FUNDAMENTALS OF DEEP LEARNING

FINAL PROJECT

ARIVAZHAGAN ANTOSIBIRAYAN

MURUGESAN BHARANIDHARAN

SUBRAMANI PRAVEEN

5 APRIL 2024

Project

Vehicle Detection and Classification in Accident Images

Project background

This project delves into the realm of neural networks to tackle the challenge of vehicle detection and classification in accident scenarios.

- **Multi-Layer Perceptron Networks (MLP):** Our journey begins with a baseline MLP model, designed to understand the intricate relationships between extracted image features and vehicle types.
- **Convolutional Neural Networks (CNN):** Next, we'll leverage the power of CNNs, specifically designed to extract features from images. This architecture holds promise for significantly improved performance.
- **Transfer Learning:** Finally, we'll investigate the potential of transfer learning, where pre-trained networks honed on massive image datasets are utilized to enhance both efficiency and accuracy of our model.

To conduct this comparative analysis, we'll be utilizing a focused subset of the "Accident Images Analysis" dataset. This subset comprises 2,946 images of damaged vehicles, categorized as "light vehicle," "heavy vehicle," or "motorcycle." Each image is formatted in RGB colour with a resolution of 224x224 pixels. By meticulously evaluating the performance of each model architecture and its hyperparameters, we aim to identify the most effective approach for real-world accident scenarios. Our goal is achieving the highest accuracy in vehicle detection and classification within accident images.

Data Preprocessing

A custom Python function efficiently handles the loading of images from a designated base directory. These images are converted into NumPy arrays, a format conducive to streamlined processing within machine learning frameworks. Moreover, all images undergo resizing to a consistent size of 224x224 pixels, aligning with the typical requirements of image recognition models.

Subsequently, the dataset undergoes partitioning into distinct training and testing subsets, employing the common 70/30 split ratio. This division allocates 70% of the images to the training set, facilitating model development. The remaining 30% constitute the test set, enabling rigorous evaluation of the model's performance on previously unseen data. This strategic split aids in preventing overfitting and promotes the model's ability to generalize effectively.

For classification tasks, label transformation is conducted using the `to_categorical` function. This transformation involves converting integer-based labels (e.g., 0 for cars, 1 for motorcycles) into one-hot encoded vectors. Each vector's length corresponds to the total number of classes present in the dataset. Through this encoding method, the model gains the capacity to predict class probabilities accurately during classification. The utilization of one-hot encoding ensures equitable training representation across all classes and helps mitigate biases that may arise from imbalanced class distributions.

Multi-Layer Perceptron Networks (MLP)

Model Architecture

We employed Multi-Layer Perceptron (MLP) networks for analyzing our accident detection dataset. Initially, we converted the images to grayscale and normalized them to a range between 0 and 1. Subsequently, we designed three distinct MLP models with different architectures:

Model	Hidden Layers	Neurons per Layer	Activation Function	Output Layer Activation
MLP1	3	64	ReLU	Softmax
MLP2	4	128	ReLU	Softmax
MLP3	2	256	ReLU	Softmax

For training these models, we utilized the categorical cross-entropy loss function and the Adam optimizer, which is a widely used algorithm for stochastic gradient descent. In all MLPs, we employed a softmax activation function in the output layer, enabling us to generate probability distributions across the three classes.

Training and Validation

The models were trained on the training dataset for 10 epochs, utilizing a batch size of 32. Subsequently, validation accuracy scores were obtained for each model. These scores were instrumental in evaluating the performance of each model and selecting the most suitable one for our task.

Performance Evaluation and Comparison

Post-training, the MLP models were assessed on a test set, leading to the following observations:

Model	Train Acc.	Test Acc.	Train Loss	Test Loss
MLP1	0.616392	0.63009	0.768041	0.759842
MLP2	0.598933	0.643665	0.784713	0.755824
MLP3	0.603298	0.607466	0.774615	0.802383

The provided table summarizes the training and testing performance metrics of three multi-layer perceptron (MLP) models, namely MLP1, MLP2, and MLP3. Each model's accuracy and loss are recorded for both the training and testing phases. From the table, MLP2 exhibits the highest testing accuracy of 0.643665, while MLP1 has the highest training accuracy of 0.616392. Additionally, MLP2 shows the lowest test loss of 0.755824, indicating better generalization compared to the other models.

