Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Statistics and Machine Learning
2025 | LIU-IDA/LITH-EX-A--25/025--SE

3D Volumetric Defect Detection in Additive Manufacturing

Siddhesh Sreedar

Supervisor : Adhithyan Kalaivanan
Examiner : Jose M Pena

External supervisor : Filip Wänström

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Additive manufacturing, commonly referred to as 3D printing, is transforming modern production by enabling the manufacturing of complex geometries with reduced material waste. However, the presence of anomalies during the build process remains a significant challenge, often affecting part quality and leading to production inefficiencies.

This thesis investigates the use of 3D volume-based object detection techniques to identify defects within powder bed images. By treating image stacks as 3D volumes, this approach captures both spatial and temporal patterns, providing a richer representation of the build process. Machine learning and deep learning methods are implemented and evaluated to detect and localize defects using 3D bounding boxes.

Overall, traditional machine learning models like Random Forest and XGBoost outperformed deep learning architectures like ResNet in this study. Among the deep learning models, those pretrained on Med3D, a dataset from the closely related field of medical imaging, performed better than models trained from scratch or pretrained on unrelated data such as Kinetics-400. This suggests that transfer learning from domains with similar data characteristics can improve the performance. However, even the best-performing deep learning model lagged behind traditional methods, likely due to the relatively simple structure of the powder bed images. These results suggest that traditional models leveraging statistical features remain more effective in scenarios with low visual complexity.

Acknowledgments

I want to express my gratitude to my supervisor, Adhithyan Kalaivanan, for his time and continuous support throughout the thesis. The past few months have been incredibly valuable for both learning and applying the skills I gained during the master's program in a real-world industrial setting. Furthermore, I would like to thank my external supervisor, Filip Wänström, for his support and guidance during this work.

Last but not least, to my family, thank you for the constant support and comforting conversations throughout my journey. I am forever grateful to my parents for all the sacrifices they have made to ensure that I can pursue my interests and get the best education. I love you very dearly.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Additive Manufacturing (AM) commonly called 3D printing is a wave of transformative production technology that employs the principle of layer-by-layer integration of materials. This technology creates a physical object from a 3D model of the object created in Computer-Aided Design (CAD) software by sequentially layering material using a Computer-Controlled (CNC) machining tool [1]. Unlike traditional subtractive methods like machining or casting, AM has the ability to produce highly complex geometries which is often impossible or cost-ineffective with the traditional techniques. Due to this unique value proposition, various industries like aerospace, automotive and healthcare use this technology to build objects ranging from detailed turbines and lightweight structural components to personalized medical implants all while reducing material waste and production costs.

Additionally, AM seamlessly integrates with Industry 4.0 [1] principles through Digital Twin (DT) technology. A DT is a virtual representation of a physical object enabling real-time monitoring, simulation and optimization. DTs can demonstrate how the AM process is influenced by various environmental and processing parameters as well as changing conditions [2]. With Artificial Intelligence (AI), DTs can even analyze vast datasets to predict outcomes, detect anomalies and refine AM processes. This combination exemplifies the vision of Industry 4.0, fostering more resilient manufacturing ecosystems.

According to the Wohlers Report 2023, the AM industry experienced a worldwide growth of 18.3% in products and services, continuing a trend of double-digit revenue growth in 25 of the past 34 years [2]. The global AM market was valued at USD 20.37 billion in 2023 and is projected to grow at a Compound Annual Growth Rate (CAGR) of 23.3% through 2030 [3]. These figures underscore the immense potential of AM and highlight the pressing need for continued innovation to sustain and accelerate this growth.

[1] https://www.ibm.com/think/topics/industry-4-0
[2] https://wohlersassociates.com/news/wohlers-report-2023-unveils-continued-double-digit-growth/
[3] https://www.grandviewresearch.com/industry-analysis/additive-manufacturing-market

1.2 Motivation

The industries mentioned in the previous section have a high barrier to ensure the quality and reliability of the produced parts. A key challenge in AM lies in anomaly formation during the manufacturing process, which can compromise the structural integrity of parts and lead to costly build failures. While not all anomalies affect the final part significantly, they are often indicators of underlying issues that, if unchecked, can lead to build failures.

Traditional methods of defect detection are mostly applied after the build is complete (ex-situ), mainly post-build inspection. However, this is rather inefficient as they are both time-consuming and costly, and by the time an issue is detected, a significant amount of material is also used.

As a result of this, there is a growing demand for real-time detection (in-situ) techniques that allows for the identification of anomalies during the build process itself. By detecting these anomalies during the build process the operator can accordingly intervene, reducing downtime, minimizing material losses and enabling high quality production output [3].

With the growing technological innovation, this is where AI becomes particularly relevant. Along with the rise in available computational resources, AI has been at the forefront of driving advancements, making it more effective in providing value from the available data. By feeding relevant data to a model, we can automate the process and potentially detect and flag anomalies early.

By addressing this, we not only help further reduce anomalies in build jobs and improve the quality of the output but also contribute to a more efficient manufacturing cycle.

1.3 Collaboration with Interspectral

This research has been conducted in collaboration with Interspectral, a company specializing in advanced data visualization solutions for the AM industry. Their flagship software, AM Explorer, integrates monitoring and post-build data into a 3D environment, enabling the creation of digital twins for in-depth analysis. As part of this collaboration, Interspectral provided access to powder bed image datasets and supporting hardware infrastructure.

1.4 Aim

This research aims to contribute to in-situ techniques by implementing and evaluating methods for 3D bound box defect detection using a sequence of images instead of just using a single slice of an image. Essentially, the image stack represents a continuous view of the build job and its processes. This allows the capturing of both temporal and volumetric patterns, thereby enabling the detection of spatial features as well as changes over time. This thesis will use Machine Learning (ML) and Deep Learning (DL) techniques to tackle this problem. Subsequently, the performance of the models used will be evaluated according to standard metrics and the results will be analyzed. Specifically, the following research questions are explored:

1. What modeling approaches are most effective in accurately predicting the 3D bounding boxes around the defects?

2. What level of performance can these models achieve in defect detection?

The report is structured as follows: initially, the theory to introduce relevant concepts needed for this research will be explained. Following this, relevant literature will be reviewed. Afterward, the data along with the workflow process which includes the evaluation approach will be described. Lastly, the results obtained are presented and discussed, with the report concluding by outlining future work and final reflections.

2 Theory

This chapter introduces the core concepts and methods that form the basis of this research, offering the necessary background for understanding the techniques and models used.

2.1 Powder Bed Fusion

Powder Bed Fusion (PBF) is a type of AM process that utilizes either a laser or an electron beam to melt and fuse metal alloy powders, such as titanium and aluminium, into a solid form.

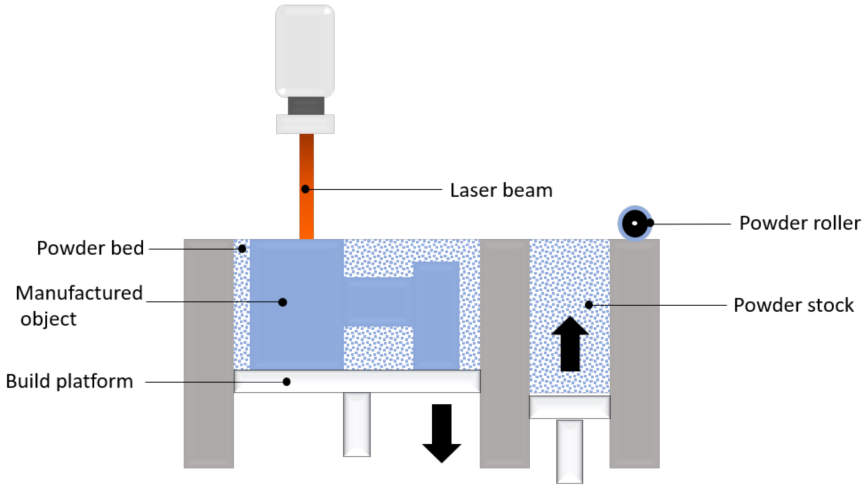

Figure 2.1: Schematic representation of PBF [4]. Available under CC license.

Figure 2.1 provides an overview of the process. It is set up in a vacuum to prevent contamination and begins with a powder roller spreading a thin layer of powder material onto the build platform. A laser/ electron beam then selectively fuses this layer into a solid structure. Next, the build platform lowers slightly, allowing a new layer of powder to be spread over the previous one using the powder roller. The laser/electron beam fuses the new layer to the

already solidified structure, and this cycle continues until the entire build job is completed. Once the printing is finished, post-processing steps begin, including the removal of unfused powder and support structures.

2.2 Anomalies

There are various types of anomalies that can occur at different stages, each arising due to different factors. These anomalies can vary in size, shape and location within the build area. Some of the most common anomalies include [5]:

1. **Porosity** - Small voids within the material, often caused by insufficient fusion or trapped gas. This can lead to structural failure over time.
2. **Splatter** - The powdered metal material may be splattered during the laser/electron beam interaction, leading to random deposition of material in unintended locations. This can cause surface roughness and irregularities in future layers.
3. **Wrapage/Part Distortion** - This occurs when thermal stresses cause the part to deform or warp during solidification. It is a serious issue because it can lead to interference with the powder roller, which spreads the powder for subsequent layers, potentially damaging the part or affecting powder distribution.

As a reference, a normal powder bed images can be seen in Figure 2.2.

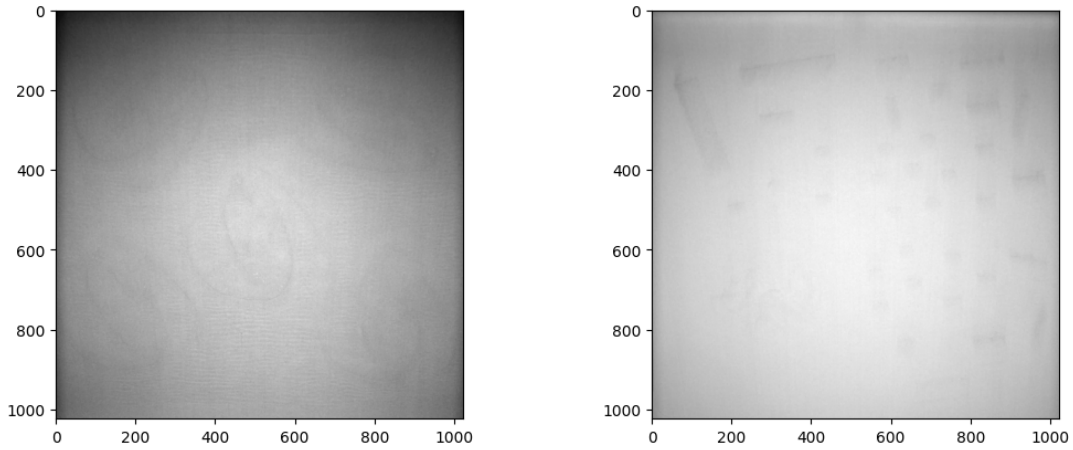

Figure 2.2: Normal powder bed images without anomalies.

In contrast, those with anomalies are illustrated in Figure 2.3. The example show anomalies within a single layer, meaning they are present in a 2D slice of the build. However, in this thesis, we will analyze entire layer sequences, considering 3d volumetric anomalies that evolve over time (3D visualization of anomalies can be viewed in Section 4). Anomalies that originate in one layer can propagate into subsequent layers, increasing in severity and complexity. Additionally, these anomalies do not necessarily have to grow in a straight line across layers but can also have a curved structure. However, there can also be anomalies that do not grow and get automatically healed during the fusion process of subsequent layers. These pointers further reiterate the potential of 3D defect detection in providing value.

