Icon

Description automatically generated

Defect Inspection

Table of Contents:

[**I. INTRODUCTION** 2](#_Toc111132160)

[**II. Problem statement** 3](#_Toc111132161)

[**III. Dataset** 4](#_Toc111132162)

[**IV. Experiments** 4](#_Toc111132163)

[**V. Model** 4](#_Toc111132164)

[**VI. Evaluation and Results** 4](#_Toc111132165)

[**VII. Conclusion** 4](#_Toc111132166)

[**VIII. REFERENCES** 5](#_Toc111132167)

Defect Inspection: Deep Learning Approach on TIG welding Aluminum

Abdullah Shehata, Aya Adel, Maria George, Azza Ali

***Abstract*—** Tungsten Inert Gas (TIG) welding, also known as gas tungsten arc welding, is an arc welding process that uses a non-consumable tungsten electrode to produce the weld. In TIG welding, there generally occurs some problems like burn through, contamination problems, etc. Deep Learning era has showed a great success in computer vision problems in general and Defect detection in specific, so a huge number of experiments using deep Learning Techniques has been implemented to reach the highest accuracy for this problem and the one which has the best accuracy was the pre-trained model: squeeze net with 99.2% test accuracy on TIG Aluminum 5083 dataset.

# **I. INTRODUCTION**

Deep Learning is evolving as one of the crucial practices in industries like manufacturing, hospitality, digital assistants (IoT), automotive, etc. With the increased use of deep learning, the industries are leveraging the use of Deep Learning techniques to be used as a part of the industry. TIG welding Aluminum is a very vital industry nowadays. It is obvious that the TIG welding process demands a great deal of skill if performed manually, which makes it challenging to repeat welding of complicated geometries and results in significant scrappage rates. As a result, the welding process must be automated. The automation of the process depends on the monitoring and control of the weld pool to guarantee quality, uniformity, and repeatability. So, this paper proposes an automated solution to detect the defection in the process of welding the aluminum by implementing several experiments using deep learning techniques to reach a satisfying model that can detect the defection correctly. So, the results of the experiments showed that using the pretrained model: squeeze net, which is a convolutional neural network that is 18 layers deep that is pretrained on more than a million images from the ImageNet database [1], has showed a magnificent result on TIG Aluminum 5083 dataset.

# **II. PROBLEM STATMENT**

TIG is an arc welding process that uses a non-consumable tungsten electrode to produce the weld. The weld region and electrode are shielded from oxidation or other air contamination by an inert shielding gas (argon or helium), and a filler metal is typically used. TIG is used to assemble expensive, precise components. It is most frequently used to weld copper alloys, stainless steel, aluminum, and magnesium alloys [2]. While giving the welder more control over the welding process, but it is challenging to master. Therefore, there are some opportunities for welding mistakes when it comes to TIG welding. Some of them are poor gas coverage, dirty base/filler metal and improper arc length control and this results in making 5 different types of defects like:

1. Burn Through: when the weld metal completely penetrates the base metal.
2. Contamination: Contamination causes the welding to become porous, which reduces the strength of the welding. Contamination may occur from polluted electrodes, metal, or shield gas flow.
3. Lack of fusion: It may be caused by ineffective welding methods. if travel speed is too slow, the leading-edge of the arc, which should be ahead of the puddle, will be in the puddle so it will result in a lack of fusion.
4. Misalignment: It can be due to poor component fit-up or relative motion between the components during the welding process.
5. Lack of penetration: incomplete penetration happens when a weld bead does not begin at the weld groove's base which is typically produced from low current level.

Graphical user interface, application

Description automatically generated

Fig.1. TIG welding process layout [2].

# **III. DATASET**

The dataset used in this paper is Tig Aluminum 5083 From Kaggle which contains TIG welding footages recorded with HDR camera. The dataset contains 33.3k image split into two folder train and test.

Graphical user interface, text, application, email

Description automatically generated

Fig.2. Dataset Main Folders.

Each folder of them contains some subfolders that are unlabeled. However, there are 2 JSON files that contains subfolder names and their labels. There are 46 subfolders in the train folder and 14 subfolders in the test. each subfolder contains some images of a certain class.

Graphical user interface, text, application

Description automatically generated

Fig.3. Sample from train subfolders.

There are 6 classes: good weld, burn through, contamination, lack of fusion, misalignment, and lack of penetration represented as 0,1,2,3,4,5 respectively in the JSON file.

Graphical user interface

Description automatically generated

Fig.4. Dataset samples of aluminum TIG welding. (a) good weld; (b) burn through; (c) contamination; (d) lack of fusion; (e) misalignment; (f) lack of penetration.

As a first step, the dataset should be labeled so we made a dataframe that contains each subfolder name, number of images in each folder and the label. Then we created 6 folders with class names and move the images to these folders based on their label. After finishing experiments that will be mentioned in the next section, we decide to divide the data to only 2 classes the first class is good weld and gathered all the other classes as defect weld as this give us the highest test accuracy.

