*Your name here*

*Your report title here*

Table of Contents

[1. Introduction 1](#_Toc99955248)

[2. Background 1](#_Toc99955249)

[3. Problem statement 1](#_Toc99955250)

[4. Project objectives 2](#_Toc99955251)

[5. Data warehouse details 2](#_Toc99955252)

[6. Interesting insights 2](#_Toc99955253)

[7. Data pre-processing 2](#_Toc99955254)

[8. Measuring performance 2](#_Toc99955255)

[9. Algorithms applied 3](#_Toc99955256)

[9.1 Algorithm 1 3](#_Toc99955257)

[9.2 Algorithm 2 3](#_Toc99955258)

[9.3 Algorithm 3 3](#_Toc99955259)

[9.4 Comparison 3](#_Toc99955260)

[10. Discussion 3](#_Toc99955261)

[11. Conclusion 3](#_Toc99955262)

[References 4](#_Toc99955263)

1. **Introduction**

Skin diseases affect almost 900 million people in the world at any time (World Health Organization, 2018). The diseases can impact an individual’s quality of life and health. In this report we will present three machine learning models that can classify eight different skin diseases. It will include the performance and accuracy of each model, as well as comparisons.

By making a machine learning model that can classify skin disease fast and accurately, it can be a tool for doctors to diagnose skin diseases. Furthermore, it can increase efficiency and reduce costs in the health care system.

1. **Background**

The healthcare industry is one of the biggest industries in the world (Wikipedia,2023). This is one of the reasons why we chose to base our project on this domain. The possibilities machine learning can have in the healthcare industry is monumental and can be used as a tool to help improve both the quality of care and efficiency.

According to (Džakula et al., 2022) the shortage of healthcare workers is a worldwide problem. Additionally, the health care industry is estimated to need 80 million more workers by 2030 to meet demand (Deloitte,2023). This highlights the need for tools to make the system more efficient, but also make the machine learning model that we created a tool for healthcare workers by maybe making their job easier. Furthermore, it can also help the patients who are affected with skin diseases get diagnosed and get treatment faster.

**2.1** **Skin diseases**

A skin disease is a condition that affects the skin. The skin disease can cause inflammation, rashes, itchiness and/or other skin changes (Cleveland Clinic, 2021). For example, cellulitis, which is one skin disease we are training the machine learning model to classify, it can cause redness, swelling, warmth, and pain in the affected area. There are also different causes to skin diseases, e.g., bacteria, fungal infections, or viruses (Higuera, 2023). The skin diseases we are learning the machine learning model to identify can be caused by bacteria, fungal, parasites and viruses. The cause and the names of the skin diseases we are classifying is:

1. Bacterial Infections- cellulitis
2. Bacterial Infections- impetigo
3. Fungal Infections - athlete -foot
4. Fungal Infections - nail-fungus
5. Fungal Infections - ringworm
6. Parasitic Infections - cutaneous-larva-migrans
7. Viral skin infections - chickenpox
8. Viral skin infections – shingles

So, there is four different causes to the skin diseases that the machine learning model is going to classify. Some of these skin diseases are common, other ones are rarer. For example, cellulatis is a common bacterial skin infection (Cellulitis: All you need to know 2022). Other ones are less common, for example, cutaneous larva migrans, because this is most common in tropical environments (Dinulos, 2023). Unfamiliar skin diseases can be a challenge for doctors due to their limited exposure to these cases. Next section we will look at the diagnosing and treatment of skin disease.

* 1. **Diagnosing and treatment**

In a study done the results were that nondermatologists perform poorly in both diagnosing and treatment of skin diseases (Federman et al., 2004). The most common way for a doctor to diagnose skin disease is by visually examining the skin (Cleveland Clinic, 2021). Which is why a machine learning model that is trained on images of skin disease can be an effective tool. Some of the skin diseases we are training the machine learning model to classify is a little similar, e.g., shingles and chickenpox. Because of this it can be difficult for a doctor who is not specialized in skin disease to diagnose the skin disease accurately. With the machine learning model, we want to make a model that is enough accurate to distinguish between these two. There are several ways to treat a skin disease. Some of the skin diseases may also go away by its own. Some common treatments of skin diseases include creams, lotions, oral medication, light therapy, and surgery (Cleveland Clinic, 2021).

