# A Deep Learning Approach for Flood Detection from Social Media Imagery

Vani Bhat

Department of Applied Data Science, San Jose State University

DATA 270: Data Analyst Processes

Individual Research Report (Model Development)

Dr. Eduardo Chan

December 5, 2022

## 4. Model Development

Getting to know about the situation immediately after any calamity/disaster is extremely difficult. Social media like Twitter, Facebook, Instagram, and Flicker could be the source of up-to-date and reliable information. Images shared on social media by users in and around disaster zones could be useful for locating and delivering emergency assistance. The research project aims to collect the images from Twitter and classify them as flood / non-flood by leveraging deep learning technologies. It can assist disaster management organizations in delivering essential emergency responses and saving lives. The project follows CRISP-DM (Cross Industry Process for Data Mining) process by Saltz J. and Hotz N. (2018), where business understanding, data understanding, and data preparation phases have been accomplished.

For the research, a sample image dataset was collected consisting of images related to floods and hurricanes. It was sourced from CRISISMMD. The sampled dataset is labelled and is 1.57 GB in size and contains roughly 1,000 images that are specific to hurricanes Harvey, Irma, Maria, and the floods in Sri Lanka. A comma-separated values (CSV) file consists of information about the images like the event during which images were collected, the corresponding tweet id, the associated text, and the attached image.

The collected image data was further explored to understand the label and pixel distribution in the images using exploratory data analysis. Data were first cleaned to handle inconsistent, incomplete, and missing data. Then, various pre-processing techniques like image resizing and denoising were applied. The data was then transformed by performing image normalisation. As a preparation for the modelling phase, the data was divided into 80% and 20% for the training set and testing set respectively. The 80% of the data was further split in a 3:1 ratio such that 60% will be used for training, 20% for validation and 20% for testing in the next phase of model development.

### 4.1 Model Proposals

To accurately classify and detect the flood in the images, the right choice of model concerning image classification is a must. The selected model should be able to process the image, identify the object and area within an image, able to handle massive image data and make accurate predictions. Keeping in mind the above requirements, overall, four different state-of-the-art architecture models VGG-16, ResNet-50, AlexNet and GoogLeNet are proposed for image classification of flood / non-flood. Convolutional Neural Network (CNN), a supervised deep learning algorithm forms the base for all four proposed image classification models. However, the research paper focuses only on the GoogLeNet model.

#### *4.1.1 GoogLeNet Model*

First introduced by Szegedy et al. (2015b) at ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14) set a breakthrough in classification and object detection. In comparison to previous models AlexNet and ZF-Net which were the winners of ILSVRC 2012 and 2013 respectively, GoogLeNet has provided a noticeable decrease in the error rate which is less than the error rate by VGG (2014 runner-up). Since then, the model is being used by various researchers in a variety of applications to solve classification problems. A few of them are listed below.

The researchers Szegedy et al. (2015b) in their paper talk about how GoogLeNet architecture reduced the computational operations within the network by going deeper and wider by keeping the computational budget constant. They said that the easiest way to improve network performanace is by increasing the network size which entails expanding the network's breadth, or the number of units at each level, as well as its depth, or the number of levels. However, this could make the network more prone to overfitting and increase the computational resources when dealing with a limited number of labelled datasets. Hence, they suggest making use of sparsely connected architecture rather than fully connected, even

within the convolution blocks. Furthermore, they say that the architecture is great at handling the computing capacity though there is an increase in the number of units at each step. They called GoogLeNet as Inception / Inception V1.

Bocanegra and Haddad (2021) in their paper, used Convolutional Neural Network (CNN) models AlexNet, GoogLeNet and ResNet50 on the images taken by Unmanned Air Vehicle (UAV) to classify all the different types of damages caused by natural disasters. They suggest using this information to help people avoid the routes during disaster times. They also talk about different metrics they choose to compare the outputs of various chosen network which are precision, recall, and F1 score for the image classification task.

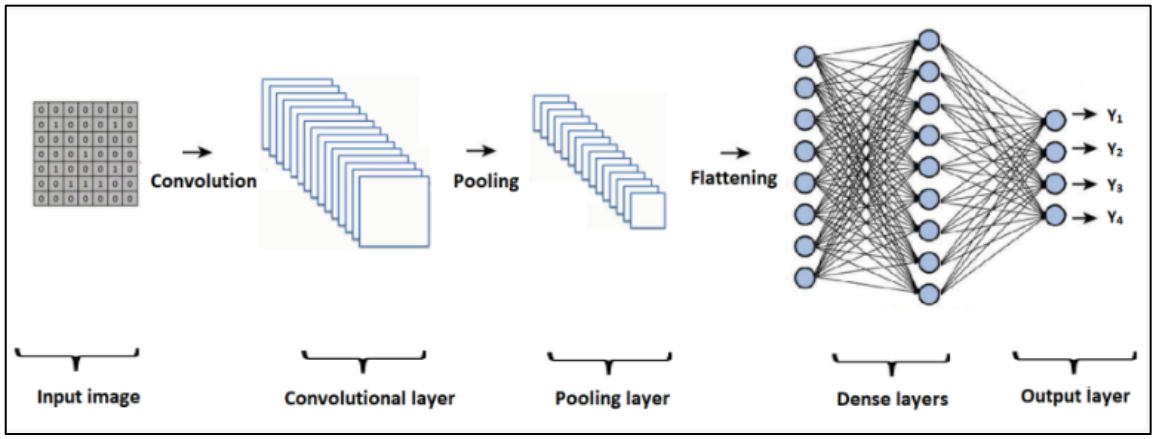
Alam et al. (2020) in their paper classify the images into four disaster types of floods, fire, earthquake, and other. They used individual data and used data from AIDR, Bing Google, Flicker and Instagram and performed classification using deep learning models SqueezeNet, InceptionNet, MobileNet, ResNet18, ResNet50, and ResNet101. They conclude by saying consolidated datasets provided greater efficiency and EfficientNet outperforms all other models.

Rahnemoonfar et al. (2021) in their paper bring about the importance of using and collecting Unmanned Air Vehicle (UAV) images rather than satellite images because of their clarity. They compare the performances of different baseline CNN models with Fully Convolutional network (FCN) models for image classification using UAV and satellite images. They conclude that UAV images performed better over satellites as they would be taken at low altitudes and the images would have high resolution.

