Journal of Information Systems and Informatics
Vol. 6, No. 3, September 2024 e-ISSN: 2656-4882 p-ISSN: 2656-5935
DOI: 10.51519/journalisi.v6i3.788 Published By DRPM-UBD

Predicting Forest Areas Susceptible to Fire Risk Using Convolutional Neural Networks

Ansh Gupta[1]

[1]DPS International Gurgaon, Haryana, India
Email: [1]anshgupta0621@gmail.com

Abstract

Wildfires pose a grave danger and threat to both human health and the environment, which is why early detection of wildfires is crucial. In this study, a convolutional neural network, which is a deep learning technique for computer vision, that is capable of classifying satellite imaging of forest cover in Canada as either being prone to wildfires or not being prone to wildfires is created. This model achieved an accuracy of 95.06% and is not only accurate but also reliable and unbiased in terms of the training set and the test set. We also review an existing model for the same dataset. Furthermore, this study discusses the application of this model in the real world, its feasibility, its future scope, and strategies to improve it.

Keywords: Convolutional Neural Network, Deep Learning, Satellite Imaging

1. INTRODUCTION

A wildfire is an out-of-control and unpredictable fire that burns because of the combustion of vegetation [1]. Despite wildfires being a natural event, crucial for forest regeneration, they have various negative impacts, such as increased air pollution, habitat loss, soil erosion, and more [2], [3]. The Canadian wildfires of March 2023 escalated in June and July of 2023, destroying 22 million acres of land. Furthermore, states in the USA, such as New York, have suffered from an orange haze covering the skies in early June, making it the city with the worst air pollution at the time [4]. For this reason, preventing wildfires is crucial. Wildfires have the natural role of strengthening soil and maintaining biodiversity; zowever, due to global warming, the frequency of wildfires has increased making it a more troublesome issue [5], [6].

Satellites monitor wildfires today by observing land-use and land-cover changes (LULCC) [7]. They both slightly differ in definition, as Land-cover is the natural and anthropogenic characteristics that one can observe on the surface, enabling it to cover different biomes and types of land such as wetlands, rainforests, water bodies, etc. On the other hand, land use refers to the activities that take place on

land, such as cars, urban infrastructure, etc. LULCC plays a key role in tracking and observing wildfires as well as collecting data that can help us form solutions. One way of using LULCC data is to train artificial intelligence models that can help us predict when or where a wildfire is likely to occur. With the rise of artificial intelligence in modern times, the time to use AI to aid in wildfire prevention has never been better. Furthermore, the integration of artificial intelligence in satellite imaging can increase the predictive power for detecting signs of wildfires [8]. Computer vision is a field of artificial intelligence that helps computers interpret and understand visuals, whether they are videos or pictures. Using data on wildfire-prone areas from ongoing and past wildfires, such as the Canadian wildfires, can help us build strong computer vision models [9]. This paper aims to investigate how the integration of computer vision in satellite imaging can help us detect and mitigate forest fires.

To build such models, we need to use the widely known deep learning technique known as convolutional neural networks (CNNs), which is used by machine learning engineers throughout the world because of its efficient training capacity and ability to identify edges and shapes of objects in images [10]. Neural networks interpret images as a 3D matrix of pixels, where each pixel is an RGB (red, blue, and green) matrix. Regular neural networks struggle to work with images as they pile up too many parameters, leading to overfitting. However, convolutional neural networks are neural networks with a more sensible architecture specifically made for image inputs [11]. The architecture of a CNN consists of three fundamental types of layers: a convolutional layer, a pooling layer, and a fully connected layer [11]. The convolutional layer strides a filter matrix over the input image, computing the dot product between each local area to generate a new matrix containing features of the image that the neural network will use for classification. The pooling layer down-samples the image along the spatial dimensions (width and height of the image) to reduce the number of parameters and prevent overfitting. The fully connected layer, the last layer, computes the class scores (probabilities of an image belonging to the classes or categories). In summary, the input of CNNs is an image matrix with a depth of 3 for each color channel, with the output being a matrix of class cores. In this paper the CNN's performance is improved upon by experimenting with the model's number of layers and by experimenting with various hyperparameter values.

2. METHODS

2.1 Research Flowchart

The research flowchart is a visual summary of the process followed in this paper as shown in Figure 1. By following these steps, existing models are improved, and

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

the real-life applicability, performance and model reliability is investigated. By following the process, structured research is carried out.