There are several challenges associated with diagnosing and treating skin diseases. One challenge is that there are many different types of skin diseases, and they can sometimes be difficult to distinguish from each other, which is mentioned above. Another challenge is that some skin diseases, such as shingles, can be difficult to manage and treat (Johns Hopkins Medicine, 2023). Furthermore, a machine learning model can maybe faster diagnose a skin disease, which can lead to faster treatment. If we take shingles as an example, it is important to start treatment early with this skin disease because some medication is not as effective if the rash has been present for 2-3 days (Johns Hopkins Medicine, 2023).

* 1. **Machine learning**

Machine learning has the potential to be a powerful tool to the diagnosis and treatment of skin diseases. Machine learning models can be trained on large datasets of images of skin diseases to learn to identify patterns that are associated with different skin diseases. This can help doctors to diagnose skin diseases more quickly and accurately than traditional methods. We hope that trough the machine learning model that we have made, we will be able to achieve this.

In the future, machine learning could play an even greater role in dermatology. For example, machine learning could be used to develop new diagnostic tools, such as mobile apps that can allow people to take pictures of their skin and receive a diagnosis. Machine learning could also be used to develop new treatments for skin diseases.

1. **Problem statement**

There is a need for more efficient and accurate methods for diagnosing skin diseases.

1. **Project objectives**

Develop a machine learning model that can accurately diagnose skin diseases from images.

1. **Data warehouse details**

The data we used came from a free open-sourced website called Kaggle. The URL to the website is <https://www.kaggle.com/datasets/subirbiswas19/skin-disease-dataset>. It has a public domain license, which means that it free to use by anyone, and there is no restriction (Research guides @ fordham, 2023).

The dataset was collected using internet search. The images of the different skin diseases were then put in labeled folders, with the cause of the skin disease and the name e.g., bacterial infections-cellulites. There is a total of eight skin diseases. The different skin diseases were also then put into two folders, one test\_set folder and train\_set folder. The train\_set folder has the highest number of images with minimun 100 images. The test\_set folder has a minimun of 30 pictures. There is a total of 1159 images of the different skin diseases in the dataset.

Since the dataset already was divided into train\_set folders and test\_set folders we downloaded the folders as they were. But we thought that there might not be enough images to train the models on. The more images the models have to train on the better the accuracy. Which is why we decided to augment the data so there would be more images for the model to train on. Furthermore, the augmented data may be more beneficial for the machine learning model because it can prevent overfitting and it can make the model work well on unseen data.

**5.1 Ethical considerations**

The ethical use of the data is important. The data we are using are pictures of skin diseases. Skin diseases can also appear in faces, which can cause a privacy issue. The images used to train and test the machine learning model we have has no copyright to it, but still there is important to think about the ethics and privacy. We have looked through the images of the data and the data looks anonymous because there are no full pictures of faces. For example, in the folders with skin disease of impetigo, there are pictures of faces, but not of the eyes, which makes it impossible to see who that person is.

Also, let’s say that we want new images to train and test the machine learning model on, then it is important to get permission from the individual so that we can use it in our machine learning model. Furthermore, if this model was made into an app, and people could take pictures of the affected area to classify the skin disease, then it is important that the data is not stored or used without permission.

There are also ethical considerations when there comes to bias in the dataset. We have looked through the data and there seems to be mostly images of white people, this can make the model biased against people of color. This can cause an issue with the accuracy of the model. For example, ringworm, which is one of the skin diseases we are training the model on can appear different on darker skin. In light skin the ring is usually red or pink, but it is usually brown or gray on darker skin (Higuera, 2023).

