Fire Severity Classification Using Neural Networks

Group Member: Quan Nguyen, Phunsok Norboo, Yu Xu, Huawan Zhong

Introduction & Motivation

Fires are among the most destructive events that demand rapid responses to minimize damage and save lives. While early fire detection is critical, understanding the severity of a fire is equally important to determine the appropriate emergency response. A minor fire demands a very different reaction compared to a large-scale, fast-spreading blaze. Most traditional fire detection models focus only on identifying whether fire is present, offering little insight into the urgency of the situation. This project addresses that gap by building a model that classifies fire images into three severity levels: small, medium, and large. By doing so, it aims to provide richer, more actionable information for firefighting and disaster response. Explainability methods, LIME and SHAP, are also included to make the model's predictions more transparent. These techniques help verify which parts of the input images influenced the model's decisions, ensuring that the model focuses on meaningful visual features like flames and smoke rather than irrelevant background noise. In doing so, it advances traditional fire detection toward a more practical, reliable, and real-world deployable solution.

Dataset and Preprocessing

For our fire classification project, we utilized the DFS Fire and Smoke Dataset, available on GitHub. This dataset originally consisted of 9,054 annotated images categorized into four classes:

- Large Fire
- Medium Fire
- Small Fire
- Other

Cleaning and Refinement

During data exploration, we identified a major issue: all images in the "Other" class included a clearly visible red bounding box labeled "other", making the classification task trivially easy for the model through text recognition rather than visual understanding. To eliminate this unintended bias, we removed all 'Other' class images, reducing the dataset to 7,305 images.

These remaining images were balanced across:

- Large Fire
- Medium Fire
- Small Fire

Preprocessing Workflow

To prepare the dataset for training, we implemented the following key preprocessing steps:

1. Label Handling and File Mapping:
    - Image file paths were updated to point to their correct locations on Google Cloud Storage (GCS).
    - Labels were encoded using LabelEncoder to convert class names into numerical targets.
2. Train-Validation Split:

- We used stratified splitting (via train_test_split) to ensure each class was proportionally represented in both the training and validation sets.
- The dataset was split 70% training / 30% validation.

3. Image Processing:
    - Images were decoded, resized to 224×224, and normalized to the [0, 1] range.
    - TensorFlow's tf.data API was used for efficient, parallelized dataset creation and prefetching.
4. Data Augmentation *(Training Set Only)*:
    - Applied random transformations to help the model generalize better:
        - Horizontal flips
        - Random brightness
        - Random contrast
    - Augmentation was separated from base preprocessing to ensure that only the training set received transformations.
5. Handling Class Imbalance:
    - To address the class imbalance present in the dataset — where medium and large fire images significantly outnumbered small fire images — we applied a class weighting strategy during model training. Specifically, we assigned a weight of 0.595 to the medium fire class, 0.836 to the large fire class, and a substantially higher weight of 8.05 to the small fire class. This approach was intended to compensate for the underrepresentation of small fire images, helping the model to treat each class more equally and avoid bias toward the more frequent categories.

Methodology

We developed and evaluated five deep learning models to classify fire severity into three categories: small, medium, and large. These models ranged from custom CNN architectures to transformer-enhanced and pretrained networks. Each model was trained using the same data preprocessing and training pipeline to ensure a fair comparison.

Training Setup (applied to all models except Pretrained MobileNetV2)

- Input shape: (224, 224, 3)
- Loss function: Sparse Categorical Crossentropy
- Optimizer: Adam
- Evaluation metric: Accuracy
- Batch size: 32
- Epochs: 20–30 (early stopping applied if needed)

Model 1: Custom CNN

The first model was a straightforward CNN built from scratch. It consisted of two convolutional blocks, each including:

- A Conv2D layer with ReLU activation
- Kernel.regulizer L2 (only on the second Conv2D layer)
- Batch Normalization
- Max Pooling
- Dropout for regularization

To successfully connect Conv2D layers and the Dense layer, there is a Flatten layer between the two parts.

This was followed by:

- A fully connected Dense layer (ReLU)
- Batch Normalization
- Dropout
- A final Dense layer with **softmax activation** for 3-class classification

The model had approximately **3M trainable parameters** and served as a strong performance baseline. It trained quickly and performed well on simple fire patterns but struggled with complex scenes involving large or diffused fires.