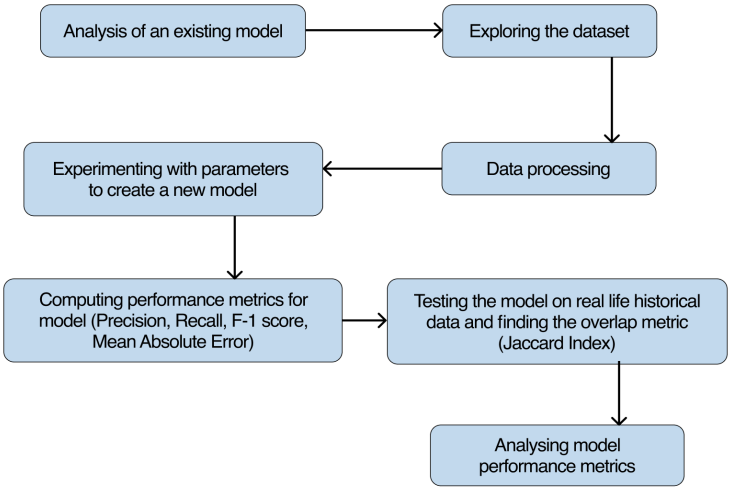

Figure 1. Research Process

2.2 Data Description

To build a CNN specific to classifying wildfire-prone and wildfire-safe areas, we needed a large number of past satellite images. In this paper, a dataset from Kaggle consisting of satellite images of forests in Canada is used to categorize pictures into wildfire-prone and non-wildfire-prone areas [12]. The images are all 350x350 in resolution, with approximately 42000 images divided into 70% training, 15% testing, and 15% validation data. This data will be pre-processed and formatted into the correct data structure so that it can be fed into the neural network for training for the most accurate metrics.

2.3 Existing Model Analysis

The model chosen for analysis is a notebook created by Abdelghani Aaba, Bouchra Rakhiss, and Msalek Aicha on Kaggle for the same dataset used to create our model [12], [13]. The convolutional neural network analyzed features the following architecture:

Table 1. Architecture of existing model

Layer [type]	Output Shape	Params
Conv2D	(None, 349, 349, 8)	104

Max Pooling	(None, 174, 174, 8)	0
Conv2D	(None, 173, 173, 16)	528
Max Pooling	(None, 86, 86, 16)	0
Conv2D	(None, 85, 85, 32)	2080
Max Pooling	(None, 42, 42, 32)	0
Dropout (0.5)	(None, 42, 42, 32)	0
Flatten	(None, 56448)	0
Dense	(None, 300)	16934700
Dropout (0.4)	(None, 100)	0
Dense	(None, 2)	602

Total parameters: 16,938,014
Trainable parameters: 16,938,014
Non-trainable parameters: 0

1) Convolution Layers

There are 3 convolution layers in this architecture, in which each successive Conv2D layer gains more depth each time. As mentioned earlier, convolution layers extract features from images by sliding filters over them. In this case, a filter of size 2x2 is used [14].

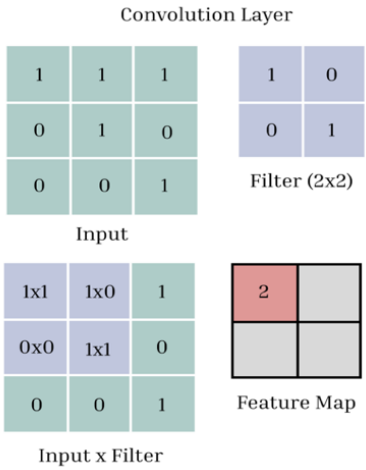

Figure 2. Visualization of a convolution layer

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

Depth in a convolution layer refers to the number of filters in the layer. The reason behind the ascending order of the number of filters is that the initial image is inputted as raw pixel data, which is quite noisy, so complex shapes are harder to extract at the raw pixel value. This is why the initial convolution layers have only a few filters so that basic edges and shapes can be extracted, and later in the layers, the more complex features can be extracted on a less noisy image. The more filters, the greater the number of features that can be extracted from the image.

2) Max Pool layers

Max Pooling is a popular down-sampling method that helps reduce the resolution of the image [19][14]. This is important to avoid over-fitting, which is when the model fits the training set well but performs poorly on the test set, and to reduce the training time of the model. In this architecture, the model uses 3 MaxPool layers, one after each of the convolution layers, with a pool size of 2x2. Max Pooling works by taking the maximum value of the 2x2 area of the image and moving on to the next 2x2 area.

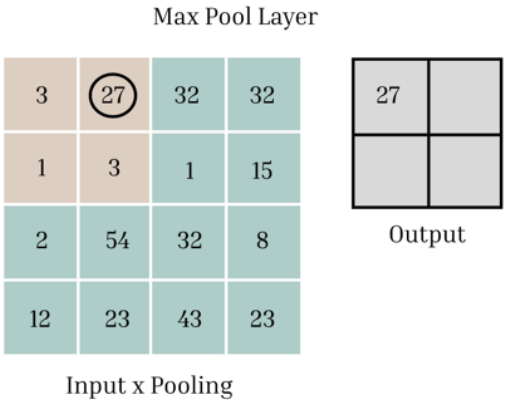

Figure 3. Visualization of a MaxPool layer

3) Flatten Layer

The flatten layer is a standard layer used near the end of all CNNs [14]. It converts the output of the CNN to a 1-dimensional vector so that this vector can be passed onto the final classification layers.

4) Dense Layers:

The dense layers, also known as the fully connected layers, perform the final classification on the feature maps extracted from the convolutional layers [14]. The final dense layer in the network is the final classification layer, which takes an integer as a parameter that defines the number of classes the model needs to predict. In this model, the final dense layer takes in 2 classes, as there are 2 classes: wildfires and no wildfires.

