Auto Claims Image Classification

CMSE 492 Final Project Write-up

Landon Buskirk

Github Repo: https://github.com/landonbuskirk/cmse492_project

Background and Motivation

Insurance claims processing is the evaluation of an insured damage or loss and the appropriate payment from the insurer. Many insurance companies must manage thousands of claims each year. Furthermore, insurance fraud is a large obstacle for these companies, and so claims often have to be analyzed diligently by hand. As an employee of a large auto insurer, Auto-Owners Insurance Company, I have seen first hand the large volume of human resources dedicated to processing claims by hand. Claim representatives make calls to many parties involved, collect hand written notes and evidence documents into a database, and are often sent to investigate damages in person. Machine learning has a unique opportunity to save claims reps time from sifting through images of car damage. This project aims to create a model to classify client-submitted images of car damage as a common type of auto claim (dent, flat tire, broken glass, etc). Predictions can automate labeling that can then help speed up claims analysis and fraud detection by flagging mismatches or triaging claims.

Figure 1: example of a client-submitted image of vehicle damage.

Other ways to solve this problem include hand labeling (current approach) or a rules-based system. The drawback of hand labeling is clearly the human resource required. I believe an ML approach is much better suited for the task than a rules-based system most importantly because images submitted from client will vary greatly in factors other than the damage type. In other words, it would be incredibly difficult to capture all possible arrangements of images for each class with a set of rules. Machine learning will give us the flexibility needed to make predictions on a wide range of images.

ML task and objective

The project will be a multi-classification project with 5 classes. The ML algorithm will assign a class label to an input image. Because my training set includes class labels, this is a supervised learning task. If this project is used for claims triaging (assigning claims to representatives based on image labels), even a moderate accuracy could help processes. For this reason, I will define a model with test accuracy of 95% and above to be considered useful. Note that I was not able to get access to the HPCC in time for this project, so I will be pursuing an ideal ML approach that can be found with minimal computational power.

Metrics

For this project, I will primarily use accuracy and weighted f1 score. Accuracy is the most intuitive metric for classification and will give us a sense of our error rate (1 - accuracy). Despite this, accuracy can be misleading for imbalanced datasets. Although not grossly imbalanced, the most prevalent class has about four times as many observations than the least frequent class. Weighted f1 score better handles imbalanced classes. Additionally, because type 1 and type 2 errors are equally unwanted in our application, f1 score is a better choice than picking only precision or recall.

Initial and Exploratory Data Analysis

The data for this project comes from the Kaggle dataset, Vehicle Damage Insurance Verification at https://www.kaggle.com/datasets/sudhanshu2198/ripik-hackfest. The dataset consists of 7200 labeled color images and 4800 unlabeled color images. To avoid spending time labeling the unlabeled images, I ignore this data and stick to using the 7200 labeled images. Each image of vehicle damage has one label among 6 classes (crack, scratch, tire flat, dent, glass shatter, lamp broken). Out of the 7200 images, only 171 were labeled crack. Because of the lack of data, and the visually similarity between cracks and scratches, I chose to throw out these 171 images, leaving the analysis with 7029 images among 5 classes.