The GoogLeNet architecture is a deep CNN-based architecture built on top of it by repeated use of different layers in CNN. Hence, CNN forms the backbone of GoogLeNet Understanding CNN algorithm concepts and the layers in form a foundation for the detailed overview and data flow of the GoogLeNet model architecture. Albelwi & Mahmood (2017) in their research paper describe the CNN algorithm model architecture shown in figure 1. The associated layers and their functions are also explained with necessary equations.

**Figure 1**

*Sample CNN Algorithm Architecture with Convolutional, Pooling, Dense and Output Layers*




*Note.* Albelwi, S., & Mahmood, A. (2017). A Framework for Designing the Architectures of Deep Convolutional Neural Networks. Entropy, 19(6), 242.

GoogLeNet is 22 layers deep consisting of five fully connected / dense layers, five pooling layers, two SoftMax activation layers, and nine inception blocks consisting of convolutional, max pooling and average pooling layers. Overall, it makes use of more than 100 layers for object detection and classification. Each of the layers in the GoogLeNet deep CNN architecture is explained below with the necessary equations each layer uses for computation whenever the input passes through that layer.

The input image which is a matrix of numbers is given to the convolutional layer. It comprises kernels or filters which is again a matrix of numbers whose main function is feature extraction of the image. The operation of convolution is performed by sliding the

filter across the height and width of the input image and calculating the input and filter dot product which is given in equation (1) as follows.

$$a \,.\, b = \sum_{i=1}^{n} a_i * b_i \tag{1}$$

Feature maps / Activation maps of the size of the filter are generated by repeated convolution of the filter with the input image. The filter size and stride (steps at which convolution should happen) determine the output feature map size, therefore when we convolve an input image of size (I x I) with the filter of size (F x F), padding (P) and stride (S), the resulting output size (O x O) is given by equation (2) below.

$$O = \frac{(I - F) + 2P}{S} + 1 \tag{2}$$

where P is padding. It is given by the number of pixels added to the image at the boundary when the image is being processed by CNN and is done for the filter to cover the image and is given by equation (3) below.

$$P = \frac{K-1}{2} \tag{3}$$

The early convolutional layers extract the low-level feature information from the images, whereas the later layers collect the information about the high-level features, such as lines shapes and specific objects. Rectified Linear Units (ReLU), a non-linear activation function is commonly used to introduce non-linearity into the model. It outputs 0 in case of negative input, and for the positive value, it returns the suitable value. It is given by equation (4) as follows. In general, ReLU makes training faster in comparison to other activation functions like tanh, sigmoid etc.

$$f(x) = \max(0, x) \tag{4}$$

The feature maps generated from the convolutional layer output are then given to the pooling layer. Pooling splits the input feature maps into different regions with size (R x R). It produces single output from each region. This layer dimensionally reduces feature map size

and decreases the computational power needed for model training. Pooling is of two types. Max pooling returns the maximum value amongst the input covered by the filter area. Average pooling gives the average value of the area covered by the filter in the given feature map.

The output from the pooling layer is then given to the Fully Connected layers (FC Layers) also called densely connected layers which do a linear transformation to the input with a weight matrix. Taking the dot product of the input and weight matrix, and then applying the non-linear transformation using an activation function gives the output vector. It is represented by the below equation (5).

$$Y_{jk}(x) = f(\sum_{i=1}^{n_h} w_{jk}\, x_i + w_{j0}) \quad (5)$$

where $w_{j0}$ is the bias that is introduced, $w_{jk}x_i$ is the dot product of weight matrix and input which is wrapped by the activation function f. The last layer which is usually used in many of the CNN-based models like GoogLeNet is the SoftMax layer. The layer converts the vector to the probabilistic distribution of target classes. It is given in equation (6) below.

$$\mathrm{Y(i)} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad (6)$$

On a high level, GoogLeNet requires the model input to be 224 x 224 x 3 (height x width x number of colour channels). Hence, the images were resized to the required size as part of data pre-processing and transformation. It also required the image to be RGB with three channels Red, Green and Blue. Hence, the images were not converted to grayscale. The research project uses RGB images of flood / non-flood with size 224 x 224 x 3 as an input, passes through each of the deep network layers, and then classifies the image if it is flooded or non-flooded based on the probabilistic distribution of outputs.

Given the wide range of design options, defining, and designing the model architectures can be very difficult. One can't be aware of the ideal model architecture which could give the best results/predictions. It is all about changing and experimenting on certain

important parameters on which the model is highly dependent and verifying how the model is behaving. The parameters for which the model behaves the best can be used for further deployment. These parameters which are used to control the model's learning process are hyperparameters. As GoogLeNet is a CNN model, the hyperparameters that apply to CNN are also mostly applicable to GoogLeNet. However, the suitable hyperparameters for GoogLeNet are of two categories. Network structure hyperparameters include kernel size (size of the filter), stride, padding, activation functions and hidden layer. Trained network hypermeters which include batch size, momentum, learning rate, and the number of epochs. One can experiment with multiple hyperparameters to derive the best parameters and then train the model with the best ones. Below table 1 shows different hyperparameters that can be used to optimize the GoogLeNet model.

**Table 1**

*Sample table showing hyperparameters for GoogLeNet model*

| Hyperparameter | Abbreviation | Range |
|---|---|---|
| Kernel size | kernel_size | Anything suitable |
| Stride | strides | Anything suitable |
| Padding | padding | [‘same’, ’valid’] |
| Activation function | activation | [‘relu’, ’lrelu’, ‘sigmoid’, ’softmax’] |
| Optimiser | optimiser | ['Adam', 'SGD', 'Adam', 'SGD'] |
| Learning rate | learning_rate | [0.001, 0015, 0.002, 0.003] |
| Momentum | momentum | [0.9, 0.95, 0.99] |
| Number of epochs | epochs | Anything suitable |
| Batch size | batch_size | [32, 64, 128, 256] |

*Note.* Table shows hyperparameters and their approximate range for GoogLeNet model

## 4.2 Model Supports

### *4.2.1 Environment, Platform, Tools*

The end-to-end implementation of the model requires the proper choice of hardware, software, and necessary tools to keep the overall costs minimal. Modelling includes developing, training, testing, and evaluating the model on the dataset.

For the research project, the model development phase would need sturdy hardware i.e, a MAC system, or a Personal Computer (PC) with Central Processing Unit (CPU) and Random Access Memory (RAM). It would also need a few software's like mac Operating System (OS) Monterey version 12.6 or Windows OS 11, Anaconda distribution package for macOS / windows with python version 3.9 which serves as an integrated development environment (IDE) platform, Google Collab also as IDE as it provides Graphics Processing unit (GPU) and RAM which can be used to train, test, validate the developed model and to maintain computational efficiency. Tools like ClickUp and Zoom are also needed for project management and collaboration with faculty respectively. The various hardware, software and tool requirements along with their usage are given in table 2 below.