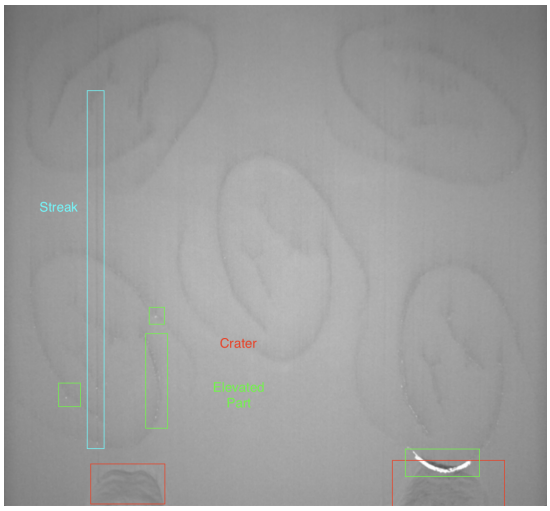

Figure 2.3: Powder bed image with anomalies. We see different types of anomalies with varying sizes and some of them are often subtle. The most notable error type is the "Elevated Part".

2.3 Image Feature Extraction

Feature extraction on images is an image processing technique that is used to identify and represent distinct structures within an image. These structures can range from edges and corners to more complex shapes. The main idea is to extract important information from these raw pixels while ignoring noise present in the image, which helps improve performance in various downstream tasks like object detection, image classification, segmentation, etc.

2.3.1 Sobel Operation

Sobel operator is a technique, particularly within the edge detection algorithms where it creates an image emphasizing edges. It is a discrete differentiation operator that is used to calculate the approximate gradient of an image.

It works by applying two 3x3 convolutional kernels to an image, one kernel is used for detection gradients in the x-direction while the other is used for detection gradients in the y-direction. These kernels are designed to detect edges marked by sudden changes in pixel intensity. A high value implies a steep change while a low value implies a shallow change [6].

The 2 kernels used are [1]:

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{2.1}$$

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{2.2}$$

The values in these kernels are symmetrical across a particular direction. The center values being zero help preserve the original intensity while the surrounding values help capture the intensity differences.

Each of these kernels is convolved with the input image I to compute the gradient G, producing Sobel images with edges enhanced in the respective directions.

[1] https://learnopencv.com/edge-detection-using-opencv/#how-are-edges-detected

$$G_x = K_x * I \tag{2.3}$$

$$G_y = K_y * I \tag{2.4}$$

where $*$ represents the convolution operation.

We then combine the two gradient values to find the strength of the edge at each pixel:

$$G(x,y) = \sqrt{G_x^2 + G_y^2} \tag{2.5}$$

Additionally, we determine the orientation of the edge at a particular pixel by computing the gradient direction $\theta(x,y)$ as:

$$\theta(x,y) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.6}$$

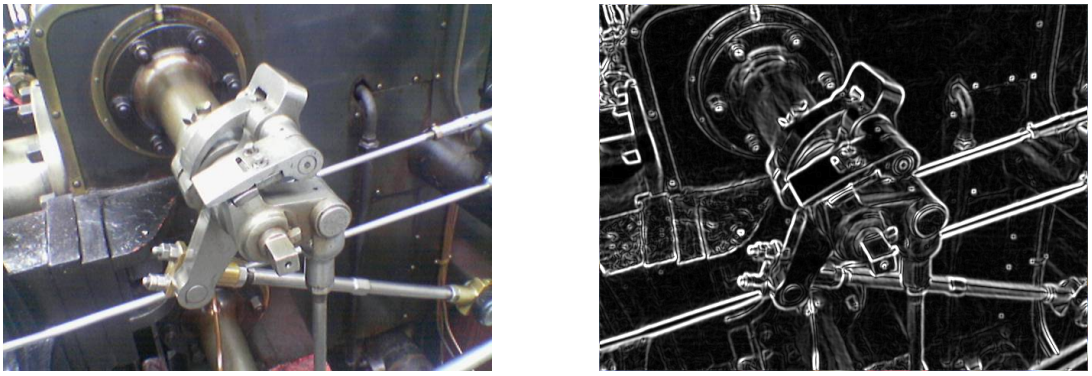

Figure 2.4: Comparison of the original image (left) and the processed image after applying the Sobel operator (right). Source: `https://en.wikipedia.org/wiki/Sobel_operator`

2.4 Machine Learning

2.4.1 Decision Trees

Decision trees are one of the most commonly used supervised ML algorithms, they can be used for both classification and regression tasks. For classification, it works by recursively binary splitting the data into smaller subsets based on the feature set to achieve the best separation of the classes [7]. These splits are determined based on a purity measure which should be maximized for the resulting nodes, meaning that each split should ideally create subsets containing mostly a single class.

To evaluate the purity of a node, we compute the proportion of each class m within a given node R_l, denoted as $\hat{\pi}_{lm}$. This is calculated as:

$$\hat{\pi}_{lm} = \frac{1}{n_l} \sum_{i:x_i \in R_l} \mathbb{I}(y_i = m) \tag{2.7}$$

where n_l is the number of samples in node R_l, and $\mathbb{I}(y_i = m)$ is an indicator function that equals 1 if the sample belongs to class m, otherwise 0.

Once the class proportions are determined, a label $\hat{y}_l$ can be assigned to the node (or leaf) by selecting the class with the highest proportion:

$$\hat{y}_l = \arg\max_m \hat{\pi}_{lm} \tag{2.8}$$

To decide how to split the data at each step, decision trees use impurity measures that quantify how mixed the classes are in a node. One common measure is the *Gini index*, which is defined as:

$$Q(R_l) = \sum_{m=1}^{M} \hat{\pi}_{lm}(1 - \hat{\pi}_{lm}) \tag{2.9}$$

where M is the total number of classes.

A lower Gini index indicates a purer node (where one class dominates), meaning a better split while a higher Gini index imples that the classes are more mixed.

The algorithm starts with the entire dataset at the root node, it then evaluates all the features and possible splits to determine which results in the best split. The separation criteria is computed as:

$$\arg\min_{j,s} (n_1 Q_1 + n_2 Q_2) \tag{2.10}$$

where:

- n_1 and n_2 denote the number of training data points in the left and right child nodes, respectively.
- Q_1 and Q_2 represent the Gini index associated with each of the two nodes.
- j refers to the index of the selected feature, and s is the split threshold for that feature.

By using this, decision trees can iteratively select the best splits until a stopping criterion is reached. Examples of such criteria include reaching a maximum tree depth or having too few data points in a node. Once the split stops, the leaf nodes contain the final classification.

2.4.2 Random Forest

Random Forest builds on the concept of a decision tree but rather than using a single decision tree, it leverages an ensemble of decision trees and combines their outputs to improve accuracy and reduce overfitting.

Each tree in the forest makes an independent prediction, essentially voting for a specific class based on the input features. Then, the probability of a particular class is determined by the fraction of trees that predict that class.

A single tree generally tends to have low bias but high variance, so it overfits the data and does not generalize well. Bagging (Bootstrap aggregation) reduces this variance by training on multiple trees and in each tree a different random subset of the data is used. However, since all trees use the same features, their outputs remain correlated, limiting the variance reduction.

Random Forest addresses this by introducing random feature selection at each tree split. This way the outputs of the trees are less correlated, further reducing the variance.

Consider the expectation and variance of an ensemble of B trees. Given predictions z_b from each tree in the ensemble, the expected value and variance are:

$$\mathbb{E}\left[\frac{1}{B}\sum_{b=1}^{B} z_b\right] = \mu, \tag{2.11}$$

$$\text{Var}\left[\frac{1}{B}\sum_{b=1}^{B} z_b\right] = \frac{1-\rho}{B}\sigma^2 + \rho\sigma^2. \tag{2.12}$$

where:

- μ represents the mean of the ensemble's output.
- σ^2 represents the variance of a single decision tree.
- ρ is the correlation between the predictions of different trees.
- B is the number of trees in the ensemble.

Equation (2.11) indicates that averaging multiple identically distributed random variables does not alter μ. Meanwhile, equation (2.12) shows that variance decreases with averaging, provided that the correlation ρ is less than 1. In the variance expression (2.12), the first term can be minimized significantly by increasing B , while the second term remains dependent solely on the correlation ρ and variance σ^2. We can then further reduce ρ by using random forest. However, this tends to increase σ^2 but it has been shown that the reduction in ρ has a more dominant effect [7].

2.4.3 eXtreme Gradient Boosting (XGBoost)

XGBoost is another build-up of the concept of a decision tree and also uses an ensemble of trees like Random Forest. However unlike Random Forest which trains trees independently, XGboost builds decision trees sequentially with each tree trying to correct the mistakes made by the previous one. These decision trees are often stumps (Single-split trees) and have low variance, but high bias. While Random Forest is primarily focused on reducing variance, XGBoost focuses on reducing bias.

The objective function is [8]:

$$\mathcal{L}(\phi) \;=\; \sum_{i=1}^{n} L(\hat{y}_i,\, y_i) \;+\; \sum_{k=1}^{t} \Omega(f_k) \tag{2.13}$$

where:

- $L(\hat{y}_i, y_i)$ is the loss function.
- f_k represents an independent tree structure.
- $\Omega(f_k)$ is the regularization term which discourages overly complex trees.

The regularization term is defined as:

$$\Omega(f_k) = \gamma\, T + \frac{\lambda}{2}\sum_{j=1}^{T} w_j^2 \tag{2.14}$$

where:

- T is the number of leaves in the tree.
- γ is a regularization parameter.
- λ is a parameter that penalizes the weight of the leaves w_j.
- w_j is the weight of leaf j.

The model is initialized with a constant value:

$$\hat{y}_i^{(0)} \;=\; \arg\min_{\theta} \sum_{i=1}^{n} L\big(y_i, \theta\big). \tag{2.15}$$

In each iteration, the gradient (first derivative) and Hessian (second derivative) of the loss with respect to its current prediction is calculated. These values serve as pseudo-residuals (what remains to be learned) and weights (how strongly each sample should drive the next tree). It then grows a new tree f_k by choosing splits that maximize the Gain:

$$\text{Gain} \;=\; \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} \;+\; \frac{G_R^2}{H_R+\lambda} \;-\; \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] \;-\; \gamma \tag{2.16}$$

- G_L, G_R are the sums of gradients in the left and right child nodes
- H_L, H_R are the sums of Hessians in the left and right child nodes

The optimal leaf weight w_j is then calculated to minimize the regularization objective:

$$w_j \;=\; -\frac{G_j}{H_j+\lambda} \tag{2.17}$$

The new tree's predictions are then added to the ensemble's output. Those updated predictions feed into the next iteration's gradient/Hessian calculation, so each successive tree focuses on correcting the previous ensemble's residual errors while γ and λ keep individual trees from becoming too complex or having large weights.

2.5 Deep Learning

2.5.1 2D Convolutional Neural Networks (2D CNNs)

CNNs are mainly used in the field of computer vision due to their ability to extract features from images automatically without the need for manual feature extraction, like in the case of traditional ML models. Additionally, it is able to capture a lot of varying and complex features which makes it more robust.

2D images are represented as three dimensions: height (H), width (W) and channels (C). In grayscale images, there is only one channel, whereas in RGB images, there are three channels (Red, Green and Blue). These are stored in an array containing pixel intensities.

Traditional neural networks rely only on fully connected layers [9], while CNNs use spatial hierarchies to preserve local features and parameter sharing to reduce the number of trainable parameters. This, along with other concepts which will be explained in the following paragraphs, helps to better summarize information and strengthen the features in the image.

The core of a CNN is the convolutional operation, where a kernel (filter) commonly of dimension 3 X 3 slides across the entire image.

The process can be mathematically expressed as [7]:

$$q_{ij} = h\left(\sum_{k=1}^{F}\sum_{l=1}^{F} x_{i+k-1,j+l-1} W_{k,l}\right) \tag{2.18}$$

where:

- x_{ij} represents the zero-padded input to the layer.
- q_{ij} is the output at location (i, j).

- $W_{k,l}$ represents the kernel (filter) with size $F \times F$, learned through backpropagation.
- h is the activation function (e.g., ReLU).

The above formula can be adapted for RGB images of 3 channels by applying independent kernels to each channel and summing the results.

Additionally, multiple kernels are trained to allow CNNs to learn different aspects of the image, enhancing feature extraction. The collection of all q_{ij} values from one kernel form a single feature map. When multiple kernels are applied, they generate multiple feature maps, which are then stacked together as the output of the convolutional layer.

