Comparative additive manufacturing defect prediction accuracy with a few transfer learning implementations of deep learning models

Anamol Bajpai, *Srinivasa Prakash* Regalla*, *Subhendu Kumar* Sahoo, and *Pranav* Yenishetti

Birla Institute of Technology and Science Pilani, Hyderabad Campus, Hyderabad, Telangana, India

Abstract. This paper addresses the problem of a comprehensive quality assurance strategy for additively manufactured components with integrated in-situ inspection and artificial intelligence and machine learning (AIML) models. A custom test setup was created around a fused filament fabrication (FFF) 3D printing apparatus, incorporating sensors to capture real-time data concerning part quality. This image data was harnessed to formulate an AI/ML dataset for training a Convolutional Neural Network (CNN). A robust framework for predicting defects in real-time during the additive manufacturing process and validating the accuracy of AIML predictions has been presented. A test setup, generating a varied dataset, crafting AI/ML models, and optimizing the AI/ML model for precise defect prediction has been made. The proposed methodology's practical applicability and potential to redefine quality assurance in additively manufactured parts have been presented. Results were compared between the Matlab AlexNet pre-trained model and the user-designed model on Google Colab, which has the capability of hyperparameter tuning. Performance parameters, including accuracy, loss, precision, and recall, were plotted over epochs to analyze the model's merit after training. With the addition of hyperparameter tuning to the AlexNet, the best model was chosen as per the training accuracy. Much better accuracies were observed in the earlier epochs, with the initial loss being highly reduced compared to the non-hyperparameter-tuned model. The ResNet50 model, which did not have hyperparameter tuning capabilities, produced less accurate results. On the other hand, the loss was much lower with ResNet50 compared to both the AlexNet Matlab model and the AlexNet model on Google Colab without hyperparameter tuning. ResNet50, at the same time, gave accuracy comparable to that of the AlexNet Model with hyper-parameter tuning.

1 Introduction

Additive manufacturing (AM) of polymers and metals is rapidly becoming a game-changing technology for multiple sector of industry including aerospace and automotive. AM involves rapidly generating the physical component true to its computer-aided design (CAD) solid

* Corresponding author : regalla@hyderabad.bits-pilani.ac.in

model using a novel layer-by-layer building strategy. The AM technology, although it is geometry-agnostic and a rapid manufacturing process, still has some limitations, such as the products being prone to defects, primarily three of which our study aims to deal with:

- Lack of Fusion: This defect is mainly due to a non-sintered area or improper fusion between successive tracks or layers. The leading cause of this effect is insufficient energy.
- Spike Defect: This defect involves the formation of thin, protruding structures on printed objects, typically caused by factors like improper print parameters, material properties, or inadequate support structures, affecting both appearance and functionality.
- Mismatch Defect: This defect occurs when printed layers fail to align correctly, resulting in irregularities or gaps in the final object's surface, often caused by issues such as printer misalignment, material shrinkage, or inadequate cooling, impacting dimensional accuracy and structural integrity.

The dataset comprises images categorized into three classes: "spike," "mismatch," and "lof," with each class containing 25 images, totaling 75 images. These images represent instances of defects in additive manufacturing processes. On navigating inside the “Dataset_Defects” folder, three folders can be found containing images of the defect the folder is named after. The data split for further stages was done on an 80:20 split between training and validation.

By training a Convolutional Neural Network (CNN) on diverse datasets using transfer learning techniques, the study aims to enable AI-driven defect prediction.

Pre-trained machine learning models are robust because they have already been trained using large amounts of data for one type of classification can be adapted to classify newer types of entities using the concept of transfer learning. This technique saves training a new CNN model from scratch, leveraging the knowledge gained from solving one classification problem to handle a different number and type of classes. The timeline and the various phases for the project have been shown above. Each step and architecture has been further described in the report.