Model 2: CNN + Attention

This model builds on the baseline CNN by incorporating an attention mechanism to help the model focus on important spatial regions in the image.

Convolutional Feature Extractor:

- Three Conv2D blocks with ReLU activation
- L2 regularization applied to deeper convolutional layers
- Batch Normalization after each Conv2D
- MaxPooling is applied after Block 1 and Block 2
- Block 3 does not include MaxPooling to preserve high spatial resolution for the attention layer

- Dropout layers added to prevent overfitting

Attention Module:

- Multi-head self-attention layer applied after the final convolutional output
- Enables the model to capture spatial dependencies and focus on fire-related features (e.g., flames, smoke)
- An additional Conv2D layer is placed after the attention module to further process focused regions

Residual Module:

- Residual connection added around the attention block to retain original spatial features and support gradient flow
- Reshape operations used before and after attention to maintain input-output compatibility for Conv2D layers

Classifier Head:

- Flatten layer connecting convolutional features to dense layers
- Dense layer with ReLU activation
- Batch Normalization and Dropout
- Final Dense layer with softmax activation for 3-class fire severity classification

Training Setup:

- Trained for 22 epochs using EarlyStopping, monitored by validation loss
- Training accuracy reached approximately 70.6%

- Validation accuracy peaked at 73.35% (epoch 17)
- Best validation loss: 0.6592
- Training and validation curves showed good convergence, with minor fluctuations likely due to harder or more variable images

The attention-augmented CNN showed clear improvement over the baseline, especially in visually complex scenes. This model served as the foundation for further enhancement using Transformer blocks in Model 3.

Model 3: CNN + Attention + Transformer

This model further enhances the attention-based CNN architecture by introducing a Transformer block to improve long-range spatial feature learning.

Convolutional Feature Extractor:

- Three Conv2D blocks with ReLU activation
- L2 regularization applied to deeper convolutional layers
- Batch Normalization after each Conv2D
- MaxPooling is applied after Block 1 and Block 2
- Block 3 does not use MaxPooling to retain high spatial resolution for attention and Transformer layers
- Dropout layers are included to prevent overfitting

Transformer Module:

- Multi-head self-attention layer applied after the final Conv2D block

- Helps the model attend to spatially important regions across multiple perspectives
- The transformer block is added as a substitute for the attention model in Model 2
- Includes another multi-head attention mechanism
- Followed by a feed-forward network with two Dense layers applied independently to each position
- Residual connection around the Transformer block to preserve earlier learned features and stabilize training
- Reshape operations performed before and after the attention and Transformer blocks to ensure compatibility with Conv2D layers

Classifier Head:

- Flatten layer connecting high-level features to dense layers
- Dense layer with ReLU activation
- Batch Normalization and Dropout
- Final Dense layer with softmax activation for 3-class classification

Training Setup:

- Trained for 25 epochs using EarlyStopping and ReduceLROnPlateau to optimize learning and prevent overfitting
- Training accuracy reached approximately 71%
- Validation accuracy peaked at 73.5%

- Final validation loss reduced to approximately 0.66
- Training and validation curves showed stable convergence, with minor fluctuations attributed to more difficult or noisy samples

This model achieved the strongest performance among the three, demonstrating better generalization and robustness in challenging fire image conditions. The addition of the Transformer block allowed the network to capture both local and global context more effectively.

Model 4: Pretrained ResNet50

We experimented with transfer learning by fine-tuning a pretrained ResNet50 model. The original top layer was removed and replaced with:

- A Functional Layer
- A Global Average Pooling layer
- Batch Normalization
- A Dense layer (ReLU)
- Dropout
- A final Dense softmax layer for a 3-class output

Only the final layers were initially trainable, and then the full model was unfrozen for fine-tuning. While it performed well, its accuracy remained slightly below our custom attention-Transformer architecture.

Model 5: Pretrained MobileNetV2

Lastly, we tested MobileNetV2, a lightweight pretrained model. Similar to ResNet50, we replaced the top classifier with our own layers and fine-tuned the model. Despite being more efficient and faster to train, MobileNetV2 delivered lower accuracy compared to the other models, especially on medium and large fire classes. Its strength was model size and inference speed, making it better suited for mobile or edge deployment, but not our final choice.