**Table 2**

*Hardware, Software and Tool Requirements for Model Development*

| Requirements | | Configuration / Version/Specification | Usage |
|---|---|---|---|
| Hardware | *PC/MAC System* | 32 GB RAM, 8-core processor, with minimum 256 GB storage space. | Developmental activities, training, and testing of model |
| Software | MacOS / Windows OS 11 | 12.6/11 | Manage computer resources like CPU and provide software services |
| | Anaconda distribution package for MAC/Windows | Python 3.9, Jupyter Notebook | IDE platform to develop a deep learning model |
| Tools | Google Drive | Desktop | Data storage and to run Google Colab instance for mode development |
| | Google Colab with runtime GPU | K80 GPU and 12 GB RAM | IDE platform to develop a deep learning model and training and testing deep learning model with GPU |
| | ClickUp | Desktop | Assign, manage and track project tasks |
| | Zoom | Client | Collaboration |
| | GitHub | Desktop version | Repository for code storage and version control |

For the implementation of the deep learning model in Jupyter Notebook / Collab the research makes use of multiple libraries and packages which are provided in detail in the below table 3.

**Table 3**

*Libraries/Methods and their Usage in Model Development*

| Library | | Method | Usage |
|---|---|---|---|
| Pandas | DataFrame | head | To display records from top in the DataFrame |
| | | info | To print the information about the DataFrame |
| | | shape | To display the number of rows and columns in the DataFrame |
| | | read_csv | To read the .csv file in the system |
| Numpy | | array | Convert list values to array |
| | random | seed | To make sure same data is used across model development |
| | | zeros | Define array of zeros |
| cv2 | cvtColor | COLOR_BGR2RGB | To convert BGR color format to RGB color format |
| | | resize | Changing the dimension (height and width) of the image |
| | | imread | Read the image |
| | | fastNlMeansDenoisingColored | Eliminating noise in the image |
| | | normalize | To perform min-max normalisation of the image |
| | | Open | Open the image |
| Os | | listdir | List all the files and directories in the given path |
| Matplotlib | pyplot | plot | To plot the graph |
| | | xlabel, ylabel, title, legend | To set x axis label, y axis label, title and show legend values |
| | | show | To display the plot |
| Tensorflow | | Data.dataset from_tensor_slices() | Convert the data to Tensorflow datatype and slice the input array and convert into TensorFlow object form. |
| | Keras.layers | Input | To specify the input to the model in a certain shape |

| Library | | Method | Usage |
|---|---|---|---|
| Tensorflow | Keras.layers | Dense | To compute dot product of the input with kernel and assign weights and biases to the same |
| | | Conv2D | Convolution operation of input with filter |
| | | MaxPooling2D, AveragePooling2D | To down sample the input and reduce dimension |
| | | Flatten | To convert the feature map into single dimensional vector. |
| | | Dropout | To filter only certain inputs that are necessary for the output. |
| | | BatchNormalization | To normalise the data such that mean is close to zero and standard deviation to one. |
| | | concatenate | Combine the output of multiple previous |
| | Keras.model | summary | To print the summary of the model |
| | | fit | To fit the created model to the data |
| | Keras.utils | Plot_model | To plot the diagram of the model |
| | Keras.metrics | Sparse categorical crossentropy | To define the loss function for the model |
| | Keras.optimisers | ADAM | To alter the learning rate and weights to reduce model loss |

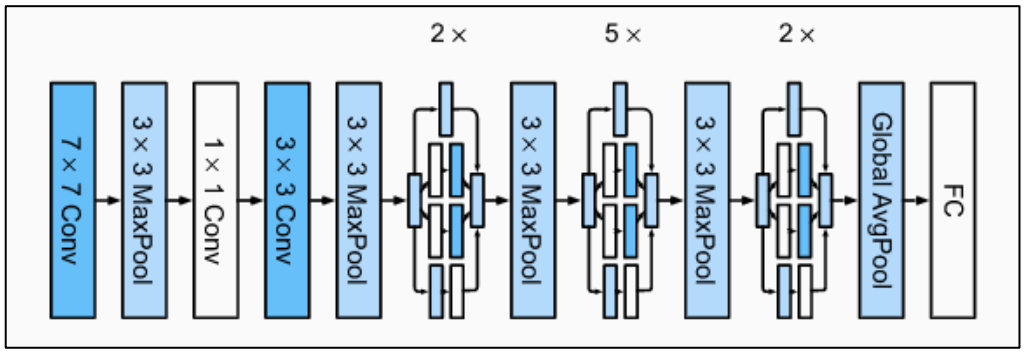
### *4.2.2 Model Architecture*

Knowledge and understanding of the base model and necessary resources required for model development set a suitable background to understand the architecture and the data flow of the proposed model GoogLeNet. This model can detect around 1000 different types of objects and was built using a different kind of architecture that enables the expansion of the network's width and depth while preserving its computational capacity. Figure 2 shows

the simplified architecture of GoogLeNet. The detailed architecture proposed by Szegedy et al. (2015b) is shown in the appendix section.

**Figure 2**

*Simplified GoogLeNet architecture*




*Note.* Dive into Deep Learning — Dive into Deep Learning 1.0.0-alpha1.post0 documentation.(n.d.).

The model required the image to be of the size of 224 x 224 x 3 (height x width x number of channels) which was done as part of data pre-processing and transformation. From the pre-processed data stored in google drive, each image is read and served as input for the model. The input of size 224 x 224 x 3 is convolved with a convolutional layer with 64 filters of kernel/filter size 7x7, the stride of 2 and padding as 'same'. A sample calculation of how the output is generated is as below using equation (1) and equation (2).

