Multi-Label Waste Classification with Data Augmentation

Tom Welch and Max Comolli
tcwelch@stanford.edu, mcomolli@stanford.edu

Introduction

Our project aims to use transfer learning in order to classify different types of waste. We were motivated to do this with the ultimate goal of improving the efficiency of waste management systems, eventually reaching the point where people do not need to sort their waste, and an autonomous system can process it according to what type of waste it is. The end goal is to have a model that distinguishes recyclable waste from garbage, and labels recycling according to five classes, which are described below in the dataset section. Such a model could be used either in waste management facilities or in households to help educate people on what can and cannot be recycled. The input to the algorithm is a set of images, and the output is predictions corresponding to one of six classes of waste.

We utilized transfer learning from a well-known neural network architecture called DenseNet121 [15], which has enabled us to take advantage of a model trained on a wealth of data, which benefits us as our dataset is comparatively small (2,527 images before data augmentation). We also utilized data augmentation in an attempt to improve our accuracy.

Related work

In our research, we came across a past CS230 project by Hrushikesh N. Kulkarni [2] which utilized transfer learning from ResNet50 (He et al. 2016). Another paper by Adedeji et al achieved 87% accuracy on a similar dataset (called trashNet) also using transfer learning from ResNet50 [4].

We also identified a paper by Wei-Lung Mao, Wei-Chun Chen, Chien-Tsung Wang, and Yu-Hao Lin with a similar objective (multi-label waste classification) that utilized transfer learning from DenseNet121 [3]. We decided to implement transfer learning with both of these structures based on the success of these two experiments, as both seemed to perform well on large datasets for multi-label waste classification. However, our results, as discussed in more detail below, indicated that DenseNet121 performs better than ResNet50, and we chose this model to proceed with our further experiments.

Dataset and Features

We used a garbage classification dataset with examples of recyclable and non-recyclable waste from kaggle.com [1]. The dataset contains 2,527 images, each labeled as one of six classes: cardboard, glass, metal, paper, plastic, or garbage. Our training set consists of 1,768 images, our validation set is 328 images, and our test set is 431 images. Each image has a similar white background, which may pose an obstacle to training our model for generalizable classification with other types of backgrounds.

Upon initial training we notice some particularly challenging training examples as seen in the images below. One visual commonality is that many images across classes share the trait of having text, barcodes or nutrition labels. Our initial training on the imbalanced data set also showed high confusion rates between glass and plastic (see Fig. 6), which might be explained by the fact that a plastic bottle and a glass bottle look a lot alike. We approached this problem by balancing our data set and increasing the total number of images from their initial breakdown - 354 glass, 403 paper, 287 cardboard, 347 plastic, 286 metal, 91 trash images - to 1000 of each using data augmentation. To augment our data, we randomly chose examples and then randomly chose to flip up-down or left-right or rotate the image by three 90 degree increments. It goes without saying that a flipped or rotated image of a piece of trash is still recognizable as a piece of trash.

Figure 1 - Examples of challenging images from each class [1]

Methods

We utilized transfer learning in modelling trash classification. Transfer learning allows us to take advantage of computationally expensive training done previously on related data and add on that to learn a classifier specific to the six trash classes. As discussed in the related work section, we tried out both ResNet50 and DenseNet121. ResNet50 performed significantly worse than DenseNet121 in initial testing, see comments in the results section, and so we primarily focused on DenseNet121. DenseNet121 follows a similar architecture to most CNN's, convolution and pooling layers that decrease in width and height but grow in channels with depth, with the exception that every layer is fully connected to subsequent layers. This advantage allows deeper filters, that make decisions on higher order classifications, to have access to all prior lower level attributes discovered. After processing data through the pretrained DenseNet121 we then apply our own deep neural net to the trash identification problem.

