**Chapter 5**

**Methodology**

This chapter consists of the work flow throughout the research. It includes data collection, building databank for training and validation set, developing deep CNN for training, classifying crack and non crack images.

**5.1 Data Source**

The datasets are collected [55]. The datasets include pictures of different textures of concrete with and without cracking. The image data is split into two in a separate folder for image classification as negative (without crack) and positive (with crack). With a total of 40000 images of 227 x 227 pixels with RGB channels, each class has 20000 images. The dataset is created from 458 (4032x3024 pixel) high-resolution images using the method proposed by Zhang et al (2016). High-resolution photographs have been shown to have high differences in terms of surface finish and state of lighting. No data augmentation is implemented in terms of random rotation or flipping or tilting.

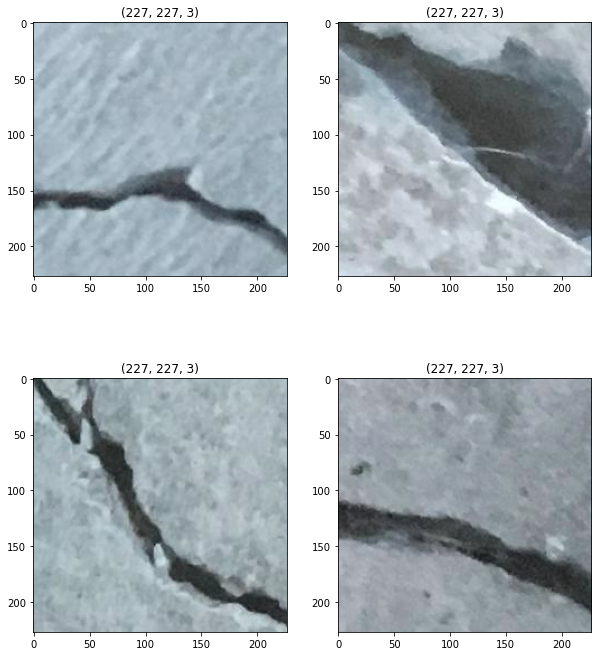
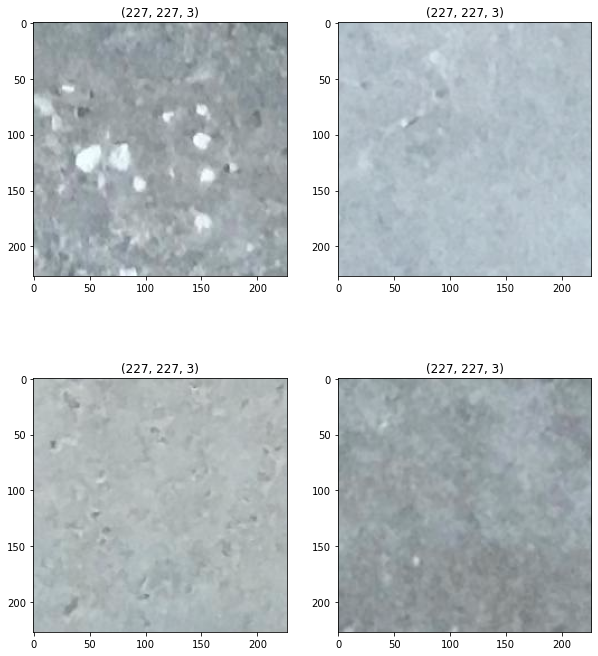


Fig 5.1: Crack Images[55]

Fig 5.2: Non-Crack Images[55]

All the images are in JPG Format. This a balanced dataset with equally distributed crack and non-crack images.

**5.2 Data Preprocessing**

I have spent a fair amount of time on strategies for data preprocessing typically used for image processing. This is because in most deep learning applications, preprocessing takes a large amount of time. As the dataset contains 40,000 images with RGB channel, I’ve performed two image processing techniques.

**5.2.2 Image Resize**

The first step in data pre-processing is the creation of images of the same size to feed our pre-trained CNN model. For this I’ve used keras built in function[56]. The images are re-sized into 150 x 150.

**5.2.3 Normalization**

Image normalization is a method often used in the preparation of machine learning data sets in which multiple images are positioned in a standard statistical distribution in terms of scale and pixel values; but within themselves, a single image may also be normalized. Typically, the phase entails both spatial and intensity normalization.

This is the most crucial step in image pre-processing. This corresponds to rescaling the values of the pixels so that they lie within a confined range. One of the reasons for doing so is to help with the challenge of gradient propagation. For normalization I’ve divided each pixel by 255 and re-scaled them between 0 to 1.

