Homework 2: Project Setup and Initial Analysis

Landon Buskirk

1. Project Overview

Auto Claims Image Classification

Repository link: https://github.com/landonbuskirk/cmse492_project

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. This project aims to create a model to classify images of car damage as a common type of auto claim (dent, flat tire, broken glass, etc). Predictions can help speed up claims analysis and fraud detection by flagging mismatches or triaging claims.

2. Project Setup

The directory structure of my project is as follows:

```
cmse492_project/
│
├── data/
│   ├── raw/
│   └── processed/
│
├── models/
│
├── notebooks/
│   ├── EDA.ipynb
│   └── baseline_models.ipynb
│
├── results/
│   ├── figures/
│   └── model_predictions/
│
├── reports/
│
├── src/
│   ├── main.py
│   ├── data_loader.py
```

```
│   ├── train.py
│   └── evaluate.py
│
├── .gitignore
├── README.md
└── requirements.txt
```

Although the raw image data will not be included in the repo due to size, smaller processed images along with their labels are included under data/processed. Investigative work that does not need to be reused will be in a Jupyter notebook in the notebooks directory, whereas more complex CNN training and testing will happen in scripts in the src directory. Predictions, performance metrics, and any produced visualizations or tables will live in the results directory. All dependencies are located in the requirements.txt file.

How to Run the Code

1. Clone the Repository

First, clone this repository to your local machine:

```
git clone https://github.com/your-username/your-project-
name.git
cd your-project-name
```

2. Set Up a Virtual Environment (Optional but Recommended)

Create a virtual environment to isolate the project's dependencies:

```
# On macOS/Linux
python3 -m venv venv
source venv/bin/activate

# On Windows
python -m venv venv
venv\Scripts\activate
```

3. Install Dependencies

Install the required dependencies listed in the requirements.txt file:

```
pip install -r requirements.txt
```

For the most part, this project uses the following python packages: sklearn, keras, numpy, pandas, matplotlib, and pillow.

4. Run the Code

The images under the directory data/processed/ are compressed and ready for model training, however, code that performed this preprocessing can be seen in src/data_loader.py

Running the main script will (eventually) train several CNNs, tune hyper parameters, and save performance metrics in the results folder.

```
python src/main.py
```

5. Deactivating the Virtual Environment

Once you are done, you can deactivate the virtual environment by running:

```
deactivate
```

3. Completed Tasks

EDA

EDA is necessary I first downloaded the train data from my Kaggle dataset because the test set did not include labels and to save time I chose not to hand label these. I graphed a histogram of class labels to see if there were any small classes worth combining or deleting, as well as to better know what performance metrics and training techniques to use if we have imbalanced classes. I found the average aspect ratio to determine the standard image size I will convert the raw images to.