5) Dropout Layers:

Dropout layers are used to prevent overfitting [15]. Overfitting is an issue that occurs during the training stage of making a model. It occurs when the model has good metrics on the training data but can't achieve similar metrics on validation or test data. This means that the model is not generalized and is weak at predicting new images. To prevent this, the model uses dropout layers that disconnect the connection between a certain percentage of neurons in the fully connected layer. This model uses two dropout layers that disconnect 40% of the neurons in the first layer and 50% of the neurons in the second layer. By reducing the number of connections, the model becomes less complex, which allows it to generalize to new data better.

6) Activation Functions

Activation functions are used in CNNs to add non-linearity to the model. The relationship between labels and images isn't linear, so this is important to derive a proper relationship. There are many activation functions, but the most popular in CNNs is ReLU, which stands for Rectified Linear Unit [16], as shown in Equation 1. This function returns 0 if the input is negative and returns the same value if the input is positive (Figure 4).

$$ReLU\ Function: f(x) = max(0, x) \quad (1)$$

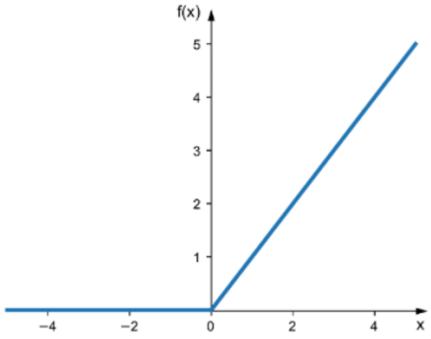

Figure 4. ReLU Function Visualization

This model uses a ReLU activation function in all the convolution and fully connected layers except in the last layer, where it uses a SoftMax function. A SoftMax function is a function usually used at the end of the CNN that calculates the output probabilities of the image passed [17], as shown in Equation 2. For example, if we pass a wildfire image, the function may predict a 95% chance of it belonging to the wildfire class and a 5% chance of it belonging to the no wildfire class.

$$SoftMax\ function: S(y)_i = \frac{exp(y_i)}{\sum_{j=1}^{n} exp(y_j)} \quad (2)$$

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

7) Optimizer

TensorFlow optimizers are predefined optimizers with set hyperparameters for model training. The most popular optimizer is Adam, which consists of the various necessary parameters required for proper training such as learning rate, weight decay, beta 1, beta 2, etc., which will be discussed later [18],[19].

8) Parameters

The first parameter that needs to be discussed is the learning rate. Learning rate has to do with gradient descent, which is a concept that is associated with reducing the loss metric of the model during training. During the initial iteration, the ML model sets its training parameters to a random value and computes the loss (how far the predicted value is from the actual value). It then changes the model parameters based on the learning rate to reduce the loss and continues doing this until it reaches the minimum possible loss. The learning rate is a parameter that tells us how much we change the parameters of the model at every iteration of training. During training, the learning rate is tuned to make sure the most optimal learning rate is chosen to ensure the training time is the least while the loss is also the least. In this case, the model was trained with a learning rate of 0.0001. Another parameter that the model had applied for its training was early stopping, which is a parameter that stopped the training if the loss didn't improve on the model for an n number of iterations. Besides the learning rate, the other parameter that was modified during the training was weight decay. Large weights in a model can cause overfitting, so by implementing weight decay, the weights of the model are reduced closer to 0 to prevent overfitting. During the training of this model, the weight decay was set to 10-5. Another hyperparameter that is changed is the batch size. The batch size is the number of images or data points that are run through the model at each iteration [20]. Batch size can be used to reduce training time when there are too many images in a dataset. During the training of this model, the batch size is 256. Apart from these two hyperparameters, the remaining hyperparameters used were the default hyperparameters of Adam [21]. The model was then trained for 50 epochs (iterations).

9) Final Metrics and Performance of the Model

The sake of comparison, I retrained this model without early stopping to keep the factors controlled between my model, which come later in the paper, and the model taken from Kaggle. The following were the metrics:

Testing accuracy: 0.9565
Testing Loss: 0.1412

It is visible that the loss of the model is quite low, and the accuracy is quite high, which means that it is an accurate model. Furthermore, the validation metrics are

like the training metrics, which means that the model is generalized and can predict new data presented to it well.

2.4 Data Processing and Visualization

After analyzing the model by Waleed Gul, my model creation began. Before creating the CNN architecture for training, the data needed to be cleaned and preprocessed. The higher the quality of the data used, the more accurate the prediction of the machine learning model is. The data was taken from Kaggle and then analyzed. A couple of the preprocessing steps were already taken by the creators of the dataset, as all the images had the same resolution (350 x 350) and file format (.jpg). Furthermore, the dataset was also divided into training, testing, and validation sets. A few more steps were taken to further increase the quality of the dataset.

