Convolutional Neural Network (CNN) Architecture and Performance Analysis for Emergency Vehicle Classification

1. Introduction

This report discusses the configuration, implementation, and optimization of a CNN model for binary classification of emergency vehicle images. The model predicts whether an image represents an emergency vehicle or not, based on spatial features extracted by the CNN. This study examines the architecture, regularization techniques, and hyperparameter tuning applied to optimize performance. Results are analyzed through graphs and metrics to evaluate the model's generalization ability and address overfitting.

2. Methods

2.1 Data Preparation

The dataset used for this study was sourced from Kaggle.com. Following retrieval of the dataset, key Python libraries, including TensorFlow and Keras for model development, and Pandas for data handling were adopted. Each image was labeled either as "emergency" or "non-emergency" in a corresponding CSV file. The training dataset included images mapped to their respective labels, while the test dataset contained unlabeled images for performance evaluation. The images were provided as raw files within a folder structure, requiring preprocessing for use. To prepare the data, a custom defined function was used to reorganize the images into directories based on their labels, enabling further use with Keras' ImageDataGenerator. The dataset was augmented using transformations such as random rotations, zoom, and brightness adjustments. These methods are effective in increasing variability and improving model robustness, as outlined by Shorten and Khoshgoftaar (2019).

2.2 CNN Architecture

CNNs have proven highly effective for image classification tasks by extracting spatial hierarchies of features, as demonstrated by Krizhevsky, Sutskever, and Hinton (2012) and further refined by Simonyan and Zisserman (2015). For this study, the CNN was configured with three convolutional layers, each with a kernel size of (3, 3), 16 filters and ReLU activation. These layers were chosen for their ability to detect spatial features such as edges and textures while maintaining simplicity. To reduce the spatial dimensions of the feature maps for improved efficiency, MaxPooling layers were situated after each convolutional layer. Batch normalization layers were situated after each convolutional layer to stabilize and accelerate training by normalizing intermediate activations, which helped prevent divergence during optimization.

To prevent overfitting, dropout layers were added with a rate of 50% after the second and third convolutional layers and the dense layer. The high dropout rate was selected after observing overfitting in earlier iterations of the model, where lower dropout rates (e.g., 30-50%) failed to align the training and validation loss curves to meet expectations. These layers randomly disable neurons during training, minimizing the model’s potential overreliance on on specific features.

The dense layers consisted of a fully connected 16 units layer and ReLU activation, which aggregated spatial features into meaningful representations, followed by a final output layer

with a single unit and sigmoid activation for binary classification. This simple configuration ensures spatial features are meaningful representations in binary outputs. The use of ReLU activation in hidden layers was implemented to address the vanishing gradient problem, while the sigmoid function provided outputs suitable for binary decisions (Brownlee, 2020; Hackernoon, 2023).

The architecture of the first model is summarized in Table 1. Subsequent models where modified to vary outputs and their configurations are attached herewith.

Table 1: Summary of the CNN architecture – Model 1

Layer (Type)	Output Shape	Paremeters
Conv2d (Conv2D)	(None, 126, 126, 16)	448
Max_pooling2d (MaxPooling2D)	(None, 63, 63, 16)	0
Batch_normalization (BatchNormalization)	(None, 63, 63, 16)	64
Conv2d_1 (Conv2D)	(None, 61, 61, 16)	2,320
Max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 16)	0
Dropout (Dropout)	(None, 30, 30, 16)	0
Batch_normalization_1 (BatchNormalization)	(None, 30, 30, 16)	64
Conv2d_2 (Conv2D)	(None, 28, 28, 16)	2,320
Max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 16)	0
Dropout_1 (Dropout)	(None, 14, 14, 16)	0
Flatten (Flatten)	(None, 3136)	0
Dense (Dense)	(None, 16)	50,192
Dropout_2 (Dropout)	(None, 16)	0
Dense_1 (Dense)	(None, 1)	17

2.3 Regularization Methods

To address the risk of overfitting, two regularization techniques were incorporated into the model. L2 regularization (*kernel_regularizer*) was applied to the dense layers with a penalty factor of 0.01, which constrained the magnitude of weights, an approach adopted to promote model simplicity, similarly explored by Ng (2004). By discouraging large weight values, the model was encouraged to adopt simpler solutions, thereby improving its generalization performance. Dropout was applied to convolutional and dense layers with a rate of 0.5, which further reduced overfitting by randomly deactivating neurons during training as highlighted by Srivastava et al. (2014). This ensured the model learned robust feature representations rather than memorizing specific patterns.

