KLE Technological University

Creating Value
Leveraging Knowledge

School of
Computer Science and Engineering

Mini Project Report

on

Inspection for Defect Detection using Thermal Images

By:

1. **Akshay Joshi** USN: 01FE20BCS102
2. **Kartik Kalal** USN: 01FE20BCS116
3. **Dhiraj Bhandare** USN: 01FE20BCS117

Semester: V, 2022-2023

Under the Guidance of

Dr. Uma Mudenagudi

Dr. Ujwala Patil

K.L.E SOCIETY'S
KLE Technological University,
HUBBALLI-580031
2022-2023

SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that project entitled **"Inspection for Defect Detection"** is a bonafide work carried out by the student team of **"Akshay Joshi (01FE20BCS102), Kartik Kalal (01FE20BCS116) and Dhiraj Bhandare (01FE20BCS117)"**. The project report has been approved as it satisfies the requirements with respect to the mini project work prescribed by the university curriculum for BE (V Semester) in School of Computer Science Engineering of KLE Technological University for the academic year 2022-2023.

Dr. Ujwala Patil	**Dr. Meena S. M**	**Basavraj S A**
Guide	**Head of School**	**Registrar**

External Viva:

Name of Examiners **Signature with date**

1.

2.

ACKNOWLEDGMENT

The sense of contentment and elation that accompanies the successful completion of our project and its report would be incomplete without mentioning the names of the people who helped us in accomplishing this.

We are indebted to our guide Dr. Uma Mudenagudi and Dr. Ujwala Patil who were nothing but a constant source of enthusiasm and whose profound guidance, valuable suggestions, and beneficent direction was mainly responsible for us to complete this project.

We take this opportunity to express our deep sense of gratitude and sincere thanks to our Head Dr. Meena S. M. for her tremendous source of inspiration and help in challenging our effort in the right direction.

Last but not least we like to thank all the course faculty, teaching and non-teaching staff for helping us during the project.

Akshay Joshi
Kartik Kalal
Dhiraj Bhandare

ABSTRACT

In the manufacturing of steel sheets, ceramic tiles and many more mechanical products defects such as internal holes, pits, abrasions and scratches arise due to failure in design or fault in the machine. In industrial process of manufacturing, one of the most important tasks when it comes to ensuring the proper quality of the finished product. Workers are trained to identify complex surface defects. However, it is very time consuming, inefficient, and can contribute to a serious limitation of the production capacity. And some of the internal defects may be not identifiable to naked eyes. So Automatic defect-detection technology has obvious advantages over manual detection it not only adapts to an unsuitable environment but also works in the long run with high precision and efficiency. Traditionally, classical machine-vision methods have been applied to automate visual quality inspection processes, however, deep-learning based algorithms have started being employed. Large capacity for complex features and easy adaptation to different products and defects without explicit feature hand-engineering made deep-learning models well suited for industrial applications. Transfer learning has become increasingly popular in recent years due to the success of deep learning models, which can be fine-tuned for a new task using transfer learning. This allows the use of large, pre-trained models that have been trained on a large amount of data, which can improve the performance of the model on the new task. In this work, we have come up with a deep learning model that uses concepts of transfer learning and fine tuning.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we discuss about defect detection and the importance of inspecting products for its defects that occur during industrial manufacturing processes. It also discusses the recent advancements in the field of deep learning and how a deep learning approach can be implemented to automate the process of defect detection. Further we list the objectives of the of our work. At the end, we discuss several available approaches and give an insight about our approach towards the inspection of defect detection.

1.1 Motivation

Defect detection is important in industries for a number of reasons. First and foremost, detecting defects early on can help prevent accidents and other safety hazards, as well as avoid costly repairs and downtime. Additionally, defect detection can help improve product quality and reliability, which can lead to increased customer satisfaction and loyalty. In some industries, such as aerospace and automotive, defect detection is also critical for compliance with safety regulations. Overall, defect detection is a crucial part of ensuring the safety, reliability, and quality of products and materials in various industries. One of the main advantages of using deep learning for defect detection is its ability to quickly and accurately analyze large amounts of data. Unlike humans, deep learning algorithms can process vast amounts of data and make predictions with a high degree of accuracy. This can be particularly useful in industries where there is a large amount of data to be analyzed, such as in the manufacturing of complex products. Another advantage of using deep learning for defect detection is its ability to identify subtle patterns and variations in data that may be difficult for humans to detect. This can help identify defects that are not immediately obvious, such as those that are hidden or occur infrequently. Additionally, deep learning algorithms can be trained to improve over time, which means that they can become more accurate and efficient at detecting defects as they are exposed to more data. This can help to improve the overall effectiveness of the defect detection process. Thermal imaging is a non-destructive testing method that uses specialized cameras to detect temperature differences on the surface of an object. This technology is often used for defect detection, as it can help identify areas of a material or structure that are not performing as expected due to the presence of defects or abnormalities. In defect detection using thermal images, the camera captures a thermal image of the object, which is then analyzed to identify areas of potential concern. This technique is particularly useful for detecting defects that are not visible to the naked eye, as it can help identify variations in temperature that may indicate the presence of a defect. Overall, the use of deep learning for defect detection has the potential to greatly improve the efficiency, accuracy, and reliability of the process, making it an attractive option for many industries.

1.2 Problem statement

The challenge is to develop a deep learning algorithm to inspect the presence of defects on given metal surface images and classify to the appropriate category the image falls into.

1.3 Objectives

- To create a deep learning model that is able to accurately identify and classify defects in images.
- To use the model to reduce the number of defective products, or improve the quality of the products being produced.
- To increase the efficiency and speed of the defect detection process by using the model to automate the inspection process.
- To reduce the need for manual inspection by using the model to identify defects that would be difficult or time-consuming for human inspectors to detect.
- To improve the overall accuracy and reliability of the defect detection process by using the model to identify defects that may be missed by human inspectors.

1.4 Literature survey

1.4.1 Segmentation-Based Deep-Learning Approach for Surface-Defect Detection [10]

Domen Tabernik et al. have proposed a segmentation-based deep-learning architecture that is designed for detection and segmentation of surface anomalies. They have performed experiments on a newly created dataset based on a real-world quality control case. The dataset contains 25-30 defective training samples. As shown in Fig. 1.1Novel segmentation and decision network was proposed in order to learn from a small number of defective training samples.