Image Preprocessing

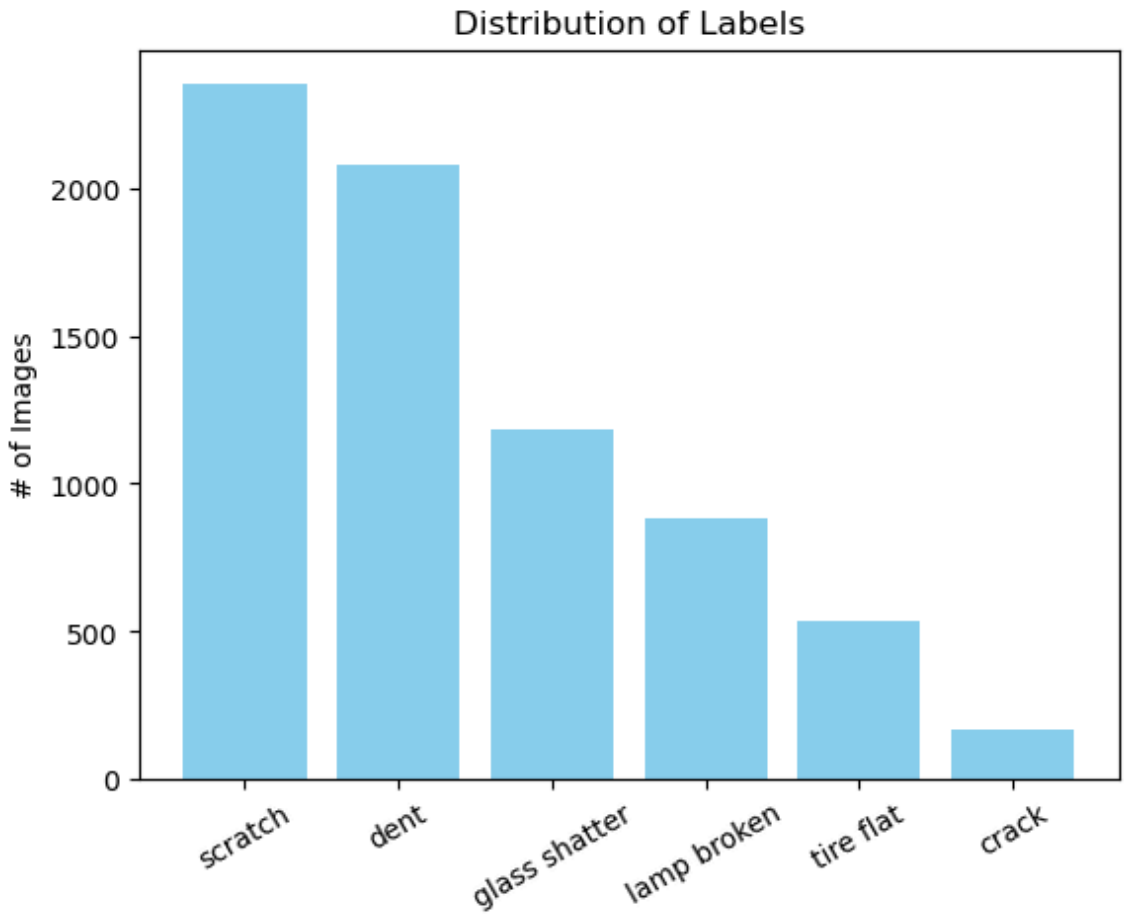
Image preprocessing is necessary to ensure the data passed into our models is best designed to give us desirable training and predictions. I decided to reduce the image sizes to 200 by 150 and crop an outer band of 10 pixels, landing on a final size of 180 by 130. These 23,400 pixels shrink the size as much as possible to assist memory limitations and training speed, all while preserving enough detail for our eyes to understand the image. Next, I convert the RGB images into a single black and white channel to most easily train our baseline logistic regression and MLP. Lastly, I used a stratified train-test split to designate 80% of our images for training, saving the other 20% to measure final model performance.

Baseline Models

Evaluating baseline models gives us a bare minimum performance to hit with our more complicated models. It can also gives us insights in what family of models may be best suited for our problem. For this homework, I decided to evaluate four different models: a majority class classifier, logistic regression, multi-layer perceptron (MLP), and a shallow and quickly trained convolutional neural network (CNN).

4. Initial Analysis and Findings

First, I analyzed the distributions of classes from our image labels, graphed below. Because "crack" has so few images, and the images from this class can not easily be grouped into another class, I've decided to drop these images from my project's analysis. All other labels will be kept, so I will be using 5 unique classes.



Next, I wanted to determine a common size to convert my images to because they all varied somewhat. Below, we can see the average size is about 805 by 1073 pixels. This would be a total of over 860,000 pixels per image, which would make training much more difficult. It is also unlikely that we need this much information to make reasonable predictions. After experimenting with several size reductions, I determined that scaling the length and width by a factor of 5.35 (to kept the same aspect ratio) yielded much smaller images that still had high enough resolution for the human eye to understand what is happening in the image. I also found that important information was rarely at the edge of images, so I crop 10 pixels on each side of the image.

```
sizes = []
for file in labels["filename"].values[:500]:
    img =
Image.open(f'../data/raw/images/{file}').convert('L')
```