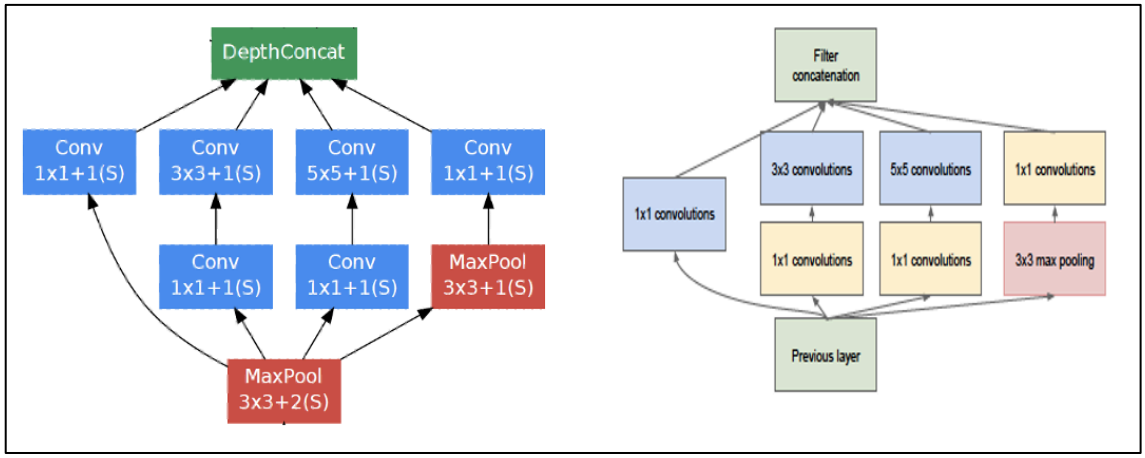
output image size = ((224 -7 + 2(3)) /2 ) + 1

The output of this layer would be 112 x 112 x 64 with a depth of 1. The first convolutional layer uses a relatively large filter size of 7 x 7 which is quite large than other filters in the entire network. The goal is to reduce the image size/dimension without losing any feature information. This serves as an input for the max pooling layer which has a filter of size 3 x 3, stride 2 and padding as 'same'. Using a similar calculation as in equations (1) and (2) the output of the max pooling layer would be 56 x 56 x 64 with a depth of 0. This output is batch normalised and is sent through a 1 x 1 convolutional layer with 64 filters and 3 x 3

convolutional layers with 192 filters simultaneously to get output with image dimensions 56 x 56 x 192 with a depth of 2. This is passed through the max pooling layer with a 3 x 3 filter and stride of 2 which dimensionally reduces the size of the input feature map and decreases the computational power needed for model training. The output size at this stage is 28 x 28 with 192 feature maps (28 x 28 x 192). Prior to reaching the inception module, the image size has already been reduced by a factor of eight. The output is then moved to the inception module which is the whole point of why GoogLeNet architecture was designed. The inception module in the network is shown in figure 3 below.

**Figure 3**

*Inception module with dimensionality reduction feature*




As the network gets deeper, due to an increase in parameters, overfitting can come into the picture. To solve this problem, Szegedy et al. (2015b) in their architecture used filters with multiple sizes which can be operated at the same level where in a way the network gets wider rather than deeper addressing the problem. The inception module makes use of two 1 x 1 convolution which is used to decrease the weights and biases (parameters) in the network which in turn decreases the number of operations drastically before 3 x 3 and 5 x 5 expensive convolutions are performed which ultimately leads to computational efficiency.

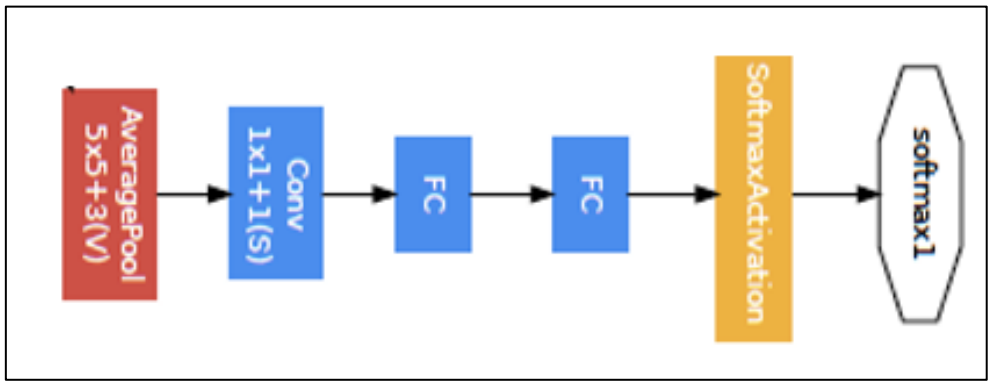
The output with of size 28 x 28 x 192 from the max pool layer is passed through two inception blocks (3a) and (3b) and is then again through the max pool layer with the filter of

3x3 and stride 2 to get them out the size of 14x14x480. The details on the size and the parameter at each stage are given in the table shown in figure 4. The output then passes through seven more inception blocks (4a), (4b), (4c), (4d), (4e), 5(a) and 5(b) with a max pooling layer between 4(e) and 5(a).

As the network gets bigger, another problem is they suffer from a vanishing gradient problem. To tell it in layman's terms, during training, the network just stops learning. To avoid this, auxiliary classifiers are added to the network to inception 4(a) and 4(b). The auxiliary block from the GoogLeNet architecture is shown in figure 4 below.

**Figure 4**

*Auxiliary block from inception module*




Auxiliary blocks consist of two fully connected layers, one average pooling layer, one convolution layer, and a softmax activation layer. It is only used during training instances are eliminated during validation/testing. The function of the auxiliary classifier is to do classification based on inputs in the middle of the network and add the training-related loss back to the network's overall loss.

The output of the inception (5b) module will have a drastically reduced image size of 7 x 7 x 1024 which is then sent to the global average pooling layer with a filter size of 7 x 7 and stride as 1. This again averages the output to 1 x 1 making the trainable parameters 0 and in turn improving the accuracy. It is then passed through a fully connected layer, where the input is linearly transformed into a vector. It is then passed through the softmax activation

layer, which makes use of the softmax activation function. Its function is to assign probabilities to the input based on the target class which is flood / non-flood in the research project. Figure 5 shows in detail the output size, depth, parameters, and the number of operations at each layer of the network, as the data flow from one layer to another.

**Figure 5**

*Detailed architecture, parameter, and output details at each stage of GoogLeNet*

| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

### 4.3 Model Comparison and Justification

The research project has made use of four different CNN models for the classification of images as flood/non-flood. They are ResNet, VGGNet, AlexNet, GoogLeNet. As this research paper is focused on GoogLeNet, its comparison is made against all other models to see how it is different from all three models.

