# Individual Report

Stephen Davanzo

**DATS 6303**
**Prof. Amir Jafari**
**Group 3**

# 1. Introduction

Group 3 used an iterative approach to improving the code from our baselining model. Everytime an advancement was made, the team would weigh-in and someone else would build off of it. In this way, our progress followed a linear progression, with certain people working on more of the code at one than than the others.

Improving classification power on the TrashNet dataset was truly a team effort. In this individual report I will explain my own contributions.

# 2. Contributions

## a. Initial Research

When looking for a dataset on Kaggle, I came across the TrashNet dataset. I am passionate about renewable technology, so I thought working on a project with an environmental focus sounded fun. I found the "*Classification of trash for recyclability status*" paper written by the original data creators and was shocked at how little they had done to improve their CNN. I saw it as an opportunity to make a meaningful contribution to the research or classifying recyclable goods, by attempting to improve their work by using transfer-learning.

Surely, we were not the first to think of this improvement. The data on Kaggle had many contributions, so I decided to find effective, but modest code to serve as our baseline. The baseline compared six different models side-by-side, and found acceptable results. With little tuning being done to the models, I knew that we could contribute.

## b. Repo and Data

After getting approval from my team members that they liked the idea of TrashNet and baseline code, I sought out to build the group GitHub repository.

Even though our data was small enough to fit in GitHub, I knew for the purposes of the project that it had to be hosted in a more professional environment. I taught myself how to use Google Cloud Platform (GCP) and put the entire dataset in a GCP bucket with a public URL. I then set up the git repository with a ```.gitignore``` feature if it sees a directory called "Data/". This allows users in the repository to download the data once, and are free to push, pull and change branches while the data stays put and is not tracked by git.

# c. Initial Baseline

I turned my attention to getting the baseline code running in the EC2 instance. Only minor changes were needed to get it running. The results were similar to the notebook found in Kaggle.

| | Model | Test Accuracy | Test Loss | Train Accuracy | Val Accuracy |
|---|---|---|---|---|---|
| 0 | ResNet152V2 | 0.7913 | 0.6628 | 0.9708 | 0.7639 |
| 1 | ResNet101V2 | 0.7734 | 0.7330 | 0.9395 | 0.7638 |
| 2 | MobileNetV2 | 0.7555 | 0.7577 | 0.9355 | 0.7163 |
| 3 | MobileNet | 0.7495 | 0.6749 | 0.9082 | 0.6683 |
| 4 | MobileNetV3Large | 0.4473 | 1.4096 | 0.5645 | 0.3353 |
| 5 | MobileNetV3Small | 0.3579 | 1.5780 | 0.4083 | 0.2361 |

# d. EfficientNetV2S

After a few interactions of improvement on the baseline code, we saw improvements, but not drastic ones. We had agreed to find new models to compare against the others, so I spearheaded the research effort to find a high performing model on a small dataset. After finding EfficientNetV2S and reading some promising articles, I tested it and achieved ~81% accuracy.

The team was in agreement that this model was a good idea, so I set out to learn everything I could about it, and learn why it worked so well. This research led me to learn about MBConv/fused-MBConv layers and some of the advanced regularization techniques that it employs. I also found myself learning about parameter heavy models, and how we needed to strike a balance between efficiency and depth to fit our data. All research concluded that this model was the best for our use case.

# e. Weights and Plots

Much of the model tuning had been completed and agreed upon by this point. I decided to add confusion matrices and noticed that our minority classes, most “Trash”, were being misclassified and hurting our score.

I use sklearn’s ```compute_class_weight module``` to attempt to balance the majority and minority classes, hoping it would help the model. The initial weighting gave too much weight to “Trash”, and the model was not effective at classifying the remaining classes.

The solution was to add an adjustable weighting function that allowed me to pick an alpha level being 1 and .1. The alpha level .2, was a good balance between help the minority class while preserving the majority. Resultant class weights are (5 is “Trash) :

**Class Weights: {0: 1.0090157154673283, 1: 0.9681304058549568, 2: 1.0054471544715446, 3: 0.9418069584736252, 4: 0.9747579529737206, 5: 1.4148418491484185}**

After fine tuning the weights, I added the learning curves to help diagnose overfitting. With the benefit of the visualizations. We were jointly able to agree with further fine tuning that ultimately resulted in the success of our project.

# 3. Results and Conclusions

Our team's results were a dramatic improvement in the original research paper and baseline code.

| Model | Our Result | Baseline Result | Difference |
|---|---|---|---|
| EfficientNetV2 | 86.25% | / | / |
| ResNet152V2 | 80.62% | 78.93% | 1.69% |
| ResNet101V2 | 80.21% | 77.14% | 3.07% |
| MobileNetV2 | 77.29% | 75.35% | 1.94% |
| MobileNet | 75.21% | 75.15% | 0.06% |
| MobileNetV3Large | 44.79% | 44.53% | 0.26% |
| MobileNetV3Small | 26.04% | 35.59% | -9.55% |

My individual experiments led to the implementation of our most successful model ,identified roadblocks, implemented solutions, and helped fine-tune the final result. I feel strongly that the knowledge gained during this past semester led me to be able to achieve the previously stated outcomes. I would not have been able to read the EfficientNet research paper with a keen eye, or effectively diagnose the problem of a neural network base on the learning curve without the skills I gained in this class.

# 4. Code %

146-12/146+60 = 65%