INDIVIDUAL ASSIGNMENT

CT100-3-M-DL

DEEP LEARNING

APDMF2112DSBA(DE)(PR)

NOVEMBER 2022

TITLE: CONCRETE SURFACE CRACK DETECTION BY DEEP LEARNING

LEE KEAN LIM

TP065778

LECTURER: PROF. DR. MANDAVA RAJESWARI

ABSTRACT

Structural maintenance plays an essential role in ensuring the stability and capacity of existing structures. Concrete is the most common type of material used to construct the core structural elements of a building. However, these concrete structures are often subjected to stress and strain induced by cyclic loading and weathering. Which induces the formation of cracks within the concrete. Therefore, structural inspections are often performed to detect crack formations, but such tasks are labor and time intensive. This study proposes the use of image processing to assist the concrete crack detection procedure to improve the efficiency and accuracy of the detection. Where an image classifier using convolutional neural network (CNN) is developed along the implementation of data augmentation to improve the classification performance. The CNN is developed using three layers of convolution and global average pooling to converge the output. The dataset utilized in this study consists of 20,000 images of concrete surface with crack and 20,000 images of concrete surface without crack. The data augmentation introduces noise to the images to simulate natural environment in a job site which is often hazy or lack of illumination. Random selection of pixels from each image is transformed to either a fully white or fully black pixel. The performance of the classifier is compared before and after introducing augmented data into the training set. It is identified that the classifier with the implementation of augmented data yields an outstanding accuracy of 99.22% while the classifier without the inclusion of augmented data yielded an accuracy of 95.54%. The study shows that the implementation of data augmentation can significantly improve the accuracy and robustness of the image classifier which can be adapted to real world application.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN..... Convolutional Neural Network

SIFT..... Scale-Invariant Feature Transform

BRIEF..... Binary Robust Independent Elementary Features

SURF..... Speeded-Up Robust Features

SECTION 1

INTRODUCTION

1.1 Background

Concrete beams and columns form the core structural elements for many buildings and structures. These elements function as a load transfer system to support the standing structure. Therefore, the concrete making up these elements experiences a high level of stress and strain. Concrete stress and strain are typically induced by cyclic loading, temperature change, and weathering. Eventually, these stresses can cause the concrete to crack, and the crack can propagate throughout the structure. Over time, the crack would become bigger and finally lead to the collapse of the structure. Therefore, very frequently structural inspections are performed to detect for signs of structural degradations and cracks. This is to ensure reliability and structural health can be maintained, and preventive measures can be taken to avoid structural failures which can endanger lives.

The task of structural crack detection is often time and labor intensive which is typically performed using two techniques namely invasive and non-invasive detection. The invasive technique often involves the use of specialized equipment operated by skilled personnel to perform the detection and testing on site or off site. Examples of invasive detection methods include ultrasonic examination, thermal testing, concrete sample testing in laboratory, etc. In addition, the results from these testing often require interpretation from structural experts. However, with the advancement of computing architecture and camera technology, non-invasive crack detection techniques such as the use of image analysis have been feasible and yield reliable results. The use of non-invasive crack detection techniques can significantly reduce the amount of labor required which enhances the work efficiency.

The use of image analysis for concrete crack detections involves the use of photos of concrete surfaces and a classifier to automatically identify whether cracks are present or absent. This typically combines the application of computer vision and deep learning to improve the speed and accuracy of the detection. This application has been increasingly applied in industry, specifically along with the utilization of robotics to provide a continuous structural inspection on jobsites. This significantly reduced the need for labor and assessment time while improving the safety and reliability of the assessment process.

Deep learning algorithms such as convolutional neural networks (CNN) have been widely applied in image analysis which facilitate the automation of image processing and cracks identification. Example in the works of Sorguç (2018), where different CNN architectures have been utilized to classify images of concrete into cracked and uncracked labels. The study shows the capability of CNN in image classification for concrete crack detections where the classifier was able to achieve an outstanding accuracy of more than 99%. In addition, the author released the dataset for researchers to further enhance and develop non-invasive image processing technique to be used in concrete crack detection.

The dataset has been widely adopted by various researchers and various classifiers have been developed with exceptional accuracies (Atkin, 2020; Chen et al., 2019; Le et al., 2021). However, Golding et al. (2022) mentioned the limitation of the dataset where the images are produced in a laboratory controlled environment. Where the images are mostly in high resolution and have standardized image capturing process. The author argues that such images are not a true representation of the actual condition in a jobsite. Thus, the classifier produced would typically not perform well when provided actual photos from a jobsite for classification.

Photos taken in jobsite conditions are typically prone to some noises. The noise would significantly affect the ability of the classifier to determine the presence of cracks on concrete surfaces. Therefore, this study proposes the use of data augmentation to introduce noises to a portion of the dataset to allow the classifier to learn noisy images. This is to produce a classifier with improved robustness and applicability for when actual images from a jobsite are presented.

1.2 Problem Statement

Public image datasets that are available for the study of detection of concrete surface cracks are often taken in a laboratory setting. Where high resolution cameras are used, optimal illumination provided, and angle of photo taken is consistent. This produces crisp and clear images which a prediction model can easily learn and classify with high accuracy. However, in a realistic jobsite condition, photos taken are often polluted by some type of noises such as shadows, blurred images, different angles, foreign substances on surface, etc. This results in classifier models that are developed using the laboratory produced dataset, often unable to perform when photos are received from a realistic jobsite. Therefore, to improve model performance and applicability in the industry, photos used for model training should reflect the actual condition on a jobsite. However, collecting such photos is very costly, thus this study proposes the use of data augmentation. As a representation of noisy images taken in an actual

jobsite condition to be used as the training data for the classifier to improve classification accuracy and model robustness.

1.3 Research Questions

To develop a practical image classifier for detecting the presence of concrete cracks on a set of images, the following questions will be addressed:

1. How will noises be introduced to the image dataset?
2. How will an image classifier be developed?
3. How will the classifier to be tested and evaluated?

1.4 Research Aim and Objectives

The aim of this study is to develop an image classifier to automatically classify images with the presence of cracks on concrete surface using deep learning and the introduction of noisy images as part of the training dataset to increase classification accuracy and model robustness. The following outlines the objectives of this study:

1. To introduce noises to the image dataset to simulate photos taken on an actual jobsite condition.
2. To design and develop an image classifier using deep learning algorithms with hyperparameter tuning to achieve optimal classification accuracy.
3. To evaluate performance of the classifier using evaluation metrics and compare the performance against related literatures.

SECTION 2

LITERATURE REVIEW

2.1 Computer Vision

Computer vision is a field of study that is focused on the development of techniques to enable computers to see and understand the contents of digital images or videos. Various formats of digital images can exist such as from video sequences, medical imaging data, multi-camera views, etc. To allow computers to understand images, it starts with the pixels which are the smallest possible unit of an image. These pixels represent the presence and intensity of the three primary colors namely red, green, and blue. A matrix of these pixels formulates an image, where the computers can apply operations on these matrices to gain an understanding of the images based on the spatial relationship between the pixels.

There are various tasks typically associated with computer vision, including but not limited to object classification, object detection, object segmentation, image restoration, and motion analysis. Table 2.1 shows the description for each of the mentioned computer vision tasks.

Table 2.1: Typical tasks in computer vision

Task	Description
Object classification	Identifies the class of an object in an image
Object detection	Identifies the class and location of an object in an image using classification and localization
Object segmentation	Identifies the boundary of an object in an image and then separating different objects into different regions
Image restoration	Noise removal from images
Motion analysis	Motion estimation of an object in a sequence of images

Application of computer vision can be seen in various domains and are increasingly being adopted as technology continues to advance and innovate. Domains that have applied computer vision in some way to assist their daily tasks include but not limited to manufacturing,

healthcare, automotive, retail, finance, and social media. Table 2.2 outlines some of the examples of how computer vision is utilized in each of the mentioned domains.

Table 2.2: Example of applications of computer vision in various domains

Domain	Computer vision Application
Manufacturing	Defect detection, safety inspection, product assembly, anomalies detection in machines, warehouse management optimization
Healthcare	Medical imaging analysis, medical imaging enhancement, biological organism tracking and counting, clinical decision support
Automotive	Autonomous vehicles, vehicle performance analysis, intelligent driving assistance, traffic detection
Retail	Surveillance, retail shelf analysis, store layout optimization, footfall analysis
Finance	Real estate valuation, document extraction, fraud detection, customer verification and authentication, insurance claim process
Social Media	Facial recognition, augmented and mixed reality, customer engagement analytics, content discovery, image tagging

The major process in computer vision is to extract features from images. The main features which are extracted from the image can include color, edges, and corners. These features form the fundamental components for a computer vision model to function effectively. Therefore, it is essential that the feature extraction phase is performed with utmost accuracy and quality. Traditionally, the feature extractions phase are performed using algorithms such as SIFT (Scale-Invariant Feature Transform), BRIEF (Binary Robust Independent Elementary Features), and SURF (Speeded-Up Robust Features) (Rani & Devi, 2020). However, these approaches are resource intensive which require the definition of feature structures and compositions for each image. Which becomes inefficient and impractical when the number of class labels increases.

With the advent of deep learning, the task of defining the feature structures and compositions is automated by the model. This significantly reduced the workload required and improved the scalability to allow the inclusion of a higher number of class labels. However, drawbacks of utilizing deep learning for a computer vision task is the requirement of large amount of training data and computational power. Figure 2.1 shows the process flow of a computer vision task using a traditional approach versus using the deep learning approach. Where it can be observed

that several steps in the traditional approach are automated and condensed into a single step when using the deep learning approach.

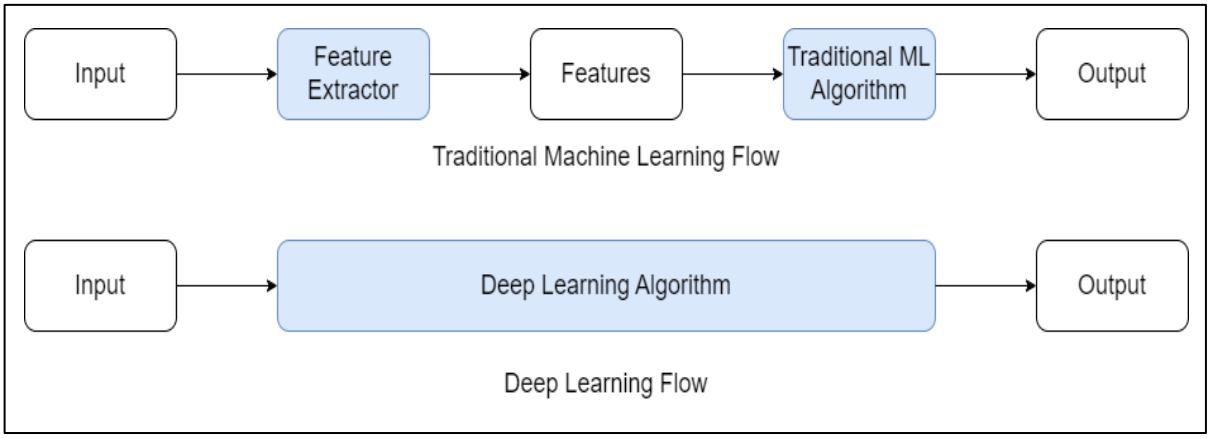


Figure 2.1: Machine learning versus deep learning flow for computer vision (Rani & Devi, 2020)

Various machine learning algorithms can be utilized in a computer vision task, such as k-means clustering, support vector machine, k-nearest neighbor, neural network, etc. However, the deep neural network, specifically the CNN is widely popular and adopted for computer vision related task (Khan & Al-Habsi, 2020). With the advancement of computing hardware and open-sourced artificial intelligence framework, the computation of a neural network model has become increasingly accessible and efficient. Where a deep neural network model can be trained in a matter of minutes as compared to the past where such computation would take days or weeks.

2.1.1 Computer Vision in Construction

Artificial intelligence has benefitted many domains and the construction domain is not left out either. Various tasks across all phases in the construction sector have started adopting artificial intelligence to enhance the accuracy, security, efficiency, quality, and safety of work. However, the use of computer vision is particularly notable in the jobsite of a construction. The following provides a brief overview of how computer vision has been utilized in the construction industry.

1. **Site Progress Measurement**

   Measuring the site progress provides a critical indicator to allow efficient resource management and keeping the project on schedule. The site progress can be measured using drones and robots equipped with cameras to take photos and videos of the

construction site. These data are then used to compare with a 3-dimensional digital model of the building to identify the work progress in real-time and potentially identifying any construction errors which early mitigation can be conducted to prevent cost and schedule overruns.

2. **On-Site Safety Monitoring**

   Accidents on site can significantly hinder the progress of the project. Upkeeping safety standards on a jobsite is essential but requires intensive labor and constant monitoring. The use of computer vision and cameras on-site allows the automation and continuous monitoring for safety hazards. Where potential safety risks will be flagged and informed to the safety personnel which immediate actions can be taken to mitigate risks and reduce accidents. Example, the monitoring of compliance of workers with the wearing of safety gear on-site, which can be detected by the cameras and a reminder can be issued to the worker to prevent unwanted accidents.

3. **Tracking Tools and Machinery**

   Different jobs require different tools, equipment, and machinery. This becomes a problem in a huge jobsite where the items are not properly stored and tracked, which results in low utilization, theft, and potentially forgotten that such resources are available. Where a vast amount of time is spent looking for the right and available tools on site to perform the job. Computer vision in combination with cameras on-site can assist in the task of identifying the location of the required object on the jobsite. This resulted in huge time saving, which enhanced the productivity of workers and utilization rate of the tools. In addition, this provides information to managers to manage resources more efficiently by allocating the right balance of resources to each jobsite.

4. **Machinery Assistance**

   Operators of machinery are often unable to keep track of their surroundings when performing their duty due to the high concentration required when performing their task. In addition, due to the size of the machinery, multiple and large areas of blind spots are an issue for the operators. Computer vision in combination with cameras on the machinery provides a continuous monitoring of the surroundings which provides a warning or even automatically stops the movement of the machines to avoid collisions resulting in on-site accidents. This enhanced the safety of operating machinery on site

in addition allowing the operators to focus on the critical task instead of looking out for the environment to prevent accidents. Moreover, less personnel are required to guide the operators through blind spots which frees up human resources for other tasks.

5. **On-Site Quality Inspection**

   Workmanship is a critical factor for all construction where the primary purpose is to ensure the structural soundness of the building. Work inspections are routinely performed daily on every task on-site to ensure the quality and safety of the completed work is delivered. However, the inspection job is labor intensive and time consuming in addition human error may miss out on several details during inspection. The usage of computer vision and robotics can assist in the work inspection process in which photos and videos are taken and analyzed to detect potential structural defects in real-time. Such technologies enhance the productivity and accuracy of work inspection which rectification can be performed at the early stage to prevent catastrophic accident.

2.2 Image classification

Image classification is a type of image processing task that accepts an input image and attempts to categorize the image into a predefined class label. The image classification task can be framed as a supervised learning or unsupervised learning problem. However, in this study the focus is only on the supervised learning image classification. An image classification model typically comprises of two phases namely model training and model evaluating. In the model training phase, a dataset of labeled images is provided to the model to undergo feature learning. While in the model evaluating phase, another set of unseen image datasets is utilized to evaluate the performance of the fitted model. Table 2.3 shows the strengths and weaknesses of several algorithms that are used for image classification.

Table 2.3: Algorithms used in image classification

Algorithm	Strength	Weakness
Convolutional Neural Network	- Suitable for both classification and regression task - Robust to outliers - Able to facilitate multi-class classification problem	- Model prone to overfitting - Requires extensive hyperparameter tuning for optimal model - Difficulty in interpretation of how the outcome is derived
Support Vector Machine	- Facilitate non-linear class boundaries - Model is less prone to overfitting - Computational complexity can be reduced to quadratic optimization problem	- Slower computation as compared to Naïve Bayes and decision trees - Hyperparameter tuning is difficult for non-linearly separable data - Difficulty in interpretation of how the outcome is derived
Fuzzy Logic	- Produce reasonable results without the requirement of precise inputs	- Domain expert is required to define the rules for the controller - Regular update of rules in the fuzzy logic controller is required
Logistic Regression	- Explainable derivation of the outcome - Easy implementation and efficient training - Results are quick to compute	- Performance limited to simple dataset and linearly separable data
k-Nearest Neighbor	- Instance based learning - Allows addition of new data to existing model - Easy to implement with only two parameters for tuning	- Performance limited to simple dataset and linearly separable data - Sensitive to noise - Identifying optimal K value is difficult
Random Forest	- Less prone to overfitting - Automatically impute missing values	- Computation is expensive due to computing numerous trees and combining them

Based on Table 2.3, several algorithms are mentioned which have been utilized in image classification tasks. However, not every algorithm is effective and efficient for image classification. Gavali and Banu (2019) compared the image classification performance of CNN, support vector machine, and fuzzy logic using the ImageNet dataset. It was found that the CNN model easily outperformed the support vector machine and fuzzy logic model. In addition, the author mentioned that neural networks can benefit from the utilization of graphics processing units to accelerate the computation which results in a lower computation time while yielding very accurate results. Therefore, with the combination of hardware acceleration and the

continuous innovation in the hardware, neural network algorithms are gaining popularity for their accuracy and speed. In addition, deep CNN is found to be more effective as compared to shallow CNN for computer vision tasks due to the higher-level feature extractions performed at the deeper layers however it does come with a cost of higher computation resources.

Image classification has been utilized in various fields. Example in healthcare, where an image classifier is developed to identify the presence of breast cancer in mammogram images (Zebari et al., 2019). In leather industry, where image classification is used to detect leather surface defects and to classify grade of leather (Aslam et al., 2019). In retail surveillance, where image classification is used to classify human gender based on gait energy images (El-Alfy & Binsaadoon, 2017).

2.2.1 Image Classification in Construction

Photos are often taken in construction projects for documentation and evidence of work completions. However, the pictures collected are stored in an unorganized manner. Where the photos are not properly categorized and indexed complicates the process of retrieving the photos at a later stage. Therefore, Gil et al. (2018) utilized image classification to partially automate the documentation task by automatically classifying photos taken into 27 different categories covering various construction trades. The Google Inception v3 classifier was utilized and trained on 1,208 images. Although a minimal number of images are used for training, the model obtained a substantial accuracy of 92.6%.

Detecting degradation in construction material is a labor-intensive task due to the vast coverage area and fine details of deterioration that can easily be neglected which requires a keen eye and the use of specialized equipment. Therefore, Fan et al. (2018) developed a model which utilized image classification to ease the detection process in identifying presence of cracks on pavements. The study proposed to split the original image into thousands of mini patches which will be used for model training. The intuition of the proposed method is that data are often difficult and costly to obtain and requires annotation for supervised learning. Cao et al. (2020) also mentioned the issue of costly data collection which then suggests the application of data augmentation such as rotation, flipping, mirroring, adding noise, changing illumination, etc. which can be an effective solution to the lack of training data problem. Thus, by using a small dataset of raw images, more data can be generated to facilitate the model training. The method yielded a substantial result which evaluated using precision (91.78%), recall (88.12%), and F1-score (89.54%). In addition, it was mentioned that photos taken in natural conditions typically

contain a lot of noises, such as oil spots, water stains, shadows, and non-uniform illumination. These noises in the images can significantly influence the performance of the prediction models.

In a similar work, Palevičius et al. (2022) attempts to perform image classification on concrete surface crack detection using an augmented dataset. Where shadows are artificially generated and blended with the images, creating additional data to be used for model training. The study shows that by combining augmented data with the original dataset to be used for model training, a more robust classifier model can be developed. Where the model using the proposed dataset achieves an outstanding accuracy of 99.41% while the model using the original dataset achieves only 90.45%. This shows the viability of using augmented data as an alternative for costly data collection from an actual jobsite. In addition, this promotes the applicability of image classification in the industry, which the model can differentiate complex and noisy data which represents realistic environment.

2.3 Convolutional Neural Network

CNN is initially designed for image recognition where the primary purpose of the network is to automatically learn and preserve the hierarchies of features from the image, which the feature extraction starts from the low-level feature and slowly builds up to a higher-level feature. It has gained popularity due to the notable results achieved in complex computer vision and natural language processing tasks. The capability of automated feature extraction and preserving the spatial relationship between pixels are the main reasons why CNN is so successful as compared to other algorithms. The basic building blocks of a CNN comprise of three layers namely convolutional layer, pooling layer, and the fully connected layer. Illustrated in Figure 2.2, shows the basic structure of a typical CNN.

