FireEye - A Deep Learning Approach

Spas Angelov

December 9, 2024

1. Introduction

The devastating impacts of climate change have become increasingly evident in recent years, significantly disrupting ecosystems, properties, and human lives. One particularly alarming consequence is the rise in the frequency and intensity of wildfires, which pose severe threats to local ecologies, economies, and public safety. To mitigate these risks, predictive tools that can accurately forecast wildfire behavior are critical for improving response strategies.

This project, named *FireEye*, aims to leverage satellite data to predict the daily progression of wildfires. By providing actionable insights, this tool can serve as an invaluable resource for incident responders and policymakers tasked with wildfire management.

Building on the groundwork laid in the CSCA 5622 Supervised Machine Learning project, this updated work is tailored for CSCA 5642 and focuses on implementing Convolutional Neural Networks (CNNs) for fire spread prediction. Specifically, the project utilizes deep learning techniques to generate next-day fire prediction maps from readily available satellite imagery. The objective is to evaluate and compare various CNN architectures to identify the most effective model for accurate wildfire prediction.

Additionally, this iteration seeks to address limitations identified in the original study, including challenges associated with the uncertainty mask. By refining these aspects, I aim to enhance the robustness and practical applicability of the predictive model.

2. Data

The data for my original supervised learning project was collected and analyzed using Google Earth Engine (GEE). While GEE provides a robust platform for accessing satellite imagery, this updated project leverages the dataset from the original paper[**1**], which is available on Kaggle as a collection of TensorFlow Record files. This approach was chosen to reduce computational overhead and increase data availability for training and evaluation.

The Kaggle dataset closely mirrors the methods used in my original data collection process via GEE. Due to these reasons, this paper will include the data collection process in the following section. While an initial attempt to use my own data was made, I was only able to extract around 300 valid samples in about a day of processing. The Kaggle dataset includes

around 18,000 samples which would have taken about 60 days to extract from GEE, which was not enough time given the time constraints of this project.

2.1. Pulling Data From Google Earth Engine

Google Earth Engine (GEE) provides access to *ImageCollections*, which represent groups of geospatial *Images*. Each *Image* contains spatially referenced data, often structured as a 2D array, where each pixel represents a specific geospatial measurement. An example of an *Image* is shown in Figure 2.1.

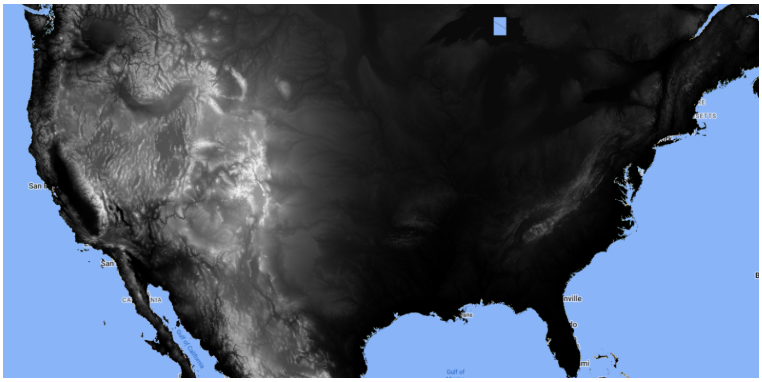

Figure 2.1: Elevation *Image* of the United States. Each pixel represents elevation in meters, with black indicating the lowest elevations and white indicating the highest, interpolated linearly to create a gradient.

For this project, several *Images* were retrieved from GEE, each offering critical data for wildfire prediction. The selected *Images* and their resolutions are listed below:

- Fire Bit Mask at 1 km resolution [**2**]
- Elevation at 90 m resolution [**3**]
- Weather at 4 km resolution [**4**]
- Drought at 4.5 km resolution [**5**]
- Vegetation at 0.5 km resolution [**6**]
- Population Density at 1 km resolution [**7**]

Images often contain multiple bands of information. For instance, the weather *Image* includes bands for precipitation, humidity, temperature, and more. Table 2.1 provides an overview of the selected features, their corresponding *Images*, and units.

The next-day fire mask is derived by accessing the fire mask *Image* one day into the future. Both the fire mask and next-day fire mask *Images* use a 2-bit mask, where:

- The presence of the first bit indicates uncertainty due to external factors, such as cloud coverage.
- The presence of the second bit signifies that the 1 km × 1 km region contains an active fire.

Feature	Unit	Image Source
fire_mask	2-bit mask	Fire [**2**]
fire_mask_next_day	2-bit mask	Fire [**2**]
elevation	meters	Elevation [**3**]
wind_direction	degrees	Weather [**4**]
wind_speed	meters/second	Weather [**4**]
energy_release_component	NFDRS index	Weather [**4**]
burn_index	NFDRS index	Weather [**4**]
precipitation	millimeters/day	Weather [**4**]
temperature_min	kelvin	Weather [**4**]
temperature_max	kelvin	Weather [**4**]
drought_index	Palmer Drought Severity Index	Drought [**5**]
vegetation	NDVI	Vegetation [**6**]
population_density	persons/km^2	Population Density [**7**]

Table 2.1: Selected Features and Their Units.

2.2. Data Cleaning & Preprocessing

The elevation dataset, derived from the SRTM dataset[**3**], is a static *Image*, meaning it does not vary over time. In contrast, other datasets, such as weather and fire data, have temporal components. For instance, weather and fire data are updated daily, vegetation data is captured every 16 days, and drought data is updated every 8 days.