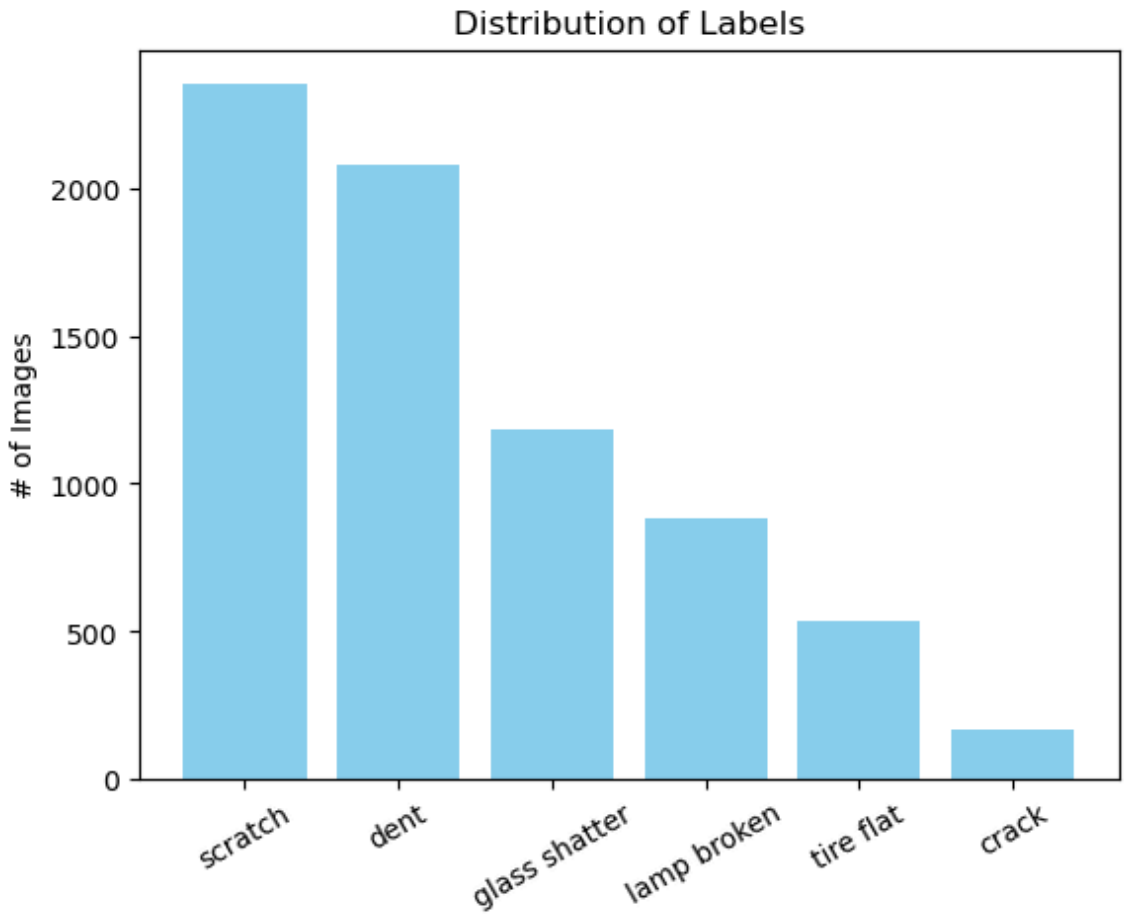

Figure 2: Distribution of class labels in the vehicle damage dataset.

All images varied greatly with respect to resolution and aspect ratio, which we needed standardized for modeling. Also, to reduce training time, I wanted to greatly reduce the total pixel count for images. Because most images were horizontally oriented and around a pixel dimension of 1070 by 800, I decided to transform all images to 200 by 150 pixels. Additionally, I noticed most images had uninformative imagery on the edges, so to further reduce data size I cropped a border of 10 pixels for the images to be 180 by 130 pixels. Finally, because our dataset is imbalanced and has many classes, I performed a stratified split of test size 0.2 to create a training set of 5623 images and a testing set of 1406 images.

To ensure there was not an easy clustering solution to our problem, I ran Principal Component Analysis on our training data. Below, I plot each image based on their first two principal components and color by class. As we can see, no obvious clusters form, warranting a more sophisticated machine learning approach.