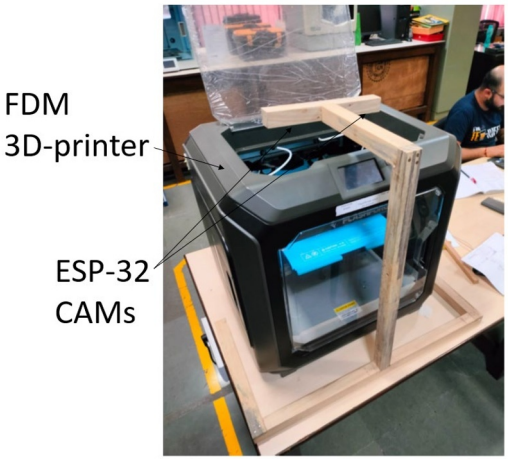
The transfer learning methodology was previously used by adapting the well-known AlexNet CNN to classify the AM defects in fused deposition modeling with a good accuracy of up to 95% [1]. The well-known image recognition CNN, VGGnet, was adapted to predict the microstructure and mechanical properties of laser powder bed fusion (LPBF) made stainless steel parts [2] and the geometric accuracy of parts [3]. A long-term short memory (LTSM) type machine learning model was employed to predict the tensile strength of FDM parts [4], whereas a support vector machine (SVM) model was used to predict the process performance of AM [5]. Many further extended functions that the adaptation of AIML can fetch in the process improvement of AM are already being recognized [6].

The study of the existing literature showed that the application of AIML in the process control and improvement of AM technologies is gradually picking up momentum. Although a few ML models have been employed as adaptations of existing CNN models, modifying multiple well-proven CNN models to fit the defect prediction in FDM and studying their comparative performance was not attempted, which is precisely the objective of the present work. Especially, the ResNet-50 was adapted to address the defection prediction in AM, which is a novel step in the present work. Two different existing and robust CNN models, namely, the AlexNet and ResNet-50, were adapted using the transfer learning technique to predict the defects of lack of fusion, spike, and mismatch in FDM in the present work. Their prediction accuracies have been compared. Since all of the specific changes needed to adapt a well-trained pre-existing CNN architecture, AlexNet, on a proprietary Matlab™ platform to the current problem through transfer learning were previously explained by the authors’

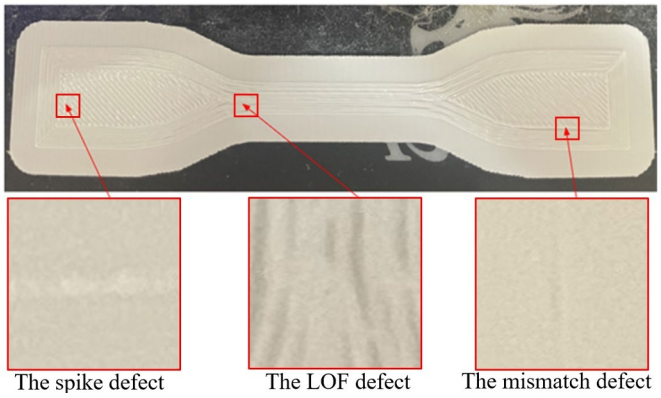
previous paper [1], the description of methodology in the present paper lays greater emphasis on how to use an open source platform of Google Colab using Python coding.

2 Materials and methods

A test rig was designed, fabricated, and used to capture the images of the layers during the part building on the Flash Forge Creator Pro 3 - 3D Printer (Figure 1). The test rig enabled placing two digital cameras on the horizontal flange with their view axes perpendicular to the build plate. Two ESP-32 CAM digital cameras were used to capture layer images. The total number of images of the layers captured by these cameras was directly sent to a Google folder for further processing by the CNN.



(a)



(b)

Fig. 1. The experimental work of capturing the images of the FDM layers (a) The test rig, and (b) A typical image of a layer with defects.

Mounting the Google Drive onto Google Colab enabled seamless integration of the AIML environment, facilitating access to image files stored on Google Drive directly within the Colab Notebook. By importing the necessary module from the Google Colab library and mounting Google Drive to a specified directory ('/content/drive'), users can authenticate and

authorize access to their Google Drive account, establishing a connection between the Notebook and their storage. This integration streamlines collaboration, enables access to datasets, and simplifies the storage of project-related files, enhancing workflow efficiency. Mounting Google Drive is a fundamental step in setting up the development environment, as it provides easy access to essential resources in a seamless way.

The python scripts developed in the present work for transfer learning of AlexNet and ResNet50 networks began with setting the path to the dataset directory. The number of images in the dataset is then estimated and declared. The script extracts class names from the subdirectories, excluding any checkpoints, and prints both the class names and the total number of classes. This information provides a quick overview of the dataset's composition. Finally, it prints the content of the dataset directory, offering an idea of the structure of the data. These steps are crucial for understanding the dataset's size, class distribution, and organization, which are essential considerations in the initial stages of a machine-learning project.