To extract relevant data, *ImageCollections* were filtered to select the most recent *Image* for each timestamp.

Resolution Standardization: The datasets vary in spatial resolution. As in the original study, all data was standardized to a resolution of 1 km × 1 km. Higher-resolution data was downsampled by averaging values across grid cells, while lower-resolution data was upsampled using bicubic interpolation. The resulting data was exported as a series of 64 km × 64 km pixel patches, with each pixel representing a 1 km × 1 km area. The data was stored in TensorFlow Dataset format.

Figure 2.2 illustrates a 64 km × 64 km "patch" encompassing multiple features from the selected *Images*.

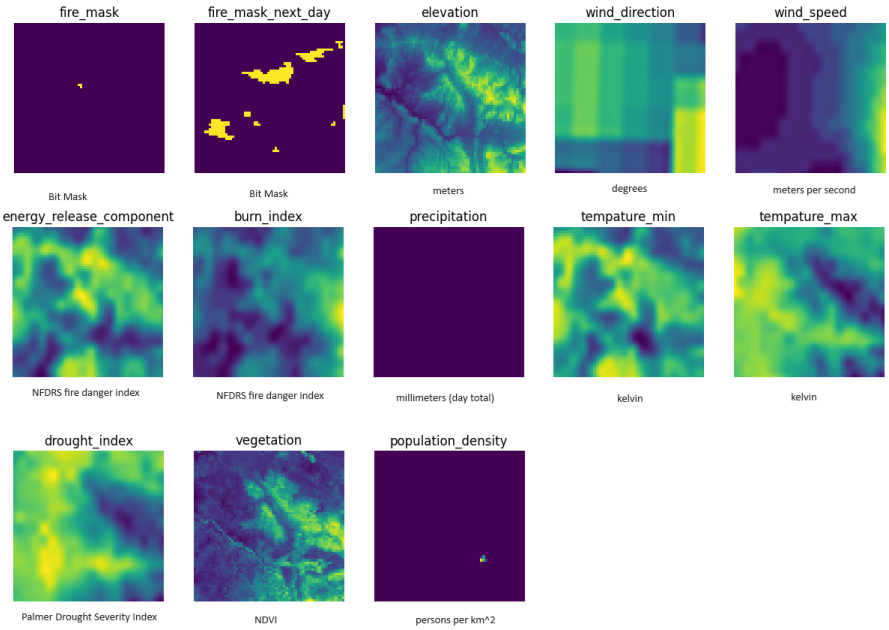

Figure 2.2: 64km x 64km patch over selected features

Fire Evolution Filtering: As collected, the data is not immediately usable for the project's goals. Wildfires typically ignite due to external factors, such as human activity, which are not captured in this dataset. Consequently, predicting the initiation of wildfires is beyond the scope of this study. Instead, the focus is on modeling fire evolution and spread. To ensure relevance, only patches where the fire mask feature contains at least one active fire pixel (1 km $\times$ 1 km) were retained.

After filtering, the dataset is well-suited for ingestion into convolutional neural networks, allowing the model to leverage spatial data effectively. This preprocessing step marks the divergence between this project and my previous supervised learning project.

Transition to Kaggle Dataset: Following preprocessing, the Kaggle dataset was adopted for training purposes. The Kaggle dataset is largely equivalent to the dataset prepared using Google Earth Engine, with two notable differences: burn_index feature is not included in the Kaggle data, while humidity is a new feature. Humidity is tracked in % units. The omission of the burn_index is not too prohibiting as this feature was highly correlated with the energy_release_component. As for the humidity, this feature was available as a channel on the weather image[**4**]. Other features remain identical, ensuring consistency between the datasets.

Figure 2.3 provides a few example patched from the Kaggle dataset.

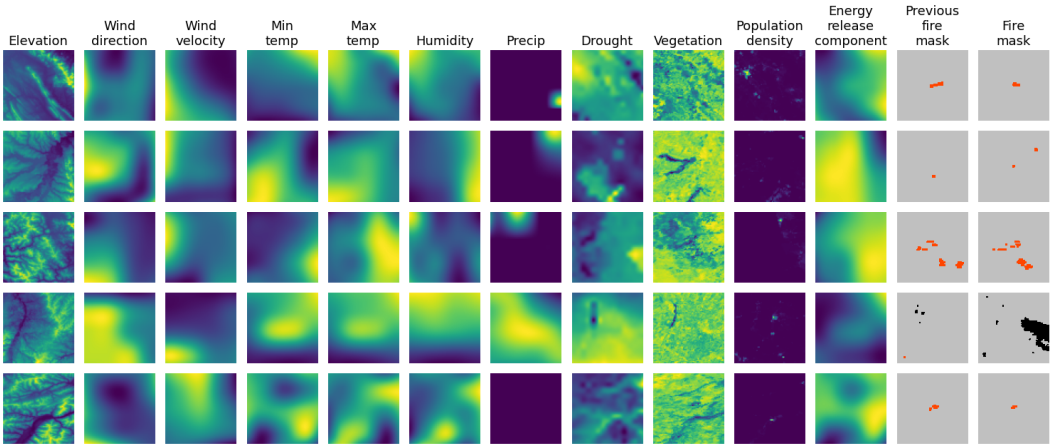

Figure 2.3: 64km x 64km Kaggle Training Dataset

Correlation between Features To ensure that the input features are not highly correlated, the correlation values were aggregated across the entire dataset. Figure 2.4 presents the correlation matrix, which aligns with the findings from the original supervised learning study. Most features exhibit low correlations, with a few notable exceptions:

- The minimum and maximum temperature features are strongly correlated.
- Elevation and the Energy Release Component (ERC) also show a notable correlation.