When the kernel slides across the image, it moves in steps defined by the stride. If it is set to 1, the kernel moves one pixel at a time, which captures finer details and a larger feature map size. If it is greater than 1, it moves multiple pixels at a time, which leads to a smaller feature map size and lower computational cost. To ensure that information is not lost on the border of the image, padding is applied by adding zeros around the border.

After each convolution operation, a non-linear activation function is applied. The most common one is ReLU (Rectified Linear Unit) [10]. This helps introduce non-linearity, ensuring the model can learn complex patterns. After applying the activation function to a convolved feature map, a pooling layer is included which reduces the feature space by maximizing or averaging over it. For example, a 2×2 window slides across the feature map, keeping only the max value. This allows for better generalization.

The next step is to pass the resulting feature map into fully connected layers by first flattening the last feature map output into a 1D vector. The number of neurons in the final layer depends on the number of output classes. For classification tasks, the sigmoid function is applied in the final layer:

$$P(y = 1) = \frac{1}{1 + e^{-z_i}} \tag{2.19}$$

where:

- z_i is the input logit from the final layer.
- the output represents the probability of $y = 1$.

2.5.2 3D Convolutional Neural Networks (3D CNNs)

CNNs generally operate on 2D data, such as grayscale or RGB images. However, 3D CNNs use 3D convolutional filters instead of 2D filters. This 3D filter has to convolve over three axes, thus capturing context information between slices, but also requiring far more computational resources than its 2D counterpart. Apart from this, the process is exactly the same as that of a 2D CNN.

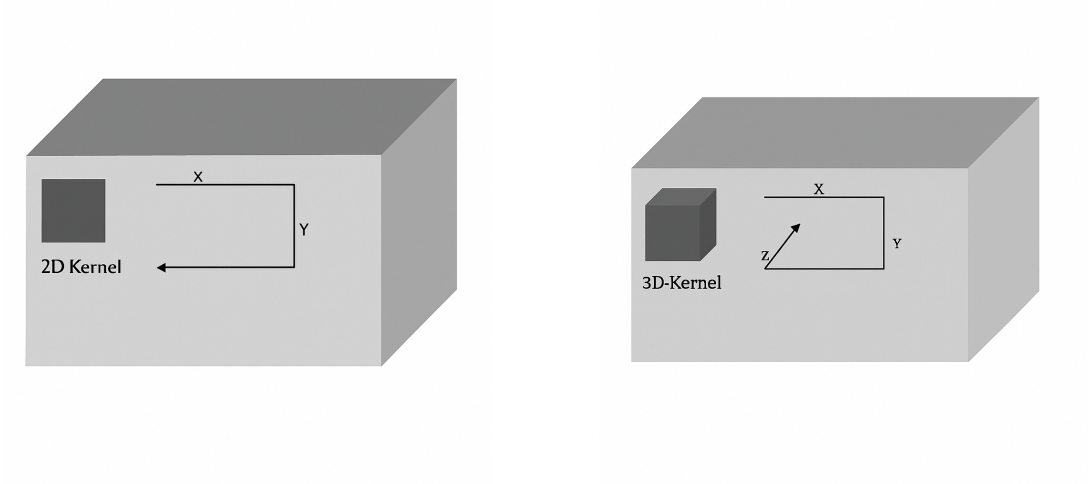

Figure 2.5: Visual representation of how 2D and 3D kernels operation works. In the 2D case (left), the kernel moves along the X and Y dimensions and operates on each 2D slice, where depth (Z) typically represents image channels. In contrast, for the 3D case (right), the Z-dimension corresponds to image stacks, allowing the kernel to extract volumetric features in 3D data.

2.5.3 Residual Networks (ResNet)

As deep learning models evolved, researchers believed that increasing the number of layers in neural networks would lead to better feature extraction and improved performance. However, in practice, stacking many layers leads to a degradation problem, where increasing network depth does not necessarily improve accuracy. Instead, deeper networks often reach a saturation point and then decline rapidly in performance. This was also proven not to be related to overfitting, as the training error was increasing.

To solve this problem, residual learning was introduced [11]. In traditional deep networks, each layer learns a direct mapping from input to output. However, in ResNets, identity mapping is performed by taking the input of a block x and adding it to the output of the block as shown in Figure 2.6. This is achieved through a shortcut connection that bypasses one or more layers and directly adds the input x to the output of the deeper layers. This allows gradients to flow more easily during backpropagation, reducing the risk of vanishing gradients and making it possible to train much deeper networks. In a single residual block, the input x is passed through multiple convolutional layers, but instead of relying entirely on these transformations, the original input x is added back before applying the activation function. This ensures that even if deeper layers struggle to refine the features, the network can still retain the identity mapping.

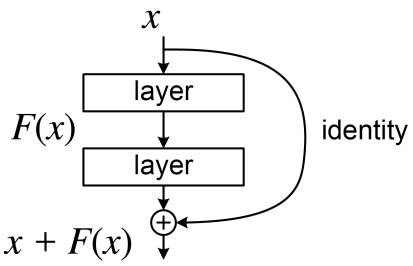

Figure 2.6: Residual learning: A building block Source: `https://en.wikipedia.org/wiki/Residual_neural_network`

By stacking multiple residual blocks, different architectures such as ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152 were developed, each enabling deeper networks with improved stability and performance as shown in Figure 2.7.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.7: ResNet architectures with different layers [11]

Building on this, 3D versions of these architectures have also been developed that primarily replace 2D convolution [2] with 3D convolution [3].

2.5.4 Transfer Learning

One key advantage of using DL is Transfer Learning, which is a technique where a model pre-trained on one task is fine-tuned on a new but related task. Training DL models from scratch is a time-consuming and intensive process, requiring not only a large number of data but also high computational resources.

TL [12] significantly reduces the need for extensive data and computation by leveraging the knowledge encoded in the pre-trained model. Instead of training a model from the ground up, only the later layers are generally fine-tuned, allowing the model to adapt to new datasets while retaining essential base knowledge. The lower layers of the model typically capture general features (e.g., edges, textures and basic structures), while the *Head* as shown in Figure 2.8, which signifies the upper layers, specializes in task-specific information. By freezing the lower layers and fine-tuning the higher ones, TL ensures efficient learning with limited data.

Furthermore, TL enables models to benefit from pre-training on large-scale datasets, such as ImageNet [13], which contain millions of labeled examples. This means that even when

[2] `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`
[3] `https://pytorch.org/docs/stable/generated/torch.nn.Conv3d.html`

working with a relatively small dataset, the model inherits fundamental knowledge from massive, resource-intensive training processes, making it feasible to achieve high performance without requiring extensive labeled data.

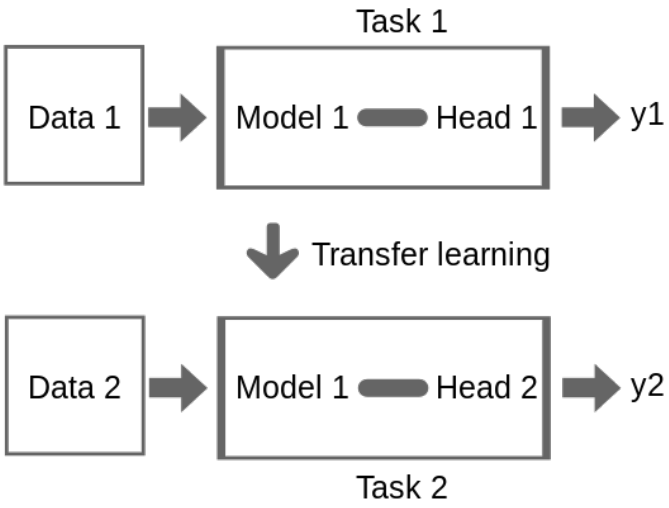

Figure 2.8: Illustration of transfer learning. Source: `https://en.wikipedia.org/wiki/Transfer_learning`

2.6 Loss Functions

2.6.1 Binary Cross-Entropy (BCE)

Binary Cross-Entropy (BCE) is the most commonly used loss function in classification tasks. It is computed as follows:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right) \tag{2.20}$$

Where N is the total number of samples, y_i is the true label for sample i, where $y_i \in \{0,1\}$, and $\hat{y}_i$ is the model's predicted probability for sample i, where $0 \leqslant \hat{y}_i \leqslant 1$.

2.6.2 Focal Loss (FL)

Focal loss generalizes BCE by reducing the weight of easy-to-classify samples and gives more importance to hard-to-classify examples. This helps the model focus more on difficult cases, which is particularly useful for imbalanced datasets. It is computed as follows:

$$FL = -\frac{1}{N}\sum_{i=1}^{N}(1-\hat{y}_i)^{\gamma}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right) \tag{2.21}$$

where γ is the focusing parameter that controls how much weight is given to hard examples. $(1-\hat{y}_i)^{\gamma}$ is the weighting term that reduces the contribution of easy-to-classify samples (where $\hat{y}_i$ is high) and boosts hard examples (where $\hat{y}_i$ is low).

- When $\gamma = 0$, Focal Loss reduces to BCE.
- When $\gamma > 0$, hard-to-classify examples contribute more to the loss, and easy ones contribute less.

2.7 Model Optimization

2.7.1 Stochastic Gradient Descent (SGD)

Stochastic gradient descent is an iterative algorithm that is used to minimize a pre-defined loss function. At each iteration, the training weights of the model are updated using the gradient of the loss function. In traditional gradient descent, the gradients are computed based on the entire dataset, which can be computationally expensive for large datasets. SGD on the other hand, uses only a small batch of data points at each iteration. This makes the computation much faster.

Algorithm 1 Stochastic Gradient Descent [7]

Require: Objective function $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \theta)$, initial $\theta^{(0)}$, learning rate $\gamma^{(t)}$
Ensure: $\hat{\theta}$

Set $t \leftarrow 0$
while Convergence criteria not met **do**
  for $i = 1, 2, \ldots, E$ **do**
    Randomly shuffle the training data $\{\mathbf{x}_i, y_i\}_{i=1}^{n}$
    for $j = 1, 2, \ldots, \frac{n}{n_b}$ **do**
      Approximate the gradient using the mini-batch

$$\{(\mathbf{x}_i, y_i)\}_{i=(j-1)n_b+1}^{jn_b}, \quad \hat{\mathbf{d}}^{(t)} = \frac{1}{n_b} \sum_{i=(j-1)n_b+1}^{jn_b} \nabla_\theta L(\mathbf{x}_i, y_i, \theta^{(t)})$$

      Update $\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \hat{\mathbf{d}}^{(t)}$
      Update $t \leftarrow t + 1$
    end for
  end for
end while
return $\hat{\theta} \leftarrow \theta^{(t-1)}$

2.7.2 Adaptive Moment Estimation (Adam) Optimizer

The idea behind Adam optimization is to adjust the learning rate adaptively for each parameter in the model based on the history of gradients calculated for that parameter. This helps the optimizer converge faster and more accurately than fixed learning rate methods like SGD. This is done by tracking both the first moment (mean of gradients) and the second moment (uncentered variance of gradients) using Momentum and RMSProp respectively [14].

Algorithm 2 Adam Optimizer

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1]$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

2.8 Hyperparameter Tuning

2.8.1 Bayesian Optimization

Bayesian optimization is an effective approach for finding the best hyperparameters for models. Unlike random search, which is independent of past evaluations, Bayesian optimization iteratively builds a probabilistic model of an objective function guiding the search toward parameter values that are most likely to yield optimal results. This method uses the Bayesian process to reduce the search space and find the optimal configurations.

Finding the true objective function is expensive since that would require evaluation on all possible hyperparameter combinations. To mitigate this, Bayesian optimization builds a surrogate model, which is an approximation of the objective function and is cheaper to evaluate on. This is done using the Gaussian Process (GP) [15].

At the start of the process, a random selection of a small set of hyperparameter combinations is chosen and evaluated to give the initial set of observations. These observations are used to initialize the surrogate model. For a set of observations x, the GP models the objective function as a Gaussian distribution with a mean $\mu(x)$ and variance $\sigma^2(x)$. This provides a prediction of the performance at any point in the hyperparameter space and also a measure of the uncertainty.

