CS-7641-ML-Project-Site

AutoSpec

Group Members

Chinmayi Kompella, Jack Keller, Shawn Wahi, Sophie Imhof, Manavi Rao

TA Mentor

Shail Patel

Introduction & Background

Literature Review

A study in Turkey tackles vehicle make and classification issues with an advanced machine learning framework, combining SSD and CNN models [1]. Another paper presents a robust Vehicle Make and Model Recognition (VMMR) system, achieving high accuracy with Random Forest and SVM algorithms [2]. Additionally, another study introduces a novel car make and model recognition method using Inception-v3 neural network on a dataset of Brazilian vehicles [3].

Dataset Description

The dataset contains 16,185 images of 196 classes of cars. Each data point contains the image data, an image id, bounding box coordinates of the image, and a label. The label contains the car's make, model, and year.

Dataset Link

Problem Definition

Problem

Cars are not easily identifiable from first glance, due to distance or lack of knowledge about cars.

Motivation

With cars being integral to daily life, rapid identification offers numerous benefits. For safety reasons, it is sometimes necessary to quickly identify a vehicle from a distance, like in active crime scenes. Car identification can also be used for enforcement of parking tickets, or for customization in car services like a car wash.

Methods

Data Preprocessing Methods

- **Image Resizing and Normalization:** Standardize image dimensions and scale pixel values (0 - 1) to enhance computation efficiency using libraries like OpenCV and PIL.
- **Data Augmentation:** Implement rotation, zoom, flip, and shift through TensorFlow's ImageDataGenerator or PyTorch's torchvision.transforms to diversify the training dataset.
- **Label Encoding:** Convert categorical labels (make, model, year) into numerical format using scikit-learn's LabelEncoder.

Below is a couple examples of image augmentation that we performed to our dataset after resizing and normalization:

Screenshot 2024-04-02 at 3 56 59 PM Screenshot 2024-04-02 at 3 57 26 PM

Additionally, we have included a visualization of the distribution of classes for traning and test images:

Screenshot 2024-04-02 at 3 57 36 PM Screenshot 2024-04-02 at 3 57 43 PM

ML Algorithms/Models

- **Principal Component Analysis (PCA):** Implement principal component analysis (using TensorFlow's linalg.svd function) to perform dimensionality-reduction on the input data for the various supervised models.
- **Convolutional Neural Networks (CNNs):** Utilize TensorFlow or PyTorch to extract features from images through convolutional layers for efficient classification.
- **Pre-trained ResNet-50 Transfer Learning:** Use pre-trained image classification model, ResNet-50, and add downstream layers to fit our problem domain.
- **K-Nearest Neighbors (KNN):** Use KNN for unsupervised similarity finding and pre-clustering, leveraging scikit-learn's implementation.

Unsupervised & Supervised Learning Methods

- **Unsupervised Learning:** Perform principal component analysis (PCA) on the input images in hopes of reducing overfitting and improving testing accuracy via dimensionality-reduction.

Due to compability issues with Tensorflow and Scikit-learn's PCA implementation, we implemented PCA by hand using linear algebra methods including correlation matrices and singular value decomposition. It is important to note that we tried to implement Tensorflow's built-in PCA function, but we ran into version-compability issues with other necessary packages for the project such as NumPy.

To implement the algorithm, we relied heavily on the class homework and lecture slides which covered the topic [4]. Furthermore, we applied PCA on both testing and training images. The steps of our algorithm are as follows, and they were repeated for each image. First, we calculated the correlation matrix of the image data. This was done by first normalizing the data, centering it (subtracting the mean of each feature), and finally computing the covariance matrix (which results in the correlation matrix since the data was normalized). Next, we performed Singular Value Decomposition (SVD) on the correlation matrix. The principal components were acquired by multiplying the U and S matrices together. The first k columns of the V^T matrix were sliced to obtain the principal directions, where k is the number of desired components. Lastly, the principal components and principal directions were multiplied together to obtain the reduced data, which was then reshaped back to the original image size with 3 RGB channels.

Notably, a major limitation with this algorithm was its computational load. In order to perform the algorithm without crashing the program, the images had to be compressed to 32x32x3 pixels, with one channel for each color. Our matrix calculations were broadcasted efficiently, so this is likely due to inefficiencies within the TensorFlow library or simply the computationally expensive nature of PCA.

- **Supervised Learning:** Use a fine-tuned CNN, KNN for primary classification, training on labeled images to accurately classify car features.

For our project, AutoSpec, we implemented three different models to tackle the supervised learning task for image classification on the cars196 dataset (Stanford Cars dataset).

A CNN was selected due to its efficiency in being able to extract features and its ability to learn spatial hierarchies from the images in the dataset. This was crucial due to the fact that proper classification is based on the model's ability to understand the complex visualizations associated with automobile images. Furthermore, supervised learning allows the model to learn from a dataset where each image has a label corresponding to a specific car model. The cars196 dataset is pre-labeled and from the input and output data, the model learns the correlation. Our CNN architecture has several layers: convolutional layers with ReLU activations, batch normalization, dropout layers for regularization, and a global average pooling layer, finishing with a dense layer for the final classification. This architecture was carefully designed to capture the finer details of car images, ensure that the model can generalize well to new car images, and prevent overfitting of the model. Our supervised learning model has benefited significantly from the proposed strategies and techniques for data preprocessing and augmentation. One of the most important data preprocessing techniques is normalization – driven by the need to bring the values of all pixels in images to the range [0, 1]. We have seen that normalization has greatly improved the efficiency of training the model, as it provided the data in a uniform format. The second critical strategy is data augmentation, which includes random flips, rotations, and zooms of the training dataset. The process has proven vital in supervised learning, as it adds variability and diversity to the training images. This operation creates a more comprehensive set of possible inputs that the model will experience, improving its generalization.