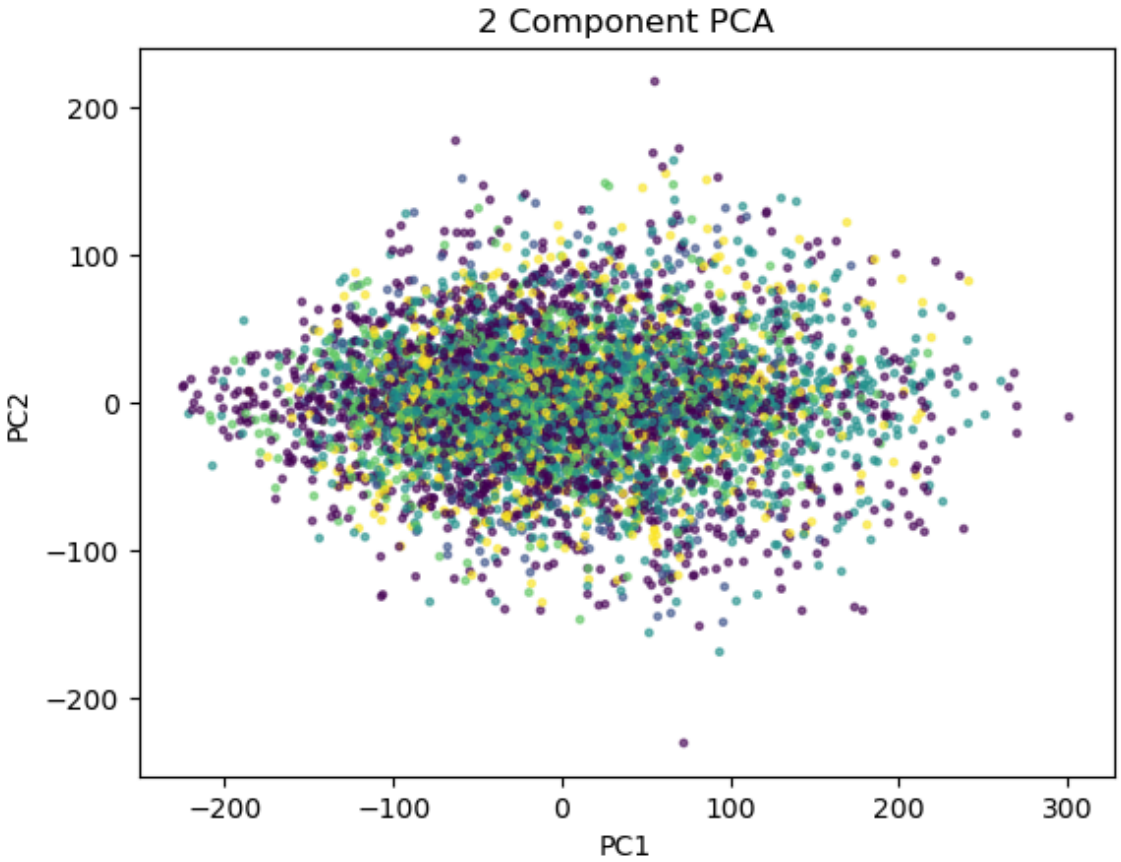

Figure 3: Scatter plot of the first two principal components of each image, colored by class label.

A prominent challenge with this data is the uncontrolled nature of the images. When it comes to angle, lighting, distance from vehicle, location of damage, etc., there is no consistency. Because CNNs look for similar patterns between images, the other varying

factors will likely add a lot of noise for our model to handle. Despite this, I believe this will significantly help achieve our goal, as uncontrolled images are much more realistic to this model's application. A bigger concern of mine is the size of the data. For a noisy deep learning task with 5 classes, 5623 images is likely not enough to reach excellent performance, however, I believe this data will still work great as a proof of concept analysis.

Models

Baseline Models

To establish a performance to improve upon, I chose four baseline models to evaluate on our metrics.

1. **Maximal Class Classifier (MCC):** instead of using random guessing, an unbalanced dataset lends itself better to a simple model that always predicts the most prevalent class in the training data (for our model we predict the class 'scratch').
2. **Multiple Logistic Regression (MLR):** MLR trains coefficients on each pixel to find a logit for each class. The softmax function then is used to predict a probability for each class. The model is quick to train, but lacks anyway to incorporate interactions between pixels into predictions.
3. **Multilayer Perceptron (MLP):** a feed-forward neural network. I chose two hidden layers with 250 and 50 neurons to connect the input layer of 23400 neurons and output layer of 5 neurons.
4. **Shallow Convolutional Neural Network (CNN):** I chose a simple CNN with a convolutional layer with 32 filters and relu activation, a max pooling layer, a 2nd convolutional layer with 64 filters and relu activation, a 2nd max pooling layer, a 128 Dense hidden layer with relu activation, and a 5 neuron softmax output layer.

I trained the baseline models on the training data and recorded the weighted f1 score and accuracy on the testing data in the table below.

Model	Weighted F1	Accuracy
MCC	0.167	0.334
Logistic	0.263	0.278
MLP	0.231	0.318
CNN	0.261	0.297

The MCC has the best accuracy thanks to the large, imbalanced 'scratch' class. Though, the other three models provide a better f1 score which tells us they do significantly

better on the smaller classes. Logistic, MLP, and CNN all perform relatively similar, with marginal improvements. As we can see from observing the coefficients of the Logistic model, these simple models are not able to detect very meaningful insights. The heatmap of logistic coefficients on each class, shown below, mostly appear as random noise.