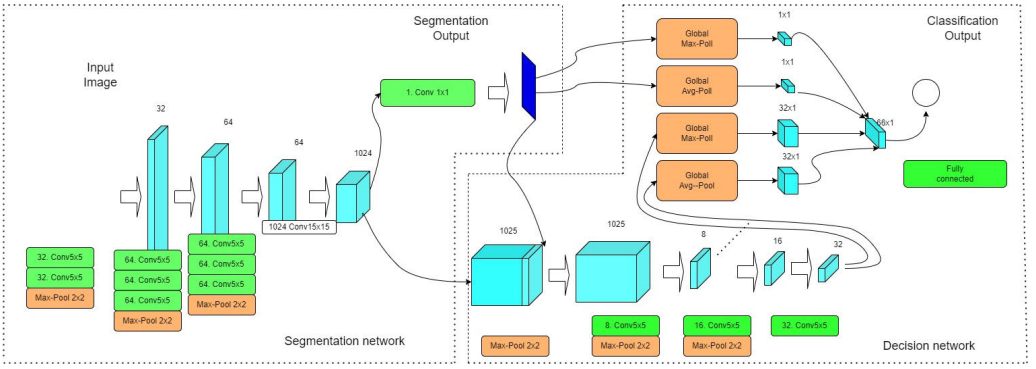

Figure 1.1: Architecture used in [10]

1.4.2 Infrared Thermal Imaging-Based Crack Detection Using Deep Learning [12]

In this work, Yang et al. have conducted an experiment wherein a horizontal heat conduction method is researched to thermally excite the surface of the steel using a thermal excitation source as illustrated in Fig. 1.2. The temperature difference between normal area and different cracked depths is analysed to study the influence of temperature change. 3000 infrared thermograms labelled for penetrating cracks, non-penetrating cracks and surface cracks are fabricated into a data bank. The authors have used Faster CNN for feature extraction and have validated the model on the data bank.

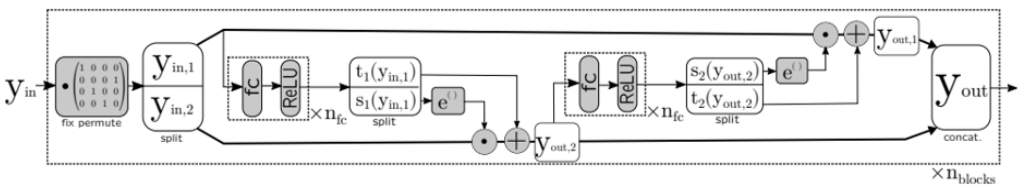

Figure 1.2: Faster RCNN Architecture used in [12]

1.4.3 Mixed Supervision for Surface-Defect Detection: from weakly to fully supervised learning [1]

The authors have proposed a deep-learning architecture, as shown in Fig. **??**that is composed of two sub-networks yielding defect segmentation and classification results. The model was evaluated on several datasets for industrial quality inspection. The authors have also presented a new dataset termed KolektorSDD2. In particular, they have proposed a deep-learning model for anomaly detection trained with weak supervision at the image-level labels, while at the same time utilising full supervision at the pixel-level labels when available. They have used the convolutional part of AlexNet as the feature extractor and applied global average pooling on each feature map at each scale. The normalising flow consists of 8 coupling blocks with fully connected networks. The model was trained for 192 epochs with a batch size of 96 and learning rate of 0.0002.

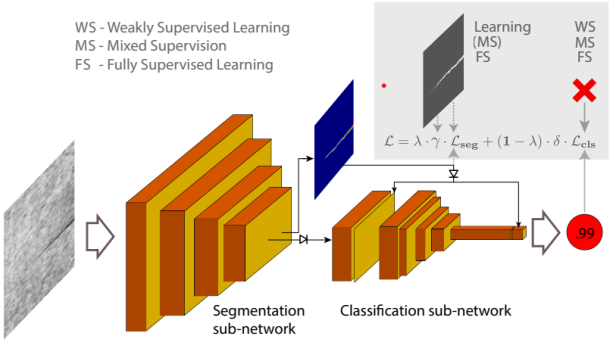

Figure 1.3: Segmentation and Classification used in [1]

1.4.4 Semi-supervised anomaly detection using Auto-Encoders [5]

In this paper, Minhas et al. have proposed an architecture similar to UNet architecture. The Encoder-Decoder architecture is as shown in Fig. 2.5. The encoder uses progressively decreasing filter sizes from 11x11 to 3x3. This decreasing filter size is chosen to allow for a larger field of view for the network without having to use large number of smaller size filters. The decoder structure has kernel sizes that are in the reverse of the encoder order and uses Transposed Convolution Layers. Adam optimizer was used and the training was done for 50 epochs, which led to an average F1 score of 0.885 on two datasets.

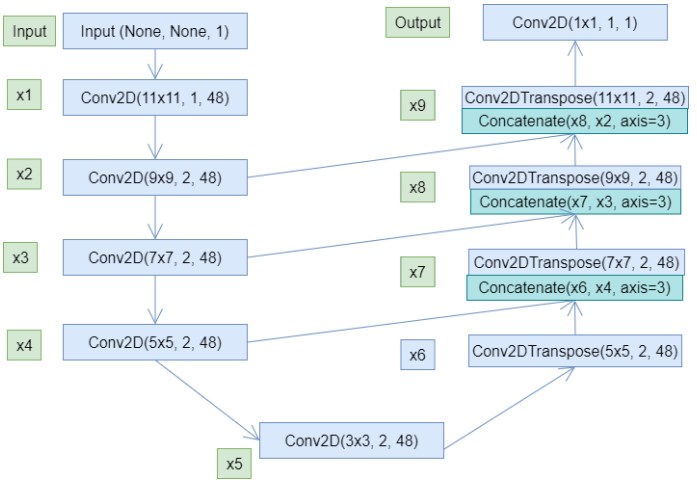

Figure 1.4: Architecture of AutoEncoder used in [5]

1.5 Application in Societal Context

- Quality control in manufacturing processes to ensure that products meet specific standards and requirements.
- Inspection of materials, components, and structures to identify and address potential issues before they cause problems.
- Monitoring of industrial processes and equipment to detect and diagnose problems before they cause disruptions or failures.
- Non-destructive testing (NDT) of structures, materials, and components to identify internal defects without causing damage.
- By detecting and addressing defects early on, renewable energy operators can ensure that their equipment is operating at peak performance, which can help to maximize the benefits of renewable energy and support the transition to a more sustainable energy system.
- Thermal imaging can help identify defects in industrial facilities, such as leaks in pipes or faulty equipment, to prevent operational disruptions and potential safety hazards.