The second model we implemented with a pre-trained ResNet-50 model. ResNet is a deep convolutional neural network that addresses the vanishing gradient issue that often arise in deep networks with many layers. ResNet-50 has also been trained on over 14 million images, allowing it to extract features and be succeed at various image classification tasks. We froze most of the model's layer in order to preserve the pretrained weights. We unfroze a number of layers (varying from 3 to 8) at the end to allow for some fitting to our classification task. We fed the output of the ResNe-50 model into a series of Batch Normalization, Dense, Relu activation, and dropout layers to ultimately classify the images into our 196 data classes. After experiencing extreme overfitting to the training dataset, we tuned some of hyperparameters in order to maximize our results. Best results were obtained with a one dropout layer with a 0.5 dropout weight, unfreezing the last 5 layers in the ResNet model, and adding L2 kernal regulization. Additionally, performance was also improved after using the resnet preprocessing methods before feeding our data into the model.

The third model we implemented used KNN methods. Our implementation of the K-Nearest Neighbors (KNN) model for car identification is configured to use five neighbors, a parameter choice aimed at achieving a good balance between accuracy and computational efficiency. In our approach, the KNN algorithm classifies new images based on the majority voting of the labels of the five nearest training images in the feature space. This proximity is determined by calculating the Euclidean distance between feature vectors of the images. To prepare the data for KNN, we first extract features using the convolutional base of the pretrained ResNet-50 model. By removing the top layers of ResNet-50, we capture essential high-level features from images resized to 224x224 pixels. This feature extraction step is critical as it transforms raw image data into a form that is more amenable for analysis and classification by the KNN algorithm. We use the TensorFlow and scikit-learn libraries to efficiently handle these operations, allowing us to process the dataset in manageable batches and optimize the overall workflow. This setup ensures that our KNN model can accurately classify car images by effectively recognizing patterns and similarities among the feature-rich data extracted from the images.

Results & Discussion

Quantitative Metrics

Based on the classification outputs of our models we will evaluate the following quantitative metrics:

- Accuracy
- Cross-Entropy (CE) Loss

Project Goals & Expected Results

- >70% accuracy for car classification

CNN Results & Analysis

Quantitative Metrics

- Training accuracy: 8.44%
- Training loss: 4.42
- Validation accuracy: 4.73%
- Validation loss: 5.09

Our model's relatively poor performance could be due to a number of factors:

- Our model may not be complex enough to fully capture the patterns in the images.
- There might not be enough data to successfully train the model. While we have a large amount of images, there are also 196 classes, leading to potentially not enough data per class.
- Our model might be using ineffective hyperparameters such as the optimizer, learning rate, ect.

Visualizations

accuracy

The model's training accuracy (blue line) is increasing steadily, which is a good sign that the model is learning from the training dataset. The validation accuracy (orange line) is also increasing but at a slower pace and with some fluctuations. This is normal to some extent, as the validation dataset is used to simulate real-world, unseen data. The gap between training and validation accuracy suggests that the model may be overfitting the training data slightly, as it performs better on the training data than on the unseen validation data.

loss

The training loss (blue line) is decreasing consistently, which indicates that the model is becoming better at making predictions on the training data. The validation loss (orange line) decreases initially but then exhibits some volatility, with a significant peak around epoch 10. This peak in validation loss indicates that the model's performance on the validation set worsened significantly at this point, which could be due to the model learning noise or specific patterns in the training data that do not generalize well to the validation data. Despite the spike, the validation loss seems to be decreasing overall, suggesting that the model is still learning useful patterns.

ResNet-50 Results & Analysis

Quantitative Metrics

- Training accuracy: 0.9966
- Training loss: 0.5981
- Validation accuracy: 0.3403
- Validation loss: 3.4677

Our model had a much greater performance compared to our initial CNN model. Some reasons include:

- ResNet architecture is a much deeper neural network, allowing it to capture more complex patterns in our data.
- ResNet addresses the vanishing gradient issue that often occurs with traditional CNN's with many layers.
- ResNet has been pretrained and finetuned on large image datasets, making it easily adaptable to our problem domain.

While the performance was improved, there is still a lot of improvement to be made. One reason is that the architecture is overfitting the training data. We tried to combat this issue by adding more regularization and dropout layers. However, this was met with lower accuracies in the validation data. Further hyperparameter tuning and modifications to the downstream layers should be continued to be fine tuned in order to reach the sweet spot between overfitting and validation accuracy.

Visualizations

Screenshot 2024-04-22 at 9 45 06 AM

The models training and validation accuracy began by steadily increasing. However, the training accuracy increased too rapidly to 99% while the validation accuracy plateaued around 35% after about 5 epochs. The loss followed similar patterns to the accuracy. The training loss decreased steadily towards 0, while the validation loss converged around 3.8.