GoogLeNet is a depth and width based 22 layers deep architectural network possessing seven million parameters consisting of nine inception modules, five fully connected layers, three average pooling layers, five max-pooling layers, four convolutional layers, and three softmax layers in the network for the auxiliary classifiers. VGGNet-16 is a shallow 16-layer deep architectural network possessing 138 million parameters which are

quite difficult to handle. It consists of two fully connected layers, 13 convolutional layers, and one softmax layer which is ideally a small network comparatively. AlexNet on the other hand is eight layers deep possessing 60 million parameters and consisting of five convolutional layers, three fully connected layers which are smaller than all other networks used in this project. ResNet50 is a 50-layer depth architectural network possessing 25.6 million parameters with one max pool layer, 48 convolutional layers, and one average pool layer. Overall, when it comes to network size AlexNet has a smaller network and when it comes to complexity, GoogLeNet is less complex with just seven million parameters to handle.

Given the fact that audio, video, and images can be represented numerically the models used can process all these data types and they work well when the dataset is large. Even with the large data, all four models are prone to overfitting and underfitting. The model networks are designed in a way to address overfitting and underfitting which is shown in table 4. Small data can lead to overfitting in all the models resulting in improper predictions which need to be handled by various techniques like large datasets, cross-validation, data augmentation etc.

All the models need data pre-processing to a certain extent like data cleaning, data transformation and augmentation before feeding into the model. GoogLeNet, AlexNet, and VGGNet-16 required input of dimension 224 x 224 x 3. However, ResNet50 required input to be of the size 64 x 64 x 3. The good part is these models do not require any dimensionality process to be done as the network is capable of handling it by using different filters size. The training time of each of these models depends on data size, and the number of cycles for which the model is trained. Originally on the ImageNet data, AlexNet took 6 days to train for 90 cycles, GoogLeNet took a week, VGGNet16 took two weeks and ResNet50 took 14 days. All four models would need storage space and the support of a good GPU to train the model.

Szegedy et al. (2015b) in their paper talk about GoogLeNet's main advantages which are the presence of an inception module with a 1 x 1 filter which can dimensionally reduce data and reduce the overall computations making it efficient and avoid overfitting because of sparely connected architecture. However, the limitation they mention is the memory usage due to the time it takes to train the model on a large dataset.

On the other hand, AlexNet's architecture is small and hence requires less computational time and resources. The model also addresses overfitting by the usage of a dropout layer after every fully connected layer. However, the limitation is low accuracy.

The VGGNet-16 main advantage is its faster learning which requires less training time. However, it comes with a limitation of less accuracy

ResNet50 resolves the issues which most of the deep networks face which is the saturate and degraded accuracy. It does this by increasing the number of layers by using skip connections called residuals. However, it comes with a limitation of more training time and resources.

***Table 4***

*Characteristic comparison of state-of-the-art architecture models*

| Characteristic | AlexNet | GoogLeNet | ResNet | VGGNet |
|---|---|---|---|---|
| Network version | AlexNet | GoogLeNet / Inception V1 | ResNet-50 | VGGNet-16 |
| Architecture | Deeper | Deeper and wider | Residual | Shallow |
| Parameters (In millions) | 60 | 7 | 25.6 | 138 |
| Network depth | 8 | 22 | 50 | 16 |
| Training time | Low | High | High | Low |
| Layers | Five convolutional layers, there fully connected layers | Five convolutional layers, five fully connected layers, three average pooling layers, three softmax layers,5 max pooling layers, nine inception blocks. | 48 convolutional layers, one max pool layer, one average pool layer | 13 convolutional layers, two fully connected layers, one softmax layer |
| Supported data type | Image, Audio, Video | Image, Audio, Video | Image, Audio, Video | Image, Audio, Video |
| Addressed overfitting by | Dropout layer after every fully connected layer | Moving from fully connected to sparsely connected architecture | Dropout techniques and regularisation of weights | By adding L2. Dropout, batch norm regularisation |
| Input image size | 227 x 227 x 3 | 224 x 224 x 3 | 64 x 64 x 3 | 224 x 224 x 3 |
| Preferred dataset size | Large | Large | Large | Large |

## 4.4 Model Evaluation Methods

The goal of the research project is to classify the images as flood / non-flood which is a binary classification task. Once the model is developed it is important to know and understand how the model is performing if the model is very poor in predictions or doing a great job in accurately classifying the images. Researchers have already defined the metrics which help evaluate the developed classification model. The metrics used for the evaluation of the GoogLeNet model are loss, accuracy, f1-score, precision, and recall. A confusion matrix for a binary classifier helps understand most of the metrics here.

### *4.4.1 Confusion matrix*

It gives an interpretation of how many images are classified correctly as flood and non-flood and how many are classified incorrectly (images with flood classified as non-flood and vice versa). It can be shown in table 5 below.

**Table 5**

***Confusion Matrix***

| **Class designation** | | **Actual class** | |
|---|---|---|---|
| | | flood | non-flood |
| **Predicted class** | flood | True Positive (TP) | False Positive (FP) |
| | non-flood | False Negative (FN) | True negative (TN) |

The data point is True Positive (TP) if the image with flood has been classified as a flood. The data point is True Negative (TN) if the image with non-flood has been classified as non-flood. The data point is False Negative (FN) if the prediction is made as non-flood when it is a flood. It is also called a Type 1 error. The data point in the confusion matrix is False Positive (FP) when a flood is predicted, however, what happened is non-flood. Accuracy, F1-score, precision score and recall can be calculated using the confusion matrix as below.

### *4.4.2 Accuracy*

Accuracy is calculated as the number of correctly made predictions by the total number of predictions. Here, it is the sum of the count of images for which the model classified as flood and non-flood accurately to the total number of predictions made. It is given by as in the below equation (7)

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \tag{7}$$

### *4.4.3 Precision*

Precision is the ratio of the number of images correctly classified as a flood to all images predicted to be a flood. This metric is important when we want to give importance to False Positives than False Negatives. It is given by as in the below equation (8)

$$\text{Precison} = \frac{\text{TP}}{\text{TP+FP}} \tag{8}$$

### *4.4.4 Recall*

The recall is defined as correctly predicted flood classes to all images that are flooded. It is given by as in the below equation (9)

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{9}$$

### *4.4.5 F1-Score*

F1-Score is defined as an indicator of a test's accuracy, and it is a great performance when multiple classifiers must be compared. It takes both precision and recall into account. It is given by as in the below equation (10) and equation (11)

$$\text{F} - \text{Score} = \frac{2*\text{Precison}*\text{Recall}}{\text{Precision}*\text{Recall}} \tag{10}$$

$$\text{F} - \text{Score} = \frac{\text{TP}}{\text{TP}+\frac{1}{2}*(\text{FP+FN})} \tag{11}$$

### *4.4.6 Loss*

The GoogLeNet model makes use of sparse categorical loss entropy function to calculate Cross Entropy Loss (CEL) between predictions and labels. It is used when the target

has two or more labels like flood/non-flood in this case. It is a representation of how model's misclassification for a specific example. The loss of the model is zero if the model can predict accurately; otherwise, the loss is higher. The entire goal of training the model is to find suitable weights and biases to be added to the input such that the overall loss of the model is low. Easy-to-classify photos may have a considerable impact on the CEL's loss function value and, consequently, govern the gradient that restricts learning from challenging images. The CEL is given by equation 12 as below.

$$\text{CEL} = \sum_{i=1}^{n} \log (p_i) \quad (12)$$

Where $p_i$ is the softmax probability of $i^{th}$ class and a log is a log to the base 2 or ln ().