To decide which hyperparameter combination to evaluate next, an acquisition function is used. This function is used to balance exploration and exploitation. A commonly used acquisition function is Expected Improvement (EI):

$$EI_{y*}(x) = \int_{-\infty}^{\infty} \max(y^* - y, 0)\, p(y \mid x)\, dy \tag{2.22}$$

where:

- $p(y \mid x)$: the surrogate model. y is the true objective function score and x is the hyperparameter.
- y^*: the minimum observed true objective function score so far.
- y: new scores sampled from the surrogate model.

The next hyperparameter setting to evaluate is the one that maximizes the function in Equation 2.22. The selected hyperparameter combination is evaluated on the objective function, producing a new observation which is then added to the existing set of observations x. The GP is then updated with the new observation set and the process repeats by calculating the acquisition function based on the new GP. The process repeats until a stopping criterion is met such as a maximum number of iterations, no significant improvement across iterations etc.

2.9 Evaluation Metrics

2.9.1 Intersection over Union (IoU)

IoU, also know as the Jaccard index is a metric that quantifies the degree of overlap between the ground truth and the predicted bounding box. Consider two arbitrary shapes (or volumes), A and B, within a given space S in $\mathbb{R}^n$ [16]. It is defined as:

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{2.23}$$

where:

- $A \cap B$ is the intersection, representing the common area shared by both shapes.
- $A \cup B$ is the union, representing the total area covered by both shapes.
- $|\cdot|$ denotes the area in 2D or volume in 3D of the respective regions.

IoU values range between 0 and 1, where:

- IoU $= 0$ indicates no overlap between A and B.
- IoU $= 1$ indicates a perfect match between A and B.
- Higher IoU values indicate a more accurate prediction.

2.9.2 Dice Coefficient

The Dice coefficient is another metric similar to IoU that assesses the overlap between the ground truth and the predicted bounding box. However, it gives more emphasis to the correctly classified area due to the double weighting in the numerator, making it less strict than IoU, which penalizes false positives more heavily. It is defined as:

$$\text{Dice} = \frac{2 * |A \cap B|}{|A| + |B|} \tag{2.24}$$

2.9.3 Precision and Recall

Precision and recall are commonly used metrics for classification problems and can be used in bounding box-based defect detection as well. Precision measures the proportion of correctly predicted bounding boxes among all the boxes predicted by the model. A high precision score indicates that the model makes fewer false positive predictions, meaning it does not wrongly classify background regions with a bounding box.

The precision score is given by:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.25}$$

where:

- **True Positives (TP)**: The number of predicted bounding boxes that correctly overlap with a ground truth box (Based on an IoU threshold).
- **False Positives (FP)**: The number of predicted bounding boxes that do not achieve the pre-determined overlap threshold with any ground truth box.

Similarly, recall measures the proportion of correctly detected bounding boxes out of all actual bounding boxes present in the dataset. A high recall means fewer false negatives, meaning the model successfully detects most of the ground truth bounding boxes.

The recall score is then given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.26}$$

where:

- **False Negatives (FN)**: The number of ground truth bounding boxes that were not detected by the model.

A model with high precision may avoid false positives but might miss some actual bounding boxes (lower recall). Conversely, a model with high recall may detect most bounding boxes but also introduce false positives, reducing precision. It is important to balance these two metrics, and that is where the trade-off comes into play.

2.9.4 F1-Score

F1-score combines both precision and recall to give an overall accuracy and is very useful when working with imbalanced datasets. Maximizing the F1 score implies simultaneously maximizing both precision and recall.

It is computed as follows:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.27}$$

3 Literature Review

3.1 Additive Manufacturing Field

Research in anomaly detection in AM has primarily focused on analyzing defects at the 2D image slice level. While this approach has led to significant advancements in detecting and classifying powder bed defects, it does not fully capture the volumetric nature of anomalies present. Despite this, several studies have successfully leveraged different techniques to improve defect detection and classification from single-layer images.

Xiao et al. [17] proposed a two-stage convolutional neural network for detecting three common types of powder bed defects. In the first stage, a pre-trained ResNet-101 was used to extract multiscale features, followed by a region proposal network to detect the object-level defect bounding box. In the second stage, a Fully Convolutional Neural Network (FCN) refined these predictions to generate instance-level defect regions.

Ansari et al. [18] focused on a binary classification approach to determine whether an image contained porosity. They developed a custom CNN trained from scratch, optimized through hyperband tuning to find the best hyperparameter configuration. Their findings highlighted the importance of dataset balancing, showing that a balanced dataset improved model performance compared to training on an imbalanced dataset.

Schmitt et al. [19] developed a powder bed monitoring system using semantic segmentation to classify and differentiate melted vs unmelted areas in powder bed images. Using an Xception-style neural network, their approach compared segmented layers against reference layers. They calculated geometric deviations, area and centroid shifts to detect part failures and thermal distortions. Their findings demonstrated that segmentation-based monitoring can enhance defect detection and quality assurance, although limitations in camera resolution and lighting conditions affected accuracy.

Due to the different kinds of anomalies that can exist, Ansari et al. [20] used transfer learning to apply a Deep Convolutional Neural Network (DCNN) model trained on porosity detection to the task of detecting surface deformations. Their research validated that the knowledge learned from detecting one type of anomaly could be effectively transferred to identify different defects, even with limited labeled data.

While the focus has been heavily on solving this problem using CNNs, there has been less focus on using traditional ML models to see if we can get comparative performance. Khan et al. [21] employed a Random Forest Classifier to segment anomalies from optical images.

They made this choice primarily due to ML models requiring less volume of training data unlike CNNs to generalize effectively. 42 features were extracted from the images to train the model such as median filter, pixel intensity, Gaussian blur, Sobel edge detection etc.

As AM processes inherently involve volumetric changes, moving toward 3D-based object detection methods is essential for more precise and comprehensive anomaly identification.

Wong et al. [22] used 3D CNNs in particular 3D U-Net for volumetric segmentation of defects on X-ray Computed Tomography (X-CT) images of AM samples.

Iuso et al. [23] investigated both supervised and unsupervised DL approaches for voxel-wise porosity classification using X-CT images. The supervised models included the U-Net family of models, while the unsupervised models were the Variational AutoEncoders (VAE) family of models. To make use of 3D volumetric data efficiently, 3D patches of 64×64×64 voxels were extracted and supplied to the neural networks. Additionally, Focal Tversky loss was used to address the class imbalance that arises from the low porosity in the training datasets. Overall, the unsupervised VAE-based models outperformed the supervised models, particularly when combined with post-processing techniques to reduce misclassifications.

3.2 Medical Field

When exploring methods for 3D volumetric object detection, the medical imaging field stands out as the closest parallel to AM. The field relies on analyzing volumetric 3D image data via Computed Tomography (CT), Magnetic Resonance Imaging (MRI) scans to detect tumors, fractures and organs, etc. Given this similarity, research in medical imaging serves as a strong foundation for developing more anomaly detection techniques in AM. Additionally, the medical imaging field is significantly more mature in its research and the knowledge gained from the medical imaging domain can offer valuable insights.

Prior to the widespread adoption of DL methods, the field relied on traditional ML models for object detection. Criminisi et al. [24] used multi-variate Regression Random Forest (RRF) models for localization to find the 3D bounding box coordinates of organs using CT images. Gauriau et al. [25] proposed a global-to-local cascade of RRF for fast and accurate localization of multiple organs. The first Random Forest is trained to model the global relationships between organs, learning their parameters simultaneously, while the second Random Forest refines localization independently for each organ. By integrating confidence maps based on shape priors (probabilistic atlases), their method improves localization accuracy beyond traditional bounding boxes results.

Hartmann et al. [26] performed a comparative study between a Random Forest approach and DCNN. While the paper focused on 2D medical image segmentation, it can still provide insights. The results found that the Random Forests, based on a feature extraction architecture, were able to achieve excellent results and were comparable to DCNN but not as good as them. However, the RF approach have the advantage that it can also be used in smaller IT infrastructures without the need for a GPU cluster.

With the move towards DL, several new approaches and models have been explored. De Vos et al. [27] used DCNN on CT scans to detect anatomical Regions of Interest (ROI) in 2D image slices and the combination of these results provided a 3D bounding box around it.

Majority of approaches to object detection in this field follow a 2-step process: localization and segmentation. Localization involves identifying the region of interest by generating a 3D bounding box, while segmentation refers to the pixel-level segmentation of the organ or object within that region. This leads to a more efficient approach, with a smaller region to process for the segmentation step, the computational load decreases, making the method more resource-efficient. Additionally, by narrowing down the focus, segmentation models achieve better accuracy compared to processing the entire region.

One approach [28] employed a 3D FCN, where Canny edge detection was used to select high-response voxels (32×32×32), which were then fed into the network. Another study [29]

utilized a Multi-Layer Perceptron (MLP) with a similar feature extraction process, selecting only significant voxels to optimize computational efficiency. A third approach [30] applied a volumetric CNN on a 3D sliding window (128×128×128) over the entire volume, using a 10-layer Very Deep Convolutional Networks (VGG)-style architecture to classify windows as containing the object or not.

The first two methods framed localization as a regression problem, predicting the six coordinates of the bounding box, while the third treated it as a classification problem, labeling windows as positive or negative based on occupancy thresholds.

Other models used are 3D ResNet [31] and 3D Faster R-CNN [32].

For the segmentation step, models such as 3D U-Net and V-Net were used to refine the object's boundaries within the localized region.

A comparative study done by Uemuraa et al. [33] evaluated the performance of three different 3D CNNs on the classification of CT scans— 3D-DenseNet, 3D-ResNet and 3D-VGG. Results showed that the 3D-DenseNet yields a statistically significant improvement over the others and also in reducing false positives.

3.3 Comparative Analysis with Related Work

In terms of similarities, my work draws inspiration from both medical imaging and AM research. For example, similar to approaches used in medical image analysis and AM, I use a sliding window based on a voxel size to process the 3D data. This idea of working with voxel-wise context is something that's been effectively applied in medical tasks for localization and segmentation. On the other hand, what sets my work apart is the focus on applying transfer learning using pretrained 3D models that were originally developed for medical data but applying it in the context of AM.

4 Data

The dataset used in this work consists of six 3D volumes which are a subset of stacks from a few build jobs originating from real-world manufacturing processes. The annotation of the defects is performed using a combination of automatic detection pipelines and manual refinement performed by domain experts. Additionally, only the most prominent error type (Part Elevation) is labeled.

Each volume has a consistent shape of 512×1024×1024 (Z×Y×X), where Z denotes the number of layers, and X and Y represent the width and height of each layer image, respectively. These grayscale images have a resolution of 1024 × 1024 per slice, capturing detailed powder bed information across the layers of the build job.

Selected 2D slices with corresponding annotations are visualized in Figure 4.1. Likewise, a subset of volumes is also rendered in 3D to illustrate the spatial distribution of the annotated bounding boxes throughout the build in Figure 4.2

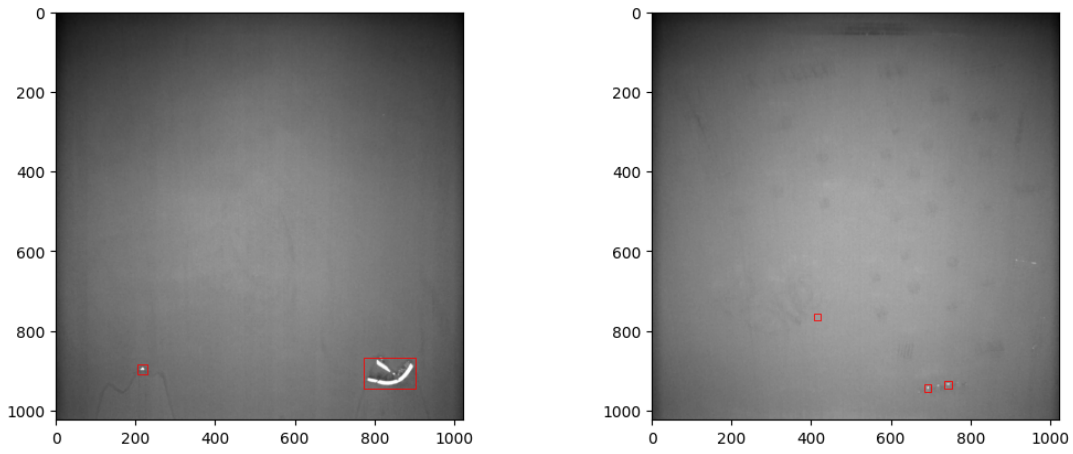

Figure 4.1: Selected 2D slices with corresponding ground truth annotations (in red) from two different volumes.