Explainability Techniques

To better understand and validate our model’s predictions, we applied two post-hoc interpretability methods: SHAP and LIME. These techniques allowed us to visualize which parts of each image most influenced the model’s classification decisions.

SHAP

We initially used SHAP (SHapley Additive Explanations) to assess pixel-wise feature importance through a masking-based approach. The goal was to identify regions that contributed most to each predicted severity class. While SHAP produced some visual explanations, the results were often noisy and lacked clarity in highlighting semantically meaningful features like flames or smoke. Due to these limitations, SHAP was not selected as the primary interpretability method.

LIME

We then applied LIME (Local Interpretable Model-Agnostic Explanations), which perturbs localized image regions and observes the impact on model predictions. LIME generated much clearer and more intuitive heatmaps, successfully highlighting fire-relevant features such as active flame zones and smoke plumes. These visualizations aligned with human expectations and offered valuable insight into the model's decision-making process. As a result, LIME was used as the primary tool for model interpretation and validation.

Model Performance

Model 1: Custom CNN Model

The first model was a custom-built convolutional neural network consisting of two convolutional blocks followed by dense layers. It was trained with early stopping to prevent overfitting.

- Best validation accuracy: ~66.6% (at epoch 15)
- Best training accuracy: ~86.9%
- Best validation loss: ~0.9082
- Best training loss: ~0.2873

Initially, the model showed steady improvements in both training and validation accuracy, but the gap between them suggested mild overfitting starting around epoch 11. While training accuracy climbed above 84%, the validation accuracy plateaued around 70%, and the validation loss began fluctuating slightly. The lowest validation loss was reached before early stopping at epoch 16, indicating that the model began to overfit soon after. Despite this, the model performed decently given its simplicity.

Model 2: CNN + Attention

This second model expanded upon the base CNN by incorporating an attention mechanism and deeper convolutional layers. These additions aimed to help the model focus on relevant features in the fire images, especially since fire can have irregular shapes.

- Best validation accuracy: ~73.3% (at epoch 17)
- Best training accuracy: ~71.5%
- Best validation loss: ~0.6592
- Best training loss: ~0.5760

Compared to Model 1, this model showed a clear improvement in both accuracy and generalization. The validation accuracy increased by roughly 3% over the previous model, while the validation loss dropped to 0.6592. The attention layers appeared to help the model detect more nuanced patterns, leading to more reliable validation performance. Early stopping occurred at epoch 22, reverting to the best-performing weights from epoch 17.

Model 3: CNN + Attention + Transformer

Model 3 builds on Model 2 by incorporating transformer-style attention and deeper layers, which improved the model's ability to capture complex spatial dependencies in fire images.

- Best validation accuracy: 73.1% (higher than Model 2's 72.4%)
- Best validation loss: 0.6626 (comparable to Model 2's 0.6605)
- Early stopping at epoch 25 with the best weights restored from epoch 20
- Training accuracy steadily improved to ~71.9%, with a well-aligned validation curve, suggesting reduced overfitting

Overall, Model 3 outperformed Model 2, albeit slightly. The improvement in validation accuracy and consistency demonstrates the benefit of the additional attention mechanism. However, the gains came at the cost of significantly longer training time and higher model complexity. It is a better option if performance is prioritized over speed and simplicity.

Model 4: Pretrained ResNet50

This model utilized a pre-trained ResNet50 architecture and was trained for 20 epochs using class weighting to address class imbalance. Throughout training, both training and validation accuracy improved steadily, with no signs of severe overfitting.

- Final validation accuracy reached 65.8%, which is slightly below the best results from the custom CNN + attention models (Model 2 and Model 3).
- Validation loss gradually decreased and stabilized around 0.74, suggesting good convergence and relatively stable learning.
- Training accuracy also peaked at around 68.8%, showing the model was learning effectively without overfitting aggressively.
- Notably, the performance is respectable for a pretrained model with no custom architectural tuning or attention layers.

While this ResNet model offered stable and consistent training performance, it fell slightly short of the best-performing attention-based models in terms of validation accuracy. However, it remains a strong baseline given its simplicity and plug-and-play nature. With fine-tuning, data-specific pretraining, or added attention mechanisms, performance could likely be improved further.