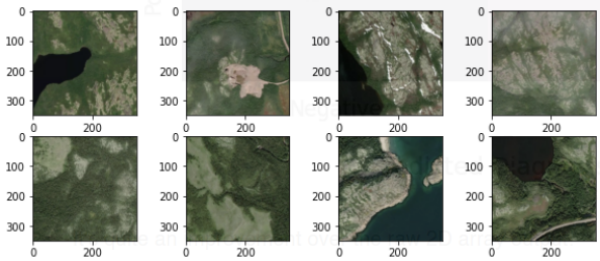

Figure 5. Sample Wildfire Images

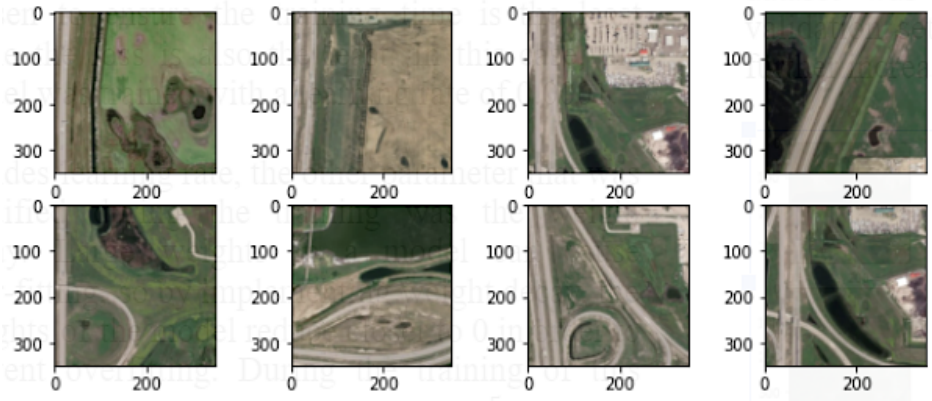

Figure 6. Sample No Wildfire Images

1) Checking for Duplicates

Eliminating duplicates is often a good idea, as duplicates can lead to poor generalization of models and slight overfitting. The approach to eliminating duplicates is to convert all the images into image matrices, which would have a shape of (350, 350, 3). Then, the list of image matrices is iterated through to see if

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

identical matrices exist in the dataset. A total of 32 duplicates are found in the dataset and are eliminated.

2) Observing and analyzing the distribution of file sizes across the dataset
To ensure that all the data in the dataset is similar, the distribution of file sizes is analyzed. Furthermore, we need to make sure that the data is well distributed instead of being skewed to reduce bias. First, the minimum, first quartile, median, third quartile, and maximum values are calculated for the file sizes. Using these values, a box and whiskers graph is plotted to visualize the distribution of data as shown in Figure 7.

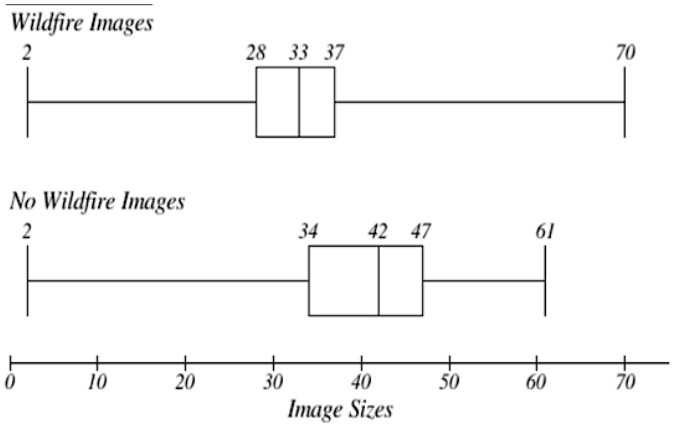

Figure 7. Box plot of unprocessed dataset

Then, the interquartile range of the file sizes is used to find the upper and lower bounds for identifying outliers. Approximately 400 outliers are eliminated from the wildfire image data set, and 600 outliers are eliminated from the non-wildfire image data set. Then, the mean is calculated to compare it to the median. For both classes, the median is similar to the mean, which means that the data is well distributed.

3) Viewing the dataset after pre-processing
After pre-processing, the training set had 22382 images in the wildfire class and 19561 images in the no wildfire class. There were approximately 3,000 more images in the wildfire class. Though both classes had a relatively similar number of images, there were more images in a wildfire, which meant the model would have more predictive power for the wildfire class. For this reason, the possibility of the model being biased toward the wildfire class was kept in mind (Figure 8).

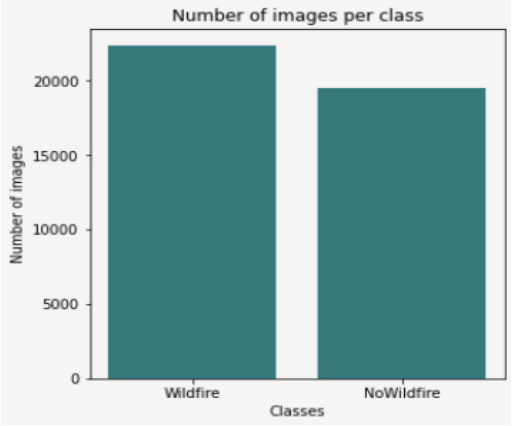

Figure 8. pre-processed data

2.5 Model Creation / CNN Architecture

After data processing, the model creation phase is started with a simple CNN structure: 1 convolutional layer, 1 max pool layer, 1 flattened layer, and 1 dense layer. The optimizer is set to Adam, and the base model (Figure 9) was tested for its metrics