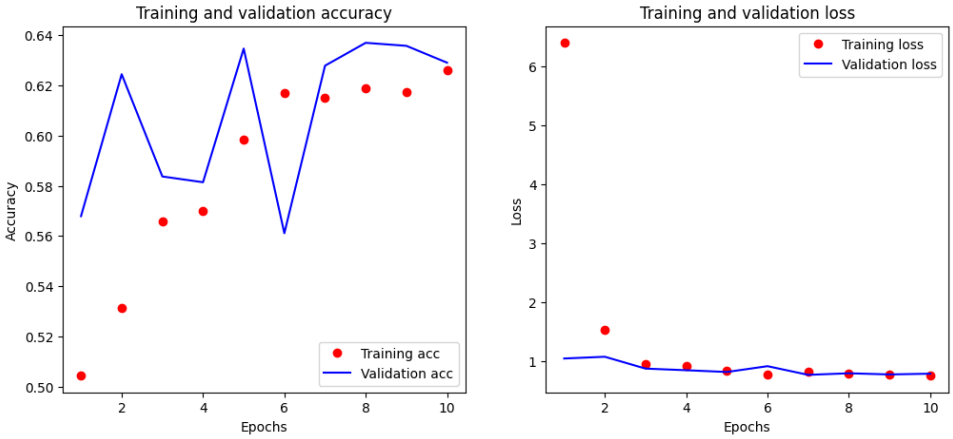


Figure 1 illustrates the comparison of training and test loss and accuracy results for MLP2.

Discussion on Architecture and Parameters

Our experimentation involved testing three distinct architectures, each with varying numbers of hidden layers and neurons per layer. The selection of neuron numbers was made after considering the trade-off between model complexity and accuracy, aiming to strike a balance that captures complex data relationships without overfitting to the training data. Additionally, we conducted experiments with different batch sizes and numbers of epochs to identify the optimal hyperparameters for each model. This iterative process ensured that our CNN architecture was fine-tuned for optimal performance on our accident detection dataset.

We utilized ReLU activation in dense layers for its effectiveness in image classification. Softmax activation ensured output represented a probability distribution over three classes. Training employed categorical cross-entropy loss and the Adam optimizer, renowned for its efficiency in stochastic gradient descent. This choice of activation functions and optimization techniques bolstered the performance and reliability of our CNN models for accident detection.

Convolutional Neural Networks (CNNs)

Model Architecture

For our accident detection project, we built a special kind of neural network called a CNN. Think of it like the image recognition part of your brain, but way more powerful! Here's why CNNs are perfect for this task:

The architecture incorporates several essential layers:

Batch Normalization (BN): Imagine a smoother learning curve for the model. BN helps it learn faster and avoid getting stuck.

Max Pooling: This layer acts like a zoom-out button, shrinking the image size while keeping important details. It also helps the model focus on what truly matters.

Global Average Pooling: This trick condenses the image data into a neat package, making it easier for the final layer to classify it.

Dropout: Picture randomly turning off some neurons during training. This helps prevent the model from memorizing and instead learn to identify accidents in general, not just specific image patterns.

The architecture includes below parameter:

Padding: Maintains output feature map size akin to the input by adding extra pixels. "Same" padding pads input with zeros to ensure output feature map matches input spatial dimensions.

We devised six distinct CNN models, each featuring varying architectures:

Model	Convolutional Layers	Pooling Layers	Other Layers	Output Layer
CNN1	2 (32, 64 filters)	2 (MaxPooling)	2 (Dense)	Dense
CNN2	2 (32, 64 filters)	2 (MaxPooling)	2 (Dense)	Dense
CNN3	3 (32, 64, 128 filters)	3 (MaxPooling)	2 (Dense)	Dense
CNN4	3 (32, 64, 128 filters)	3 (MaxPooling) + GlobalAveragePooling2D	-	Dense
CNN5	1 (32 filters)	1 (MaxPooling)	1 (Dense), 1 (BatchNormalization)	Dense
CNN6	4 (32, 64, 128, 256 filters)	4 (MaxPooling)	2 (Dense), 4 (Dropout)	Dense

The models are compiled utilizing the categorical cross-entropy loss function paired with the Adam optimizer.

Training and Validation

The models underwent training on the training dataset for 10 epochs, employing a batch size of 32. Validation accuracy scores were obtained for each model, aiding in the assessment of their performance. The model with the highest validation accuracy was selected as the optimal choice for the task.