While correlations between input features can be a concern in traditional supervised learning models, they are less problematic in deep learning, especially when regularization techniques are applied. Techniques such as Dropout layers, which will be discussed in Section 3, help mitigate overfitting and enhance model generalization.

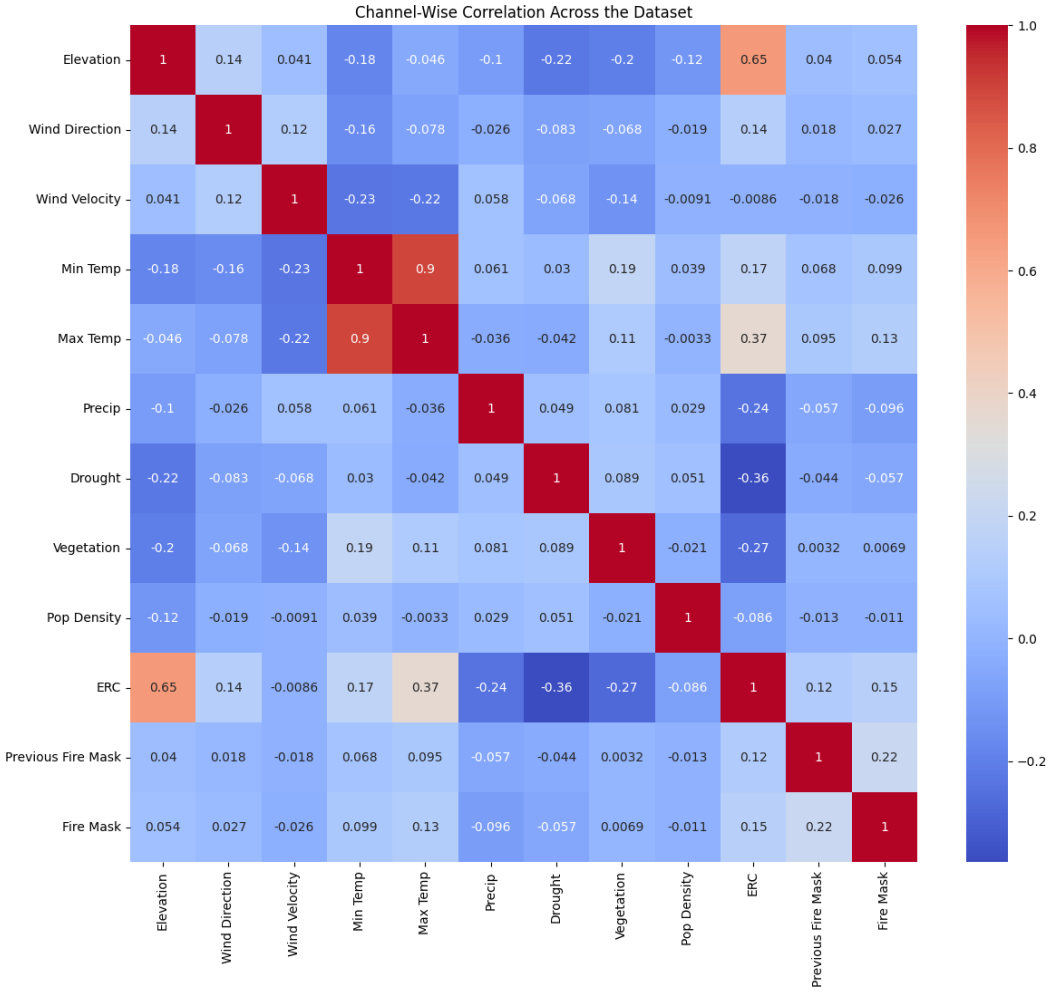

Figure 2.4: Correlation matrix of the dataset features. Strong correlations are observed between specific pairs of features, such as temperatures and elevation vs. ERC, while most features remain uncorrelated.

2.3. Further Exploratory Data Analysis

Our downloaded dataset contains 18545 input examples, which I will split into train, validation, and test datasets respectively. Because these inputs will be fed into a CNN, it is good practice to normalize all input features to roughly mean 0 and standard deviation 1. Table 2.2 gives a breakdown of each input feature mean, standard deviation, minimum and maximum. In some cases the minimum/maximums are given by a logical value, rather than taken directly from the dataset. In general these values are given by taking the 99% th percentile of each minimum and maximum rather than the true values. The minimum and maximum values computed will be used to clip the dataset.

Feature	**Min**	**Max**	**Mean**	**STD**
elevation	0.0	3141.0	657.3003	649.0147
wind_direction	0.0	360.0	190.32976	72.59854
wind_speed	0.0	10.0243	3.8501	1.4109
humidity	0.0	1.0	0.0071	0.0042
energy_release_component	0.0	106.2489	37.3262	20.8460
precipitation	0.0	44.5303	1.7398	4.4828
temperature_min	253.15	298.9489	281.0876	8.9823
temperature_max	253.15	315.0922	295.1738	9.8154
vegetation	-9821.0	9996.0	5157.625	2466.6677
population_density	0.0	2534.0629	25.531384	154.72331

Table 2.2: Feature Distribution Breakdown

Input data not included in Table 2.2 will not be normalized prior to passing it to the model. For example, the fire masks shouldn't be altered, since that layer is essentially a bit indicating if fire is present or not.