**5.3 Model Architecture**

For the classification I’ve used pre-trained CNN model VGG16 [57]. VGG is an acronym for the Oxford University Visual Geometric Group, and VGG-16 is a network suggested by the Visual Geometric Group with 16 layers. The trainable parameters are found in these 16 layers and there are other layers, such as the Max pool layer, but there are no trainable parameters. This architecture was the first heir to the 2014 Visual Perception Competition, i.e. ILSVRC-2014 which was founded by Zisserman and Simonyan. Figure 5.3 shows a VGG16 architecture.

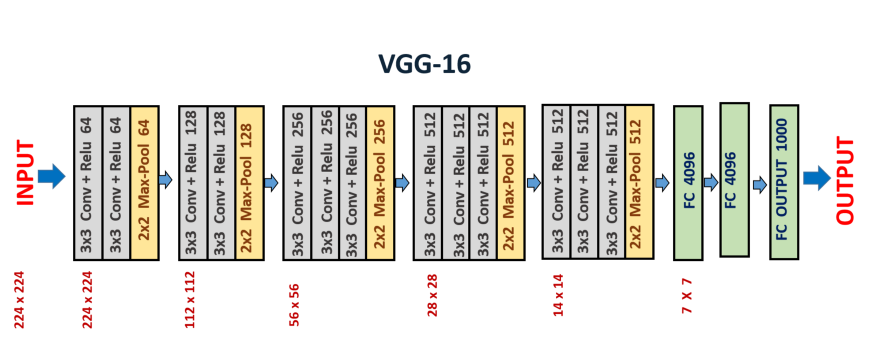
****

Fig 5.3: VGG16 Architecture[57]

The architecture is simple. It has 2 contiguous blocks of 2 convolution layers, followed by max-pooling, then it has 3 contiguous blocks of 3 convolution layers, followed by max-pooling, and we eventually have 3 thick layers. In different architectures, the last 3 layers of convolution have different depths.

**5.3.1 Features of VGG16 Network**

1. **Input Layer:** It accepts color images as an input with the size 224 x 224 and 3 channels i.e. Red, Green, and Blue.
2. **Convolution Layer:** The images pass through a stack of convolution layers where every convolution filter has a very small receptive field of 3 x 3 and stride of 1. Every convolution kernel uses row and column padding so that the size of input as well as the output feature maps remains the same or in other words, the resolution after the convolution is performed remains the same.
3. **Max pooling:** It is performed over a max-pool window of size 2 x 2 with stride equals to 2, which means here max pool windows are non-overlapping windows.
4. Not every convolution layer is followed by a max pool layer as at some places a convolution layer is following another convolution layer without the max-pool layer in between.
5. The first two fully connected layers have 4096 channels each and the third fully connected layer which is also the output layer have 1000 channels, one for each category of images in the imagenet database.
6. The hidden layers have ReLU as their activation function.

**5.3.2 Implementation of The Architecture**

The input to the conv1 layer is a fixed image size of 224 x 224 RGB. The picture is passed through a stack of convolutional (conv.) layers where the filters have been used with a very narrow receptive field: 3 x 3 (which is the smallest size to catch the left/right, up/down, center). It also uses 1 x 1 convolution filters in one of the setups, which can be used as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1; conv's spatial padding. The layer input is such that after convolution, the spatial resolution is retained, i.e. the padding for 3 x 3 conv is 1-pixel. Of textures. Five max-pooling layers, which obey some of the conv, perform spatial pooling. Of textures (not all the conv. layers are followed by max-pooling). Max-pooling, with stride 2, is done over a 2 to 2 pixel window.

A stack of convolutional layers (which has a different depth in different architectures) is preceded by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third has 1000-way ILSVRC classification and thus includes 1000 channels (one for each class). The last layer is soft-max layer. The configuration of fully linked layers on all networks is the same.

Both hidden layers are fitted with non-linearity for rectification (ReLU). Furthermore, it is remembered that none of the networks (except for one) contain Local Response Normalization (LRN), which does not boost the efficiency of the ILSVRC dataset, but results in increased memory usage and computation time.

**5.3.2 Training The Model**

As it is mentioned earlier about the proposed CNN model. For training the model the validation set was splitted from entire dataset. A validation dataset is a dataset of examples used for tuning a classifier's hyperparameters. Often it is also referred to as the “Development set” or the "dev set". The number of hidden units in each layer includes an example of a hyperparameter for machine learning.