Table

Description automatically generated

Fig.5. Test Dataframe.

# **IV. EXPERIMENTS**

After preparing the data, there are some experiments done to choose the best model for defect detection. We used python-based Notebook to design our experiments. There are many cloud computing platforms, but we used Google Colab that give us K80 GPU and 12GB RAM. A fair test relies just on one distinction or modification. Most experimental investigations aim to determine if the change we make is indeed responsible for the impact we observe. It's crucial to only alter one variable at a time in a fair test. So, we maintain the same environments for all of the experiments and also the pre-processing steps which are resizing the image, convert them into tensors and normalize them. We used pre-trained models from PyTorch library. These experiments can be separated into three categories: the first category uses data that is divided into six classes, the second category uses data that is divided into four classes, and the third category uses data that is divided into two classes.

**First category:**

Here the images are labeled into 6 classes: good weld, burn through, contamination, lack of fusion, misalignment, and lack of penetration, preprocessed with the aforementioned steps and also trained with the same number of epochs (10). The only difference here is the model as we used 10 different pre-trained model from PyTorch library [4] and get their accuracies. The results shows that the best accuracy was obtained by SqueezeNet model [1] with training accuracy equals to 98.53% and validation accuracy equals to 0.9929 and the worst accuracy was obtained by ShuffleNetV2 with training accuracy equals to 0.6526 and validation accuracy equals to 67.57%.

Graphical user interface, application, table, Excel

Description automatically generated

Fig.6. accuracies of the first category.

**Second category:**

Here we maintained the model and the preprocessing steps the only variable here was to drop some classes and combine similar classes that the model did not get it right. The model used in this section is SqueezeNet with 5 epochs.

1. With dropping contamination class and combining Misalignment and Lack of penetration:

The training accuracy was 98%, validation accuracy equals to 99% and finally a test accuracy equal to 81%

Graphical user interface

Description automatically generated with low confidence

Fig.7. confusion matrix when dropping contamination class and adding Misalignment + Lack of penetration together.

1. With dropping contamination class only:

The training accuracy was 99%, validation accuracy equals to 99% and a test accuracy equal to 78%

Graphical user interface, application

Description automatically generated

Fig.8. confusion matrix when dropping contamination class.

1. With dropping contamination class and combining Misalignment and Lack of fusion:

The training accuracy was 99%, validation accuracy equals to 99% and a test accuracy equal to 83%.

A picture containing chart

Description automatically generated

Fig.9. confusion matrix when dropping contamination class and combining Misalignment + Lack of fusion.

**Third category:**

At this point the maximum test accuracy was 83% so we decide to divide the data to only 2 classes the first class is good weld and combined all the other classes as defect weld and this made the test accuracy reach 92%. From the previous experiments was found that the model may be overfitted to the data and to treat this overfitting is to split the classes in train and test data with the same ratio. All the data was added in one folder (combined test and train folders together) then we used split-folder library to split them to train and test folders with ratio 80:20.

After that the data was labelled and divided into 2 classes good and defect. The model was SqueezeNet, and the preprocessing steps was the same as before. After splitting the data to equal ratio, the model reached 99% test accuracy.

# **V. MODEL**

This paper proposes two models the first one is using the pretrained SqueezeNet version 1.1 from PyTorch library to detect if it is good or defect and the second model was also SqueezeNet version 1.1 to detect which type of defection from the 5 classes: burn through, contamination, lack of fusion, misalignment, and lack of penetration.

**SqueezeNet:**

SqueezeNet model maintain Alexnet accuracy level with 50x fewer parameters. To achieve competitive accuracy with fewer parameters there was 3 strategies:

**Strategy 1**. Replace 3x3 filters with 1x1 filter since a 1x1 filter has 9X fewer parameters than a 3x3 filter

**Strategy 2.** Decrease the number of input channels to 3x3 filters. Think of a convolution layer that just uses 3x3 filters. The total number of parameters in this layer is (3\*3) \* (number of filters) \* (number of input channels). Therefore, it is crucial to reduce both the number of 3x3 filters and the input channels to the 3x3 filters in order to keep a low overall number of parameters in a CNN. Using squeeze layers, that will be discussed later, the number of input channels has been reduced to 3x3 filters.

**Strategy 3.** Down sample late in the network so that convolution layers have large activation maps. In a convolutional network, each convolution layer produces an output activation map with a spatial resolution that is at least 1x1 and often much larger than 1x1. The height and width of these activation maps are controlled by: (1) the size of the input data and (2) the choice of layers in which to down sample in the CNN architecture.