The total number of annotated defects per volume varies considerably. Across the six build jobs, the number of 3D bounding boxes ranges from 11 to 35. Furthermore, the sizes of

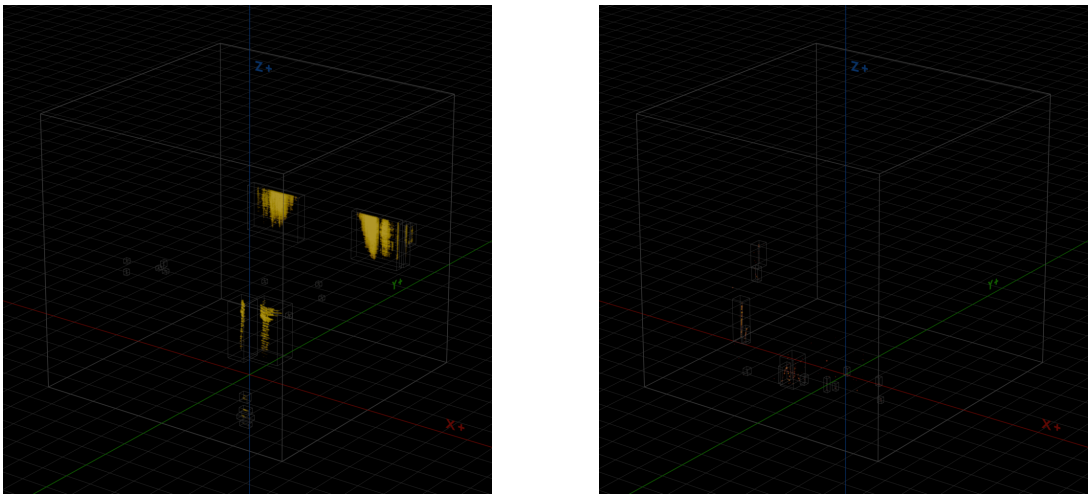

Figure 4.2: 3D visualization of volumes with annotated bounding boxes. The annotations showcase how subtle and close together some of the anomalies are (Especially on the right image), highlighting the complexity of the problem.

these bounding boxes differ significantly, with sizes ranging from as small as 20×15×15 to as large as 519×82×238. The location of these bounding boxes also varies; some are distributed sparsely throughout the volume, while others appear in tight clusters. Defects may appear near the beginning of the job, emerge sporadically in the middle, or accumulate toward the later stages.

With such diversity, it emphasizes the importance of using volume-based models capable of capturing not just local features but also temporal information throughout the build job. However, the proximity of some bounding boxes to each other may introduce challenges in distinguishing individual defect instances, which must be taken into account in detection and evaluation.

5 Method

The following subsections are structured to showcase the flow of the work carried out with regard to the model selection process, along with its implementation. Following this, the evaluation approach is discussed.

5.1 Exploratory Data Analysis (EDA)

The data was first split into train and test, with 4 volumes of data going to the training set and 2 for the testing set. Due to the limited size of the dataset, no separate validation set was used. Instead, hyperparameters were tuned using the training data itself. This could lead to potential overfitting, which should be kept in mind when evaluating the model's performance.

The next step involved EDA to allow for a better understanding of the data, how the powder bed images vary across different layers and what the anomalies looked like. Due to the high memory and computational requirements of processing full 3D volumes, each consisting of up to several hundred high-resolution image slices, it is impractical to analyze the entire volume in one pass. To address this, a sliding window approach was applied that processes smaller sub-volumes from the full build volume. The dimensions of these sub-volumes, which we shall now refer to as voxels, were determined based on the bounding box annotations. Given that the smallest annotated bounding boxes were approximately 15–20 pixels in dimensions, the fixed dimension for each voxel was set to 16×16×16. When processed across one volume with a stride of 8 (chosen to ensure fine details aren't missed), this results in the creation of around a million voxels. Going forward, since we will now be working at the voxel level, each voxel needs to be assigned a label: anomaly (1) or no anomaly (0). Inspired by the setup done in [30], a voxel is labeled as an anomaly (1) if 85% or more of its volume is covered by any annotated bounding box, and as no anomaly (0) if less than 25% is covered. The remaining voxels falling between these thresholds are considered ambiguous and are not used. A simple visual representation of this structure can be seen in Figure 5.1. With this labeling strategy, the task is framed as a binary classification problem. (Details on how the predicted anomaly voxels are merged to form the final bounding box prediction is explained in the upcoming Section 5.5)

Upon further analyzing the voxel-level data resulting from this setup, we observe a significant imbalance, as shown in Figure 5.2. The vast majority of voxels are labeled as normal

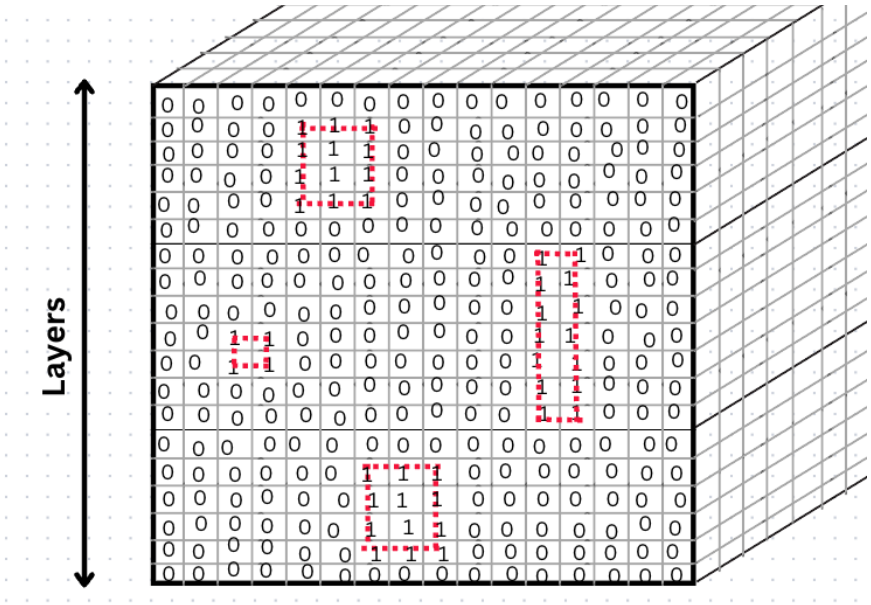

Figure 5.1: Voxel-level representation of a 3D volume, where each cube represents a 16 × 16 × 16 voxel. The red highlighted regions correspond to annotated bounding boxes, with voxel values set to 1 (anomaly) if sufficiently covered by annotations, and 0 (no anomaly) otherwise.

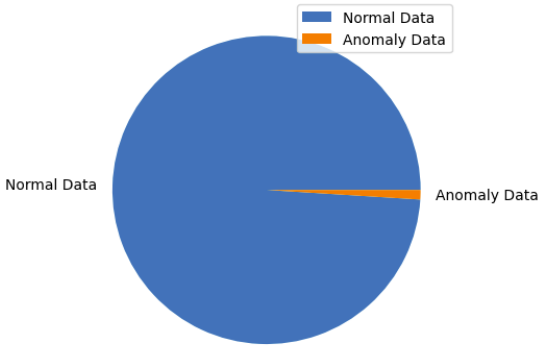

Figure 5.2: Voxel-level Data Proportion.

(0), while only a small fraction are classified as anomalies (1). This imbalance is important to consider, as it may affect the performance of classification models.

Since pixel intensities range from 0 to 255, the values were normalized to [0, 1] to avoid skewness in statistical measures. After looking at various statistical metrics of the pixel intensity like mean, median and variance, the metric that proved to be the most insightful is the average 95th percentile for each voxel. This is determined by calculating the 95th percentile pixel intensity for each layer (Z) and then taking the average (To account for temporal changes across the Z-axis). We observe distinct differences in their value ranges by plotting this density distribution split by normal and anomaly classes (Figure 5.3). The normal data tends to be distributed more towards the lower end of the intensity range, starting from the left and peaking around the center, whereas the anomaly data is more concentrated towards the higher end, spanning from the middle to the right. Despite this separation, there is a considerable overlap between the two distributions, which could be attributed to the sparsity and noise of the anomalies in the annotations as noticed in Figure 4.2.

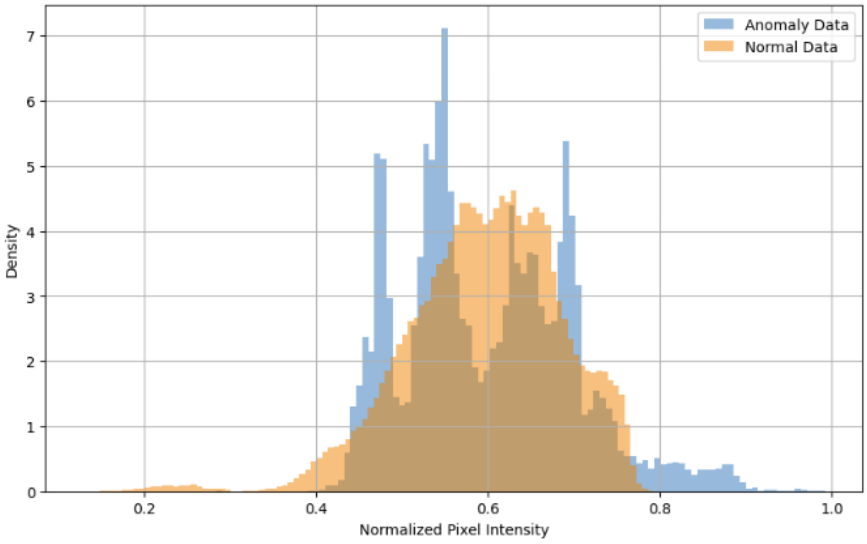

Figure 5.3: Voxel-level 95th Percentile Distribution on Raw Pixel Intensities.

This insight resulted in a simple yet informative modeling approach that serves as a useful baseline by classifying voxels based on their intensity characteristics, called "Pixel Thresholding".

5.2 Pixel Thresholding

Pixel thresholding in this work classifies each voxel individually based on an intensity-based metric. For each voxel, the metric is computed and compared against a predefined threshold. If the metric exceeds the threshold, the voxel is classified as anomalous, otherwise, it is considered normal.

As determined in the previous section, the average 95th percentile is used as the metric for evaluation. To identify a suitable threshold, we further analyzed the distribution of this metric in both normal and anomalous data in Figure 5.3. While there is some overlap between the two distributions, the anomaly data tends to show a heavier skew toward higher values. Based on this shift, we select a threshold informed by the upper tail of the normal data's distribution, such that values significantly beyond it are more likely to correspond to anomalies. However, the high degree of overlap between the normal and anomaly classes could result in poor performance. To address this, image processing techniques were experimented with that can help separate the foreground from the background. This would allow for a better separation of the two distributions, thus making the pixel thresholding approach more effective.