2.4. Data Augmentation and Splits

To enhance the diversity of the dataset, I will implement **random cropping** as the primary data augmentation technique. This process will randomly crop the 64×64-pixel input images down to 32×32 pixels. Random cropping serves two key purposes:

1. Reducing the input size, which facilitates faster training.
2. Generating augmented samples where no fires may be present in the current fire mask, improving the model's robustness to such scenarios.

Figure 2.5 illustrates several examples of randomly cropped input images derived from the original dataset.

For the dataset splits, I will use a ratio of **80% for training**, **10% for validation**, and **10% for testing**. This ensures a sufficient amount of data for training while reserving meaningful subsets for validation and testing. With a batch size of 32, this setup prepares the dataset for training the model efficiently.

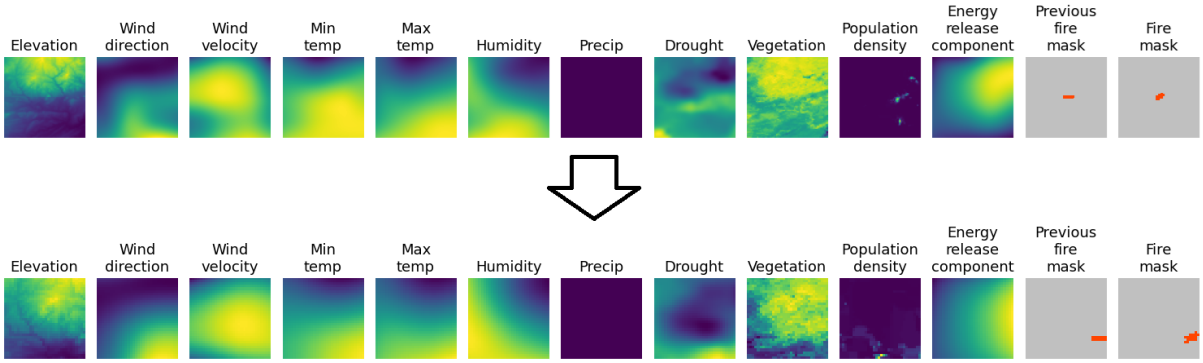

Figure 2.5: Random Crop Example - 64x64 → 32x32

3. Models

This paper explores two distinct architectures based on the auto-encoder framework. Auto-encoders are particularly advantageous for image processing tasks as they learn a latent, compressed representation of the data. By focusing on the most salient features rather than raw pixel values, the encoder effectively reduces dimensionality. The decoder then utilizes this latent representation to reconstruct or generate the output image.

3.1. Residual Model

3.1.1. Architecture

The residual model architecture follows the implementation described in the original paper. It consists of a convolutional encoder to extract features, a series of residual blocks for robust representation learning, and a convolution-transpose decoder to reconstruct the output image. Figure 3.1 shows the model block architecture. Every Conv2D and Conv2DTranspose block has padding = 'same'. Conv2Ds not inside the Residual blocks have strides = 1. Inside the Residual block, the central and skip convolution connections have strides = 2. Because of this the residual blocks reduce the input size by half regardless of if the mode is set to 'maxpool' or 'conv2d'. The Conv2DTranspose blocks have strides = 4, so that the upscaling resolution is enough to overcome subsequent Residual blocks. The final Conv2D block has 1 filter so we get the desired output channels of 1, and has a sigmoid activation to put all values between [0, 1].

3.1.2. Hyperparameter Tuning

Filter sizes, Filter count, and Dropout rates are treated as hyperparameters and will be tuned using the Keras Tuner library. In addition to these 3, I will also test a few different learning rates as well. Table 3.1 shows the different tuning values that the tuner used.