Overall, while the data is open access there is still important to keep the ethical considerations in mind. Privacy and accuracy are crucial when dealing with healthcare-related datasets.

1. **Interesting insights**

**Having a first look** at the data that we were going to work with is very important, there could have been things we would have to take into consideration when creating the code such as the color usage in the images, which kind of body-parts the infections happened to be located at, similarities and differences in the classes. These were different things that could be seen by quick inspection of the dataset. We decided to keep all the eight different classes that the dataset provided us with even though we saw similarities and problems. We were optimistic that we could create models that could distinguish these similarities in the dataset and took this as a challenge.

Since our dataset consisted of only images there were not many other insights, we could get outside from just having a look at the dataset, there would have been a different scenario if the dataset was purely datapoints. Which in that case, different tools could have been used to have a look at the data, such as finding patterns or strong features. In our case, we just had a look at the dataset and kept some of the finds in mind for later.

1. **Data pre-processing**

**Data preprocessing** is a necessary step when preparing for machine learning, the data that goes in needs to be in the correct format to train and predict. When dealing with data it is important to first get an overview of how the data looks to begin with, then what size and shape the desired data needs to be. The goal of this project was to get an image input, then to predict what kind of disease detected in the image. This meant that we had to figure out how we would have to process image data in a way that the machine learning models accepted.

The first step was experimenting with loading an image using the OpenCV library, this would then load in the image for further processing. Numpy is a powerful library for data processing, and combined with OpenCV we could read and shape the data. The dataset of choice contains large amounts of images, this meant that we had to process one image at a time, then combine all these images into a larger array containing all the images in each class. This is where we began, and we quickly realized that there had to be a better way. This is when we started creating our own DataProcessor class, that would handle all aspects of the data processing.

The DataProcessor class has a lot of functions such as generating X and Y data, generating necessary folders, augmenting data using OpenCV, train-test and validation splitting and visualizing the data. The reason for this choice is flexibility, we wanted a great way of testing different techniques fast, such as generating data from the original dataset and the augmented dataset and the different ways of augmenting the images. Putting the extra work into the data loading we were able to automatically detect classes within the dataset folder and load in the images in the desired format and splitting this into X and Y without having to manually change how different parts of the code worked. Using parameters, we were able to create a powerful class that allowed for easy changes.

When creating this class, we thought of different ways in which the data could be shaped, the data needed to be flattened such that the image data and the label only required one row. This means that every pixel, consisting of 3 values (R,G,B) using images of 224 x 224 would give us a total of 150 528 features per image. The data processing takes these steps:

**Load class folder -> Load image -> Reshape to 224 x 224 -> Flatten Data -> Add Label**

When this process was complete, the function loops over each class and adds all the data to a final array, which could then be used for training. All these classes are detected automatically using the OS library and browsing the different folders in the dataset.

When training the model, we quickly realized that the number of images, and the vast number of features would require a lot of computing power to process, especially when we started augmenting the images. The model training using augmented images ended up taking roughly 1 hour and 40 minutes. This was not optimal when also considering that predicting one image would take close to a minute to process.

This realization made us wonder if there could be a better way to extract the features of the image, without losing accuracy. After a lot of experimenting, we found a method that we never expected to work but would later be great. Instead of flattening the image we attempted to sum each column in the image such that we would only have one row, this row was then divided by the height of the image leaving us with the average of all the rows in each column into one single row, this data was then flattened such that the RGB values split up into their own columns. This meant that instead of the data having 150528 features, they would rather have 672 features (224 x 3). For comparison, the graph data generated with this technique (*Image 2*) and the original image (*Image 1*) are shown below.