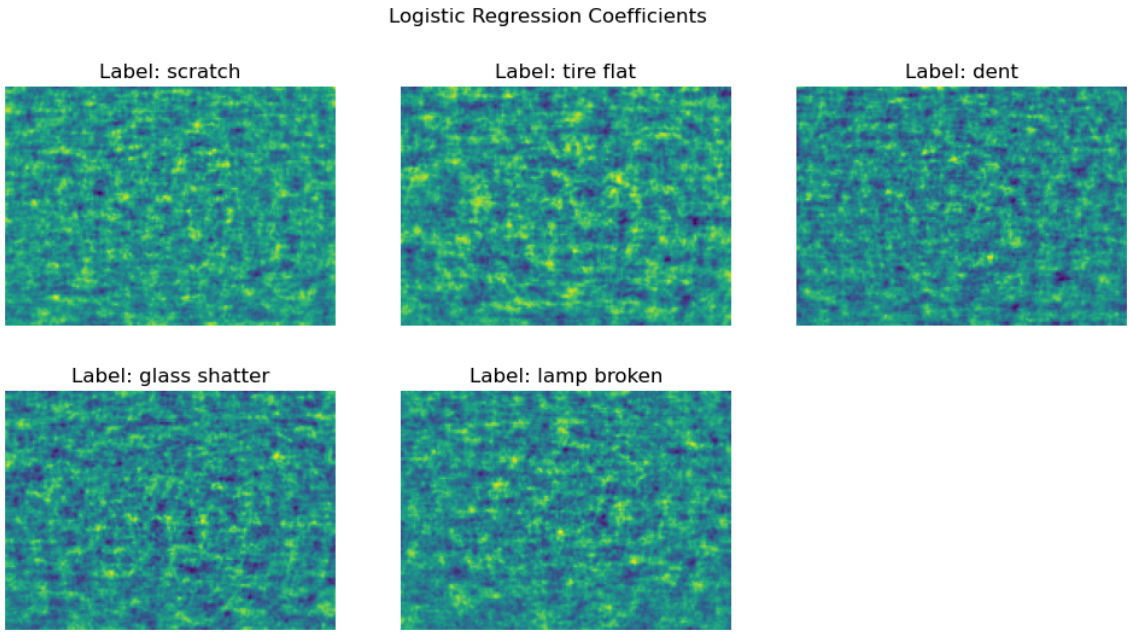

Figure 4: heatmaps of the multi-class logistic regression coefficients for each pixel across each class.

Though, from our metrics, the CNN may have the best combination of f1 score and accuracy. Given that the CNN has the most flexibility for its efficiency, we can likely improve its performance significantly by finding better architecture/hyperparameters. I also have defined a performance goal: I must at least outperform the MCC's accuracy of .334, and I should be able to improve significantly on our best weighted f1 score of .263.

CNNs

CNNs have a large range of possible architectures and can take a lot of time/computing resources to train all convolutional and dense layers. I was not able to receive access to the MSU's HPCC in time for this project, so I was limited to local computing power. This situation could be similar to pursuing proof of concept in academia or industry before requesting expensive resources for a more thorough analysis. Transfer learning is the process of using information gained from a previous machine learning task to inform, improve, and/or accelerate another machine learning task. Instead of varying and training different convolutional layer architectures, I will use well-tested models with pre-trained convolutional layers that generally work well for image classification problems. This will reduce my computational requirement as I will only need to train the dense layers of each model (the feed-forward network that comes after the convolutional and

pooling layers). I will be testing 4 different pre-trained models. To further speed up training, I will be modifying the beginning of each architecture to only take in 1 color channel instead of 3 and passing in my images as black and white.

In order of increasing complexity, these are the four architectures I will evaluate:

1. **LeNet:** simple architecture with two convolutional, two pooling, and two dense hidden layers before the softmax output layer.

Figure 5: diagram of LeNet's architecture.

2. **MobileNetV2**: features depthwise separable convolutions that reduce computational cost and linear bottleneck layers that prevent loss of information during downsampling. MoblieNetV2 is still much more complex than LeNet.

3. **EfficientNetB0**: uses a similar structure as MobileNetV2 with depthwise separable convolutions and bottleneck blocks, but also includes Squeeze-and-Excitation (SE) blocks to enhance feature representation and compound scaling to simultaneously scale input resolution, network depth, and width.

4. **ResNet50**: A deeper CNN built with residual connections, enabling the training of very deep networks. Residual blocks are shortcut connections that help mitigate the vanishing gradient problem. The model includes a total of 50 layers (including convolutional, pooling, and dense layers) making it computationally heavy but highly flexible.

Between the pre-trained architecture and output layer, each model will also include a pooling layer, a 256 neuron dense layer, and a dropout layer. The dropout layer randomly sets 50% of the outputs from the dense layer to zero. By reducing the dependency on any single neuron and introducing randomness in training, this process helps prevent overfitting and improves generalization of our model.