The script prepares image data for training and validation using TensorFlow's ImageDataGenerator. Initially, it sets parameters such as batch size, image dimensions, and steps per epoch. The glob library is imported for file path handling, and train_test_split from sklearn.model_selection is utilized for splitting data into training and validation sets. The ImageDataGenerator is then initialized to preprocess and augment image data. Images are rescaled to a range of [0,1] to facilitate model training. Additionally, a validation split of 20% is specified to allocate a portion of the data for validation. Two data generators are created: one for training data (train_data_gen) and another for validation data (val_data_gen). These generators flow images from the specified directory (data_dir_str) in batches, shuffling the training data while keeping the validation data unchanged. The target_size parameter ensures images are resized to match the required input shape for the model. Finally, the classes parameter is set to ensure that images are classified according to the provided class names. Overall, this code efficiently prepares image data for training and validation, which is crucial for training machine learning models, especially for image classification tasks.

We define a model-building function for AlexNet architecture [7] using keras_tuner. The keras_tuner is a hyperparameter tuning library for Keras, which is an open-source artificial neural network library available in Python. It enables developers to automatically search for the optimal hyperparameters for their deep learning models. The function build_alexnet_model takes a hp (hyperparameters) argument from the keras_tuner and constructs the AlexNet model architecture.

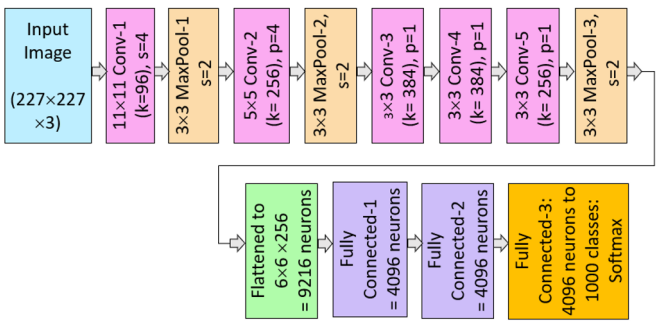


Fig. 2. The Architecture of the standard AlexNet CNN.

The AlexNet model's structure consists of several convolutional layers, batch normalization, max-pooling layers, and fully connected layers (Figure 2). The hyperparameters such as the number of filters in each convolutional layer (conv1_filters to

conv5_filters), the number of units in the fully connected layers (dense_units), and the learning rate (learning_rate) are tuned using the keras_tuner.

A convolutional layer in a CNN applies learnable filters to input data, producing feature maps that capture spatial hierarchies of features. Key characteristics include local connectivity, shared weights, and optional pooling operations. Convolutional layers enable effective feature extraction in tasks like image recognition and computer vision.

The learning rate, batch size, and the number of hidden layers are the hyperparameters that are set before the training of the CNN begins. Tuning these hyperparameters manually can be time-consuming and requires extensive experimentation. The keras_tuner automates this process by efficiently exploring the hyperparameter space and finding the combination that maximizes the performance of the model on a validation dataset.

The keras_tuner provides several algorithms for hyperparameter optimization, including Random Search, Hyperband, and Bayesian Optimization. It seamlessly integrates with Keras, allowing users to define their model architecture using the Keras API and specify the hyperparameters desired to be tuned.

Hyperparameter tuning is being implemented by Hyperband, which is a state-of-the-art algorithm for hyperparameter optimization that combines random search with successive halving. It iteratively evaluates different sets of hyperparameters, training models for a few epochs and eliminating poorly performing configurations, while allocating more resources to promising configurations. The key parameters for the Hyperband tuner instantiation are as follows:

- ‘objective’: Specifies the metric to optimize during hyperparameter tuning. In this case, 'val_accuracy' is chosen, indicating that the tuner will aim to maximize the validation accuracy.
- ‘max_epochs’: Sets the maximum number of epochs to train each model during a single iteration of Hyperband.
- ‘factor’: Determines the reduction factor for the number of epochs and the number of models considered in each iteration of Hyperband.
- ‘directory’ and ‘project_name’: Specify the directory where the tuner saves its logs and checkpoints, as well as the name of the project.

