BT5153 Applied Machine Learning for Business Analytics
Group Project Proposal
Concrete Surface Crack Detection

Bai Tong	A0262700R
Le Meiyan	A0262707A
Luo Jianwei	A0262763Y
Zhang Ruixu	A0262828W
Zhang Xingyu	A0262692X

Abstract

This report presents an approach for concrete surface crack detection using Convolutional Neural Network (CNN) models. Four different CNN models including a baseline CNN, VGG16, ResNet50, and Inception v3 were explored. The results show that the Inception v3 model outperforms the other models and achieves the highest testing accuracy and lower testing loss in detecting concrete surface cracks. Further, the report investigates the importance of image features for concrete surface crack detection and the basis of prediction making by CNN models.

1. Background

The safety and well-being of building occupants and the public depend on the proper maintenance of structures. Structural inspections are required to guarantee that buildings continue to meet government standards. These inspections are conducted by a Professional Engineer (PE) who recommends repairs via a visual inspection report to the Building and Construction Authority (BCA) [1].

One of the key areas of concern in building inspections is the detection of cracks in concrete surfaces, as they are a major indicator of structural weakness. This project aims to automate the process of detecting cracks in concrete images captured by CCTV cameras, thus providing a more efficient and accurate assessment of a building's health. PEs will have less fieldwork to perform, the inspection process will be faster, and the results will easily integrate into the visual inspection report. Further, the BCA currently requires non-residential buildings to undergo inspections every 5 years, and residential buildings every 10 years. However, with the help of automation, the frequency of crack detection can be significantly increased to proactively identify and address potential safety hazards caused by structural defects.

2. Dataset introduction

The Surface Crack Detection dataset from Kaggle contains 40000 concrete surface images collected from various Middle East Technical University Campus Buildings [2]. These images are labelled either “positive” (with crack) or “negative” (without crack).

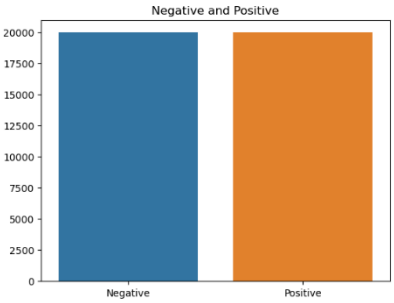

Figure 1: Balanced Dataset

The two classes each contains 20000 images.

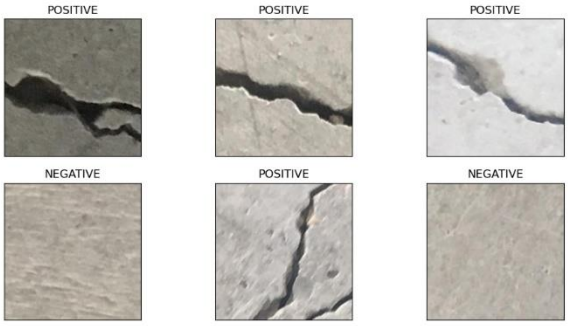

Figure 2: Image Samples

Github repository: https://github.com/AstridZh/BT5153-Group-Project

The images in the dataset all have 227×227 pixels and are coloured through RGB channels. They are transformed from high-resolution images that are sized 4032×3024 pixels using the technique proposed by Zhang et al in 2016 [3]. These high-resolution images have variance with regard to surface finish and illumination conditions. No data augmentation such as random rotation or flipping is applied.

3. Pre-processing

To analyse the input data, it is necessary to know the data type and storage method. The number of files exceeds 40,000 and is separated into 'Positive' and 'Negative' folders, making uploading the data to Google Colab unfeasible. Processing the data locally is also considered, but due to the requirements for deep modelling and the possibility of interruptions arising from differing laptop settings, this is not ideal. Therefore, the code is written on Kaggle, providing the benefit of downloading and writing the code directly inside the system. Additionally, Kaggle allows easy sharing among participants with accounts, facilitating the transfer of code within the group.

After setting up the system, paths are created for the two types of images separately. Then, a function is defined to create a dataframe that stores the file path for individual images and their corresponding class label.