Et bilde som inneholder tekst, Plottdiagram, diagram

Automatisk generert beskrivelseEt bilde som inneholder føtter, tå, barbeint, person

Automatisk generert beskrivelse

*Image 2*

*Image 1*

We also knew that there are existing libraries out there for augmenting data, however we wanted to have full control over what happened to these images. This is why we decided to create our own augmentation function with the help of OpenCV. The augmentation function is designed to take in a few parameters such as minimum and maximum brightness, contrast and zoom. The images were also flipped vertically and horizontally before the images were augmented. During this process, it was important for us to focus on variability while keeping the features intact.

1. **Measuring performance**

When creating different machine learning models, it is important to keep in mind which kind of way the model’s performance will be measured. In some cases, it might be smart to measure how well the model predicts positives also called recall or sensitivity. In some cases, Specificity is a great measure of how well the model correctly predicts negatives.

However, in the case of this project, we found f1-score to be the best way to measure our performance. The f1-score is a combination of both recall and specificity and will see how many correct predictions the model made from the entire dataset.

F1-Score worked great in our project, since the goal was to find how many images the model managed to correctly predict. The way we gathered data for generating performance measurements was that we created a model, trained it and then we used the models predict function to predict a certain number of images. For the graph data that we generated, we used approximately 2000 images (since each prediction was way faster) and for the entire images we predicted around 200 images (due to the slow prediction times). Then we used scikit-learn’s classification\_report function to compare all the predictions to the actual labels of the same images. The output of this function looks something like the table shown in *Table 1* below:

Et bilde som inneholder tekst, skjermbilde, nummer, Font

Automatisk generert beskrivelse

*Table 1*

As shown in table-1 there is also a column for support, which is how many predictions of the certain class that has been done, in the example we got an accuracy of 0.66 % which was not quite the performance we wanted to get considering the importance of a high accuracy when dealing with patients and disease treatments.

1. **Algorithms applied**

**9.1 KNN**

The K-nearest neighbor algorithm is a machine learning technique used for both classification and regression tasks. The KNN algorithm uses lazy learning to either perform numeric prediction or classification. We have both eager learning and lazy learning. Lazy learning can be referred to as instance-based learning since the learning is based on stored instances of data. While this method is computationally demanding during classification, it can effectively model complex decision spaces that other approaches may not.

The KNN algorithm operates under the assumption that each data point is represented by a tuple in an n-dimensional pattern space, where n corresponds to the number of features in the tuple. Upon encountering a new tuple for classification, the algorithm scans the pattern space to identify the k nearest neighbors to the tuple. For classification tasks, the tuple is assigned the class that is most prevalent among these neighbours. In a numeric prediction task, the output is determined by averaging the numeric values of these neighbor. The value of k, representing the number of neighbor contributing to the similarity assessment, can be optimized through a process of incremental experimentation. The determination of the closest neighbors relies on distance calculations, such as Euclidean distance. Euclidean distance involves calculating the squared difference between corresponding features in two tuples, squaring the result, and adding it to the squared difference of the same calculation for all remaining features. The square root of the sum of these calculations represents the Euclidean distance.

Here is an image that show how KNN works.

Et bilde som inneholder tekst, skjermbilde, diagram, design

Automatisk generert beskrivelse

Image 1:

("K-Nearest Neighbor Algorithm for Machine Learning," n.d .,<https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>)

**Advantages of K-nearest neighbors**

KNN is easy to use because it only needs to find the difference between two points based on their features. This difference can be easily found using formulas like the Euclidean distance or the Manhattan distance (Soni,2020). Furthermore, KNN doesn't need a training phase because the data itself is like a model that's used to make predictions. This makes KNN very fast when it comes to testing new data (Soni,2020). Additionally, since the model doesn't need to be trained, new data can be added anytime. This is because new data won't change the model (Soni,2020).

**Disadvantages of K-nearest neighbors**

While KNN is easy to implement, the efficiency and speed decrease as the dataset size increases (Genesis, 2018). Furthermore, a challenge with KNN is choosing the best number of neighbors to consider when classifying new data point and KNN is easily affected by outliers because it relies on distance when selecting neighbors (Genesis,2018). KNN can't make a general model that's easy to use on new data. It doesn't create rules or charts that can be used to quickly predict new results. Instead, KNN uses all the training data to make its predictions, making it harder to handle large datasets. It also doesn't show how important each piece of information is for making the prediction (MyEducator, n.d).