1.6 Organization of the report

Chapter 1 discusses the need of Defect Detection, along with the motivation to come up with a solution that can solve the problem of industrial manufacturing defects. It also discusses several available approaches towards solving the problem and gives an insight about our approach towards the inspection of defect detection. In Chapter 2, the flowchart of the proposed approach has been illustrated where we came up with several design methods and have chosen the most suitable one. Chapter 3 discusses the implementation details along with the complete system architecture. It also discusses few important algorithms used in our work. Chapter 4 tells about the need of optimization in deep learning and the the choice of optimizer opted for our work. Chapter 5 is related to the results and discussions, where we analyse the performance of the proposed approach with the help of several visualisation techniques. Chapter 6 concludes the proposed work and talks about the future scope pertaining to defect detection.

Chapter 2

System design

In this Chapter, we list out the components required to meet the objectives.

2.1 Flowchart

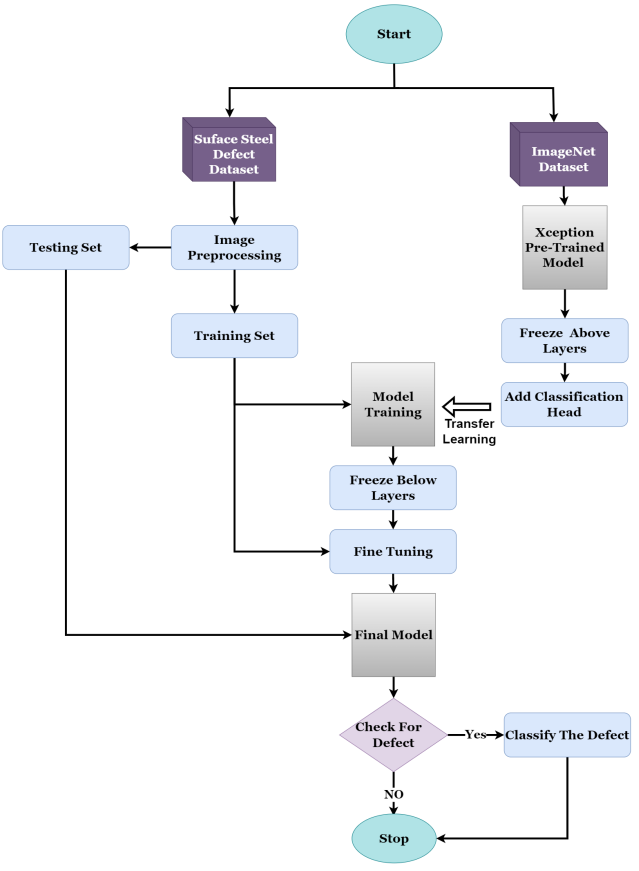

Figure 2.1: Flowchart of the system

2.2 Design alternatives

Approach 1:

As shown in Fig. 2.2, the convolutional neural network is composed of multiple convolutional layers followed by pooling layer which are responsible for learning a different set of features from the input data. The fully connected layers are followed by a final output layer (softmax layer), which produces the final classification for the input data. This approach is simple and easy to train, however it is prone to over-fitting and takes a lot of time to train.

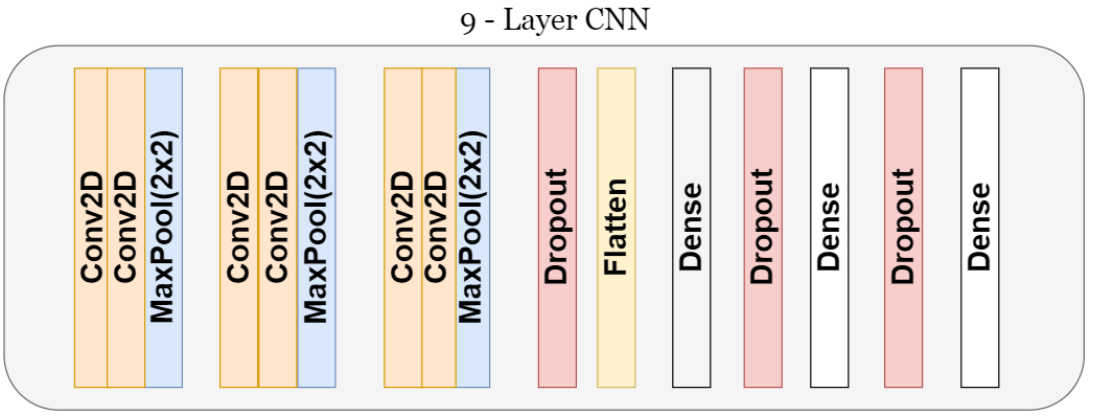

Figure 2.2: Convolutional Neural network with 9 weight layers

Approach 2:

Fig. 2.3 describes a 5-layer convolutional neural network. This is a simple network which is easy to understand. However, it is prone to the problem of overfitting and consumes lots of time in training the model.

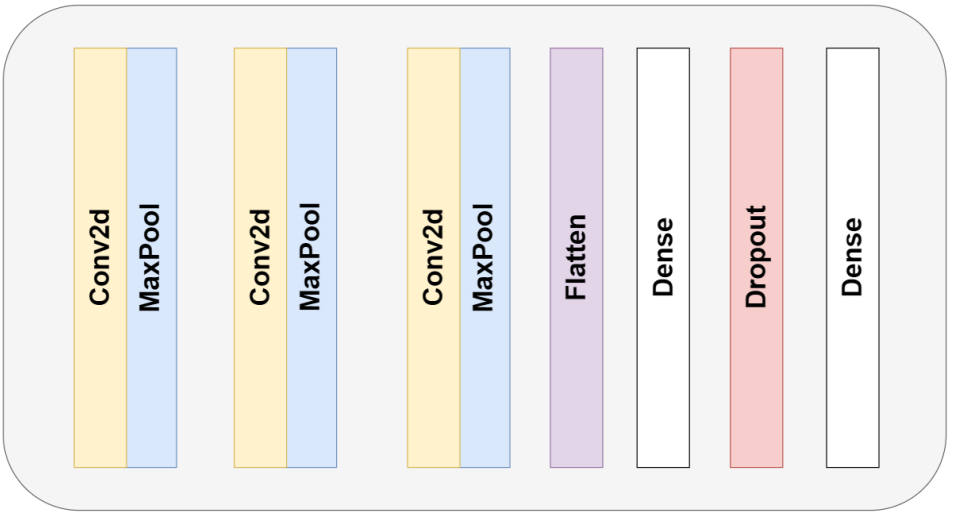

Figure 2.3: Convolutional Neural network with 5 weight layers

Approach 3:

This approach describes the use of transfer learning and fine-tuning. Transfer learning is a technique where a model trained on one task is reused for another related task. As shown in Fig. 2.4, a base model is used that is pre-trained on benchmark dataset. The working of this model is transferred so as to meet the requirements of the new task. The model is reconstructed and the weights are adjusted accordingly to meet the requirements. The adjustments is weights is done by fine-tuning. Once this is done, the model is made to predict the class for the given input image.

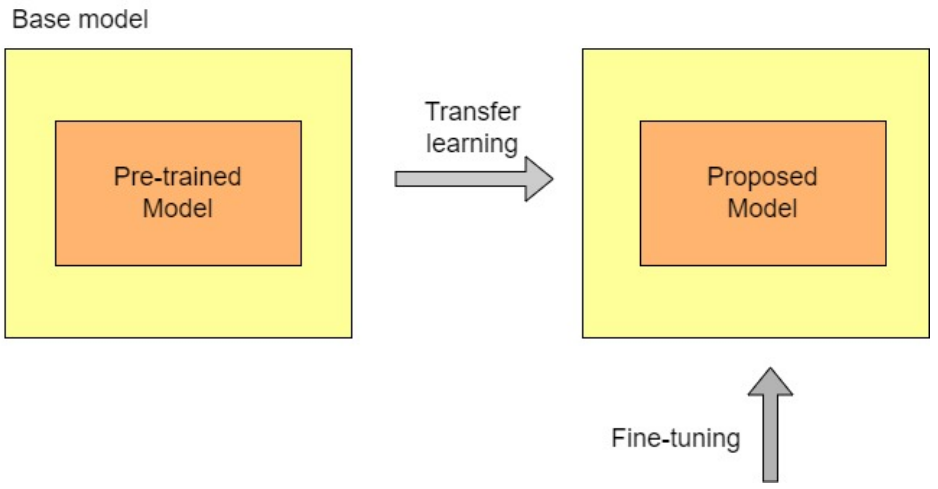

Figure 2.4: Transfer learning and Fine-tuning

2.3 Design approach followed in this work

2.3.1 Design pipeline

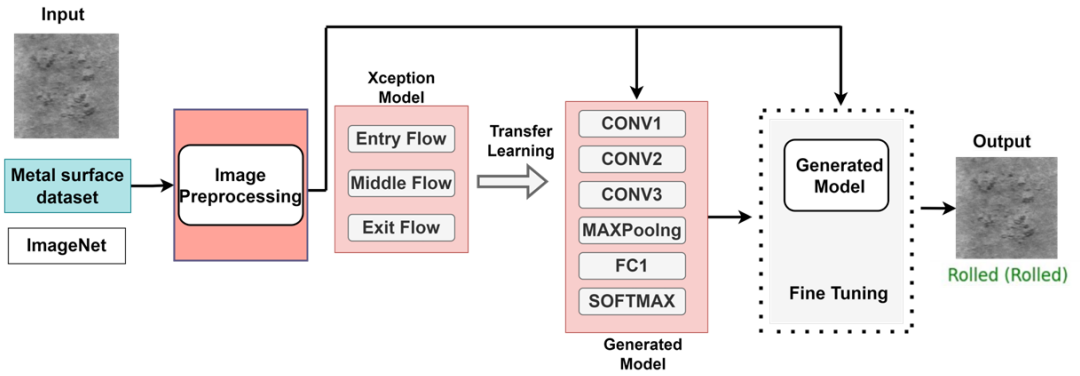

Figure 2.5: Final design pipeline of the proposed approach.

2.3.2 Why Xception Base Model?

Table 2.1: Comparison of various CNN models with Xception model

Model	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	**0.790**	**0.945**

As shown in Table 2.1, Xception has shown to achieve similar or better performance compared to other popular CNN architectures, such as VGG, ResNet and Inception, while requiring fewer parameters and less computation. This makes it an attractive choice for many image classification tasks.

Transfer learning is a machine learning technique where a model trained on one task is repurposed on a second related task. In the context of defect detection, this involves using a pre-trained Xception model on a large dataset, such as ImageNet and then fine-tuning it to learn to identify defects provided in the metal surface defect dataset. This can save a lot of time and resources compared to training a model from scratch, and can also improve the performance of the model.

2.3.3 Description of the design

- The dataset used in this work is Metal Surface Defect Dataset that contains six kinds of typical surface defects namely rolled-in scale, patches, crazing, pitted surface, inclusion and scratches. The dataset includes 1800 grayscale images, 300 samples of each of six different kinds of surface defects.
- The dataset is split into train set, test set and validation set.
- Xception model pre-trained on ImageNet dataset is used for transfer learning.
- The above layers of the model are freezed, so that they're non-trainable. Then, we add a classification head of our own desire and pass this combined model to the training on our dataset.
- The training will stop if the training accuracy reaches 98%. This is done to avoid over-fitting of the model since our dataset is small.
- However, the performance of the model can be increased by fine tuning the model. This is done by freezing the below layers and training the above layers so that the parameters are adjusted to the new dataset. The learning rate is decreased and the epochs are increased so that the accuracy increases whilst not making the model to overfit.
- The final model is evaluated against the test set.

