

Project #2

Deadline: 23:59 November 5, 2025

1 Background & Introduction

Image recognition has become a critical advancement within Artificial Intelligence (AI), particularly for tasks requiring automated visual inspection in aerospace engineering. In aircraft maintenance, regular inspections are necessary to identify and address issues such as cracks, missing screws, and surface degradation, all of which are critical for the safety and longevity of the aircraft structure.

In this project, you will focus on a dataset containing images of aircraft skin defects, including cracks, missing screw heads, and paint degradation. These defects need to be identified early to prevent further damage and ensure the structural integrity of the aircraft. In real-world applications, automating the detection of such defects using Deep Convolutional Neural Networks (DCNNs) would enhance both efficiency and accuracy compared to traditional manual inspection methods.

You are tasked with building a DCNN model that can classify these defects into their respective categories. This project will involve data processing, designing neural network architectures, hyperparameter tuning, evaluating model performance, and testing the model on unseen images. An example of an image from the dataset can be seen in [1](#) below.

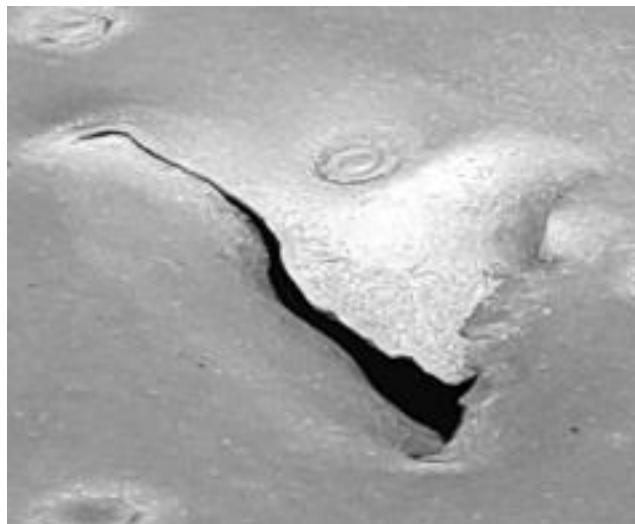


Figure 1: Crack Image Data Example

2 Project Outline

Based on the image dataset provided, you are required to build two variations of DCNN models which can classify the crack images into 3 classes: crack, missing-head and paint-off. This overall task has five major components, data processing, neural network architecture design, network hyperparameter selections, accuracy & loss evaluation of the model, and model testing on new images. Packages required for all steps below: **ImageDataGenerator from keras**, **models from keras**, **layers from keras**, **numpy**, and **pandas**.

2.1 Step 1: Data Processing - 20 Marks

Data processing is the first step to perform any type of image classification task. There are a few steps involved with this step.

- The first is to define the input image shape which is required to be (500,500, 3) which is the desired width, height and channel of the image for model training.
- Establish the train and validation data directory (use relative paths). The data is split into 3 folders - Train, Validation and Test which contain 1942, 431 and 539 images respectively.
- Perform data augmentation such as re-scaling, shear range and zoom range by using packages such as Keras' image preprocessing pipeline, or torchvision transforms for the train data and validation data (only apply re-scaling for validation).
- Create the train and validation generator using Keras's built-in *imagedatasetfromdirectory* function which takes in the data directory, image target size, batch size (32), and class mode (categorical) or by using PyTorch's *Dataloader* .

2.2 Step 2: Neural Network Architecture Design - 30 Marks

For this stage you are required to build your own custom neural network. You are allowed to use any machine learning platform. If you are using Tensorflow or PyTorch, you can start with the following tutorials: [Tensorflow CNN Tutorial](#) or [Pytorch CNN Tutorial](#). The neural network will have multiple layers. The layers to extract the features from the given images are outlined below.

- Convolutional layers which are accessed through the *Conv2D* package which filters the images and extract key features. The three main components to explore within this layer are: number of filters, kernel size, and stride. (The main focus can be on filters and kernel size).

- The next layer is the *MaxPooling2D* layer which follows the convolutional layers and pools the extracted features and takes the highest value.
- The *Flatten* layer is then utilized after all the convolutions and max pooling layers, which is applied before the dense layers.
- Lastly, the fully connected *Dense* and *Dropout* layers are utilized to perform the final predictions. (The final dense should only have 3 neurons, correlating to the label classes).
- **Note:** You can adjust the convolutional and fully connected layers to improve the performance of the network.

2.3 Step 3: Hyperparameter Analysis - 20 Marks

Hyperparameters within the neural network architecture are primarily the activation functions within the convolutional and dense layers.

