Internship Report

Gabriel WARDE

Artifact Detection in Immunofluorescence Microscopy Slides

U1286 INFINITE Laboratory

Supervisors: Clément CHAUVET - Dr. Adán JOSÉ-GARCÍA - Pr. Vincent SOBANSKI

Address: UFR3S - Faculty of Medicine – Research center, 5th floor Center, Place Verdun, 59045 Lille Cedex

Lille, France

August 14, 2024

Contents

1 Introduction

This internship involved detecting artifacts in immunofluorescence microscopy slide images. During the internship, I worked with the ENDOMIC (ENDOtyper les Maladies Inflammatoires Chroniques) research team, led by Pr. Vincent Sobanski, whose work can be found at https://endomic.github.io/. The research was conducted at the INFINITE (Institute for Translational Research in Inflammation) U1286 laboratory, located on the 5th floor of the UFR3S - Médecine - Pôle Recherche, Place Verdun, 59045 Lille Cedex.

1.1 Artificial Intelligence in Medicine

In the rapidly evolving landscape of modern healthcare, the integration of advanced technologies has become indispensable. Among these, data science (DS), machine learning (ML), and deep learning (DL) stand out as pivotal tools that are reshaping the medical field. These technologies are not just additional tools but have become essential components that drive innovations in diagnostics, treatment, and patient care.

The growing volumes of medical data, from electronic health records to imaging data, necessitate sophisticated analytical methods to extract meaningful insights. Data science offers robust frameworks for managing, analyzing, and interpreting this data, enabling healthcare professionals to make evidence-based decisions. Machine learning algorithms enhance this process by identifying patterns and predicting outcomes with unprecedented accuracy. Deep learning, a subset of ML, further amplifies these capabilities by leveraging neural networks to perform complex tasks such as image recognition, anomaly detection, and natural language processing, which are crucial for medical diagnosis and personalized treatment plans.

In hospitals, these technologies could optimize operations, improve patient outcomes, and enhance the efficiency of healthcare delivery. For doctors, they provide powerful tools to aid in diagnosis, monitor patient progress, and customize treatment strategies based on predictive analytics. The convergence of data science, ML, and DL in healthcare not only enhances the capabilities of medical professionals but also opens the door for innovations that can lead to early detection of diseases, precision medicine, and improved patient care protocols. From Chiranjib Chakraborty, *et al.,* 2024 [3]

1.2 What Is Deep Learning

Deep learning is a specialized area within the broader field of machine learning, distinguished by its use of artificial neural networks to imitate the human brain's capacity for learning and decision-making from data. These neural networks, designed to mimic the interconnected structure of neurons in the brain, enable deep learning models to process and analyze vast amounts of data, identifying patterns and making predictions.

Unlike traditional machine learning algorithms, which rely on humans to manually extract and select features from raw data, deep learning models automatically discover and learn representations of data through multiple layers of abstraction. This layered approach allows the models to build increasingly complex and high-level features from simpler ones, improving their performance in tasks such as image and speech recognition, natural language processing, and other complex problem-solving applications. From Eugene Dorfman, 2022 [5].

During this internship, our focus was on using deep learning with an Anti-Nuclear Antibodies (ANA) image dataset obtained from the CHU Lille. Below is a concise explanation of ANAs.

1.3 AntiNuclear Antibodies (ANAs)

1.3.1 Definition

Anti-nuclear antibodies (ANAs) are autoantibodies produced by the immune system that target the body's own cell components. These antibodies are associated with a range of autoimmune diseases, such as Systemic Lupus Erythematosus, Rheumatoid Arthritis, Sjogren's Syndrome, and Scleroderma.

1.3.2 Nature of ANAs

These antibodies target substances found in the nucleus of cells, and their presence often indicates an abnormal immune response. Clinicians utilize ANA tests to help diagnose conditions such as systemic lupus erythematosus, rheumatoid arthritis, and systemic sclerosis. Antinuclear antibodies (ANAs) serve as crucial biomarkers in the diagnosis and prognosis of various autoimmune diseases. The detection of ANAs thus plays a pivotal role in both identifying autoimmune diseases and predicting their progression.
By measuring the level and type of ANAs in a patient's blood, healthcare providers can gain insights into the likelihood and severity of an autoimmune disorder, aiding in the formulation of customized treatment plans and ongoing disease management. From Lisa Carnago *et al.*, 2023 [14].

1.3.3 Visualization technique

At the Centre Hospitalier Universitaire de Lille (CHU Lille), antinuclear antibodies (ANAs) are routinely tested, with approximately 20,000 images analyzed annually using an indirect immunofluorescence technique.

In this process, standardized cells are placed on a slide, and the patient's serum, which contains their antibodies, is added to these cells. Next, fluorescent antibodies are introduced, which specifically bind to the patient's antibodies if present. This combination results in the formation of visible complexes.

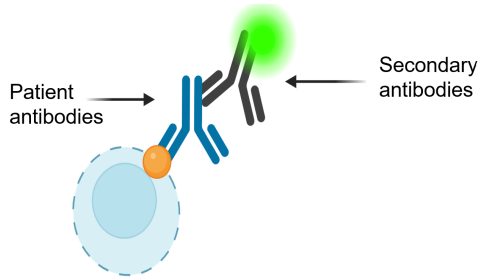


Figure 2: Close up image of this process

The slide is then examined under a microscope, where the presence and patterns of fluorescence reveal the presence of antinuclear antibodies (ANAs). Indirect immunofluorescence identifies different ANA types by showing distinct patterns of fluorescence on cells, reflecting the specific antigens the antibodies target. From Chan *et al.*, 2015 [4].

This method is essential for diagnosing and monitoring autoimmune diseases, as it provides detailed insights into the patient's immune system activity.

1.3.4 Different ANAs Classes

There are various classes of ANAs, each targeting different nuclear antigens. These classes include, for instance, anti-double-stranded DNA (dsDNA), anti-Smith (Sm), anti-ribonucleoprotein (RNP), anti-histone, and anti-centromere antibodies. The presence of specific classes of ANAs can aid in the

diagnosis and classification of different autoimmune diseases. Understanding the diverse classes of ANAs is crucial for clinicians in diagnosing and managing autoimmune conditions effectively. There are 30 different ANAs classes, and it is a hard task to classify them as their classification is visual.

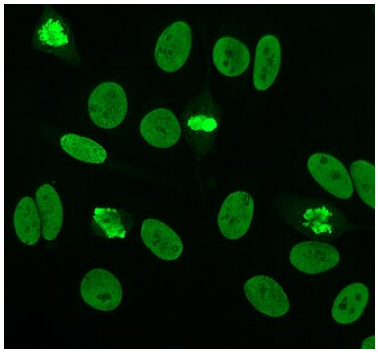


Figure 3: Close up image of ANAs.
Source: *Antinuclear antibody - Wikipedia,* 2024 [19]

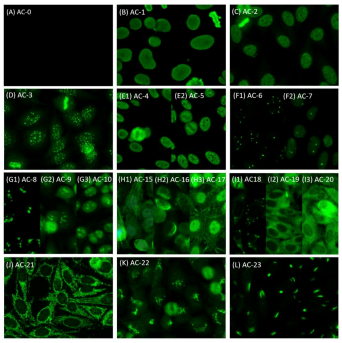


Figure 4: Example of different ANAs classes.
From: Yi-Da Wu *et al.,* 2021 [20]

Some classes, such as AC-8 and AC-9, are challenging to differentiate because they appear very similar, making it difficult even for experts to distinguish between them. Here's an example:

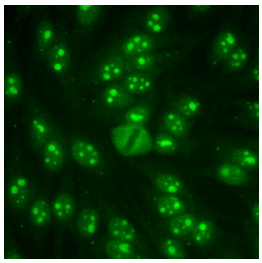


Figure 5: ANAs image of class AC-8.
Source: *anapatterns.org* [9]

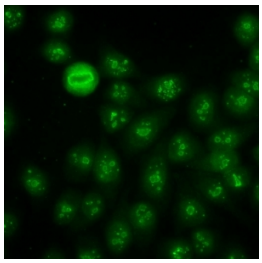


Figure 6: ANAs image of class AC-9.
Source: *anapatterns.org* [9]

1.3.5 Deep Learning Classification Model

The ENDOMIC research team attempted to develop a deep learning classification model to categorize these images into the 30 different classes of ANA. However, for some ANA classes, the number of available images was low due to the rarity of these antibodies in real-life cases. To address this issue, the team employed generative models such as Generative Adversarial Networks (GANs) to generate more images.

After generating ANAs images with GANs, the team encountered issues with the quality of the generated images. Specifically, around 20-30% of the generated images displayed unusual anomalies.

Upon examining the large dataset used to train these models, we discovered several abnormal images (detailed in section 1.4). These images represented around 5% of the original dataset, yet they were far over-represented in the images generated by GANs.

The objective of my project is to test and evaluate different CNN models that can self-learn to classify images as good (clear and defect-free), anomalous, or blurry with the highest possible accuracy. By identifying images as anomalous or blurry, the project aims to enable the team to investigate how these types of images affect both image classification and generation processes.
This approach ensures that only high-quality images are used for training and further analysis, enhancing the overall performance and reliability of the deep learning classification model.

1.4 ANAs Anomaly detection

As mentioned earlier, problems can sometimes occur in microscopy imaging because of different reasons, including mistakes in the imaging procedure or due to limitations of the imaging technology. These problems might manifest as artifacts on images, like bubbles, debris on the slides, or abnormal fluorescence patterns. Additionally, technical challenges like incorrect focusing or movement during image capture can cause blurry images, reducing the quality of the data obtained.

To tackle this challenge, my tutors and I agreed to formulate a classification system, using Convolutional Neural Networks, comprising three distinct classes: 0 for Anomaly images, 1 for Blurry images, and 2 for Good (clear and standard) images. By categorizing the images into these classes, we aim to enhance the robustness and accuracy of the CNN models by enabling them to differentiate between these different types of images and isolate the unwanted ones. Below, I present examples illustrating each of these classes to provide more clarity.