	Filepath	Label
0	../input/surface-crack-detection/Positive/0574...	POSITIVE
1	../input/surface-crack-detection/Positive/1870...	POSITIVE
2	../input/surface-crack-detection/Positive/0967...	POSITIVE
3	../input/surface-crack-detection/Negative/0791...	NEGATIVE
4	../input/surface-crack-detection/Positive/1400...	POSITIVE
...	...	...
39995	../input/surface-crack-detection/Positive/0854...	POSITIVE
39996	../input/surface-crack-detection/Negative/1944...	NEGATIVE
39997	../input/surface-crack-detection/Positive/0977...	POSITIVE
39998	../input/surface-crack-detection/Positive/1504...	POSITIVE
39999	../input/surface-crack-detection/Negative/1099...	NEGATIVE

Figure 3: Filepath and Label Dataframe

Due to the limited computation power, 1000 samples were randomly selected to speed up modelling and testing. The next step is to perform train-test split. The training size is set to be 70% and the testing size to be 30%. ImageDataGenerator from the TensorFlow Keras library is used to generate data by producing batches of tensor image data with augmentations. In this case, the data is rescaled and further split on the training dataset, which is then divided into training and validation datasets. The flow_from_dataframe function is employed to train a classifier capable of classifying input images into classes.

```
Found 560 validated image filenames belonging to 2 classes.
Found 140 validated image filenames belonging to 2 classes.
Found 300 validated image filenames belonging to 2 classes.
```

Figure 4: Train-Test Split

4. Machine learning model

4.1 Baseline CNN

The Convolutional Neural Network (CNN or ConvNet) is a subtype of Neural Networks that is mainly used for applications in image and speech recognition. Its built-in convolutional layer reduces the high dimensionality of images without losing its information [4].

In Tensorflow, the Convolutional Neural Network can be built by defining the sequence of each layer. For the baseline model, a stack of Convolutional Layer and Max Pooling Layer is used twice. The input images have 120 height dimensions, 120 width dimensions, and 3 colour channels (red, green, and blue). The Convolutional Layer uses 32 and then 64 filters with a 3×3 kernel as a filter, and the Max Pooling Layer searches for the maximum value within a 2×2 matrix. A Global Average Pooling layer is added to average all the values according to the last axis and flatten the dimensions. Then, one more hidden layer with a total of 100 neurons is added before the model ends in the output layer with one neuron for binary classification. The completed model has a total of 8,489 parameters.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 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 (MaxPooling  (None, 28, 28, 32)       0
 2D)

 global_average_pooling2d (G  (None, 32)               0
 lobalAveragePooling2D)

 dense (Dense)               (None, 100)               3300

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 8,489
Trainable params: 8,489
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Baseline CNN Model, Summary

To ensure simplicity in training and comparability across different models, the following inputs are used to compile and fit all CNN models: learning rate = 0.001, weight decay

= 0.001, loss function = binary cross entropy, optimizer = Adam, number of epochs = 20, and batch size = 1.

After compiling and fitting the model to the training data, the progression of the model's loss and accuracy is plotted.

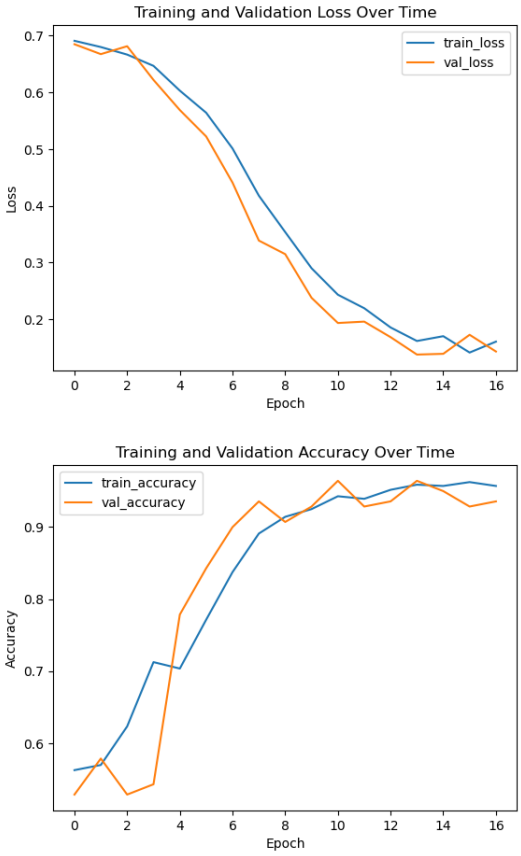

Figure 6: Baseline CNN Model, Training and Validation Loss and Accuracy Over Time

By the end of the 20 epochs, the learning curves have converged, indicating that there is no overfitting. However, since the training loss is still decreasing, underfitting is a potential issue.

