Repo: https://github.com/rsm-etl015/Image-Classification-Models-for-Fire-Detection

Image Classification Models for Fire Detection

Ethan Lu

Abstract

This study explores deep learning models for fire classification to enhance wildfire detection and prevention. A dataset of 2,700 aerial and ground-based images is used to evaluate a custom convolutional neural network alongside five pre-trained models: MobileNetV2, DenseNet121, NASNet-Mobile, EfficientNet-B0, and ResNet50. This study applies image preprocessing and hyperparameter tuning to optimize model performance, incorporating early stopping to address computational constraints. Model effectiveness is assessed based on classification accuracy and robustness in distinguishing fire from non-fire images. This study identifies the best-performing model and integrates it into a real-time fire detection application, demonstrating the feasibility of automated wildfire monitoring. Findings contribute to improving deep learning-based fire detection systems, supporting early wildfire response and disaster prevention.

1 Introduction

1.1 Background

Wildfires are among the most devastating natural disasters, causing widespread destruction to ecosystems, human settlements, and infrastructure. Over the last decade, an average of 36 people per year have died due to wildfires in the United States (Statista, 2024) . The Palisades Fire in Los Angeles, which occurred in January 2025, resulted in 12 confirmed civilian fatalities (CAL FIRE, 2025). These recent disasters underscore the urgent need for more effective wildfire detection and response systems.

1.2 Dataset

This study utilizes the Wildfire Dataset, an open-source dataset designed to enhance deep learning-based wildfire detection (El-Madafri et al., 2023). The dataset consists of 2,700 aerial and ground-based images sourced from public repositories, capturing diverse environmental conditions, forest types, and fire scenarios (see **Figure 1**). Images are categorized into fire and non-fire classes, with non-fire images further including forested areas without confounding elements, fire confounding elements, and smoke confounding elements, while fire images contain smoke from fires and both smoke and fire.

To facilitate model training, the dataset is pre-split into training, validation, and test sets, with 70% allocated for training, 15% for validation, and 15% for testing (see **Table 2**). This structured division ensures effective model evaluation while preventing data leakage.

Figure 1: Sample Images

Set	Class	Subclass	#
Training (70%)	Nofire	Clear Forest	591
	Nofire	Fire Elements	254
	Nofire	Smoke Elements	330
	Fire	Fire Smoke	463
	Fire	Smoke and Fire	286
Validation (15%)	Nofire	Clear Forest	127
	Nofire	Fire Elements	50
	Nofire	Smoke Elements	69
	Fire	Fire Smoke	99
	Fire	Smoke and Fire	57
Test (15%)	Nofire	Clear Forest	128
	Nofire	Fire Elements	52
	Nofire	Smoke Elements	71
	Fire	Fire Smoke	100
	Fire	Smoke and Fire	59

Table 2: Dataset Class Distribution

2 Literature Review

2.1 CNN Architecture Overview

This study evaluates 6 convolutional neural network (CNN) models for fire image classification, including one custom designed model, BaselineCNN, and five widely used architectures: MobileNetV2, DenseNet121, NASNetMobile, EfficientNet-B0, and Res-Net50 (see **Table 3**). Each model has distinct architectural features and computational trade-offs. This section introduces the structure of these models, their theoretical foundations, and their advantages and limitations in computer vision tasks.

Model	# of Parameters	Size[1]
BaselineCNN	17,232,478	69 MB
MobileNetV2	2,260,546	9 MB
DenseNet121	7,040,579	28 MB
NASNetMobile	5,326,716	21 MB
EfficientNet-B0	5,300,000	21 MB
ResNet50	25,636,712	98 MB

Table 3: Model Size and Parameters

2.2 BaselineCNN

BaselineCNN is a custom-designed deep learning model inspired by VGGNet, with modern enhancements including ReLU activation and the Adam optimizer.

[1] Model sizes are estimated based on the number of parameters, assuming each parameter occupies 4 bytes (32 bits)

ReLU activation function:

$$f(x) = max\ (0, x)$$

ReLU activation prevents the vanishing gradient problem and accelerates training (Glorot et al., 2011) . The Adam optimizer combines momentum and adaptive learning rates, improving convergence speed and robustness in noisy environments. For binary classification, binary cross-entropy loss ensures effective training by penalizing incorrect pre-dictions (Goodfellow et al., 2016). While BaselineCNN provides flexibility in network design, its high parameter count increases the risk of over-fitting and requires substantial computational resources.