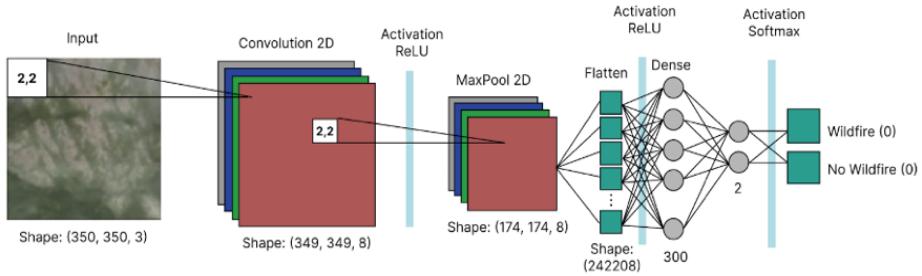

Figure 8. Optimizer

Then choose one parameter or layer to change in the architecture, for example, the activation function, and then record the metrics of the model with different activation functions until I find the best. I also experiment with various aspects of the convolutional neural network, such as the filter size, number of filters, number of layers, optimizer, etc., Finally see improvements in the metrics. Furthermore, to reduce training time and negate bias, only 10,000 images from each class are trained. The best architecture that I create is fairly similar to that of the model that was analyzed. To deal with overfitting I use various methods to prevent it, such as using dropout layers and adding an L2 regularizer to the model architecture [22]. The L2 regularizer works similarly to weight decay, where it makes the model simpler by reducing large weights to a value close to 0 but not 0. Smaller weights

help prevent overfitting. One dropout layer at the end of the model and experiment with different values. Eventually, they ended up getting the same combination of dropout layers as the analyzed model. This approach helps find the optimal architecture for a convolutional neural network as you change one variable at a time while keeping the rest constant. The final model architecture was quite like the analyzed model as shown in Table 2.

Table 2. Architecture of proposed model

Layer (type)	Output Shape	Params
Conv2D	(None, 349, 349, 8)	104
Max Pooling	(None, 174, 174, 8)	0
Conv2D	(None, 173, 173, 16)	528
Max Pooling	(None, 86, 86, 16)	0
Conv2D	(None, 85, 85, 32)	2080
Max Pooling	(None, 42, 42, 32)	0
Flatten	(None, 56448)	0
Dropout (0.1)	(None, 56448)	0
Dense	(None, 300)	16934700
Dropout (0.4)	(None, 300)	0
Dense	(None, 2)	602

Total parameters: 16,938,014
Trainable parameters: 16,938,014
Non-trainable parameters: 0

In this architecture, each convolution layer has a ReLU activation function, and the penultimate dense layer has a ReLU activation function. The last layer (the dense layer) has a SoftMax activation function to calculate the probability scores of the categories.

2.6 Hyperparameter Tuning

Learning rate: The learning rate is a hyperparameter that determines how big of a step the model's weights take when moving down the gradient descent. The

optimum learning rate, often referred to as the 'Goldilocks learning rate, is when the weights of the model converge to the lowest loss efficiently.

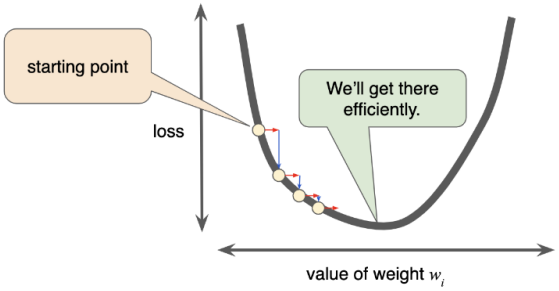

Figure 10. Visualization of gradient descent [23]

Beta 1 (β1) - β1 is a hyperparameter that refers to the first derivative of the gradient descent. So, the smaller the value of β1, the less steep the slope of the gradient descent is. β1 helps with smoother convergence in the loss curve. Like the approach for building the model architecture, I experiment with different optimizers to finally derive Adam as the best optimizer. Then I experiment with the other hyperparameters, such as learning rate, so that I can improve the model metrics. The Goldilocks learning rate for this model is 0.00001, as this learning rate steadily reduces the loss, allowing it to converge instead of shooting over the minimum loss value or taking too much time to converge. However, the loss curve doesn't converge smoothly even with this adjustment in learning rate. Even at this stage, there is slight overfitting visible in the model. Another issue is that the initial drops in loss are very large, which makes the gradient of the loss curve very high in the beginning. To deal with this, I am going to experiment with the beta 1 (β1) hyperparameter. I find that a β1 value of 0.8 is the best for ensuring the convergence of the loss curve is smooth and the validation data is converging similarly to the training data. Reducing the Beta 1 value helps the curves converge more smoothly but also increases the training time. After the hyperparameters are tuned, the final model is trained for 50 epochs to get the best possible metrics.