Finally, the testing data is used to obtain the model's confusion matrix and classification report.

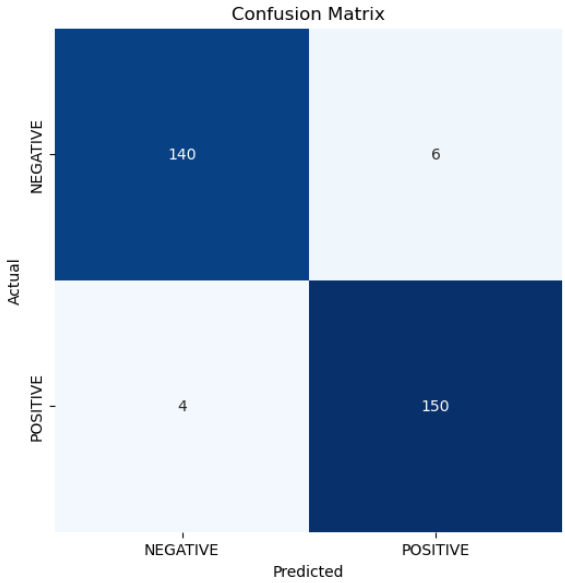

Figure 7: Baseline CNN Model, Confusion Matrix

```
Classification Report:
----------------------
              precision    recall  f1-score   support

    NEGATIVE       0.97      0.96      0.97       146
    POSITIVE       0.96      0.97      0.97       154

    accuracy                           0.97       300
   macro avg       0.97      0.97      0.97       300
weighted avg       0.97      0.97      0.97       300
```

Figure 8: Baseline CNN Model, Classification Report

In order to be considered superior in comparison to the baseline model, the pre-trained models need to produce a higher overall testing accuracy than the baseline model.

4.2 VGG-16

VGG-16 is a subtype of Convolutional Neural Networks (CNN) used for large-scale image recognition. It was proposed by K. Simonyan and A. Zisserman from Oxford University and published in a paper called "Very Deep Convolutional Networks for Large-Scale Image Recognition" [5]. The model has 16 layers with weights, consisting of 13 convolutional layers, 5 Max Pooling layers, and 3 Dense layers. The convolution layers have a 3x3 filter with stride 1 and always use the same padding and a max pooling layer of a 2x2 filter of stride 2. This makes VGG-16 one of the popular algorithms for image classification and easy to use with transfer learning [6].

The pre-trained VGG-16 model in TensorFlow Keras can be fine-tuned to classify concrete surface images. First, the model is instantiated with pre-loaded weights trained on ImageNet. The convolutional base is then frozen to prevent the weights in any layer from being updated during training.

Finally, a Pooling and a Dense layer are applied to the end of the model to convert these features into a single prediction per image [7].

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 3, 3, 512)         14714688

 global_average_pooling2d_1  (None, 512)               0
 (GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 1)                 513

=================================================================
Total params: 14,715,201
Trainable params: 513
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 9: VGG-16 Model, Summary

The model is compiled and fitted to the training data using the same inputs as mentioned earlier. The progression of the model's loss and accuracy is plotted, and the model is tested using the testing data.