2.3 MobileNetV2

MobileNetV2 is an efficient deep learning model designed for mobile and embedded applications, featuring inverted residual blocks and linear bottlenecks. Unlike traditional networks, it expands representations, pro-cesses them, and projects them back into a lower-dimensional space (Sandler et al., 2018). This structure maintains expressiveness while reducing computation. MobileNetV2 also employs depth wise separable convolutions, reducing the number of parameters without significant accuracy loss. However, its lightweight design may limit performance in complex datasets requiring extensive feature extraction.

2.4 DenseNet121

DenseNet121 employs dense connectivity, where each layer receives inputs from all preceding layers, promoting feature reuse and gradient propagation (Huang et al., 2017). The architecture consists of four dense blocks, interspersed with transition layers that perform down sampling. This design improves parameter efficiency, leading to better feature propagation and reduced redundancy. While DenseNet121 achieves high accuracy with fewer parameters, its memory consumption is higher due to feature concatenation. It has been effectively applied in medical imaging and object recognition tasks (Ezzat et al., 2020; Zare and Pourkazemi, 2021).

2.5 NASNetMobile

NASNetMobile is a model designed through Neural Architecture Search (NAS), which optimizes network design based on computational

efficiency and accuracy trade-offs (Zoph et al., 2018). The architecture consists of normal cells (which maintain spatial dimensions) and reduction cells (which downsample feature maps). These auto-matically discovered blocks improve efficiency, making NASNetMobile well-suited for mobile applications and low-power devices (Ghodekar and Kumar, 2023). Despite its efficiency, NAS models require significant computational resources during training due to the automated architecture search.

2.6 EfficientNet-B0

EfficientNet-B0 utilizes compound scaling, where network depth, width, and resolution are uniformly scaled using fixed coefficients (Tan and Le, 2019). Unlike traditional models, which scale only one dimension (e.g., width or depth), EfficientNet-B0 balances all three factors, leading to improved performance with lower computational requirements. This model has achieved strong results in image classification, medical diagnostics, and edge computing applications (Harahap et al., 2024). However, its performance gain diminishes when scaled to extremely deep architectures.

2.7 ResNet50

ResNet50 is a deep convolutional neural network designed to mitigate the vanishing gradient problem using residual learning (He et al., 2016). It consists of residual blocks, where identity connections allow gradients to bypass certain layers, facilitating the training of very deep networks. ResNet50 is widely used in medical imaging, object detection, and segmentation tasks (Musthafa et al., 2024). While residual learning improves convergence, the model's large size increases computational demands, making it challenging for deployment in low-resource environments (Ramirez Amador et al., 2024).

3 Image Preprocessing

3.1 Importance of Image Preprocessing

Image preprocessing is a crucial step in deep learning pipelines, ensuring that input data is standardized, reducing computational overhead, and improving model generalization. Proper preprocessing stabilizes the training process and optimizes feature extraction by eliminating inconsistencies in the dataset. In this study, image normalization and lightweight data augmentation were applied to enhance the quality of input images and improve model performance.

3.2 Image Normalization

Normalization is essential for scaling pixel values to a uniform range, preventing large gradient values that may cause instability during training. Each image in the dataset originally contained pixel values in the range [0, 255], which were rescaled to [0, 1] using a Rescaling layer (Krizhevsky et al., 2012). This transformation helps improve the convergence rate of deep neural networks by ensuring consistent numerical representations across training samples.

The normalization process was implemented as follows:

$$Rescaling(1.0/255)$$

This ensures numerical stability and prevents feature dominance issues, particularly in models with batch normalization, where maintaining input scale consistency significantly impacts performance (Ioffe and Szegedy, 2015).

3.3 Data Augmentation

To improve generalization and reduce overfitting, lightweight data augmentation was applied. Augmentation artificially increases the dataset size by introducing small variations that help the model become more robust to minor transformations. The following transformations were used:

- Random Flip: Horizontally flips images to improve robustness against orientation shifts.
- Random Rotation: Rotates images within a 10% range to account for slight perspective variations.
- Random Zoom: Slightly zooms images in or out within 10% of the original size, helping the model handle scale variations.

These augmentations were chosen to retain the original image structure while introducing minor transformations, ensuring that the model generalizes better without excessive computational cost (Shorten and Khoshgoftaar, 2019).