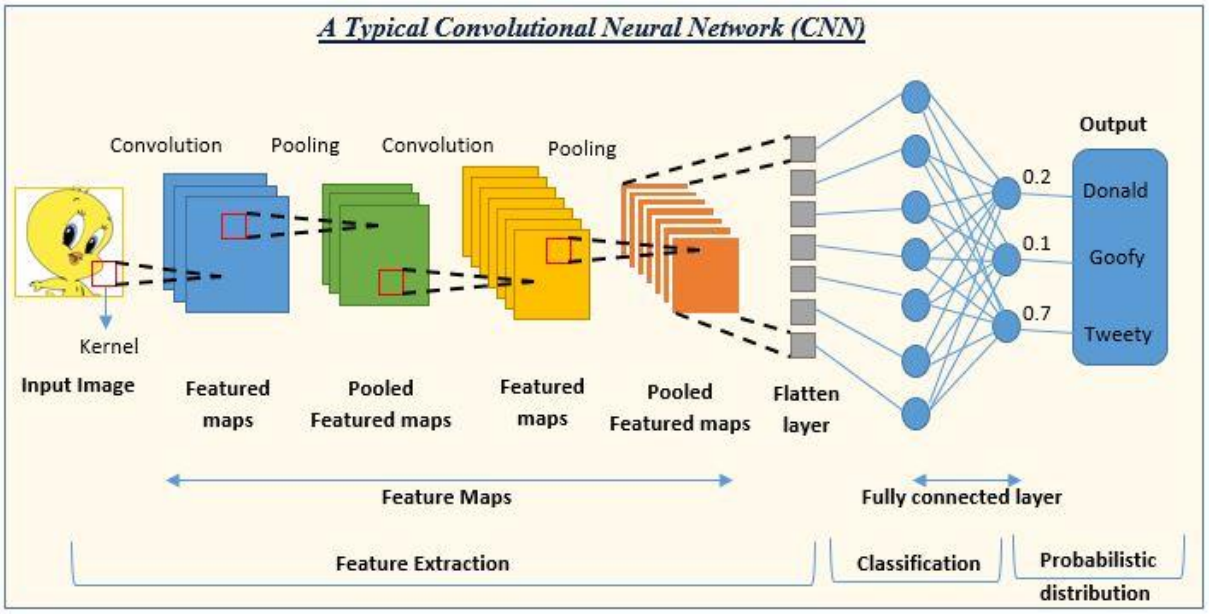


Figure 2.2: Basic building blocks of a typical CNN (Shah, 2022)

2.3.1 Convolutional Layer

The convolutional layer comprises of filters and feature maps which their primary purpose is to detect and extract features in images such as edges, color, lines, etc. The filter, also known as kernel, is a matrix of values that behaves like a template to detect a specific feature. The filter moves across the entire input image to detect the presence of a specific feature. An output is provided by the summation of the element-wise product of the matrices between the kernel and the input image. The output value represents the confidence of the presence for that specific feature, where a higher number indicates higher confidence, and a lower number indicates lower confidence. Therefore, several kernels are typically required to fully extract the features available from the input image. It is encouraged to use fewer numbers of filters at the early layers while increasing to a higher number of filters in the deeper layers. This is due to the increasing complexity of features to be extracted at the deeper layers. The higher number of filters would facilitate the feature extraction of more complex patterns such as circles, squares, or a mouth from a human.

Figure 2.3 illustrates a basic convolution operation based on a single filter, where the element-wise product between matrices results in a scalar output. The filter will move across the entire input image matrix and would compute the remaining values to complete the output matrix. The output matrix is also referred to as the feature map which contains the convolutions generated by the filter based on the input. In addition, the filter can have various sliding intervals which determine the distance moved by the filter in each step. This function is controlled using

the stride parameter. The stride value also indicates how much details to be reflected in the feature map. Where a smaller stride value would conserve more details from the input but with the cost of higher computational power. While a bigger stride would have the effect of downsampling the input where the smaller details are neglected. However, increasing the stride provides more efficient computation.

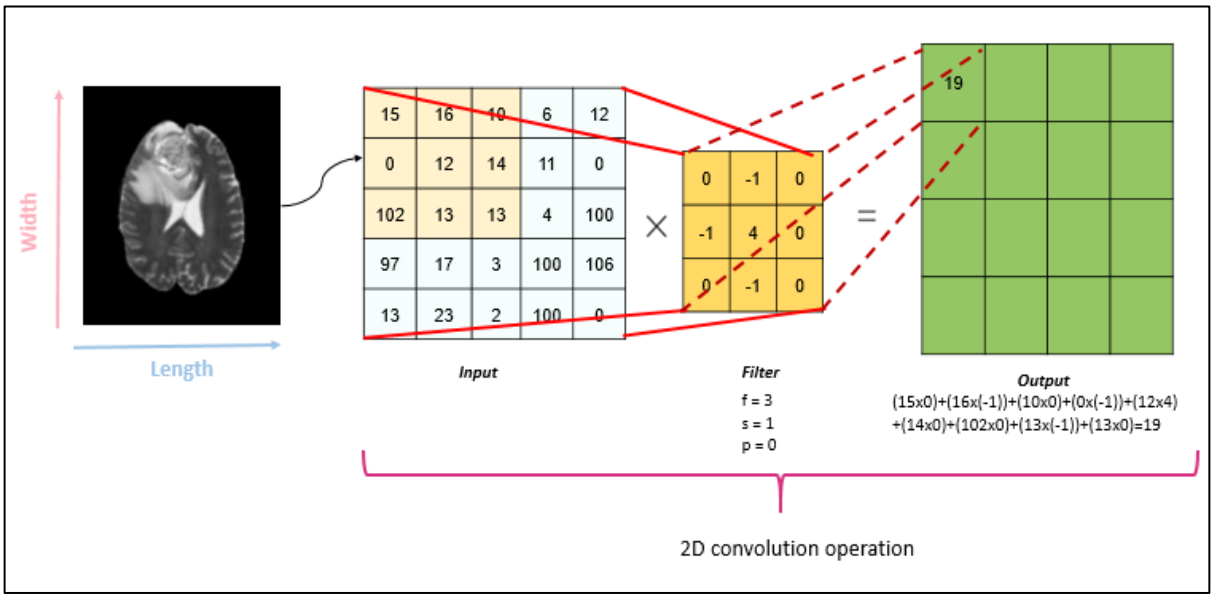


Figure 2.3: Convolution operation illustration (Yahyaoui, 2021)

The computation from the convolution operation would result in a smaller sized output as compared to the input. Example, in Figure 2.3 the input size is 5 x 5 while the output size is reduced to 4 x 4. As the input continues moving down the feature extraction chain, the output size would continuously reduce to a point where much of the information is lost due to the size reduction. This can be overcome by using padding, which introduces additional pixels to the perimeter of the input to conserve the output size. A typical technique called zero-padding is applied by adding additional pixels with zero values around the input perimeter. Combined with the right kernel size, the desired output size can be obtained. In addition, the benefits of applying padding would likely result in an improved model performance, as more of the information from the input can be retained and used in the deeper convolutional layers. However, the cost of retaining more information is the requirement of higher computational power. Illustrated in Figure 2.4, an example using input size of 5 x 5 and kernel size of 3 x 3 with one layer of zero-padding around perimeter would result in an output size same as the input size.

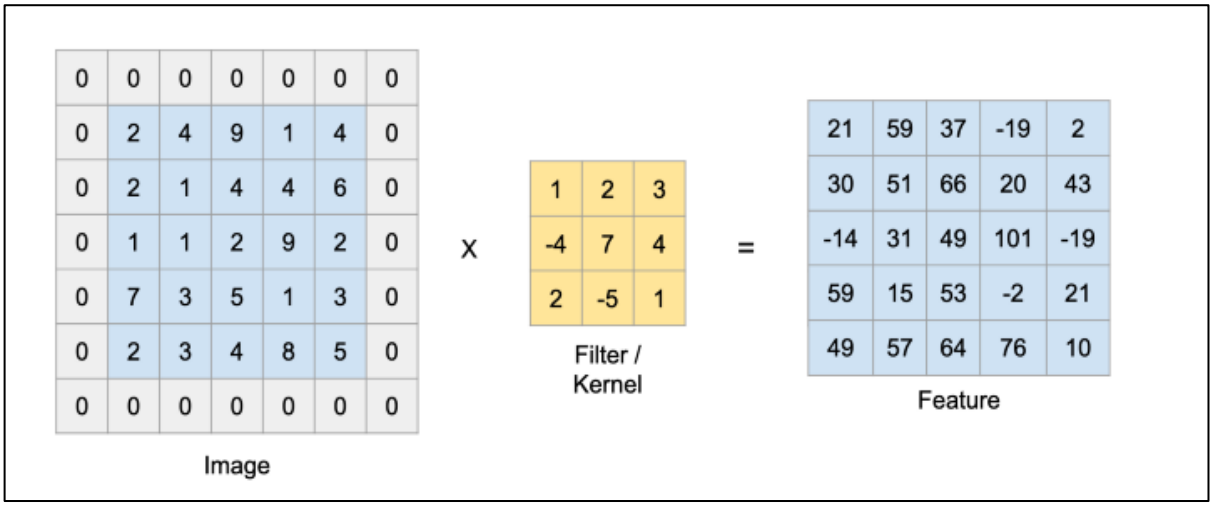


Figure 2.4: Padding operation illustration (Patel, 2019)

The dimension of the output from the convolutional layer can be represented by the following equations, where the padding layer, kernel size, and stride value are considered:

$$Height_{out} = \frac{Input_h + Padding_{h\_top} + Padding_{h\_bottom} - Kernel_h}{Stride_h} + 1 \quad (2.1)$$

$$Width_{out} = \frac{Input_w + Padding_{w\_top} + Padding_{w\_bottom} - Kernel_w}{Stride_w} + 1 \quad (2.2)$$

Where:

$h = height$

$w = width$

2.3.2 Pooling Layer

The pooling layer functions as a downsampling for the feature map, where the feature representations are consolidated. A limitation of the output from the convolutional layer is that small movements of the feature in the input image will likely result in a different feature map. By applying the pooling operation, a lower resolution of the input is generated, where finer details or noises will be neglected but the significant structural elements will remain. This results in a condensed feature map that is more robust to changes for features with slight movement in position. In addition, the downsampling feature is likely to facilitate the reduction of overfitting in the model.

The pooling operation functions like the filter operation in the convolutional layer. However, the filter in the pooling layer is typically kept at a size of 2 x 2 with a stride of two pixels. Which indicates that the size of the feature map will always be reduced by a factor of two after each pooling operation. There are two commonly used pooling operations namely average pooling and max pooling. Max pooling selects the maximum pixel value found in the receptive field while average pooling computes the average value based upon all the pixel values in the receptive field. Figure 2.5 illustrates the max pooling and average pooling in action. Where an input image of size 4 x 4 is reduced to a size of 2 x 2 for the output after the pooling operation.

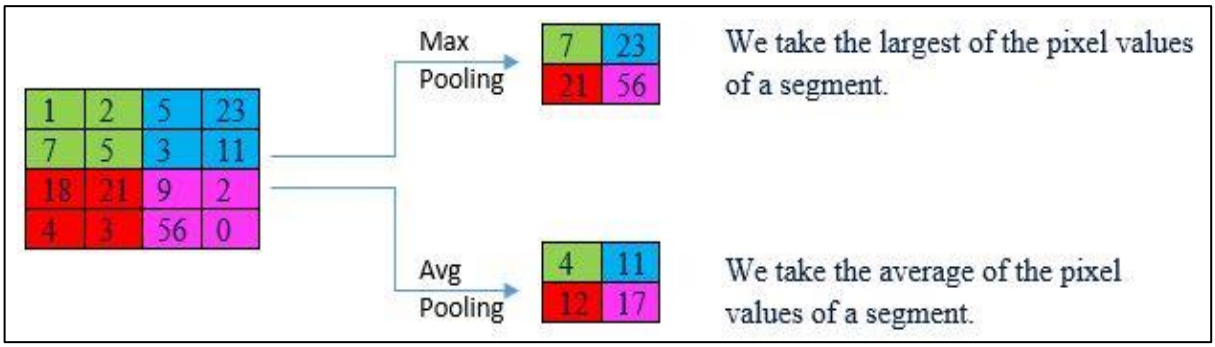


Figure 2.5: Max pooling and average pooling (Shah, 2022)

The values computed by the pooling operation represent the presence of a particular feature. Due to the selection of the maximum value among the pixels, the max pooling operation places higher weightage on the more prominent features in the receptive field. While for average pooling, a smoothed feature map is produced due to the use of the average value. Which in practical sense, average pooling is useful for reducing the size of the feature map while max pooling is useful for identifying prominent features from the input.

2.3.3 Fully Connected Layer

The fully connected layer is a typical feedforward neural network which is deployed after the feature extraction phase. It functions as the output layer which produces the probability for each prediction class. Prior to the fully connected layer, the feature map will need to undergo flattening to produce a feature vector. The flattened feature map will be used as the predictors for the feedforward neural network which then outputs the final prediction. Depending on the number of class labels, the output neuron would typically utilize the Softmax activation function or the Sigmoid activation function to produce the probability for a multiclass classification or a binary classification output respectively.

2.3.4 Model Architecture

The following section documents some of the recent advances of neural network models used in image classification. Table 2.4 tabulates the features and advances of some CNN models and the newly anticipated transformers. Attached under Appendix A, shows the figures displaying the framework and architecture for each of the development mentioned in the table.

Table 2.4: Compilation of recent neural network models in image classification

Network Name & Reference & Type	**Network Depth**	**Dataset**		**Features / Updates**	**Remark**
		Name	**Accuracy**		
EfficientNetV2 by Tan and Le (2021) Type: CNN	8	- ImageNet - CIFAR-10 - CIFAR-100 - Flowers - Cars	- 85.7 - 99.1 - 92.3 - 98.8 - 95.1	- Uses progressive learning where image size increases progressively throughout training - Replacing some of the MBConv layers in EfficientNet with Fused-MBConv layers - Dynamic scaling by adding more layers at later stages and restriction to maximum image size	- Faster training and better parameter efficiency than EfficientNet (11x faster & 6.8x smaller)
RepVGG by Ding et al. (2021) Type: CNN	22, 28	- ImageNet	- 78.78	- The body uses stacks of 3x3 Conv and ReLU only - Architecture does not have branches, uses a straightforward feed-forward like topology - Addition of parallel 1x1 Conv and an identity branch for each 3x3 Conv block	- 3x3 Conv is more efficient in computation as compared to 1x1 or 5x5 Conv - Single-path architecture is more efficient due to parallelism and consumes less memory - Faster than ResNet-50, ResNet-101, EfficientNet, RegNet

ResNet-RS by Bello et al. (2021) Type: CNN	105	- ImageNet	- 86.2	- Improvement to training method by adding Squeeze-and-Excitation and ResNet-D to architecture - Reduction of weight decay amount to reduce over regularization - Increase image resolution more gradually as larger image resolution yields diminishing return - Scale model depth in regimes where overfitting can occur which width scaling is better than depth scaling	- Faster than EfficientNet by 1.7x to 2.7x, uses less memory, and achieves similar result
ConvMLP by Li et al. (2021) Type: CNN-MLP	30, 45, 54	- ImageNet - CIFAR-10 - CIFAR-100 - Flowers	- 80.2 - 98.6 - 89.1 - 99.5	- Architecture combines the use of convolutional layers and multilayer perceptron layers - Stage-wise, co-design of convolutional layers and multilayer perceptron layers - Allows dynamic scaling of width and depth of architecture depending on input size	- Achieves comparable results with fewer parameters when compared to other methods - Performance improves consistently with increasing model scaling
ViT by Dosovitskiy et al. (2021) Type: Transformer	12, 24, 32	- ImageNet - CIFAR-10 - CIFAR-100 - Pets - Flowers	- 88.55 - 99.50 - 94.55 - 97.56 - 99.68	- Split images into fixed-size patches for processing - Uses standard transformer encoder which typically seen in natural language processing models	- Requires less computational resource to train - Achieves substantial result with pre-training on large dataset

CaiT by Touvron et al. (2021) Type: Transformer	24, 36, 48	- ImageNet - CIFAR-10 - CIFAR-100 - Flowers - Cars - iNat-18 - iNat-19	- 84.8 - 99.4 - 93.1 - 99.1 - 94.2 - 78.0 - 81.8	- Introduction of LayerScale which is applied to the output of each feed forward and self-attention block, allowing the control of weightage given to residuals - Introduction of class-attention layers to separate the information spreading and information routing layers	- Improves convergence and accuracy of transformers at larger depths - Novel architecture improves class embedding processing - Class-attention layers allow the optimization of each task separately which avoids contradictory signals towards learning objective
CrossViT by Fu et al. (2021) Type: Transformer	12, 15, 16, 18, 21	- ImageNet - CIFAR-10 - CIFAR-100 - Pets	- 82.8 - 99.11 - 91.36 - 95.07	- Introduction of dual-branch transformer to extract multi-scale feature representations - Introduction of feature fusion method using cross-attention module	- Cross-attention module achieves high accuracy while maintaining linear computation and memory complexity instead of quadratic - Ability to combine image patches of different sizes to produce stronger visual features
MViTv2 by Hao et al. (2022) Type: Transformer	10, 16, 24, 48, 80	- ImageNet	- 86.0	- Improvement to pooling attention mechanism that incorporated decomposed relative positional embeddings - Incorporated residual pooling connections in attention block	- Residual pooling compensates for the effect of pooling strides - Residual pooling reduces computation and memory complexity - Ability to process high-resolution visual input

When the number of parameters in a model decrease, this in turn leads to the reduction in model complexity. Which produces a smaller sized model that can easily fit into the memory of a computer of typical specification. However, the drawback of lesser parameters is the degradation of model performance. Therefore, this is the typical challenge faced when using neural networks, where the goal is to reduce parameters while maintaining comparable performance. A newly released type of neural network coined transformers has been gaining popularity due to the performance and ability to overcome typical problems faced in the existing neural networks models.

Initially developed for natural language processing, transformers have been developed to adapt in different applications. Transformers are proven to have the capacity to outperform CNN in image classification. In recent years, the utilization of transformers for image classification tasks has observed a significant increase as compared to using CNN. This is attributed to the faster computation, better accuracy, capacity to cater for bigger dataset, higher efficiency, and less computational resources required when using transformers as compared to CNN.

However, at the time of this study, several limitations are mentioned when using transformers. Such as, optimization methods for image transformers are difficult and still lacks in development (Touvron et al., 2021). In addition, hybrid approaches which utilizes the combination of CNN and self-attention, have limited scalability as compared to pure attention-based transformers (Fu et al., 2021). To achieve comparable results, vision transformers typically require a larger dataset to train the model which lead to expensive computation (Dosovitskiy et al., 2021). Moreover, Hao et al. (2022) mentioned that although transformers are highly capable in image classification, the application in high-resolution object detection remains a challenge due to the quadratic scaling in complexity of the self-attention blocks when the density of visual signals increases.

2.4 Challenges in CNN

In the current age where data generated is increasing exponentially, this led to the increasing demand for more efficient and effective machine learning algorithms. Which models are becoming more complex to cater for bigger dataset and producing higher accuracy. A model with high complexity would typically introduce more parameters which require tuning to achieve optimal model performance. Example of optimization parameters include learning rate, regularization, network architecture, etc. Generally, there is no fixed rules to conduct searching of optimal parameters, which often requires expensive computation through trial and error to

determine the optimal model (Obaid et al., 2020). Specifically in image classification using CNN, He (2020) mentioned that these parameters play a significant role in determining the performance of the model. Which include the number of neurons in each layer, number of hidden layers, type of activation functions used, type of optimization algorithms used, decay function, number of epochs, and batch size.

A typical problem faced when using CNN, particularly deep CNN, is the problem of model overfitting. Model overfitting can be derived from the poor model performance based on the evaluation using a testing dataset. Which signifies the poor generalizability of the model towards new and unseen data. However, this problem can be overcome by using regularization. Several regularization techniques can be adopted such as implementation of dropout layers or applying batch normalization layers. Batch normalization facilitates the model regularization by easing the internal covariate shift. Models utilizing batch normalization would typically compute faster and have lower affectability to initialization. However, using too much regularization may lead to model underfitting which undermine the performance of the model (Aggarwal & Kaur, 2018). Therefore, a balance point is to be achieved between applying too much or too little regularization to avoid overfitting and underfitting models.

To produce an optimal image classification model using CNN, a large amount of training data is typically required. Which Cha et al. (2017) mentioned that to produce a comparable model, at least 10,000 training data should be provided. However, the data collection process is often an expensive endeavor. In addition, the collected data requires preprocessing to prepare the data in a higher quality format and suitability to be used by the algorithm. To overcome this issue, data augmentation is generally applied, which also acts as a regularization for the model. It is a popular technique to artificially increase the size of a dataset. Additional data are created by means of transformation such as applying translation, reflections, color intensity alteration, noise introduction, etc. These additional data provide a larger pool of training data for the model in addition to providing variation of the original data which can improve the performance and robustness of the model against variance in unseen data.

The implementation of data augmentation also facilitates one of the shortcomings of CNN, where a slight variation of an image would likely result in the model producing a different outcome. In specific, the CNN do not encode position and orientation of the object. The same images can be taken from many different angles, different brightness, and having a slightly different background. Humans can easily identify the objects in such images, but for a CNN

model, it is a completely different image for every slight variation. Data augmentation produces images that are induced with some type of variations, which allowing the model to learn the variations will likely result in better model performance.

With the requirement of large training dataset and complex CNN architecture to cater for the need of high accuracy. This inevitably translates to the requirement of higher computational resources. The training of a CNN model, particularly a deep CNN would require a very long time. A hardware accelerator such as a graphics processing unit is typically encouraged to speed up the training process. This resulted in an additional cost requirement to train a huge network where immense computational power is required to keep the computation time feasible.

2.5 Related work

The following section documents the literature on structural defects detection using deep learning. In addition, network architecture for the mentioned literature will be attached under Appendix A. Table 2.5 tabulates the findings from the literature.