KNN Results & Analysis

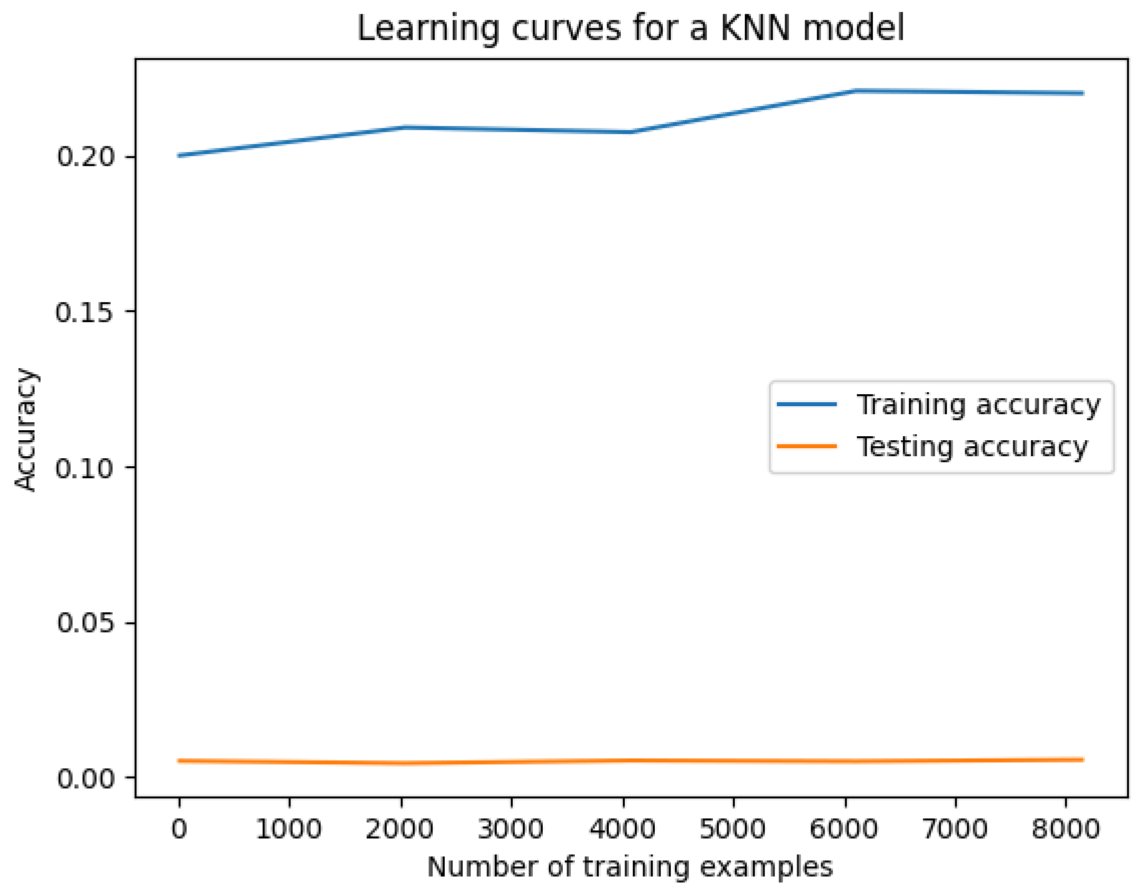
Quantitative Metrics

- Test accuracy: 0.56%
- Validation Accuracy (Cross-Validation): 0.68%

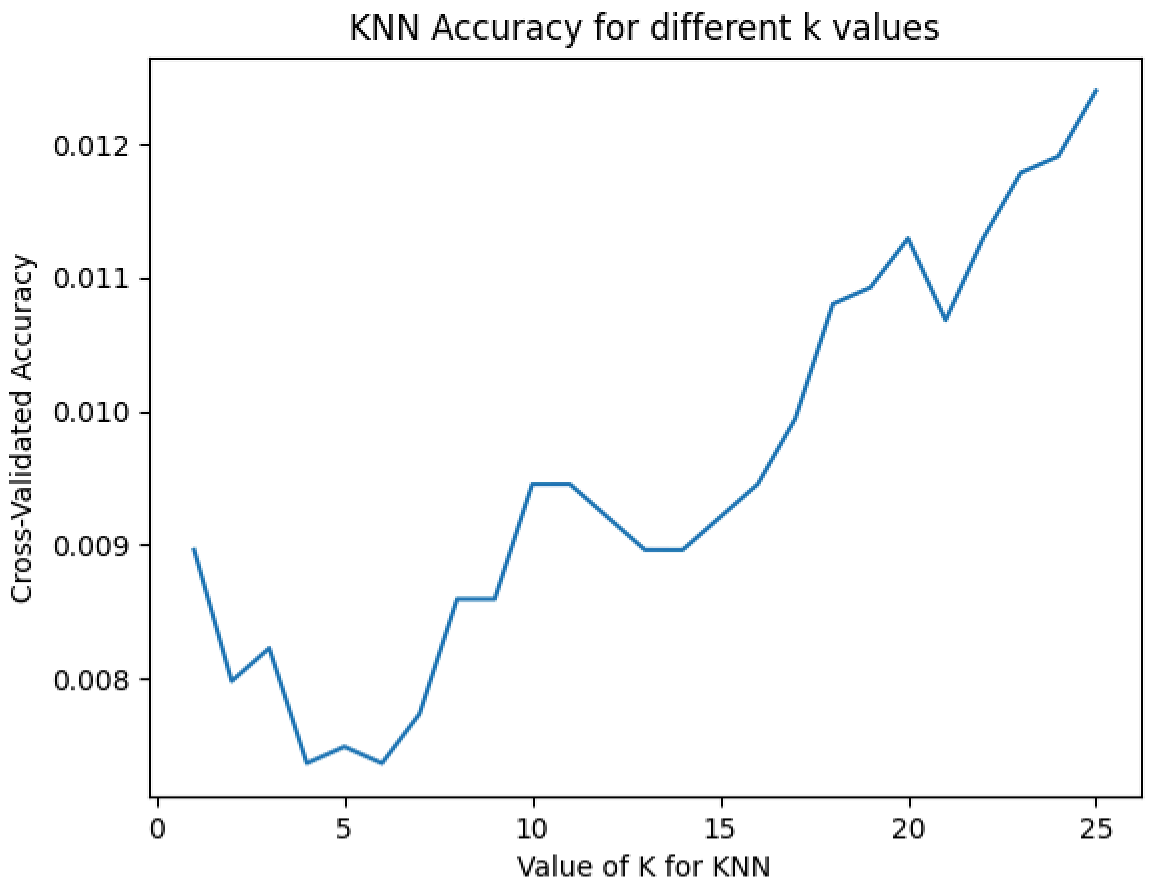
These metrics illustrate a significant challenge in achieving high performance, with both test and cross-validation accuracies being substantially below the project goal of over 70% accuracy for car classification. The low performance could be due to the complexity of the classification task which possibly exceeds the representational capabilities of the KNN model, even when combined with integrated PCA.