For proposed method about 30 percent of the total dataset was used for validation. The validation set was selected randomly. So 12000 images were used for validation and rest 28000 images were used for training containing both crack and non crack images. After that the training and validation data were feed into VGG16 model. For VGG16 the weights are initialized with “imagenet” [58] a pretrained model weight initialization. The batch size is 64 and epoch is 20. VGG16 has 5 block with 13 convolutional layer. So the training and validation dataset are fast go through 5 blocks of VGG16. After that the output is flattened from 4 x 4 x 512 to 8192 x 1 which is a one dimensional vector. Three fully connected layers follow the stack of convolutional layers. The first two has 1024 units and final layer is ‘softmax’ layer with output node one.

Validation and testing were performed in training period. ‘ReLU’ function was used in each hidden layers. It is effective in computing and converges much faster than most other activation functions. “ReLU” converts the negative calculated value to 0 and no negative value passes to next layer. Hidden layers are the feature extracting layer. So using “ReLU” function in hidden layers doesn’t effect the output much because of the ignorance of the negative values. “Sigmoid” is used in final layer. **The sigmoid non-linearity has the mathematical form . It takes a real-valued number and “squashes” it into a range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. As the input dataset is binary class sigmoid function merges the output into crack or non crack images. 0 for non crack image and 1 for crack image.**

“Dropout” is used before the output layer to overcome the overfitting problem in the network [48]. Overfitting is' the development of a study that correlates too closely or precisely to a specific data set and may therefore struggle to match additional data or accurately predict future findings.' An overfit model is a mathematical model containing more parameters than the data can explain [59]. To overcome this problem dropout is used. Dropout is a method where randomly selected neurons are dropped during training. They are “dropped-out” arbitrarily. This infers that their contribution to the activation of downstream neurons is transiently evacuated on the forward pass and any weight refreshes are not applied to the neuron on the backward pass. The dropout is selected 20 percent of total nodes.

The network is trained all over using Adam optimizer with initial standard parameters [60]. During training the network weights may stack into local minima problem. Adam helps to find the global minima when the network stacks into local minima problem. Adam updates network weights iterative based in training data.

The model has two classes, crack and non crack. Regarding the problem “binary\_crossentropy” function was used to minimize the error [56]. The binary crossentropy is very convenient to train a model to solve many classification problems at the same time, if each classification can be reduced to a binary choice. The crack detection is a binary classification task. The output is a binary label ; representing the absence of crack or not respectively. In the training set, weighted binary cross entropy loss is optimized. A table of the summary of the whole network is given below:

Table 5.1 Model Summary

|  |  |  |
| --- | --- | --- |
| **Layer(Type)** | **Output Shape** | **No of Parameter** |
| **input** | (150, 150, 3) | 0 |
| **block1\_conv1** | (150, 150, 64) | 1792 |
| **block1\_conv2** | (150, 150, 64) | 36928 |
| **block1\_pool** | (75, 75, 64) | 0 |
| **block2\_conv1** | (75, 75, 128) | 73856 |
| **block2\_conv2** | (75, 75, 128) | 147584 |
| **block2\_pool** | (37, 37, 128) | 0 |
| **block3\_conv1** | (37, 37, 256) | 295168 |
| **block3\_conv2** | (37, 37, 256) | 590080 |
| **block3\_conv3** | (37, 37, 256) | 590080 |
| **block3\_pool** | (18, 18, 256) | 0 |
| **block4\_conv1** | (18, 18, 512) | 1180160 |
| **block4\_conv2** | (18, 18, 512) | 2359808 |
| **block4\_conv3** | (18, 18, 512) | 2359808 |
| **block4\_pool** | (9, 9, 512) | 0 |
| **block5\_conv1** | (9, 9, 512) | 2359808 |
| **block5\_conv2** | (9, 9, 512) | 2359808 |
| **block5\_conv3** | (9, 9, 512) | 2359808 |
| **block5\_pool** | (4, 4, 512) | 0 |
| **flatten** | (None, 8192) | 0 |
| **Dense** | (None, 1024) | 8389632 |
| **Dropout** | (None, 1024) | 0 |
| **Dense** | (None, 1) | 1025 |

**5.4 Conclusion**

The chapter is about the methodology of building the architecture. VGG16 is a pretrained architecture with total number of five blocks containing thirteen convolutional layer distributed in each block. Each block has a maxpooling layer following the convolutional layers.

**CHAPTER 6**

**Results and Performance Analysis**

**6.1 Environment**

The VGG16 model was built in kaggle notebook [57]. The notebook editing session allows 9 hours execution time. It has 20 Gigabytes of auto saved disk including 4 CPU cores, 16 Gigabytes of RAM. And the GPU specs with 2 CPU cores and 13 Gigabytes of RAM. It provides NVIDIA Tesla P100. Keras API on top of TensorFlow (CUDA toolkit 9.0, cuDNN SDK v7 and python 3.6) were used [58].