Chapter 3

Implementation details

3.1 Specifications and system architecture

3.1.1 Xception Base Model

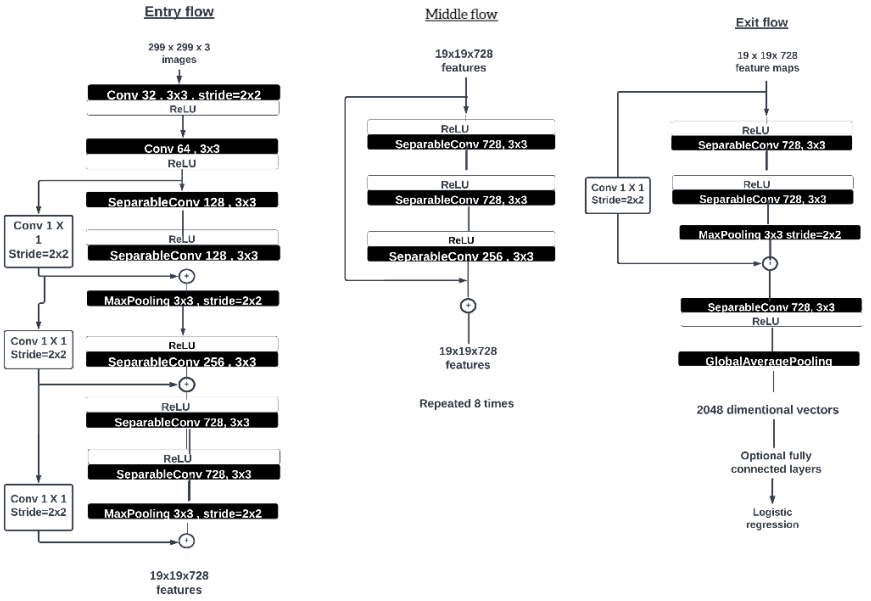

Figure 3.1: Xception Base Model

Entry Flow of Xception: The entry flow of the Xception model begins with a standard convolutional layer that processes the input image as shown in Fig. 3.1. This is followed by a series of depthwise separable convolutional layers, which apply a single filter to each input channel, followed by a pointwise convolutional layer that combines the output of the depthwise convolutional layers. This process is repeated multiple times, with each subsequent block of depthwise separable convolutions increasing the number of filters applied to the input. The output of the entry flow is then passed through a series of additional convolutional layers, pooling layers, and fully connected layers before being fed into the final output layer of the model. This output layer uses a softmax activation function to produce a probability distribution over the possible classes for the

input image, allowing the model to make a prediction about the class of the input image. Overall, the entry flow of the Xception model is designed to efficiently extract high-level features from the input image and pass them on to the later layers of the model for further processing and classification.

Middle Flow of Xception: Fig. 3.1 depicts the middle flow of the Xception model that is made up of a series of blocks, each of which consists of a set of depthwise separable convolutional layers followed by a pointwise convolutional layer. As in the entry flow, the depthwise convolutional layers apply a single filter to each input channel, followed by the pointwise convolutional layer which combines the outputs of the depthwise convolutional layers. The number of filters used in each block increases as the input progresses through the middle flow, allowing the model to learn increasingly complex and abstract features from the input image. Additionally, the middle flow includes skip connections, which allow the output of earlier layers in the network to be directly added to the output of later layers, allowing the model to better preserve spatial information and make more accurate predictions. The output of the middle flow is then passed through additional convolutional and pooling layers before being fed into the exit flow of the model. This helps to further refine and abstract the features extracted by the middle flow, preparing them for the final classification step in the model. Overall, the middle flow of the Xception model is designed to efficiently learn increasingly complex and abstract features from the input image, using a combination of depthwise separable convolutions and skip connections to improve performance and accuracy.

Exit Flow of Xception: As illustrated in Fig. 3.1, the exit flow of the Xception model begins with a series of convolutional layers, pooling layers, and fully connected layers, which process and abstract the features extracted by the middle flow of the model. This is followed by a final global average pooling layer, which combines the output of the previous layers by taking the average of each feature map. The output of the global average pooling layer is then fed into the final output layer of the model, which uses a softmax activation function to produce a probability distribution over the possible classes for the input image. This allows the model to make a final prediction about the class of the input image, based on the features extracted and abstracted by the earlier layers of the network. Overall, the exit flow of the Xception model is designed to process and abstract the features extracted by the middle flow of the model, and use them to make a final prediction about the class of the input image. This is done using a combination of convolutional, pooling, and fully connected layers, along with a global average pooling layer and a softmax output layer.

3.1.2 Transfer learning

To perform classification using transfer learning with the Xception model, we would first need to obtain a pre-trained Xception model on a large dataset. Hence, the model was pre-trained on ImageNet dataset. Once the pre-trained model is obtained, we have used it as the starting point for our own classification task by fine-tuning the model on Metal Surface Defect Dataset. In this, we have freezed the convolutional base from the pre-trained model to use as a feature extractor. Additionally, we have added a classifier on top of it to train the top-level classifier.

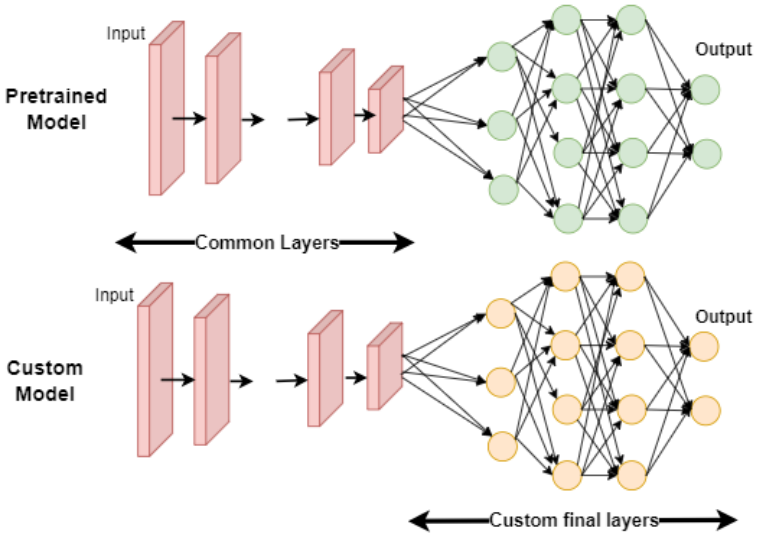

Figure 3.2: Process of transfer learning

3.1.3 Fine Tuning

Fine-tuning is the process of adapting a pre-trained model to a new task by adjusting its parameters to improve its performance on that specific task. This is often done in transfer learning, where a model trained on one task is re-purposed on a second related task. Fine-tuning a pre-trained model typically involves adjusting the model's parameters by training it on a new dataset that is specific to the new task. This allows the model to learn the characteristics of the new dataset and improve its performance on the new task. Fine-tuning can be an effective way to improve the performance of a pre-trained model on a new task, and can save a lot of time and resources compared to training a model from scratch. In the feature extraction step, we have trained a few layers on top of an Xception base model. The weights of the pre-trained network were not updated during training. One way to increase performance even further is to fine-tune the weights of the top layers of the pre-trained model alongside the training of the classifier we added. The training process will force the weights to be tuned from generic feature maps to features associated specifically with the given surface defect dataset. The first few layers learn very simple and generic features that generalize to almost all types of images. As you go higher up, the features are increasingly more specific to the dataset on which the model was trained. The goal of fine-tuning is to adapt these specialized features to work with the new dataset, rather than overwrite the generic learning.