3. RESULTS AND DISCUSSION

3.1 Accuracy and Loss Graphs

The results of training a machine learning model over 50 epochs. In Figure 11, the accuracy graph illustrates that both the training and validation accuracies start at different levels but converge towards a high value close to 1.00 as training progresses. This suggests that the model becomes increasingly proficient at correctly predicting both the training data and unseen validation data. Figure 12 shows the loss over the same period, with both training and validation losses

starting at higher values and steadily decreasing to just below 0.05 by the end of the training. This reduction in loss indicates that the model is effectively minimizing errors in its predictions over time. The behavior of both graphs, with the validation metrics closely following the training metrics, suggests that the model is generalizing well to new data without significant overfitting.

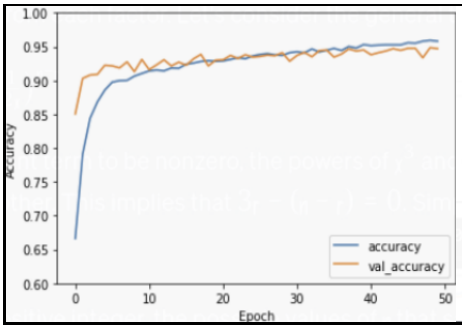

Figure 11. Accuracy / Epochs.

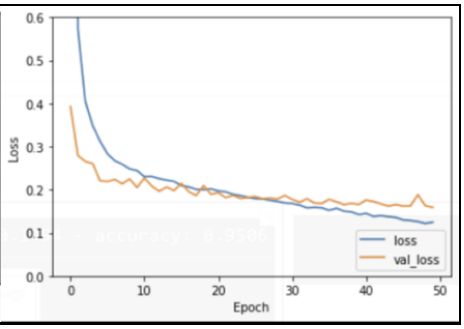

Figure 12. Loss / Epochs

3.2 Performance Metrics

The final model is evaluated using a testing set that contained approximately 6,000 images that the model had never seen before. After running the model on the test set, these were the following results:

Test Accuracy: 0.9506
Test Loss: 0.1494
Mean Absolute Error: 0.053

Then using the predictions conducted on the test set, a confusion matrix is shown in Figure13.

Figure 13. Confusion matrix of my model

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

A confusion matrix is a summary of the predictions made on the test set [24]. On the y-axis, the actual labels of the predictions are matched against the x-axis, the prediction labels from the model. The diagonal going from the top left to the bottom right shows the percentage of values that were predicted correctly. The Wildfire/Wildfire square and the No Wildfire/No Wildfire square show all the correctly predicted values. This confusion matrix (figure 13) shows that the model created is not only accurate, as there is a high percentage of 'true wildfire' and 'true no wildfire' prediction and a low percentage of 'false wildfire' and 'false no wildfire', but because the values are balanced with 95% correct prediction for both wildfire and no wildfire, the model is also unbiased. Using the confusion matrix, we can calculate 3 more metrics: Precision, Recall, and F1 score. Precision and Recall are metrics that help us compare the true-positive values with the false-positive and false-negative values. It is also used in the calculation of the F1 score which gives us an idea of how balanced our model is.

Precision: 0.948
Recall: 0.953
F1 Score: 0.950

Since precision and recall are similar and the F1 score is high, it means that the model is well-balanced and unbiased.

3.3 Comparison of my model with existing model

Both models have a very similar architecture with the same number of layers. The only difference in the architecture is that the model taken from Kaggle has a dropout layer combination of a 50% dropout in the first layer and 40% dropout in the second layer, whereas the model created by me has a dropout layer combination of 10% dropout in the first layer and 40% dropout in the second layer. This means that the model from Kaggle has reduced complexity as more neurons were disconnected.

Besides the architecture, the other differences between the two models are in the hyperparameters in training. While they both use the Adam optimizer, the Kaggle model uses a batch size of 256 to reduce its training time; however, for my model, I didn't include a specified batch size and instead let all the data run through the model during training. Furthermore, to make the loss converge, the Kaggle model reduced the weight decay to 10^{-5}; However, instead of changing the weight decay, I reduced the $\beta 1$ hyperparameter to 0.8. After training the models, the difference in loss and accuracy is almost negligible, as the difference in the metrics is in the third decimal place. The loss for both models is approximately 0.14 to 0.15, and the accuracy is approximately 0.95.

The major difference between the models is in the convergence of the loss graph (Figures 14 and 15). During training, the Kaggle model converges with less noise but is much steeper, whereas my model converges with a lower slope but more noise in the validation curve.

Figure 14. Loss / Epochs of existing model

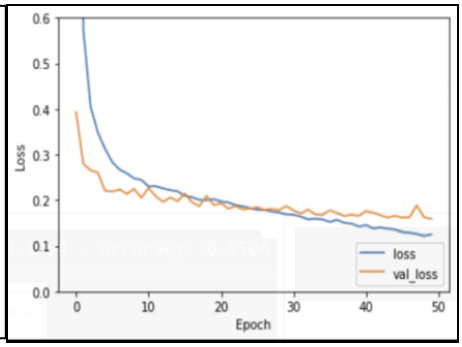

Figure 15. Loss / Epochs of my model

3.4 Analyzing the effectiveness of this model in real life.

To analyze the usefulness of the model, a test data set was created using Canadian satellite images from ArcGIS, an online geographic information software. [25] The model was then used to predict vulnerability to wildfires on the test set. Figures 16 and 17, are some predictions made by the model on random instances of the test set. The prediction of an area being prone to wildfires is represented by the number '0' and, the prediction of an area not being prone to wildfires is represented by the number '1'. In figure 16, some of the correct predictions made by the model can be observed. In figure 17 majority of the wrong predictions are areas prone to wildfires being predicted as not-prone meaning that the model still has room for improvement in making predictions for wildfires.