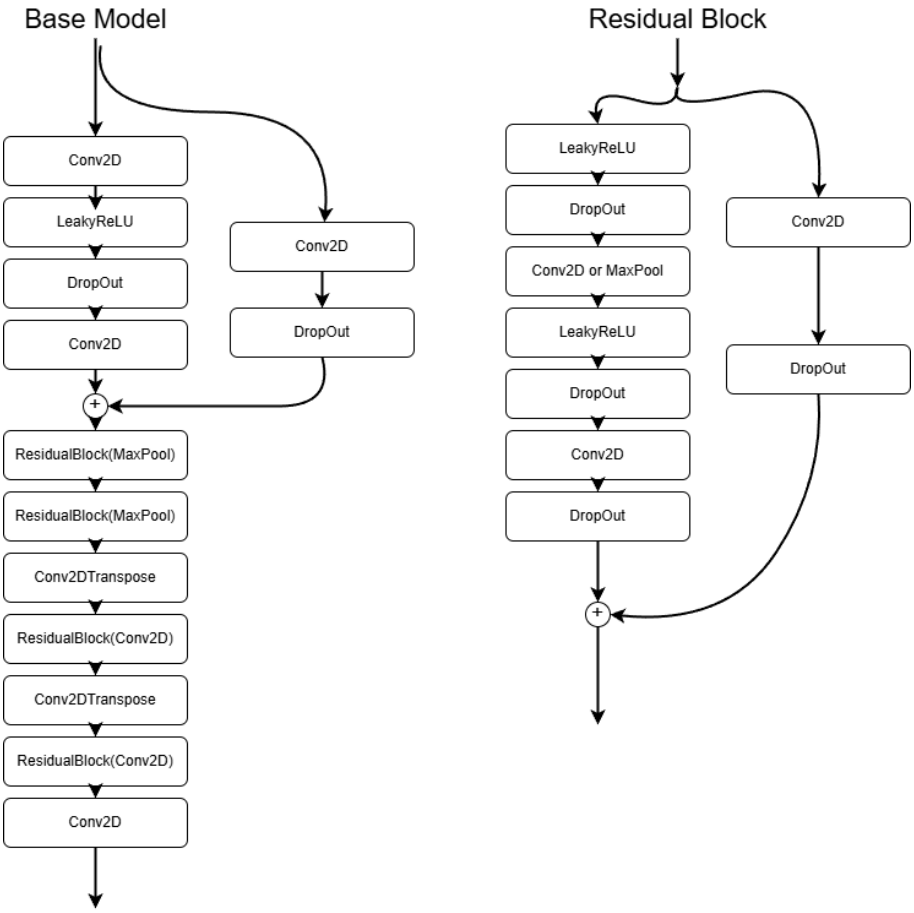

Figure 3.1: Base Model Architecture

Table 3.1: Hyperparameter Tuning Values

Hyperparameter	**Values Tested**
Filter Sizes	{3, 5}
Filter Count	{16, 32, 64}
Dropout Rates	{0.1, 0.2, 0.3, 0.4, 0.5}
Learning Rates	{1e-3, 1e-4, 1e-5, 1e-6}

I ran tuning for a maximum of 50 epochs, with a tuning factor of 5. The tuner ran through the different models using Hyperband tuning, which is a combination of random search and early stopping. Models were evaluated on a modified binary cross entropy loss function computed on the validation set. The specifics of the loss function will be discussed in section 3.3.

Table 3.2 shows the results of the top 5 models, which all perform fairly similarly on achieving the lowest loss:

Table 3.2: Top 5 Residual Models Based on Lowest Loss

Rank	Learning Rate	Dropout Rate	Num Filters	Kernel Size	Validation Loss
1	1e-4	0.3	64	5	0.219
2	1e-3	0.1	32	5	0.221
3	1e-3	0.1	16	5	0.223
4	1e-3	0.2	32	5	0.226
5	1e-4	0.3	64	5	0.239

The original paper also conducted hyperparameter tuning, but they employed Grid-Search. This searching method, although more accurate, takes a while to evaluate each model. Their optimal hyperparameters ended up being a kernel size = 32, size = 3, dropout = .1, and a learning rate of 1e-3.

3.2. Inception Model

3.2.1. Architecture

The Inception model is based on Inception blocks, which were first introduced with Google's paper "Going Deeper with Convolutions"[**8**]. There are a couple of advantages when using Inception blocks in a network architecture meant for images. Inception blocks are composed of multiple Conv2D parallel layers which have a range of filter sizes. This allows the inception block to extract features with different granularities. The multiple paths that inputs flow through also gives stability to the gradients during back-propagation, allowing for even deeper networks than was possible before. This project employs the use of Inception Blocks in between the encoder and the decoder portions of the auto-encoder. See Figure 3.2 for the architecture diagram. For the encoder and decoder portions the Conv2D and Conv2D transpose blocks all have strides = 2 effectively scaling the image resolution down and then back up. The Inception blocks keep the resolution of the inputs on output, but can have a variable filter size, and thus can have varying output channels.

3.2.2. Hyperparameter Tuning

The procedure for hyperparameter tuning is much the same as for the Residual models. I test values as seen in Table 3.1 with hyperband tuning. All Conv2D blocks outside of the