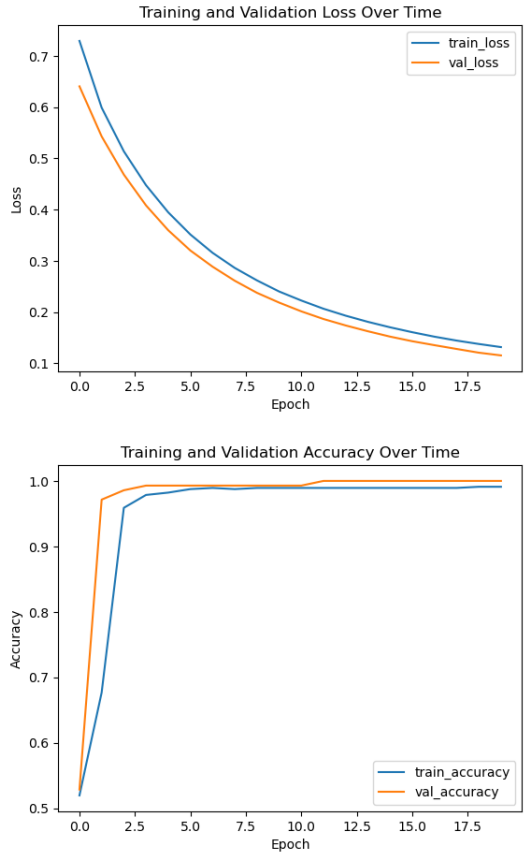

Figure 10: VGG-16 Model, Training and Validation Loss and Accuracy Over Time

The training and validation accuracy curves converge by around 4 epochs and remain flat after that, which in isolation suggests that the model is a good fit. However, the still-decreasing loss curves indicate that there may be an underfit and that the model can be further trained.

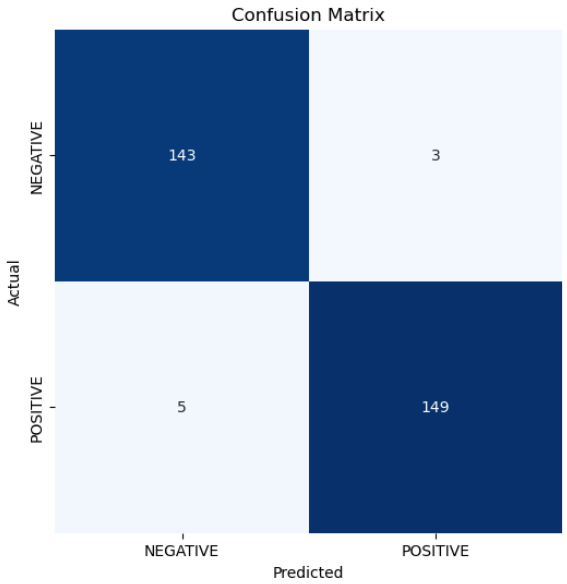

Figure 11: VGG-16 Model, Confusion Matrix

```
Classification Report:
----------------------
              precision    recall  f1-score   support

    NEGATIVE       0.97      0.98      0.97       146
    POSITIVE       0.98      0.97      0.97       154

    accuracy                           0.97       300
   macro avg       0.97      0.97      0.97       300
weighted avg       0.97      0.97      0.97       300
```

Figure 12: VGG-16 Model, Classification Report

The overall accuracy improves upon the baseline CNN model, and more importantly, the testing accuracy is even across positive and negative data.

4.3 ResNet50

ResNet50, short for Residual Network, is a CNN architecture introduced in 2015 by He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian in their paper "Deep Residual Learning for Image Recognition" [8]. ResNet50 consists of 50 layers, including 48 convolutional layers, one MaxPool layer, and one average pool layer. It can be applied to various computer vision tasks, such as image classification, object localization, and object detection, and its depth can also enhance non-computer vision tasks while reducing computational expenses [9].

To implement the pre-trained ResNet50 model, a similar approach to that of the VGG-16 model is followed. The ResNet50 model is initialized with pre-trained weights

from ImageNet, and a Pooling and a Dense layer are applied at the end to transform the features into a single prediction per image.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 4, 4, 2048)        23587712

 global_average_pooling2d_2  (None, 2048)              0
 (GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 1)                 2049