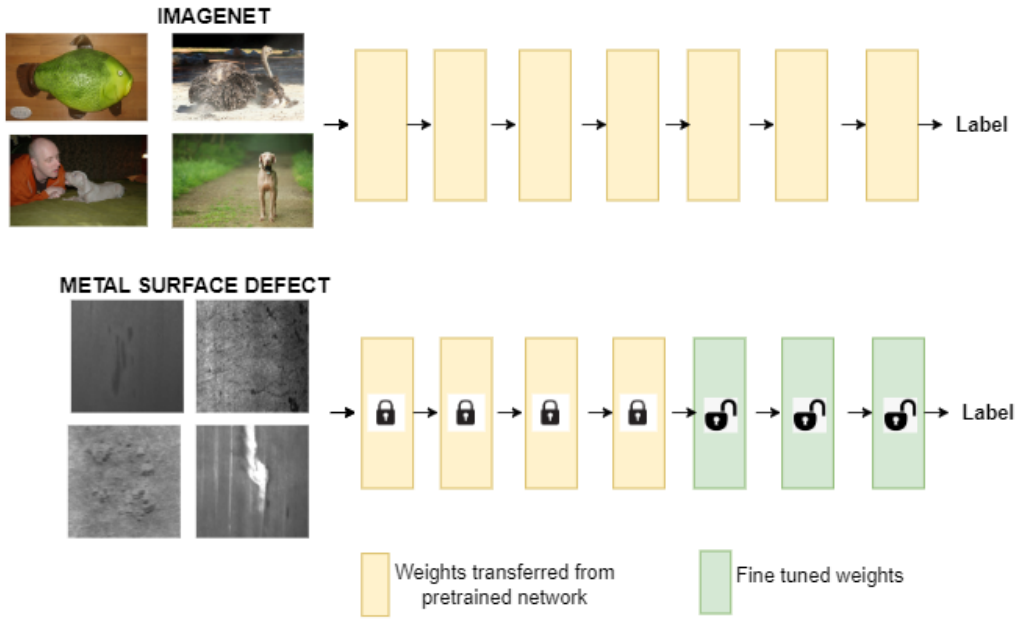

Figure 3.3: Process of fine-tuning

3.2 Algorithm

3.2.1 Pseudo code for Transfer learning

Load pre-trained model
model = load_pretrained_model(model_path)

Freeze layers of pre-trained model
for layer in model.layers:
layer.trainable = False

Add new layers for new task
x = model.output
x = Dense(1024, activation="relu")(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation="softmax")(x)

Create new model for new task
model_new = Model(inputs=model.input, outputs=predictions)

Compile new model
model_new.compile(optimizer=Adam(lr=0.0001),
loss="categorical_crossentropy")

Train new model on new task-specific dataset
model_new.fit(X_train, y_train, epochs=10, batch_size=32)

3.2.2 Pseudo code for Fine-tuning

Load the pre-trained model and the new data for the task
model = load_pretrained_model()
X_train, y_train = load_new_data()

Freeze the layers of the pre-trained model to prevent the weights from being updated during training
for layer in model.layers:
layer.trainable = False

Add additional layers to the pre-trained model to adapt it to the new task
output = model.layers[-1].output
output = Dense(128, activation="relu")(output)
output = Dense(64, activation="relu")(output)
output = Dense(32, activation="relu")(output)
predictions = Dense(1, activation="sigmoid")(output)

Create a new model with the pre-trained model as the base and the new layers on top
model = Model(inputs=model.input, outputs=predictions)

Compile the model with a loss function and an optimizer
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

Train the model on the new data
model.fit(X_train, y_train, epochs=10, batch_size=32)

Unfreeze some of the layers of the pre-trained model and fine-tune the weights of these layers
for layer in model.layers[:3]:
layer.trainable = True

Compile the model again with a different learning rate to fine-tune the weights of the unfrozen layers
model.compile(loss="binary_crossentropy", optimizer=Adam(lr=0.0001),
metrics=["accuracy"])

model.fit(X_train, y_train, epochs=10, batch_size=32)

Chapter 4

Optimization

In this chapter, we discuss about the optimization technique utilized in this work. Also, we discuss the need of optimization and provide justification for the optimization technique used.

4.1 Introduction to optimization

Optimization in deep learning refers to the process of finding the values for the weights of a deep learning model that minimize the loss function of the model. This is an important step in training a deep learning model, as the values of the weights determine the performance of the model on a given task. In general, optimization in deep learning involves iteratively updating the weights of the model in a direction that reduces the loss on the training data. This process is repeated until the model reaches a satisfactory level of performance on the task.

4.2 Selection and justification of Adam Optimization

Adam (Adaptive Moment Estimation) is an optimization algorithm commonly used in deep learning. It is a variant of stochastic gradient descent that uses an adaptive learning rate and momentum to improve the convergence of the model during training. It is an efficient algorithm that can be used to train deep learning models with large amounts of data. It has been shown to outperform other optimization algorithms in a variety of applications, including natural language processing and computer vision.
The Adam algorithm is an iterative method that updates the weights of a deep learning model based on the gradient of the loss function with respect to the weights. It calculates an exponentially weighted average of the past gradients, which is used to determine the direction in which the weights should be updated. It also calculates an exponentially weighted average of the squared past gradients, which is used to adjust the learning rate of the model. One of the key advantages of Adam is that it uses a dynamic learning rate, which means that it can automatically adjust the learning rate of the model during training. This can help the model converge faster and avoid getting stuck in local minima. Adam also uses momentum, which can help the model avoid oscillating around the minimum of the loss function and converge more smoothly.
Adam is considered to be a better optimization algorithm than other algorithms, such as SGD and RMSprop, in many cases because of its efficient convergence, dynamic learning rate, ease of use, and robust performance.