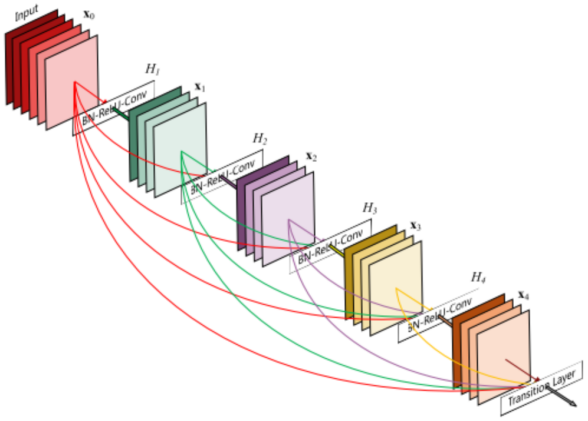


Figure 2 - DenseNet121 Architecture [7]

In our implementation we pass our images through the transfer network and take the output from the last fully connected layer before the softmax of the transfer network to be the input to our own deep neural network. Our network followed the convention of layers with decreasing numbers of nodes and had layer dimensions: 1024; 256;64;16;6 - where the final layer was a six class softmax. We trained a deep neural network in keras with an Adam optimizer, to give our algorithm momentum and make it less stochastic, on the categorical cross entropy loss:

$$Categorical\ Cross\ Entropy\ Loss\ =\ -\sum_{i}^{C} y_i log(\widehat{y_i})$$

Here, y_i is the true label and $\widehat{y}_i$ is the softmax output for class 'i'. We initially encountered problems with vanishing gradients. To address this problem we added glorot uniform random kernel initialization so that neuron parameter weights in the same layer would not be the same on each epoch from zero initialization. We also noticed that after many epochs the loss remained relatively unchanged with a training accuracy near 20%. We addressed these issues by normalizing the input to both our transfer network and our own trained network. This trick allows us to stay near the middle of the multi-class sigmoid function, the softmax, where the gradient is highest allowing gradient descent to converge more quickly. We additionally did some fine tuning to find that batch sizes of 25 enabled for a more stochastic approach since we noticed that it allowed our loss to make more periodic improvements jumps. Finally, as discussed in the results section, we noticed a high variance between train and validation sets, so we added L2 regularization with $\lambda\ =\ 10^{-4}$ to prevent overfitting. We noticed that making λmuch larger slowed learning tremendously, taking an additional 10 or more epochs to make the same progress. This is likely because it inhibited the weights from changing each epoch. With these changes our algorithm performs well and converges quickly, within five epochs (see Fig. 3).

When implementing data augmentation we tried a number of scenarios which focussed on adding images only to the classes with the highest confusion and balancing the entire dataset with 500 and 1000 images for each class. We found that balancing the dataset with 1000 images gave the best performance across multiple metrics.

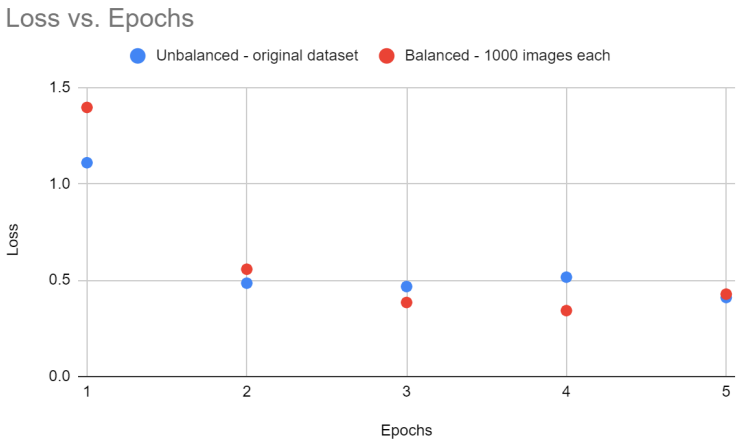


Figure 3 - Loss vs. Epochs. This figure demonstrates the convergence of our model within five epochs.

Results

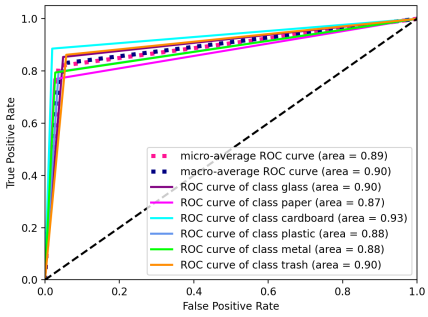
Because this project was focused on multi-label classification, some of the typical metrics used to evaluate binary classification algorithms are not as relevant. We chose to focus on micro- macro- average precision and accuracy in order to get a sense for the overall performance across all classes, as well as using the ROC curve and confusion matrix to understand on which classes our project was underperforming relative to others. It is important to note that, because of the nature of micro-averaged performance metrics, micro-average precision and recall end up being the same quantity, because a false positive for one class is by definition equal to a false negative for another class.

