimal solution. Along with the problem-solving mindset, we developed a closer knowledge of machine learning models, their differences, their algorithms, their ideal hyperparameters, and its implementation in a real situation field such as biometrics in which we as a team were not familiarized. Our strategy focused on the performance of the one-shot classification by initially learning deep convolutional Siamese neural networks for verification. We expected our model to perform all available baselines by a significant margin and come close to the best numbers in the accuracy. Nevertheless, with three classes on the training dataset and one class on the testing dataset we achieved a 56% accuracy for a 3-way testing, which was our highest accuracy level with a batch size of 2 due to the hardware limitations, over 4,000 iterations.

Concerning the future plan of the project, we can run the model on a powerful computer adding extra data for simplicity and refinement. On the other hand, with respect to the member's plans regarding the future, this project influenced and extended our view in machine learning, computer vision and artificial intelligence in general.



## References

- (n.d.). Retrieved November 29, 2020, from <https://cs231n.github.io/convolutional-networks/>
- Chen, D. (2020, April 12). Statistical Learning: Data Sampling & Resampling. Retrieved November 29, 2020, from <https://towardsdatascience.com/statistical-learning-ii-data-sampling-resampling-93a0208d6bb8>
- Hlamba28. (n.d.). Hlamba28/One-Shot-Learning-with-Siamese-Networks. Retrieved November 29, 2020, from <https://github.com/hlamba28/One-Shot-Learning-with-Siamese-Networks>
- Kazarinoff, P. (2019, February 02). Plotting a Gaussian normal curve with Python and Matplotlib. Retrieved November 29, 2020, from <https://pythonforundergradengineers.com/plotting-normal-curve-with-python.html>
- Koch, G., Zemel, R., & Salakhutdinov, R. (n.d.). Siamese Neural Networks for One-shot Image Recognition. Retrieved November 30, 2020, from <https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf>
- Lamba, H. (2019, February 17). One Shot Learning with Siamese Networks using Keras. Retrieved November 29, 2020, from <https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d>
- Raj, J. (2019, March 14). Dimensionality Reduction for Machine Learning. Retrieved November 29, 2020, from <https://towardsdatascience.com/dimensionality-reduction-for-machine-learning-80a46c2ebb7e>
- Steen, H., & Mann, M. (2004). THE ABC'S (AND XYZ'S) OF PEPTIDE SEQUENCING. Retrieved November 30, 2020