Table 2.5: Related literature in structural defects image classification

Reference	Methodology	Hyperparameter	Dataset & Size	Result Accuracy	Remark
Golding et al. (2022)	- Four datasets applying different augmentation techniques utilized for the study: (1) Original RGB (2) Grayscale (3) Otsu (4) Sobel Filter - Utilized VGG16 as the architecture and retrained using the augmented dataset	- Optimizer: Adam - Loss: Binary cross-entropy - Batch size: 32 - Epoch: 10, 20	Public Dataset: Concrete Crack Images for Classification - 20,000 with cracks - 20,000 without cracks (227x227x3)	10-Epoch (1) 99.43% (2) 99.33% (3) 98.85% (4) 99.07% 20-Epoch (1) 99.53% (2) 99.55% (3) 98.82% (4) 99.13%	- Using pretrained models can outperform CNN models trained from scratch - Study shows that models do not identify cracks based on color features - Suggest converting to grayscale for improving computational speed and reducing memory consumption
Palevičius et al. (2022)	- Introduction of shadow augmentation technique to the dataset - Utilized AlexNet as the architecture and retrained using the augmented dataset	- Optimizer: SGD - Batch size: 15 - Epochs: 25 - Learning Rate: 0.01	Public Dataset: Concrete Crack Images for Classification - 20,000 with cracks - 20,000 without cracks (227x227x3)	99.41%	- Shadow augmentation technique can be utilized as an alternative to expensive data collection - The proposed method produced a more robust classifier that performs well with noisy images - Reduced learning rate factor is beneficial to improve accuracy

Yang et al. (2021)	- Utilized AlexNet, VGGNet13, and ResNet18 as the architecture and retrained using the augmented dataset - Data collection considers environmental factors which induces noise on images such as strong light, moisture, distortion, and darkness - Original photo is 1024x1024, sliding window cropping is utilized to split into small images of 227x227 - Data augmentation using horizontal flip, vertical flip, rotation 180-degree, random zoom, random cut, and different brightness and saturation change.	- Learning rate: 0.001 - Epoch: 30	- Collected photos of concrete cracks - Preprocess photos to generate 20,000 images - 10,000 with cracks - 10,000 without cracks (227x227x3)	AlexNet: 97.6% VGG13: 98.3% ResNet18: 98.8%	- ResNet18 achieves the best performance among the three models - The deeper layers of ResNet18 likely resulted the better performance which linked to better feature extractions - Batch normalization and dropout used to prevent overfitting
(Le et al., 2021)	- Self-defined CNN architecture with 10 layers - Studies the applicability of CNN in image classification for concrete crack images	- Learning rate: 0.01 - Momentum: 0.9 - Epochs: 10	Public Dataset: Concrete Crack Images for Classification - 20,000 with cracks - 20,000 without cracks (227x227x3)	96.5%	- Models exhibit better performance when images have high contrast between cracks and background - Models could not detect crack when images are in low resolution, crack present in corner of image, crack size too small.

Zeeshan et al. (2021)	- Preprocessing by resizing image to 224x224 - Utilized VGG16 as the architecture and modified the last two convolutional blocks - Introducing dropout of 0.4 to the modified layers	- Optimizer: Adam - Learning rate: 0.0001 - Loss: Binary cross-entropy - Batch size: 32 - Epoch: 100	Public Dataset: SDNET2018 - 7,500 with cracks - 7,500 without cracks (256x256x3)	91.76%	- Lower learning rate likely to result in better accuracy - The modification resulted in a slight improvement to existing literature - The dropout addition to convolutional blocks 4 and 5 facilitate reduction of model overfitting
Atkin (2020)	- Uses a self-defined simple CNN architecture - Utilized global average pooling to replace dense layers	- Optimizer: Adam - Loss: Binary cross-entropy - Epochs: 100	Public Dataset: Concrete Crack Images for Classification - 20,000 with cracks - 20,000 without cracks (227x227x3)	97.17%	- No extensive hyperparameter tuning conducted - Simple architecture with two convolutional layers is sufficient to provide high accuracy
Chen et al. (2019)	- Study aims to identify optimal network architecture for image classification - Self defined CNN architecture by trial and error with extensive hyperparameter tuning	- Optimizer: Adam - Learning rate: 0.0001 - Dropout: 0.5	Public Dataset: Concrete Crack Images for Classification - 20,000 with cracks - 20,000 without cracks (227x227x3)	99.71%	- Use of Adam optimizer and batch normalization facilitate faster model convergence - Study shows that models perform better with images of high contrast between crack and background

Silva and Lucena (2018)	- Utilized VGG16 as the architecture and retrained using the dataset - Original dataset 851 photos of 756x756 sliced into smaller images of 256x256 - Studies the effect of learning rate, nodes in fully connected layer, and training dataset size in influencing the model accuracy	- Learning rate: 0.01, 0.1, 1.0 - Nodes in fully connected layer: 32, 64, 128, 256 - Training dataset size: 25%, 50%, 75%, 100%	Photographs taken from concrete specimens after test - 2,336 with cracks - 1,164 without cracks	92.27%	- Training dataset size has a strong influence on accuracy - Nodes in fully connected layer and learning rate has negligible influence on accuracy
Sorguç (2018)	- Original photo is 4032x3024, cropping is utilized to split into small images of 227x227 - Utilized AlexNet, VGG16, VGG19, GoogleNet, ResNet50, ResNet101, and ResNet152 as the architecture and retrained using the dataset - Study aimed to compare performance between using different pretrained networks	- Epochs: 10	Photographs taken from buildings in METU Campus - Preprocess photos to generate 40,000 images - 20,000 with cracks - 20,000 without cracks (227x227x3)	AlexNet: 89.6% VGG16: 99.6% VGG19: 99.1% GoogleNet: 90.0% ResNet50: 81.0% ResNet101: 86.0%	- In terms of performance, VGG16, VGG19, and GoogleNet provide best accuracies - In terms of convergence speed, AlexNet, VGG16, VGG19, and GoogleNet were able to converge quickly - Larger dataset used for training will likely result in higher accuracy - Models prone to overfitting when using high epochs - Number of convolutional layers is more significant than number of learnable

				ResNet152: 61.0%	parameters when high accuracy is of concern - AlexNet has difficulty transferring learned features to new cases - ResNet having more than 50 convolutional layers are prone to overfitting
Cha et al. (2017)	- 332 high resolution photos of concrete surface are taken in various conditions, where some contain cracks, and some do not - The high-resolution image is cropped into many smaller sized images (256x256) - The cropped images are used for training and testing of a classifier model - Self defined CNN architecture by trial and error with extensive hyperparameter tuning	- Optimizer: SGD - Batch size: 100 - Epoch: 60 - Weight decay: 0.0001 - Momentum: 0.9 - Dropout: 0.5	Photographs taken from buildings in University of Manitoba - 40,000 concrete images, but did not mention how many with and without crack present (256x256x3)	97.95%	- Costly data collection process as some places is hard to reach and safety is a concern - Suggest at least 10,000 images to be used for training to obtain sufficient model robustness - The developed model is robust for noisy images as it is trained with images under various environmental conditions

2.5.1 Discussion

Based on the literature survey, it can be observed that the majority of studies are utilizing the CNN algorithm to develop the image classifier for structural crack detections. Although the trend of using vision transformers for image classification in other domains is increasing, the adoption of the transformer-based network in structural crack detections is still lacking. However, the prediction accuracies from the CNN models are very satisfactory which typically achieves an accuracy of greater than 90%. Nevertheless, if a higher classification accuracy or higher model efficiency is desired, the vision transformers would likely be preferred over the CNN. However, more training data would be required with the use of vision transformers.

It can be observed that most studies utilize a pretrained network architecture instead of developing one from scratch. Which the studies showed that a pretrained network architecture typically performs better than developing a new network (Golding et al., 2022). In addition, developing a new network would require more computational resources and does not guarantee better results. It is mentioned that the number of layers in the network architecture significantly influences the classification outcome. Several architectures are identified to have better performance in concrete crack image classification task namely VGG16, VGG19, and GoogleNet. It is mentioned that the number of convolutional layers ranged between 16 and 22 is optimal for structural crack detection tasks. Which facilitated the feature extraction process thus resulted in the model achieving the best results (Sorguç, 2018).

As for the dataset used for concrete crack detection studies, several authors utilized the same dataset sourced from Sorguç (2018), where the name of the dataset is “Concrete Crack Images for Classification”. It is mentioned that the data collection process is tedious, and noises are typically present in the natural environment. Several studies recommended the utilization of data augmentation as an alternative to collecting more data. In addition, performing data augmentation would likely result in a better model which is more robust and more accurate as more training data and variation of data is provided for model training. It is shown in several studies that models using the augmented dataset performed better than models using only the original dataset (Golding et al., 2022; Palevičius et al., 2022; Yang et al., 2021). Furthermore, using augmented dataset would likely represent a more realistic application, where photos taken from a construction jobsite would typically be polluted by various noises.

While for the model hyperparameters, several studies have identified that certain hyperparameters provided a higher significance which highly influences the classification result.

These hyperparameters include learning rate, regularization, optimizers, and epochs. It is mentioned that a lower learning rate would likely result in better model accuracy. While for regularization, dropout layers and batch normalization are encouraged to be applied to facilitate faster model convergence and reduce model overfitting. In addition, several studies that utilized pretrained network architecture have identified that a low epoch number ranging between 10 to 30 is sufficient for the model to obtain substantial results. Multiple studies have utilized the Adam optimizer which facilitates faster model convergence.

SECTION 3

METHODOLOGY

3.1 Dataset

The dataset contains 40,000 images of concrete surface, where 20,000 images will have cracks present on the surface and 20,000 images will not have cracks present on the surface. Two class labels will be defined, where **positive** indicates images with a crack present and **negative** indicates images without a crack present. The images have a size of 227 x 227 pixels with RGB channels. Figure 3.1 shows a few sample images from the negative and positive class.

The dataset is derived from the works of Sorguç (2018) and Zhang et al. (2016) which can be retrieved from the following link: https://data.mendeley.com/datasets/5y9wdsg2zt/2

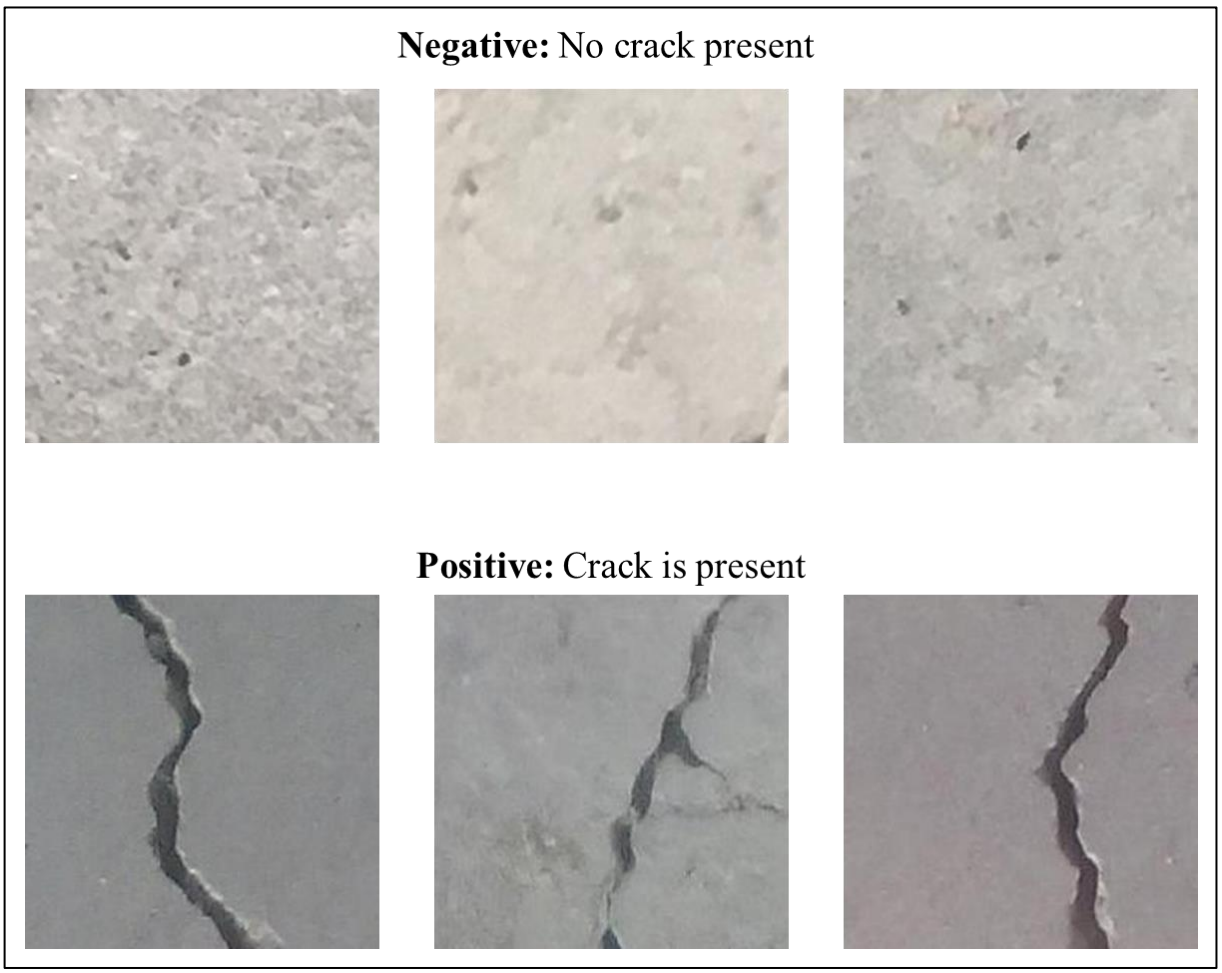


Figure 3.1: Sample data view

3.2 Data Preprocessing

This section outlines the data preprocessing tasks which comprise of data augmentation by means of noise introduction, resizing, and pixel normalization. In addition, data split which indicates portion of the dataset for specific use and the number of augmented data and original data in each subset will be mentioned.

3.2.1 Noise Introduction

The salt & pepper noise will be introduced to a portion of the dataset as a means of data augmentation. This type of noise is typically associated with the error in data transfer causing some pixels to be corrupted. However, it will be utilized in this study to represent noisy images, simulating photos taken in a construction jobsite which is typically polluted by some type of noises. The introduction of noisy images as part of the training dataset for the classifier would likely result in a more robust model.

A random selection of pixels accounting for 10% of the total number of pixels in each image will be converted to either a fully black or fully white pixel. A fully white pixel will have the pixel value of 255, while a fully black pixel will have the pixel value of 0.

3.2.2 Image Grayscale & Resizing and Pixel Normalization

The original images exist in the size of 227 x 227 pixels, which a downsizing is required to ensure the compatibility with the network architecture selected. Where the original images will be downsized to 224 x 224 pixels. The reduction in size will affect the quality of the images which will likely influence the classification outcome. However, since the reduction is small, the overall image quality can be conserved which influence to the outcome would not be significant. In addition, the smaller sized images provide the benefits of faster computation and lower memory consumption.

In addition, grayscale will be applied to the images to reduce computation cost which reduces the arrays from three to one. Applying grayscale to images would signify the loss of some information which may result in poorer performance. However, studies have shown that identifying cracks on concrete images are not dependent on the color features (Golding et al., 2022). In contrast, it is likely to improve the model performance when grayscale is applied.

The values of the pixels are normalized into the range between zero and one. The operation is performed by dividing each pixel value by 255. Where 255 is the representation of the total

number of color code available. The normalization of the pixel values facilitates the model training efficiency which helps the model to train faster.

3.2.3 Data Split

Two datasets will be prepared in this study where each dataset is utilized for a different experiment. The experiments will be compared to study the model performance before and after introducing noisy data as part of the model training.

The first experiment will have model training and validation using samples from only the original dataset. While for model testing, samples will be used from both augmented and original dataset. For the second experiment, the model training, validation, and testing will be using samples from both augmented and original dataset. Table 3.1 summarizes the composition of the dataset for each experiment.

Table 3.1: Dataset composition for each experiment

	Dataset Composition	
	Experiment 1	**Experiment 2**
Training	Original	Augmented + Original
Validation	Original	Augmented + Original
Testing	Augmented + Original	Augmented + Original

For both experiments, 30,000 images will be used for training, 5,000 images for validation, and 5,000 images for testing. The following tabulates the compositions of the dataset used in each experiment. Where Table 3.2 represents the data composition for experiment 1 and Table 3.3 represents the data composition for experiment 2.

Table 3.2: Data composition for experiment 1

	Positive		**Negative**		**Total**
	Original	**Augmented**	**Original**	**Augmented**	
Training	15,000	0	15,000	0	30,000
Validation	2,500	0	2,500	0	5,000
Testing	1,250	1,250	1,250	1,250	5,000

Table 3.3: Data composition for experiment 2

	Positive		Negative		Total
	Original	**Augmented**	**Original**	**Augmented**	
Training	7,500	7,500	7,500	7,500	30,000
Validation	1,250	1,250	1,250	1,250	5,000
Testing	1,250	1,250	1,250	1,250	5,000

3.3 Model Architecture

A simple CNN architecture with three layers of convolution and global average pooling will be utilized in this study to produce the image classifier. The number of convolutional layers is kept low to ensure feasible computation time while providing sufficient feature extraction capability to produce a reasonable image classifier. The global average pooling is chosen as opposed to the typical fully connected layers for producing the output. This is due to the number of parameters produced by the fully connected layers are significantly more than the global average pooling which increases the computation time. In addition, utilizing the global average pooling would reduce the number of hyperparameters for tuning. With the large dataset utilized in this study, the global average pooling is more favorable as compared to the fully connected layers when computation time is of concern.

3.3.1 Hyperparameter

A typical challenge when utilizing neural networks is the tuning of the hyperparameters. Different datasets typically require a different combination of hyperparameters to develop an optimal prediction model. In this study, the layers within the GoogleNet architecture will remain as it is with only slight modification to the output layers to fit the number of class labels aligning with this study. The network will be trained using the datasets outlined in section 3.2. Due to the size of the dataset and the network, only two hyperparameters namely learning rate and epoch will be tuned in this study.

Learning Rate

The learning rate signifies the scaling of the magnitude of parameter updates during gradient descent. Which controls the step size each time the model weights are updated. A higher learning rate would facilitate faster model training. However, it may result in a sub-optimal model where the loss function is less likely to be the global minima or resulting in an unstable training process. While a lower learning rate is more likely to be able to identify the global minima, it requires much more computational resources and a higher number of epochs to

complete the training. Four learning rates will be utilized in this study and their performance will be compared to identify the best performing model. The learning rates are: 1e-3, 1e-4, 1e-5, and 1e-6.

Epoch

The epoch signifies the number of passing of the full dataset through the network. Example of one epoch would indicate that the dataset is passed forward and backward through the network once. A lower number of epochs would hinder the model from completing the learning process which likely result in an underfitting model. However, a higher number of epochs may result in model overfitting, where the model would perform well on the training data but performs poorly on new and unseen data. Three different epochs will be utilized in this study and their performance will be compared to identify the best performing model. The epochs are: 50, 100, and 200.

3.4 Model Evaluation

Several evaluation metrics will be utilized to evaluate the performance of the classifier namely accuracy, precision, recall, and the confusion matrix. These metrics provide an indication of the predictive capability of the classifier. In addition, performance between developed models can be compared to identify the optimal model based on best use case.

Confusion Matrix

The confusion matrix is a table visualization of the model performance typically for a classifier. It is applicable to binary classification tasks but can be extended to accommodate multiclass classification tasks. Table 3.4 illustrates the confusion matrix. For a binary classification task, there will be four quadrants in the confusion matrix:

- True Positive (TP): When actual prediction is True and predicted outcome is True.
- True Negative (TN): When actual prediction is False and predicted outcome is False.
- False Positive (FP): When actual prediction is False but predicted outcome is True.
- False Negative (FN): When actual prediction is True but predicted outcome is False.

Table 3.4: Confusion matrix

		Predicted Values	
		Negative	**Positive**
Actual Values	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Accuracy

Accuracy is the direct measurement representing the total number of correct predictions against the total number of predictions made. A high accuracy value would indicate better prediction capability of the model as compared to a lower accuracy value. However, relying only on the accuracy indicator may not present a true view of the capability of the classifier. Thus, the sensitivity and specificity metrics are often utilized to complement the model performance evaluation. The following shows the equation of the accuracy metric:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

Recall

Recall measures how well the model identified the actual positives. A high recall value would indicate fewer false negative produced by the model. The following shows the equation of the sensitivity metric:

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

Precision

Precision measures how well the model identified the predicted positives. A high precision value would indicate fewer false positive produced by the model. The following shows the equation of the specificity metric:

$$Specificity = \frac{TP}{TP + FP} \tag{3.3}$$

SECTION 4

IMPLEMENTATION

4.1 Noise Introduction

The task involves introducing noises to the images, specifically the salt and pepper noise will be introduced. A random selection of 10% of the total image pixels will be converted to either black or white pixel. Figure 4.1 shows the code snippet of the function utilized to perform the noise introduction to the images.

```
# Function for adding salt and pepper noise to image

def add_noise(img, percent=10):

  # Getting dimension of the image
  row, col, rgb = img.shape

  # Random selection of pixels to turn into white color
  # Original image comes in 227 x 227 pixels in RGB channels
  # Percent signify the percentage of pixels to be randomly applied noise
  limit = percent/100 * (227*227)

  for i in range(int(limit)):
    # Random y coordinate selection
    y_coor = random.randint(0, row-1)

    # Random x coordinate selection
    x_coor = random.randint(0, col-1)

    # Apply white color to the randomly selected pixel
    img[x_coor][y_coor] = [255, 255, 255]

  for i in range(int(limit)):
    # Random y coordinate selection
    y_coor = random.randint(0, row-1)

    # Random x coordinate selection
    x_coor = random.randint(0, col-1)

    # Apply black color to the randomly selected pixel
    img[x_coor][y_coor] = [0, 0, 0]

  return img
```

Figure 4.1: Code snippet salt and pepper noise function

The original images are loaded into the environment one by one, and noises are introduced to each image using the function. Once the operation is completed, the augmented image will be saved in another folder for future use. Figure 4.2 illustrates a sample from each positive and negative class, showing the effect before and after applying noises.