Image processing techniques, as discussed in Section 2.3 involve extracting important information from these raw pixels while ignoring the noise present in the image. Several filtering methods were initially tested to identify the most effective preprocessing step, and the best method was ultimately used in the final model pipeline. Different techniques were evaluated based on how well the distribution separates the normal and anomaly classes without losing valuable information from the images. Initially, a Gaussian blur [1]was applied which acts as a simple low-pass filter to smooth noise. Building on this, a Difference of Gaussian (DoG) [2] filter was applied, which highlights edges by subtracting two Gaussian-blurred versions of the image at different scales. While DoG improved contrast over the basic Gaussian

[1]https://en.wikipedia.org/wiki/Gaussian_blur
[2]https://en.wikipedia.org/wiki/Difference_of_Gaussians

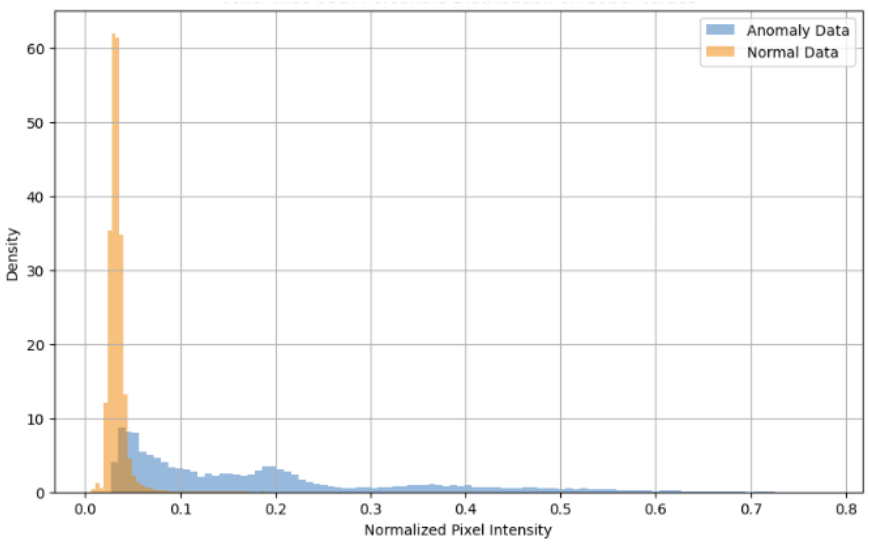

Figure 5.4: Voxel-level 95th Percentile Distribution on Sobel Pixel Intensities.

blur, it still lacked creating a good separation. Next, a Sobel operator was applied as explained in Section 2.3.1. Initially, the standard approach of applying a Gaussian blur before the Sobel filter was done. However, motivated by academic sources [34], a bilateral filter was tested as a replacement for Gaussian blur. Unlike Gaussian filtering, bilateral filtering preserves edges while smoothing noise, which proved especially beneficial in this case.

Lastly, Canny edge detection [3] was tested. It builds upon the core principle of the Sobel operator while adding several additional steps to improve the accuracy and robustness of edge detection. However, it led to an excessive loss of information. As a result, Canny edge detection was not suitable for this use case.

The combination of bilateral filtering followed by Sobel resulted in the best separation between normal and anomaly distributions, making it the preferred approach, as shown in Figure 5.4.

After this, the pixel thresholding process was applied. The only parameter to optimize is the threshold value and that is what was determined using the training data. The threshold that yielded the best performance was selected and subsequently applied to the test data.

5.3 Tree-Based Modeling

Discriminative modeling works by learning the decision boundary between classes by estimating the conditional probability distribution of the output variable given the input features $p(y/x)$. Given input x, output y (0 or 1), we directly learn a parametric model that predicts $p(y/x)$. The parametric models that will be used in this section are Random Forest and XGBoost as described in Section 2.4.2 and Section 2.4.3, respectively.

The model expects tabular data, so we first need to structure the data into the correct format before feeding it into the model. Each voxel becomes a single row in this table. For every voxel, we extract a set of characteristics that serve as the input variables, while the corresponding output variable is binary, either 1 or 0. The input variables consist of several statistical metrics that capture important information about each voxel. These include the coordinates, mean, median, variance and the 95th percentile of the voxel, as well as the average of the 95th percentile values, as described in Section 5.2. In addition to these, we apply a Sobel filter and extract the same metrics, since we found earlier that the Sobel filter improves

[3] https://pyimagesearch.com/2021/05/12/opencv-edge-detection-cv2-canny/

the separation between the two classes. This processing was applied to the training and test sets to create the final dataset used by the parametric model.

The performance of ML models usually depends on the selected hyperparameters. To optimize hyperparameters more systematically, we implemented Bayesian Optimization as discussed in Section 2.8.1. The objective function to maximize is the F1-score. The optimization was performed using 5 fold cross-validation with the *skopt* [4]package.

After modeling training, further calibration is needed to ensure the model is not overconfident, which can be the case when working with imbalanced datasets. Calibration adjusts the probability estimates so they better reflect the true likelihood of an event. For this, we used *Scikit-learn's CalibratedClassifierCV* [5] with Platt scaling [6]. This approach fits a logistic regression model to the output scores of the base classifier, transforming them into better calibrated probability estimates.

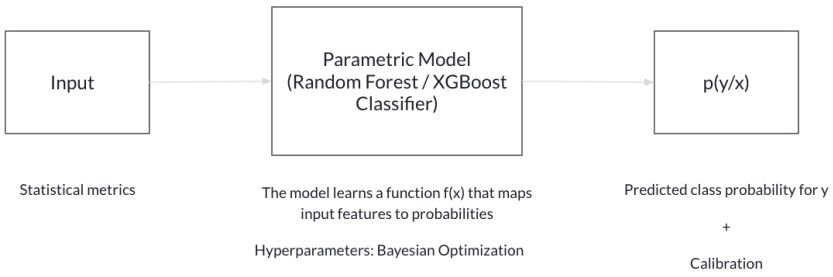

Figure 5.5: Structure of the Tree-based Modeling Pipeline

The probability estimates outputted by the model now need to have a threshold set to classify if it is 1 or 0. Different threshold are evaluated on the training set to find the best threshold value.

5.4 3D-ResNet

The next step was to explore whether DL approaches could further improve performance. Unlike models such as Random Forest, which rely on manually extracted features, DL models have the ability to automatically learn patterns. This opens up the possibility of uncovering hidden features that may not be easily captured by traditional methods.

Based on the review of the literature discussed in Section 3, ResNet-based architectures were among the most commonly used models for similar tasks. Additionally, due to the limited size of our dataset, transfer learning was a natural strategy to adopt. Pre-trained 3D ResNet models are available, which makes them suitable for transfer learning.

Therefore, ResNet was selected as the DL model to investigate in this work. The two 3D-ResNet variants used in this study are based on different pre-training datasets, which are discussed in the following subsections.

[4] https://scikit-optimize.github.io/stable/

[5] https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

[6] https://en.wikipedia.org/wiki/Platt_scaling

5.4.1 Kinetic-400

This dataset contains 400 human action classes, with at least 400 video clips for each action. Each clip lasts around 10 seconds. The actions are human focussed and cover a wide range of classes including human-object interactions like playing instruments, as well as human-human interactions like shaking hands [35]. While this dataset is completely different from the one used in this study, there have been approaches to utilize the 3D spatial information by initializing networks pre-trained on it [36]. It will be interesting to see how this performs on our data.

The model architecture is an 18-layer 3D ResNet model [37] available directly via the PyTorch package [7]. Edits were made to the input and output of the architecture to ensure its aligns with the data format and the expected output. For each voxel, all its pixel intensities and corresponding labels were stacked and converted into PyTorch tensors. Specifically, the voxels was stacked into a tensor of shape (N, 1, T, H, W) — where N is the number of voxels, 1 represents the single input channel (since the images are grayscale), T is the temporal dimension (which is treated as spatial depth in this context), and H, W correspond to the height, and width of each voxel (in our case, 16×16×16). These tensors were then loaded into a PyTorch DataLoader with a batch size of 128.

Different combinations of hyperparameters were attempted to improve the training performance of the model, these parameters are:

- Loss function: BCE or FL
- Optimizer: SGD or Adam
- Learning rate
- Weight decay
- Freezing the initial layers vs Fine-tuning all layers
- Layer-wise learning rate and weight decay
- Weighted Sampler: To address class imbalance by ensuring that minority class samples are more likely to be selected in each batch.

The model was trained in the GPU. The model outputs raw logit values, which are then passed through a sigmoid function to convert them into values between 0 and 1. As with the previous modeling approach, a range of threshold values is evaluated to determine the optimal threshold for classification.

5.4.2 MedicalNet (Med3D)

Med3D provides 3D models pre-trained on a large and diverse collection of medical imaging datasets, covering various organs and pathologies [36]. Figure 5.6 provides a visual glimpse into how the dataset may look. While Med3D still differs from the domain characteristics of additive manufacturing images, these medical imaging volumes have the closest structure to the powder bed dataset that was publicly available. Additionally, the early layers of the pre-trained model focus on learning to detect basic features such as edges, textures and shapes, which are transferable and beneficial for powder bed images.

A series of 3D-ResNet models pre-trained on 23 datasets are available and will be used. These include 3D-ResNet architectures with 18, 34 and 50 layers.

[7] https://pytorch.org/vision/0.20/models/generated/torchvision.models.video.r3d_18.html#torchvision.models.video.r3d_18

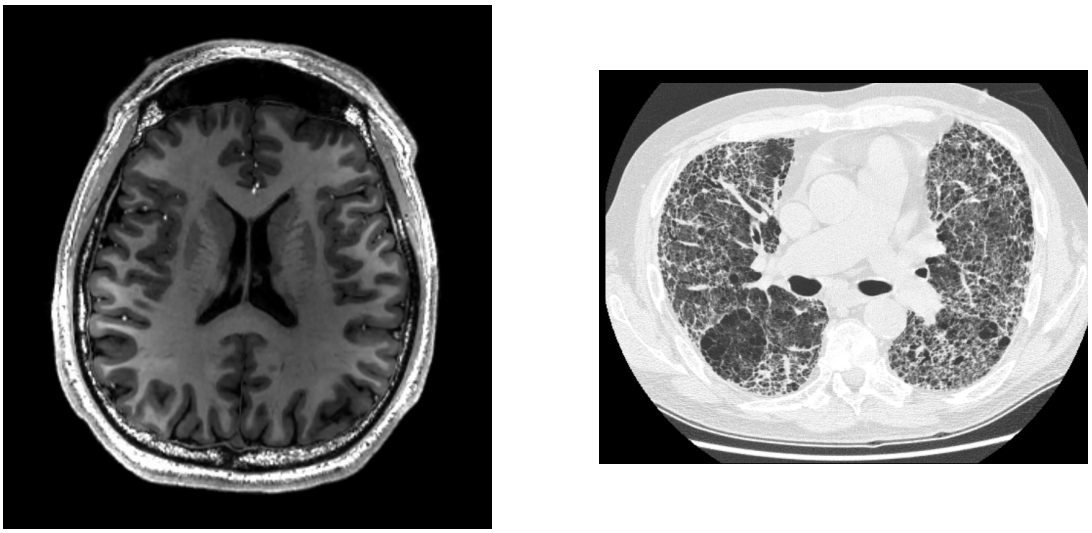

Figure 5.6: Example 2D slices from medical imaging — MRI of the brain (left) and CT of the lungs (right). Source: Wikipedia.

A similar setup, as discussed in Section 5.4.1, is applied here as well. The same hyperparameter configurations are also explored here along with the various architectures available (10, 34 and 50 layers 3D-ResNet).

Additionally, to assess the impact of pre-training, the base 3D-ResNet models without any pre-training are also tested for comparison.

5.5 Model Evaluation

The performance of the models is evaluated using the metrics introduced in Section 2.9. As mentioned previously, the modeling has been on a voxel level, so further pre-processing is needed to get the final prediction for evaluation. The following are the steps taken:

1. After the model predicts anomalous voxels across the volume, adjacent or overlapping predicted voxels are merged to produce localized 3D bounding boxes. This ensures that multiple close-by voxel predictions are consolidated into a single volumetric detection.

2. As discussed in Section 4, the ground-truth annotations are dense in some areas and often located adjacent to one another. In the absence of a strict rule to determine when to stop merging voxels, we apply a many-to-one matching strategy. In this approach, each ground-truth bounding box is allowed to match only once using the Dice coefficient. A match takes place if the Dice coefficient exceeds a defined threshold. To handle multiple matches, the matches are sorted by the Dice score and the highest scoring match is selected. However, a single predicted bounding box can be matched to multiple ground-truth boxes, provided it meets the Dice threshold for each. This many-to-one matching approach helps address the challenge of densely clustered annotations, while ensuring effective evaluation.

   Due to the 3D nature of the problem and the way we have set up the merging of voxels, a predicted bounding box might entirely enclose the ground truth but be significantly oversized, leading to a low Dice score, which can be misleading. To mitigate this, the Dice threshold was intentionally set to a low value of 0.01. A value enough to enforce some overlap, while still allowing the highest overlap prediction to be selected as the best match.