Chapter 5

Results and discussions

In this chapter, we will discuss the dataset used in this work along with the performance analysis of the proposed model.

5.1 Dataset Description

Table 5.1: Metal Surface Defect Dataset

Class Labels	**No. of images**		
	Train set	***Test set***	***Valid set***
Rolled-in Scale	276	12	12
Patches	276	12	12
Crazing	276	12	12
Pitted surface	276	12	12
Inclusion	276	12	12
Scratches	276	12	12

The dataset used in this work is Metal Surface Defect Dataset that contains six kinds of typical surface defects namely rolled-in scale, patches, crazing, pitted surface, inclusion and scratches as shown in Table 5.1. The dataset includes 1800 grayscale images, 300 samples of each of six different kinds of surface defects. The images are thermal images that are mapped to gray scale. The dataset is split into train set, test set and valid set.

5.2 Performance Analysis

As shown in Fig. 5.1, the accuracy of the model after training it was found out to be 98.25%. This training accuracy was achieved before fine-tuning the model. The model would overfit, had we let it train for more epochs. The blue line indicates the train accuracy and the orange line indicates the test accuracy.

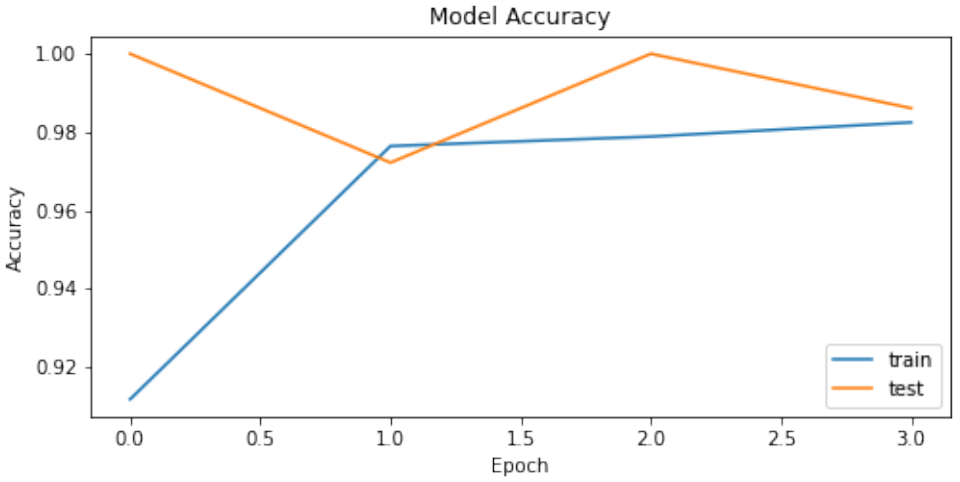

Figure 5.1: Accuracy of the model before fine-tuning

As shown in Fig. 5.2, the loss value of the model after training it was found out to be 0.0434. This loss value was achieved before fine-tuning the model. The blue line indicates the training loss and the orange line indicates the testing loss.

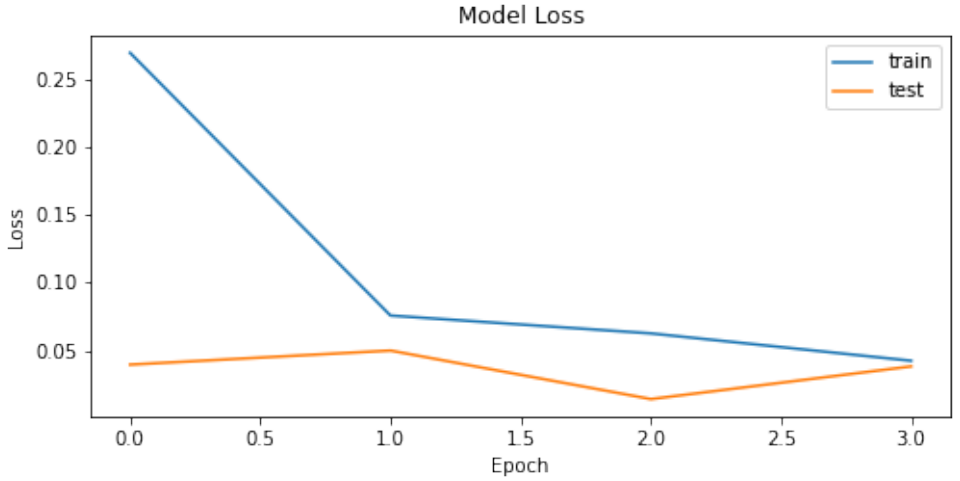

Figure 5.2: Loss value of the model before fine-tuning

As shown in Fig. 5.1, the accuracy of the model after training it was found out to be 98.25%. This training accuracy was achieved before fine-tuning the model. However, after the model underwent the process of fine-tuning, it achieved a training accuracy of 99.58%. This is depicted in Fig. 5.3. The blue line indicates the train accuracy and the orange line indicates the test accuracy.

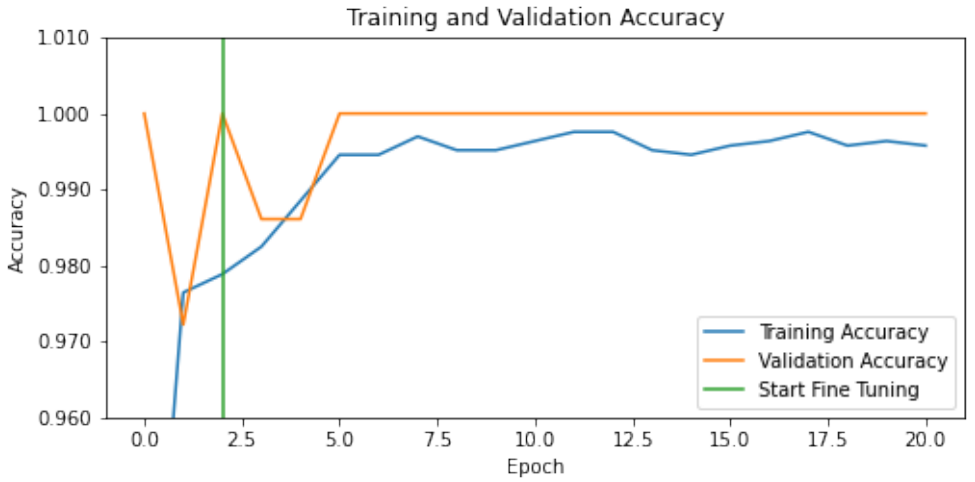

Figure 5.3: Accuracy of the model after fine-tuning

As shown in Fig. 5.4, the loss value was decreased after the model underwent the process of fine-tuning. The loss value was found out be 0.0109.