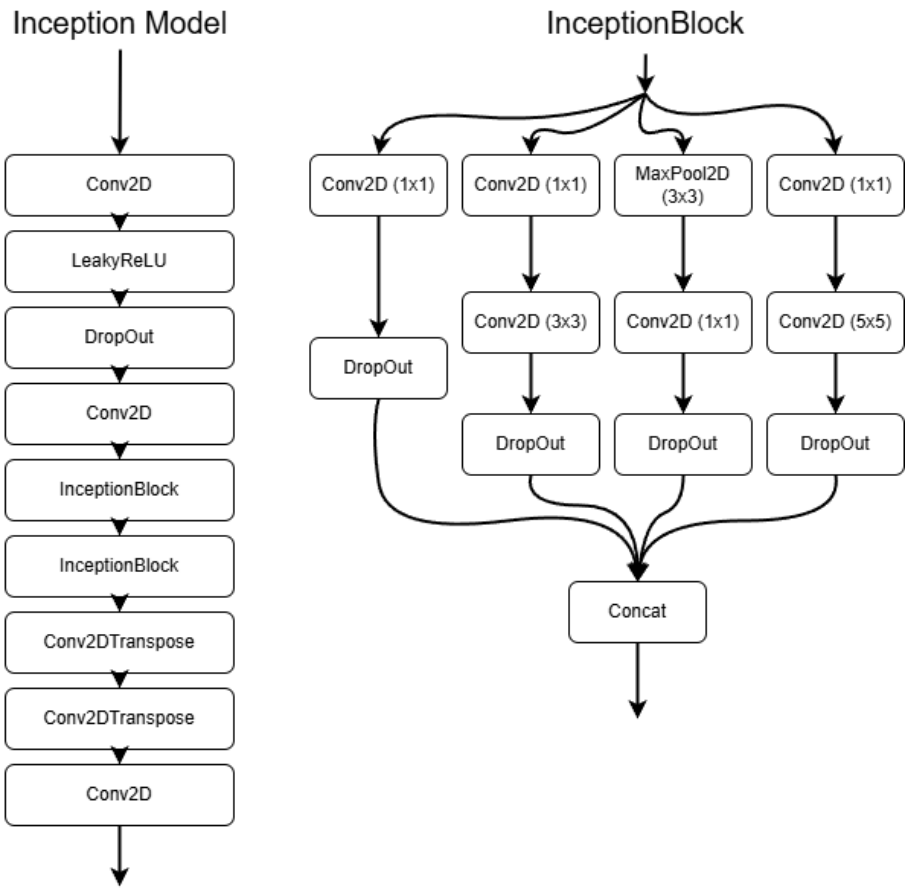

Figure 3.2: Inception Model Architecture

Inception blocks have their kernel size altered. Table 3.3 shows the results of the tuning, ranked by the best validation loss achieved.

Table 3.3: Top 5 Inception Models Based on Lowest Loss

Rank	**Learning Rate**	**Dropout Rate**	**Num Filters**	**Kernel Size**	**Validation Loss**
1	1e-3	0.1	64	5	0.211
2	1e-3	0.3	32	3	0.214
3	1e-3	0.1	64	5	0.216
4	1e-3	0.2	64	5	0.217
5	1e-3	0.1	64	5	0.222

Overall this model performs ever so slightly better than the Residual models over the training period.

3.3. Further Training

Taking both top models of each the Residual Networks and Inception Network further training can be done to reach their highest predictive power. I employ early stopping to prevent over-fitting to the training dataset, and let the models run for up to 100 more epochs. As seen in Fig 3.3, both models stop fairly quickly and don't see further improvements in the Validation Loss. It seems like the Residual Model has more of a tendency to over-fit.

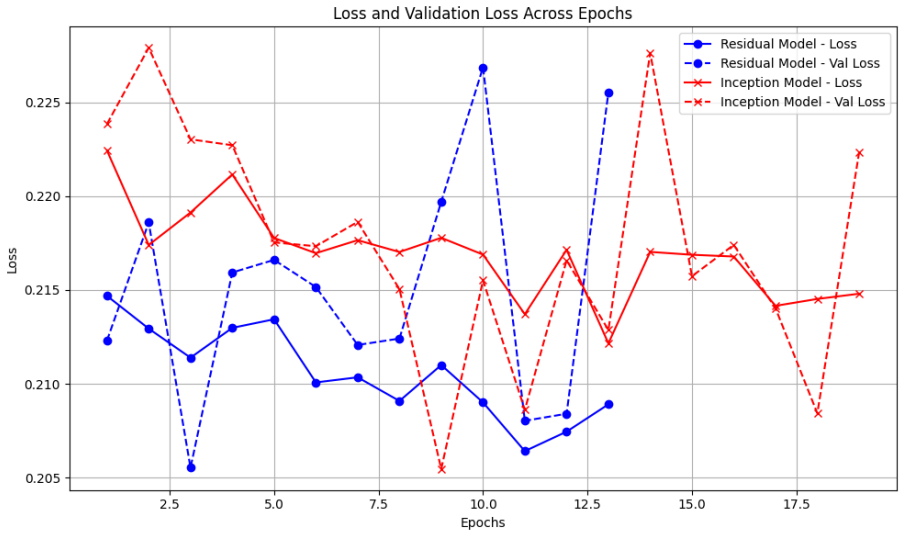

Figure 3.3: Inception and Residual Top Models Further Training

3.4. Modified Binary Cross Entropy Loss

3.4.1. Uncertainty in Observation

Due to uncertainty in the next day fire mask from the satellite imagery, special care must be taken when evaluating loss. The models should not be penalized for guessing either fire presence or not when there is uncertainty in the observation. The binary cross entropy loss function must be modified to account for these uncertain cases. The methodology used is to create a mask of uncertain values, and to element wise multiply this mask to both the predicted value and ground truth value prior to computing the loss.

3.4.2. Class Imbalance

In most of our input images, there is some presence of fire pixels, but these pixels are heavily outweighed by non-fire pixels. To account for this in the loss calculation a weight is applied

to the binary cross entropy after computation, but prior to averaging. Each fire pixel is given 4 times the importance of non-fire pixels. If this procedure isn't done, the models highly prefers to label most pixels as non-fire ones for the output. This weighting is taken from the original paper.