The best hyperparameters found during the hyperparameter search conducted with the keras_tuner are retrieved. The “tuner.get_best_hyperparameters(num_trials)” function retrieves the best hyperparameters discovered during the hyperparameter search. The num_trials parameter stipulates the upper limit of the trials to consider when searching for the best hyperparameters. In the present work, the upper limit was set to 1000 trials. If fewer trials were conducted during the search, it will consider all available trials. This index accessor [0] is used to access the first (best) set of hyperparameters from the list of best hyperparameters returned by get_best_hyperparameters(). If multiple sets of hyperparameters are tied for the best performance, only the first one encountered during the search is returned. The variable “best_hps” stores the best set of hyperparameters retrieved from the tuner.

The model is built using the best hyperparameters found during the hyperparameter search conducted with the keras_tuner, and then it is trained on the full dataset. The “model = tuner.hypermodel.build(best_hps)” step constructs a new instance of the model using the best hyperparameters (best_hps) obtained from the tuner. The tuner.hypermodel refers to the hypermodel used by the tuner, and the “.build(best_hps)” constructs the model with the specified hyperparameters. The step “history = model.fit(...)” trains the model on the full dataset using the fit() method. It specifies the train_data_gen, which indicates the training data, the val_data_gen, which indicates validation data, the number of epochs (epochs), steps per epoch (steps_per_epoch), and batch size (batch_size). As the training proceeds, the

model's parameters are updated based on the provided data, optimizing its performance according to the specified loss function and metrics.

The optimizing function, Adam, was used to compile the model with a dynamically chosen rate of learning, which is suggested by the tuner. The loss function is based on cross-entropy and is suitable for multi-class classification tasks, and estimation of accuracy, precision, recall, sensitivity, specificity, and Area Under the ROC Curve (AUC), which are specified for model evaluation during training. The Adam optimizer is a widely used adaptive learning rate optimization algorithm in deep learning. By incorporating momentum, Adam accelerates gradient descent by accumulating an exponentially decaying average of past gradients, enhancing the efficiency and convergence speed of training.

Categorical cross-entropy works as a loss function in multi-class classification tasks such as the present one, and it quantifies the dissimilarity between true and predicted class distributions by calculating the cross-entropy loss for each instance. Minimizing this loss during training improves model accuracy by encouraging accurate class predictions based on learned probability distributions.

Data pre-processing was done using the "Image Batch Processor" app on MATLAB™. Similar to the model on Colab, the data split was again 80:20, to better compare the two models. The pre-trained network was loaded, and the output layers were modified according to the required classes. The SoftMax layer is critical for converting the output of a neural network into a probability distribution, making it easier to interpret and use for classification purposes. The classification layer tailors the network's output to the particular requirements of the classification task, producing either class scores or probabilities. The choice of activation function in the classification layer depends on the nature of the classification problem, whether binary or multi-class.

The transition from AlexNet to ResNet-50 [8] marks a pivotal moment in the evolution CNNs. The AlexNet, devised in 2012, revolutionized the field with its pioneering use of deep learning techniques. It featured an eight-layer architecture incorporating novel concepts like ReLU activation functions, local response normalization, and dropout regularization. This breakthrough led to a dramatic improvement in image classification accuracy, winning several prestigious challenges. Subsequent advancements saw the emergence of architectures like VGG (2014) and GoogLeNet (Inception v1) (2014), which further pushed the boundaries of performance and computational efficiency. However, it was ResNet (2015) that truly transformed the landscape (Figure 3).

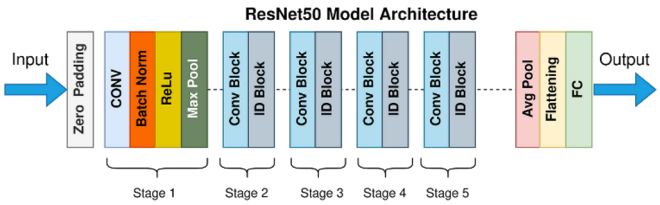


Fig. 3. The architecture of the standard ResNet CNN.

ResNet introduced the concept of residual connections, or skip connections, which allowed for training of exceptionally deep networks (up to 152 layers) without encountering the vanishing gradient problem. ResNet-50, a variant of this architecture, comprised 50 layers and demonstrated unparalleled accuracy on the ImageNet dataset, surpassing all previous benchmarks. This evolution from AlexNet to ResNet-50 underscores the iterative nature of scientific progress, with each architectural iteration building upon the successes and innovations of its predecessors to achieve new heights of performance and scalability in image classification tasks.

For the current model for our study, the data preprocessing has been done in the same way as in the AlexNet model. A pre-trained ResNet50 model is loaded and customized for a specific image classification task.