=================================================================
Total params: 23,589,761
Trainable params: 2,049
Non-trainable params: 23,587,712
_________________________________________________________________
```

Figure 13: ResNet50 Model, Summary

The model is compiled and fitted to the training data using the inputs mentioned earlier, and the loss and accuracy of the model's progress are plotted.

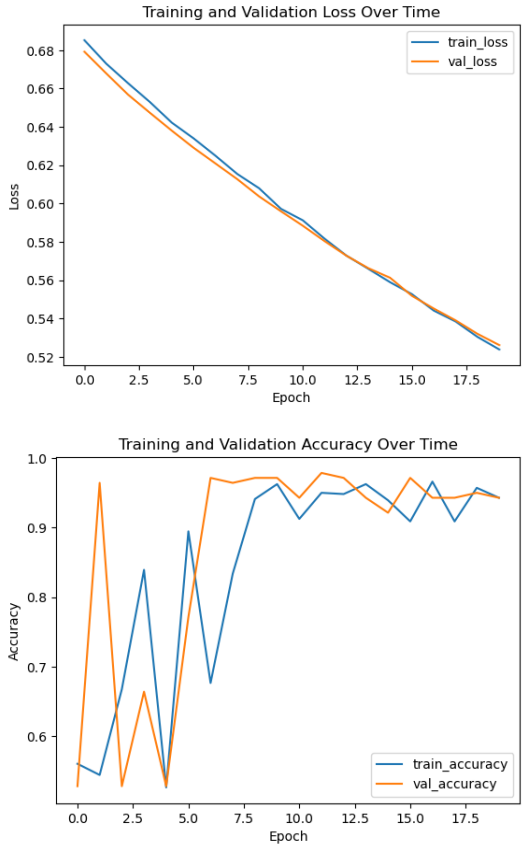

Figure 14: ResNet50 Model, Training and Validation Loss and Accuracy Over Time

The learning curves suggest that the model underfits, and the underfitting situation is much more apparent than the VGG-16 model. ResNet50 is therefore less optimal especially under limited computing power.

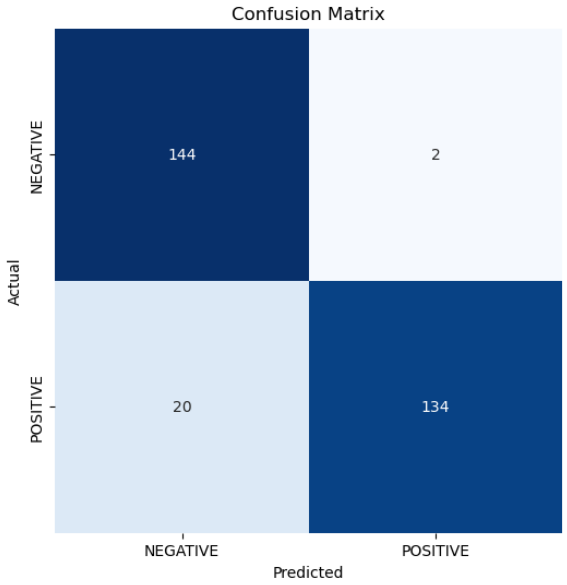

Figure 15: ResNet50 Model, Confusion Matrix

```
Classification Report:
----------------------
              precision    recall  f1-score   support

    NEGATIVE       0.88      0.99      0.93       146
    POSITIVE       0.99      0.87      0.92       154

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300
```

Figure 16: ResNet50 Model, Classification Report

Since the model underfits, it is expected that the overall testing accuracy is also lower. The model also predicts positive labels much better than it does negative labels. Within the context of concrete surface crack detection, this is a minor concern because misidentifying crack-free surfaces is not harmful. Nonetheless, the imbalanced prediction precision is still a drawback in performance.

4.4 Inception v3

Inception v3 is a CNN architecture designed to aid in image analysis and object detection, introduced by Szegedy et al. in their paper "Rethinking the Inception Architecture for Computer Vision" [10]. It is the third generation of Google's Inception Convolutional Neural Network, which was originally developed for the ImageNet Recognition Challenge. Inception v3 comprises 42 layers and aims to support deeper networks while maintaining a manageable number of parameters. It incorporates several modifications, including Label Smoothing, Factorized 7 x 7 convolutions, and an auxiliary classifier to propagate

label information down the network, as well as batch normalization for layers in the sidehead [11].

The implementation of the pre-trained Inception v3 model is similar to that of the previous models. The model is initialized with pre-trained weights from ImageNet, and a Pooling and a Dense layer are added at the end to convert the features into a single prediction per image.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 2, 2, 2048)        21802784

 global_average_pooling2d_3  (None, 2048)              0
 (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 1)                 2049

=================================================================
Total params: 21,804,833
Trainable params: 2,049
Non-trainable params: 21,802,784
_________________________________________________________________
```

Figure 17: Inception v3 Model, Summary

The model is then compiled and fitted to the training data using the inputs mentioned earlier.