Training Methodology

Because our dataset is imbalanced and has many classes, I perform a stratified split of test size 0.2 to create a training set of 5623 images and a testing set of 1406 images. For my loss function I use categorical crossentropy, also known as multi-class log loss. The categorical crossentropy is defined as

$$L(\hat{y}, y) = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} log(\hat{y}_{ij})$$

where N is # of observations in test set, M is number of classes, y_{ij} is the true label (1 if i belongs to class j, otherwise 0), and $\hat{y}_ij$ is the predicted probability that observation i belongs to class j.

For each pre-trained architecture, I perform 5-fold cross validation, meaning I will split the training set into five equal parts and undergo 5 training iterations where each iteration uniquely has one split that serves as a validation set. I will run each training for 10 epochs, however, to prevent overfitting I will use early stopping with a patience of 3. Early stopping means after each epoch we check to see if the validation loss has improved from the last epoch. If the loss improved, we save the new model parameters, but if 3 epochs pass without improvement to the validation loss, we will end training. For each iteration, the model with best validation loss is then used to calculate a validation accuracy and weighted f1 score. These validation scores for the 5 folds are then averaged together to get final CV scores that we will use for model selection.

For example, below is the training process for the final model (with early stopping patience of 10). We see in this iteration, the best validation loss is reached at epoch 11. We train 10 more epochs, and once we see no improvement, training is concluded.

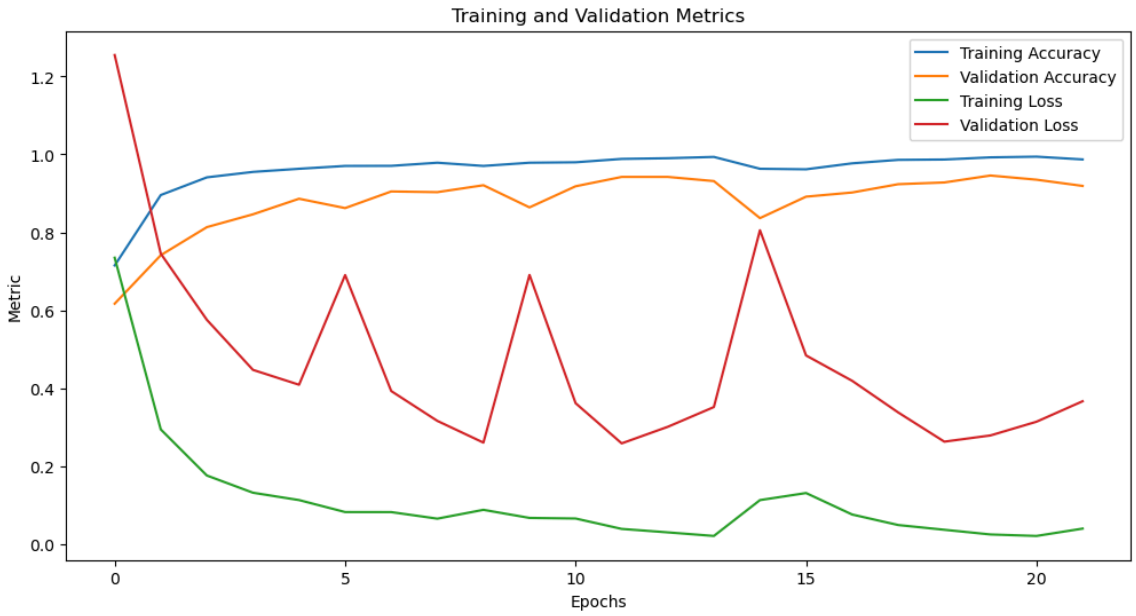

Figure 6: Accuracy and loss for the training and validation set for each epoch of training for the final selected model of EfficientNetB0.

Results and Model Comparison

Model Selection

Below I compare the training time of each pre-trained CNN architecture for 5-fold CV given my somewhat consistent computing resources on my Macbook Air (M2 Chip/24

GB Memory).

Model	Training Time (hr)
MobileNetV2	5
EfficientNetB0	8
LeNet	0.6
ResNet50	14

As we can see, training time correlates well with the complexity of the architecture. Though, the discrepancy is only off by a single order of magnitude, so unless a more simple architecture performs very similarly to one that is more complex, this likely will not be a deciding factor.

Below, I plot the average CV accuracies and weighted f1 scores for each model. EfficientNetB0 is by far outperforming the rest of the models. It may take significant investigation to understand why the other models are not performing as well because of the CNNs are not easily interpretable. MobileNetV2 and LeNet may not be complex enough to learn the patterns necessary to distinguish the car damage types. In contrast, ResNet50 may be too flexible for our small dataset and may be struggling to generalize since its prone to overfit to noise. But it may also not be this simple. Although ResNet50 is a more complex model, the convolutional layers may not be as optimized for our task. Furthermore, EfficientNetB0 is a newer architecture (created in 2019 by a team of researchers at Google AI) I conclude that EfficientNetB0 is the best pre-trained model out of the four for this task, and so we will use it for our final model.