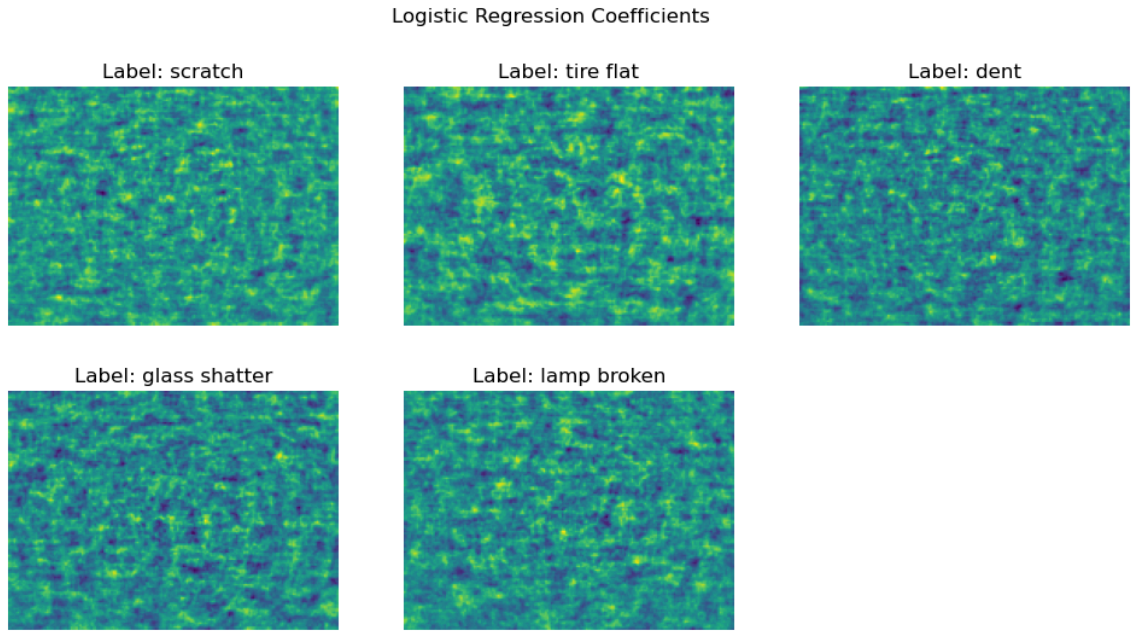
```
    sizes.append(np.array(img).shape)
print('avg img size:', np.mean(sizes, axis=0))
print('reduce length & width by factor of 5.35:',
np.mean(sizes, axis=0)//5.35)
```

```
avg img size: [ 804.558 1073.238]
reduce length & width by factor of 5.35: [150. 200.]
```

Finally, I trained the four 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 Maximal Class Classifier 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 heatmaps of logistic coefficients on each class, shown below, mostly appear as random noise.



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 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.

5. Proposed Approach

I propose to develop and optimize a Convolutional Neural Network (CNN), chosen for this task due to their performance in my baseline analysis and their proven success in image-based problems. CNNs are widely regarded as the most effective and state-of-the-art technique for image classification due to their ability to capture spatial hierarchies in images, automatically learning low-level features such as edges and textures, as well as higher-level features like shapes and patterns.

The following components will be included:

- Convolutional Layers: To extract important features from images using filters that detect patterns, shapes, or textures.
- Pooling Layers: To reduce the dimensionality of feature maps, retaining important features while decreasing computation.
- Fully Connected Layers: To map the learned features to the final output classes.
- Dropout Layers: To reduce the risk of overfitting by randomly ignoring a subset of neurons during training.
- Output Layer: A softmax output layer suited for the multiclass classification problem with imbalanced classes.

To further enhance the model's performance, I will also explore techniques like:

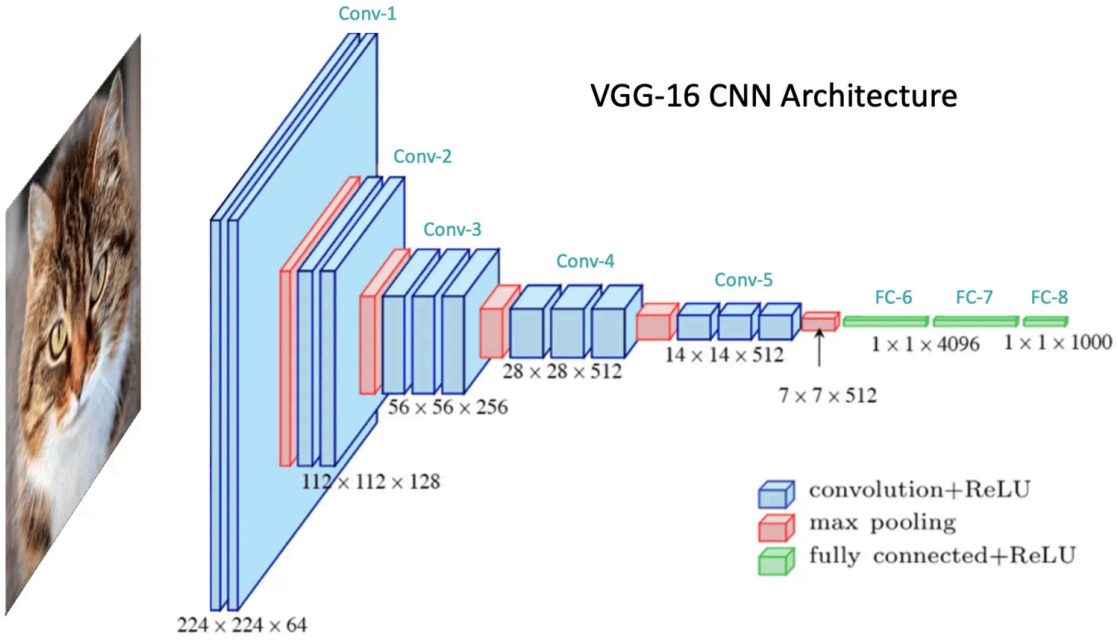
Hyperparameter Tuning: I will apply grid search or random search on the model’s hyperparameters, including the number of filters, filter size, learning rate, batch size, and number of layers, to identify the optimal configuration.
Data Augmentation: Generating additional training data through transformations like rotation, flipping, and zooming to artificially expand the dataset and improve model generalization.
Transfer Learning: Utilizing pretrained CNNs such as VGG16 or ResNet that have been trained on large image datasets to initialize the weights of my network and fine-tune them for the specific image classification task.