The integration of PCA into the KNN framework aimed to enhance feature representation and computational efficiency. However, the low test and validation accuracies highlight significant underfitting, implying that the model fails to capture essential patterns within the data. Despite the advanced feature extraction using the ResNet-50 model's convolutional base and the dimensionality reduction through PCA, the inherent limitations of the KNN algorithm in handling such a complex dataset are evident.

Visualizations



The learning curves display the progression of the model’s training and testing accuracy as a function of the number of training examples used. Notably, the training accuracy remains nearly constant, indicating that the model is not benefiting from additional training data and may lack the capability to generalize well. The testing accuracy also does not show improvement and remains near zero throughout the training process.



The graph shows the fluctuating cross-validated accuracy of the KNN model for varying values of k. While there is an upward trend, suggesting a slight improvement with a higher number of neighbors, the accuracy remains very low, indicating that the model is struggling to effectively classify the car images.

PCA Results & Analysis

Image Transformations

Below are two examples of images from the testing dataset with PCA applied. We applied PCA to both the training and the testing datasets. Throughout the project, we captured the first three principal components and reduced the image size to 32x32x3 (3 RGB channels) pixels before feeding it into the PCA algorithm. Otherwise, the program would crash due to how expensive the computations involved are. We chose to utilize the first three principal components because they transform nicely back into RGB image data.