Using the defined metrics above, model accuracy, precision, recall, F1 and loss curves can be plotted and can be compared for training and validation data during each epoch.

**4.5 Model Validation and Evaluation Results**

Once end-to-end model implementation is completed, it can be evaluated using the metrics defined in section 4.4 to check the overall performance of the model in terms of classification of the target as flood/non-flood. Table 6 below shows the evaluation metrics of validation data for the GoogLeNet model after implementation. Figure 6, figure 7, and figure 8 show the supporting results for each of the metrics through the plots.

**Table 6p**

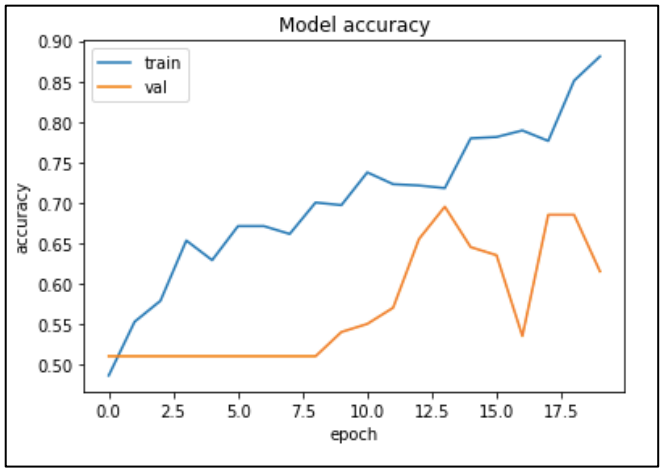
*Evaluation metric table for GoogLeNet*

| Model | Accuracy | Precision | Recall | F1-Score | Loss |
|---|---|---|---|---|---|
| GoogLeNet | 0.5938 | 0.5729 | 1 | 0.7251 | 1.9586 |

One can say that only 59% of the time model correctly classifies the target output as flood or non-flood. Upon observation of the metrics in table 5, we can see that for the model, recall is 1 which is perfect indicating the model never classifies the data as a false negative that is non-flood in this case. However, this is not the case here looking at other metrics accuracy which is 59% and loss which has a high value of 1.9586, indicating that the model is not able to classify the image appropriately.

**Figure 6**

*Model Accuracy Plot for Training and Validation Data*




The curve shows how the training accuracy and validation accuracy of the model varies for 20 epochs (When all training data points are completely passed through the algorithm once it is called one epoch). One can observe that the training accuracy is ~ 87% and it keeps increasing with each epoch indicating the model is performing well on the training data. However, the overall validation accuracy is ~59% which is slowly increasing throughout all epochs which is quite less in comparison to training. There are high chances model is overfitted which can be improved by taking certain measures.

**Figure 7**

*Model Loss Plot for Training and Validation Data*


One can observe from figure 7 that the model loss is decreasing over time for training data which is expected. However, the model loss is increasing for the validation data for each epoch indicating a decrease in the prediction of the model. As the loss is 1.95 which is more for the validation data, it will not be able to make the right prediction from the image if it is flooded/non-flood. For a good-performing model, both training and validation loss should decrease over time.

**Figure 8**

*Model Precision, Recall and F1-score Plot for Training and Validation Data*

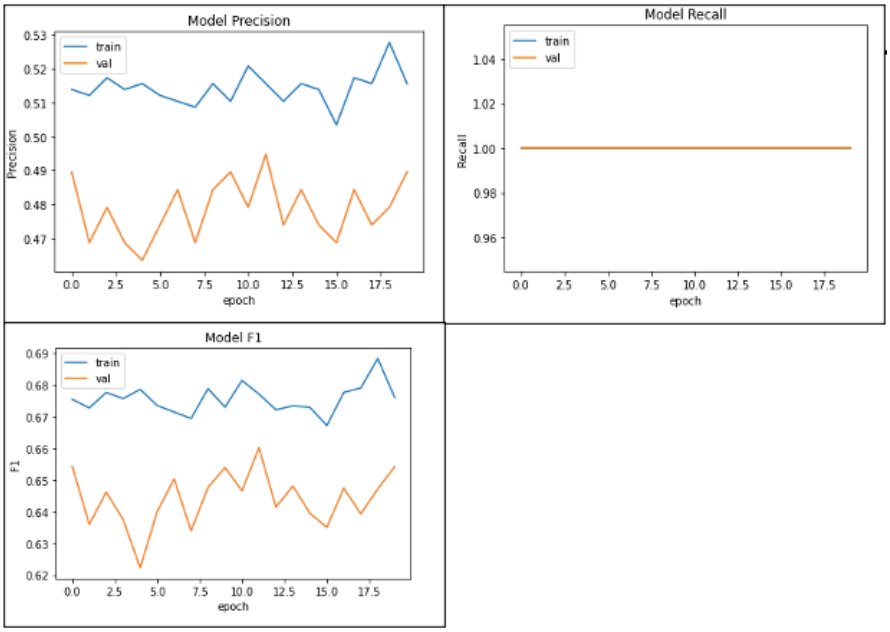


Figure 9 shows the model precision, recall and F1 curve for training and validation data for each epoch. For each epoch, training has good precision and f1over validation. The model is trained to correctly classify flood images as flood and non-flood images as non-flood. However, on the validation data, positive prediction on the images is very poor which is indicated by the precision value of 0.5729. Also, one can observe that the recall is 1. This may be the reason why the F1 score is high comparatively with a value of 0.7251.

These values indicate the model is overfitted and hence is unable to generate good predictions in terms of flood/non-flood classification of images. The reason could be the use of a smaller dataset.