Overall, the K-nearest neighbors (KNN) algorithm is a simple and powerful machine learning algorithm. It is widely used for classification and regression tasks. It is based on the idea that it is based on the idea that similar data points are likely to have similar labels, and it uses the k nearest neighbors of a new data point to classify or predict its label. The KNN algorithm is well-suited for several applications. It is both simple and efficient and can be a good choice when making a machine learning model. However, it does have some limitations. The algorithm efficiency decreases with larger datasets, and it is sensitive to outliers. Additionally, KNN does not provide any insights into the importance of individual features, which can make it difficult to interpret the results.

**9.2 Convolutional Neural Network (CNN)**

**Convolutional Neural Networks** is a widely used deep learning technique for accurate and advanced machine learning projects. Large companies use CNN’s for creating accurate and efficient deep learning models, some examples include advanced image classification, sign language translation, lip reading algorithms and way more. CNN’s are in many cases used with image data, either frames in a video or standalone images.

The way a Convolutional Neural Network works is by learning patterns within data. With images, the CNN would be able to learn certain patterns and recognize those patterns with the classes. The structure of the network is highly configurable and requires a lot of experience to understand what really happens behind the scenes. The machine learning engineer will define each layer of the network, some layers search for larger patterns within an image, while other layers might search for smaller patterns.

The most common library for such deep learning is TensorFlow, which is developed by Google. An example of how the layers in a convolutional neural network are structured are shown below in (*code example 1*).

There are Conv2D layers, which are smaller portions of the images that will try to learn patterns in that shape, the first Conv2D layer is also called the input layer in our case, where the input shape of the image is specified. The Dropout layers will randomly disable some neurons in the network, this can help with preventing overfitting. The last layer is a dense layer that is specified with 8, which is the number of classes. An activation function, in our case it is the SoftMax activation function which will return a value between zero and one for each class, the class with the highest value is the predicted class.

Et bilde som inneholder tekst, skjermbilde, Font

Automatisk generert beskrivelse *(code example 1)*

When training a CNN, the model will generate training data loss and validation data loss. It then uses these results to adjust the weights in the network to gain a better result in the next epoch. The progress of the training can be plotted with matplotlib to get a visual representation on how the training process went, also to compare the training accuracy versus the validation data accuracy. If the training accuracy ends up being too high, it might be a sign of overfitting, which will result in the model poorly predicting unseen images.

Et bilde som inneholder line, Plottdiagram, tekst, diagram

Automatisk generert beskrivelse

After training our convolutional neural network we achieved a training and validation loss and accuracy as shown in (*graph 1*)

Although the results looked very promising, the model was bias and was almost never able to predict the correct class. Cellulitis was the class that seemed to be bias each time.

*(graph 1)*

To provide an overview of how such a prediction looked, below is a table (Table 2) of all classes and which probabilities each class had to be the prediction image which in this case was an image containing **impetigo:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| cellulitis | impetigo | athlete-foot | nail-fungus | ringworm | cutaneous-larva-migrans | chickenpox | shingles |
| 96,7% | 26,5% | 29,2% | 32,6% | 55,1% | 27,1% | 17,1% | 17,1% |

(Table 2)

We knew that this exam allowed the use of preexisting base models, which could have been done for this exact purpose. ResNet50 is a popular CNN model architecture consisting of 50 layers (<https://datagen.tech/guides/computer-vision/resnet-50/>), which has been proven to work well with this dataset, this can be seen in a notebook linked within the dataset page (<https://www.kaggle.com/code/tantranduc/skin-disease-resnet50-acc-97>). By using the ResNet50 structure, this user was able to get an accuracy of 97%.