Firstly, the ResNet50 model, which was pre-trained on the ImageNet dataset, was loaded using the ResNet50 function from the Keras library. The weights parameter is set to 'imagenet' to load the weights trained on ImageNet and include_top is set to False to exclude the fully connected layers at the top of the network. Additionally, the input_shape parameter is specified to match the dimensions of the input images in the dataset.

Next, the loaded ResNet50 model's layers were frozen using the base_model.trainable = False statement. Freezing the layers prevents their weights from being updated during training, preserving the pre-trained weights.

A custom model architecture is then constructed using the tf.keras.Sequential API. The base ResNet50 model is followed by a GlobalAveragePooling2D layer, which averages the spatial dimensions of the feature maps output by the ResNet50 model. This reduces the spatial dimensions to a vector, effectively summarizing the extracted features. Finally, a Dense layer with a softmax activation function is added to produce class probabilities for the image classification task.

The compile method is employed to configure the model, selecting 'adam' as the optimizer and 'categorical_crossentropy' as the loss function, ideal for multi-class classification tasks. Furthermore, various evaluation metrics are incorporated, encompassing accuracy, precision, recall, sensitivity at specificity, specificity at sensitivity, and area under the ROC curve (AUC-ROC), ensuring comprehensive assessment of the performance of the model on the validation dataset.

Fig. 4. Accuracy variation with epocks in Matlab based AlexNet.

3 Results and discussion

The training history of a neural network model is visualized using Matplotlib. The performance parameters of accuracy, loss, precision, and recall are plotted over epochs to analyze the model's performance after training.

On observing the AlexNet Matlab model and the corresponding Colab model without hyperparameter Tuning (Figure 4). Both models exhibited training accuracy surpassing 95% by the 40th epoch. Initially, the training loss for the MATLAB™ model was notably high, gradually approaching zero after the 20th epoch.

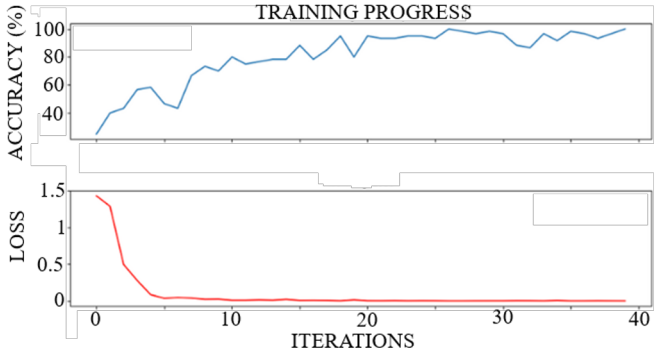


Fig. 5. Accuracy and training loss variation in the Python based Google Colab AlexNet.

Conversely, the Colab model displayed a much lower initial loss, nearing zero by the 5th epoch. Both models underwent 80 iterations: the MATLAB™ model completed 80 epochs, while the Colab model reached 40 epochs with two iterations per epoch (Figure 5).

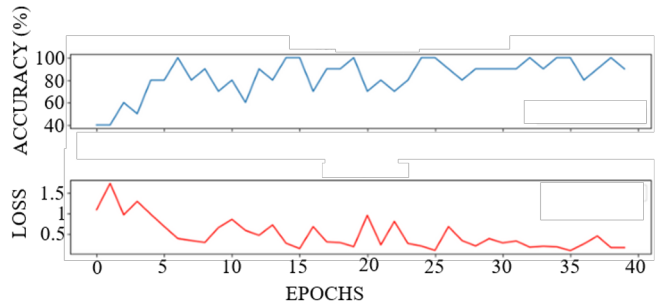


Fig. 6. Accuracy and training loss variation in the Python based Google Colab AlexNet with Hyperparameter tuning. Hyperparameter tuning leads to early convergence and smoother loss reduction.

With the addition of Hyperparameter tuning to the AlexNet, the best model was chosen as per the training accuracy; much better accuracies were observed in the earlier epochs, with the initial loss being highly reduced compared to the non-hyperparameter-tuned model (Figure 6).