1.4.1 Standard Images

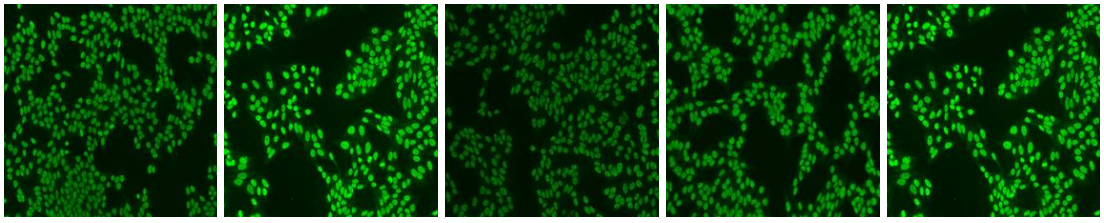


Figure 7: Example of standard/normal ANAs images

1.4.2 Anomaly Images

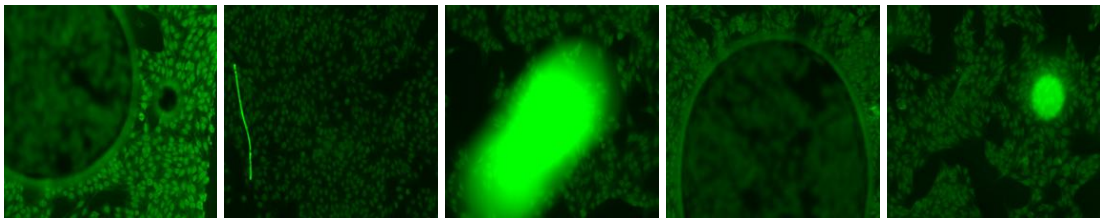


Figure 8: Example of anomalous ANAs images

1.4.3 Blurry Images

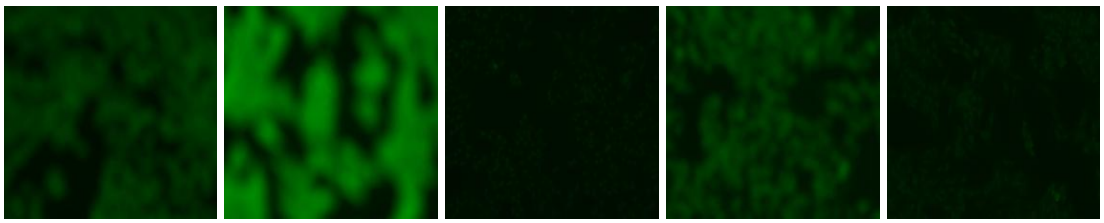


Figure 9: Example of blurry ANAs images

2 Background

In this section, I will provide a brief overview of key deep learning concepts before diving into the specific CNN model architectures selected during the internship. These chosen models represent state-of-the-art advancements that have demonstrated high performance in ImageNet competitions.

2.1 Neural Networks

2.1.1 Definition:

Neural networks consist of interconnected layers of neurons, where each neuron processes input data by applying weights and biases. These weights and biases are adjusted during training using optimization algorithms, aiming to minimize prediction errors (loss). In neural networks, neurons in the first layers typically learn to detect simple features such as edges and textures, while neurons in the last layers learn to recognize complex patterns and specific objects.

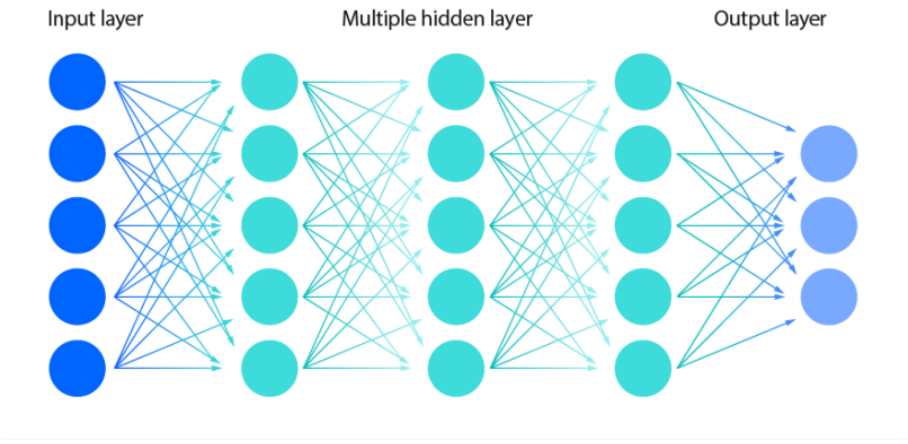


Figure 10: Simple Neural Network Architecture

The training process involves two key phases: forward propagation and backward propagation. Both are essential for training a neural network effectively, as they work together to enable the network to learn from data and optimize its performance.

2.1.2 Forward Propagation

Forward propagation is the process through which input data is passed through a neural network to generate predictions. It begins when data is introduced to the input layer of the network. As the data moves through the network, each neuron computes a weighted sum of its inputs, adds a bias term, and then applies an **activation function** to introduce non-linearity, which allows the network to model complex relationships in the data. Specifically, for a given neuron, the computation of the **activation function is defined as**:

$$z = \sum_i (w_i \cdot x_i) + b$$

where:

- w_i represents the weights of the neuron,
- x_i represents the input values,
- b is the bias term.

The activation function f is then applied to z to produce the neuron's output a:

$$a = f(z)$$

where f can be any activation function, such as ReLU, sigmoid, or tanh.

The different activation functions are:

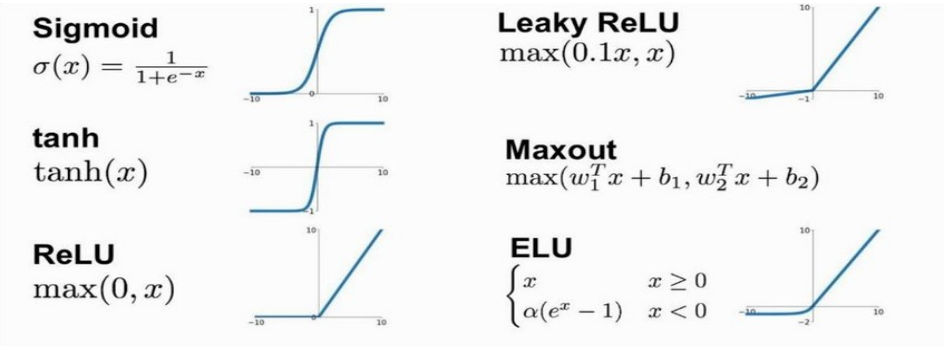


Figure 11: Various types of activation functions. From Rahul Jayawardana *et al.,* 2021 [10]

This process continues layer by layer, with each subsequent layer transforming the data further by extracting increasingly complex features. The final output layer produces the network's prediction or classification based on the processed information. Essentially, forward propagation involves feeding the input data, calculating weighted sums and biases, applying activation functions, and propagating the data through each layer until the network generates an output.

Afterward, the **loss** is computed by comparing the network's predictions with the true labels. The **loss function for regression tasks** is the Mean Squared Error (MSE), defined as:

$$L(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

where:

- L: Loss function,
- $\hat{y}_i$: Predicted value for the i-th example,
- y_i: Actual target value for the i-th example,
- n: Number of examples.

2.1.3 Backward Propagation

Backward propagation, or backpropagation, is a critical technique used to adjust the neural network's weights and biases to minimize the loss calculated during forward propagation. In backpropagation, the gradient of this loss function is then calculated with respect to the network's output. This gradient of error is propagated backward through the network, starting from the output layer and moving towards the input layer.

As the error signal moves backward, gradients of the loss with respect to the activations, weighted

sums, weights, and biases are computed using the chain rule. For example, the gradient of the loss with respect to the weights w is:

$$\frac{\partial \text{Loss}}{\partial w}$$

These gradients are then utilized to update the weights and biases through optimization algorithms such as gradient descent. The weight update is performed using a formula like:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial \text{Loss}}{\partial \theta}$$

where:

- θ: Model parameters (weights and biases),
- η: Learning rate, which controls the step size in updating the parameters,
- $\frac{\partial \text{Loss}}{\partial \theta}$: Gradient of the loss function with respect to the model parameters,
- t: Iteration index, representing the current step in the optimization process.

This iterative process of computing gradients and updating parameters continues over multiple epochs. By minimizing the loss through this process, the network gradually refines its parameters, improving its accuracy and ability to make accurate predictions over time. This ensures that the model learns effectively from the training data and generalizes well to new and unseen data.

2.1.4 Applications

Deep learning finds application across diverse domains, including computer vision (e.g., object detection, image segmentation), natural language processing (e.g., text generation, sentiment analysis), speech recognition, autonomous vehicles, healthcare (e.g., medical imaging analysis, disease diagnosis), finance (e.g., fraud detection, algorithmic trading), and recommendation systems. These applications leverage deep learning's ability to handle large amounts of data, extract meaningful features, and make accurate predictions, driving advancements in technology and enhancing decision-making processes.

2.1.5 Computational Demands & Tools In Deep Learning

Deep learning demands significant computing power. High-performance GPUs are particularly well-suited for this task, as they can perform extensive calculations across multiple cores with ample memory. Distributed cloud computing can also be beneficial. Such computational capabilities are essential for training deep learning algorithms. However, managing multiple GPUs on-site can place a heavy burden on internal resources and can be very expensive to scale. In terms of software, most deep learning applications are developed using one of three frameworks: Keras, PyTorch, or TensorFlow, cited by IBM, 2024 [8].