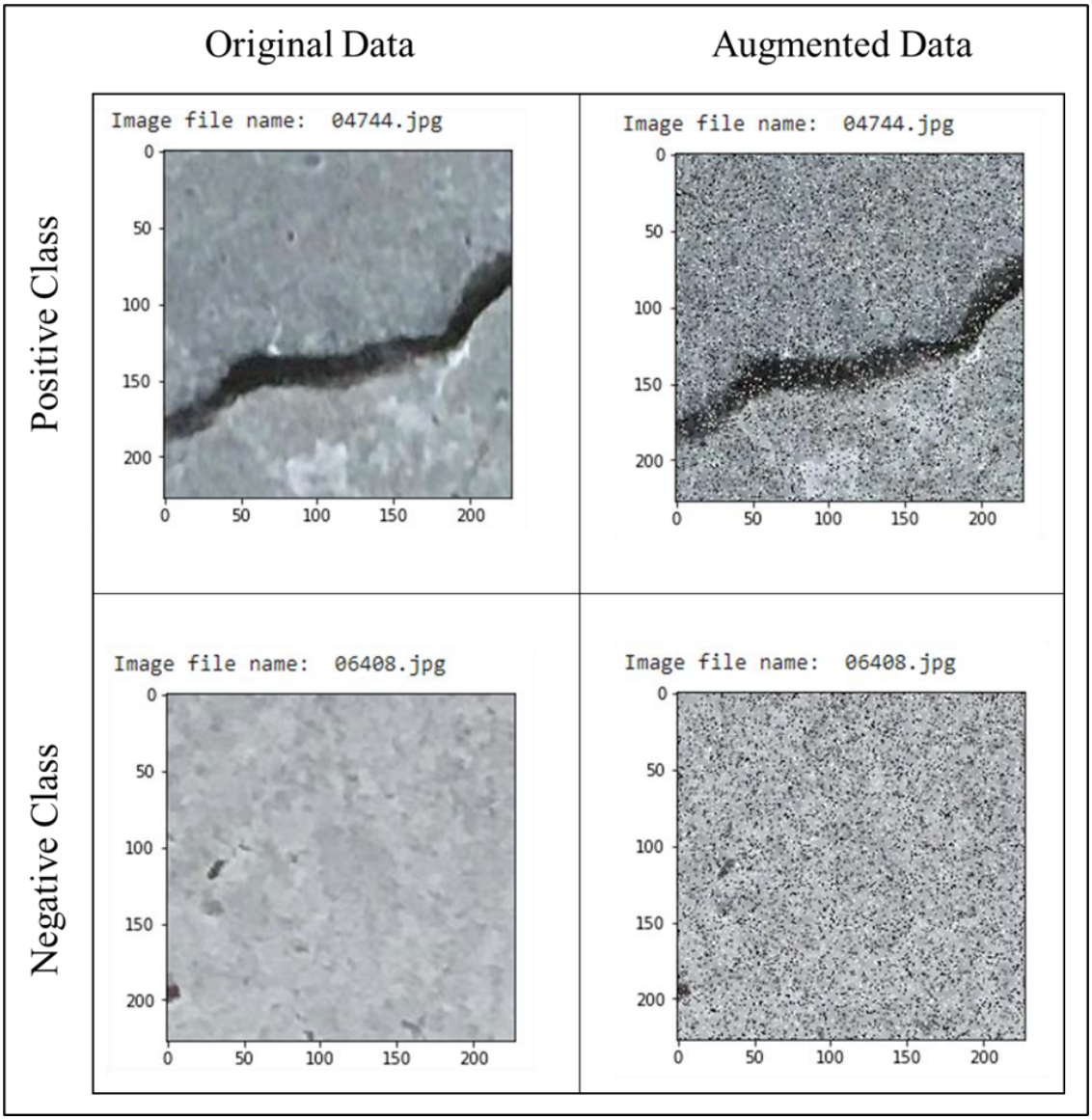


Figure 4.2: Before and after noise introduction

4.2 Directory Folders

The ImageDataGenerator from Keras will be utilized to assist the batch processing and batch augmentation of the images. Thus, a setup of folders in the directories is required to facilitate the process. The directories will contain the data subsets, categorized by training, validation, and testing dataset. In addition, within each branch, a positive class folder and a negative class folder will contain the actual images. Figure 4.3 illustrates the folder structure for each experiment.

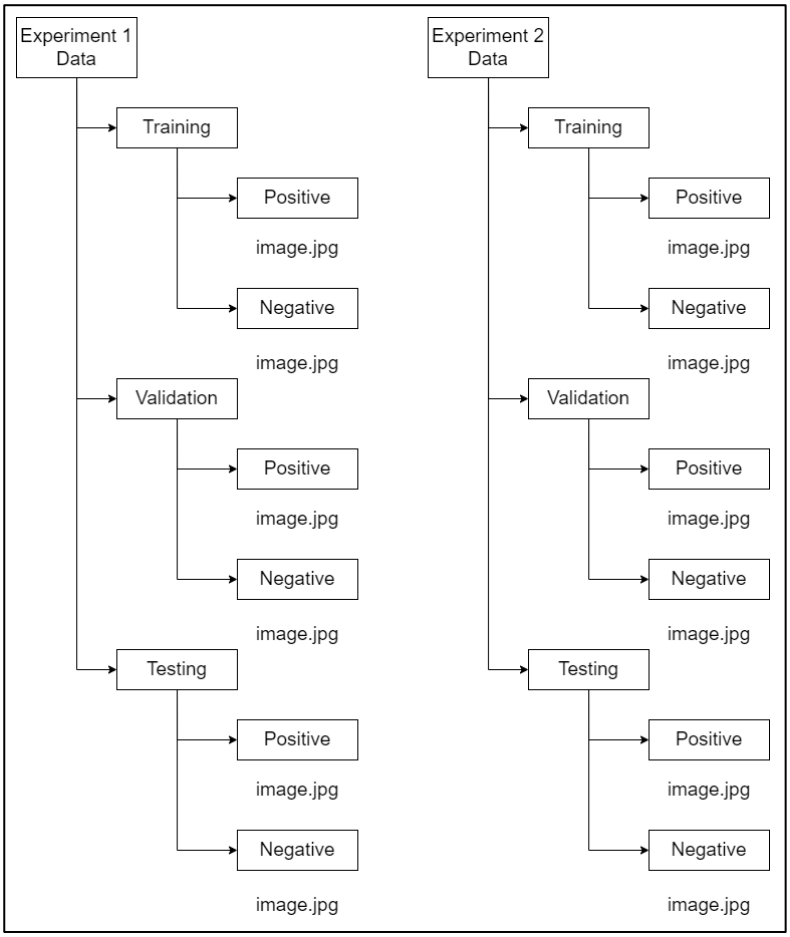


Figure 4.3: Dataset folder structure

Figure 4.4 and Figure 4.5 show the quantity of images in each subfolder for experiment 1 and experiment 2 respectively. It can be observed that each class has an equal number of images which signify a balanced dataset.

```
# Checking number of saved images in Gdrive
if display_file == True:
  train_p = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Train/Positive/*"))
  train_n = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Train/Negative/*"))
  val_p = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Val/Positive/*"))
  val_n = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Val/Negative/*"))
  test_p = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Test/Positive/*"))
  test_n = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Test/Negative/*"))

  print('Quantity of positive class images in training set saved: ', train_p)
  print('Quantity of negative class images in training set saved: ', train_n)
  print('Quantity of positive class images in validation set saved: ', val_p)
  print('Quantity of negative class images in validation set saved: ', val_n)
  print('Quantity of positive class images in testing set saved: ', test_p)
  print('Quantity of negative class images in testing set saved: ', test_n)
```

```
Quantity of positive class images in training set saved:  15000
Quantity of negative class images in training set saved:  15000
Quantity of positive class images in validation set saved:  2500
Quantity of negative class images in validation set saved:  2500
Quantity of positive class images in testing set saved:  2500
Quantity of negative class images in testing set saved:  2500
```

Figure 4.4: Image quantity in each subfolder for experiment 1

```
# Checking number of saved images in Gdrive
if display_file == True:
  train_p = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E2_Train/Positive/*"))
  train_n = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E2_Train/Negative/*"))
  val_p = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E2_Val/Positive/*"))
  val_n = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E2_Val/Negative/*"))
  test_p = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E2_Test/Positive/*"))
  test_n = len(glob.glob("/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E2_Test/Negative/*"))

  print('Quantity of positive class images in training set saved: ', train_p)
  print('Quantity of negative class images in training set saved: ', train_n)
  print('Quantity of positive class images in validation set saved: ', val_p)
  print('Quantity of negative class images in validation set saved: ', val_n)
  print('Quantity of positive class images in testing set saved: ', test_p)
  print('Quantity of negative class images in testing set saved: ', test_n)
```

```
Quantity of positive class images in training set saved:  15000
Quantity of negative class images in training set saved:  15000
Quantity of positive class images in validation set saved:  2500
Quantity of negative class images in validation set saved:  2500
Quantity of positive class images in testing set saved:  2500
Quantity of negative class images in testing set saved:  2500
```

Figure 4.5: Image quantity in each subfolder for experiment 2

4.3 ImageDataGenerator

Several operations will be performed onto the images utilizing the ImageDataGenerator from Keras. The operations include pixel rescaling, image resizing, and color setting. These operations are performed with the purpose of improving computation efficiency to optimize the usage of processing units and memory. In addition, the batch size for the generator will be fixed at 50 due to the memory limit of the machine. Figure 4.6 shows the code snippet of the ImageDataGenerator with the mentioned operations. It can be observed from the output that the training set contains 30,000 images, validation set contains 5,000 images, and the testing set contains 5,000 images. Each of the subsets has two classes which signify the positive and negative class. Figure 4.7 shows the shape and data type of a single generated batch of data. From the output, it can be observed that the image has reduced from three channel RGB to a single channel grayscale.

```
batch_size = 50

# Folder paths
path_train_e1 = '/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Train'
path_val_e1 = '/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Val'
path_test_e1 = '/content/drive/MyDrive/Colab Notebooks/APU - Deep Learning/Assignment/Data Generator/E1_Test'

# Datagen arguments for pixel normalization
datagen_args = ImageDataGenerator(rescale=1./255)

# Training set Data Generator
train_generator_e1 = datagen_args.flow_from_directory(path_train_e1,
                                                      target_size=(224, 224),
                                                      color_mode='grayscale',
                                                      batch_size=batch_size)

# Validation set Data Generator
val_generator_e1 = datagen_args.flow_from_directory(path_val_e1,
                                                    target_size=(224, 224),
                                                    color_mode='grayscale',
                                                    batch_size=batch_size)

# Testing set Data Generator
test_generator_e1 = datagen_args.flow_from_directory(path_test_e1,
                                                     target_size=(224, 224),
                                                     color_mode='grayscale',
                                                     batch_size=1)
```

```
Found 30000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
```

Figure 4.6: Code snippet of ImageDataGenerator

```
# Checking data shape and type
x1, y1 = next(train_generator_e1)
print('x: ', type(x1))
print('y: ', type(y1))
print('x: ', x1.shape)
print('y: ', y1.shape)

# Class Index
print(train_generator_e1.class_indices)
```

```
x:  <class 'numpy.ndarray'>
y:  <class 'numpy.ndarray'>
x:  (50, 224, 224, 1)
y:  (50, 2)
{'Negative': 0, 'Positive': 1}
```

Figure 4.7: Code snippet of image batch from ImageDataGenerator

The following shows the augmented images generated by the ImageDataGenerator. Figure 4.8 shows the generated images for experiment 1 where noises are not included. While Figure 4.9 shows the generated images for experiment 2 where noises are introduced to a portion of the images.

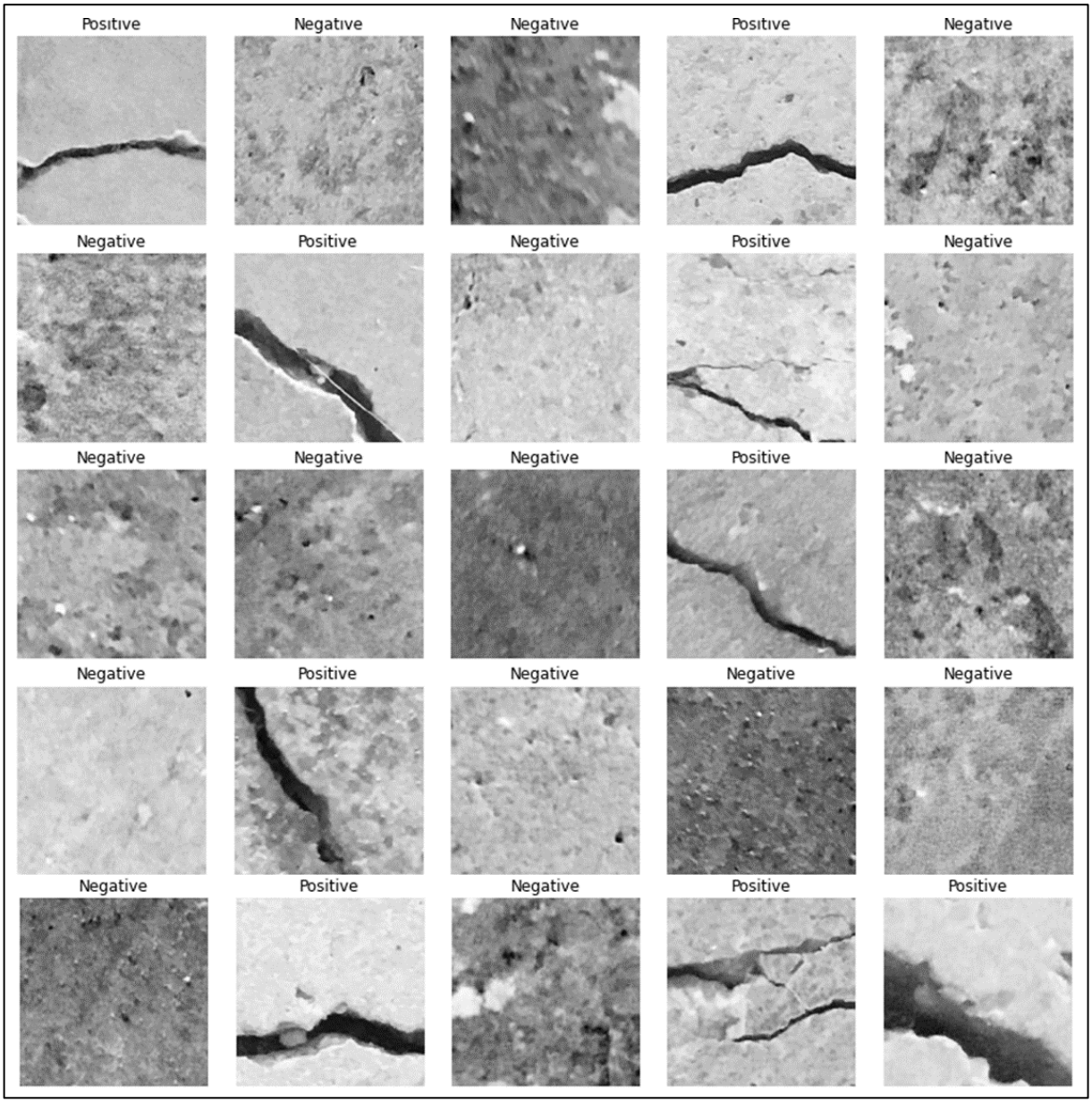


Figure 4.8: Image samples after augmentation for experiment 1

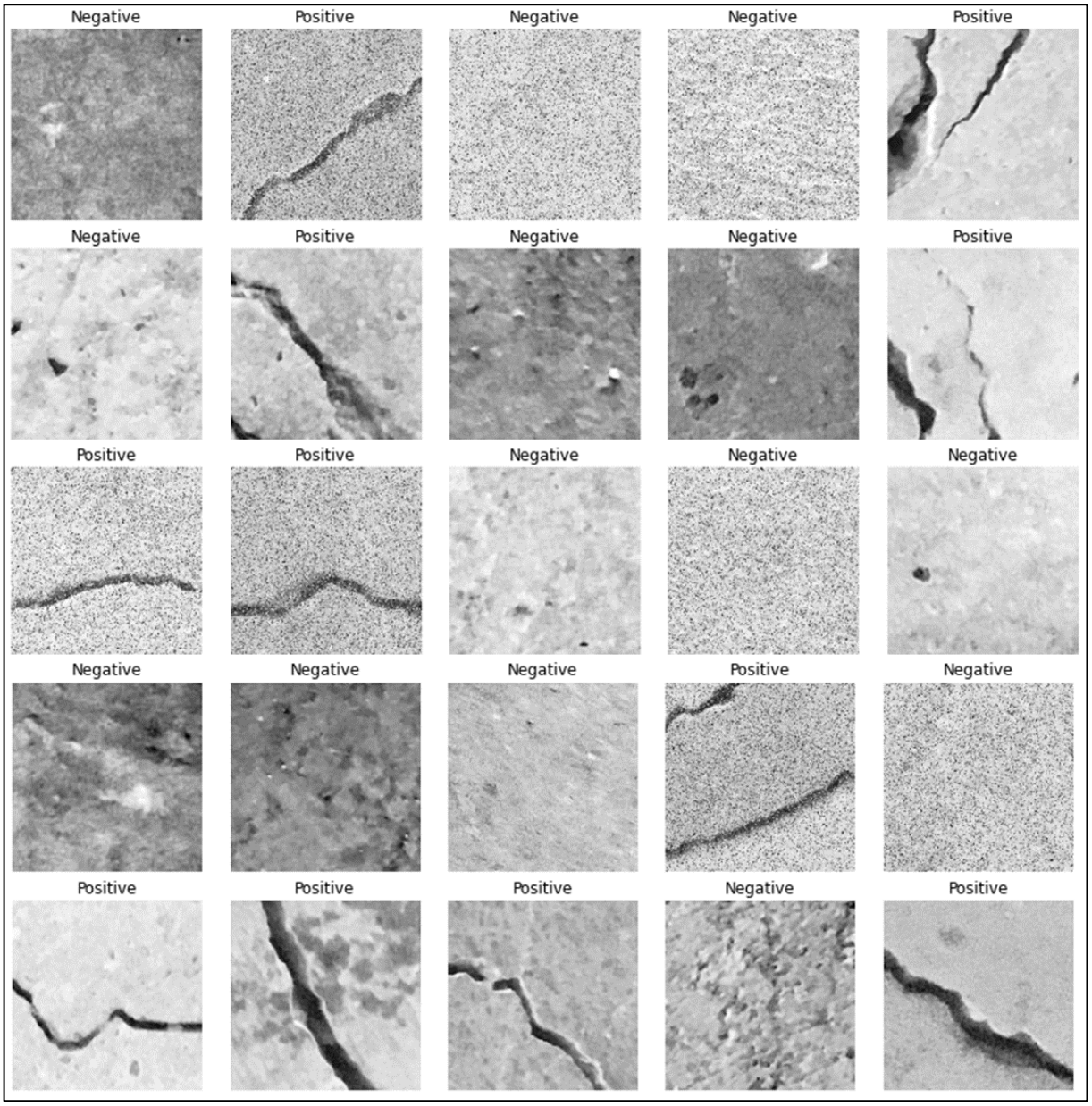


Figure 4.9: Image samples after augmentation for experiment 2

4.4 Model Development

The model is developed using three convolutional layers followed by global average pooling to produce the output. Several parameters are fixed in the convolutional layers such as padding is applied, filter size is 3 x 3, and uses ReLu activation. The number of neurons in the first convolutional layer is 32 units, increases to 64 units in the second layer, and finally increases to 128 units in the third layer. After each convolutional layer, a max pooling layer is applied.