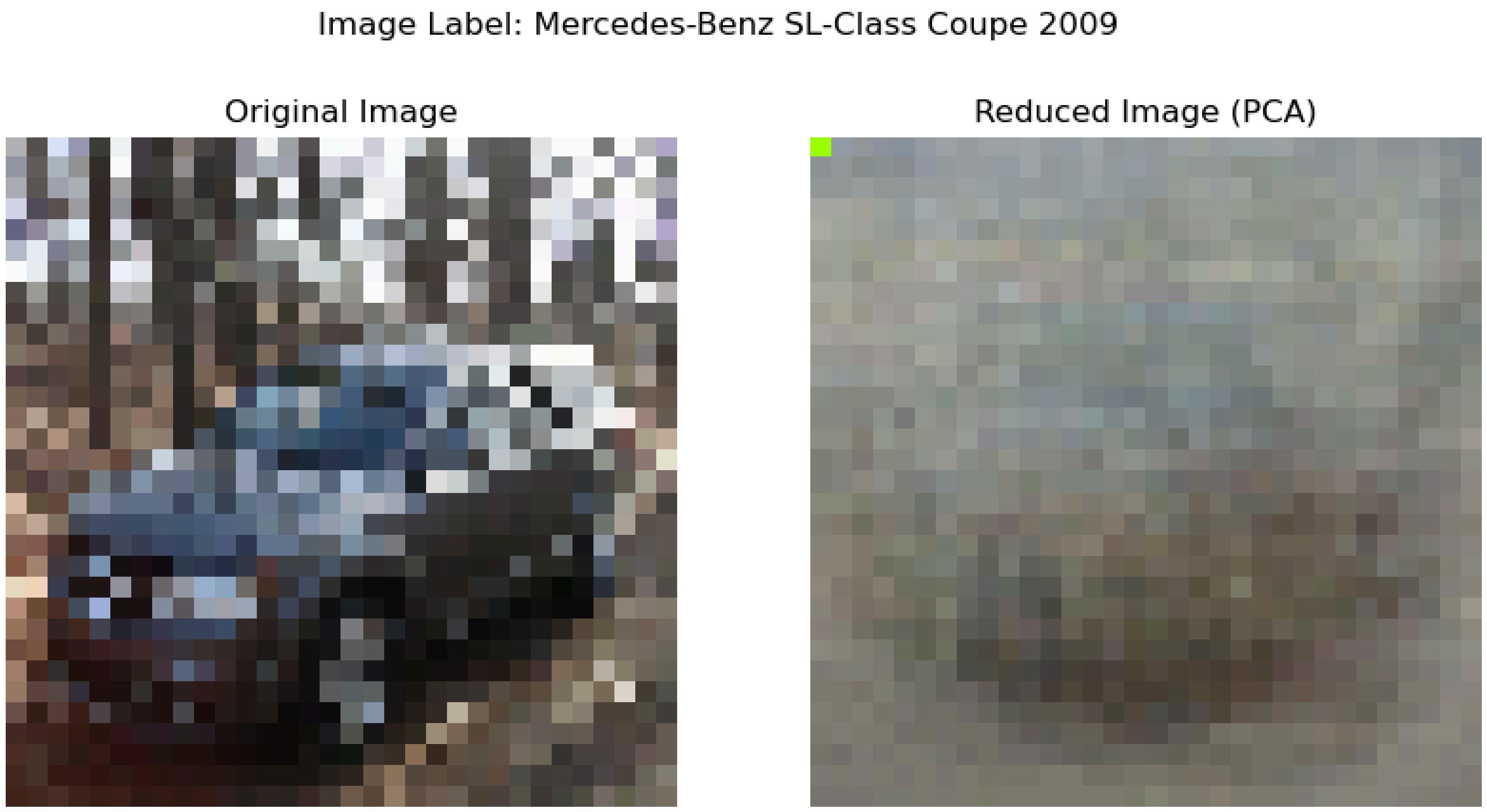
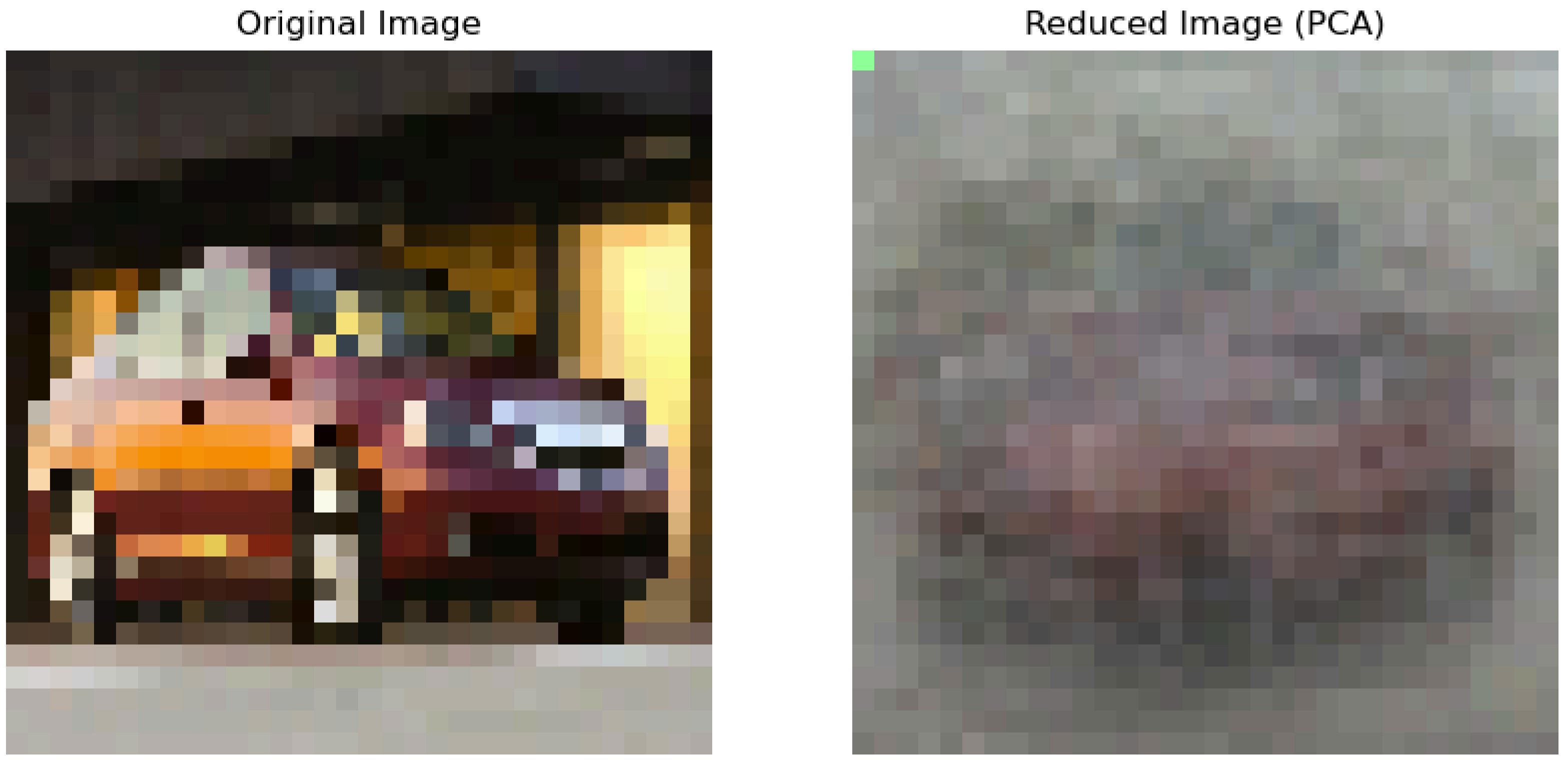


Image Label: Suzuki Aerio Sedan 2007



Quantitative Metrics

We ran PCA on both training and testing images and fed the input data into two different models: our custom CNN and the pre-trained ResNet-50. As mentioned earlier, the images were compressed to 32x32x3 pixels in order to perform PCA. The results for each are described below.