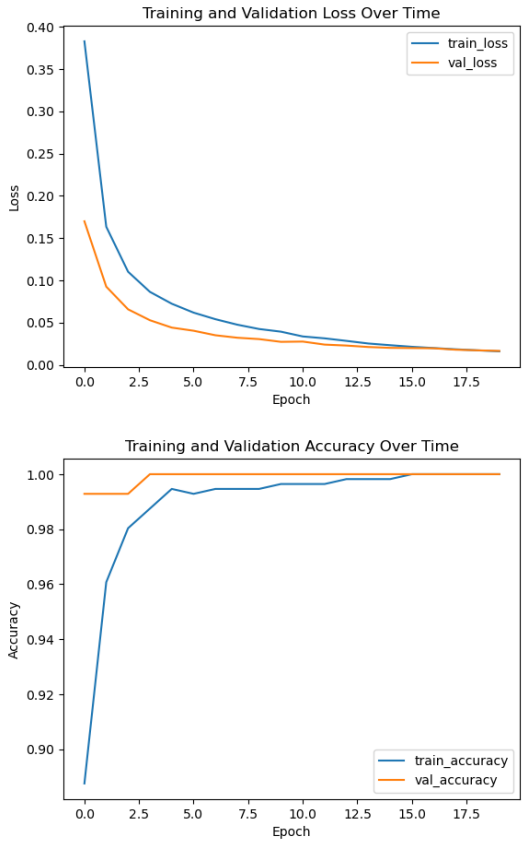

Figure 18: Inception v3 Model, Training and Validation Loss and Accuracy Over Time

By the end of the 20 epochs, the learning curves have converged and have a slope of almost 0. Therefore, there is neither an underfit nor overfit; the Inception v3 model is a good fit.

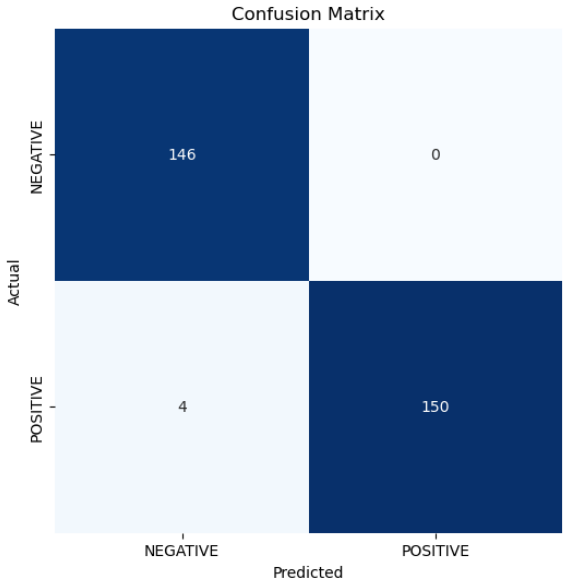

Figure 19: Inception v3 Model, Confusion Matrix

```
Classification Report:
----------------------
              precision    recall  f1-score   support

    NEGATIVE       0.97      1.00      0.99       146
    POSITIVE       1.00      0.97      0.99       154

    accuracy                           0.99       300
   macro avg       0.99      0.99      0.99       300
weighted avg       0.99      0.99      0.99       300
```

Figure 20: Inception v3 Model, Classification Report

The model also performs well on testing data. Although it predicts positive labels slightly better than negative labels, the difference is small and the accuracy on negative labels is still higher than the three other CNN models.

4.5 Model comparison

When summarizing the loss and accuracy of all four explored models, it can be observed from the table that the Inception v3 model exhibits the lowest testing loss and highest testing accuracy.

	loss	accuracy
baseline_cnn	0.148990	0.966667
vgg16	0.148374	0.973333
resnet50	0.525620	0.926667
inceptionv3	0.042461	0.986667

Figure 21: Model Loss and Accuracy Comparison

It can be concluded that the Inception v3 model performs the best out of the four models. Further hyperparameter tuning can be done on the Inception v3 model to improve its performance and/or reduce its runtime.

4.6 Hyperparameter tuning

As Inception v3 is a pre-trained model, the hyperparameters available for training are the inputs during the compiling and fitting stages. Specifically, the hyperparameters chosen for tuning include the learning rate, weight decay, number of epochs, and batch size.

To streamline hyperparameter tuning, the Optuna package is employed, and the objective is to minimize the validation loss. As a result, the optimal hyperparameters for the model are determined as learning rate = 0.01, weight decay = 0.002, number of epochs = 30, and batch size = 80, with the optimal validation loss being 2.608e-05.

Additionally, Optuna provides various visualization features to assist with the analysis of optimization results.