Additionally, Dice coefficient was used over IoU as Dice is less sensitive especially in 3D cases. In 3D, even slight differences in the spatial extent between predicted and ground truth bounding boxes can lead to a significantly lower IoU, which makes it a stricter metric. Dice coefficient is more forgiving based on how it is calculated as discussed in Section 2.9.2, making it a better choice in this situation.

3. From this matching result, we can determine the TP, FP and FN values. Using these, we can calculate Precision, Recall and F1 score to evaluate the performance of the model. These metrics were specifically chosen due to the imbalance in the dataset. In such scenarios, relying on accuracy can be misleading as a model that mainly predicts the majority class can achieve a high accuracy despite poor detection of actual defects (minority class). In contrast, precision and recall account for this imbalance and provide a clearer picture.

4. To analyze the impact of different confidence thresholds on model performance, the above metrics are calculated across different thresholds to determine the optimal value.

6 Results & Discussions

The following subsections are structured to present the optimized parameters obtained for each of the models introduced in the previous section, followed by a comparison and evaluation of their performance on the test set.

6.1 Pixel Thresholding

To identify an effective intensity threshold, a range of values between 0.05 and 0.3 was selected, as this is where a clear separation emerges in Figure 5.4. For each threshold level, the evaluation metrics were calculated on the training set as shown in Figure 6.1. As the threshold value increased, the precision improved while the recall reduced. This is expected, as higher thresholds tend to reduce false positives at the expense of missing some true positives. The F1 score and recall initially increased, but then plateaued after a certain point.

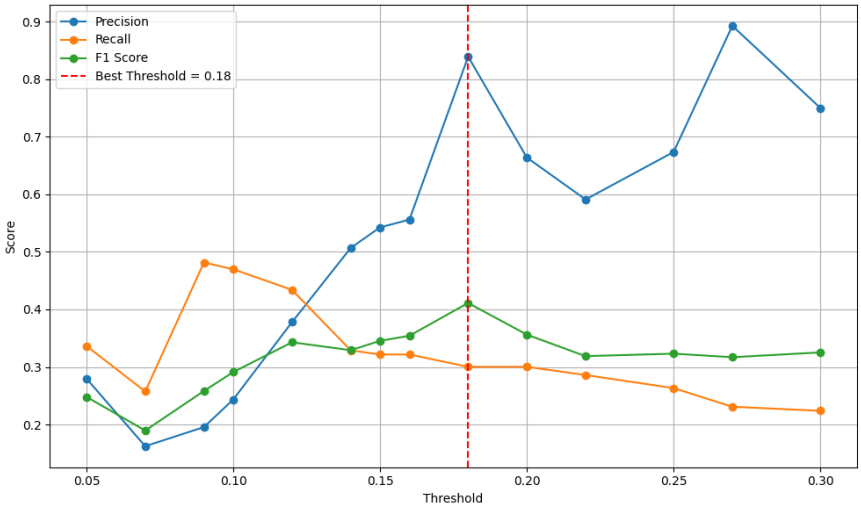

Figure 6.1: Pixel Thresholding: Model Metrics vs Pixel Intensity Threshold values on training set.

Among all these values, a threshold of 0.18 yielded the best performance based on the F1 while keeping the precision high. This value suggests that it effectively differentiates between normal and anomalous voxels.

6.2 Tree-Based Modeling

For the Random Forest modeling, with the help of Bayesian optimization, the best hyperparameters obtained are listed in Table 6.1.

Hyperparameter	Value
Maximum tree depth	None (unlimited)
Number of trees (estimators)	157
Minimum samples to split a node	6
Number of features per split	sqrt

Table 6.1: Selected hyperparameters for Random Forest

After selecting the optimal hyperparameters, the next step was was to determine an appropriate probability threshold for classifying a voxel as normal or an anomaly. This threshold defines the cutoff above which a predicted probability is considered a positive (anomaly). To avoid confusion, it is important to note that this is not a pixel intensity threshold (as used in pixel thresholding), but rather a classification probability threshold based on the model's output.

Using the training set, a range of probability thresholds from 0.1 to 0.9 was evaluated. Figure 6.2 shows that the metric scores remain consistent across majority of the threshold values. A significant drop to the recall and F1-score after a threshold of 0.7 is observed, indicating that the model became too conservative and began missing true positives. Since we see no change in the values between thresholds 0.4 and 0.7, a threshold of 0.5 was selected as the final cutoff. This was selected as it is the standard probability threshold often used in binary classification.

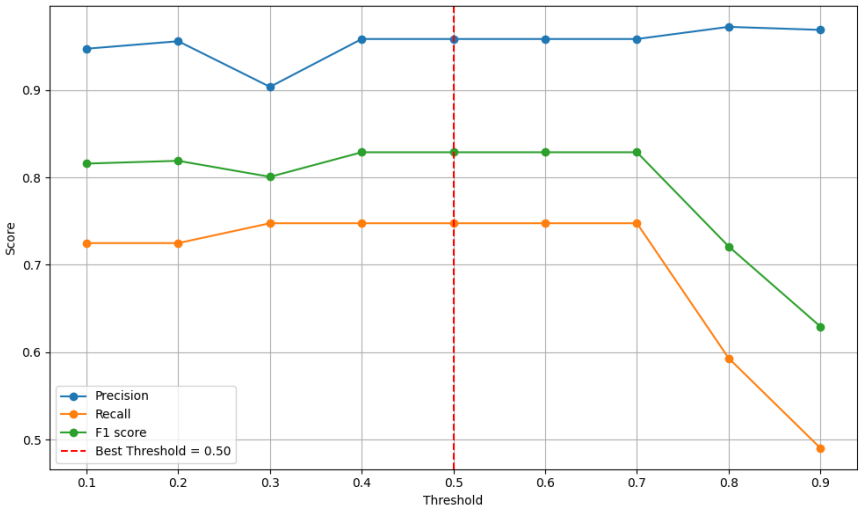

Figure 6.2: Random Forest: Model Metrics vs Probability Threshold values on training set.

Moving on to the XGBoost model, table 6.2 shows the best hyperparameters achieved.

Hyperparameter	Selected Value
Maximum tree depth	10
Number of trees (estimators)	196
Minimum child weight	1
Column sampling ratio per tree	0.7
Class imbalance weight	3

Table 6.2: Selected hyperparameters for XGBoost

A similar probability threshold analysis was conducted for the XGBoost model, as shown in Figure 6.3. Similar to the threshold evaluation for the Random Forest model, the XGBoost model showed a stable trend across thresholds and also a noticeable drop in both recall and F1 score beyond a threshold of 0.7. Based on this observation, a threshold of 0.6 was selected as the optimal cutoff.

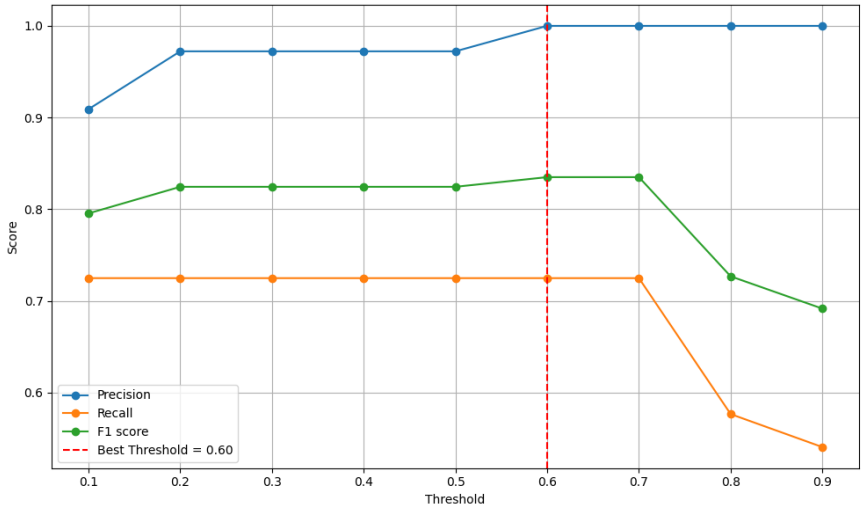

Figure 6.3: XGBoost: Model Metrics vs Probability Threshold values on training set.

6.3 3D-ResNet

Following the parameter search outlined in Section 5.4.1, various combinations of training settings were experimented with. The configuration that achieved the best performance on the training set is summarized in Table 6.3. This setup was used consistently across both pre-trained model variants.

Fine-tuning was applied to all layers of the network, as it outperformed both partial fine-tuning (only final layers) and an approach involving layer-specific learning rates. This suggests that, despite being pre-trained, the early layers still benefited from being fine-tuned to learn the characteristics of the dataset. One reason for this could be that the domain-specific features in the input data differ enough from the original pre-training dataset, requiring the full network to be adjusted for optimal performance. This is reasonable since the Kinetic-400 and MedicalNet datasets aren't exactly aligned with the target dataset.

Additionally, Focal Loss was selected over the weighted sampler as it performed better to address the class imbalance issue. However, using them together (i.e., Focal Loss with a weighted sampler) led to degraded performance. This may be due to redundancy, the

weighted sampler already increases the frequency of minority class examples during training, and combining this with Focal Loss, which further emphasizes harder, misclassified examples, likely led to overcompensation.

Training Parameter	Value
Batch size	128
Number of epochs	100
Loss function	Focal Loss
Optimizer	Adam
Learning rate	1×10^{-5}
Weight decay	1×10^{-5}

Table 6.3: Training configuration for the model

Different probability thresholds of the model outputs was experimented on the training set and Table 6.4 shows the best selected threshold for the different model variants.

Model	Pretraining Source	Selected Threshold
ResNet-18	Kinetics-400	0.6
ResNet-18	Med3D	0.7
ResNet-34	Med3D	0.6
ResNet-50	Med3D	0.7
ResNet-18	No pretraining	0.6
ResNet-34	No pretraining	0.8

Table 6.4: Selected probability thresholds for each ResNet variant and pretraining configuration.

6.4 Model Comparison

Based on the finalized model configurations described in the previous sections, each model was evaluated on the test set to assess its performance. Table 6.5 presents a comparison of the models using the evaluation metrics. Among all the models, Random Forest achieved the highest precision, indicating strong confidence in its positive predictions with fewer false positives. In contrast, XGBoost achieved the highest recall and F1 score, suggesting that it was the best model found.

Additionally, as seen in Figures 6.2 and 6.3, we observe that the metrics on the training set are often higher than those on the test set. This discrepancy is likely due to the limited test data which introduces higher variance, as evaluation was performed on only two 3D volumes.

Model	Pretraining Source	Precision	Recall	F1 Score
Pixel Thresholding	No pretraining	0.55	0.325	0.395
Random Forest	No pretraining	**0.833**	0.273	0.401
XGBoost	No pretraining	0.585	**0.350**	**0.438**
ResNet-18	Kinetics-400	0.314	0.170	0.152
ResNet-18	Med3D	0.477	0.170	0.233
ResNet-34	Med3D	0.257	0.220	0.233
ResNet-50	Med3D	0.200	0.170	0.154
ResNet-18	No pretraining	0.134	0.311	0.186
ResNet-34	No pretraining	0.564	0.170	0.157

Table 6.5: Performance comparison across models using average precision, recall and F1 score.

Among the DL models, the best overall performance was observed from ResNet-34 pretrained on Med3D, which achieved a higher recall compared to its ResNet-18 counterpart, despite both models having the same F1 score. While ResNet-18 (Med3D) offered slightly better precision, the improved recall of ResNet-34 (Med3D) suggests it was more effective at detecting a larger portion of true anomalies, which is an important factor in defect detection tasks. For this reason, ResNet-34 (Med3D) is considered the strongest deep learning configuration among those tested. Interestingly, increasing the model depth further to ResNet-50 resulted in a drop in performance. This may suggest that the deeper architecture introduced unnecessary complexity relative to the task and dataset, potentially leading to poorer performance. While deeper networks theoretically allow for better feature learning, their effectiveness depends on the availability of a large and diverse training dataset, which was not the case here. Another observation is that models pretrained on Med3D performed better, which aligns with expectations, as Med3D is derived from medical imaging volumes and is structurally closer to the powder bed dataset. On the other hand, Kinetics-400, being an action recognition dataset composed of natural videos, offered little relevant feature alignment. This highlights the importance of using pretraining sources that are well-matched to the target domain. Furthermore, models trained from scratch (no pretraining) performed worse, reinforcing the value of transfer learning.