CNN:

- Training accuracy: 0.83%
- Training loss: 5.27
- Validation accuracy: 0.85%
- Validation loss: 5.27

ResNet-50:

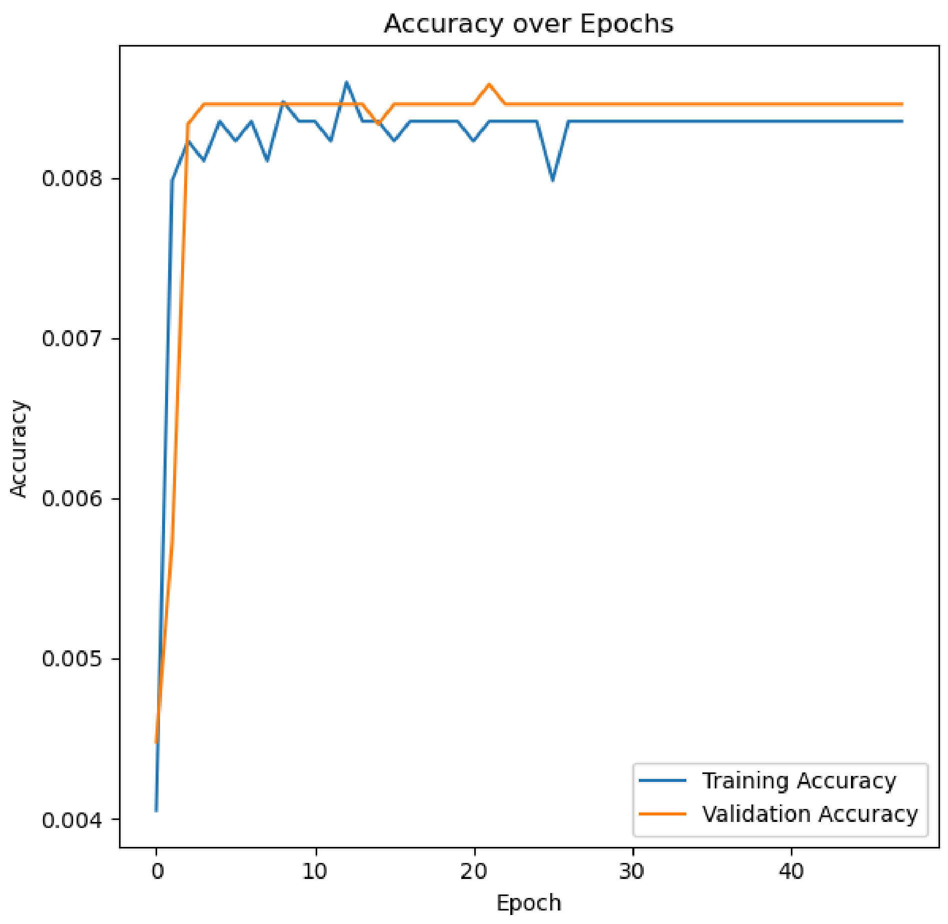
- Training accuracy: 88.06%
- Training loss: 5.90
- Validation accuracy: 88.52%
- Validation loss: 5.89

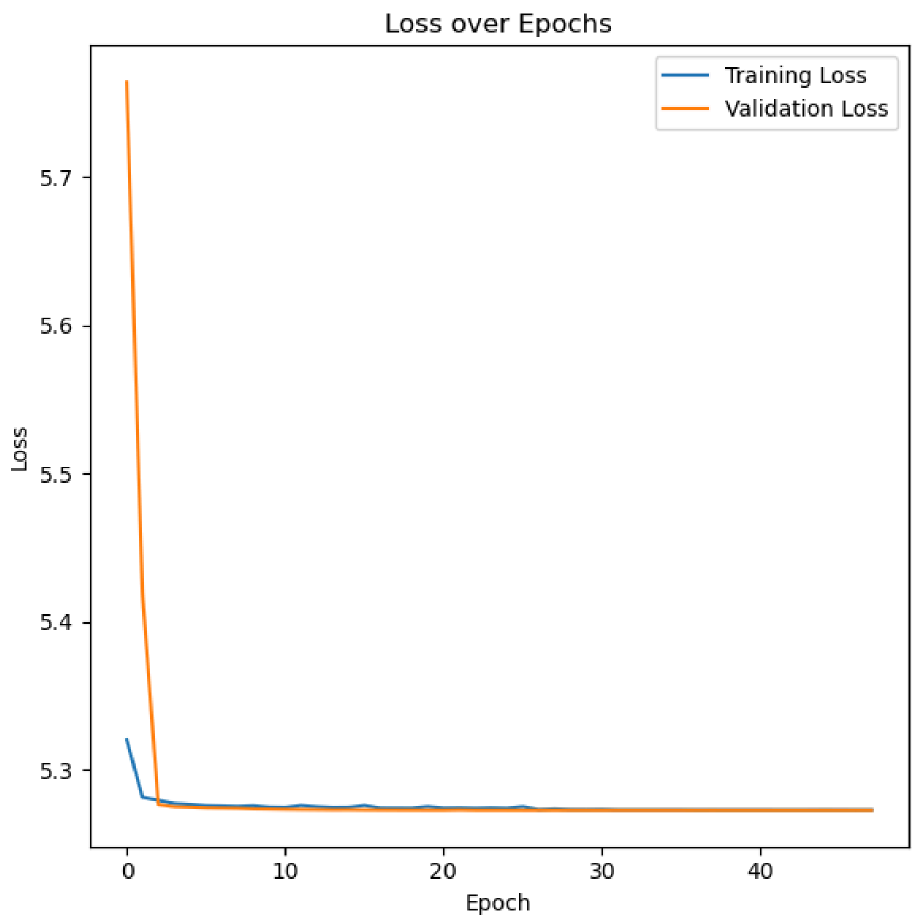
The CNN model performed poorly when combined with PCA-reduced inputs. Both its training and validation accuracies were far lower than the traditional CNN's accuracies. This is likely due to the relatively poor performance of the CNN model on its own combined with the fact that the input images were extremely compressed, which was necessary to perform PCA.