Performance Evaluation and Comparison

Following training, we assessed the performance of the CNN models using a dedicated test set. Upon evaluation, we observed specific outcomes and comparative measures to discern the effectiveness and relative performance of each model.

Model	Train Acc.	Test Acc.	Train Loss	Test Loss
CNN1	0.986906	0.649321	0.082941	1.757216
CNN2	0.980601	0.635747	0.0762	2.171284
CNN3	0.982541	0.661765	0.070057	2.20854
CNN4	0.553346	0.565611	0.814723	0.794872
CNN5	0.990786	0.695701	0.042618	1.084589
CNN6	0.537827	0.556561	0.835174	0.81091

The combined performance evaluation of various CNN models reveals notable differences in their training and testing accuracies and losses. CNN5 demonstrates the highest test accuracy of 0.696, substantially outperforming other models. Its high accuracy is accompanied by a remarkably low-test loss of 1.085, indicating efficient learning and generalization. Conversely, CNN6 exhibits the lowest performance with a test accuracy of 0.557 and a relatively high-test loss of 0.811. Notably,

while CNN4 displays moderate training accuracy (0.553) and test accuracy (0.566), its comparatively high-test loss (0.795) suggests potential issues with model calibration. This comprehensive comparison underscores CNN5 as the most effective model for the given classification task, warranting further investigation into its architectural design and training methodology for insights applicable to future model development efforts.

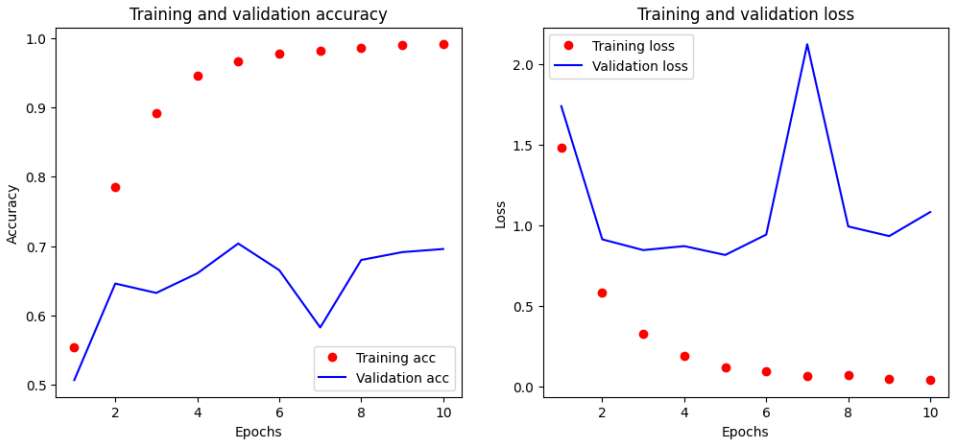


Fig 2. CNN5 Training and Test Loss and Accuracy Results Comparison

Discussion on the Choice of Architecture

The performance and efficiency of Convolutional Neural Networks (CNNs) hinge significantly on their architecture and parameter settings. CNN1 and CNN2 differ mainly in their use of padding, affecting how edge information is handled. CNN3 adds depth with an extra convolutional layer, aiming to capture more complex features. CNN4 introduces Global Average Pooling to reduce parameters and help mitigate overfitting. CNN5 employs Batch Normalization to normalize inputs, speeding up training and improving model stability, leading to the highest test accuracy among the models discussed. Lastly, CNN6 utilizes dropout extensively to prevent overfitting but at a potential cost to learning efficiency. The varied performances underscore the importance of strategic architectural and parameter choices in optimizing CNNs for image classification tasks.

Data Augmentation

The experiments demonstrate the use of data augmentation to improve the robustness of a CNN model (cnn5) against overfitting by applying transformations like rotation, shifts, shear, zoom, and flipping to images. Different strategies were tested, including varying augmentation parameters, using early stopping mechanisms to prevent overtraining, and employing different optimizers (Adam, RMSprop, and SGD). These variations aimed to enhance the model's generalizability, resulting in distinct test accuracy and loss outcomes, showcasing the impact of augmentation and training strategies on model performance.