The reason we stayed away from this solution was that we wished to create something on our own instead of using a preexisting structure that is known for great results. Experimenting and learning was our main objective of attempting to use convolutional neural networks, and using a preexisting structure would give great results quickly, which was not necessarily our goal.

When stepping into this project, we wanted to learn more about CNN’s even though it is not a part of this course. We learned a lot about how the Convoutional Neural Networks are built and how they operate. However, we had a hard time getting good results with the little knowledge we had at the time. We managed to properly load in the data using the TensorFlow data pipeline, define a model structure and run predictions. The results of the CNN training seemed very promising, however when predicting images, we never got good results. In most cases, the model overfitted and was very bias in one of the classes.

We are sure that with proper knowledge and experience with convolutional neural networks, it would be the best possible model in this projects case. It has been proven to give very good results, even on data with smaller details. The model structure must be engineered and by trial and error, to eventually get the wanted results.

**9.3 Support Vector Machines (SVM)**

**Support Vector Machines** are a collection of different machine learning techniques, which includes support vector regressor, support vector classifier. These methods can be used for different projects, depending on the goal of the machine learning model. In our case, the support vector classifier is used since we aimed to classify different diseases within 8 different classes. The regressor is used in cases where the model must predict a value instead of a class.

The support vector classifier is a powerful machine learning model, which can accurately learn and predict new data. The algorithm is also one of the faster ones when it comes to prediction times. The standard parameters provided with the **SVM** usually gives good results, however it is possible to tune the model to give even better results, for this we decided to use the GridSearchCV functions from SkLearn.

The best way to understand how the support vector classifier works is to see 2d data classified by the classifier. This also allows for easier understanding on how the different kernels works. Kernels are mathematical equations that decides on how the model will classify, the SVC has four different kernels, which are linear, polynomial, rbf and sigmoid. Which one used depends on what is required from the project. For simplicity there is an image illustration below (*image 3*) which displays three of the different kernels.

Et bilde som inneholder tekst, hest, skjermbilde

Automatisk generert beskrivelse

*Image 3 (*[*https://scikit-learn.org/stable/modules/svm.html*](https://scikit-learn.org/stable/modules/svm.html)*)*

these images display how the model would separate the iris dataset, comparing sepal width and sepal length depending on which kernel is being used. This illustration shows how the kernels would classify a 2d dataset and would create a line separating the different classes. If the data has more than two dimensions, the line would be called a hyperplane, which would be like a piece of paper shaped to separate the different classes in multidimensional space.

The way that the support vector classifier generates these hyperplanes is by calculating where there is the most separation between the datapoints, this is quite efficient since the algorithm only must consider the datapoints that are the closest to this separation as shown in the image below (*image 4*).

Et bilde som inneholder skjermbilde, line, diagram, Plottdiagram

Automatisk generert beskrivelse

This separation in the data will be the decision boundaries for the classification, as shown in this image the two orange datapoints and the blue datapoint form what is called a **support vector**, which allows the algorithm to calculate the shape of the hyperplane.

*image 4* (<https://scikit-learn.org/stable/modules/svm.html>)

During our testing and evaluation we found the support vector classifier to do a great job on the skin disease dataset, we gathered a few different results depending on if the training was done on the original dataset, augmented dataset and graph dataset from both of these which we generated within our DataProcessor class using the generate\_xy\_by\_axis() function. To summarize the results, we gathered are shown in the table below (*table 3*)

|  |  |  |
| --- | --- | --- |
| **Dataset** | **Graph Data / Image Data** | **Accuracy** |
| Original | Image Data | 56% |
| Augmented | Image Data | 77% |
| Original | Graph Data | 34% |
| Augmented | Graph Data | 49% |

*table 3*

In this table there is one model that did particularly well, which is the graph data on the augmented dataset. The result was not achieved using the standard svc hyperparameters, these hyperparameters were tuned using GridSearchCV in the SciKit-learn library. This allows for the machine learning engineer to create a dictionary with hyperparameters. This dictionary is then used to generate different folds of parameters that will be trained and evaluated. The hyperparameters that were tuned in our model was C and gamma, by tuning these we increased these results by approximately 20%. The reason we decided to only use GridSearchCV on the graph data was considering the computing time it would have taken if we used the entire image data consisting of around 150 000 parameters, compared to the graph images with 672 parameters.