3.4 Data Loading and Preprocessing

To optimize training efficiency, dataset preprocessing was performed before batching and caching operations. This prevents data-loading latency from slowing down training. TensorFlow's AUTOTUNE parameter dynamically optimizes the data pipeline, reducing CPU-GPU bottlenecks and improving memory usage (Abadi et al., 2016).

The dataset pipeline follows these steps:

1. Apply normalization and augmentation using TensorFlow's map() function.
2. Batch images into fixed-size groups for efficient training.
3. Prefetch using AUTOTUNE to optimize I/O performance.

4 Model Training

4.1 Structured and Hyperparameter Tuning

The study employed a structured hyperparameter tuning strategy to optimize model performance while considering computational constraints. To ensure reproducibility, all training processes were conducted using a fixed random seed (1234). The tuning procedure followed a random search strategy, where ten different hyperparameter configurations were sampled from a predefined hyperparameter list (Bergstra and Bengio, 2012). Each combination was trained for a maximum of 10 epochs, with early stopping triggered if validation accuracy did not improve for two consecutive epochs.

4.2 Hyperparameter Tuning Strategy

Hyperparameter tuning focused on optimizing key parameters, including learning rate, batch size, dropout rate, convolutional filters, kernel size, dense layer units, data augmentation techniques, and activation functions. Rather than employing an exhaustive grid search, which evaluates all possible parameter combinations, the random search method was selected for its ability to efficiently explore a diverse set of configurations while maintaining computational efficiency (Li et al., 2017).

Since each model had a distinct architecture, hyperparameter search spaces were customized rather than keeping them identical across all models. For instance, BaselineCNN was tuned by testing different learning rates (0.0001, 0.001, 0.01), batch sizes (32, 64), dropout rates (0.2, 0.5), and activation functions (ReLU, Leaky ReLU), while EfficientNet-B0 and NASNetMobile had distinct hyperparameter ranges suited to their architecture.

4.3 Resource Allocation

Training was conducted in Kaggle Notebooks using a CPU environment, as GPU runtime was limited to 30 hours per week. To efficiently use computational resources, batch sizes and network complexity were adjusted based on model architecture. Furthermore, early stopping played a critical role in ensuring efficient training, allowing more hyperparameter configurations to be explored within the computational budget.

To overcome Kaggle's 9-hour runtime limitation, models were designed to halt early if no performance improvements were observed. This significantly reduced unnecessary computations and enabled the exploration of more hyperparameter configurations within the allocated runtime (Goodfellow et al., 2016) (see **Table 4**).

Model	Ep	Runtime	Per Ep
BaselineCNN	5.9	6.29 hrs	1.07 hrs
MobileNetV2	5	3.49 hrs	0.70 hrs
DenseNet121	5.8	7.30 hrs	1.26 hrs
NASNetMobile	6.3	5.34 hrs	0.85 hrs
EfficientNet-B0	3	2.09 hrs	0.70 hrs
ResNet50	6.5	5.05 hrs	0.78 hrs

Table 4: Model Training Time Summary

4.4 Model-Specific Training Characteristics

Different CNN architectures exhibit varying training dynamics based on their depth, parameter count, and optimization strategies. Deeper models typically require more epochs to reach optimal performance, while lightweight architectures converge more quickly. Early stopping, although effective in preventing overfitting, may sometimes hinder performance by prematurely terminating training before full convergence.

For instance, ResNet50, with its deep residual connections, typically requires 6 - 7 epochs to fully optimize its feature representations. The skip connections facilitate gradient flow, reducing the risk of vanishing gradients, but the depth of the model necessitates longer training for

stable convergence (He et al., 2016). Early stopping in this case worked as intended, preventing unnecessary training while allowing the model to reach near-optimal performance.

Conversely, EfficientNet-B0 demonstrated systematic underfitting, consistently stopping at three epochs due to early stopping. EfficientNet-B0's compound scaling strategy, which jointly scales network width, depth, and resolution, requires a longer training period to fully optimize its representations (Tan and Le, 2019). The aggressive early stopping criteria forced training to halt before the model could sufficiently learn, highlighting the importance of adaptive early stopping strategies that adjust based on model complexity

5 Model Evaluation

5.1 Overall Model Performance

The six trained models were evaluated based on their training, validation, and test accuracy (see **Table 5**). DenseNet121 achieved the highest test accuracy (91.71%), outperforming all other models. NASNetMobile (87.80%) and MobileNetV2 (85.37%) followed closely, demonstrating strong generalization capabilities. The BaselineCNN model performed moderately well, reaching 82.68% test accuracy, while ResNet50 lagged slightly behind at 80.49%. EfficientNet-B0 performed the worst, with only 61.22% test accuracy, indicating severe underfitting and poor generalization.