2.1.6 Pre-Trained Models

Pre-trained models are neural network models that have been previously trained on large datasets, allowing them to serve as a solid foundation for new tasks. These models typically leverage huge datasets like ImageNet, which contains over 14 million images across 1,000 categories. The process involves training a model on this huge dataset and then using the learned features for new tasks. To customize these models for specific applications, certain layers are often "frozen" to retain the pre-learned features, while the remaining layers are fine-tuned with new data. Pre-trained models are used because they have already learned useful features from a large dataset, which allows them to save time and computational resources when applied to new tasks, often leading to better performance and faster results.

During the internship, we utilized several pre-trained models using Keras, including DenseNet-121, VGG-16, InceptionV3, and ResNet-50.

CNNs are specifically suited for visual recognition tasks, leveraging convolutional layers to hierarchically learn features from images. Here are the CNN models used during the internship journey.

2.2 LeNet-5 (1998)

2.2.1 Introduction:

LeNet-5 is a seminal convolutional neural network architecture developed by Yann LeCun, *et al., 1998* [13]. It was designed primarily for handwritten digit recognition tasks and played a pivotal role in popularizing convolutional neural networks. It contains 61,706 parameters.

2.2.2 Architecture:

1. Input Layer: LeNet-5 accepts grayscale images of size 32x32 pixels as input.
2. Convolutional Layers: The network comprises two convolutional layers, each followed by a subsampling (average pooling) layer.
The 1st conv layer: Filters: 6, Kernel Size: 5x5, Stride: 1
The 2nd conv layer: Filters: 16, Kernel Size: 5x5, Stride: 1
3. Pooling Layers: It uses average pooling, and all pooling layers are 2x2 with a stride of 2.
4. Fully Connected Layers: Following the convolutional layers, LeNet-5 includes 3 fully connected layers with 120, 84, and 10 neurons, respectively.
5. Activation Function: A tanh activation function is applied after each convolutional and fully connected layer, except the output layer.
6. Output Layer: The final layer produces class scores for the input image using softmax activation.

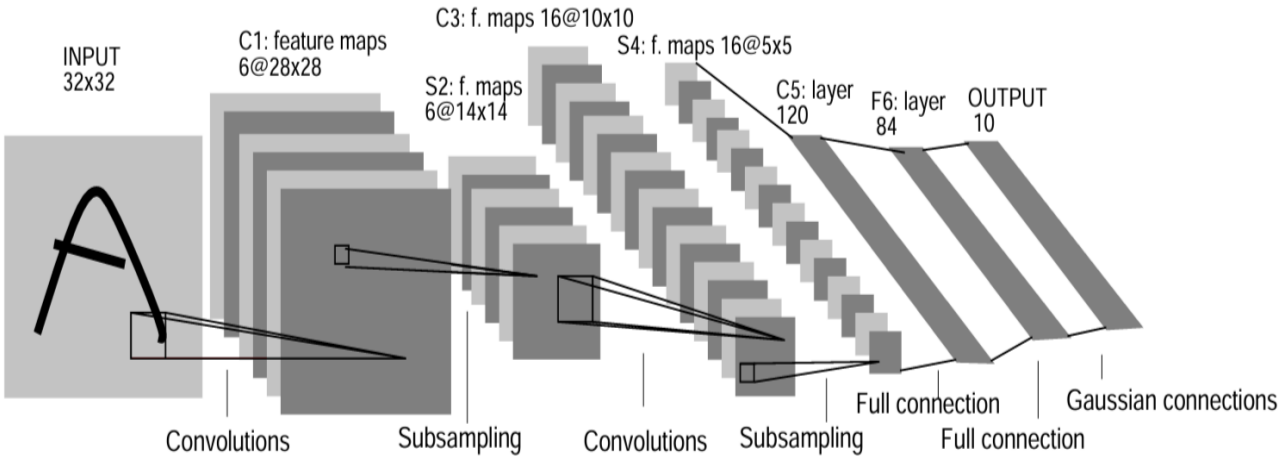


Figure 12: Architecture of LeNet-5. From the original paper [13]

2.2.3 Suitability:

LeNet-5 is best suited for simple image recognition tasks, particularly handwritten digit recognition. It may also serve as a foundational model for understanding the principles of convolutional neural networks.

2.3 AlexNet (2012)

2.3.1 Introduction:

AlexNet, a convolutional neural network (CNN) designed by Alex Krizhevsky, *et al.,* 2012 [12], won the 2012 ImageNet Large Scale Visual Recognition Challenge, revolutionizing computer vision with its deep learning architecture and significant performance improvements. AlexNet contains approximately 60 million parameters.

2.3.2 Architecture:

AlexNet contains 8 layers, including 5 convolutional layers with max pooling, 3 fully connected layers, ReLU activation, dropout, and softmax classifier.

1. Input Layer: RGB images of size 227x227 pixels.
2. Convolutional Layers: AlexNet comprises 5 convolutional layers.
The 1st conv layer: Filters: 96, Kernel Size: 11x11, Stride: 4, Pad: 0
The 2nd conv layer: Filters: 256, Kernel Size: 5x5, Stride: 1, Pad: 2
The 3rd conv layer: Filters: 384, Kernel Size: 3x3, Stride: 1, Pad: 1
The 4th conv layer: Filters: 384, Kernel Size: 3x3, Stride: 1, Pad: 1
The 5th conv layer: Filters: 256, Kernel Size: 3x3, Stride: 1, Pad: 1
3. MaxPooling: It is applied after the 1st, 2nd, and 5th convolutional layers. All max-pooling layers have: Pool Size: 3x3, Stride: 2, Padding: 0.
4. Fully Connected Layers: Three fully connected layers follow the convolutional layers, with 4096, 4096, and 1000 neurons, respectively.
5. ReLU Activation: Applied after each convolutional and fully connected layer.
6. Local Response Normalization (LRN): Applied after the first and second convolutional layers.
7. Dropout Regularization: Applied to the first two fully connected layers.
8. Output Layer: Produces class probabilities for the input image using softmax activation.

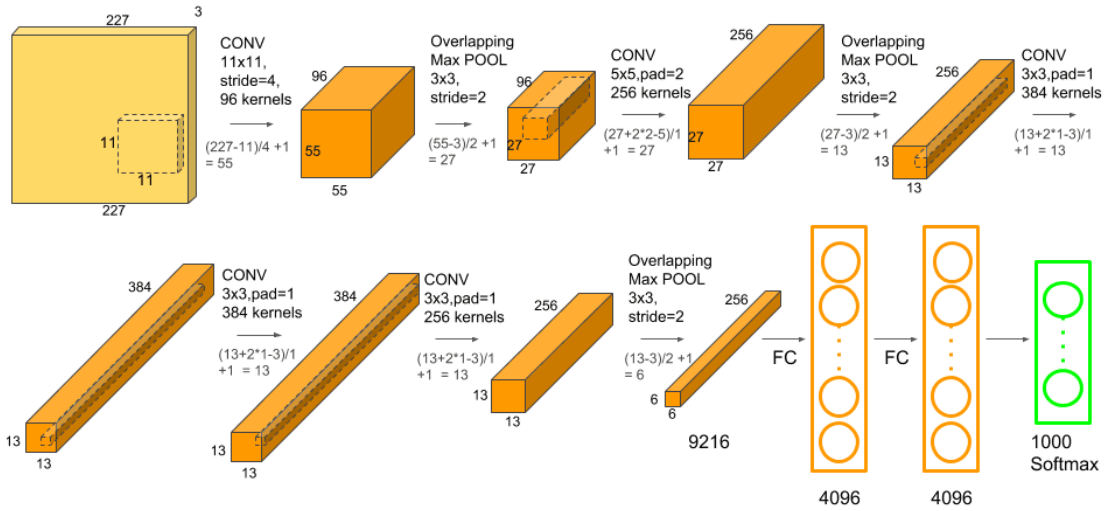


Figure 13: Architecture of AlexNet. From Abhijeet Pujara, 2020 [15]

2.3.3 Suitability:

It is ideal for large-scale image classification tasks requiring high accuracy and extensive datasets.

2.4 ZFNet (2013)

2.4.1 Introduction:

ZFNet, introduced in 2013 by Matthew D Zeiler and Rob Fergus [21], emerged as a significant contender in the ImageNet Large Scale Visual Recognition Challenge, showcasing advancements in computer vision. ZFNet is essentially an enhanced version of AlexNet with more computations and optimized layered configurations, leading to improved performance. With an increased number of filters, the network becomes larger, requiring more parameters, memory, and computational power. ZFNet contains approximately 62.3 million parameters.

2.4.2 Architecture:

ZFNet comprises 8 layers, featuring 5 convolutional layers followed by max-pooling, and 3 fully connected layers with ReLU activation, dropout, and softmax classifier.

1. Input Layer: Accepts RGB images of size 224x224 pixels.
2. Convolutional Layers: ZFNet incorporates 5 convolutional layers.
1st Conv Layer: 96 filters, 7x7 kernel size, stride of 2, padding of 1
2nd Conv Layer: 256 filters, 5x5 kernel size, stride of 2, padding of 1
3rd Conv Layer: 384 filters, 3x3 kernel size, stride of 1, padding of 1
4th Conv Layer: 384 filters, 3x3 kernel size, stride of 1, padding of 1
5th Conv Layer: 256 filters, 3x3 kernel size, stride of 1, padding of 1
3. MaxPooling Layers: It is applied after the 1st, 2nd, and 5th convolutional layers. They have a pool size of 3x3, stride of 2, and no padding.
4. Fully Connected Layers: Three fully connected layers follow the convolutional layers, with 4096, 4096, and 1000 neurons, respectively.
5. ReLU Activation: Applied after each convolutional and fully connected layer.
6. Local Response Normalization (LRN): Applied after the first and second convolutional layers.
7. Dropout Regularization: Applied to the first two fully connected layers.
8. Output Layer: Produces class probabilities for 1000 classes using softmax activation.