On the other hand, the ResNet-50 model performed excellently when it was fed PCA-reduced inputs. Not only did it perform far better than the CNN-PCA combination, but it outperformed the ResNet-50 model with traditional input data. While the ResNet-PCA training accuracy was less than the traditional ResNet-50 model, the validation accuracy was far higher. With the ResNet-PCA model, the validation accuracy was even slightly higher than the training accuracy. This indicates that feeding PCA-reduced inputs solved the overfitting problem that ResNet-50 experienced. Since PCA applies dimensionality-reduction to the input data, the risk of the curse of dimensionality problem decreases. Therefore, overfitting is less likely to occur, which was verified by the ResNet-PCA model's results.

Visualizations

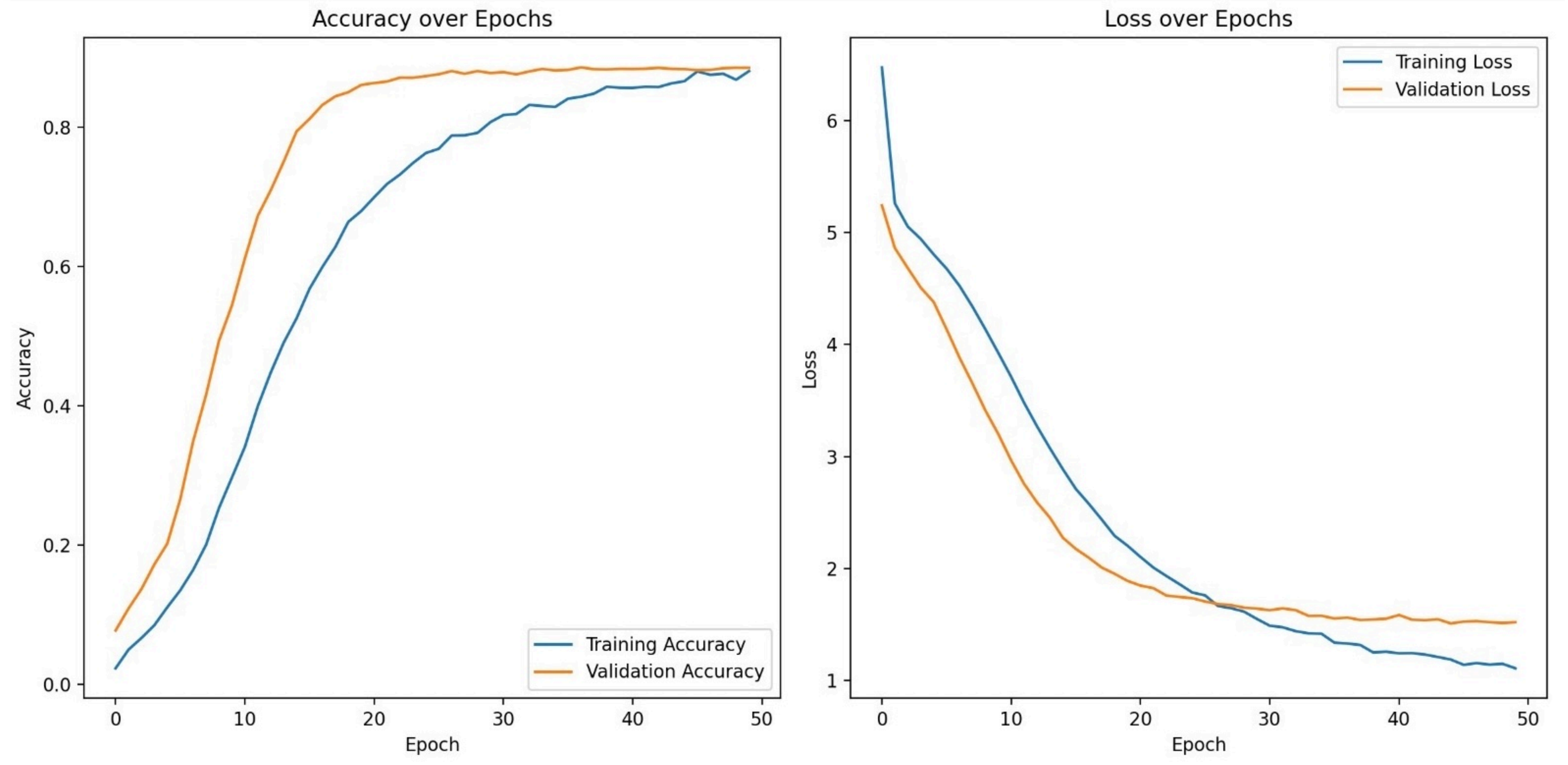
The first pair of graphs correspond to our custom CNN model when fed input data reduced via PCA.





As shown by the graphs above, both the accuracy and loss metrics for the CNN-PCA model plateaued after just a few epochs. Early stopping occurred around 50 epochs. This indicates that the model essentially froze, failing to improve at all. Additionally, both of the training and validation performances were poor. This indicates underfitting. A possible explanation for this is that when the already poor-performing CNN model was fed extremely compressed input data, it was unable to pick up on hardly any variance in the data at all. Thus, resulting in extremely poor accuracy.