The flowchart below outlines my initial proposed CNN architecture:

Input Image (130 x 180 x 1) ---> [Convolutional Layer 1] ---> [Pooling Layer 1] ---> [Convolutional Layer 2] ---> [Pooling Layer 2] ---> [Flatten Layer] ---> [Fully Connected Layer] ---> [Dropout Layer] ---> [Output Layer (Softmax)]

An example of a complex architecture that follows a similar pattern is the VGG16 architecture, displayed below. Notice, each pooling layer reduces the dimensionality until

we hit the last pooling layer or a flatten layer that creates a vector output that can be processed to our final output with a standard feed-forward network.



6. Challenges and Solutions

6.1 Handling High Dimensionality The first major challenge was the high dimensionality of the input data. Each image, after being flattened into a vector, resulted in over 860,000 features, which posed a challenge for both training time and memory consumption. As I began training a simple logistic regression, it was taking over an hour to fully train. Additionally, the large feature space combined with the relatively small number of training samples (5623), will make my modeling prone to overfitting.

To reduce the dimensionality of my data up front, I reduced the resolution of my images dramatically and cropped off edges, which had less useful information. I still, however, have about 23,000 features for each reduced image. Including multiple pooling layers in my CNN will further help. The convolutional layers help extract meaningful features, while the pooling layers reduce the dimensionality of the feature maps, leading to faster training and better generalization. An architecture may look as the following:

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(130,
180, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
```

```
    Dense(5, activation='softmax')
])
```

This architecture effectively reduces the number of parameters and makes the model more manageable, allowing for faster convergence during training.

6.2 Imbalanced Classes

The second challenge is the imbalance in the class distribution. Some categories have significantly fewer images than others, which could lead the model to favor the majority classes during training.

To address this problem, I first discarded a class with too few samples. Without enough images, the model would not have good chance of properly predicting this class, and it may instead interfere with training and predicting for other classes. Still, the 5 classes that I chose to keep are imbalanced. I could try two different techniques during training. First, I can apply class weighting to ensure the model focuses more on minority classes.

```
class_weights = {0: 1.0, 1: 1.2, 2: 0.8, 3: 1.5, 4: 0.6}
model.fit(X_train, y_train, class_weight=class_weights,
epochs=10)
```

Second, I can use data augmentation to generate more samples for the minority classes through image transformations such as rotation, zoom, and flips, which expands the training dataset and improves model robustness.

```
datagen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)
datagen.fit(X_train)
```

6.3 Importing Files from Different Directories

Finally, a more trivial coding problem that I encountered was trying to import my data_loader.py, which lives in the src directory, from my notebooks that are under the notebook directory. I want to keep reuseable functions within the src directory to provide better organization for my project. I thought I was able to write import src.data_loader.py or import ../src.data_loader.py within my notebook, but this and other similar attempts were not working.

I eventually searched how to properly do this. You must adjust your path with sys.path:

```
import sys
sys.path.append('../')
import src.data_loader as dl
```

Lessons Learned

- Dimensionality management is crucial when dealing with high-dimensional data, and CNNs with pooling layers are an effective solution for reducing complexity while preserving important features.
- Class imbalance can severely skew the performance of a model if not properly handled. Both class weighting and data augmentation are useful techniques to ensure that the model performs well across all classes.
- To save time, I should be quicker to search for a coding answer online instead of trying to stubbornly power through and figure out the answer myself.

7. Next Steps

7.1 Upcoming Tasks and Milestones As the project progresses, several key tasks and milestones need to be addressed to enhance the performance of the CNN and ensure its robustness. The upcoming tasks are as follows:

1. Training Code:

- Code reusable files like train.py and tune.py to more efficiently run experiments varying all possible hyperparameters for the CNN
- Create an evaluation pipeline to take generated models and quickly save cross-validation scores to a results table.
- Implement transfer learning to speed up my training processes
- Milestone: Have developed all code to begin training, tuning, and evaluating models

2. Hyperparameter Tuning:

- Fine-tune the architecture of the CNN using more advanced methods such as grid search or randomized search over a wider range of hyperparameters.
- Experiment with optimizers, learning rates, and batch sizes to further improve model performance.
- Milestone: Identify the optimal set of hyperparameters that yield the best validation accuracy and F1 score.

3. Generalization:

- Experiment with techniques like dropout, L2 regularization, data augmentation, class weighting, and batch normalization to reduce overfitting and enhance model generalization.
- Milestone: Implement and evaluate these techniques to minimize the generalization gap between training and validation performance.

4. Reporting:

- Create visualizations that showcase the process of my training, as well as showcase the performance of my best selected model.
- Compile code and tasks into an organized repository and final report/presentation.
- Milestone: Prepare for and complete an effective final presentation

5. Deployment

- Begin the process of preparing the model for deployment, ensuring that it can efficiently handle new image data.
- Milestone: Set up a basic pipeline to allow for model inference on unseen data in real-time.

7.2 Adjustments to the Original Project Plan

Based on the challenges encountered and insights gained from the initial phases of the project, a few adjustments have been made to the original project plan:

- Extended Timeline for Hyperparameter Tuning: The complexity of CNNs and the number of hyperparameters involved necessitate more time for tuning. As a result, more resources and time will be allocated for this phase.
- Incorporating Transfer Learning: Initially, the plan was to solely rely on a custom-built CNN. However, after further research, transfer learning with pretrained networks will be used to speed up training and help performance.
- Success Threshold: While originally defining success as a relatively high model test accuracy, I have realized that this task will not be easy given the amount of data and resources I have. I would redefine my success as significantly improving from the baseline models. This still has the potential to achieve the goal of assisting insurance companies make better triage decisions.

7.3 Timeline for the Next Phase

I have about 5 1/2 weeks until our final project presentations. I expect to follow my next tasks according to the following timeline:

Task	Estimated Timeframe
Training Code	1 week
Hyperparameter Tuning	2 weeks
Generalization	1 week
Project Reporting and Model Deployment	1.5 weeks

8. Conclusion

8.1 Summary of Current Project Status

The project has made significant strides, progressing from exploratory data analysis (EDA) to the implementation of baseline models and the selection of a more advanced deep learning approach. After conducting initial experiments with traditional machine learning models, the decision was made to focus on a Convolutional Neural Network (CNN) for its superior ability to handle image data. Baseline models provided valuable insights into the data and highlighted some key challenges, such as class imbalance and the need for hyperparameter optimization. At this stage, the project has successfully laid the foundation for the development of a high-performing image classification model.

8.2 Reflection on Progress and Lessons Learned

Throughout the start of the project, several important lessons have been learned:

- **The Importance of Exploratory Analysis:** A thorough EDA phase allowed for the identification of class imbalances, image quality issues, and trends within the dataset, all of which were crucial for shaping the project's direction.
- **Handling Large Feature Spaces:** Given the high-dimensional nature of the images, dealing with a large number of features has been a significant challenge, particularly in terms of computational efficiency. This led to exploring strategies like dimensionality reduction and the use of CNNs over traditional models like logistic regression.
- **Model Selection and Performance Trade-offs:** Early baseline models helped demonstrate that while traditional models like logistic regression or MLPs could achieve some improvement on f1 score, CNNs should be more suited to complex image data. This understanding informed the choice of focusing on CNNs moving forward.
- **Challenges of Class Imbalance:** The project has also highlighted the difficulties of dealing with class imbalances in a multi-class setting. Strategies like weighted loss functions, oversampling, and data augmentation techniques will be critical for success in the next phase.
- **Time and Resource Management:** Another lesson learned is the importance of balancing model complexity with computational resources. Training large neural networks can be time-consuming, so optimizing both the model architecture and the workflow is essential to ensure progress within deadlines.

8.3 Outlook for Project Completion

The project is on track to meet its goals, though several key tasks remain to be completed. The next steps will involve fine-tuning the CNN through hyperparameter tuning, exploring transfer learning, and addressing the class imbalance issue. There is a clear plan in place to complete these tasks, and with continued progress, the project is expected to be completed within the next 5 weeks as outlined in the previous section.

Upon completion, the model will be evaluated for both accuracy and robustness across various classes, with a focus on ensuring its generalizability to new data. The final

deployment phase will also involve preparing the model for if it were to be used on real-world use cases.

9. References

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.