After three layers of convolution, the global average pooling is utilized to generate one feature map corresponding to each class label. The resulting vector would be utilized by the Softmax

layer for the classification. Two neurons are present in the Softmax layer signifying two class labels. Figure 4.10 shows the code snippet of the development of the sequential CNN model. While Figure 4.11 shows the model summary which the model has a total trainable parameter of 92,930. This is significantly less than conventional models that utilize fully connected layers.

```
# A simple CNN architecture with 3 layers of convolution and
# Uses global average pooling in replace to fully connected layer
def verySimpleCNN():
  model = Sequential()
  # Convolutional 1 - 32 neurons with 3x3 filters
  model.add(Conv2D(32,3,padding="same", activation="relu", input_shape = (224,224,1)))
  model.add(MaxPool2D())
  # Convolutional 2
  model.add(Conv2D(64, 3, padding="same", activation="relu"))
  model.add(MaxPool2D())
  # Convolutional 3
  model.add(Conv2D(128, 3, padding="same", activation="relu"))
  model.add(MaxPool2D())
  # Global average pooling replacing dense layer
  model.add(GlobalAveragePooling2D())
  # Output
  model.add(Dense(2, activation="softmax"))

  return model
```

Figure 4.10: Code snippet for developing the CNN sequential model

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      320

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 128)      0
 2D)

 global_average_pooling2d (G  (None, 128)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 2)                 258

=================================================================
Total params: 92,930
Trainable params: 92,930
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.11: Model summary

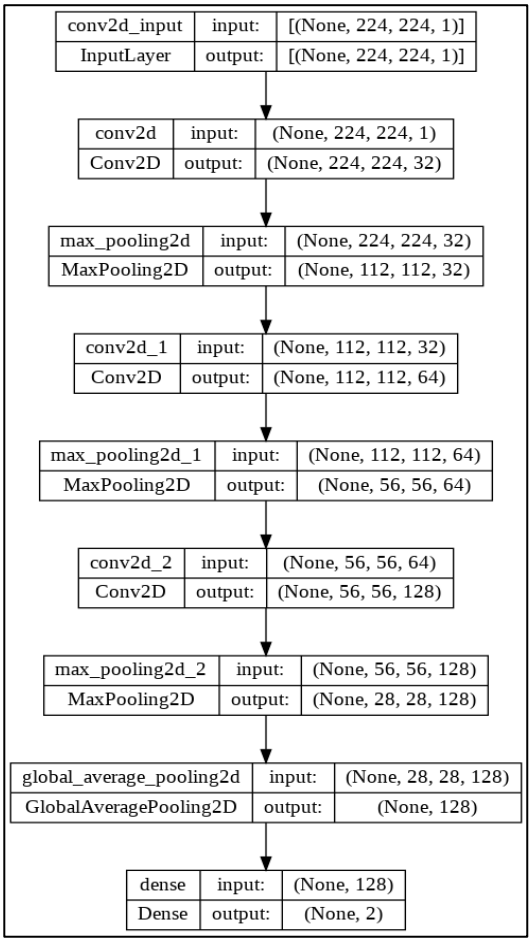


Figure 4.12: Illustration 1 of the CNN model architecture

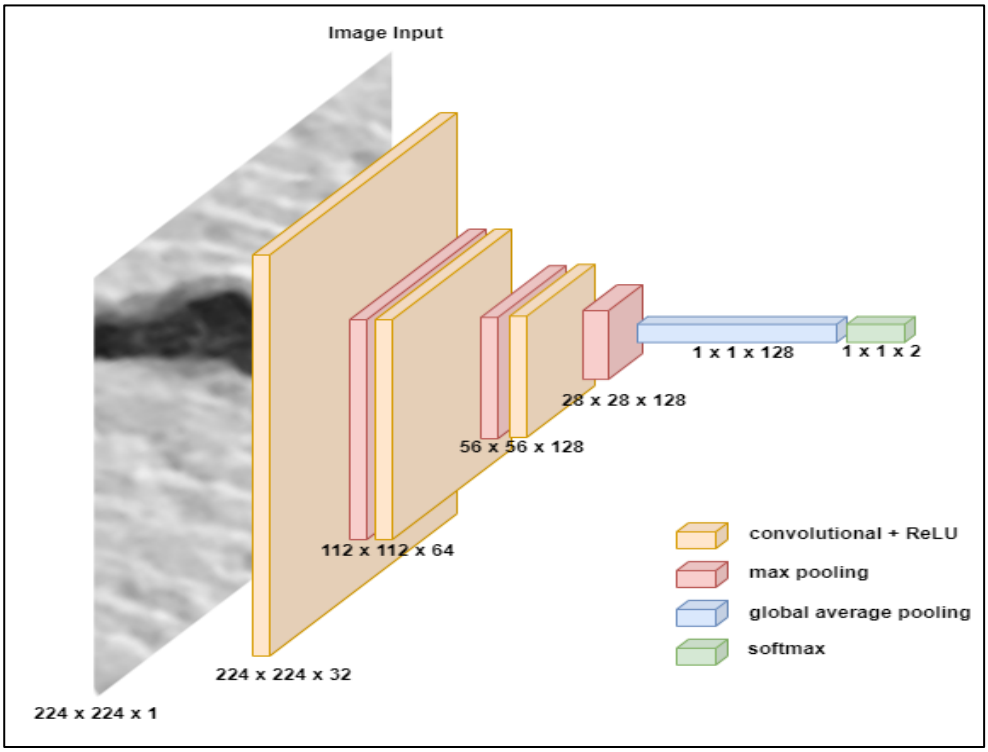


Figure 4.13: Illustration 2 of the CNN model architecture

Figure 4.12 and Figure 4.13 show the alternative illustration of the model architecture. These figures have a clearer indication of the input size to each of the layers.

```
for epochs in (10, 20, 30): # 10, 20, 30
  for learning_rate in (1e-3, 1e-4, 1e-5, 1e-6): # 1e-3, 1e-4, 1e-5, 1e-6

    keras.backend.clear_session()
    model = verySimpleCNN()

    # Optimizer learning rate setting
    opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)

    # Compiler
    model.compile(loss=['categorical_crossentropy'],
                  optimizer=opt,
                  metrics=['accuracy'])

    # Model fitting
    model_fit = model.fit(train_e1,
                          steps_per_epoch=math.ceil(train_generator_e1.samples // batch_size),
                          epochs=epochs,
                          validation_data=val_e1,
                          validation_steps=math.ceil(val_generator_e1.samples // batch_size))
```

Figure 4.14: Code snippet for model fitting

Figure 4.14 shows the code snippet for developing the model fitting process. The procedure is confined within a nested loop of the iterating hyperparameters of epochs and learning rates. The Adam optimizer and categorical cross entropy are utilized as the optimization algorithm and loss function respectively. The model is fitted with the training data and validation data obtained from the ImageDataGenerator. While the batch size for each epoch is computed by dividing the number of samples by 50, which is the number of samples in each generator. Each of the developed models is saved for future analysis.

4.4.1 Model Training Losses

Models in experiment 1 and experiment 2 have undergone hyperparameter tuning. Based on the iterating hyperparameters of learning rates and epochs, a total of 12 models are developed for each experiment. Table 4.1 and Table 4.2 show the model training and validation losses for experiment 1 and experiment 2 respectively. Attached under Appendix B, shows the plots of the training and validation loss for each model.

Table 4.1: Training and validation loss for models in experiment 1

Learning Rate	Epochs					
	10		20		30	
	Training Loss	Validation Loss	Training Loss	Validation Loss	Training Loss	Validation Loss
1e-3	0.043280	**0.03676**	0.02010	**0.02238**	0.01679	**0.01647**
1e-4	0.08089	0.06910	0.05382	0.05814	0.04385	0.04507
1e-5	0.23167	0.19929	0.12994	0.11418	0.11782	0.09188
1e-6	0.67463	0.67262	0.65735	0.65521	0.60399	0.59625

Table 4.2: Training and validation loss for models in experiment 2

Learning Rate	Epochs					
	10		20		30	
	Training Loss	Validation Loss	Training Loss	Validation Loss	Training Loss	Validation Loss
1e-3	0.05714	**0.05740**	0.03233	**0.04133**	0.02398	**0.03021**
1e-4	0.10309	0.09159	0.08127	0.08122	0.05503	0.07517
1e-5	0.45594	0.43565	0.34352	0.32389	0.25876	0.23283
1e-6	0.68493	0.68472	0.67789	0.67585	0.66274	0.65903

Based on Table 4.1 and Table 4.2, it is observed that for both experiments, all the models achieved the lowest validation loss when a learning rate of 1e-3 is utilized. While decreasing the learning rate, the model performance is observed to be degrading. In addition, referring to the loss plots, models using a lower learning rate are observed to be converging at a slower pace. Which signifies more epochs are required for models utilizing lower learning rates.

4.5 Model Evaluation

The model performance is evaluated by four metrics namely accuracy, precision, recall, and confusion matrix. These metrics are computed by passing the testing dataset to each model for inference and compared with the actual labels. Table 4.3 and Table 4.4 show the accuracy, precision, and recall for each model for experiment 1 and experiment 2 respectively. Selection of the optimal model for each experiment will prioritize models that produced a high recall rate. This is due to a higher recall rate which would result in fewer false negative which is favorable for the concrete crack detection use case.

Table 4.3: Evaluation metrics for models in experiment 1

Model Number	Learning Rate	Epochs	Accuracy	Precision	Recall
E1-1	1e-3	10	0.7454	0.6636	**0.9952**
E1-2	1e-4		0.7386	0.6612	0.9788
E1-3	1e-5		0.7248	0.6527	0.9608
E1-4	1e-6		0.6184	0.5895	0.7796
E1-5	**1e-3**	**20**	**0.9554**	0.9664	0.9436
E1-6	1e-4		0.7418	0.6627	0.9848
E1-7	1e-5		0.7338	0.6580	0.9736
E1-8	1e-6		0.6310	0.5940	0.8276
E1-9	1e-3	30	0.9388	**0.9863**	0.8900
E1-10	1e-4		0.7414	0.6627	0.9832
E1-11	1e-5		0.7374	0.6598	0.9800
E1-12	1e-6		0.6518	0.6016	0.8984

Table 4.4: Evaluation metrics for models in experiment 2

Model Number	Learning Rate	Epochs	Accuracy	Precision	Recall
E2-1	1e-3	10	0.9808	0.9967	0.9648
E2-2	1e-4		0.9712	0.9892	0.9528
E2-3	1e-5		0.8500	0.9211	0.7656
E2-4	1e-6		0.6558	0.7547	0.4616
E2-5	1e-3	20	0.9896	**0.9984**	0.9808
E2-6	1e-4		0.9776	0.9885	0.9664
E2-7	1e-5		0.8996	0.9587	0.8352
E2-8	1e-6		0.6552	0.7563	0.4580
E2-9	**1e-3**	**30**	**0.9922**	0.9947	**0.9896**
E2-10	1e-4		0.9762	0.9954	0.9568
E2-11	1e-5		0.9264	0.9793	0.8712
E2-12	1e-6		0.6628	0.7834	0.4500

Based on Table 4.3, the overall best model identified for experiment 1 is **model E1-5**. Which achieves the highest accuracy (95.54%), a very satisfactory precision (96.64%), and high recall (94.36%). The optimal model is developed using 20 epochs with 1e-3 learning rate.

Based on Table 4.4, the overall best model identified for experiment 2 is **model E2-9**. Which achieves the highest accuracy (99.22%), a very satisfactory precision (99.47%), and the highest recall (98.96%). The optimal model is developed using 30 epochs and 1-e3 learning rate.

It is observed that the default learning rate of 0.001 resulted in the best model performance. However, a higher number of epochs can be experimented to potentially allow better model convergence which likely result in a better model.

4.5.1 Model Fitting

This section outlines the model fitness for the optimal model identified in the previous section. Shown in Figure 4.15 and Figure 4.16, the model loss plot for the optimal model identified for experiment 1 and experiment 2 respectively.

It can be observed from the figures that both models were able to achieve a good fit on the dataset. Where initial training, a steep descent on the losses is observed followed by a gradual descent and finally stabilizes at a low level of losses. In addition, for both models, the training loss and validation loss at the end of the epoch arrives at almost a similar value, which signifies an optimal model fitness.

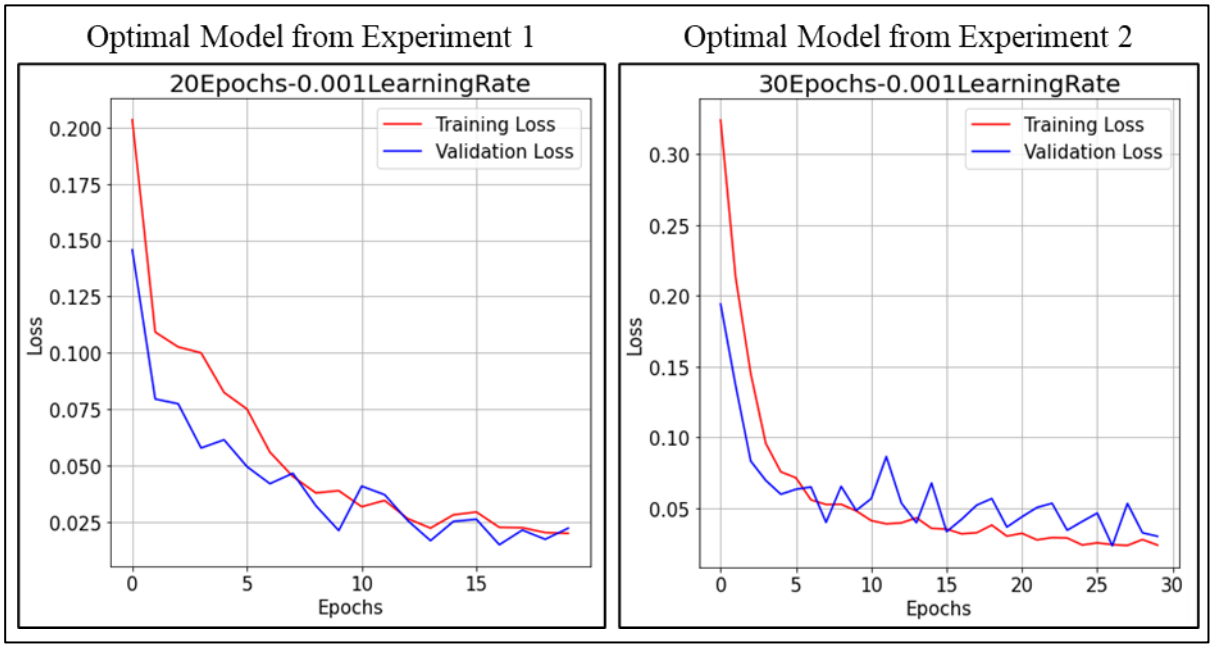


Figure 4.15: Model E1-5 loss plot of experiment 1 (Left) and Model E2-9 loss plot of experiment 2 (Right)

SECTION 5

RESULTS AND DISCUSSION

5.1 Summary Result

Table 5.1 shows the optimal model along with the best hyperparameters and evaluation metrics for each experiment. The performance between the models will be compared and discussed in the following section.

Table 5.1: Evaluation metrics of optimal models from experiment 1 and experiment 2

Optimal Model	Learning Rate	Epochs	Accuracy	Precision	Recall	Confusion Matrix
Experiment 1 – E1-5	1e-3	20	0.9554	0.9664	0.9436	<table><tr><td></td><td></td><td colspan="2">**Predicted**</td></tr><tr><td></td><td></td><td>**Negative**</td><td>**Positive**</td></tr><tr><td rowspan="2">**Actual**</td><td>**Negative**</td><td>2418</td><td>82</td></tr><tr><td>**Positive**</td><td>141</td><td>2359</td></tr></table>
Experiment 2 – E2-9	**1e-3**	**30**	**0.9922**	**0.9947**	**0.9896**	<table><tr><td></td><td></td><td colspan="2">**Predicted**</td></tr><tr><td></td><td></td><td>**Negative**</td><td>**Positive**</td></tr><tr><td rowspan="2">**Actual**</td><td>**Negative**</td><td>2487</td><td>13</td></tr><tr><td>**Positive**</td><td>26</td><td>2474</td></tr></table>

5.2 Discussion

To reiterate, experiment 1 uses original unaugmented data for training while uses a mixture of original and augmented data for testing. For experiment 2, a mixture of original data and augmented data is utilized for training and testing. The study aims to develop an improved image classification model by introducing artificially augmented data for training to improve the model performance when realistic noisy images are used. The purpose is to enhance the practical application of image classifier in the construction setting.

For the use case of structural defect detection, an image classifier with higher recall rate is more favorable. A high recall rate indicates a lower number of false negative predictions from the classifier. False negative predictions signify that in actual condition, a crack is present on the concrete, but the classifier indicates that no crack is present. The inability to identify cracks present on the concrete is critical. Any cracks present on the main structure should be treated with utmost attention to avoid calamity. Therefore, classifier for structural defect detections, is more favorable to have a high recall rate as compared to a high accuracy or precision rate.

Based on Table 5.1, it is observed that experiment 2 provided a better performance where every evaluation metric has obtained a higher value than experiment 1. Based on the confusion matrix for experiment 1 (model E1-5), it is observed that a high number of false negatives (141) and false positives (82) is produced. This is likely due to the inability of the classifier to classify noisy images which resulted in a high number of false negatives and false positives. In addition, the classifier produced a higher number of false negatives as compared to false positives. This signifies that the classifier has poorer performance when given images with crack present on concrete surface. The classifier would produce a result indicating that a crack is not present, but in actual the images are present with crack. This is likely the result from the lack of noisy images utilized for model fitting. Thus, resulted in the classifier mistakenly classifying that a crack does not present on the concrete surface. Model from experiment 1 would perform poorly in practical application, where photos taken on a construction site are likely to contain some noises.

Based on the confusion matrix for experiment 2 (model E2-9), it is observed that a lower number of false negatives (26) and false positives (13) is produced when compared to model E1-5. The improved performance indicates the ability of the classifier to classify noisy images which resulted in lower false negatives and false positives. The classifier produced a lower number of false positives as compared to false negatives. This signifies that the classifier has better performance when given images without crack present on the concrete surface. Where

the classifier would produce a more accurate result indicating a crack is not present, given that actual images are without a crack present. The improved performance is likely attributed to the introduction of augmented data as part of the model fitting phase. The classifier from experiment 2 would perform better in the industry. This is due to the ability to better classify noisy images. Therefore, this shows the potential of utilizing augmented data in image classification to improve model performance for practical settings.

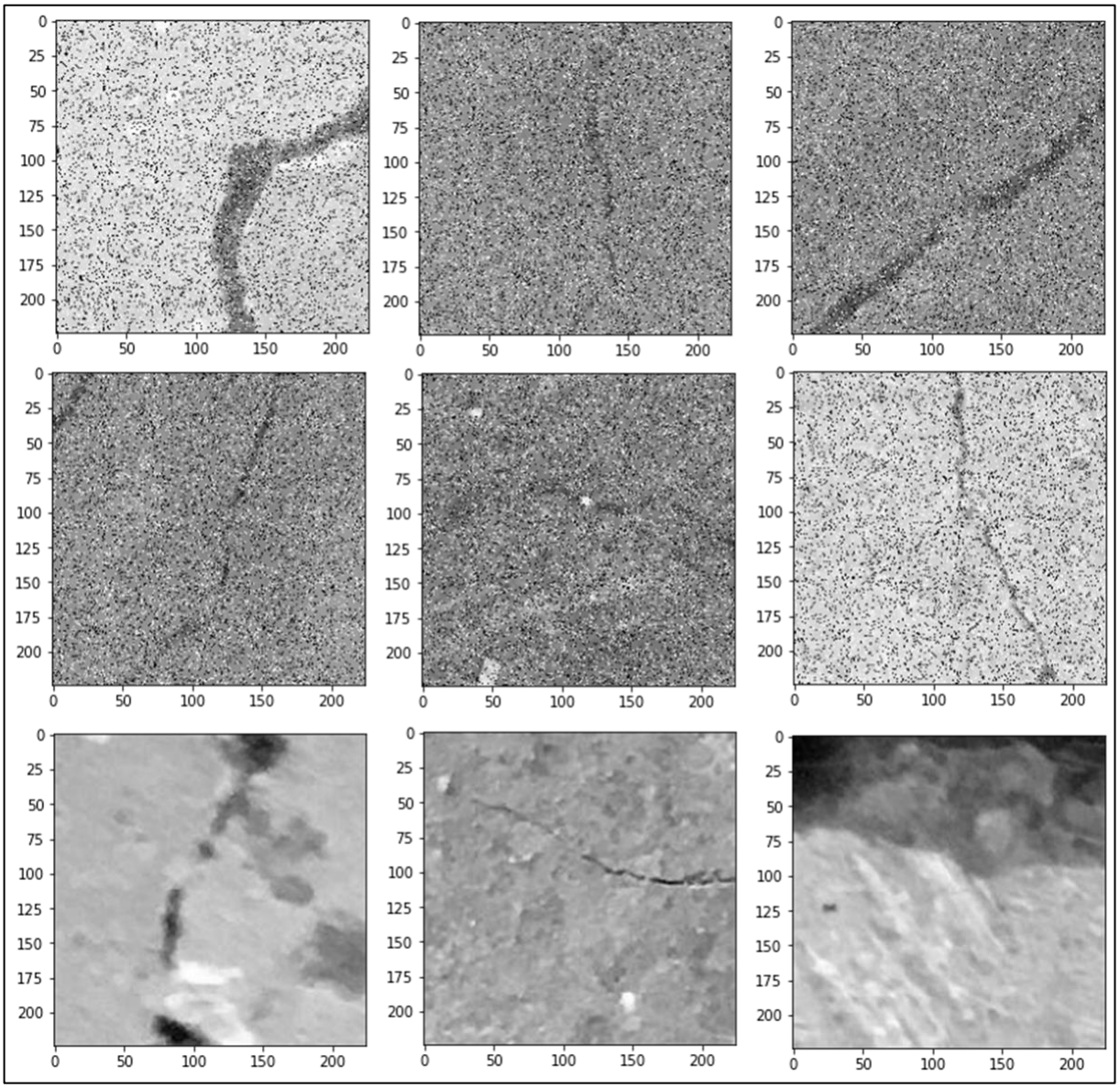


Figure 5.1: False negatives produced by model E2-9

Figure 5.1 shows a few samples out of the 26 false negatives produced by model E2-9. Based on the samples, it is observed that they have some similar characteristics. Which likely

contributed to the reason why the classifier failed to distinguish them and misclassified the images. The following outlines the probable reasons:

- The cracks are present at the side or corner of the images. Which is likely due to the inefficiency of the convolutional layers to extract features located around the perimeter of the image.
- The crack lines are very fine. Feature representation depends on the pixels, which the fine lines would indicate low concentration of pixels of the cracks. Thus, classifier may consider such pixels as noises.
- Too much noise that caused the covering up of most of the cracks. Too many noises introduced, causes the contrast between the crack and concrete surface to be undistinguishable.