- Common activation functions within the convolutional layers are either *relu* or *LeakyRelu*. These activation functions provide non-linearity to the layers which lead to better generalizations.
- Common activation functions for the fully connected dense layers are usually *relu* or *elu* which are utilized for the same purposes of non-linearity and generalization. Lastly, the activation function of the final layer for a multi-class classification is primarily *softmax*.
- The number of neurons or filters for the dense and convolutional layers respectively, are other parameters that can be tuned. The filters are usually powers with base of 2 (32, 64, 128, etc), neurons for dense layers can be varied based on the performance of the model.
- The last set of hyperparameters to tune are the loss function and optimizers which are usually variables within keras's *compile* function. The loss function and optimizer to start with would be *categorical crossentropy* and *adam*, respectively.

2.4 Step 4: Model Evaluation - 10 Marks

Model evaluation requires the assessment of the loss and accuracy performance of the model. These two components behave inverse to each other where the accuracy is expected to increase and the model loss to decrease. These variables will define the direction in which the model should be tuned to improve the performance of the model. An example of such plots is shown in Figure 2 below.

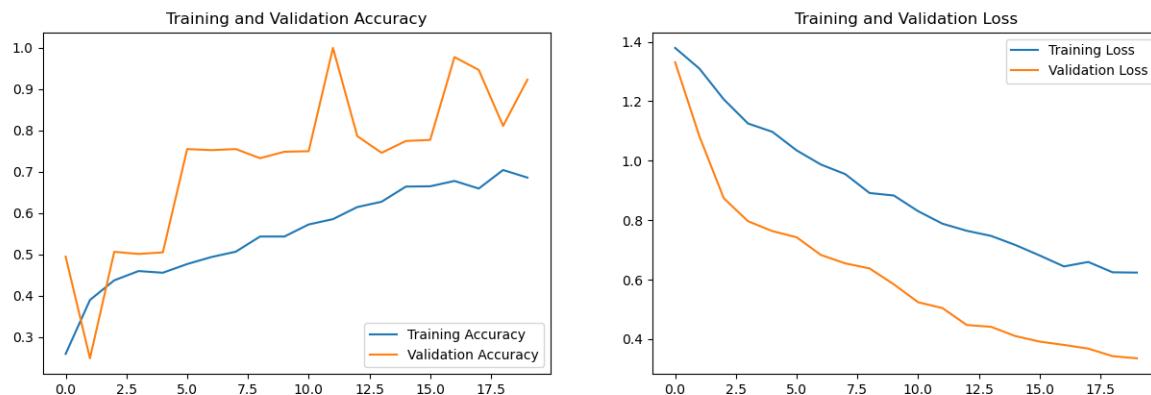


Figure 2: Model Performance Example

As seen in Figure 2, the loss and accuracy for both training and validation should be assessed to check for possibilities of overfitting.

2.5 Step 5: Model Testing - 20 Marks

The final component of this project is testing the model that was trained. This involves loading test images and predicting the class of that image from the model designed. Testing the model requires the following steps:

- Data processing of test images to convert the images to the format of that of the input data utilized for the model trained. Use the *image* package from keras preprocessing to load the image, convert the image to an array, and normalize it by dividing the image by 255.
- Since this is a multi-class classification problem and the final model layer uses the *softmax* activation function, you would need to find the maximum probability from the model's prediction.
- The final prediction of the image is expected to look similar to Figure 3 shown below.

True Crack Classification Label: paint-off
Predicted Crack Classification Label: paint-off

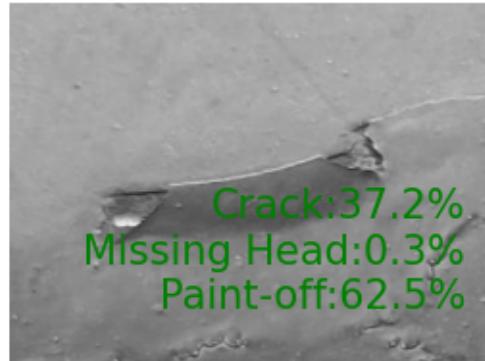


Figure 3: Model Testing Example

3 Submission

You are required to create two code files, which include one for steps 1-4 and another for step 5. Additionally, you are required to include the model performance and model testing images as shown in Figures 2 and 3 respectively. The test images to apply the model prediction will be: `"test_crack.jpg"`, `"test_missinghead.jpg"`, and `"test_paintoff.jpg"` under the `"test/crack"`, `"test/missing-head"` and `"test/paint-off"` folders, respectively.

Submission will be through GitHub. Create a new repository for this project and submit a report. The report should include the link to your GitHub repository, and 1-2 pages summary of your results (minimum 500 words and performance plots), justifications of the results, any noticeable edge cases in the test results (if applicable), and any discussion points you may want to bring up regarding your model's performance.