The combination of these techniques significantly reduced the validation loss while keeping the training and validation curves closely aligned, as shown in Figure 4. The use of both L2 regularization and dropout proved beneficial in achieving a validation accuracy of 73.81%, indicating that the model generalized well without overfitting.

2.4 Hyperparameter Tuning

Several hyperparameters were carefully tuned to optimize the performance of the model. The learning rate was initially set to 0.0005 and dynamically adjusted using a learning rate scheduler. The ReduceLROnPlateau callback halved the learning rate whenever validation loss plateaued for three consecutive epochs. This adjustment prevented the model from overshooting the optimal solution and stabilized its convergence during later epochs. This aligns with the findings of Fetterman et al. (2023), who demonstrated the importance of dynamic hyperparameter optimization for efficient training. During the training, class weights were also computed and applied to manage the class imbalance in the dataset. The weights, 0.85 for class 0 and 1.21 for class 1, ensured the model paid equal attention to both classes, which improved recall for the minority class.

A batch size of 32 was selected to balance computational efficiency and gradient stability, an approach supported by Bartz-Beielstein et al. (2021). This choice allowed the model to process enough data per iteration to calculate accurate gradients while keeping memory usage within limits. The dropout rate of 60% was found to be critical in reducing overfitting, particularly in the dense layers where overfitting risks are believed to be higher due to the large number of trainable parameters.

Figures 3 and 4 illustrate the profile of accuracy and loss during training for Model 1. These plots highlight the performance of the chosen hyperparameters, with steady improvements in validation metrics and minimal divergence between training and validation loss.

3. Results

Overfitting was evident in earlier trained-tested models, where the validation loss increased after initial epochs while the training loss continued to decrease. This divergence, combined with fluctuating validation accuracy, suggested the model was memorizing training data rather than generalizing to unseen data. After introducing L2 regularization, dropout, and a learning rate scheduler, the best performing model achieved a validation accuracy of 83.13% and demonstrated minimal overfitting. The training and validation losses decreased consistently across epochs and converged to similar values, as seen in Figure 5.

The alignment of training and validation accuracy curves further supports the reduced overfitting. Table 2 & 3 provides a summary of evaluation metrics, demonstrating balanced performance on both training and validation datasets. See Figure 2 for the model's prediction of select images from the test data.

Table 2: Model Evaluation Metrics – Training

Model	Loss	Accuracy	Precision	Recall
Model 1	0.4471	83.13%	84.13%	74.30%

Model 2	0.6241	57.84%	50.20%	86.21%
Model 3	0.8303	72.92%	67.26%	70.56%

Table 3: Model Evaluation Metrics – Validation

Model	Loss	Accuracy	Precision	Recall
Model 1	0.4632	82.54%	86.21%	70.09%
Model 2	0.5892	76.98%	92.98%	49.53%
Model 3	0.8018	77.38%	77.78%	65.42%

4. Discussion

Significant improvements to the model performance were observed after the implementation of L2 regularization and dropout, and it is believed that this was instrumental in addressing overfitting. Another possible explanation for the improved observed generalization is that the dropout, applied at a high rate of 0.5, reduced reliance on specific neurons, forcing the model to learn robust patterns. Without these techniques, earlier iterations of the model exhibited diverging loss curves and fluctuating validation accuracy, which is typically indicative of overfitting.

As earlier highlighted, Hyperparameter tuning, particularly the use of a learning rate scheduler at varied rates, improved convergence during later epochs by dynamically reducing the learning rate. This adjustment helped avoid overshooting the optimal solution, as observed in earlier training attempts. Class weighting was important in addressing the dataset's imbalance, improving recall for the minority class, although it remained relatively low at 0.3831. The consistent alignment of training and validation loss curves indicates that these methods collectively improved the model's ability to generalize.