Model	Train	Val	Test
BaselineCNN	0.8400	0.7985	0.8268
MobileNetV2	0.8463	0.8333	0.8537
DenseNet121	0.9698	0.8856	0.9171
NASNetMobile	0.9417	0.8607	0.8780
EfficientNet-B0	0.6131	0.6119	0.6122
ResNet50	0.7933	0.7363	0.8049

Table 5: Model Performance

5.2 Overfitting and Underfitting

Certain models exhibited overfitting, while others suffered from underfitting. DenseNet121 and NASNetMobile had high training accuracy, with DenseNet121 reaching 96.98%, significantly higher than its validation and test accuracy. This suggests mild overfitting, where the model learned intricate patterns that did not generalize perfectly to unseen data. BaselineCNN and MobileNetV2 demonstrated a more balanced performance, indicating they effectively generalized without excessive overfitting (see **Table 5**).

EfficientNet-B0 showed clear underfitting, as its accuracy remained consistently low across all datasets. This supports findings that EfficientNet architectures require careful tuning of scaling coefficients to prevent performance degradation (Tan and Le, 2019). The model likely failed to extract meaningful features from the dataset, contributing to its poor classification ability.

5.3 Confusion Matrix Analysis

A confusion matrix was generated for each model to analyse classification errors (see **Table 6**). The primary concern in wildfire detection is false negatives (Type II errors), where fire instances are misclassified as "no fire," potentially leading to catastrophic consequences.

DenseNet121 produced the fewest false negatives (13 cases), achieving the highest recall (91.82%) among all models. EfficientNet-B0 completely failed to classify fire instances, reinforcing its poor predictive power (see **Table 7**). MobileNetV2 and NASNetMobile also exhibited relatively high false negatives (47 and 16 cases, respectively), but their precision remained competitive, making them viable alternatives for fire detection.

Model	TN	FN	TP	FP
BaselineCNN	202	22	137	49
MobileNetV2	238	47	112	13
DenseNet121	230	13	146	21
NASNetMobile	217	16	143	34
EfficientNet-B0	251	159	0	0
ResNet50	219	48	111	32

Table 6: Confusion Matrix

Model	Precision	Recall	F1
BaselineCNN	0.7366	0.8616	0.7942
MobileNetV2	0.8960	0.7044	0.7887
DenseNet121	0.8743	0.9182	0.8957
NASNetMobile	0.8079	0.8994	0.8512
EfficientNet-B0	NaN	NaN	NaN
ResNet50	0.7762	0.6981	0.7351

Table 7: Model Performance Metrics

5.4 BaselineCNN and DenseNet121

BaselineCNN required 10 epochs to achieve peak accuracy, whereas DenseNet121 converged within

only 4 epochs (see **Figure 8**). This difference is attributed to the feature reuse mechanism in DenseNet121, which enhances gradient flow and accelerates convergence (Huang et al., 2017). BaselineCNN, lacking such optimizations, required extended fine-tuning to achieve comparable accuracy. The dense connectivity in DenseNet121 reduces redundant feature learning, improving both efficiency and stability in training.

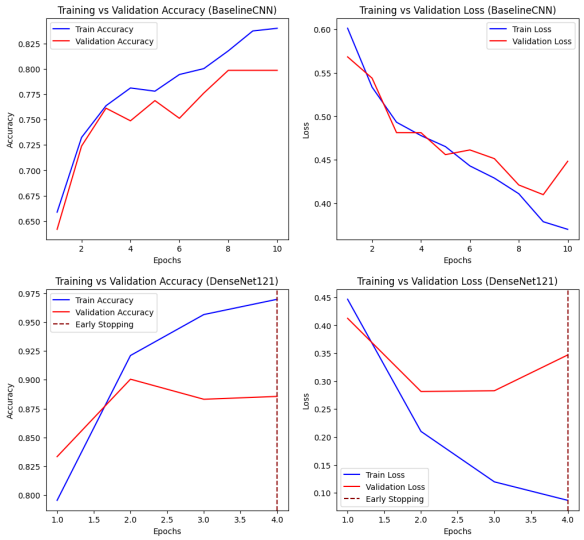

Figure 8: Training History

DenseNet121 underwent extensive hyperparameter tuning, testing 10 different configurations (see **Table 9**). The optimal configuration used a learning rate of 0.0001, batch size of 32, dropout rate of 0.5, and 16 trainable growth rate blocks with Swish activation. These settings yielded the highest accuracy (91.71%), aligning with prior research that adaptive activation functions like Swish improve deep network performance (Ramachandran et al., 2018).