Figure 16. Correct Predictions on the ArcGIS Images

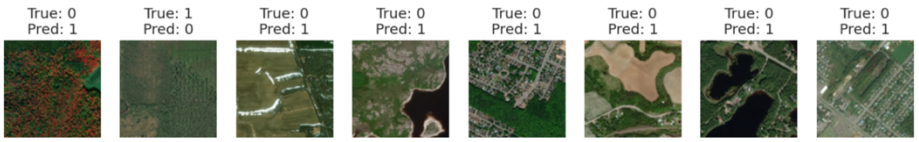

Figure 17. Incorrect Predictions on the ArcGIS Images

To look at the non-overlap in the data, which refers to areas in the test set which do not match the prediction for the vulnerability to wildfires, the Jaccard index was used, as shown in Equetion 2.

$$Jaccard\ Index: J(A, B) = \frac{|A \cup B|}{|A \cap B|} \quad (2)$$

The Jaccard Index provides the percentage of the predicted samples that do overlap with the actual samples. To quantify non-overlap, we can subtract the Jaccard index by 1:

$$Nonoverlap = 1\text{-}J(A, B) \quad (3)$$

Non–overlap results:

$$Jaccard\ Index = 0.89$$
$$Nonoverlap = 0.11$$

According to the Jaccard Index, 89% of the samples overlap whereas 11% have non-overlap. This makes the model's performance quite impressive as it has quite a high overlap percentage.

3.5 Synthesis and Analysis of The Results

The results show promising results as the model has high performance metrics with an F1 score of 0.95 and a Mean Absolute Error of 0.053. This shows that not only is the model extremely accurate, but it also performs equally well on both classes: wildfire and non-wildfire. Furthermore, testing the model on real life historical data of wildfires in Canada [25], showed that the model was able to generalize well even beyond the test dataset. By using the Jaccard Index, we found the model had a non-overlap of 0.11, which shows that 11% of the predicted values didn't match the actual values. The model can be improved further to reduce this nonoverlap value which would increase its performance even more. Looking at the performance metrics based on real-life applicability, predictive power, and bias in the classes, the model is strong for its application.

3.6 Applications of the Model in the real world

The CNN model can be used to automatically predict land that is at risk of wildfires and aid humans in marking areas susceptible to wildfires. Having this knowledge beforehand can help us prevent wildfires and reduce its negative consequences. Furthermore, our model allows the CNN to learn the features of the 2 classes creates a strong pre-trained model. This pre-trained can then be used as a base for fine-tuning for more specific use cases such as real-time monitoring

of wildfires from camera feed or drone-surveillance for wildfires. The pre-trained model will ensure high accuracy for a fine-tuned model and will also result in lower computational costs when training models for more specific use-cases in wildfire prevention. Another key application of the CNN model is its ability to create a system to update the land cover prone to wildfires overtime by running predictions routinely. As time passes, areas-prone to wildfires change due to urban expansion. Even so, with the CNN-based system in place, our knowledge of which areas are prone to fires also updates automatically. This also narrows down the areas which we need to pay attention to when looking out for wildfires.

4. CONCLUSION

Wildfires in Canada and all around the world have caused devastating damage to human health and the environment. For this reason, preventing wildfires is crucial. Using a convolutional neural network, I created an AI model that can take LULCC satellite imaging as input and output a prediction of whether that area is prone to wildfires or not. The convolutional neural network is useful as a deep learning model it has a low testing loss of 0.1494 and a high accuracy of 0.9506, which means that it can accurately predict whether land is susceptible to wildfires or not with a minimal chance of error. Furthermore, after plotting a confusion matrix and calculating the precision, recall, and F1 score for the model, the model proved to be unbiased and capable of making accurate predictions for both categories (wildfire and no wildfire). Training the CNN model on this data can help draw out distinctive features in the 2 classes that may be hard to identify from the naked eye such as the risk of wildfires in wildland-urban interface (WUI). Currently, the model only uses data from Canadian vegetation. To make the model better in terms of learned features, we can train the model with a much larger dataset from data all around the world, making it learn better features. Furthermore, the model was trained on clear images, but sometimes satellite imaging can be unclear due to factors such as fog, false color, blurring, etc. If the model were to receive such images as input, it wouldn't be able to predict as well, so to improve the model, we can use augmented and unclear images in the future. By harnessing the predictive power of CNN models, we can improve early detection, enhance resource allocation, and ultimately mitigate the impact of wildfires on both natural ecosystems and human communities.