Despite these improvements, irregular patterns suggesting that further fine-tuning may be necessary. Some of the approaches that may be considered include transfer learning or increasing filter counts in convolutional layers to enable the model to capture more complex patterns. Advanced data augmentation strategies may also improve class balance in feature representation.

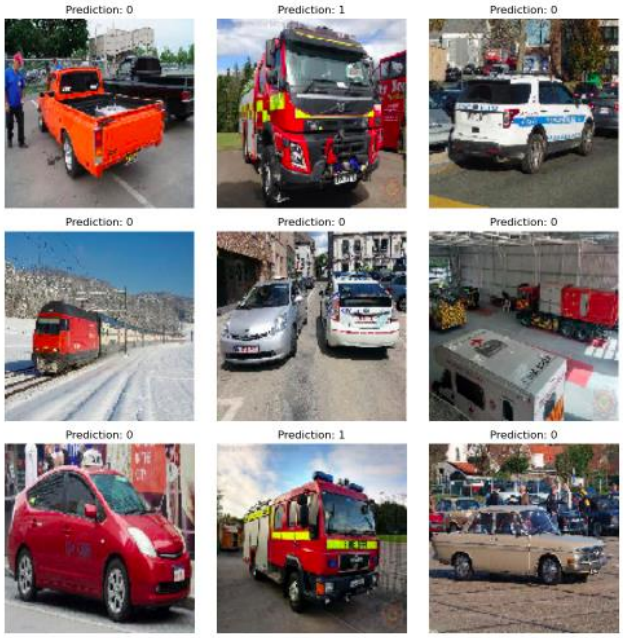

Figure 1: Model 1 Emergency Vehicle Predictions

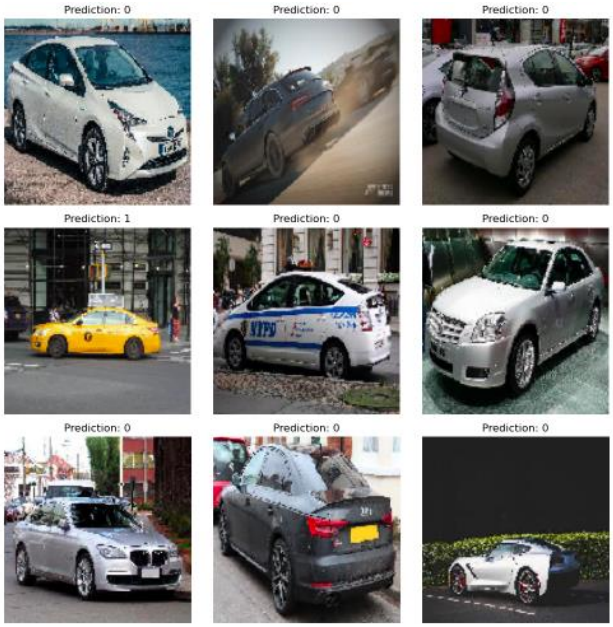

Figure 2: Model 2 Emergency Vehicle Predictions

Figure 3: Model 3 Emergency Vehicle Predictions

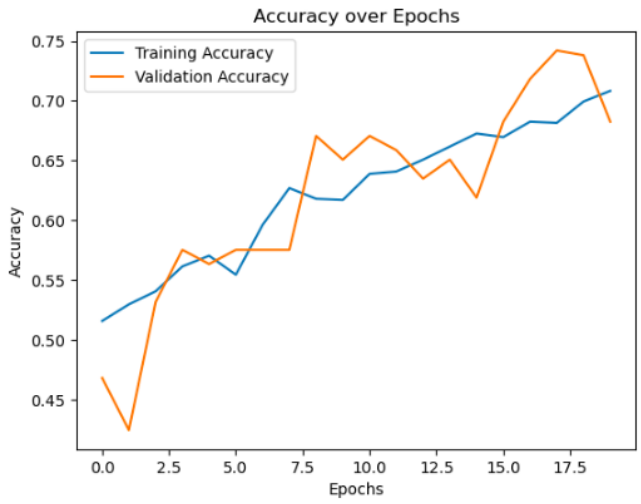

Figure 4: Model 1 Accuracy over Epochs

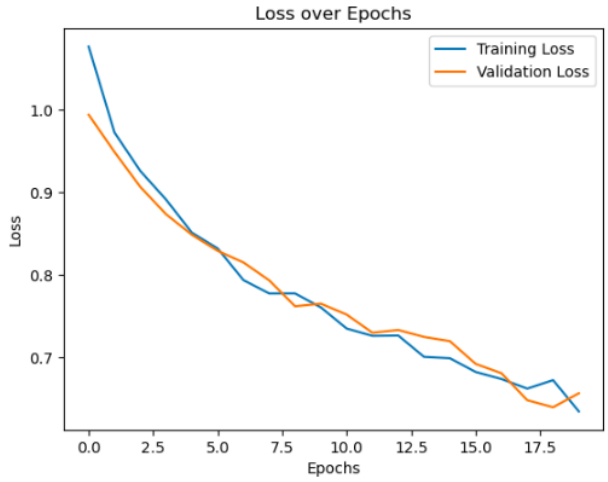

Figure 5: Model 1 Loss over Epochs

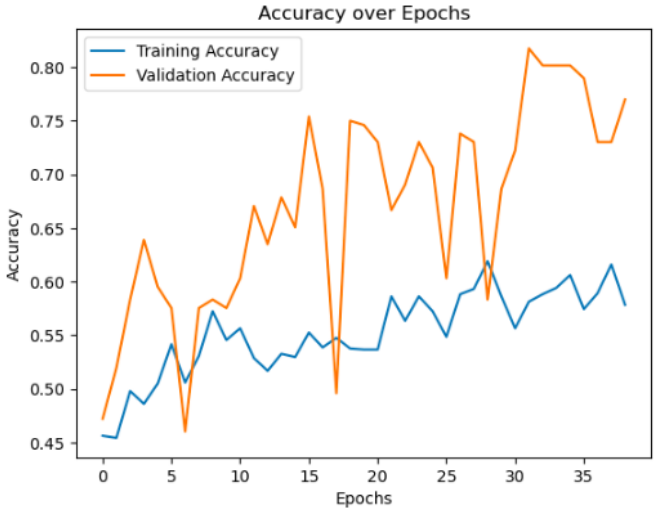

Figure 6: Model 2 Accuracy over Epochs

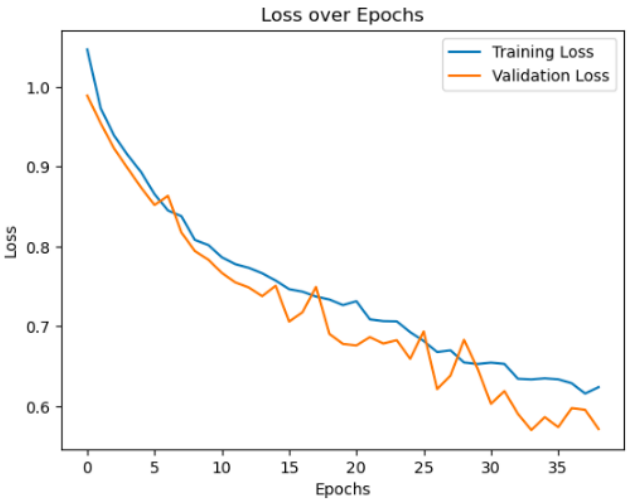

Figure 7: Model 2 Loss over Epochs

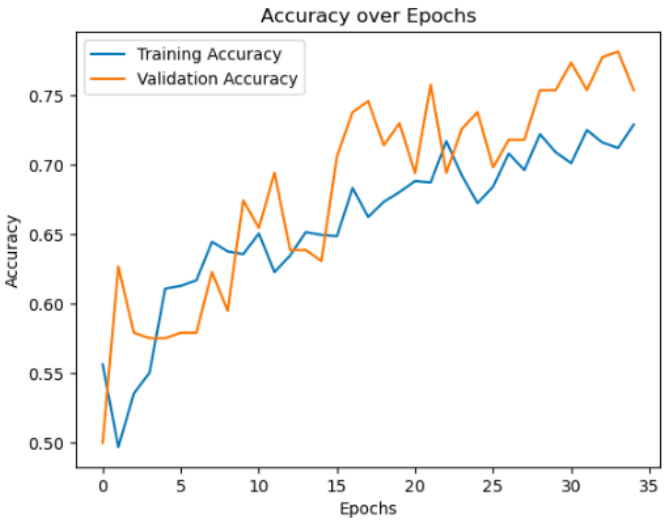

Figure 8: Model 3 Accuracy over Epochs

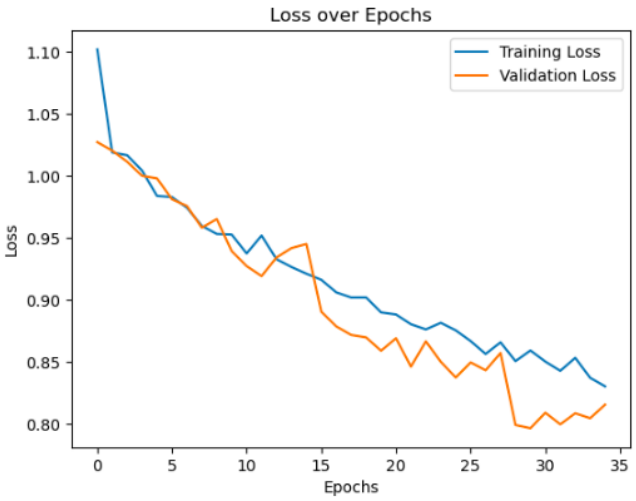

Figure 9: Model 2 Loss over Epochs

References