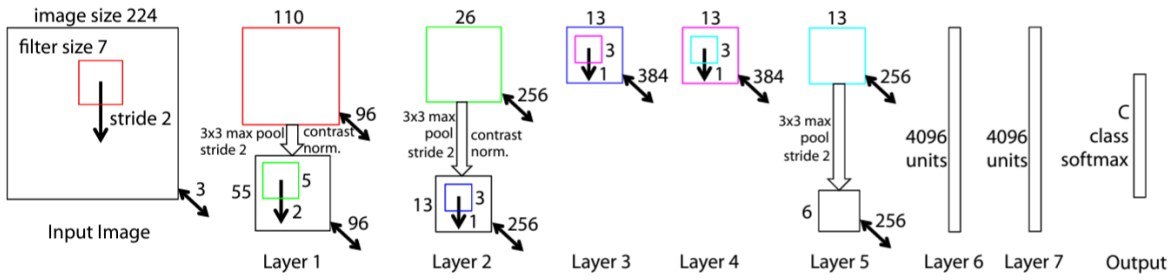


Figure 14: Schema of the ZFNet architecture. From the original paper [21]

2.4.3 Suitability:

ZFNet is well-suited for tasks demanding accurate image classification on large datasets.

2.5 VGGNet (2014)

2.5.1 Introduction:

VGGNet, proposed in 2014 by Karen Simonyan and Andrew Zisserman [16], made significant strides in image recognition tasks, particularly in the ImageNet Large Scale Visual Recognition Challenge. VGGNet is characterized by its deep architecture, consisting of 16 or 19 layers, with a uniform structure of convolutional layers followed by max-pooling layers. VGG-16 contains approximately 138 million parameters, while VGG-19 contains around 143 million parameters.

2.5.2 Architecture:

1. Input Layer: RGB images of size 224x224 pixels.
2. Convolutional Layers: VGGNet features a series of convolutional layers organized into blocks, each with a 3x3 kernel, stride of 1, and padding of 1. The number of filters increases with depth. VGG-16 has 13 convolutional layers and 3 fully connected layers, while VGG-19 has 16 convolutional layers and 3 fully connected layers.
3. MaxPooling: Applied after each convolutional block. All max-pooling layers have a pool size of 2x2 and a stride of 2. Pooling layers are typically applied after every 2 or 3 convolutional layers within a block.
4. Fully Connected Layers: Three fully connected layers follow the convolutional layers. The first two fully connected layers typically have 4096 neurons each, and the third contains 1000 neurons.
5. ReLU Activation: Applied after each convolutional and fully connected layer.
6. Dropout Regularization: Applied to the fully connected layers to prevent overfitting.
7. Output Layer: Produces class probabilities for the input image using softmax activation.

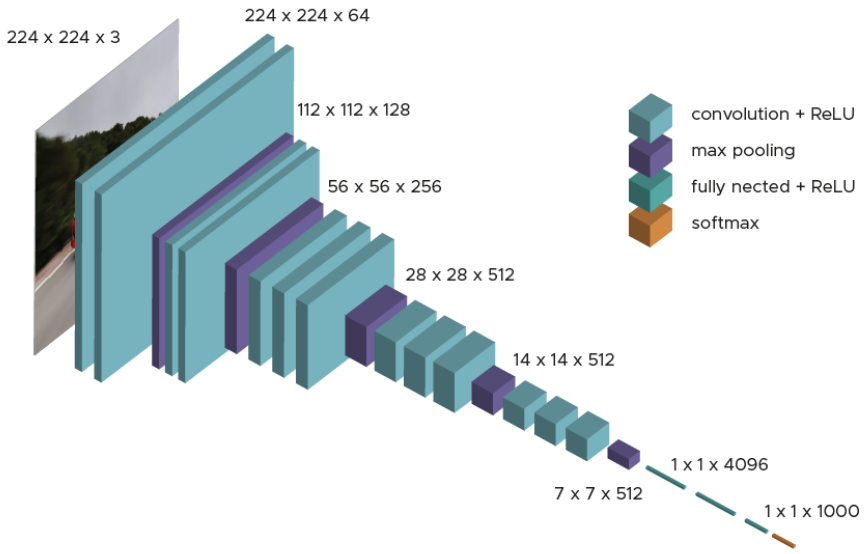


Figure 15: Schema of the VGG-16 architecture. From Raphael Kassel, 2021 [11].

2.5.3 Suitability:

The VGG architectures are well-suited for image classification tasks with high accuracy and robust feature extraction. However, they may be less efficient compared to more recent models due to their large number of parameters and computational requirements.

2.6 ResNet (2015)

2.6.1 Introduction:

ResNet (Residual Network), is a groundbreaking convolutional neural network architecture developed by Kaiming He, *et al.,* 2015 [6]. It introduced the concept of residual learning, which addressed the degradation problem encountered in deep neural networks by enabling the training of extremely deep networks effectively. The number of parameters varies based on the depth of the ResNet model. ResNet-18 has approximately 11 million parameters, whereas ResNet-50 has approximately 25 million parameters.

2.6.2 Architecture:

1. Input Layer: ResNet accepts RGB images of size 224x224 as input.
2. Convolutional Layers: The network comprises several convolutional layers organized into blocks. Each residual block typically contains two 3x3 convolutional layers with batch normalization and ReLU activation. Additionally, a 1x1 convolutional layer is sometimes used in the block to adjust the dimensions of the input for the skip connection.
3. Pooling Layers: ResNet uses max pooling right after the initial convolutional layer to reduce spatial dimensions. Global average pooling is applied at the end of the network to further reduce spatial dimensions before the final classification layer.
4. Fully Connected Layers: Following the convolutional layers and global average pooling, ResNet includes a single fully connected layer to produce class probabilities for the input image.
5. Activation Function: ReLU activation function is applied after each convolutional layer within the residual blocks.
6. Batch Normalization: Batch normalization is applied before the activation function in each residual block to stabilize and accelerate training.
7. Skip Connections (Identity Mapping): Skip connections allow gradients to flow directly through the network, mitigating the vanishing gradient problem and enabling the training of very deep networks.
8. Output Layer: That layer produces class probabilities for input images using softmax activation.

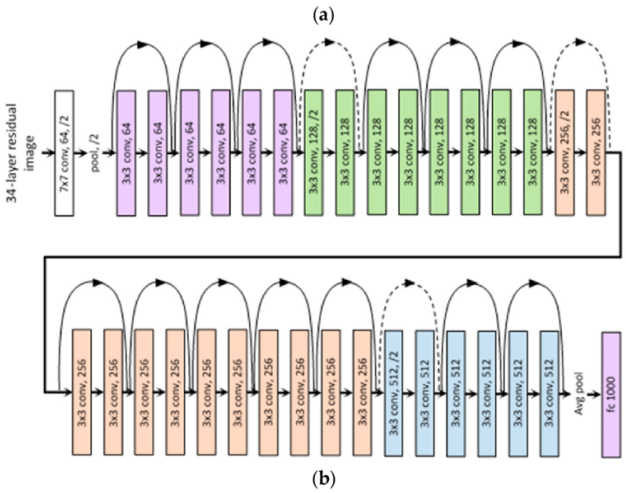


Figure 16: Architecture of a ResNet-34. From Siddhesh Bangar, 2022 [1]

2.6.3 Suitability:

ResNet is well-suited for various computer vision tasks, including image classification, object detection, and semantic segmentation. It is particularly effective in training very deep neural networks due to its residual connections, which help address the vanishing gradient problem.

2.7 InceptionV3 (2015)

2.7.1 Introduction:

InceptionV3 is an advanced version of the Inception architecture, developed by Christian Szegedy, *et al.,* 2015 [17]. It builds upon the success of earlier Inception versions by incorporating several new techniques to improve performance and efficiency. InceptionV3 achieved state-of-the-art performance on the ImageNet dataset while maintaining computational efficiency. It introduced new concepts such as factorized convolutions and aggressive regularization. The number of parameters is around 23.9 millions, significantly reduced compared to its predecessors while maintaining high performance.

2.7.2 Architecture:

1. Input Layer: InceptionV3 accepts RGB images of size 299x299 pixels as input.
2. Inception Modules: The hallmark of InceptionV3 is its inception modules, which are building blocks consisting of multiple parallel convolutional operations. Each inception module typically contains 1x1, 3x3, and 5x5 convolutions, as well as max pooling operations. These modules are concatenated along the depth dimension, allowing the network to capture features at different scales. InceptionV3 refines these modules with factorized convolutions and asymmetric convolutions to reduce computational complexity.
3. Factorized Convolutions: InceptionV3 employs factorized convolutions, breaking down larger convolutions into smaller ones (e.g., a 5x5 convolution into two 3x3 convolutions) to reduce the number of parameters and computational cost.
4. Pooling Layers: InceptionV3 uses both max pooling and average pooling layers to reduce spatial dimensions in various stages of the network.
5. Global Average Pooling: InceptionV3 replaces traditional fully connected layers with global average pooling, which computes the average of each feature map across spatial dimensions.
6. Activation Function: ReLU is applied after each convolutional layer within the inception modules.
7. Batch Normalization: Batch normalization is applied before the activation function in each convolutional layer to improve training stability and accelerate convergence.
8. Auxiliary Classifiers: It includes auxiliary classifiers in intermediate layers to provide additional gradients during training, helping to address the vanishing gradient problem and improve convergence.
9. Dropout: Dropout is not commonly used in InceptionV3; the emphasis is more on batch normalization and global average pooling.
10. Output Layer: This layer does class probabilities for the input image using softmax activation.