In addition, out of the 26 number of misclassified false negative images, 16 images are augmented, and 10 images are unaugmented. This indicates that the classifier is more prone to misclassification when images are polluted with noises. Based on Figure 5.1, the images with noises are difficult to differentiate even with human verification. Therefore, it is encouraged to reevaluate the false negatives produced by the classifier to ensure all structural cracks are identified and mitigation can be performed.

5.2 Peer Comparison

This section compares the performance of the image classifier developed in this study against several literatures that have implemented some type of data augmentation into the training data. Table 5.2 shows the accuracies of several models from the literature against the model in this study with the name “verySimpleCNN”.

Table 5.2: Model performance in peer comparison

Reference	Architecture	Accuracy
Golding et al. (2022)	VGG16	99.55%
Palevičius et al. (2022)	AlexNet	99.41%
This study	**verySimpleCNN**	**99.22%**
Yang et al. (2021)	ResNet18	98.80%
Yang et al. (2021)	VGG13	98.30%
Yang et al. (2021)	AlexNet	97.60%

Based on Table 5.2, it is observed that the model developed in this study achieves a comparable result against the literatures. The verySimpleCNN was able to outperform three models from

the works of Yang et al. (2021). This can be attributed by the use of different datasets, where the author has taken photos of concrete cracks in the natural environment where noises are present in the original images. Thus, resulted in the difference in accuracies.

Golding et al. (2022) and Palevičius et al. (2022) have utilized the same dataset as this study, it is observed that their models were able to outperform the verySimpleCNN but by only a small margin. Comparing the architecture, it is observed that the deeper the architecture, the better the performance of the model. A deeper network architecture can result in better feature extraction which is likely attributed to the increase in model performance. However, a deeper architecture signifies more computational resources are required. With the high computational expense, the benefit of performance gain is marginal. Therefore, utilizing shallow network architecture was able to achieve comparable results against deeper network architecture. In addition, the marginal gain with the utilization of deeper network architecture may signify the inability of CNN to perfectly classify every image. This can be due to the images are too corrupted with noises or cracks present in only a small area of the image. However, these models can function as a complement to the structural inspection process to increase efficiency and productivity.

SECTION 6

CONCLUSION

This study aimed to develop a robust image classifier for concrete structural crack detections. Various studies have developed similar classifiers by utilizing lab environment generated images, which are generally clean and clear of noise. This study utilizes a data generator to introduce noise to existing images to increase the variation of data allowing the classifier to increase robustness and accuracies during model evaluation with noisy data.

The implementation of data augmentation is performed by introducing salt and pepper noises to the images. A random selection of 10% of the total pixels of each image will be augmented by changing the pixel value to either 0 or 255. This is to simulate the photos taken on a construction jobsite which can be contaminated with different types of noises. These augmented images are then used as training data for an image classifier that is developed using CNN. The findings showed that a classifier integrated with augmented data performed better than a classifier without the integration of augmented data in the training phase.

The CNN model developed in this study contains three layers of convolution followed by a global average pooling layer to perform the image classification task. The network was tuned based on the epochs and learning rate hyperparameter. It was identified that 30 epochs and 0.001 learning rate produced the optimal model. The model was evaluated and achieved a comparable result with an accuracy of 99.22%, a precision of 99.47%, and a recall of 98.96%. Based on the findings, it can be concluded that an image classifier for structural crack detections does not require a very deep network, a simple three layers of convolution is sufficient to perform adequate feature extraction to provide a high accuracy model.

In addition, it is identified that some images are more prone to misclassification. These images include cracks that are present at the corners or sides or the images, low contrast between the crack and concrete surface, and high level of noise interference to the point where the crack is not observable. However, these images contribute to a small minority of the total outputs. Therefore, this study has demonstrated that a classifier with the integration of data augmentation can provide beneficial performance gain to an image classifier. Which can be utilized as a

complement in real world application of structural crack detections to improve efficiency and accuracy during structural inspection.

With the rise of vision transformers, application of transformer algorithms can be explored in this field of study to improve the computation efficiency and potentially increase the accuracy of the classifier. In addition, if time and memory complexity is of concern, shallower CNN architecture can be explored to identify the optimal network by balancing the resource utilization and performance gain. Furthermore, a higher number of epochs can be explored to ensure model converges to the global minima.

REFERENCES

Aggarwal, V., & Kaur, G. (2018). A review:deep learning technique for image classification. *ACCENTS Transactions on Image Processing and Computer Vision, 4*(11), 21-25. doi:10.19101/TIPCV.2018.411003

Aslam, M., Khan, T. M., Naqvi, S. S., Holmes, G., & Naffa, R. (2019). On the Application of Automated Machine Vision for Leather Defect Inspection and Grading: A Survey. *IEEE Access, 7*, 176065-176086. doi:10.1109/ACCESS.2019.2957427

Atkin, G. (2020). Concrete Crack Image Detection. *Kaggle*. Retrieved from https://www.kaggle.com/code/gcdatkin/concrete-crack-image-detection

Bello, I., Fedus, W., Du, X., Cubuk, E. D., Srinivas, A., Lin, T.-Y., et al. (2021). Revisiting resnets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems, 34*, 22614-22627.

Cao, W., Liu, Q., & He, Z. (2020). Review of Pavement Defect Detection Methods. *IEEE Access, 8*, 14531-14544. doi:10.1109/ACCESS.2020.2966881

Cha, Y.-J., Choi, W., & Büyüköztürk, O. (2017). Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. *32*(5), 361-378. doi:https://doi.org/10.1111/mice.12263

Chen, K., Yadav, A., Khan, A., Meng, Y., & Zhu, K. (2019). Improved Crack Detection and Recognition Based on Convolutional Neural Network. *Modelling and Simulation in Engineering, 2019*, 8796743. doi:10.1155/2019/8796743

Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., & Sun, J. (2021). *Repvgg: Making vgg-style convnets great again.* Paper presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. Paper presented at the ICLR 2021.

El-Alfy, E.-S. M., & Binsaadoon, A. G. (2017). Silhouette-Based Gender Recognition in Smart Environments Using Fuzzy Local Binary Patterns and Support Vector Machines. *Procedia Computer Science, 109*, 164-171. doi:https://doi.org/10.1016/j.procs.2017.05.313

Fan, Z., Wu, Y., Lu, J., & Li, W. (2018). Automatic pavement crack detection based on structured prediction with the convolutional neural network. *arXiv preprint arXiv:.02208*.

Fu, C. C., fu, F. Q., & Rameswar, P. (2021). *Crossvit: Cross-attention multi-scale vision transformer for image classification.* Paper presented at the Proceedings of the IEEE/CVF international conference on computer vision. doi:https://doi.org/10.48550/arXiv.2103.14899

Gavali, P., & Banu, J. S. (2019). Chapter 6 - Deep Convolutional Neural Network for Image Classification on CUDA Platform. In A. K. Sangaiah (Ed.), *Deep Learning and Parallel Computing Environment for Bioengineering Systems* (pp. 99-122): Academic Press.

Gil, D., Lee, G., & Jeon, K. (2018). *Classification of Images from Construction Sites Using a Deep-Learning Algorithm*. Paper presented at the Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC), Berlin, Germany. doi:10.22260/ISARC2018/0024

Golding, V. P., Gharineiat, Z., Munawar, H. S., & Ullah, F. (2022). Crack Detection in Concrete Structures Using Deep Learning. *14*(13), 8117. doi:https://doi.org/10.3390/su14138117

Hao, L. Y., Chaoxia, W., Haoqi, F., Karttikeya, M., Bo, X., Jitendra, M., et al. (2022). MViTv2: Improved Multiscale Vision Transformers for Classification and Detection. *IEEE/CVF Conference on Computer Vision Pattern Recognition*, 4794-4804. doi:https://doi.org/10.1109/CVPR52688.2022.00476

He, Z. (2020). *Deep Learning in Image Classification: A Survey Report.* Paper presented at the 2020 2nd International Conference on Information Technology and Computer Application (ITCA). doi:10.1109/ITCA52113.2020.00043

Khan, A. I., & Al-Habsi, S. (2020). Machine Learning in Computer Vision. *Procedia Computer Science, 167,* 1444-1451. doi:https://doi.org/10.1016/j.procs.2020.03.355

Le, T.-T., Nguyen, V.-H., & Le, M. V. (2021). Development of Deep Learning Model for the Recognition of Cracks on Concrete Surfaces. *Applied Computational Intelligence and Soft Computing, 2021,* 8858545. doi:10.1155/2021/8858545

Li, J., Hassani, A., Walton, S., & Shi, H. (2021). ConvMLP: Hierarchical Convolutional MLPs for Vision. *CoRR*. doi:https://doi.org/10.48550/arXiv.2109.04454

Obaid, K. B., Zeebaree, S. R. M., & Ahmed, O. M. (2020). Deep Learning Models Based on Image Classification: A Review. *International Journal of Science and Business, 4*(11), 75-81. doi:10.5281/zenodo.4108433

Palevičius, P., Pal, M., Landauskas, M., Orinaitė, U., Timofejeva, I., & Ragulskis, M. (2022). Automatic Detection of Cracks on Concrete Surfaces in the Presence of Shadows. *22*(10), 3662.

Patel, K. (2019). Convolutional Neural Networks - A Beginner's Guide. *Towards Data Science.* Retrieved from https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022

Rani, R. S., & Devi, P. L. (2020). A literature survey on computer vision towards data science. *International journal of creative research thoughts, 8*(6). doi:https://ijcrt.org/papers/IJCRT2006458.pdf

Shah, S. (2022). Convolutional Neural Network: An Overview. *Analytics Vidhya.* Retrieved from https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/

Silva, W. R. L. d., & Lucena, D. S. d. (2018). *Concrete Cracks Detection Based on Deep Learning Image Classification.* Paper presented at the 18th International Conference on Experimental Mechanics (ICEM18), Brussels, Belgium.

Sorguç, A. G. (2018). *Performance Comparison of Pretrained Convolutional Neural Networks on Crack Detection in Buildings*. Paper presented at the Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC). doi:10.22260/ISARC2018/0094

Tan, M., & Le, Q. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. Paper presented at the Proceedings of the 36th International Conference on Machine Learning, Long Beach, California.

Tan, M., & Le, Q. V. (2021). *EfficientNetV2: Smaller Models and Faster Training.* Paper presented at the Proceedings of the 38th International Conference on Machine Learning. doi:10.48550/ARXIV.2104.00298

Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G., & Jégou, H. (2021). *Going deeper with image transformers.* Paper presented at the Proceedings of the IEEE/CVF International Conference on Computer Vision. doi:https://doi.org/10.48550/arXiv.2103.17239

Yahyaoui, H. (2021). 3D Convolutional Neural Network using PyTorch. *ReachIT Easily.* Retrieved from https://www.reachiteasily.com/2021/06/3d-convolutional-neural-network-pytorch.html

Yang, C., Chen, J., Li, Z., & Huang, Y. (2021). Structural Crack Detection and Recognition Based on Deep Learning. *Applied Sciences, 11*(6), 2868. doi:https://doi.org/10.3390/app11062868

Zebari, D. A., Haron, H., Zeebaree, S. R. M., & Zeebaree, D. Q. (2019). *Enhance the Mammogram Images for Both Segmentation and Feature Extraction Using Wavelet Transform.* Paper presented at the 2019 International Conference on Advanced Science and Engineering (ICOASE), Zakho - Duhok, Iraq. doi:10.1109/ICOASE.2019.8723779

Zeeshan, M., Adnan, S. M., Ahmad, W., & Khan, F. Z. (2021). Structural Crack Detection and Classification using Deep Convolutional Neural Network. *Pakistan Journal of Engineering and Technology, PakJET, 4*(4), 50-56.

Zhang, L., Yang, F., Zhang, Y. D., & Zhu, Y. J. (2016). *Road crack detection using deep convolutional neural network.* Paper presented at the 2016 IEEE International Conference on Image Processing (ICIP). doi:10.1109/ICIP.2016.7533052

APPENDIX A

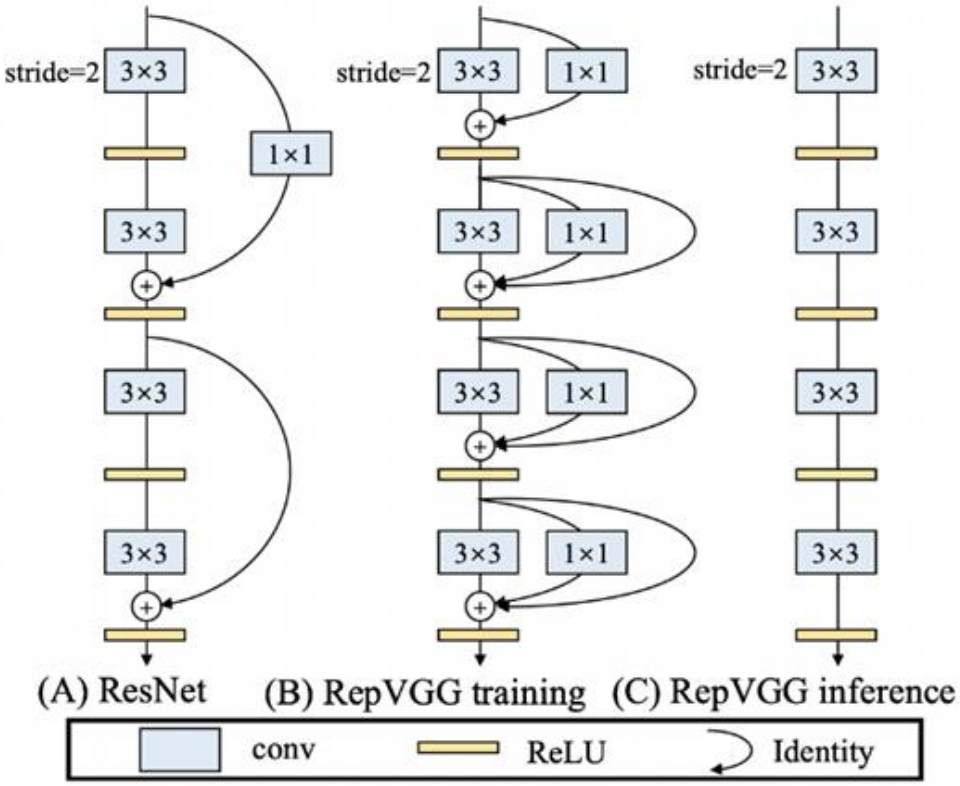
NEURAL NETWORK ARCHITECTURE

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure A.1: EfficientNetV1 architecture (Tan & Le, 2019)

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	272	15
7	Conv1x1 & Pooling & FC	-	1792	1

Figure A.2: EfficientNetV2 architecture (Tan & Le, 2021)



Sketch of RepVGG architecture. RepVGG has 5 stages and conducts down-sampling via stride-2 convolution at the beginning of a stage. Here we only show the first 4 layers of a specific stage. As inspired by ResNet [12], we also use identity and 1×1 branches, but only for training.

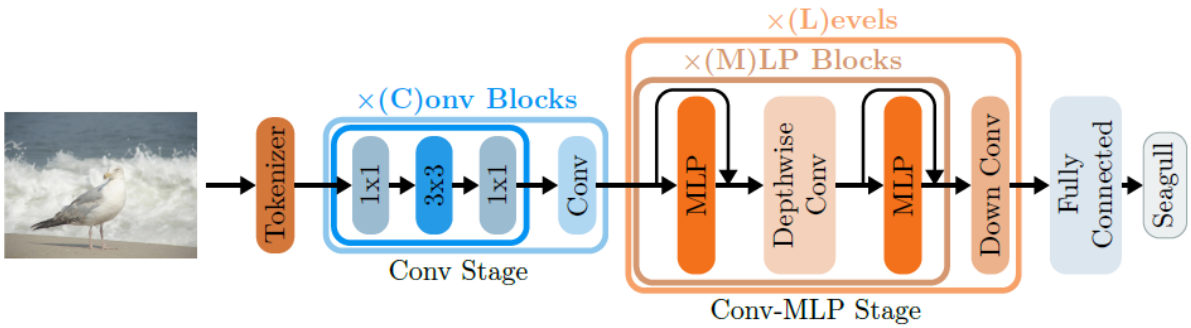
Architectural specification of RepVGG. Here $2 \times 64a$ means stage2 has 2 layers each with $64a$ channels.

Stage	Output size	RepVGG-A	RepVGG-B
1	112×112	$1 \times \min(64, 64a)$	$1 \times \min(64, 64a)$
2	56×56	$2 \times 64a$	$4 \times 64a$
3	28×28	$4 \times 128a$	$6 \times 128a$
4	14×14	$14 \times 256a$	$16 \times 256a$
5	7×7	$1 \times 512b$	$1 \times 512b$

Figure A.3: RepVGG architecture (Ding et al., 2021)

Block Group	Output Size	Convolution Layout
stem	112x112	[3x3, 64, s2 3x3, 64 3x3, 64] x1
c2	56x56	[1x1, 64 3x3, 64 1x1, 256] x3
c3	28x28	[1x1, 128 3x3, 128 1x1, 512] x4
c4	14x14	[1x1, 256 3x3, 256 1x1, 1024] x23
c5	7x7	[1x1, 512 3x3, 512 1x1, 2048] x3
	1x1	[Avg Pool Dropout 1000-d FC] x1

Figure A.4: ResNet-RS architecture (Bello et al., 2021)



Overview of ConvMLP framework. The Conv Stage consists of C convolutional blocks with 1×1 and 3×3 kernel sizes. The MLP-Conv Stage consists of Channelwise MLPs, with skip layers, and a depthwise convolution. This is repeated M times before a down convolution is utilized to express a level L. A level down samples an image $\mathcal{L} : h \times w \times c \mapsto \frac{h}{2^L} \times \frac{w}{2^L} \times 2^L c$

Figure A.5: ConvMLP framework (Li et al., 2021)

Stage	ConvMLP-S	ConvMLP-M	ConvMLP-L
Conv	$\begin{bmatrix} 1\times1 \text{ Conv} \\ 3\times3 \text{ Conv} \\ 1\times1 \text{ Conv} \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1 \text{ Conv} \\ 3\times3 \text{ Conv} \\ 1\times1 \text{ Conv} \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1 \text{ Conv} \\ 3\times3 \text{ Conv} \\ 1\times1 \text{ Conv} \end{bmatrix} \times 3$
Scale	$C_1 = 64$	$C_1 = 64$	$C_1 = 96$
Conv-MLP	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 3$	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 4$
Scale	$C_2 = 128, R = 2$	$C_2 = 128, R = 3$	$C_2 = 192, R = 3$
Conv-MLP	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 4$	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 6$	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 8$
Scale	$C_3 = 256, R = 2$	$C_3 = 256, R = 3$	$C_3 = 384, R = 3$
Conv-MLP	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 3$	$\begin{bmatrix} \text{Channel MLP} \\ 3\times3 \text{ DW Conv} \\ \text{Channel MLP} \end{bmatrix} \times 3$
Scale	$C_4 = 512, R = 2$	$C_4 = 512, R = 3$	$C_4 = 768, R = 3$

R denotes scaling ratio of hidden layers in MLP.