4. Results

Finally, let's take a look at how both models stack up in performance. Figures 4.1 and 4.2 shows the input fire mask, as well as the predictions from the Residual and Inception Networks, as well as the actual Ground truth. I kept the outputs of the Residual and Inception networks as the smooth sigmoid activations, to show more of a probability distribution rather than true predictions. Green represents the uncertainty mask, whereas the black to white is a linear scaling of the outputs from [0, 1]

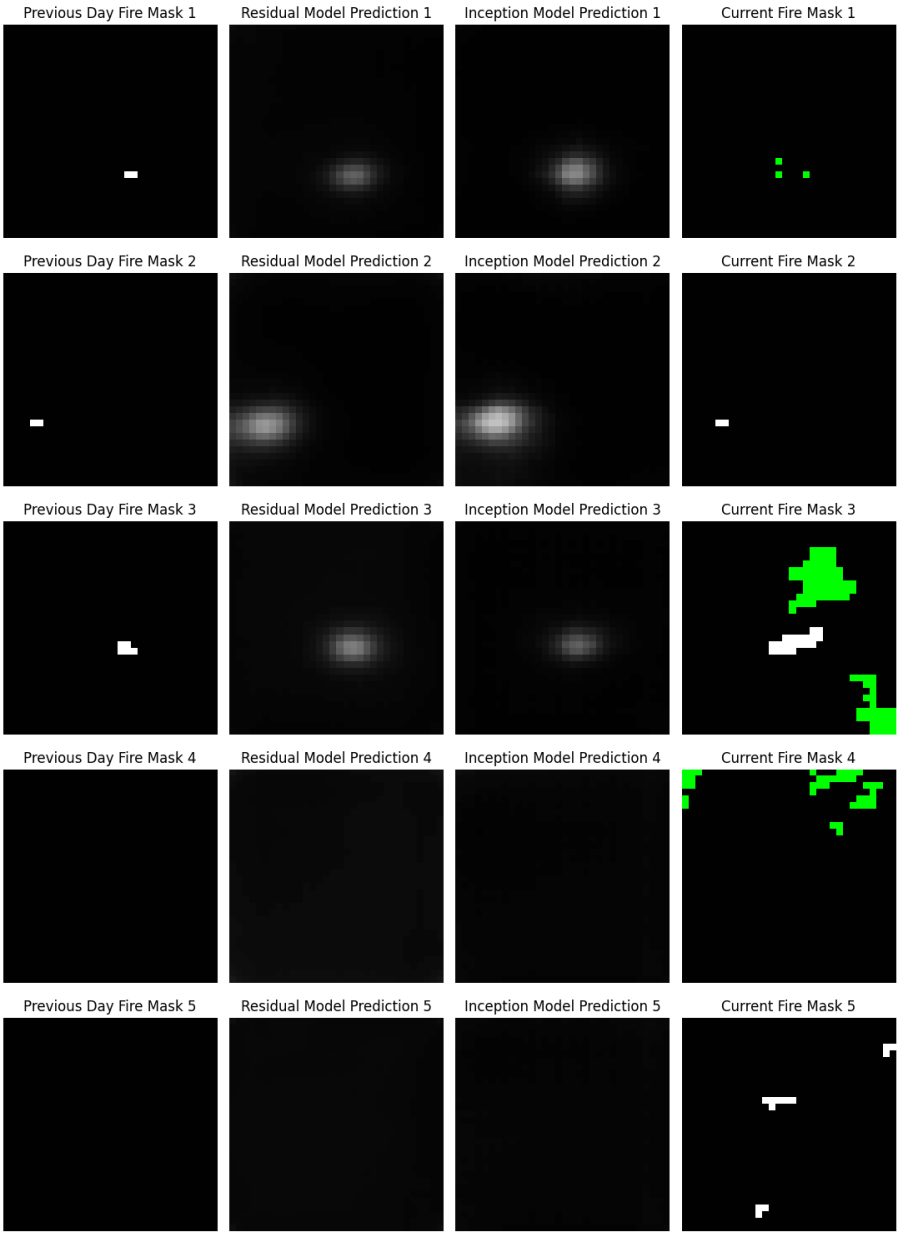

Figure 4.1: Results

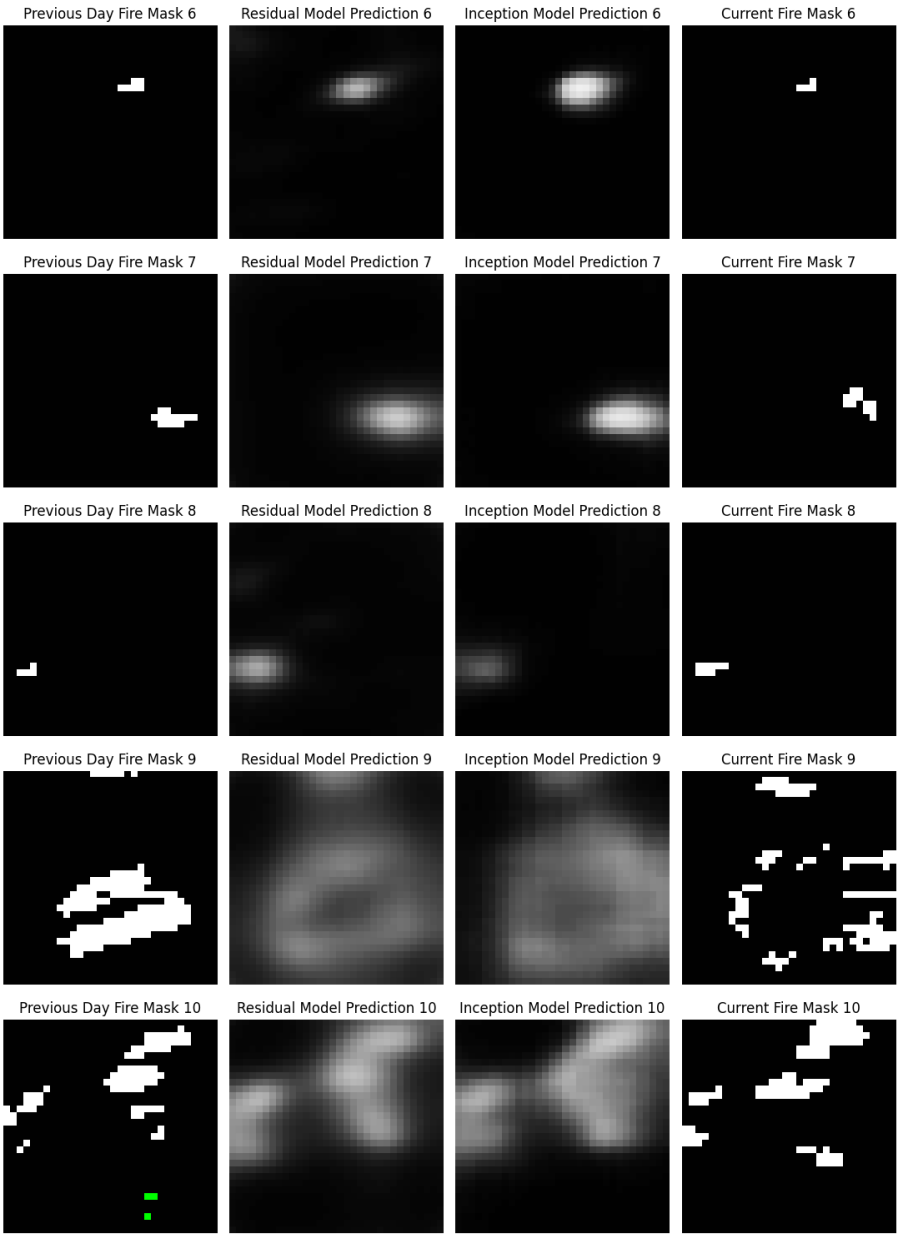

Figure 4.2: Results Continued

Overall, both models perform decently. The results were a random selection of the the test dataset, and are a good representation of how the model performs in some varied scenarios. One area the model fails at is predicting next day fires when no fire is present on the first day. This can be seen with example 5, but since fires are a spontaneous event, this prediction of no fire is what we would expect to see. Example 1 shows how the models can both fail in certain situations. In this case, the models predict the fire will die down, when it doesn't. Example 9 is a good example of predicting how fires will spread into a larger area, and is a good representation of the power of these models.

5. Conclusion & Further Improvements

Deep learning models offer a computationally efficient approach to simulating complex phenomena such as wildfire spread. These models can be applied across large geographic areas to predict fire growth and provide early warnings for regions with elevated fire risk. While they may not match the predictive accuracy of physics-based simulations, they can serve as a valuable tool for resource allocation and risk mitigation.

To further enhance the performance and realism of these models, several improvements are suggested:

- **Incorporating Physics-Informed Loss Functions:** Integrating physical constraints into the loss function can lead to more realistic predictions. This approach, known as physics-informed machine learning (PIML), enables models to align more closely with real-world dynamics, enhancing their predictive reliability.