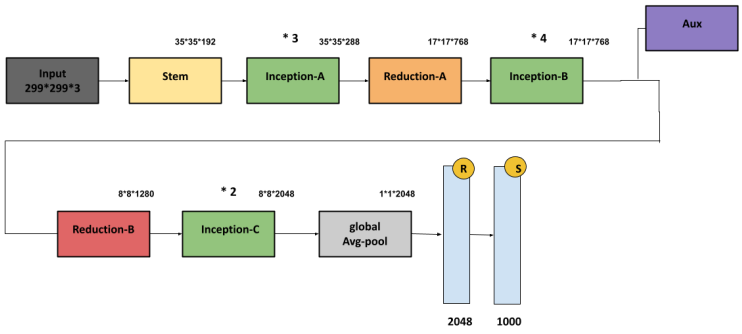


Figure 17: Architecture of InceptionV3. From Anas Brital, 2021 [2]

2.7.3 Suitability:

It is well-suited for tasks requiring high accuracy in image classification while maintaining computational efficiency. It is particularly effective for applications where computational resources are limited.

2.8 DenseNet (2017)

2.8.1 Introduction:

DenseNet, short for Dense Convolutional Network, is a convolutional neural network architecture proposed by Gao Huang, *et al.,* 2017 [7]. It introduces the concept of dense connectivity, where each layer receives feature maps from all preceding layers, leading to densely connected blocks. DenseNet aims to alleviate the vanishing gradient problem, encourage feature reuse, and promote model compactness. There are many variants such as DenseNet-121, DenseNet-169, DenseNet-201, and DenseNet-264, where the numbers correspond to the number of layers in each variant. DenseNet-121 contains approximately 8 million parameters.

2.8.2 Architecture:

1. Input Layer: DenseNet accepts RGB images of size 224x224 pixels.
2. Initial Convolution and Pooling: The network begins with a 7x7 convolutional layer with 64 filters and a stride of 2, followed by a 3x3 max pooling layer with a stride of 2. This initial stage reduces the spatial dimensions of the input image.
3. Dense Blocks: The key building blocks in DenseNet are dense blocks, where each layer is connected to every other layer in a feed-forward manner. Within each dense block, feature maps from all preceding layers are concatenated together, forming dense connections that facilitate feature reuse and information flow.
4. Bottleneck Layers: DenseNet uses bottleneck layers within dense blocks, consisting of a 1x1 convolution to reduce feature maps, followed by a 3x3 convolution to process them. This design reduces computational cost while maintaining effective feature extraction.
5. Transition Layers: These layers are used between dense blocks to reduce the number of feature maps and control feature dimension growth. They typically include 1x1 convolutions followed by average pooling.
6. Global Average Pooling: Similar to other modern architectures, DenseNet replaces traditional fully connected layers with global average pooling to reduce the number of parameters and improve generalization.
8. Output Layer: After global average pooling, it has a fully connected layer that produces the final output, which is then passed through a softmax activation function to generate class probabilities.

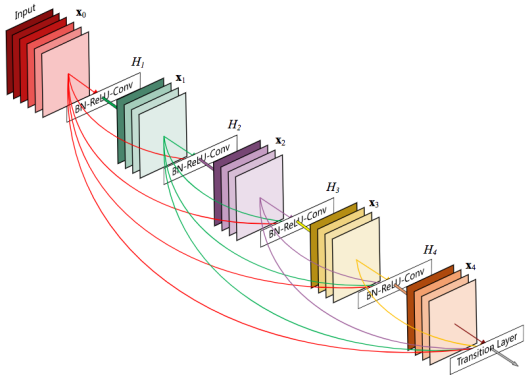


Figure 18: Architecture of DenseNet. From the original paper [7]

2.8.3 Suitability:

DenseNet is well-suited for various computer vision tasks, including image classification, object detection, and semantic segmentation. It is particularly effective in scenarios where feature reuse and information flow are crucial for performance, such as fine-grained classification and medical image analysis.

2.9 Overview of selected convolutional neural network (CNN) models

Here is a table that provides an overview of these selected convolutional neural network (CNN) models. It includes details on the release date, input size, number of parameters, various filter sizes, and whether or not skip connections are used. Additionally, it specifies whether each model was pretrained according to our choices. Find the table below.

Model	LeNet-5	AlexNet	ZFNet	VGG-16	ResNet-18	ResNet-50	InceptionV3	DenseNet-121
Year	1998	2012	2013	2014	2015	2015	2015	2017
Input size	32x32	227x227	224x224	224x224	224x224	224x224	299x299	224x224
Parameters	60,000	60M	62.3M	138M	11M	25M	23.9M	8M
Filter size	5x5	11x11, 5x5, 3x3	7x7, 5x5, 3x3	3x3	7x7, 3x3	7x7, 1x1, 3x3	1x1, 3x3, 5x5	7x7, 1x1, 3x3
Skip connections	No	No	No	No	Yes	Yes	No	Yes
Pre-trained	No	No	No	Yes	No	Yes	Yes	Yes

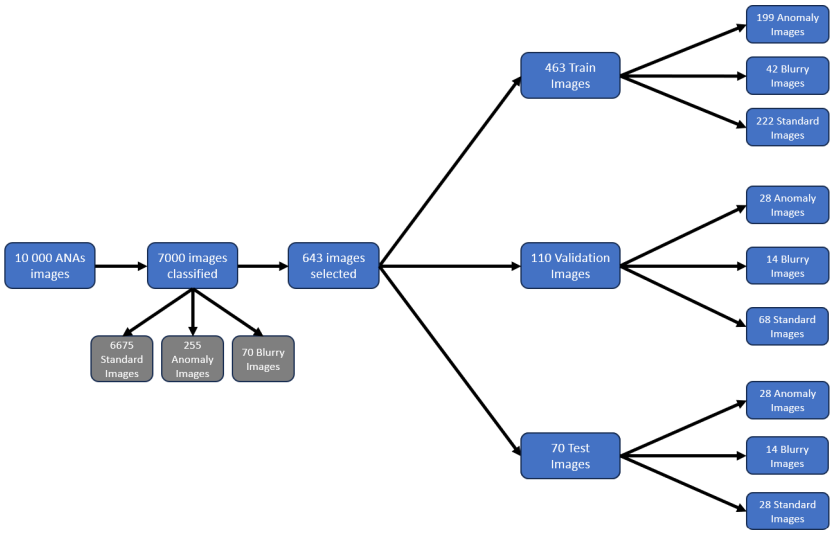
Table 1: Table showing details of the selected convolutional neural network (CNN) models

3 Methodology

In this section, I will outline the methodology used for preprocessing and preparing the data essential for running the CNN models. This includes steps taken to handle class imbalances, augment, and resize the data to ensure optimal performance during model training and evaluation. Additionally, I will incorporate some good practice techniques commonly used in Deep Learning.

3.1 Data Handling

We were provided by the Centre Hospitalier Universitaire de Lille (CHU de Lille), a dataset, containing 10,000 unlabeled ANA images in very high resolution. To make them suitable for our deep learning tasks, we resized the images to a manageable size of 224x224 pixels. This adjustment was necessary because working with images of resolutions ranging from 2000x2000 to 3000x3000 would have been impractical for our computational resources and modeling requirements. Resizing to 224x224 pixels allowed us to maintain essential details while optimizing processing efficiency and model performance in our analysis.



After spending several days labeling the images based on specific criteria, we decided to stop classifying them at 7,000 images. This decision was made because the process was time-consuming and we did not observe significant differences in the content of the images. Out of the 7,000 images, 6,675 were categorized as Good/Standard Images, 255 as Anomaly Images, and 70 as Blurry Images.

Next, the dataset was partitioned into three distinct subsets: the Train set, Validation set, and Test set.

We then decided to train the neural network models on a subset of the dataset, as there was a tremendous difference in the number of images between the classes. By selecting a smaller subset of the Good images, we reduced the huge gap between the classes, helping to prevent our models from being biased and always classifying images as 'Good'.

This dataset was composed as follows: train dataset with 463 images, validation dataset with 110 images, test dataset with 70 images.

We implemented the CNN models from the background section according to the original papers but made minor adjustments to better fit our needs. For instance, in LeNet-5, we processed the images in RGB mode, as our ANA images are in RGB, and opted for ReLU activation functions, known for their strong performance, instead of the original tanh. The original LeNet-5 processed images in grayscale, mainly for handwritten digit recognition (MNIST), and used tanh because ReLU had not yet gained popularity at that time.

3.2 Cloud GPU For CNN Training

Since my laptop does not have a GPU, we had to train the CNN models on its CPU. However, the laptop frequently overheated during these sessions, causing training times to stretch into hours.

To mitigate these challenges, we researched and implemented strategies to optimize the performance.

Subsequently, cloud GPUs provided by platforms such as Kaggle and Google Colab were utilized, significantly easing the load on the laptop and greatly shortening training times.

This transition not only enhanced efficiency but also facilitated smoother progress in model development and experimentation.

3.3 Data Augmentation

In addressing the challenge of class imbalance and the limited availability of data in the anomaly and blurry classes, we adopted a strategic approach. Initially, we ran some models using the limited data available without employing data augmentation, which yielded acceptable results but not high enough to impress us. Recognizing the need to enhance accuracy further, we implemented data augmentation techniques. This proved to be a valuable solution, significantly improving our results. Below is an example demonstrating various data augmentation techniques applied to an ANA image, highlighting different transformations.

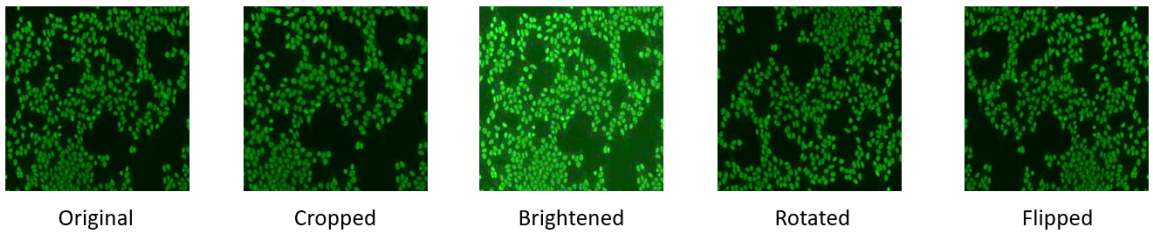


Figure 19: Example of different transformations on an ANA Image

We also attempted to address class imbalance by assigning higher weights to the blurry and anomaly classes, especially focusing on the blurry class due to its limited data availability. The intention was to prevent our models from biasing towards any particular class over others. However, upon implementation and subsequent analysis, we observed a slight decrease in performance across all models.