**6.2 Experimental Analysis**

The data set is divided into training and validation set randomly. 12000 images are selected for validation and the rest for training. Testing was performed during training period. The results presented in this work is based on accuracy and f1 score [59] which are described by the following equations:

(20)

(21) 

Where tp, tn, fp, fn represent true positive, true negative, false positive and false negative respectively. Recall is defined as the fraction of the relevant instances in a dataset that is successfully retrieved and precision expresses the proportion of the data points the model says is relevant actually are relevant.

CNN has the advantage of learning features automatically instead of manual feature extraction techniques. The self-learning ability of CNN model makes it more convenient than the traditional learning system.

Figure 6.1 represents the training and validation accuracy and figure 6.2 shows the training and validation loss.

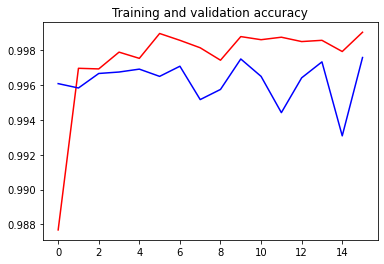


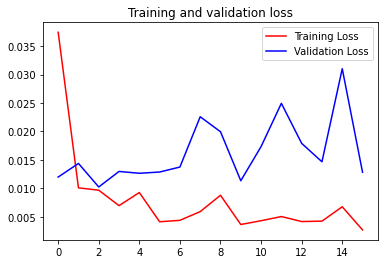
Figure 6.1: Accuracy curve for training and validation 

Figure 6.1: Accuracy curve for training and validation loss

From the figure we can see, by the increasing number of epochs the proposed model gets more accuracy with a decreasing factor of loss. It took around 233s for first epoch and gradually the time was minimized. Using equations (20) and (21), the proposed system achieved 99.90 percent accuracy, 0.990 f1 score and 0.9992 precision which outperformed the previous state of the art. A recent research [30] had the accuracy of 97.5 percent accuracy using ResNet based classifier. Though VGG16 got 99.8 percent accuracy but needed 50 epoch to acquire this accuracy which was very time consuming. Proposed method got almost same accuracy performing only 20 epoch which was very much time friendly. Performance of different of CNN architecture is shown in the table below comparing the proposed model also:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Accuracy** | **Recall** | **Precision** | **F1 Score** |
| VGG16 | 99.8% | 0.999 | 0.998 | 0.998 |
| Inception V3 | 99.7% | 0.997 | 0.998 | 0.997 |
| ResNet | 97.5% | 0.945 | 0.994 | 0.968 |
| Proposed VGG16 | 99.9% | 0.9988 | 0.9992 | 0.9990 |

From the table we can see proposed method has outperformed in all the field including accuracy, precision, recall and f1 score.

The accuracy of the network highly depends on the depth of CNN architecture [61]. VGG16 is a deep convolutional neural network with 13 convolutional layer and three fully connected layer. The hidden layers of CNN extract the features. That’s why VGG16 performs better for feature extracting methods.

“ReLU” is the activation function of VGG16 in hidden layers. “ReLU” is computationally efficient and converges much faster than most other activation functions [14]. It computes the function  **and the activation being threshold at zero. Compared to ‘Sigmoid’ function which has exponential operations ‘ReLU’ can be implemented by simply thresholding a matrix of activation at zero.**

**The kernels are initialized with ‘imagenet’ initializer [58]. This is pretrained weight for VGG16. Three ‘fully connected’ layers were used in the network.** The network was trained with Adam optimizer with initial standard parameters. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. Previous methods used ‘RMSProp’, ‘AdaGrad’[30] etc optimization tools. But among them Adam showed the best optimization in proposed method.

**6.3 Conclusion**

The proposed method achieved a quite good accuracy detecting crack images. This methodology can contribute a lot in detecting cracks in civil engineering sectors.

[56] <https://gombru.github.io/2018/05/23/cross_entropy_loss/>

[57] https://www.kaggle.com/docs/notebooks

[58] Chollet, François. "Keras." (2015): 128.

[59] Goutte, Cyril, and Eric Gaussier. "A probabilistic interpretation of precision, recall and F-score, with implication for evaluation." European Conference on Information Retrieval. Springer, Berlin, Heidelberg, 2005.

[61] [Diederik Kingma](http://dpkingma.com/) and [Jimmy Ba](https://jimmylba.github.io/) from the University of Toronto in their 2015 [ICLR](http://www.iclr.cc/doku.php?id=iclr2015:main) paper (poster) title.