The Fire module, which serves as the foundation for SqueezeNet CNN design and lead to effective implementation for the 3 strategies. The Fire module is defined as follow, a fire module consists of a squeeze convolution layer (which has only 1x1 filter), feeding into a expand layer that includes a mixture of 1x1 and 3x3 convolution filters. In a fire module, there was three adjustable dimensions (hyperparameters): s1, e1, and e3. In a fire module, s1x1 is the number of filters in the squeeze layer, e1x1 is the number of 1x1 filters in the expand layer, and e3x3 is the number of 3x3 filters in the expand layer. When the Fire modules is used the hyperparameter s1x1 is set to be less than (e1x1+e3x3), so the squeeze layer helps to limit the number of input channel s to the 3x3 filters.

The SqueezeNet CNN architecture can be described as follow. SqueezeNet begins with a standalone convolution layer(conv1), followed by 8 Fire modules (fire 2-9), ending with a final conv layer (conv10). The number of filters per fire module are gradually increased from the beginning to the end of the network. SqueezeNet performs max-pooling with a stride of 2 after layers conv1, fire3, fire5, and conv10.

We used squeezenet1\_1 by the [official squeezenet repo](https://github.com/DeepScale/SqueezeNet/tree/master/SqueezeNet_v1.1). It has 2.4x less computation and slightly fewer parameters than squeezenet1\_0, without sacrificing accuracy. The model was trained for 20 epochs. Moreover, Stochastic Gradient descent was used as an optimizer with learning rate equals to 0.001 and momentum equals to 0.9. Cross entropy was also used as a loss function.

Graphical user interface, text, email

Description automatically generated

Fig.10. SqueezeNet v1.0 vs SqueezeNet v1.

# **VI. EVALUATION AND RESULTS**

As a result, after all the experiments that has been done was found that the best experiment was when using only 2 classes good and defect after re-distributing the data of the train and test folders. The model was SqueezeNet version 1 trained with 20 epochs, optimized by stochastic gradient descent from PyTorch optimizers and cross entropy as a loss function. The preprocessing steps was as follow: resize, convert to tensor and normalization. The data was also shuffled. For evaluation we used accuracy metric and confusion matrix. The training accuracy equals to 98.7%, validation accuracy equals to 99.06% and a test accuracy equals to 99.5%.

Graphical user interface, application, Word

Description automatically generated

Fig.11. validation accuracy vs number of training epochs.

The model classified only 17 images as a defect, but it was good weld and 12 images as a good and it was a defect.

Chart, treemap chart

Description automatically generated

Fig.12.confusion matrix of 2 classes.

After knowing the image is good weld or defected. There is a second model was trained to detect which defect is in the data from the following classes, burn through, contamination, lack of fusion, misalignment, and lack of penetration. The second model used also SqueezeNet version 1. It got trained with 10 epochs, optimized by stochastic gradient descent from PyTorch optimizers and cross entropy as a loss function. The preprocessing steps was as follow: resize, convert to tensor and normalization. The data was also shuffled. For evaluation we used accuracy metric and confusion matrix. The training accuracy equals to 98.7%, validation accuracy equals to 99.66% and a test accuracy equals to 99.5%.

Graphical user interface, application, Word

Description automatically generated

Fig.13. validation accuracy vs number of training epochs for the second model.

Chart

Description automatically generated

Fig.14. confusion matrix for the second model.

# **VII. CONCLUSION**

In a nutshell, considering all the aforementioned information this paper designed a system for classifying good and defect weld using the pretrained model SqueezeNet on the TIG aluminum 5083 dataset and it reach a test accuracy equals to 99.5%. There were several experiments, and the models were trained using 6-classes, 4-classes, and 2-classes tests with highest test accuracies of 71%, 83% and 99.5%, respectively. Finally, the 2 models were deployed using Rest API with flask, the first one is to detect if the image is good weld or defect and the second one is to classify the defect in the image with test accuracy equals to 99%.

# **VIII. REFERENCES**

1. “SqueezeNet convolutional neural network - MATLAB squeezenet,” www.mathworks.com. https://www.mathworks.com/help/deeplearning/ref/squeezenet.html (accessed Aug. 11, 2022).
2. D. Bacioiu, G. Melton, M. Papaelias, and R. Shaw, “Automated defect classification of Aluminium 5083 TIG welding using HDR camera and neural networks,” Journal of Manufacturing Processes, vol. 45, pp. 603–613, Sep. 2019, doi: 10.1016/j.jmapro.2019.07.020.
3. “TIG Aluminium 5083,” www.kaggle.com. https://www.kaggle.com/datasets/danielbacioiu/tig-aluminium-5083 (accessed Aug. 11, 2022).
4. PyTorch, “PyTorch,” Pytorch.org, 2019. <https://pytorch.org/>
5. F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally, and K. Keutzer, “SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE.” [Online]. Available: <https://arxiv.org/pdf/1602.07360.pdf>
6. F. Iandola, “forresti/SqueezeNet,” GitHub, Aug. 07, 2022. https://github.com/forresti/SqueezeNet/tree/master (accessed Aug. 14, 2022).
7. “Welcome to Flask — Flask Documentation (2.2.x),” flask.palletsprojects.com. https://flask.palletsprojects.com/en/2.2.x/