Our interpretation of this unexpected outcome is that identifying blurry images was not particularly challenging for our models, leading them to excel in this task rather than focusing on the more difficult task of detecting anomalies or determining image quality. As a result, we chose to abandon the weighting approach and shifted our focus to extensive data augmentation across the entire dataset, without weighting any specific class.

3.4 Fine Tuning Parameters

We undertook the task of retraining each model multiple times, adjusting various parameters to achieve optimal convergence. This involved fine-tuning parameters such as batch size, learning rate, optimizer, and the number of epochs. The goal was to find the best possible convergence.

To achieve this, we used a trial-and-error approach, systematically tweaking each parameter and carefully observing the resulting changes in model performance. By experimenting with different batch sizes, we aimed to balance training speed and stability. Adjusting the learning rate helped in controlling the step size during gradient descent, while selecting the appropriate optimizer influenced the efficiency of reaching convergence. Additionally, we varied the number of epochs to ensure the models had sufficient time to learn without overfitting.

This iterative process was crucial in refining the models to perform at their best and achieve the desired accuracy.

3.5 Metrics

To evaluate the performance of the models, we used several key metrics. Accuracy was a primary measure, providing an overall sense of how well the model performed. In addition, we calculated confusion matrices, which offered deeper insights into the classification results.

Here's the formula to calculate the Accuracy from a Confusion matrix:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

TP = True Positives

TN = True Negatives

FP = False Positives

FN = False Negatives

From the confusion matrices, we derived precision, recall, and the F1-score. Precision helped in understanding the model's ability to correctly identify positive instances, while recall measured the model's capability to capture all relevant instances. The F1-score, a harmonic mean of precision and recall, provided a balanced measure of the model's performance.
Here's the formulas to calculate Precision, Recall and F1-score:

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN} \qquad \text{F1-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

By using these metrics, we were able to conduct a comprehensive evaluation of the models, ensuring not only strong and stable overall performance but also valuable insights into their effectiveness across each of the three classes.

3.6 Models Handling

A Keras model consists of multiple components: the architecture or configuration, which specifies the layers and their connections; a set of weight values, representing the model's state; an optimizer, defined during model compilation; and a set of losses and metrics, also defined by compiling the model. The Keras API saves all these pieces together in a unified format marked by the .keras extension, which is a zip archive consisting of the following:

A JSON-based configuration file (config.json) that records the configurations of the model, layers, and other trackables. An H5-based state file, such as model.weights.h5, containing the entire model's weights with directory keys for layers and their weights. A metadata file in JSON, storing information such as the current Keras version, as cited in the official Keras website [18].

We used this technique to save the different CNN models, allowing us to reuse them later or calculate various metrics, such as plotting the losses of all the CNN models on a same graph. After evaluating the models, we loaded the best-performing model and deployed it on the hospital's server for practical application and future use on future ANAs images.

3.7 Visualization Techniques

Effective visualization techniques played a crucial role in analyzing and interpreting the performance of our machine learning models. We plotted Graphs for the training and validation accuracies, that clearly illustrated the model's learning trajectory across epochs, thereby providing insights into any potential overfitting or underfitting.

Additionally, the use of loss plotting enabled a deeper understanding of model convergence and optimization. Moreover, confusion matrices served as valuable aids in assessing the classification performance and identifying specific areas for model improvement.

These visualization methods collectively enhanced our ability to make informed decisions and refine our approach towards achieving optimal model performance.

4 Results

In this section, we provide an in-depth examination of the results from applying each of the eight CNN models to the ANA dataset. It includes a detailed analysis using various evaluation metrics, as well as visualization techniques to offer a clear perspective on how each model performed. Based on these insights, we will identify and select the most effective model for our needs.

4.1 LeNet-5

The accuracy obtained on the validation set: 90.9%
The accuracy obtained on the test set: 81.4%

On Good Images, the Precision, Recall and F1-Score are respectively: 72.22%, 92.86% and 81.25%
On Anomaly Images, the Precision, Recall and F1-Score are respectively: 89.47%, 60.71% and 72.34%
On Blurry Images, the Precision, Recall and F1-Score are respectively: 93.33%, 100% and 96.55%

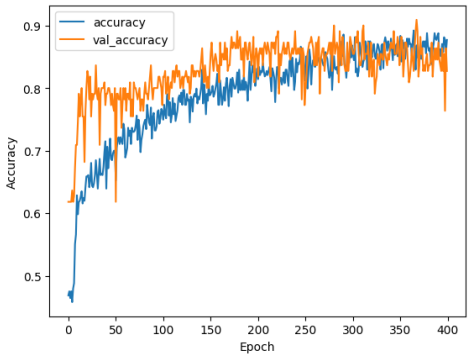


Figure 20: Graph showing the accuracy obtained on Validation set using LeNet-5

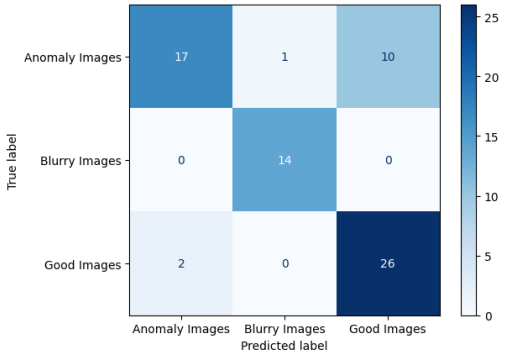


Figure 21: Confusion matrix obtained on Test set using LeNet-5

4.2 AlexNet

The accuracy obtained on the validation set: 94.6%
The accuracy obtained on the test set: 94.3%

On Good Images, the Precision, Recall and F1-Score are respectively: 90.32%, 100% and 94.91%
On Anomaly Images, the Precision, Recall and F1-Score are respectively: 100%, 85.71% and 92.31%
On Blurry Images, the Precision, Recall and F1-Score are respectively: 93.33%, 100% and 96.55%

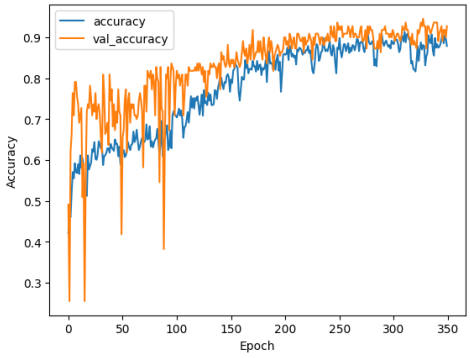


Figure 22: Graph showing the accuracy obtained on Validation set using AlexNet

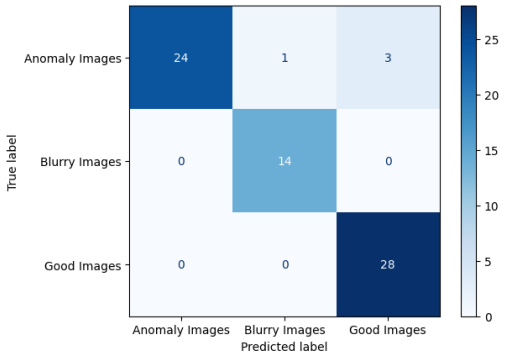


Figure 23: Confusion matrix obtained on Test set using AlexNet

4.3 ZFNet

The accuracy obtained on the validation set: 97.3%
The accuracy obtained on the test set: 91.4%

On Good Images, the Precision, Recall and F1-Score are respectively: 87.10%, 96.43% and 91.53%
On Anomaly Images, the Precision, Recall and F1-Score are respectively: 95.83%, 82.14% and 88.46%
On Blurry Images, the Precision, Recall and F1-Score are respectively: 93.33%, 100% and 96.55%

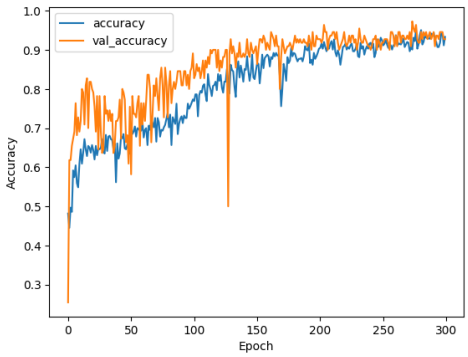


Figure 24: Graph showing the accuracy obtained on Validation set using ZFNet

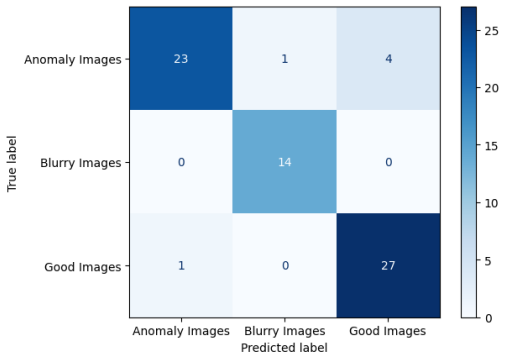


Figure 25: Confusion matrix obtained on Test set using ZFNet

4.4 VGG-16

The accuracy obtained on the validation set: 85.5%
The accuracy obtained on the test set: 87.14%

On Good Images, the Precision, Recall and F1-Score are respectively: 77.14%, 96.43% and 85.71%
On Anomaly Images, the Precision, Recall and F1-Score are respectively: 95.24%, 71.43% and 81.63%
On Blurry Images, the Precision, Recall and F1-Score are respectively: 100%, 100% and 100%