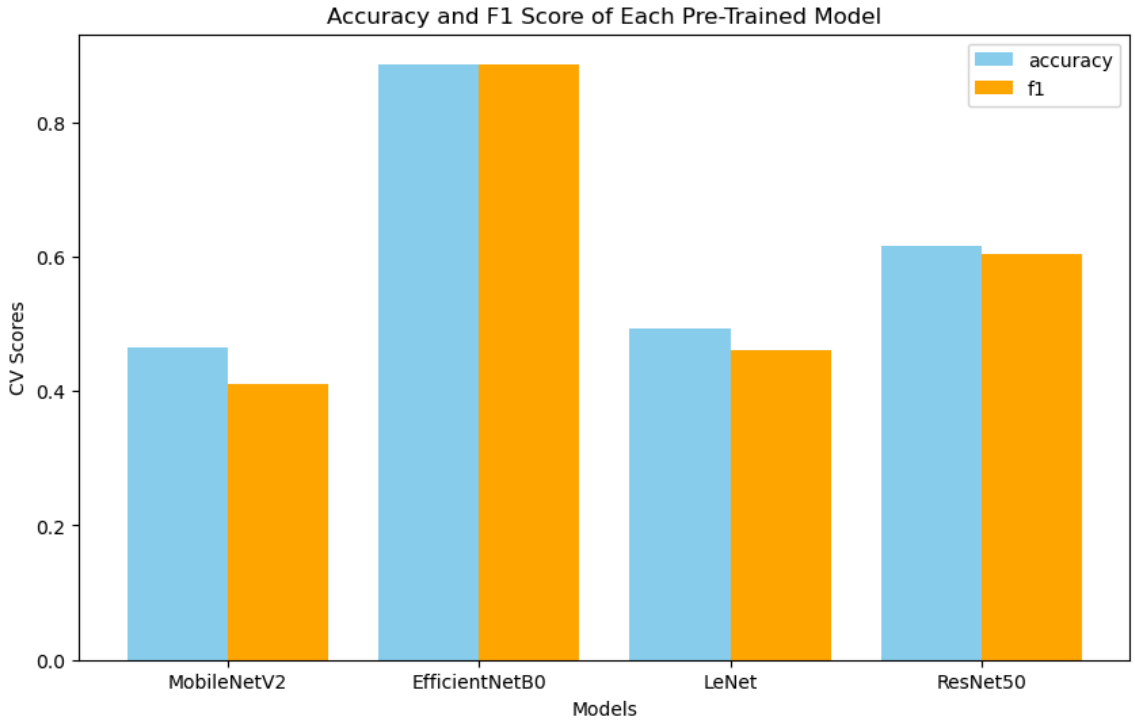

Figure 7: bar plot of average cross-validation fold scores for each model.

Final Model Training

To create our final model, I train the EfficientNetB0 model on the whole training set using a higher epoch count and patience of 100 and 10, respectively. Below, I plot the ROC curves and confusion matrix to show the test performance of our final model on each class