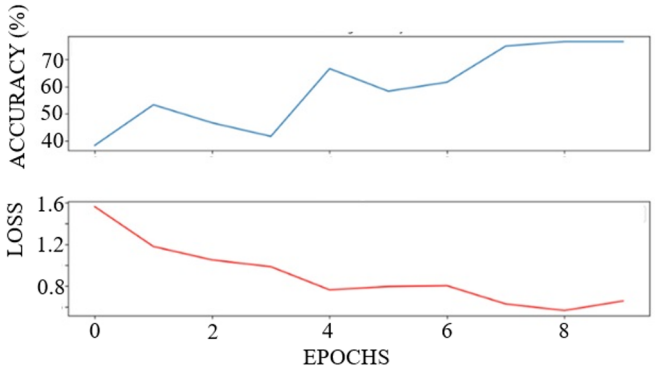


Fig. 7. Accuracy and training loss variation in the Python based Google Colab ResNet-50 model.

The ResNet50 Model, which does not have Hyperparameter tuning capabilities added, isn't able to produce many accurate results for the initial epochs, which might be because we are dealing with a very small size dataset and a very low number (10) of epochs ran (Figure 7). Still the loss was much lower, compared to the MATLAB's AlexNet model and the AlexNet model on Colab without HyperParameter tuning and comparable to the AlexNet model with Hyperparameter tuning.

The Dataset that was used to train the above given models was able to provide us dependable results that will help us to compare and further improve the accuracies of the models and with the use of architecture. The results can be more concrete with the use of a much larger dataset, thus contributing to a much more robust and fine-tuned model.

4 Conclusions

Based on the work done and results obtained, the following conclusions can be drawn. The Dataset that was used to train the above models was able to provide dependable results that helped us to compare and further improve the accuracy of the models. The results can be more concrete with the use of a much larger dataset, thus contributing to a much more robust and fine-tuned model. The importance of hyperparameter tuning in decreasing loss while increasing the model's accuracy during the earlier epochs, and the improvement of the model over the AlexNet model on MATLAB™. ResNet50, being reputed as a proven and more modern architecture, can be made to work better in the current problem, having a bigger dataset, further giving better accuracies. Since the methodology presented in the present work employed an in-situ image capture process using a normal camera applied to each layer as the part is being built on the FDM machine, it is currently limited to surface defects. The accuracy of the defect prediction can be further enhanced and expanded to defects that evolve with time as the layer-by-layer manufacturing progresses, and internal to the part, by adding more sensors, such as a microCT scanning system in the test rig. Our effort to expand the dataset by incorporating multiple cameras with their appropriate positioning around the machine and their time lag matching is in progress. By interlinking with the extrusion head path plan in this effort, we will be able to overcome the hindrance posed by the extrusion head to the imaging view direction of the cameras.

References

1. Regalla S.P., Kaushal A., Khetan S. (2023). Machine Learning (ML) Based Prediction of Defects in Extrusion-Type Additively Manufactured Parts. Advances in Science and Technology, Vol. 130, pp. 19-27, https://doi.org/10.4028/p-8EfgQY.
2. Herriotta C., Spear A.D. (2020). Predicting microstructure-dependent mechanical properties in additively manufactured metals with machine- and deep-learning methods. Computational Materials Science, Vol. 175, 109599. https://doi.org/10.1016/j.commatsci.2020.109599
3. J. Francis, L. Bian, "Deep Learning for Distortion Prediction in Laser-Based Additive Manufacturing using Big Data," Manufacturing Letters 20 (2019) 10–14. https://doi.org/10.1016/j.mfglet.2019.02.001
4. Zhang J. , Wang P. , Gao R. X. (2019). Deep learning-based tensile strength prediction in fused deposition modeling. Computers in Industry, Vol. 107, 11–21. https://doi.org/10.1016/j.compind.2019.01.011
5. Aoyagia K., Wanga H., Sudob H., Chibaa A. (2019). Simple method to construct process maps for additive manufacturing using a support vector machine. Additive Manufacturing, Vol. 27, 353–362. https://doi.org/10.1016/j.addma.2019.03.013
6. Wang C., Tan X.P., Tor S.B., Lim C.S. (2020). Machine learning in additive manufacturing: State-of-the-art and perspectives. Additive Manufacturing, Vol. 36, 101538. https://doi.org/10.1016/j.addma.2020.101538
7. Bangar, S. (2022, June 24). Alexnet Architecture explained. Medium.
8. https://medium.com/@siddheshb008/alexnet-architecture-explained-b6240c528bd5
9. Mukherjee, S. (2022, August 18). The annotated resnet-50. Medium. https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758