We found the support vector classifier to work better than expected on image data considering the similarities in the different classes. The simplicity of using the support vector classifier also allows for quick prototyping and great results.

**9.4 Comparison**

1. **Discussion**
2. **Conclusion**

**References**

*Recognizing neglected skin diseases: Who publishes Pictorial Training Guide* (2018) *World Health Organization*. Available at: https://www.who.int/news/item/08-06-2018-recognizing-neglected-skin-diseases-who-publishes-pictorial-training-guide (Accessed: 15 November 2023).

*2023 Global Health Care Sector Outlook: Deloitte*, *Deloitte Belgium*. Available at: https://www2.deloitte.com/be/en/pages/life-sciences-and-healthcare/articles/global-health-care-sector-2023.html (Accessed: 16 November 2023).

Džakula, A., Relić, D. and Michelutti , P. (2022) *Health workforce shortage - doing the right things or doing things right?*, *Croatian medical journal*. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9086817/ (Accessed: 16 November 2023).

*Healthcare Industry* (2023) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Healthcare\_industry (Accessed: 17 November 2023).

*Skin diseases: Types of, symptoms, treatment & prevention* (2021) *Cleveland Clinic*. Available at: https://my.clevelandclinic.org/health/diseases/21573-skin-diseases (Accessed: 24 November 2023).

Higuera, V. (2023) *Skin disorders: Pictures, causes, symptoms, and treatment*, *Healthline*. Available at: https://www.healthline.com/health/skin-disorders#causes (Accessed: 24 November 2023).

Federman , D. *et al.* (2004) *A comparison of diagnosis, evaluation, and treatment of patients with Dermatologic Disorders*, *Journal of the American Academy of Dermatology*. Available at: https://www.sciencedirect.com/science/article/pii/0190962295914498 (Accessed: 27 November 2023).

*Shingles* (2023) *Johns Hopkins Medicine*. Available at: https://www.hopkinsmedicine.org/health/conditions-and-diseases/shingles (Accessed: 27 November 2023).

*Cellulitis: All you need to know* (2022) *Centers for Disease Control and Prevention*. Available at: https://www.cdc.gov/groupastrep/diseases-public/Cellulitis.html (Accessed: 28 November 2023).

Dinulos, J.G.H. (2023) *Cutaneous larva migrans - skin disorders*, *Merck Manuals Consumer Version*. Available at: https://www.merckmanuals.com/home/skin-disorders/parasitic-skin-infections/cutaneous-larva-migrans (Accessed: 28 November 2023).

*Research guides @ fordham: Copyright resources: Creative commons & public domain* (2023) *Creative Commons & Public Domain - Copyright Resources - Research Guides @ Fordham at Fordham University Libraries*. Available at: https://fordham.libguides.com/copyright/Licensing (Accessed: 29 November 2023).

“K-Nearest Neighbor Algorithm for Machine Learning." n.d. <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>

Soni, A. (2020) *Advantages and disadvantages of KNN*, *Medium*. Available at: https://medium.com/@anuuz.soni/advantages-and-disadvantages-of-knn-ee06599b9336 (Accessed: 05 December 2023).

Genesis (2018) *Pros and cons of K-nearest neighbors, From The GENESIS*. Available at: https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/ (Accessed: 06 December 2023).

Myeducator (no date) *11.4advantages and disadvantages of KNN, MyEducator*. Available at: https://app.myeducator.com/reader/web/1421a/11/q07a0/ (Accessed: 06 December 2023).