The results were as follows:

Using different optimizers with a CNN5 model and data augmentation techniques yielded varied performance outcomes. The Adam optimizer resulted in a test loss of 1.25 and an accuracy of 56.1%. Switching to RMSProp improved the accuracy slightly to 57.9% with a test loss of 0.95. The most effective was the SGD optimizer, which achieved the highest accuracy of 58.5% and the lowest test loss of 0.82, demonstrating its efficiency in optimizing the model's performance in this context.

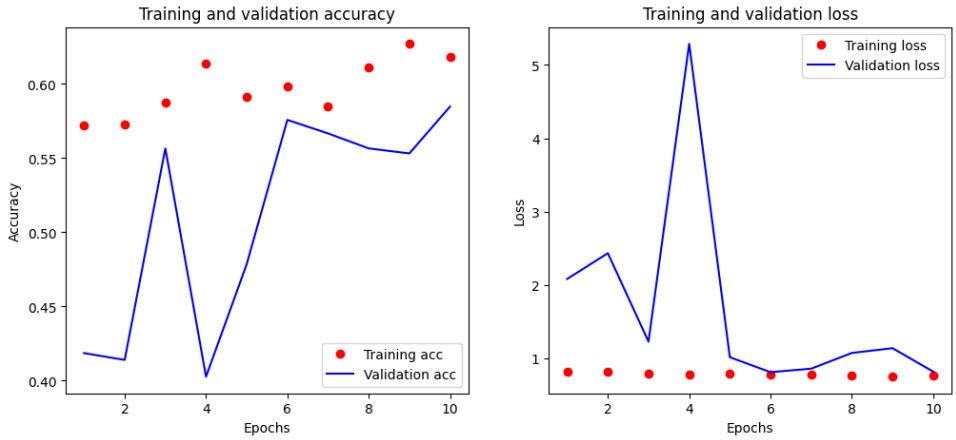


Fig 3. CNN5 with Data Augmentation Training and Test Loss and Accuracy Results Comparison

Interpretability Techniques

Two interpretability techniques for explaining decisions of the best-performing model are Activation Maximization and Deconvolution. Activation Maximization optimizes input to maximize activation of specific units in a deep architecture, providing insight into learned concepts and model generalizability 9. Deconvolution visualizes higher layer features in the input space by running the model in reverse, replacing convolutional layers with deconvolutions and max-pooling layers with unpooling layers 9. These techniques help understand model decisions and visualize important features in the input space.

Note: We defined functions for the two models. Then, we tried to visualise the two techniques but encountered errors and we were unable to visualise them. These errors have been added in markdown in the script for your reference.

Transfer Learning

Transfer learning is a powerful machine learning technique where a model developed for one task is reused as the starting point for a model on a second task. This approach is particularly beneficial in deep learning, where large datasets and extensive training periods are often required. By leveraging pre-trained models such as ResNet50 and MobileNetV2, researchers and developers can achieve significant improvements in accuracy with comparatively less data and training time for their specific tasks. Transfer learning has been instrumental in advancing the state of the art across various domains, including image classification, natural language processing, and more.

We employed transfer learning with pre-trained models:

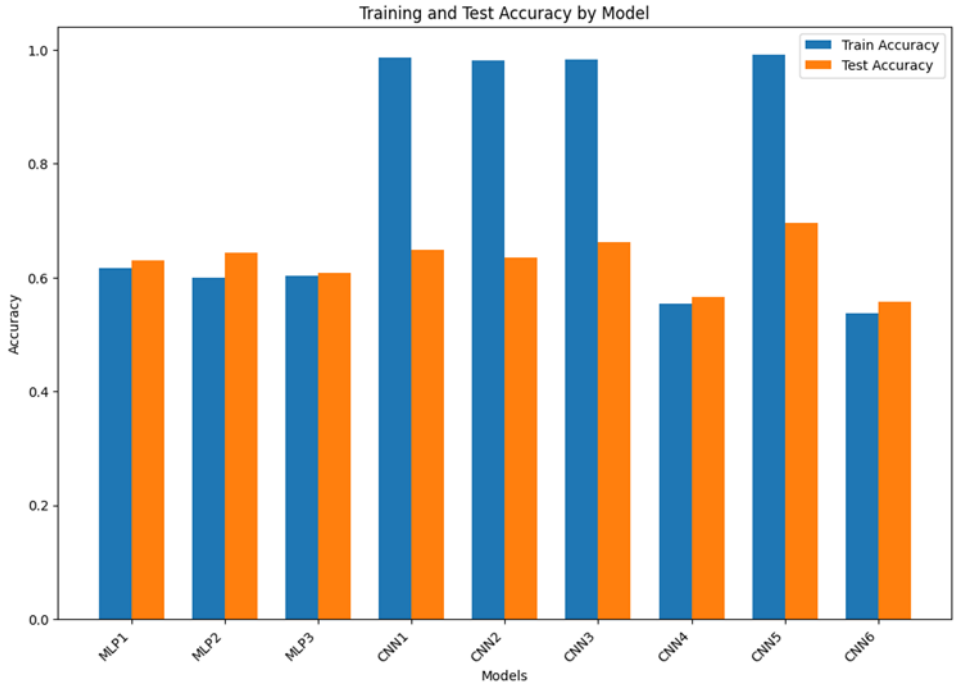
ResNet50

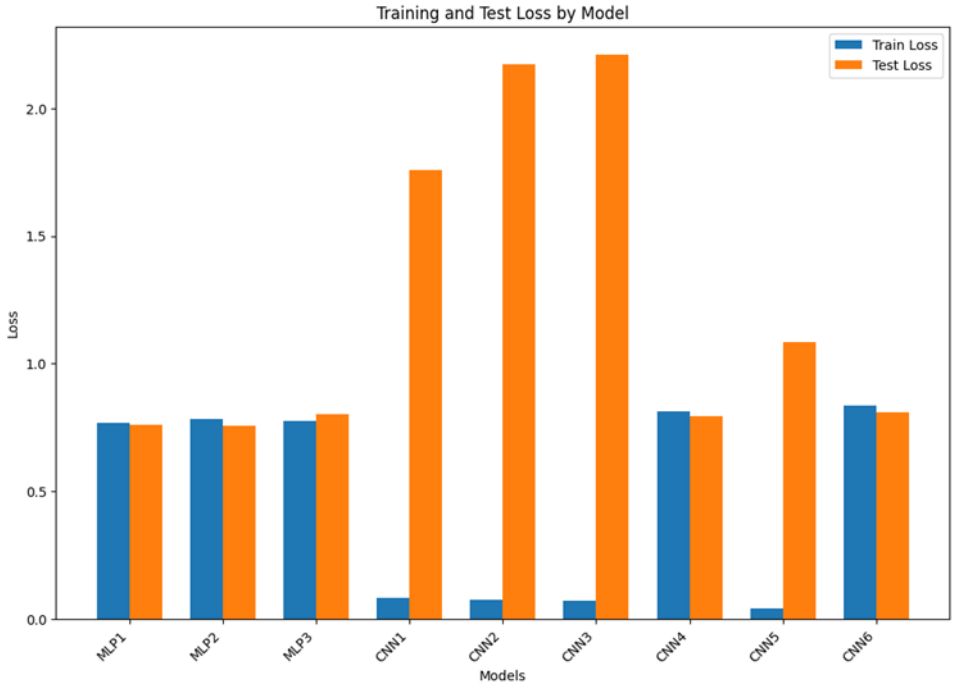
The ResNet50 architecture, known for its deep residual networks, was employed without its top layer to accommodate a new classification task. This model was further customized by adding dense layers and employing data augmentation and early stopping during training. Despite these enhancements, the model achieved a test accuracy of approximately 62%, indicating a moderate level of effectiveness for the task at hand.

MobileNetV2

The second scenario leveraged MobileNetV2, acclaimed for its efficiency on mobile devices, also customized by removing the top layer. Similar enhancements were made, including the addition of dense layers, data augmentation, early stopping, and a learning rate scheduler. This model demonstrated a slightly lower test accuracy of about 62%, closely mirroring the performance seen with ResNet50.

Model	Test Loss	Test Accuracy
ResNet50	0.7901	0.621
MobileNetV2	0.7667	0.6199





The test accuracies of ResNet50 and MobileNetV2 are similar, around 0.62. Despite architectural differences, both models perform comparably on this classification task with similar optimization and augmentation. This highlights the need to explore various architectures and fine-tuning strategies to optimize model performance.

SOURCES

1. https://www.bluecourses.com/courses/course-v1:bluecourses+BC10+2020_Q4/course/
   https://keras.io/api/applications/
2. https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049
3. 2004.14545.pdf (arxiv.org)
4. https://mycourses.ieseg.fr/fundamentals_of_deep_learning
5. https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet
6. Class section 4 journals