DenseNet121's misclassified samples included false positives (fire misclassified as no fire) and false negatives (no fire misclassified as fire) (see **Figure 10**). False negatives are particularly problematic, as undetected wildfires can lead to severe environmental damage (Lagerquist et al., 2019). Despite its high recall, DenseNet121 still produced some false negatives, emphasizing the need for further improvements, such as ensemble models or additional feature engineering.

Learning Rate	Batch Size	Drop[1]	Dense	Aug[2]
0.0001	32	0.5	512	FALSE
0.0001	32	0.2	256	TRUE
0.0001	32	0.5	128	TRUE
0.01	64	0.2	512	FALSE
0.0001	32	0.2	512	FALSE
0.01	32	0.5	256	TRUE
0.001	16	0.3	128	FALSE
0.001	32	0.5	256	TRUE
0.01	64	0.5	256	TRUE
0.01	32	0.5	128	TRUE
Train[3]	**Growth Rate**	**Blocks**	**Activ[4]**	**Acc[5]**
TRUE	16	4	swish	0.9171
TRUE	16	3	swish	0.9171
TRUE	32	4	relu	0.9098
FALSE	16	4	relu	0.8951
FALSE	16	3	relu	0.8902
FALSE	16	3	relu	0.8854
TRUE	32	5	swish	0.7878
TRUE	32	4	relu	0.7854
TRUE	32	5	swish	0.6415
TRUE	32	5	relu	0.6122

Table 9: Hyperparameter Tuning Summary

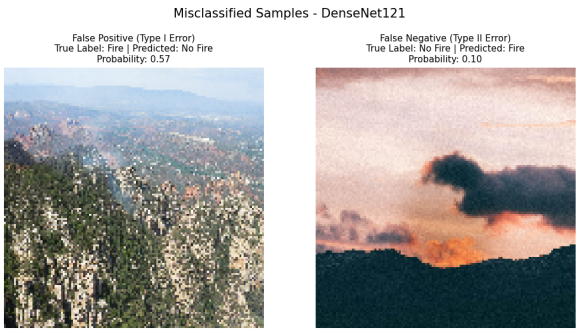

Figure 10: Misclassified Samples

5.5 ROC Curve

The Receiver Operating Characteristic (ROC) curve was plotted to compare model performance (see **Figure 11**). DenseNet121 achieved the highest AUC (0.98), confirming its superior discriminative power. NASNetMobile and MobileNetV2 followed closely, with AUC scores of 0.95 and 0.91, respectively. The BaselineCNN model achieved an AUC of 0.90, demonstrating solid performance despite its simpler architecture. EfficientNet-B0 had the lowest AUC (0.41), performing worse than a random classifier, suggesting severe instability in its training process (Tan and Le, 2019).

[1] Drop: Dropout. [2] Aug: Data Augmentation. [3] Train: Trainable
[4] Activ: Activation. [5] Acc: Test Accuracy.

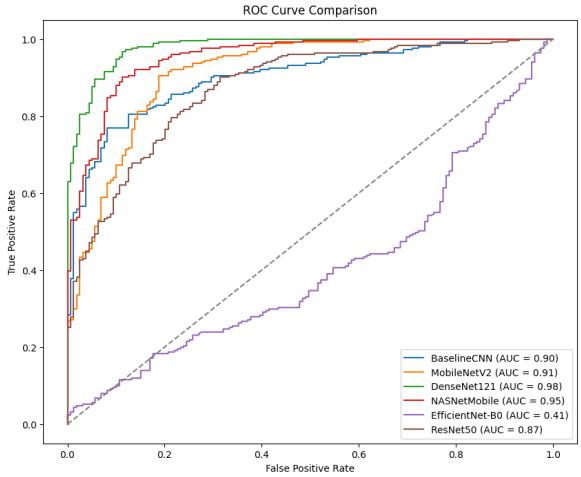

Figure 11: ROC Curve

EfficientNet-B0's AUC score of 0.41 indicates that it failed to extract meaningful features, leading to poor generalization. This is likely due to inefficient scaling of network depth, width, and resolution, a key requirement for EfficientNet models (Tan and Le, 2019). Without proper tuning, EfficientNet models can suffer from unstable gradients and ineffective feature learning, degrading performance in complex classification tasks (Pham et al., 2018).