Figure A.6: ConvMLP architecture in different scales (Li et al., 2021)

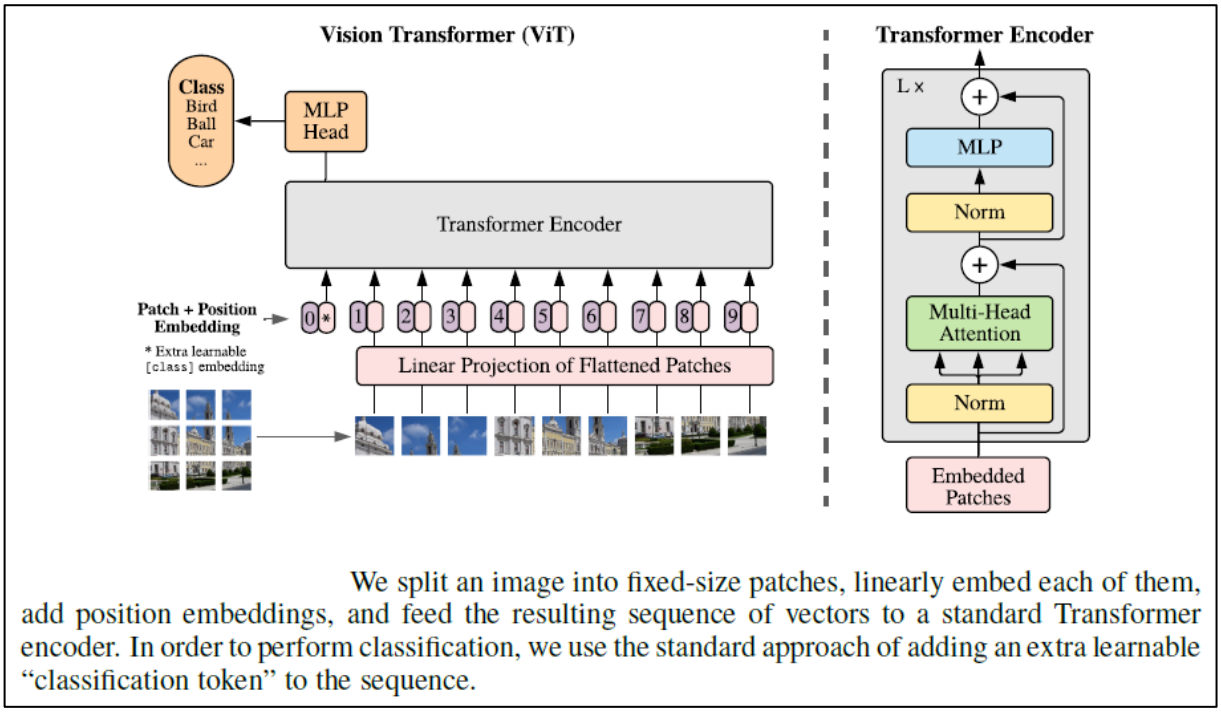
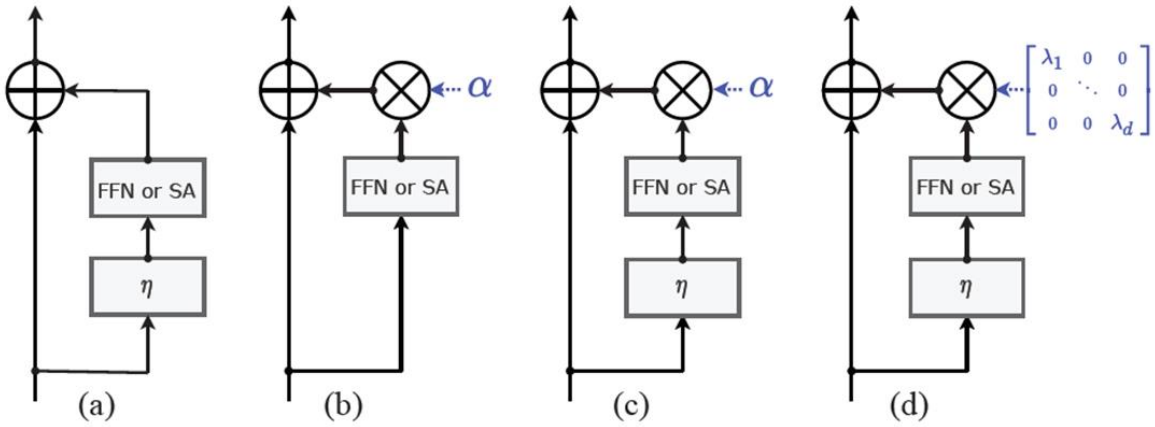


Figure A.7: ViT framework (Dosovitskiy et al., 2021)



Here we see four ways of adding the input residual to the block output. In **A** the residuals are just simply added to the output of the block, as they are in base ViT [2]. In **B** we see how residuals have been added in previous methods ReZero/Skipinit and Fixup. Here LayerNorm is taken away before the SA and FNN block and a learnable parameter is added to scale how much of the SA/FFN block output is added to the residual. The authors of this paper actually found that this method worked better if LayerNorm is added back in, which is **C**. Finally, in **D** we see LayerScale. A learnable diagonal weight matrix is dotted with the block output. As we see below, by taking the dot product of the diagonal weight matrix and block output, we allow for each channel (dimension d) of the block output to be scaled separately, thus giving the network more control over how much signal gets through from each channel. There are two diagonal weight matrices in each layer (one for the FFN block and another for the SA block), and they are initialized with values close to zero.

Figure A.8: Introduction of LayerScale in CaiT (Touvron et al., 2021)

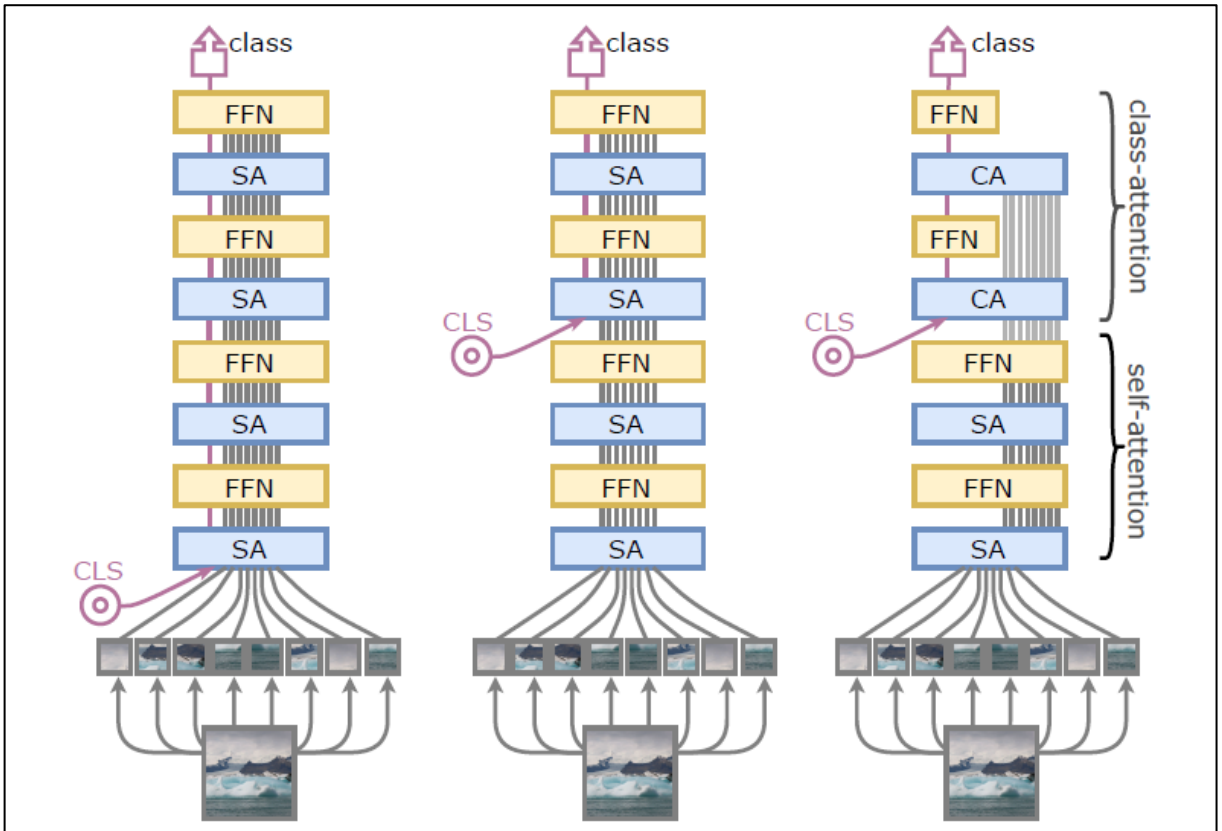
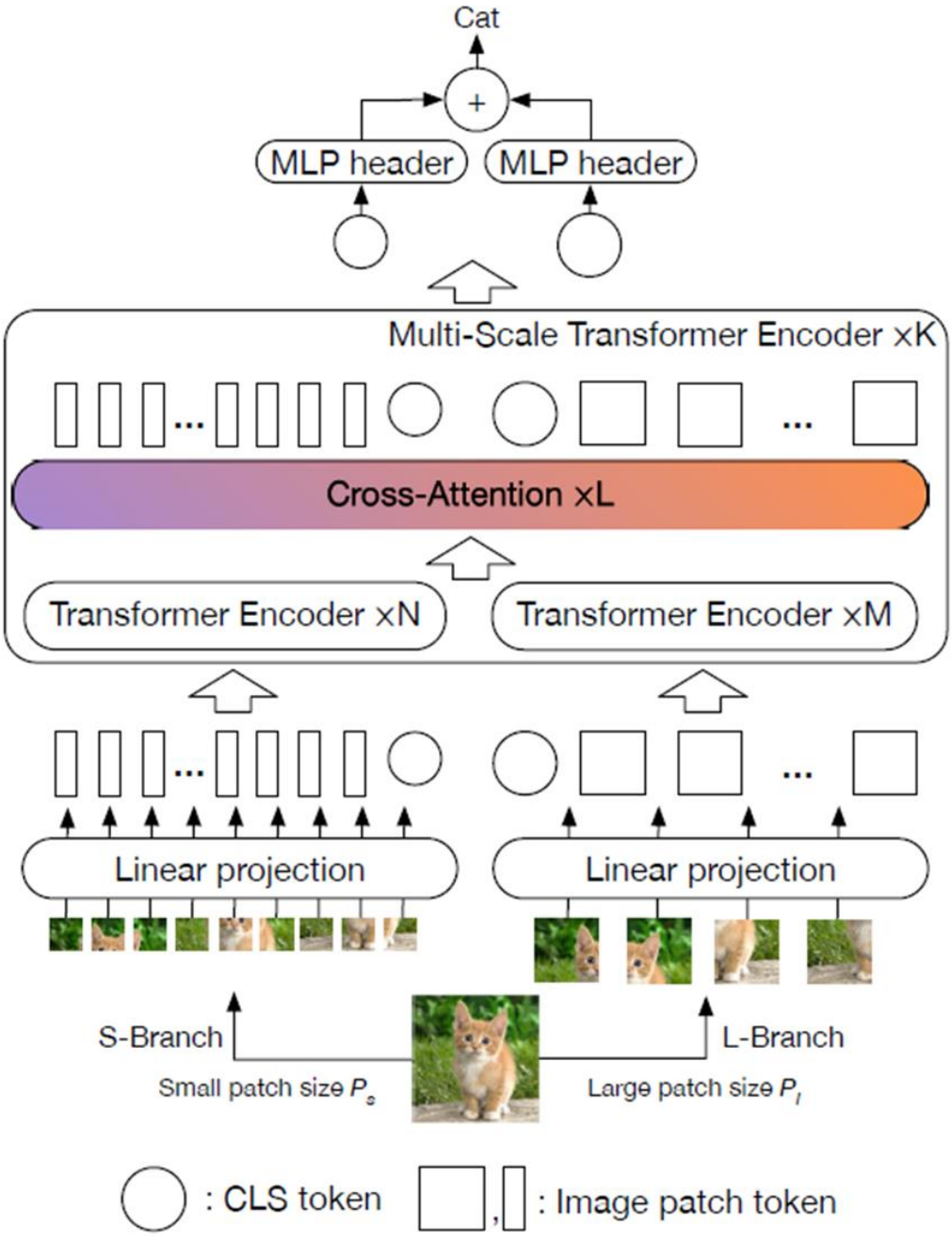


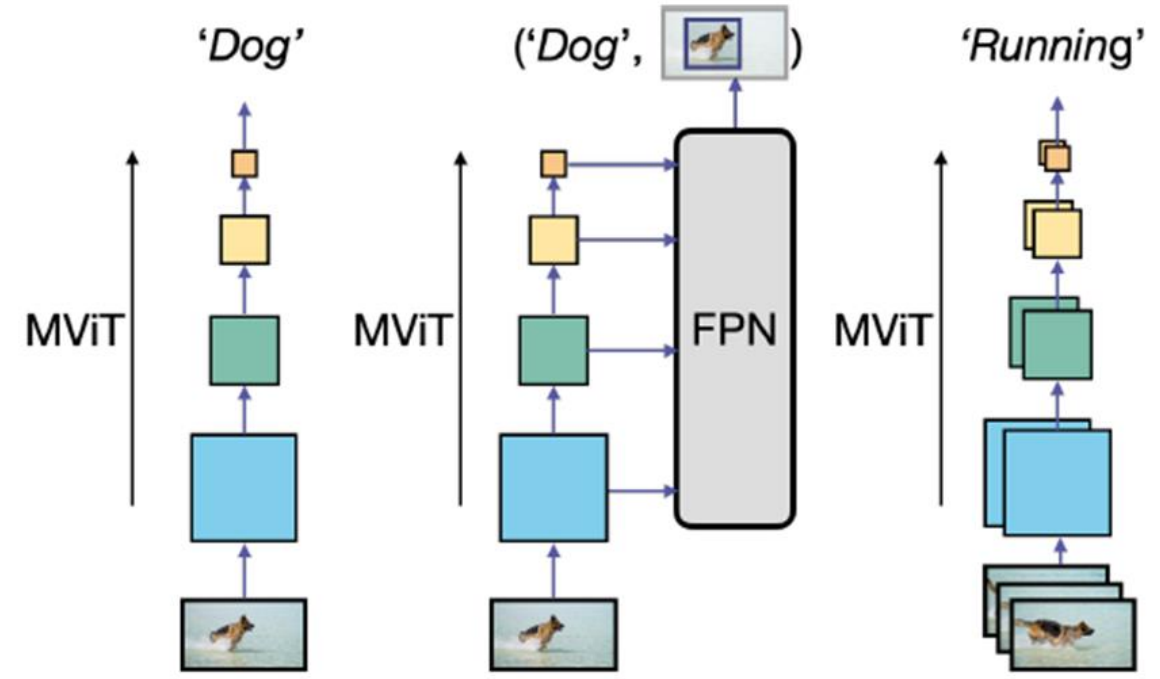
Figure 2: In the ViT transformer (*left*), the class embedding (CLS) is inserted along with the patch embeddings. This choice is detrimental, as the same weights are used for two different purposes: helping the attention process, and preparing the vector to be fed to the classifier. We put this problem in evidence by showing that inserting CLS later improves performance (*middle*). In the **CaiT** architecture (*right*), we further propose to freeze the patch embeddings when inserting CLS to save compute, so that the last part of the network (typically 2 layers) is fully devoted to summarizing the information to be fed to the linear classifier.

Figure A.9: Introduction of Class-Attention Layers in CaiT (Touvron et al., 2021)



Our architecture consists of a stack of K multi-scale transformer encoders. Each multi-scale transformer encoder uses two different branches to process image tokens of different sizes (P_s and P_l, $P_s < P_l$) and fuse the tokens at the end by an efficient module based on cross attention of the `CLS` tokens. Our design includes different numbers of regular transformer encoders in the two branches (i.e. N and M) to balance computational costs.

Figure A.10: CrossViT architecture (Fu et al., 2021)



Our **MViTv2** is a multiscale transformer with state-of-the-art performance across three visual recognition tasks.

Model	#Channels	#Blocks	#Heads	FLOPs	Param
MViT-T	[96-192-384-768]	[1-2-5-2]	[1-2-4-8]	4.7	24
MViT-S	[96-192-384-768]	[1-2-11-2]	[1-2-4-8]	7.0	35
MViT-B	[96-192-384-768]	[2-3-16-3]	[1-2-4-8]	10.2	52
MViT-L	[144-288-576-1152]	[2-6-36-4]	[2-4-8-16]	39.6	218
MViT-H	[192-384-768-1536]	[4-8-60-8]	[3-6-12-24]	120.6	667

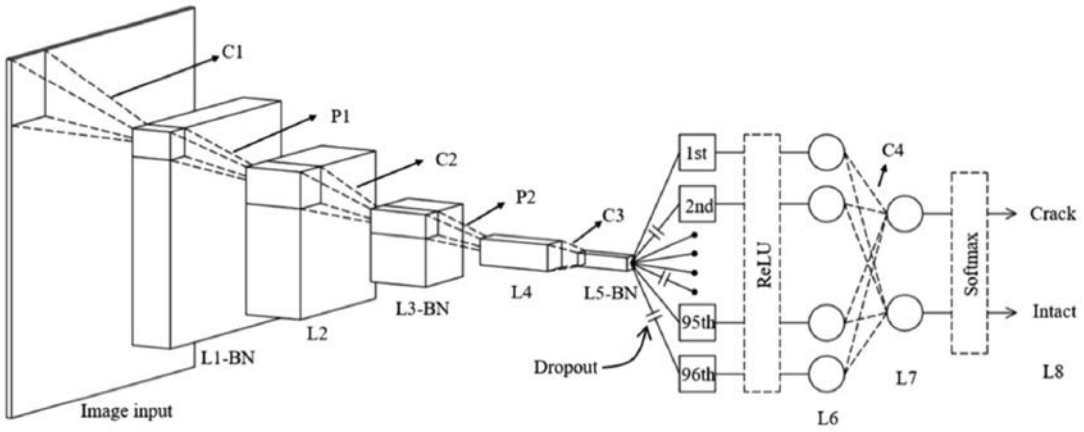
Configuration for MViT variants. *#Channels*, *#Blocks* and *#Heads* specify the channel width, number of MViT blocks and heads in each block for the four stages, respectively. FLOPs are measured for image classification with 224×224 input. The stage resolutions are $[56^2, 28^2, 14^2, 7^2]$.

Figure A.11: MViTv2 framework and architecture (Hao et al., 2022)

Architecture of AlexNet deep learning classification network.

Layer No.	Layer Name	Layer No.	Layer Name
1	Image input layer	14	Convolutional layer
2	Convolutional layer	15	ReLU layer
3	ReLU layer	16	Max-pooling layer
4	Cross-channel normalization layer	17	Fully connected layer
5	Max-pooling layer	18	ReLU layer
6	Convolutional layer	19	Dropout layer
7	ReLU layer	20	Fully connected layer
8	Cross-channel normalization layer	21	ReLU layer
9	Max-pooling layer	22	Dropout layer
10	Convolutional layer	23	Fully connected layer
11	ReLU layer	24	Softmax layer
12	Convolutional layer	25	Classification output layer
13	ReLU layer		

Figure A.12: AlexNet architecture used by Palevičius et al. (2022)



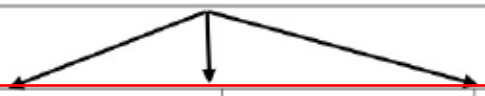
Overall architecture: L#: layers corresponding to operations (L1, L3, L5, and L7: convolution layers; L2 and L4: pooling layers; L6: ReLU layer; L8: softmax layer); C#: convolution; P#: pooling; BN: batch normalization.

Dimensions of layers and operations

Layer	*Height*	*Width*	*Depth*	*Operator*	*Height*	*Width*	*Depth*	*No.*	*Stride*
Input	256	256	3	C1	20	20	3	24	2
L1	119	119	24	P1	7	7	–	–	2
L2	57	57	24	C2	15	15	24	48	2
L3	22	22	48	P2	4	4	–	–	2
L4	10	10	48	C3	10	10	48	96	2
L5	1	1	96	ReLU	–	–	–	–	–
L6	1	1	96	C4	1	1	96	2	1
L7	1	1	2	Softmax	–	–	–	–	–
L8	1	1	2	–	–	–	–	–	–

Figure A.13: CNN architecture used by Cha et al. (2017)

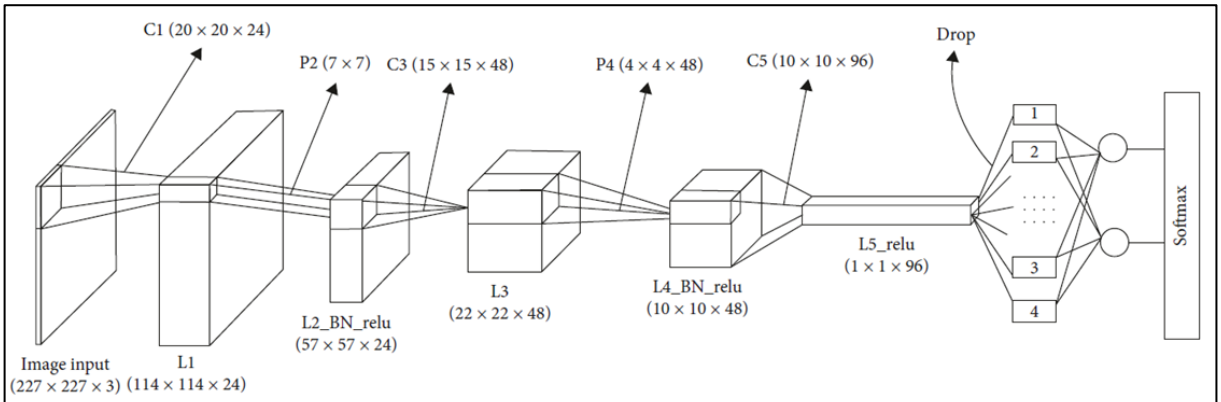
Layer (type)	Output Shape	Param #
input_6 (Input Layer)	[(None, 277, 277, 3)]	0
sequential_2 (Sequential)	(None, 227, 227, None)	0
normalization_2 (Normalization)	(None, 277, 277, 3)	7
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
Gglobal_max_pooling2d_2 (Global)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513
Total params: 14,715,208 Trainable params: 513 Non-trainable params: 14,714,695		



Layer(type)	Output Shape	Param#
input_1 (InputLayer)	[(None, 277, 277, 3)]	0
block1_conv1 ((Conv2D)	(None, 277, 277, 64)	1792
block1_conv2 ((Conv2D)	(None, 277, 277, 64)	36,928
block1_pool (MaxPooling2D)	(None, 113, 113, 63)	0
block2_conv1 (Conv2D)	(None, 113, 113, 128)	73,856
block2_conv2 (Conv2D)	(None, 113, 113, 128)	147,584
block2_pool MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
bock5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
Total params: 14, 714, 688 Trainable params: 14, 714, 688 Non-trainable params: 0		

RGB model used as displayed by Python; the VGG16 model layer has been expanded to show the CNN.