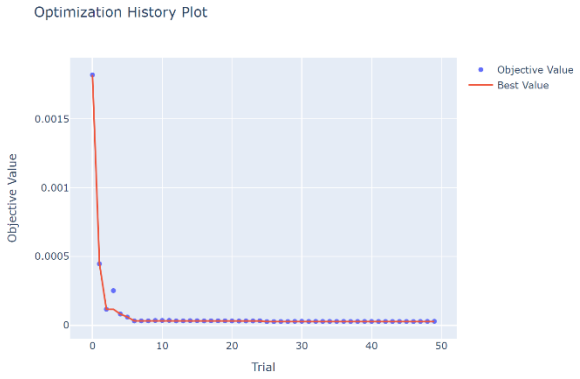

Figure 22: Optimization History

The optimization algorithm reached the best objective value by trial 25, suggesting that the algorithm's runtime can probably be improved by reducing the number of trials.

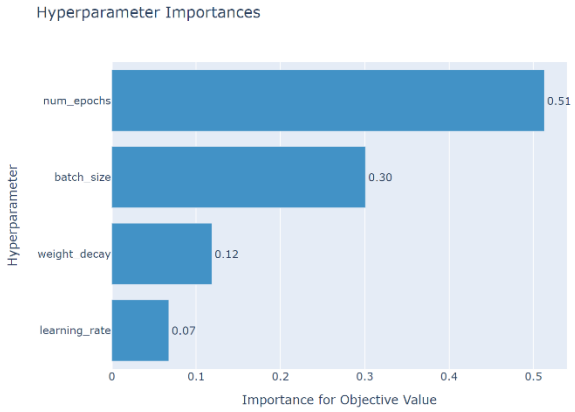

Figure 23: Parameter Importance

The hyperparameter importance plot reveals that the number of epochs has the greatest influence on validation loss, while the learning rate has the least impact.

The tuned Inception v3 model has a testing loss of 0.073 and a testing accuracy of 0.990.

	loss	accuracy
baseline_cnn	0.148990	0.966667
vgg16	0.148374	0.973333
resnet50	0.525620	0.926667
inceptionv3	0.042461	0.986667
tuned_inceptionv3	0.073157	0.990000

Figure 24: Model Loss and Accuracy Comparison

The tunned model simultaneously has higher testing accuracy and higher testing loss than the original model. A possible explanation for these two contradicting findings is that the cross-entropy loss function penalizes bad predictions much more strongly than it rewards good predictions. Thus, if there are images with very bad predictions that keep getting worse, the loss can increase while accuracy remains the same. Still, the improved testing accuracy is indicative of successful hyperparameter tuning.

5. Insights

In order to gain insights into the decision-making process of the model and ensure that the predictions are based on important features, this report utilizes LIME to provide explanations for specific predictions.

LIME, short for local interpretable model-agnostic explanations, is a method that approximates any black box machine learning model with a local, interpretable model to explain each individual prediction [12]. It helps to verify the model's correctness and to check for any biases in the training data that may influence the model's predictions.

In an image classification task, LIME generates explanations by masking out different parts of the image and observing how the classification score changes. By doing this repeatedly and fitting a simple model to the resulting scores, LIME is able to identify the parts of the image that are most important for the classification decision.

To illustrate, consider the sample image that shows a crack on a concrete surface, as shown in the figure below.

Figure 25: Image Sample from Positive Class

The portions of the image visible within the yellow boundary are where the model bases its predictions on.

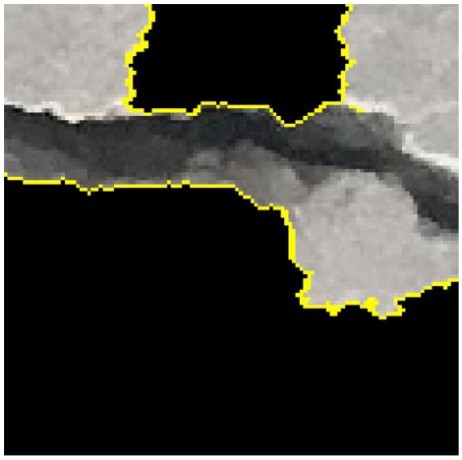

Figure 26: LIME Interpretation with Mask Boundary

The area coloured in green are the super-pixels that increase the probability of the image belonging to the positive class.