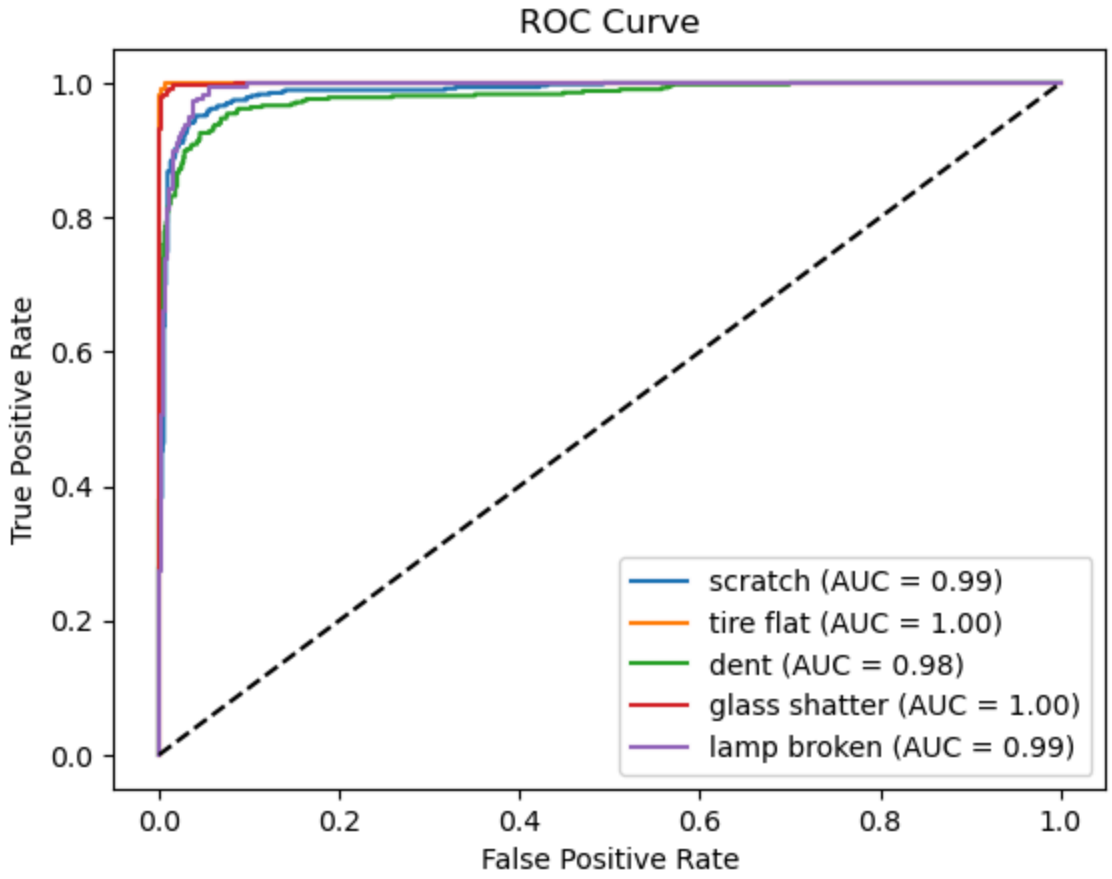

Figure 8: ROC curves showing the tradeoff of true positive and false positive rates grouped for each class label.