Figure A.14: VGG16 architecture used by Golding et al. (2022)



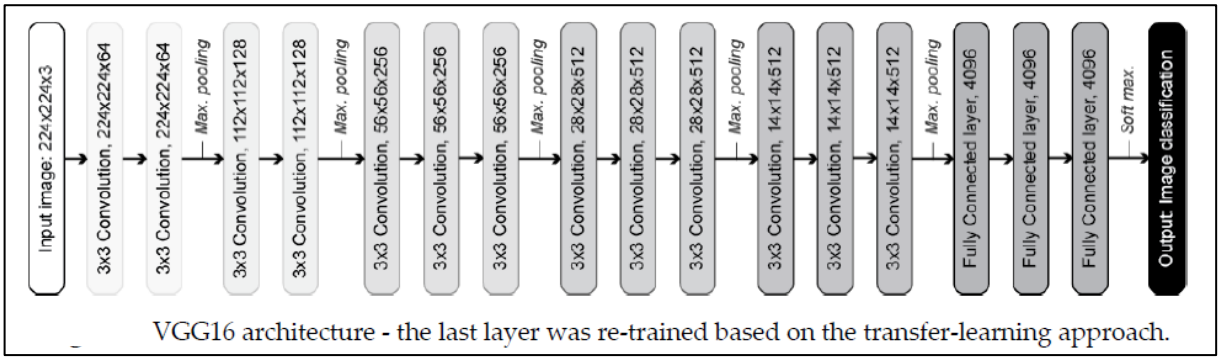
Layer	Size	Operator	Filters	Kernel_size	Strides	Padding
Input	227 × 227 × 3	C1	24	20 × 20	2	Same
L1	114 × 114 × 24	P2	—	7 × 7	2	Same
L2	57 × 57 × 24	BN	—	—	—	—
		relu	—	—	—	—
		C3	48	15 × 15	2	Valid
L3	22 × 22 × 48	P4	—	4 × 4	2	Valid
L4	10 × 10 × 48	BN	—	—	—	—
		relu	—	—	—	—
		C4	96	10 × 10	2	Valid
L5	1 × 1 × 96	relu	—	—	—	—
		Drop	—	—	—	—
L6	1 × 1 × 2	C5	2	1 × 1	1	Valid

The learning rate is set to 0.0001, and the learning rate drops to 0.99 for each iteration of 100; the dropout is set to 0.5.

Figure A.15: CNN architecture used by Chen et al. (2019)

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 120, 120, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 118, 118, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 59, 59, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 57, 57, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 28, 28, 32)        0
_________________________________________________________________
global_average_pooling2d (Gl (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
_________________________________________________________________
```

Figure A.16: CNN architecture used by Atkin (2020)



VGG16 architecture - the last layer was re-trained based on the transfer-learning approach.

Figure A.17: CNN architecture used by Silva and Lucena (2018)

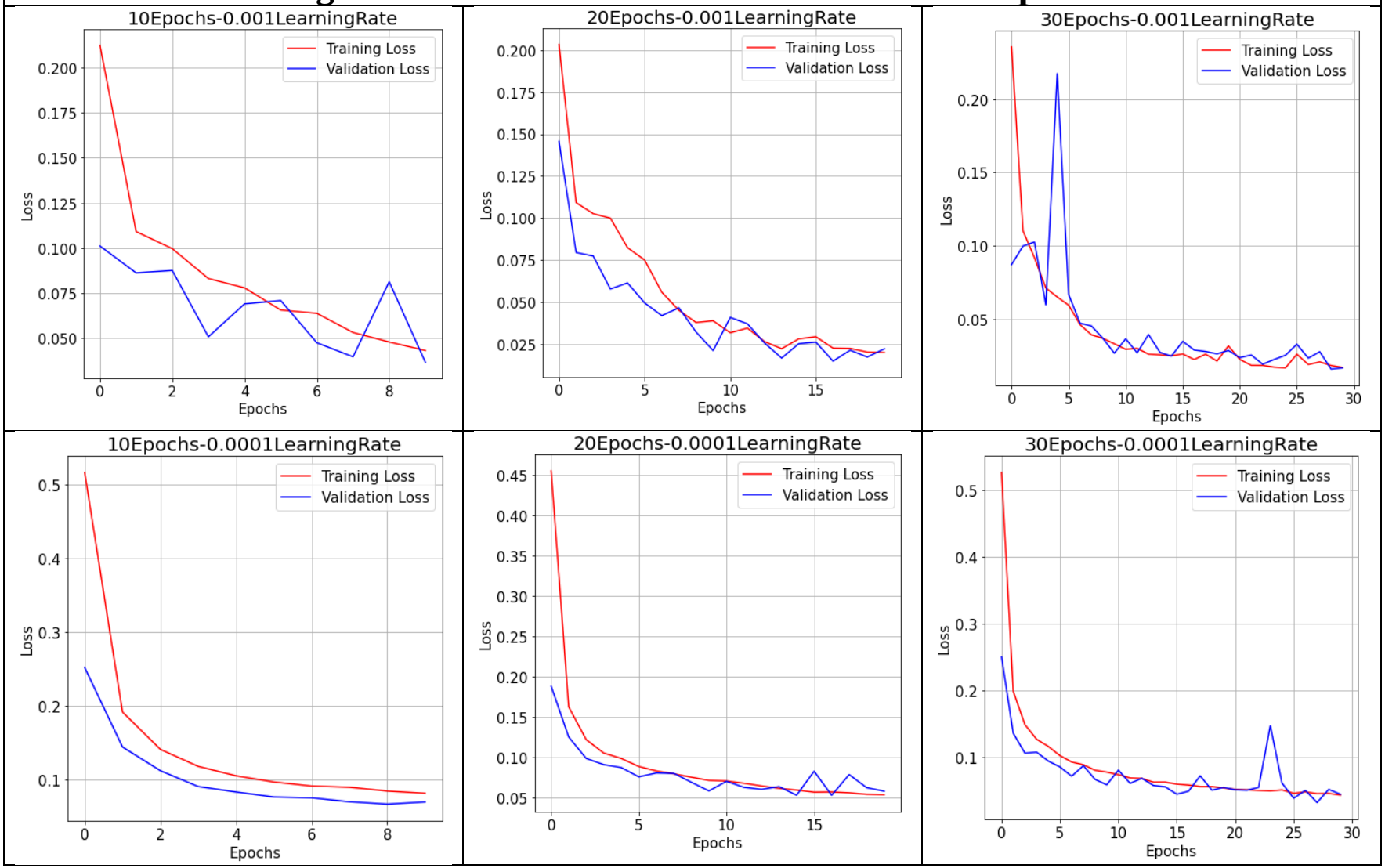
N°	Layer	Size of activation (i.e., outputs of layer)	Size of weight parameters	Size of bias parameters
1	Input layer	$227 \times 227 \times 3$	—	—
2	Convolutional layer	$227 \times 227 \times 16$	$3 \times 3 \times 3 \times 16$	$1 \times 1 \times 16$
3	ReLU layer 1	$227 \times 227 \times 16$	—	—
4	Fully connected layer 1	$1 \times 1 \times 200$	200×824464	200×1
5	Fully connected layer 2	$1 \times 1 \times 200$	200×200	200×1
6	Batch normalization layer	$1 \times 1 \times 200$	$1 \times 1 \times 200$	$1 \times 1 \times 200$
7	ReLU layer 2	$1 \times 1 \times 200$	—	—
8	Fully connected layer 3	$1 \times 1 \times 2$	2×200	2×1
9	Softmax layer	$1 \times 1 \times 2$	—	—
10	Classification output layer	—	—	—

Figure A.18: CNN architecture used by (Le et al., 2021)

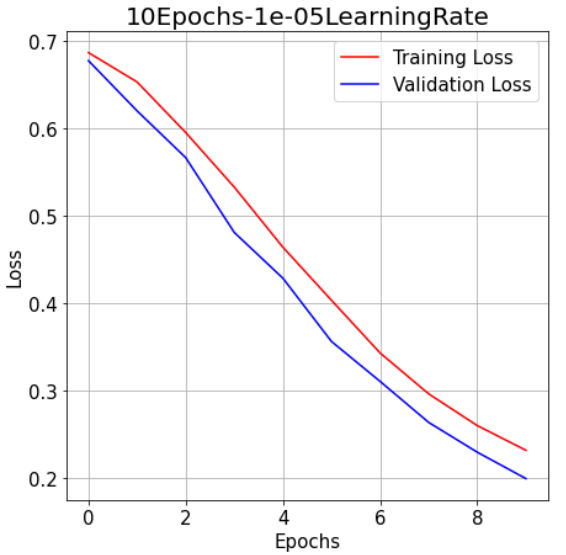
APPENDIX B

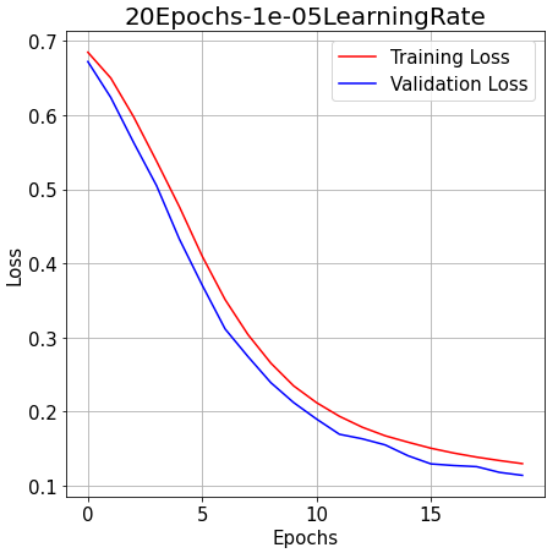
MODEL TRAINING LOSS PLOT

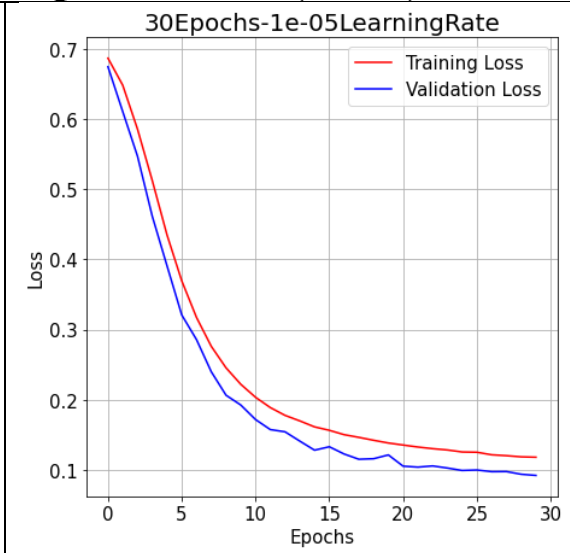
Training and Validation Loss Plot for Models in Experiment 1

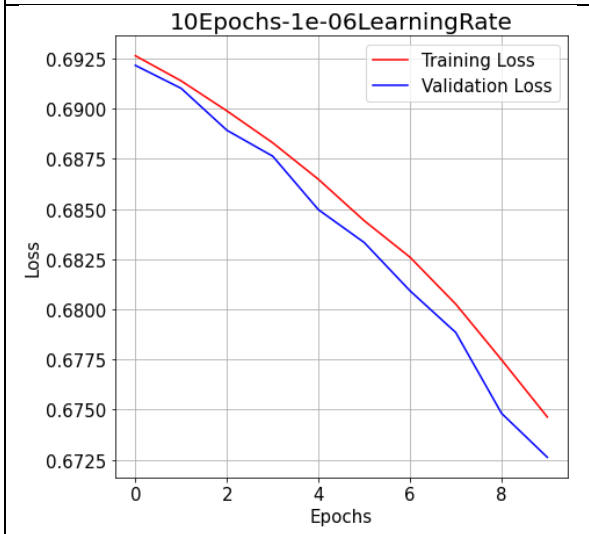


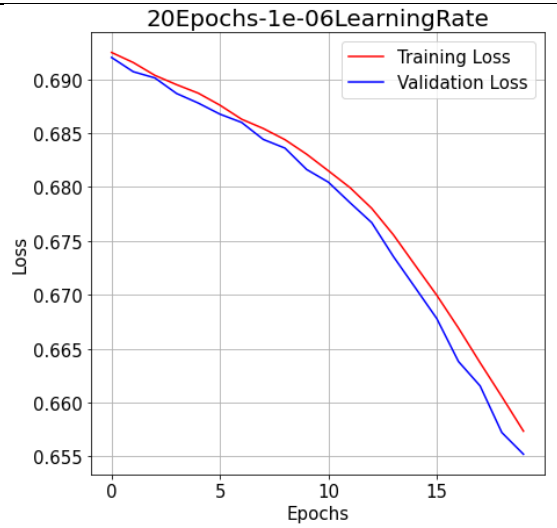
Training and Validation Loss Plot for Models in Experiment 1 (con't)

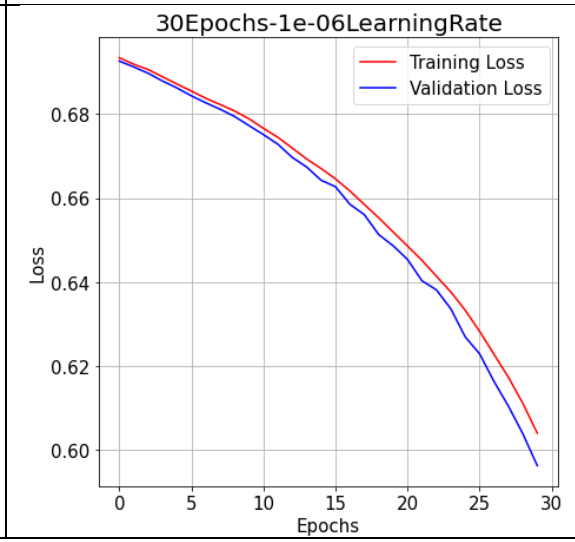




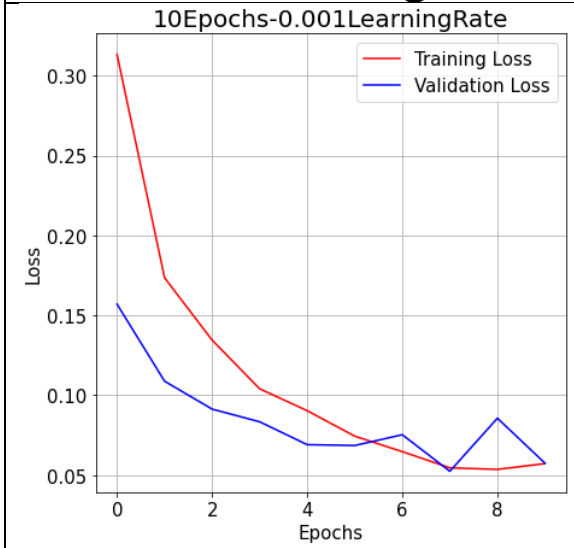


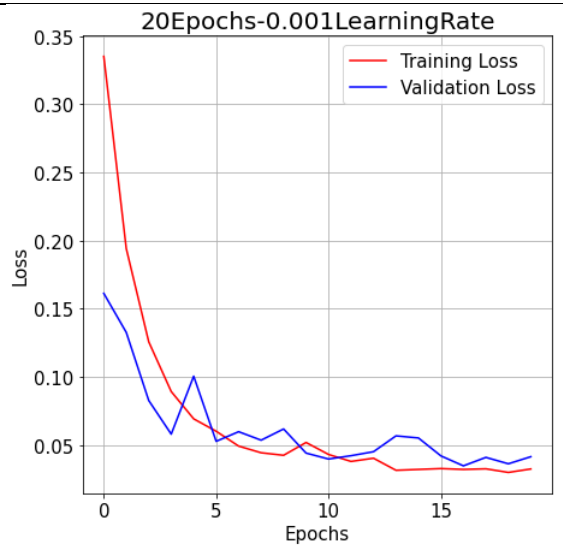


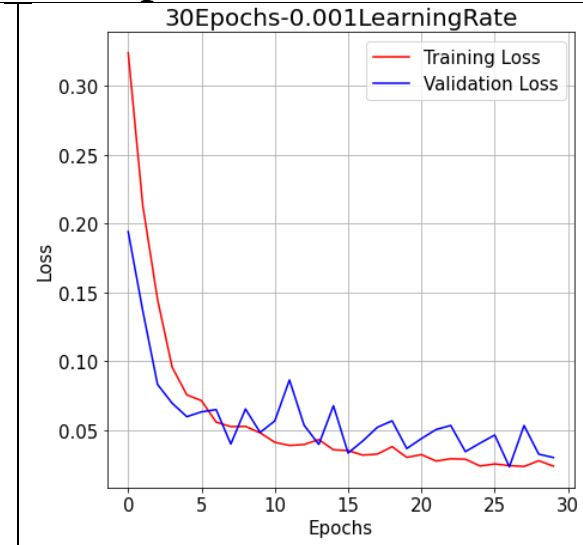


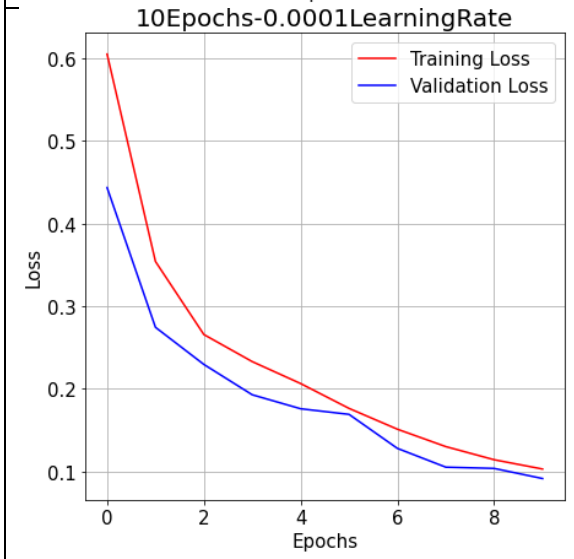


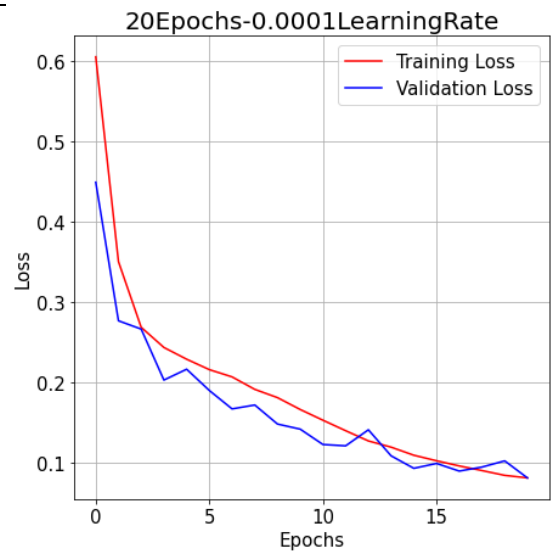
Training and Validation Loss Plot for Models in Experiment 2

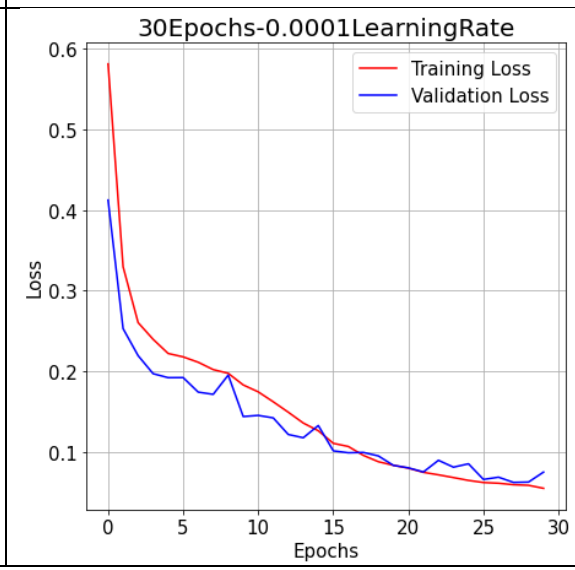












Training and Validation Loss Plot for Models in Experiment 2 (con't)