- **Refined Data Filtering:** Certain examples in the dataset, such as spontaneous fire events (e.g., Example 5 in the results section), may penalize the model unfairly. Removing such outliers during preprocessing could improve overall model performance by focusing the training on more predictable scenarios.
- **Improved Padding in Convolutional Layers:** The current use of padding in Conv2D layers may be suboptimal, particularly at image edges where valuable spatial information is present. Transitioning to a "valid" padding strategy could potentially enhance edge predictions by leveraging real data rather than introducing artificial boundary effects.

These recommendations aim to refine the current methodology and address its limitations. By implementing these enhancements, future models may achieve greater accuracy and utility in wildfire prediction.

References

[1] F. Huot, R. L. Hu, N. Goyal, T. Sankar, M. Ihme, and Y.-F. Chen, "Next Day Wildfire Spread: A Machine Learning Data Set to Predict Wildfire Spreading from Remote-Sensing Data", arXiv preprint, 2021.

[2] Giglio, L., Justice, C. (2021). MODIS/Terra Thermal Anomalies/Fire Daily L3 Global 1km SIN Grid V061 [Data set]. NASA EOSDIS Land Processes Distributed Active Archive Center. Accessed 2024-06-21 from https://doi.org/10.5067/MODIS/MOD14A1.061

[3] Jarvis, A., H.I. Reuter, A. Nelson, E. Guevara. 2008. Hole-filled SRTM for the globe Version 4, available from the CGIAR-CSI SRTM 90m Database: https://srtm.csi.cgiar.org.

[4] Abatzoglou J. T., Development of gridded surface meteorological data for ecological applications and modelling, International Journal of Climatology. (2012) doi:10.1002/joc.3413

[5] Abatzoglou J. T., Development of gridded surface meteorological data for ecological applications and modelling, International Journal of Climatology. (2012) doi:10.1002/joc.3413

[6] Didan, K., Barreto, A. (2018). VIIRS/NPP Vegetation Indices 16-Day L3 Global 500m SIN Grid V001 [Data set]. NASA EOSDIS Land Processes Distributed Active Archive Center. Accessed 2024-06-21 from https://doi.org/10.5067/VIIRS/VNP13A1.001

[7] Center for International Earth Science Information Network - CIESIN - Columbia University. 2018. Gridded Population of the World, Version 4 (GPWv4): Population Density, Revision 11. Palisades, NY: NASA Socioeconomic Data and Applications Center (SEDAC). https://doi.org/10.7927/H49C6VHW. Accessed 2024-06-21

[8] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1–9.