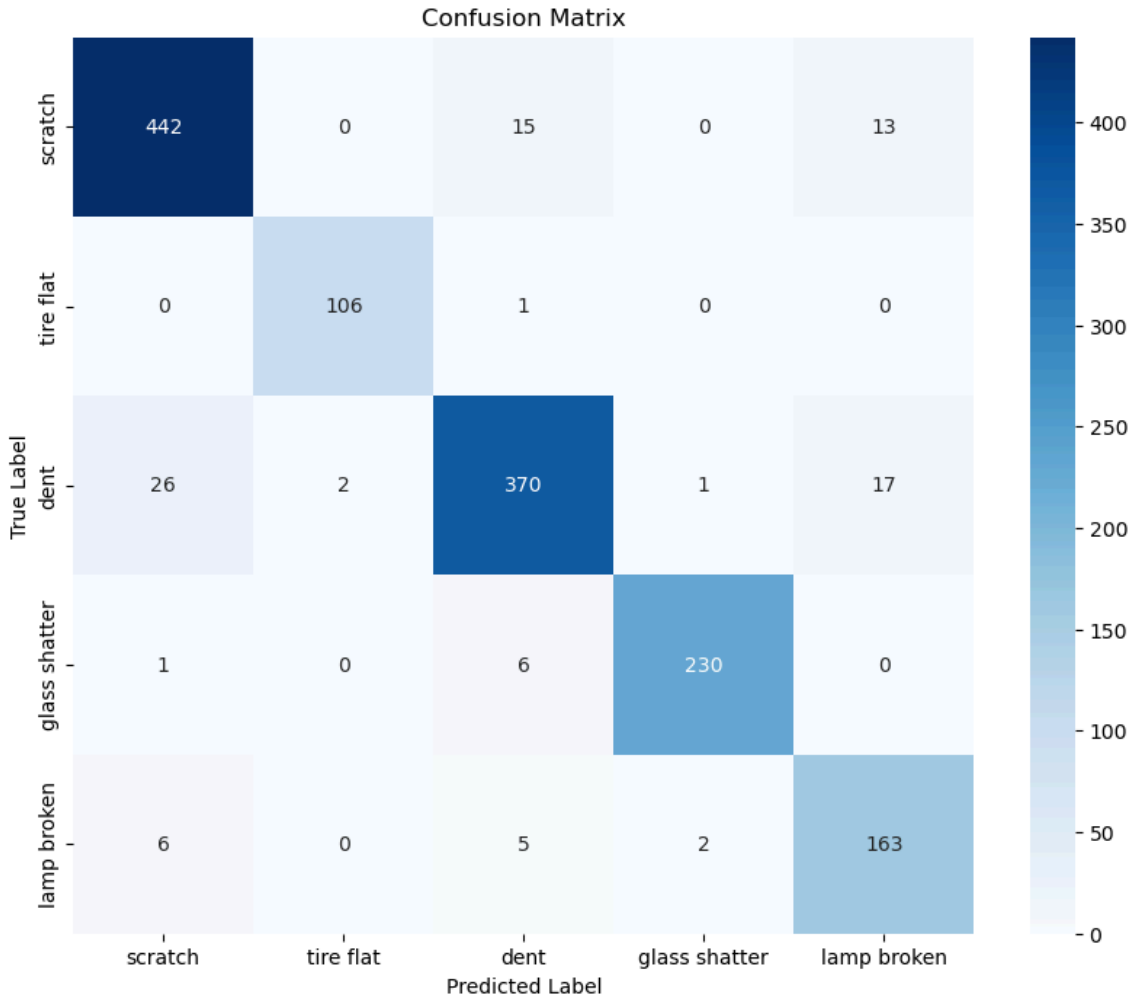

Figure 9: confusion matrix that shows the counts of how actual labels are predicted.

Test Metric	Score
Accuracy	0.9324
F1-Score	0.9326

Our final model dramatically outperforms the baseline models, but falls just short of our target test accuracy of 95%. Although, the model does not perform uniformly across classes. The ROC curves and confusion matrix inform us that tire flat and glass shatter predictions are very reliable, whereas scratch, dent, and lamp broken are commonly confused for each other. The test accuracies of the tire flat and glass shatter classes are close to 99%. If an insurance company has a particular use for identifying these types of claims, this model may be more useful.

Conclusion

Summary

This project set out to leverage machine learning to begin developing an automated pipeline for auto insurance companies. The algorithm will classify user-submitted images of vehicle damage to prevent the need for claim representatives to hand label these documents. I selected 7029 images of 5 different damage types from a public dataset. Then, I reduced resolution and cropped the images to reduce the memory and CPU load from training. After splitting the dataset into training and testing set and selecting accuracy and weighted f1-score as our evaluation metrics, to ensure a more complex ML algorithm was necessary, I tested 4 baseline models. The methods included a maximal class classifier, multi-class logistic regression, multilayer perceptron, and shallow convolutional neural network. All models ended with a test accuracy around 30% and f1-score of 0.25, which is not nearly accurate enough to ever use in practice. Therefore, I moved on to evaluating deeper CNNs, however, because of limited computational power I exclusive used transfer learning by picking 4 pre-trained CNN architectures. Using 5-fold cross-validation to calculate average validation accuracies and f1-scores, I found EfficientNetB0 to perform significantly better than the other architectures. I selected EfficientNetB0 as the best CNN and trained it on the whole training set to then evaluate on the testing set. The final test accuracy and f1-score is about 0.93, which is just shy of my desired score. The model has an accuracy and f1-score closer to 0.99 for the classes of 'tire flat' and 'glass shatter' but struggle more to differentiate between 'scratch', 'dent', and 'lamp broken'. Although I did not achieve my desired score, I do believe my results are somewhat promising for further experimentation on this task.

Obstacles and Future Improvements

The largest setback for this project was the limited computational power. If provided more resources in the future, I would evaluate more pre-trained architectures and perform another round of cross-validation to find ideal hyperparameters/ending architecture (pooling, dense, and dropout layers) for a particular pre-trained model. In addition, I could try different preprocessing, including higher resolution images, varying cropping/resizing, and 3-channel color images.

Once these avenues were exhausted, the new bottleneck would quickly become the small size of the dataset. For example, the best CV fold accuracy for the EfficientNetB0 was about 90%, whereas the final test accuracy was about 93%. Although there will be some randomness, this is a significant jump that was likely aided by the addition of 25% more data (a CV fold was trained on 80% of the training data opposed to the final model which trained on 100%). If implemented for an insurance company, it is very likely they would have a database of previous claims that could be used to create a better performing model.

References

Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning (pp. 6105-6114). PMLR.

ChatGPT (GPT-3.5) – Various insights and technical assistance provided by OpenAI's ChatGPT for code implementation, machine learning techniques, and best practices in Python. OpenAI ChatGPT.

Microsoft Copilot – Assisted with code generation and optimization for certain programming tasks. Microsoft Copilot.