The next pair of graphs correspond to the ResNet-50 model when fed input data reduced via PCA.



The graph on the left shows that ResNet-PCA's accuracy steadily increased over time until it eventually reached roughly 88% for both training and validation data. Notably, the graph indicates no signs of overfitting since training and validation runs both achieved the same accuracy. The loss graph on the right exhibits a similar trend: loss decreased steadily until it leveled out for both training and validation data around 50 epochs. At this point, our algorithm noticed that the model was no longer improving, and it stopped early.

Comparison of Algorithms/Models

Comparing the basic CNN and Resnet models, we see that there the CNN model underfits and the Resnet model overfits. The CNN model also has a higher training and validation loss compared to Resnet. After the addition of the PCA model, we saw an improvement in Resnet overfitting, as the gap between the training and validation accuracy decreased significantly. However, with the combined PCA and CNN model, the CNN model plateaus after a few epochs, which indicates that CNN architecture might not be sufficient in other ways. The KNN model has a much lower accuracy than the CNN and Resnet models, and the learning curve is flatter than it was for CNN and Resnet, indicating that KNN did not have significant learning improvements between epochs.

Conclusion

In conclusion, our project AutoSpec successfully implemented and evaluated various machine learning models to tackle the challenge of vehicle make and model recognition using the cars196 dataset. Among the different approaches tested, the combination of PCA with a pre-trained ResNet-50 model emerged as the most effective, achieving a validation accuracy of 88.52%. This hybrid approach leveraged the strengths of ResNet-50's deep learning capabilities enhanced by the dimensionality reduction provided by PCA, significantly reducing overfitting and improving generalization on unseen data.

The implementation of PCA prior to feeding data into ResNet-50 allowed the model to focus on the most informative features of the images, filtering out noise and redundant information which can lead to overfitting. This approach not only optimized the learning process but also enhanced the model's ability to classify a wide range of vehicle models accurately, demonstrating robustness across various vehicle types and conditions presented in the dataset.

Next Steps

- **PCA:** In terms of PCA, there are a few next steps we could take to refine the algorithm. First, it is extremely computationally expensive and dramatically increases the time taken to run the model. To improve this, we could look into saving the PCA output as a separate dataset so that we don't have to run PCA on-the-fly each time we run the model. Additionally, we could determine exactly which matrix operation is crashing the program when input sizes larger than 32x32 pixels are used and better optimize this calculation. Finally, though PCA succesfully reduces the image data, we suspect it may not be processing each color channel perfectly. We could grayscale the images before feeding them into PCA so that this would no longer be a problem.
- **Hyperparameter Tuning:** Employing tools like GridSearchCV, particularly for our KNN and PCA models, can be pivotal. This systematic approach to tuning parameters can help in determining the optimal settings for each model, ensuring that they are configured to achieve the best performance possible. By exploring a range of hyperparameters, we can more precisely tailor the models to our specific dataset and problem.

- **Advanced Feature Extraction:** We should consider utilizing more sophisticated techniques for feature extraction beyond PCA, such as t-SNE (t-distributed stochastic neighbor embedding). t-SNE can be particularly useful for visualizing high-dimensional data and may improve our understanding of the underlying patterns and relationships in the vehicle image dataset, potentially leading to better model performance and insights.
- **Incorporation of Temporal Data:** Introducing temporal aspects to our data—such as using images from different years—could allow our models to capture trends over time. This addition would not only enhance the robustness of the model by providing a more diverse training set but also potentially enable the model to predict future or emerging trends in vehicle features. This temporal dynamic can be particularly valuable in applications like predicting market trends or evolving design preferences.
- **MixUp Training Implementation:** Implementing MixUp training—a method that creates new training examples by combining features and labels of two random data points—could significantly help in reducing overfitting and increasing the model's robustness. This technique effectively regularizes the model by training it on convex combinations of pairs of examples and their labels, thus enriching the model's ability to generalize from its training data to unseen data.

Team Organization

Gantt Chart

Contribution Table

Name	Final Contributions
Chinmayi	Metrics/Evaluation, Algorithm Comparisons
Jack	PCA Implementation, Methods, & Results
Shawn	KNN Implementation, Methods, & Results
Sophie	ResNet Implementation, Methods, & Results
Manavi	Conclusion, Next Steps

References

[1]B. Satar and A. Dirik, “Deep Learning Based Vehicle Make-Model Classification.” Accessed: Feb. 21, 2024. [Online]. Available: https://arxiv.org/pdf/1809.00953v2.pdf

[2]M. A. Manzoor, Y. Morgan, and A. Bais, “Real-Time Vehicle Make and Model Recognition System,” Machine Learning and Knowledge Extraction, vol. 1, no. 2, pp. 611–629, Apr. 2019, doi: https://doi.org/10.3390/make1020036.

[3]L. Albini, M. Gutoski, and S. Lopes, “Deep Learning for Brazilian Car Make and Model Recognition.” Accessed: Feb. 21, 2024. [Online]. Available: https://sbic.org.br/wp-content/uploads/2019/12/CBIC2019-103.pdf

[4]M. Madhi Roozbahani. (2024). “Dimension Reduction” [PDF Document], Georgia Institute of Technology. Available: https://mahdi-roozbahani.github.io/CS46417641-spring2024/course/14-dimension-reduction-note.pdf