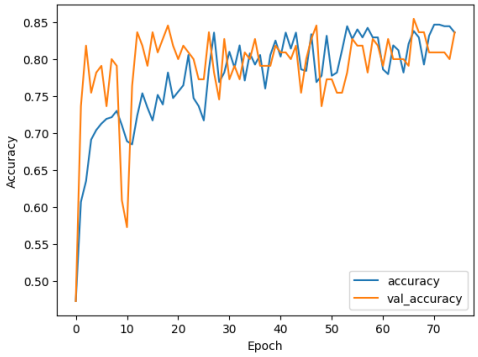


Figure 26: Graph showing the accuracy obtained on Validation set using VGG-16

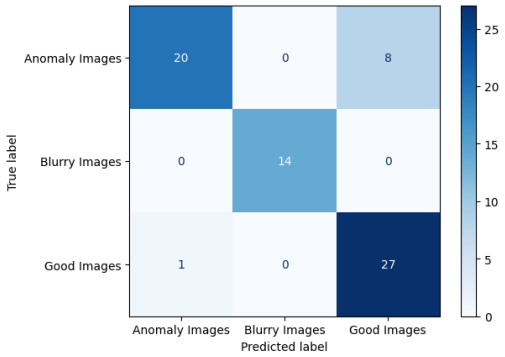


Figure 27: Confusion matrix obtained on Test set using VGG-16

4.5 InceptionV3

The accuracy obtained on the validation set: 94.6%
The accuracy obtained on the test set: 91.4%

On Good Images, the Precision, Recall and F1-Score are respectively: 100%, 78.57% and 88.00%
On Anomaly Images, the Precision, Recall and F1-Score are respectively: 82.35%, 100% and 90.32%
On Blurry Images, the Precision, Recall and F1-Score are respectively: 100%, 100% and 100%

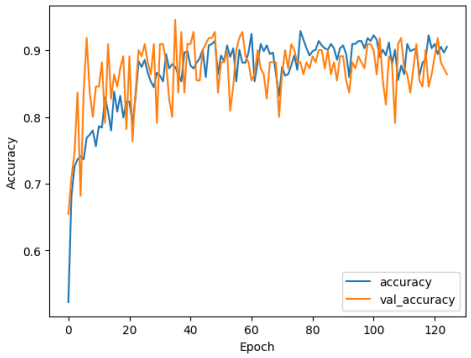


Figure 28: Graph showing the accuracy obtained on Validation set using Inceptionv3

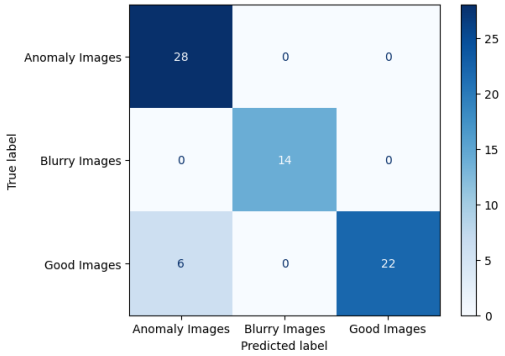


Figure 29: Confusion matrix obtained on Test set using Inceptionv3

4.6 ResNet-18

The accuracy obtained on the validation set: 96.4%
The accuracy obtained on the test set: 95.7%

On Good Images, the Precision, Recall and F1-Score are respectively: 93.10%, 96.43% and 94.74%
On Anomaly Images, the Precision, Recall and F1-Score are respectively: 96.30%, 92.86% and 94.55%
On Blurry Images, the Precision, Recall and F1-Score are respectively: 100%, 100% and 100%

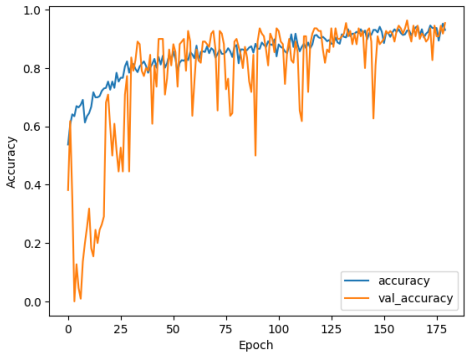


Figure 30: Graph showing the accuracy obtained on Validation set using ResNet-18

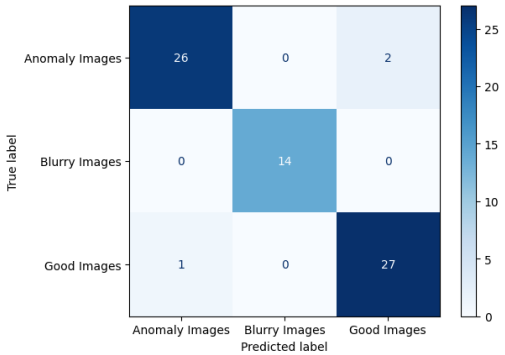


Figure 31: Confusion matrix obtained on Test set using ResNet-18

4.7 ResNet-50

The accuracy obtained on the validation set: 80.0%
The accuracy obtained on the test set: 75.71%

On Good Images, the Precision, Recall and F1-Score are respectively: 68.97%, 71.43% and 70.18%
On Anomaly Images, the Precision, Recall and F1-Score are respectively: 70.37%, 67.86% and 69.09%
On Blurry Images, the Precision, Recall and F1-Score are respectively: 100%, 100% and 100%

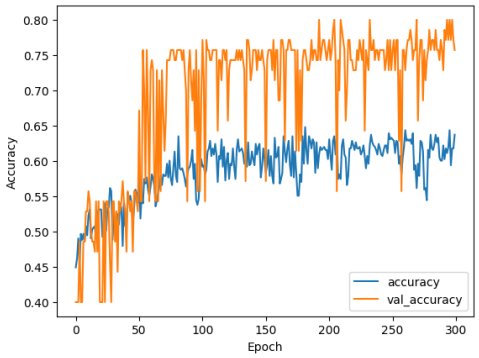


Figure 32: Graph showing the accuracy obtained on Validation set using ResNet-50

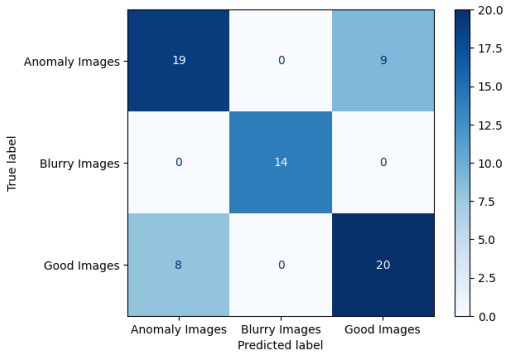


Figure 33: Confusion matrix obtained on Test set using ResNet-50

4.8 DenseNet-121

The accuracy obtained on the validation set: 91.8%
The accuracy obtained on the test set: 91.4%

On Good Images, the Precision, Recall and F1-Score are respectively: 89.29%, 89.29% and 89.29%
On Anomaly Images, the Precision, Recall and F1-Score are respectively: 89.29%, 89.29% and 89.29%
On Blurry Images, the Precision, Recall and F1-Score are respectively: 100%, 100% and 100%

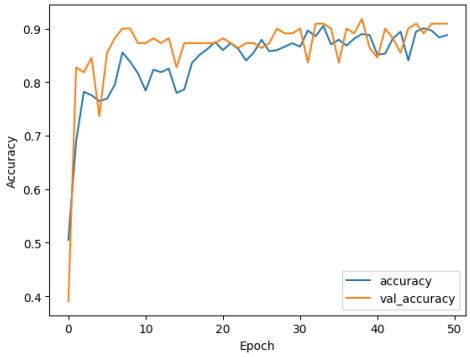


Figure 34: Graph showing the accuracy obtained on Validation set using DenseNet-121

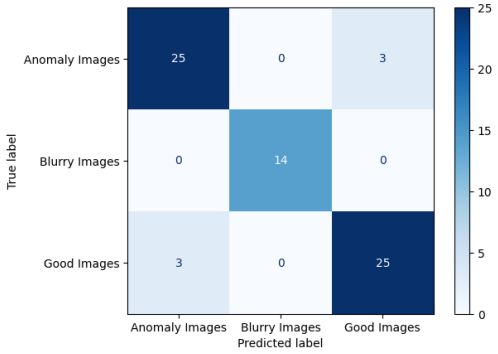


Figure 35: Confusion matrix obtained on Test set using DenseNet-121

4.9 Overview Of The Hyperparameters Used For The CNN Models

We focused on tweaking and fine-tuning hyperparameters with the goal of achieving the highest possible accuracy on the validation dataset. Employing a trial-and-error approach, we diligently adjusted various parameters and monitored their impact on the model's performance. This iterative process eventually led us to identify these effective hyperparameters, which significantly contributed to attaining high accuracies in the validation results. Find these hyperparameters in the table below:

Hyperparameters	**LeNet-5**	**AlexNet**	**ZFNet**	**VGG-16**	**InceptionV3**	**ResNet-18**	**ResNet-50**	**DenseNet-121**
Learning rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Batch size	32	32	32	64	32	32	32	32
Number of trained epochs	400	350	300	75	125	180	300	50
Number of epochs for best convergence	368	328	274	67	**36**	162	193	39

Table 2: Table illustrating the different hyperparameters used for the CNN models.

4.10 Losses Of The Different CNN Models

After implementing all the CNN models and completing the fine-tuning process, we obtained the final results. We carefully saved the models history, including the training metrics and parameters. We then plotted a graph illustrating the different losses of all eight models. This visualization helps in comparing the performance and understanding the training behavior of each model. Here below is the graph that shows the different losses of all the eight models for visual comparison.