6 Live Video Fire Detection

6.1 System Overview

The live fire detection system was developed using a DenseNet121-based ONNX model for real-time wildfire classification. To enhance reliability, colour-based fire region segmentation was integrated, leveraging HSV colour filtering to highlight potential fire areas. The system processes video frames in real-time, assigns a fire probability score, and overlays bounding boxes around suspected fire regions. This hybrid approach improves detection robustness by combining deep learning with colour-based segmentation techniques.

6.2 Colour Segmentation

To refine fire detection, HSV colour filtering was implemented, targeting red-orange intensity hues commonly associated with fire. The following predefined HSV bounds were used for segmentation:

- Lower Bound 1: (0, 50, 50)
- Upper Bound 1: (10, 255, 255)
- Lower Bound 2: (170, 50, 50)
- Upper Bound 2: (180, 255, 255)

The system converts each video frame to the HSV colour space, applies thresholding, and generates a binary mask highlighting fire-like regions. Contours are extracted, and only regions exceeding 500 pixels in area are considered valid fire detections. Bounding boxes are applied, with red boxes indicating high fire intensity and green boxes denoting uncertain fire areas.

6.3 Evaluation

The first test image illustrates a correctly classified wildfire scenario, where the DenseNet121 model assigned a fire probability of 0.96 and the colour segmentation algorithm successfully detected fire regions. This demonstrates the system's effectiveness in identifying large-scale fire events with high confidence (see **Figure 12**).

However, the second test image reveals a false positive case, where the model incorrectly classified a fox in a natural environment as wildfire with a 0.90 probability. This error likely stemmed from sunlight reflections or high-red hues in the background, leading the colour segmentation algorithm to misinterpret non-fire elements as fire (see **Figure 12**).

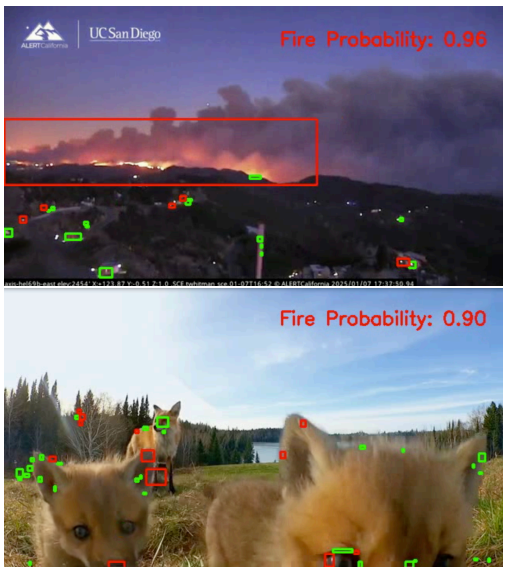

Figure 12: Live Video Fire Detection

6.4 Challenges and Future Improvements

Despite improvements in fire detection reliability, the system remains vulnerable to false positives, particularly in non-fire scenarios containing strong

red hues. Previous studies have reported similar issues, where fire detection models exhibit reduced precision in complex backgrounds and varying lighting conditions (Muhammad et al., 2018). Potential enhancements to mitigate these challenges include:

- Refining the fire probability threshold: Implementing adaptive thresholds based on scene context rather than a fixed probability cutoff.
- Incorporating temporal analysis: Tracking fire progression across multiple frames instead of relying on single-frame classification, which can help filter out transient false positives.
- Integrating multispectral data: Utilizing infrared imaging in addition to RGB input, as prior research has shown that thermal signatures improve fire detection accuracy in remote sensing applications (Chuvieco et al., 2019).

7 Conclusion

This study explored the use of deep learning models for wildfire detection, evaluating a custom CNN (BaselineCNN) and five pre-trained architectures (MobileNetV2, DenseNet121, NASNetMobile, EfficientNet-B0, and ResNet50). The research demonstrated that DenseNet121 achieved the highest test accuracy (91.71%) and recall (91.82%), making it the most effective model for fire classification. NASNetMobile and MobileNetV2 also showed strong performance, balancing precision and recall, while BaselineCNN performed moderately well. EfficientNet-B0 failed to generalize, exhibiting severe underfitting due to aggressive early stopping and ineffective feature scaling.