$$Pr_{micro} = \frac{TP_1 + TP_2 + \ldots + TP_k}{(TP_1 + TP_2 + \ldots + TP_k) + (FP_1 + FP_2 + \ldots + FP_k)}$$

Figure 4 - Micro-averaged precision formula for k classes. From Cran-r project [6]

We achieved 83% accuracy on our validation set with our ResNet50 implementation before choosing to focus entirely on our DenseNet121 implementation. For comparison's sake, when we achieved this result, we were able to get around 93% accuracy with DenseNet121 after significantly fewer epochs (10 versus 50). Our detailed results are shown below.

ROC Curve with Data Augmentation



ROC Curve Without Data Augmentation

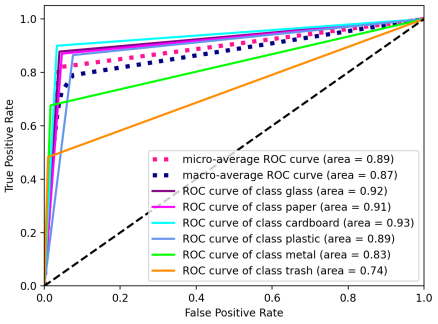


Figure 5 - Roc Curves with and without data augmentation

Performance with No Data Augmentation

	Training	**Validation**	**Test**
Accuracy	0.9938	0.9411	0.9397
Precision	0.9813	0.8232	0.8190

Performance with Data Augmentation

	Training	**Validation**	**Test**
Accuracy	0.9923	0.9400	0.9397
Precision	0.9783	0.8201	0.8190

Confusion Matrix for Test Set with No Data Augmentation							
	Label						
Prediction		1- Glass	2- Paper	3- Cardboard	4- Plastic	5- Metal	6- Trash
	1- Glass	72.00	0.00	0.00	8.00	2.00	0.00
	2- Paper	1.00	94.00	10.00	3.00	0.00	0.00
	3- Cardboard	0.00	4.00	63.00	3.00	0.00	0.00
	4- Plastic	7.00	1.00	0.00	64.00	2.00	0.00
	5- Metal	6.00	3.00	0.00	9.00	46.00	4.00
	6- Trash	0.00	7.00	2.00	4.00	2.00	14.00

Confusion Matrix for Test Set with Data Augmentation							
	Label						
Prediction		1- Glass	2- Paper	3- Cardboard	4- Plastic	5- Metal	6- Trash
	1- Glass	70.00	0.00	0.00	7.00	4.00	1.00
	2- Paper	1.00	83.00	7.00	1.00	1.00	15.00
	3- Cardboard	0.00	6.00	62.00	0.00	0.00	2.00
	4- Plastic	8.00	0.00	0.00	59.00	4.00	3.00
	5- Metal	8.00	0.00	0.00	4.00	54.00	2.00
	6- Trash	0.00	2.00	0.00	1.00	1.00	25.00

Figure 6 - Confusion matrices with and without data augmentation

The algorithm seems to perform worst on trash (mistaking it for metal) and in misclassifying cardboard as paper, as well as misclassifying glass as plastic, as demonstrated in the confusion matrices. We attempted to resolve this problem by augmenting the data, which seems to have made the ROC curve more evenly balanced but ultimately did not fully resolve it. Ultimately, the overall accuracy, as stated in the above tables, was the same with and without the data augmentation, but the error was distributed differently in each case. However, our ROC curve demonstrates that the ROC scores were more even across classes compared to without data augmentation.

We do seem to be overfitting the training data, as we consistently achieved 99% accuracy or higher. We attempted to address this both by using L2 regularization and by increasing the amount of data (via data augmentation), but it seems that further action is required in order to resolve the overfit more fully.