Figure 5.4: Loss value of the model after fine-tuning

Fig. 5.5 provides the output of the defect detection system. Class labels are assigned to each of the input image to predict the type of defect the input image contains.

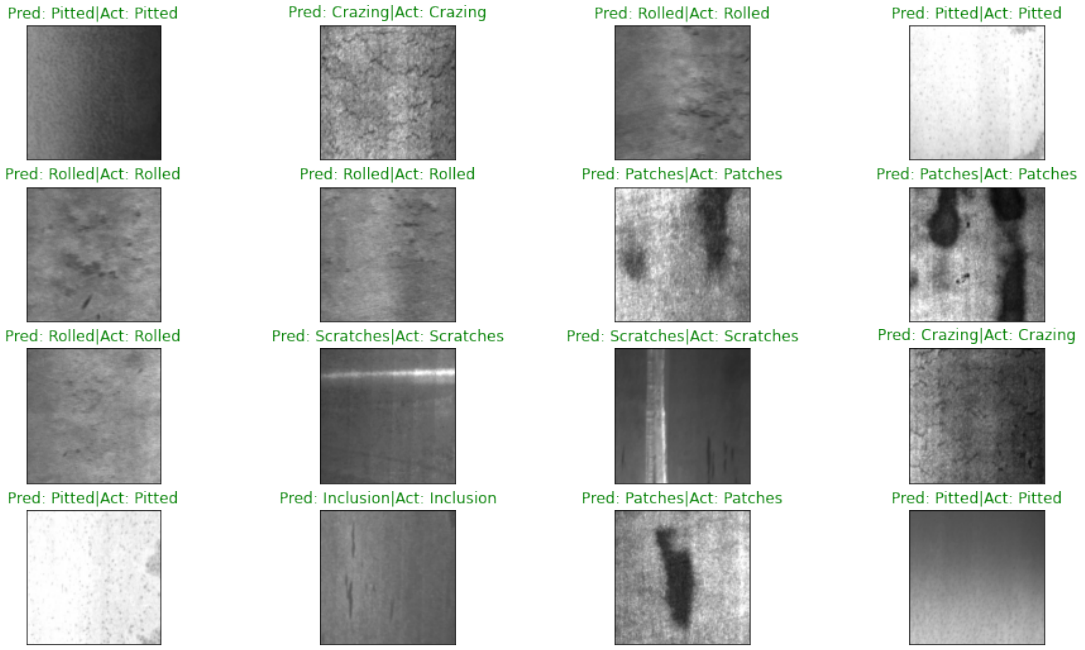

Figure 5.5: Predicting classes for given input images

Chapter 6

Conclusion and future scope

In this chapter, we summarize what we have done to achieve the objectives and give a brief gist of how our proposed model was implemented and evaluated. In addition to this, we also state the future scope of the proposed work and how it can be scaled to meet several other requirements.

6.1 Conclusion

In this work, we have presented a deep-learning approach to metal surface defect detection. We presented several approaches for detection of defects and chose the optimal one. The final design architecture has two main components: transfer learning using Xception baseline model pre-trained on ImageNet dataset and Fine-tuning to improve the performance of the model without it becoming prone to overfitting. We have provided reasoning for choosing Xception model while comparing it with several state-of-the-art classification models. Comparisons were made before and after the usage of fine-tuning. Fine-tuning resulted in an increase in accuracy from 98.25% to 99.58%. The increase in accuracy also took care of the fact that the model should not overfit. This work is able to reduce the computation speed and enhance the performance of the system.

6.2 Future scope

The use of deep learning models for defect detection is likely to become more common, as these models are able to learn complex patterns in the data and make more accurate predictions than traditional machine learning methods. This will enable companies to more effectively detect defects in their products and processes, and to take corrective action to prevent these defects from occurring in the future. Another area of growth for defect detection is the use of sensor data, such as images, videos, and audio recordings, to detect defects. This will enable companies to monitor their products and processes in real time and to identify defects as they occur, rather than relying on manual inspection or testing. This will improve the efficiency and effectiveness of defect detection, and will enable companies to more quickly identify and address problems in their products and processes.

Bibliography

[1] Jakob Božič, Domen Tabernik, and Danijel Skočaj. Mixed supervision for surface-defect detection: From weakly to fully supervised learning. *Computers in Industry*, 129:103459, 2021.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Qifan Jin and Li Chen. A survey of surface defect detection of industrial products based on a small number of labeled data. *arXiv preprint arXiv:2203.05733*, 2022.

[5] Manpreet Singh Minhas and John Zelek. Semi-supervised anomaly detection using autoencoders. *arXiv preprint arXiv:2001.03674*, 2020.

[6] Mykola Robotyshyn, Marianna Sharkadi, and Mykola Malyar. Surface defect detection based on deep learning approach. In *IntSol Workshops*, pages 32–44, 2021.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[8] Kathiravan Srinivasan, Lalit Garg, Debajit Datta, Abdulellah A Alaboudi, NZ Jhanjhi, Rishav Agarwal, and Anmol George Thomas. Performance comparison of deep cnn models for detecting driver's distraction. *CMC-Computers, Materials & Continua*, 68(3):4109–4124, 2021.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[10] Domen Tabernik, Samo Šela, Jure Skvarč, and Danijel Skočaj. Segmentation-based deep-learning approach for surface-defect detection. *Journal of Intelligent Manufacturing*, 31(3):759–776, 2020.

[11] Grega Vrbančič and Vili Podgorelec. Transfer learning with adaptive fine-tuning. *IEEE Access*, 8:196197–196211, 2020.

[12] Jun Yang, Wei Wang, Guang Lin, Qing Li, Yeqing Sun, and Yixuan Sun. Infrared thermal imaging-based crack detection using deep learning. *Ieee Access*, 7:182060–182077, 2019.