A key observation emerging from this work is the superior performance of traditional approaches such as Pixel thresholding, Random Forest and XGBoost compared to deeper neural networks. When we look at the 2D slices of the images as shown in Section 4, the images are structurally simple, typically composed of two visual regions: background and foreground (anomaly signals). This suggests that the complexity of DL models may have been excessive for the characteristics of the dataset. Furthermore, the subtlety in some of the defects may not provide enough texture for DL networks to effectively learn meaningful representations.

DL models typically extract features in a hierarchical manner, combining low-level patterns into increasingly complex representations. But in this case, the anomalies are such that there's often nothing substantial to compose. That diminishes the potential benefits of hierarchical feature learning. This raises the question of whether such a strategy is even appropriate for this type of data. In contrast, statistical methods or machine learning models rely on predefined features, often statistical metrics which may be better suited to capture the relevant distinctions more efficiently.

Moreover, DL models operate over a vast parameter space. Their high flexibility allows them to model complex patterns, but in doing so, they also face a more complicated optimization landscape. ML models are more constrained and learns what is presented through the input features. This reduced flexibility can actually be beneficial in this context, given the relatively simple structures in the input.

Another critical aspect to consider is the nature of the dataset itself. As discussed in Section 4, the annotations are only available for the most notable error type (Part elevation). This anomaly lends itself well to classic image analysis techniques, which explains the strong performance of traditional methods. It is plausible that DL models would show more advantage when applied to more complex defect types, such as noise-like clusters or subtle streaks, as shown in Figure 2.3, which may be less suitable for handcrafted methods.

However, even with the best-performing models, the overall evaluation metrics remained relatively low. For instance, the highest recall and F1 Score achieved across all models were 0.350 and 0.438 respectively, which is still modest in the broader context of defect detection tasks. This suggests that the limitation may not lie solely in the modeling approach but elsewhere too. One factor could be the subtlety and sparsity of the annotations, where many anomalies are extremely small or faint, making this an inherently challenging problem to model and detect accurately. Another contributing factor could be the limited quantity and diversity of volumetric data for training and testing, which restricts the learning capacity of the models.

7 Future Work

Building upon the findings of this research, several directions can be explored to improve the performance and generalizability of 3D volumetric defect detection in the field of AM.

1. An immediate step would be to collect a larger and more diverse dataset. This would significantly benefit model training and offer better generalization.
2. Owing to the subtleties in the anomalies, obtaining high-quality annotations is time-consuming and resource-intensive. Exploring semi-supervised learning or active learning could be valuable.
3. Another avenue involves adopting more advanced object detection architectures. Models such as 3D Faster R-CNN and 3D adaptations of YOLO could be explored to better capture varying defect sizes and shapes within volumetric data.
4. Given the relatively strong performance of traditional models in this study, another interesting direction would be to enhance these models with more engineered features.

8 Conclusion

The thesis aimed to identify 3D bounding box defect detection techniques in volumetric powder bed images. Different approaches from simple and traditional ML models to DL models were explored. Each of the research questions is addressed below.

What model approaches are most effective in accurately predicting the 3D bounding boxes around the defects?

Based on the evaluation metrics, traditional approaches like Pixel thresholding, Random Forest and XGBoost performed better compared to DL architectures. This suggests that the additional complexity offered by DL models might be excessive and traditional approaches, with their relatively lower complexity, may be better suited for the characteristics of this dataset. Among DL models, transfer learning from domains with similar data characteristics improved the performance compared to models trained from scratch or pretrained on unrelated data, emphasizing the value of transfer learning.

What level of performance can these models achieve in defect detection?

The best model performance was obtained using XGBoost, with a Precision, Recall and F1 Score of 0.585, 0.350 and 0.438, respectively. Although the performance is modest in the broader context of defect detection tasks, this hints that the limitation might not just be with the modeling approach itself. Additionally, as the selection of the best model was performed on the same test data, the reported metrics likely overestimate true generalization performance, although the inflation should be small given the limited number of models compared.

Bibliography

[1] Reem Ashima, Abid Haleem, Shashi Bahl, Mohd Javaid, Sunil Kumar Mahla, and Someet Singh. "Automation and manufacturing of smart materials in additive manufacturing technologies using Internet of Things towards the adoption of industry 4.0". In: *Materials Today: Proceedings* 45 (2021), pp. 5081–5088.

[2] Tao Shen and Bo Li. "Digital twins in additive manufacturing: a state-of-the-art review". In: *The International Journal of Advanced Manufacturing Technology* 131 (2021), pp. 63–92.

[3] Mahdi Jamshid, Michael Kottman, Kirstie Snodderly, Jacob Williams, and Mohsen Seifi. *Strategic guide: additive manufacturing in-situ monitoring technology readiness*. Tech. rep. June 2023.

[4] Ritam Pal and Amrita Basak. "Linking powder properties, printing parameters, post-processing methods, and fatigue properties in additive manufacturing of AlSi10Mg". In: *Alloys* 1.2 (2022), pp. 149–179.

[5] Amir Mostafaei, Cang Zhao, Yining He, Seyed Reza Ghiaasiaan, Bo Shi, Shuai Shao, Nima Shamsaei, Ziheng Wu, Nadia Kouraytem, Tao Sun, Joseph Pauza, Jerard V. Gordon, Bryan Webler, Niranjan D. Parab, Mohammadreza Asherloo, Qilin Guo, Lianyi Chen, and Anthony D. Rollett. "Defects and anomalies in powder bed fusion metal additive manufacturing". In: *Current Opinion in Solid State and Materials Science* 26.2 (2022), p. 100974.

[6] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing, Global Edition*. Pearson Education, 2018.

[7] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Machine learning: a first course for engineers and scientists*. Cambridge University Press, 2022.

[8] Tianqi Chen and Carlos Guestrin. "XGBoost: a scalable tree boosting system". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016, pp. 785–794.

[9] Enzo Grossi and Massimo Buscema. "Introduction to artificial neural networks". In: *European journal of gastroenterology hepatology* 19 (Jan. 2008), pp. 1046–54.

[10] Abien Fred Agarap. *Deep learning using Rectified Linear Units (ReLU)*. 2019.

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

[12] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. *A comprehensive survey on transfer learning*. 2020.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: a large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.

[14] Diederik P. Kingma and Jimmy Ba. *Adam: A method for stochastic optimization*. 2017.

[15] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger. Vol. 24. Curran Associates, Inc., 2011.

[16] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. *Generalized intersection over union: a metric and a loss for bounding box regression*. 2019.

[17] Ling Xiao, Mingyuan Lu, and Han Huang. "Detection of powder bed defects in selective laser sintering using convolutional neural network". In: *The International Journal of Advanced Manufacturing Technology* 107 (2020), pp. 2485–2496.

[18] Muhammad Ayub Ansari, Andrew Crampton, Rebecca Garrard, Biao Cai, and Moataz Attallah. "A Convolutional Neural Network (CNN) classification to identify the presence of pores in powder bed fusion images". In: *International Journal of Advanced Manufacturing Technology* 120 (2022), pp. 5133–5150.

[19] Anna-Maria Schmitt, Christian Sauer, Dennis Höfflin, and Andreas Schiffler. "Powder bed monitoring using semantic image segmentation to detect failures during 3D metal printing". In: *Sensors* 23.9 (2023).

[20] Muhammad Ayub Ansari, Andrew Crampton, and Samer Mohammad Jaber Mubarak. "Enhanced detection of surface deformations in LPBF using deep convolutional neural networks and transfer learning from a porosity model". In: *Scientific Reports* 14 (2024), p. 26920.

[21] Imran Ali Khan, Hannes Birkhofer, Dominik Kunz, Lukas Drzewietzki, and Vasily Ploshikhin. "A random forest classifier for anomaly detection in laser-powder bed fusion using optical monitoring". In: *Materials (Basel)* 16.19 (Sept. 2023), p. 6470.

[22] Vivian Wen Hui Wong, Max Ferguson, Kincho H. Law, Yung-Tsun Tina Lee, and Paul Witherell. *Automatic volumetric segmentation of additive manufacturing defects with 3D U-Net*. 2021.

[23] Domenico Iuso, Soumick Chatterjee, Sven Cornelissen, Dries Verhees, Jan De Beenhouwer, and Jan Sijbers. "Voxel-wise segmentation for porosity investigation of additive manufactured parts with 3D unsupervised and (deeply) supervised neural networks". In: *Applied Intelligence* 54 (Oct. 2024), pp. 13160–13177.

[24] Antonio Criminisi, Duncan Robertson, Ender Konukoglu, and Jamie Shotton. "Regression forests for efficient anatomy detection and localization in computed tomography scans". In: *Medical Image Analysis* 17.8 (2013), pp. 1293–1303.

[25] Romane Gauriau, Rémi Cuingnet, David Lesage, and Isabelle Bloch. "Multi-organ localization with cascaded global-to-local regression and shape prior". In: *Medical Image Analysis* 23.1 (2015), pp. 70–83.

[26] Dennis Hartmann, Dominik Müller, Iñaki Soto-Rey, and Frank Kramer. *Assessing the role of random forests in medical image segmentation*. 2021.

[27] Bob D. de Vos, Jelmer M. Wolterink, Pim A. de Jong, Max A. Viergever, and Ivana Išgum. "2D image classification for 3D anatomy localization: employing deep convolutional neural networks". In: *Medical Imaging 2016: Image Processing*. Ed. by Martin A. Styner and Elsa D. Angelini. Vol. 9784. International Society for Optics and Photonics. SPIE, 2016, 97841Y.

[28] Rens Janssens, Guodong Zeng, and Guoyan Zheng. "Fully automatic segmentation of lumbar vertebrae from CT images using cascaded 3D fully convolutional networks". In: *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. 2018, pp. 893–897.

[29] Anjany Sekuboyina, Alexander Valentinitsch, Jan S. Kirschke, and Bjoern H. Menze. "A localisation-segmentation approach for multi-label annotation of lumbar vertebrae using deep nets". In: *CoRR* abs/1703.04347 (2017).

[30] Ziming Qiu, Jack Langerman, Nitin Nair, Orlando Aristizabal, Jonathan Mamou, Daniel H. Turnbull, Jeffrey Ketterling, and Yao Wang. "Deep BV: a fully automated system for brain ventricle localization and segmentation in 3d ultrasound images of embryonic mice". In: *IEEE Signal Processing in Medicine and Biology Symposium (SPMB)* 2018 (Dec. 2018), 10.1109/SPMB.2018.8615610.

[31] Shuo Han, Aaron Carass, Yufan He, and Jerry L. Prince. "Automatic cerebellum anatomical parcellation using U-Net with locally constrained optimization". In: *NeuroImage* 218 (2020), p. 116819.

[32] Xuanang Xu, Fugen Zhou, Bo Liu, Dongshan Fu, and Xiangzhi Bai. "Efficient multiple organ localization in CT image using 3D region proposal network". In: *IEEE Transactions on Medical Imaging* 38 (Jan. 2019), pp. 1885–1898.

[33] Tomoki Uemura, Janne J. Näppi, Toru Hironaka, Hyoungseop Kim, and Hiroyuki Yoshida. "Comparative performance of 3D-DenseNet, 3D-ResNet, and 3D-VGG models in polyp detection for CT colonography". In: *Medical Imaging 2020: Computer-Aided Diagnosis*. Ed. by Horst K. Hahn and Maciej A. Mazurowski. Vol. 11314. International Society for Optics and Photonics. SPIE, 2020, p. 1131435.

[34] Carlos Tomasi and Roberto Manduchi. "Bilateral filtering for gray and color images". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 839–846.

[35] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. *The kinetics human action video dataset*. 2017.

[36] Sihong Chen, Kai Ma, and Yefeng Zheng. *Med3D: transfer learning for 3D medical image analysis*. 2019.

[37] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. "A closer look at spatiotemporal convolutions for action recognition". In: *CoRR* abs/1711.11248 (2017).