Bartz-Beielstein, T. (2023*) 'PyTorch Hyperparameter Tuning - A Tutorial for spotPython'*, arXiv preprint arXiv:2305.11930. Available at: https://arxiv.org/abs/2305.11930.

Bartz-Beielstein, T. et al. (2021) '*Surrogate Model Based Hyperparameter Tuning for Deep Learning with SPOT*', arXiv preprint arXiv:2105.14625. Available at: https://arxiv.org/abs/2105.14625.

Brownlee, J. (2020) *'How to Fix the Vanishing Gradients Problem Using the Rectified Linear Activation Function'*, Machine Learning Mastery. Available at: https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/ (Accessed: 30 November 2024).

ChatGPT, *CNN Architecture and Performance Analysis* [AI generated python code]. Prompted by Abel Ekele. 28 Nov. 2024, 18:05.

Fetterman, A.J. et al. (2023) *'Tune As You Scale: Hyperparameter Optimization For Compute Efficient Training'*, arXiv preprint arXiv:2306.08055. Available at: https://arxiv.org/abs/2306.08055.

Hackernoon (2023) *'Binary Classification: Understanding Activation and Loss Functions with a PyTorch Example'*, Hackernoon. Available at: https://hackernoon.com/binary-classification-understanding-activation-and-loss-functions-with-a-pytorch-example
(Accessed: 30 November 2024).

Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012) *'ImageNet Classification with Deep Convolutional Neural Networks'*, Advances in Neural Information Processing Systems, 25, 1097–1105. Available at: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

Ng, A.Y. (2004) *'Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance'*, Proceedings of the Twenty-First International Conference on Machine Learning, p. 78-85. Available at: https://dl.acm.org/doi/10.1145/1015330.1015435.

Shorten, C. and Khoshgoftaar, T.M. (2019) *'A survey on image data augmentation for deep learning'*, Journal of Big Data, 6(1), p. 60. Available at: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0 (Accessed: 1 December 2024).

Simonyan, K. and Zisserman, A. (2015) *'Very Deep Convolutional Networks for Large-Scale Image Recognition'*, International Conference on Learning Representations. Available at: https://arxiv.org/abs/1409.1556.

Srivastava, N. et al. (2014) *'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'*, Journal of Machine Learning Research, 15(1), pp. 1929–1958. Available at: http://jmlr.org/papers/v15/srivastava14a.html.