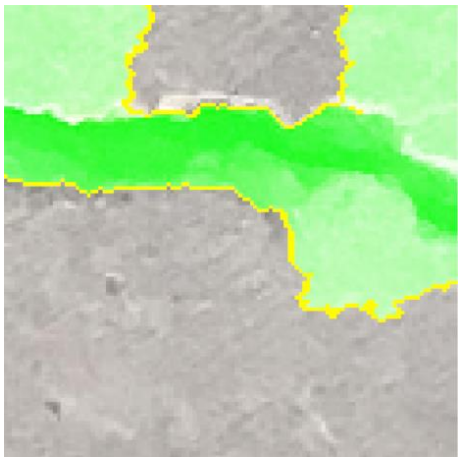

Figure 27: LIME Interpretation with Probability Colour

LIME confirms that the model makes decisions using the features of the crack itself, the dark edges of which creates a contrast against the lighter concrete surface background. There is no bias or mistake in the model's predictions.

6. Conclusion

Concrete surface cracks pose a significant threat to the safety of residents by indicating structural instability. However, traditional inspection methods require a considerable amount of human labor and cannot be conducted regularly.

This report employs four different CNN models - Baseline CNN, VGG-16, ResNet50, and Inception v3 - to identify cracks in photographs of concrete surfaces collected by CCTV. The models undergo training on the training dataset, validation on the validation dataset, and final evaluation on the testing dataset. The evaluation results reveal that the Inception v3 model has not only the lowest testing loss but also the highest testing accuracy.

After the model assessment, the Inception v3 model undergoes additional hyperparameter adjustment, utilizing the Optuna package. The optimum hyperparameters, including learning rate, weight decay, number of epochs, and batch size, are determined, minimizing the resulting validation loss. The tuned Inception v3 model achieves an accuracy of 0.99 on the testing dataset, indicating its robust prediction capability.

To gain further insight into the prediction-making process of the Inception v3 model, the LIME package is employed. The prediction example demonstrates that the model's logic aligns with human intuition, with the most important features or super pixels being those where the cracks are located.

Overall, this report demonstrates the effectiveness and potential of deep learning models in addressing real-world problems. Utilizing deep learning models such as Inception v3 can aid in ensuring the safety of residents by identifying potential structural instability in a more accessible and timely manner.

References

[1] *Periodic Structural Inspection (PSI)*. (2023, April 12). BCA Corp. https://www1.bca.gov.sg/regulatory-info/building-control/periodic-structural-inspection

[2] *Surface Crack Detection*. (n.d.). Surface Crack Detection | Kaggle. https:///datasets/arunrk7/surface-crack-detection

[3] L. Zhang, F. Yang, Y. Daniel Zhang and Y. J. Zhu, "Road crack detection using deep convolutional neural network," *2016 IEEE International Conference on Image Processing (ICIP)*, Phoenix, AZ, USA, 2016, pp. 3708-3712, doi: 10.1109/ICIP.2016.7533052.

[4] Lang, N. (2022, October 24). *Using Convolutional Neural Network for Image Classification*. Medium. https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4

[5] *Understanding VGG16: Concepts, Architecture, and Performance*. (n.d.). Datagen. https://datagen.tech/guides/computer-vision/vgg16/

[6] Learning, G. (2021, September 23). *Everything you need to know about VGG16*. Medium. https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[7] A. (2019, May 29). *How to use VGG model in TensorFlow Keras - Knowledge Transfer*. Knowledge Transfer. https://androidkt.com/how-to-use-vgg-model-in-tensorflow-keras/

[8] *ResNet-50: The Basics and a Quick Tutorial*. (n.d.). Datagen. https://datagen.tech/guides/computer-vision/resnet-50/

[9] *Understanding ResNet50 architecture*. (2020, March 30). OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/resnet50-architecture/

[10] *Papers with Code - Inception-v3 Explained*. (n.d.). Inception-v3 Explained | Papers With Code. https://paperswithcode.com/method/inception-v3

[11] *Inception v3*. (n.d.). Inception V3. https://huggingface.co/docs/timm/models/inception-v3

[12] *LIME: Local Interpretable Model-Agnostic Explanations*. (n.d.). C3 AI. https://c3.ai/glossary/data-science/lime-local-interpretable-model-agnostic-explanations/