REFERENCES

[1] L. M. Zavala, R. de Celis, and A. Jordán, "How wildfires affect soil properties. A brief review," *Cuad. Investig. Geogr.*, vol. 40, no. 2, pp. 311–332, 2014. doi: 10.18172/CIG.2522

[2] A. Nappi, P. Drapeau, and J. P. L. Savard, "Salvage logging after wildfire in the boreal forest: Is it becoming a hot issue for wildlife?" *Forestry Chron.*, vol. 80, no. 1, pp. 67–74, 2011. doi: 10.5558/TFC80067-1

[3] E. A. Keller, D. E. DeVecchio, and R. H. Blodgett, *Natural Hazards: Earth's Processes as Hazards, Disasters, and Catastrophes*, 2019. doi: 10.4324/9781315164298

[4] M. L. Childs et al., "Daily Local-Level Estimates of Ambient Wildfire Smoke PM2.5 for the Contiguous US," *Environ. Sci. Technol.*, vol. 56, no. 19, pp. 13607–13621, 2022. doi: 10.1021/ACS.EST.2C02934

[5] R. Nasi, R. Dennis, E. Meijaard, G. Applegate, and P. Moore, "Forest fire and biological diversity," *Unasylva-FAO*, pp. 36-40, 2002.

[6] T. N. Wasserman and S. E. Mueller, "Climate influences on future fire severity: a synthesis of climate-fire interactions and impacts on fire regimes, high-severity fire, and forests in the western United States," *Fire Ecol.*, vol. 19, no. 1, pp. 1–22, 2023. doi: 10.1186/S42408-023-00200-8

[7] S. F. Fonji and G. N. Taff, "Using satellite data to monitor land-use land-cover change in North-eastern Latvia," *SpringerPlus*, vol. 3, no. 1, pp. 1–15, 2014. doi: 10.1186/2193-1801-3-61

[8] N. Sisodiya, N. Dube, and P. Thakkar, "Next-Generation Artificial Intelligence Techniques for Satellite Data Processing," in *Remote Sens. Digit. Image Process.*, vol. 24, pp. 235–254, 2020. doi: 10.1007/978-3-030-24178-0_11

[9] N. R. Talukdar et al., "Forest fire estimation and risk prediction using multispectral satellite images: Case study," *Nat. Hazards Res.*, vol. 4, no. 2, pp. 304–319, 2024. doi: 10.1016/J.NHRES.2024.01.007

[10] L. Alzubaidi et al., "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, pp. 1–74, 2021. doi: 10.1186/S40537-021-00444-8

[1]] J. Wu, "Introduction to convolutional neural networks," *Natl. Key Lab. Nov. Softw. Technol., Nanjing Univ., China*, vol. 5, no. 23, p. 495, 2017.

[12] Y. O. Sayad, H. Mousannif, and H. Al Moatassime, "Predictive modeling of wildfires: A new dataset and machine learning approach," *Fire Saf. J.*, vol. 104, pp. 130-146, 2019.

[13] R. Kanwal, W. Rafaqat, M. Iqbal, and S. Weiguo, "Data-Driven Approaches for Wildfire Mapping and Prediction Assessment Using a Convolutional Neural Network (CNN)," *Remote Sens.*, vol. 15, no. 21, p. 5099, 2023.

[14] C. Nebauer, "Evaluation of convolutional neural networks for visual recognition," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 685-696, 1998.

[15] N. Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[16] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," *arXiv preprint arXiv:1803.08375,* 2018.

[17] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022. doi: 10.1016/j.neucom.2022.06.111

[18] D. Kinga and J. B. Adam, "A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, vol. 5, p. 6, May 2015.

[19] K. Ding, N. Xiao, and K.-C. Toh, "Adam-family Methods with Decoupled Weight Decay in Deep Learning," *arXiv preprint arXiv:2310.08858*, 2023.

[20] P. M. Radiuk, "Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets," *Inf. Technol. Manag. Sci.*, vol. 20, no. 1, 2018. doi: 10.1515/ITMS-2017-0003

[21] S. A. Mnati, F. I. Hussein, and A. Issa, "Development of an ANN Model for RGB Color Classification using the Dataset Extracted from a Fabricated Colorimeter," *Al-Khwarizmi Eng. J.*, vol. 19, no. 4, pp. 67-77, 2023.

[22] M. Yang et al., "Deep neural networks with L1 and L2 regularization for high dimensional corporate credit risk prediction," *Expert Syst. Appl.*, vol. 213, 118873, 2023. doi: 10.1016/J.ESWA.2022.118873

[23] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[24] T. Hernández-Del-Toro, F. Martínez-Santiago, and A. Montejo-Ráez, "Assessing classifier's performance," in *Biosignal Process. Classif. Using Comput. Learn. Intell.: Princ., Algorithms, Appl.*, pp. 131–149, 2021. doi: 10.1016/B978-0-12-820125-1.00018-X

[25] A. Rock and R. Malhoski, *Mapping with ArcGIS Pro: Design Accurate and User-Friendly Maps to Share the Story of Your Data*. Packt Publishing Ltd., 2018.