To enhance real-world applicability, a DenseNet121-based ONNX model was deployed for live video fire detection, integrating colour-based fire segmentation using HSV filtering. This system successfully detected wildfires in real-time, but false positives in high-red-hue environments remained a challenge. The study identified key areas for improvement, including:

1. Adaptive early stopping for models requiring longer fine-tuning (e.g., EfficientNet-B0).
2. Ensemble learning to reduce false negatives, crucial for fire detection applications.
3. Integration of multispectral data (infrared imaging) to improve detection robustness in challenging environments.
4. Temporal analysis to track fire progression over multiple frames, reducing single-frame misclassifications.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., & others. (2016). TensorFlow: A system for large-scale machine learning. *OSDI, 16*, 265-283.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*(1), 281-305.

CAL FIRE. (2025). *Palisades Fire Incident Information*. Retrieved from https://www.fire.ca.gov/incidents/2025/1/7/palisades-fire

Chuvieco, E., Mouillot, F., van der Werf, G. R., San-Miguel-Ayanz, J., & Tanase, M. (2019). Historical background and current developments for mapping burned area from satellite Earth observation. *Remote Sensing of Environment, 225*, 45-64.

El-Madafri, I., Peña, M., & Olmedo-Torre, N. (2023). The Wildfire Dataset: Enhancing deep learning-based forest fire detection with a diverse evolving open-source dataset focused on data representativeness and a novel multi-task learning approach. *Forests, 14*(9), 1697. https://doi.org/10.3390/f14091697

Ezzat, D., Hassanien, A. E., & Ella, H. A. (2020). GSA-DenseNet121-COVID-19: A hybrid deep learning architecture for the diagnosis of COVID-19 disease based on gravitational search optimization algorithm. *arXiv preprint arXiv:2004.05084*.

Ghodekar, A., & Kumar, A. (2023). LightLeafNet: Lightweight and efficient NASNetMobile architecture for tomato leaf disease classification. *SSRN Electronic Journal*.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 315–323.

Goodfellow, I., Bengio, Y., & Courville, Y. (2016). *Deep learning*. MIT Press.

Harahap, M., Husein, A. M., Kwok, S. C., Wizley, V., Leonardi, J., Ong, D. K., & Siahaan, S. (2024). Skin cancer classification using EfficientNet architecture. *Bulletin of Electrical Engineering and Informatics, 13*(4), 2716-2728.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770-778.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4700-4708.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097-1105.

Lagerquist, R., McGovern, A., & Smith, T. (2019). Deep learning for severe weather phenomena: Nowcasting thunderstorms. *Weather and Forecasting, 34*(5), 1097–1115.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research, 18*(1), 6765-6816.

Muhammad, K., Ahmad, J., Baik, S. W., & Kittler, J. (2018). Efficient deep CNN-based fire detection and localization in video surveillance applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 49*(7), 1419-1434.

Musthafa, M., Mahesh, T. R., Kumar, V. V., & Guluwadi, S. (2024). Enhancing brain tumor detection in MRI images through explainable AI using Grad-CAM with ResNet50. *BMC Medical Imaging, 24*, Article number: 107.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., & Dean, J. (2018). Efficient neural architecture search via parameter sharing. *Proceedings of the 35th International Conference on Machine Learning*, 4095–4104.

Ramachandran, P., Zoph, B., & Le, Q. V. (2018). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Ramirez Amador, P., Ortega, D. M., & Cesarano, A. (2024). Detection of pulmonary pathologies using convolutional neural networks, data augmentation, ResNet50, and vision transformers. *arXiv preprint arXiv:2409.14446*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. *arXiv preprint arXiv:1801.04381*.

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data, 6*(1), 1-48.

Statista. (2024). *Number of deaths due to wildfires in the United States from 1990 to 2023*. Retrieved from https://www.statista.com/statistics/1422130/usa-number-of-deaths-due-to-wildfires/

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. *Proceedings of the 36th International Conference on Machine Learning*, 6105-6114.

Zare, R., & Pourkazemi, A. (2021). DenseNet approach to segmentation and classification of dermatoscopic skin lesions images. *arXiv preprint arXiv:2110.04632*.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.