To address the same classification problem there were other models implemented whose evaluation metrics are shown below in table 7

**Table 7**

*Evaluation metric comparison table for AlexNet, ResNet50, VGGNet16, and GoogLeNet*

| Model | Accuracy | Precision | Recall | F1-Score | Loss |
|---|---|---|---|---|---|
| AlexNet | 0.6198 | 0.4921 | 0.9895 | 0.6594 | 0.8330 |
| ResNet50 | 0.60 | 0.6159 | 0.6060 | 0.6053 | 1.69 |
| VGGNet16 | 0.715 | 0.7011 | 0.6630 | 0.6815 | 0.8706 |
| GoogLeNet | 0.5938 | 0.5729 | 1 | 0.7251 | 1.9586 |

*Note.* Metrics adapted from other team members for AlexNet, ResNes50, and VGG16

From above table 6 we can observe, VGGNet16 has the better performance metrics comparatively and the GoogLeNet model has low performance. Some models like AlexNet and GoogLeNet do have the problem of overfitting which can be improved by taking certain measures like training the model for a large amount of data, cross-validation, and data augmentation.

Overall, looking at the evaluation metrics of the GoogLeNet model, one can say that model is not performing great. It can be observed from the accuracy curve in figure 6 that

though the training accuracy is good, the validation accuracy is quite low. Also, in reference to figure 8, the loss of the model is increasing for the validation data indicating the decrease in the prediction the model is making. Figure 8 shows that the recall for the model is 1 indicating an excellent classifier where the model does not predict any false negatives. However, it is contradicting the accuracy, precision and loss values generated by the model.

As the data size increases, it becomes difficult to store and process the data with the computational power provided by the system. One of the major drawbacks of GoogLeNet is the limitation concerning training time and memory usage. Szegedy et al. (2015b) in their paper say, for the larger dataset, the model could take a week to converge.

As a future scope, with an adequate amount of the resources like storage space and GPU, the model can be trained on a larger image dataset to avoid overfitting and generate better predictions of flood and non-flood labels. The performance of all the models can also be experimented with and improved upon by hyperparameter tuning the appropriate parameters. The research can be extended to live stream the tweets during any flood disasters, identify the images as flooded/non-flooded and provide appropriate help to the needy. Furthermore, the research can also be extended to various other disasters like wildfires, earthquakes, etc.

## References

Alam, Firoj, et al. “CRISISMMD: Multimodal Twitter Datasets from Natural Disasters.” *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 12, no. 1, 2018, https://doi.org/10.1609/icwsm.v12i1.14983.

Alam, F., Ofli, F., Imran, M., Alam, T., & Qazi, U. (2020). Deep Learning Benchmarks and Datasets for Social Media Image Classification for Disaster Response. *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. https://doi.org/10.1109/asonam49781.2020.9381294

Alarfaj, F. K., Malik, I., Khan, H. U., Almusallam, N., Ramzan, M., & Ahmed, M. (2022). Credit Card Fraud Detection Using State-of-the-Art Machine Learning and Deep Learning Algorithms. *IEEE Access*, *10*, 39700–39715. https://doi.org/10.1109/access.2022.3166891

Albelwi, S., & Mahmood, A. (2017). A Framework for Designing the Architectures of Deep Convolutional Neural Networks. Entropy, 19(6), 242. https://doi.org/10.3390/e19060242 Network Models for Practical Applications.

Alfredo Canziani, Adam Paszke, & Eugenio Culurciello. (2016). An Analysis of Deep Neural Network Models for Practical Applications. *ArXiv: Computer Vision and Pattern Recognition*.

Aszemi, N. M., & Dominic, P. (2019). Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms. *International Journal of Advanced Computer Science and Applications*, *10*(6). https://doi.org/10.14569/ijacsa.2019.0100638

*Dive into Deep Learning — Dive into Deep Learning 1.0.0-alpha1.post0 documentation*. (n.d.). https://d2l.ai/index.html

Maeda-Gutiérrez, V., Galván-Tejada, C. E., Zanella-Calzada, L. A., Celaya-Padilla, J. M., Galván-Tejada, J. I., Gamboa-Rosales, H., Luna-García, H., Magallanes-Quintanar, R., Guerrero Méndez, C. A., & Olvera-Olvera, C. A. (2020). Comparison of Convolutional Neural Network Architectures for Classification of Tomato Plant Diseases. *Applied Sciences*, *10*(4), 1245. https://doi.org/10.3390/app10041245

Mao, J., Harris, K., Chang, N. R., Pennell, C., & Ren, Y. (2020). Train and Deploy an Image Classifier for Disaster Response. *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. https://doi.org/10.1109/hpec43674.2020.9286248

Rahnemoonfar, M., Chowdhury, T., Sarkar, A., Varshney, D., Yari, M., & Murphy, R. R. (2021). FloodNet: A High Resolution Aerial Imagery Dataset for Post Flood Scene Understanding. *IEEE Access*, *9*, 89644–89654. https://doi.org/10.1109/access.2021.3090981

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Sony, S., Dunphy, K., Sadhu, A., & Capretz, M. (2021). A systematic review of convolutional neural network-based structural condition assessment techniques. *Engineering Structures*, *226*, 111347. https://doi.org/10.1016/j.engstruct.2020.111347

Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015a). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2015.7298594

Tammina, S. (2019). Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images. *International Journal of Scientific and Research Publications (IJSRP)*, *9*(10), p9420. https://doi.org/10.29322/ijsrp.9.10.2019.p9420