Model 5: Pretrained MobileNetV2

This model used a pre-trained MobileNetV2 architecture and was trained for 30 epochs with class weighting and early stopping. However, the model struggled to learn effectively from the data, especially when compared to the other models tested.

- Validation accuracy plateaued early at 56.03% by epoch 4 and failed to improve beyond that point throughout training.
- Validation loss remained relatively high, hovering around 1.11 to 1.13, showing little improvement or adaptation.
- The training accuracy similarly showed minimal gains, maxing out around 37.7%, indicating underfitting or poor feature extraction from the given data.

This model performed the worst among all five tested. Despite using a pretrained MobileNetV2 backbone, it did not adapt well to the fire classification task. The lack of performance improvement suggests that the feature representations from MobileNetV2 may not be well-aligned with this dataset, or that further fine-tuning and architectural adjustments would be needed to make it viable.

Fire Prediction Web App

To make our fire classification model more accessible and interactive, we developed a simple web application that allows users to upload an image of a fire and receive a prediction of whether it is a Large, Medium, or Small Fire. We used Flask, a lightweight Python web framework, to build the backend.

The app loads our trained model (final_fire_model.keras) and handles image uploads through an HTML interface. The uploaded image is preprocessed (resized, normalized, and batch-expanded), passed into the model for prediction, and the result is rendered back to the user.

We structured the project with a clear file system:

- app.py: handles routing, preprocessing, and predictions.
- templates/index.html: The front-end form for image upload.
- final_fire_model.keras: The trained model file.

All development was done on Visual Studio Code, and dependencies were installed using pip. We tested the app locally on a MacBook Air (M3, 2024) with Python 3.9.6. This app bridges the gap between research and real-world usability by providing an intuitive tool for non-technical users.

Challenges

Long Training Time

Training our deep learning models—especially the more complex architectures like CNN with Attention and Transformer layers—took a significant amount of time. On our Google Cloud Platform (GCP) instance, some models took over 4 hours to complete, even with a relatively strong setup (32 vCPUs, 128 GB RAM). This limited the number of iterations we could perform for experimentation and tuning.

Model Accuracy & Limitations

While our models reached reasonable accuracy levels (around 72%–75% validation accuracy in the best cases), there's still clear room for improvement. One key limitation is that our current setup classifies every image as some type of fire—small, medium, or large. We don't have a "no fire" class, so an image with no fire may still be classified as a fire of some size. This is a potential area of concern for real-world deployment and something we aim to improve in the future.

Overestimation as a Strategy

Although we don't have clear evidence that the model overestimates fire severity, we acknowledge that overestimation is often safer than underestimation in emergency scenarios. If a medium fire is classified as a large fire, firefighters will arrive well-equipped and prepared. But if a medium fire is misclassified as small, the response team may be underprepared, which could endanger both the crew and those affected.

→ *"Better to overestimate than to be underprepared."*

Budget Limitation

As students working within budget constraints, we quickly realized that cloud computing is expensive. The longer training times on GCP drained our credits rapidly, limiting our ability to train more models, fine-tune hyperparameters, and experiment with additional architectures.

Next Steps & Future Improvements

Data Augmentation Enhancements

We plan to implement more targeted augmentation strategies such as zooming in and out, which may help the model learn spatial features more effectively, especially for smaller fires.

Confusion Matrix Analysis

Adding a confusion matrix would allow us to better understand where the model is getting confused, and we aim to focus more on improving recall, particularly for underrepresented classes like small fires.

More Data for Small Fires

Small fire images are significantly underrepresented in our dataset. Acquiring more samples for this class would help balance the data and improve the model's sensitivity to early-stage fires.

Include a "No Fire" Class

Introducing a fourth class for images without fire would make the model more applicable in real-world use, helping to avoid false positives.

Upgrade Infrastructure

With more funding and access to stronger GPUs, we could train more advanced models more efficiently and experiment with newer architectures or larger datasets.

GOOGLE DRIVE FOLDER LINK:

https://drive.google.com/drive/folders/1RUGkHuvF4aj_938CXIgyBH3RUFLutU6K?usp=sh
aring