Conclusion

To conclude we initially tested transfer learning with ResNet and DenseNet and found an accuracy of 83% and 93% respectively. We believe ResNet's poor performance could be attributed to the fact that skip connections limit ResNet's representation capacity [12]. Upon focussing experimentation efforts on DenseNet we discovered high confusion between glass and plastic. We address this by balancing and increasing our dataset with data augmentation. We observed less confusion on cardboard, plastic, and metal classes with the same system accuracy. We still have a high variance and would focus future effort to reduce variance by increasing the size of our dataset or implementing dropout. In future work we'd also like to focus on making our add-on model a CNN to increase our ability to describe the discriminator with fewer parameters per layer on more layers.

Contributions

Max -

Most of what Max focused on was the auxiliary functions and analysis; specifically implementing the confusion matrix methods, as well as all of the performance metrics. Max also researched the performance metrics appropriate for multi-label classification. He wrote the introduction, related literature, results, and bibliography sections. Max and Tom split the implementation of the ResNet50 transfer learning.

Tom -

Tom implemented transfer learning with DenseNet121 and worked with Max to get ResNet50 running as well. Tom implemented the add-on deep neural network model that was applied after transfer learning and implemented the data processing and augmentation systems for the project. Tom and Max worked together to run experiments and fine tune parameters. Tom wrote the dataset and features, the methods, and the conclusion section of this report.

Code (Github URL)
https://github.com/tcwelch/cs230_project.git

Work Cited

[1] Cchangcs. Garbage Classification. Wuhan, Hubei, China. Kaggle, 2018. Web. https://www.kaggle.com/asdasdasasdas/garbage-classification

Because the author's first and last name were not listed, we have used their Kaggle username here instead. This is the citation for the dataset we used for our project.

[2] Hrushikesh N. Kulkarni and Nandini Kannamangalam Sundara Raman. "Waste Object Detection and Classification". CS230 at Stanford University. 2019. http://cs230.stanford.edu/projects_fall_2019/reports/26262187.pdf

[3] Wei-Lung Mao, Wei-Chun Chen, Chien-Tsung Wang, Yu-Hao Lin. Recycling waste classification using optimized convolutional neural network. Resources, Conservation and Recycling, Volume 164, 2021, 105132, ISSN 0921-3449, https://doi.org/10.1016/j.resconrec.2020.105132. (https://www.sciencedirect.com/science/article/pii/S0921344920304493)

[4] Olugboja Adedeji, Zenghui Wang, Intelligent Waste Classification System Using Deep Learning Convolutional Neural Network, Procedia Manufacturing, Volume 35, 2019, Pages 607-612, ISSN 2351-9789, https://doi.org/10.1016/j.promfg.2019.05.086. (https://www.sciencedirect.com/science/article/pii/S2351978919307231)

[5] Pedregosa et al. Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830, 2011. https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

Much of the implementation for our ROC plots was informed by the examples outlined in the link listed above.

[6] Davis Vaughan. Multiclass averaging. *The Comprehensive R Archive Network.* 26 March 2021. https://cran.r-project.org/web/packages/yardstick/vignettes/multiclass.html

[7] Aman Arora. "DenseNet Architecture Explained with PyTorch Implementation from TorchVision". 2 August 2020. https://amaarora.github.io/2020/08/02/densenets.html

[8] C.R. Harris, K.J. Millman, S.J. van der Walt et al. Array programming with NumPy. Nature 585, 357–362. 2020. DOI: 0.1038/s41586-020-2649-2. (Publisher link).

[9] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[10] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

[11] P Umesh. "Image Processing in Python". CSI Communications, 23. 2012.

This is the citation for the PIL code library used in our implementation.

[12] Chaoning Zhang, Philipp Benz, Dawit Mureja Argaw, Seokju Lee, Junsik Kim, Francois Rameau, Jean-Charles Bazin, In So Kweon. ResNet or DenseNet? Introducing Dense Shortcuts to ResNet. 3 October 2020

https://arxiv.org/abs/2010.12496#:~:text=For%20ResNet%2C%20the%20identity%20shortcut,memory%20and%20more%20training%20time.

[13] Van Rossum, G. (2020). The Python Library Reference, release 3.8.2. Python Software Foundation.

[14] Chollet, F., & others. (2015). Keras. https://keras.io.

[15] Gao Huang, Zhuang Liu, Laurens van der Maaten, & Kilian Q. Weinberger. (2018). Densely Connected Convolutional Networks. https://arxiv.org/abs/1608.06993