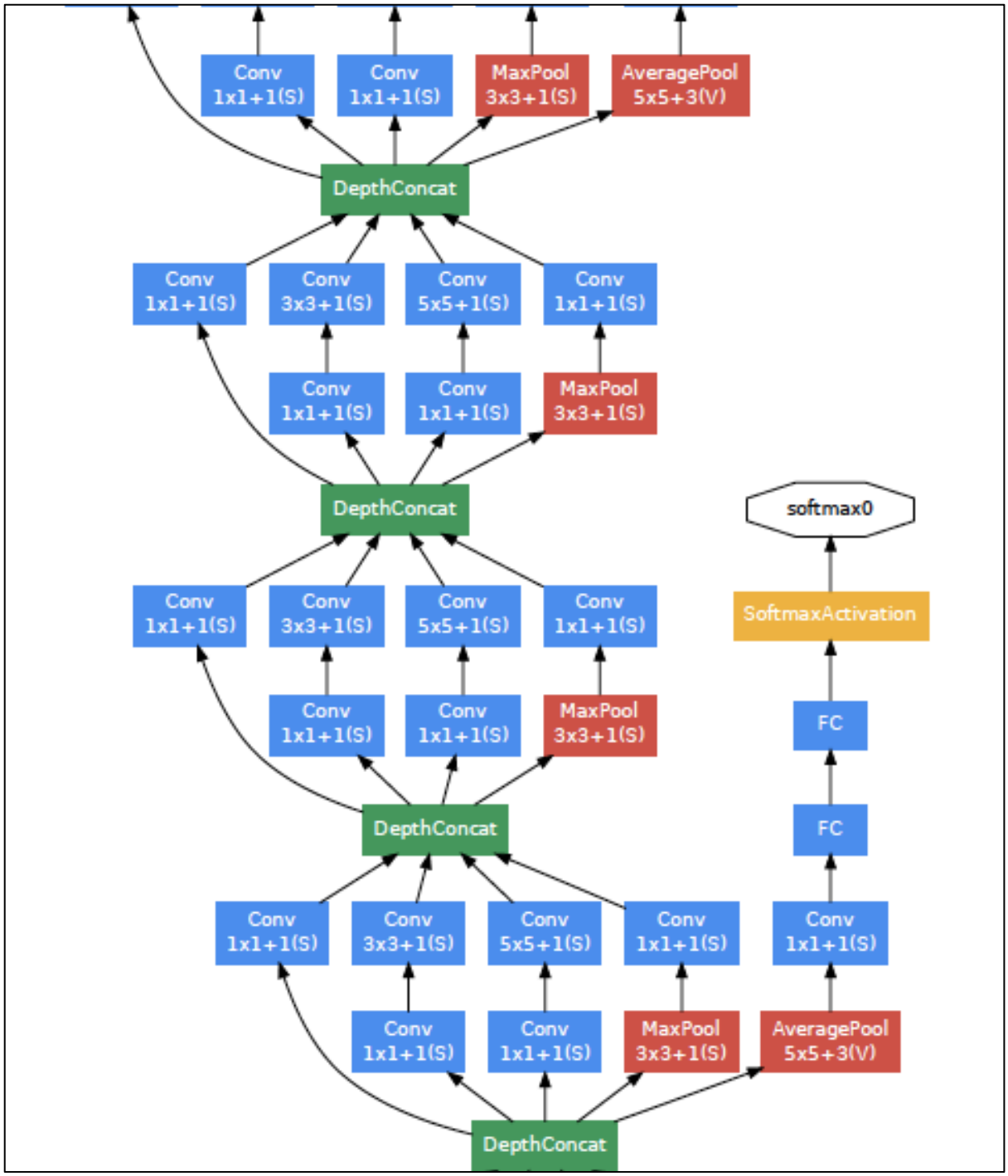
Team, K. (n.d.). *Keras documentation: Losses*. https://keras.io/api/losses/

Vujovic, E. (2021). Classification Model Evaluation Metrics. *International Journal of Advanced Computer Science and Applications*, *12*(6). https://doi.org/10.14569/ijacsa.2021.0120670

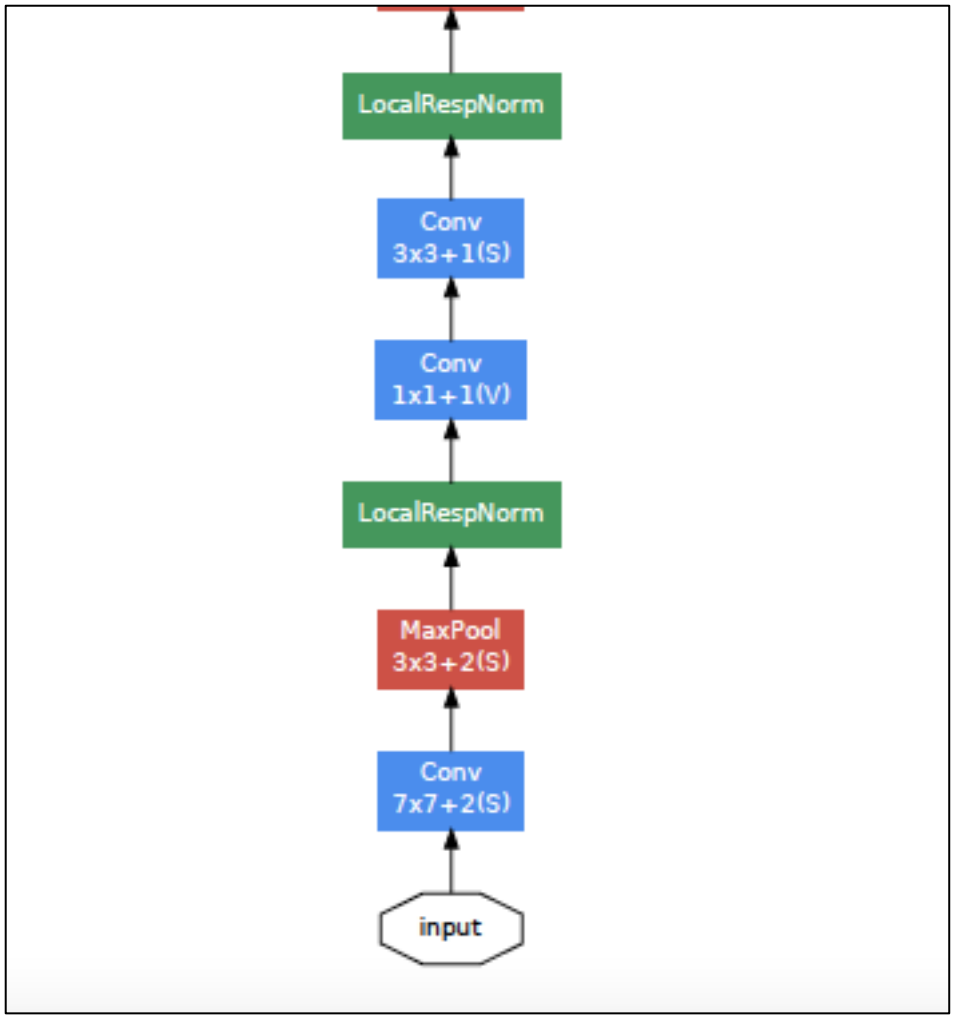
Yessou, H., Sumbul, G., & Demir, B. (2020). A Comparative Study of Deep Learning Loss Functions for Multi-Label Remote Sensing Image Classification. *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*. https://doi.org/10.1109/igarss39084.2020.9323583

# Appendix

## Detailed architecture of GoogLeNet in order


Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
SoftmaxActivation
FC
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat


DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
LocalRespNorm


LocalRespNorm
Conv
3x3+1(S)
Conv
1x1+1(V)
LocalRespNorm
MaxPool
3x3+2(S)
Conv
7x7+2(S)
input