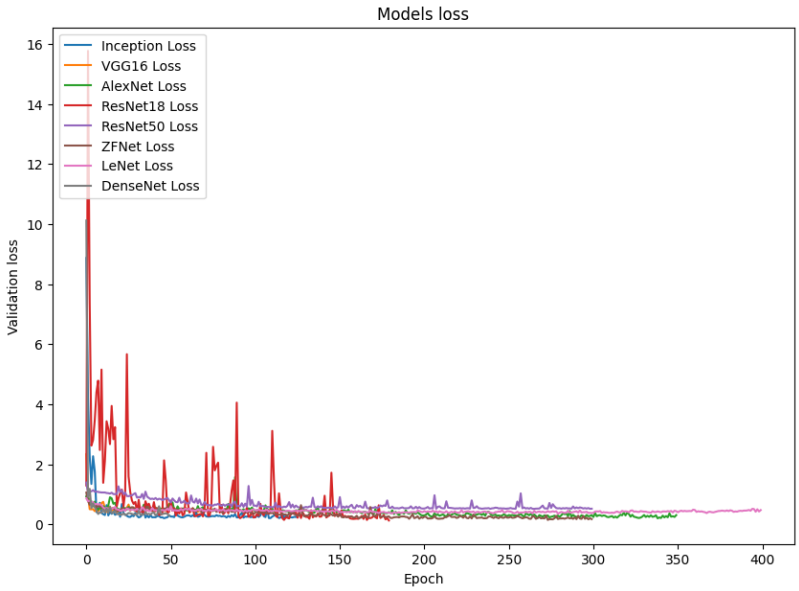


Figure 36: Graph showing the loss of the different CNN models in terms of epochs.

4.11 The Different Performance Metrics Results Of The CNN Models

Various performance metrics including precision, recall, F1-score, test accuracy, test loss, and macro F1-score were employed to evaluate and compare the performance of different CNN models. After careful consideration, the ResNet-18 CNN model was chosen due to its exceptional and outstanding performance compared to other models. The detailed comparison of the models performances is presented in the table below:

Class	**Metrics**	**LeNet-5**	**AlexNet**	**ZFNet**	**VGG-16**	**InceptionV3**	**ResNet-18**	**ResNet-50**	**DenseNet-121**
Anomaly	Precision	89.47%	**100%**	95.83%	95.24%	82.35%	96.30%	70.37%	89.29%
	Recall	60.71%	85.71%	82.14%	71.43%	**100%**	92.86%	67.86%	89.29%
	F1-Score	72.34%	92.31%	88.46%	81.63%	90.32%	**94.55%**	69.09%	89.29%
Blurry	Precision	93.33%	93.33%	93.33%	**100%**	**100%**	**100%**	**100%**	**100%**
	Recall	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
	F1-Score	96.55%	96.55%	96.55%	**100%**	**100%**	**100%**	**100%**	**100%**
Good	Precision	72.22%	90.32%	87.10%	77.14%	**100%**	93.10%	68.97%	89.29%
	Recall	92.86%	**100%**	96.43%	96.43%	78.57%	96.43%	71.43%	89.29%
	F1-Score	81.25%	**94.91%**	91.53%	85.71%	88.00%	94.74%	70.18%	89.29%
Global	Val-Loss	0.3741	0.2098	**0.1461**	0.4184	0.2215	0.1816	0.6005	0.3208
	Val-Acc	90.9%	94.6%	**97.3%**	85.5%	94.6%	96.4%	80.00%	91.8%
	Test-Loss	0.4194	0.3746	1.5297	0.3568	0.3199	**0.1473**	0.5197	0.1643
	Test-Acc	81.4%	94.3%	91.4%	87.1%	91.4%	**95.7%**	75.7%	91.4%
	Macro F1	83.38%	94.59%	92.18%	89.11%	92.77%	**96.43%**	79.76%	92.86%

Table 3: Table illustrating the performance metrics of different CNN models across various classes.

After analyzing the different results, the ResNet-18 model emerged as the best model for our task. We observed stable and high performance in terms of precision, recall, and F1-scores across the different classes. Among the eight CNN models evaluated, ResNet-18 achieved the lowest test loss (0.1473), the highest test accuracy (95.7%), and the highest macro F1-score across the three classes (96.43%).

5 Conclusion

5.1 Deployment On The Hospital's Server

After thoroughly working through our journey, debugging and resolving all issues, we achieved satisfying results. We selected the highest performing model, the ResNet-18 model, for its outstanding results in terms of Test-Loss, Test-Accuracy, and Macro F1-Score, and successfully deployed it onto the hospital server, ensuring its availability for future use on new ANAs images. This milestone represents a notable advancement in leveraging technology to enhance medical capabilities, providing improved diagnostic tools and advancing patient care.

5.2 Transfer Learning on ANAs Dataset

After using pre-trained models, we observed, as expected, fast convergence with a low number of epochs due to their prior training in recognizing edges and shapes. However, their convergence was not optimal, as some non-pretrained models outperformed them. This suggests that the ImageNet dataset likely did not include ANA images, limiting the effectiveness of transfer learning for this specific task.

5.3 Overall Gained Experience

My internship at the INFINITE U1286 Laboratory, under the mentorship of Dr. Adán José García, Clément Chauvet, and Pr. Vincent Sobanski, has been an invaluable experience. This opportunity allowed me to bridge the gap between theoretical knowledge and practical application, specifically in the realm of deep learning and Convolutional Neural Networks (CNNs). The skills and insights gained during this period have significantly enriched my understanding in this field.

Throughout the internship, I had the chance to apply the theoretical concepts I learned during my M1 Data Science year at the University of Lille. I delved into the practical aspects of deep learning, learning how to implement various techniques to enhance the accuracy and efficiency of neural networks. Some of the areas covered included coding different CNN models, preprocessing the data, addressing class imbalance, performing data augmentation, weighting classes, and utilizing pre-trained models. I also gained experience in tweaking and fine-tuning parameters to optimize model performance.

Additionally, I explored different platforms such as Google Colab and Kaggle, leveraging their GPUs to run my models faster and more efficiently. This hands-on experience was crucial in understanding the computational demands of deep learning models and the importance of computational resources. I also learned how to save CNN model information to local folders, ensuring reproducibility and ease of future use. Moreover, I expanded my knowledge of evaluation metrics beyond accuracy, including precision, recall, and F1-Score, which provided a more comprehensive understanding of model performance.

A particularly enriching aspect of this internship was the opportunity to work closely with medical doctors and attend thesis defenses of various medical students nearly every two weeks. This exposure to the medical field not only broadened my perspective but also underscored the real-world applications and impact of deep learning in healthcare.

The supportive environment encouraged by my mentors and the collaborative spirit of the ENDOMIC research team made this internship a smooth and rewarding experience. Their availability and guidance were instrumental in my learning journey. This experience has significantly motivated me to further explore and contribute to the world of data science and Artificial Intelligence.

For those interested in exploring the work presented in this report, the codes for the CNN models are available on my GitHub repository: https://github.com/GabrielWarde/Artifact-Detection-in-Immunofluorescence-Microscopy-Slides. Feel free to visit and use it as a source of inspiration for your own projects.

References

[1] Siddhesh Bangar. Resnet architecture explained. *Medium*, July 2022.

[2] Anas Brital. Inception v3 cnn architecture explained. *Medium*, October 2021.

[3] Chiranjib Chakraborty, Manojit Bhattacharya, Soumen Pal, and Sang-Soo Lee. From machine learning to deep learning: Advances of the recent data-driven paradigm shift in medicine and healthcare. *Current Research in Biotechnology*, 7:100164, 2024.

[4] Edward K. L. Chan, Jan Damoiseaux, Orlando Gabriel Carballo, Karsten Conrad, Wilson de Melo Cruvinel, Paulo Luiz Carvalho Francescantonio, Marvin J. Fritzler, Ignacio Garcia-De La Torre, Manfred Herold, Tsuneyo Mimori, Minoru Satoh, Carlos A. von Mühlen, and Luis E. C. Andrade. Report of the first international consensus on standardized nomenclature of antinuclear antibody hep-2 cell patterns 2014–2015. *Frontiers in Immunology*, 6, 2015.

[5] Eugene Dorfman. How much data is required for machine learning? *PostIndustria*, 2022. Accessed: 2024-08-12.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2017.

[8] IBM. What is deep learning?, June 2024. Accessed: 2024-06-17.

[9] ICAP ANA Patterns. Icap ana patterns, 2024. Accessed 26 July 2024.

[10] Rahul Jayawardana and Thusitha Bandaranayake. Analysis of optimizing neural networks and artificial intelligent models for guidance, control, and navigation systems, 04 2021.

[11] Raphael Kassel. Vgg: en quoi consiste ce modèle? daniel vous dit tout! *Formation Data Science — DataScientest.com*, April 2021. Accessed: 27 Avr 2021.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[14] FNP Lisa Carnago. Antinuclear antibodies (ana), February 2023. Reviewed by the American College of Rheumatology Committee on Communications and Marketing. Accessed 12 Aug. 2024. This information is provided for general education only. Individuals should consult a qualified health care provider for professional medical advice, diagnosis, and treatment of a medical or health condition.

[15] Abhijeet Pujara. Concept of alexnet: Convolutional neural network. *Analytics Vidhya*, November 2020.

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[17] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[18] Keras Team. Keras documentation: Save, serialize, and export models, 2024. Accessed 27 June 2024.

[19] Wikipedia contributors. Antinuclear antibody — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Antinuclear_antibody&oldid=1220160269, 2024. [Online; accessed 27-May-2024].

[20] Yi-Da Wu, Ruey-Kai Sheu, Chih-Wei Chung, Yen-Ching Wu, Chiao-Chi Ou, Chien-Wen Hsiao, Huang-Chen Chang, Ying-Chieh Huang, Yi-Ming Chen, Win-Tsung Lo, Lun-Chi Chen, Chien-Chung Huang, Tsu-Yi Hsieh, Wen-Nan Huang, Tsai-Hung Yen, Yun-Wen Chen, Chia-Yu Chen, and Yi-Hsing Chen. Application of supervised machine learning to recognize competent level and mixed antinuclear antibody patterns based on icap international consensus. *Diagnostics*, 11(4), 2021.

[21] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013.