CS230

Image Classification for Folding Laundry

Project Final Report

May 30th, 2022

Faizan Ali
Department of Computer Science
Stanford Center for Professional Development
fzali@stanford.edu

James D. Braza
Department of Computer Science
Stanford Center for Professional Development
jamesbraza@stanford.edu

Abstract

We explored image classification of laundry using two architectures: VGG16 and ResNet50. We detail how to apply regularization and several types of data augmentation to increase inference to 99.5% accuracy. We also experimented with few-shot learning to attain 43.8% accuracy on novel classes in an unseen dataset.

1 Motivation

The past decade has seen the proliferation of the term "smart home" and the rise of domestic robotics. Modern-day homeowners check if the garage is left open from an app, answer the front door from an office, ask robotic vacuum cleaners to handle their spills, and request their home assistants to operate the oven. However, there remain multiple tasks where manual labor is the only option, notably involving dishes and laundry. There have been multiple failed attempts from startups (FoldiMate,[3] Laundroid[8]) to create the first laundry folding robot. The engineering prowess required for laundry folding is daunting, with a pipeline of (1) unruffling, (2) discerning type, (3) folding, and (4) sorted storage. In this paper we tackle (2) discerning clothing type with multiclass image classification.

The principle motivation was to gain a sense of laundry image classification difficulty, as failure(s) in this step would result in an incorrect algorithm being applied in (3) folding. This both includes classifying both previously-known and unknown clothing types. Secondly multiclass image classification is a hallmark use case of convolutional neural networks (CNNs), a core topic of Stanford CS230.

2 Datasets

There are a variety of clothing image datasets available on Kaggle, table 1 details three available datasets with nicknames that we used during training and inference. We primarily worked with the "small"[7] dataset because it has pre-made train (supports nearly 100 batches of size 32), validation, and test sets, as well as semantically distinct classes (others have vague shirt classes). We home-grew a data loader function for the much larger "full" dataset's nonstandard folder structure, which unfortunately slowed training. Conveniently the full dataset's labels are a superset of the small dataset. Lastly, we utilized the "shirts" dataset to facilitate few-shot learning. Unfortunately this dataset contains many corrupt images, so we implemented a cleaning function to remove unusable images.

Table 1: Available Datasets

Dataset	Images	Classes
small[7]	3781	10 total: dress, hat, longsleeve, outwear, pants, shirt, shoes, shorts, skirt, t-shirt
full[5]	5762	20 total: t-shirt, longsleeve, pants, shoes, shirt, dress, outwear, shorts, not sure, hat, skirt, polo, undershirt, blazer, hoodie, body, other, top, blouse, skip
shirts[1]	2779	6 total: jacket, polo shirt, shirt, t-shirt, tank top, warm clothes

3 Architectures and Experiments

We provide a high-level overview of our two architectures and experiments in table 2:

- VGGNet[6] (Visual Geometry Group Network): a canonical CNN architecture published in 2015. We always used a VGG16 (from Keras) trained on the ImageNet dataset.
- ResNet[4] (Residual Network): another CNN architecture also published in 2015 that uses residual blocks to create a very deep network. We both used a ResNet50 (from Keras) trained on the ImageNet dataset, and we also implemented our own "DIY" ResNet using Tensorflow 2.4. Both Keras's supplied and our home-grown ResNet models match the ResNet paper's configuration.

By our project milestone we had completed a transfer-learned VGG16 baseline, which revealed we had a substantial overfitting issue. Using this knowledge, we conducted many experiments described in the following sections, and analyze the results in section 4.

Table 2: Experiments conducted and results obtained. All results were obtained on the small dataset besides the few-shot learning experiment on the shirts dataset.

Experiment	**Architecture**	**Transfer Learning**	**Dataset**	**Train/Val/Test Accuracies**	**Precision, Recall, F1 % (Test)**
Baseline	VGG16	ImageNet	small	99.6, 86.2, 84.7	85.6, 84.7, 84.2
	ResNet50			96.0, 88.3, 89.3	89.4, 89.2, 89.2
Image Randomization	VGG16	ImageNet	small	94.6, 88.3, 88.2	88.5, 88.2, 88.2
Regularization	VGG16 (Dropout)	ImageNet	small	95.9, 87.4, 82.0	84.0, 82.0, 82.3
From Scratch	ResNet50 (DIY)	n/a	small	90.7, 70.7, 62.1	66.1, 62.1, 60.5
Merged Datasets	VGG16	ImageNet	small + full	99.8, 98.2, 99.5	99.5, 99.5, 99.5
	ResNet50 (DIY)	n/a		98.1, 92.1, 97.6	97.9, 97.6, 97.6
	ResNet50	ImageNet		99.8, 95.6, 98.4	98.5, 98.4, 98.4
Few-Shot	VGG16	Merged Datasets	shirts	86.7, 43.8, n/a	36.6, 43.7, 35.9 (Validation)

3.1 Baseline Models

As a baseline, we performed transfer learning on both a VGG16 and ResNet50 pre-trained on the ImageNet dataset. For both models, we baked in a pre-processing Lambda layer to convert the images from RGB to BGR with each color channel zero-centered (required for ImageNet dataset). Additionally, we end with a softmax classification fully-connected layer.

In the VGG16 model's top, the 1st and 2nd ReLU-activated fully-connected layer's unit count matches the VGG16 paper with 4096 units. Similarly in the ResNet50's top, we use spatial global average pooling before the classification layer as the ResNet paper did.

3.2 Data Augmentation: Image Randomization

To address our baseline overfitting, we used Keras' ImageDataGenerator class to generate batches of augmented data in real-time. For augmentation types and values (all symmetric ranges), we used $\leq 20\%$ width/height translation, optional horizontal flip, $\leq 20°$ rotation, and $\leq 20\%$ zoom to generate random new images to assist in training a more generalized model. We felt shearing was not relevant as it would distort clothing images in an unrealistic manner.

3.3 Hyperparameter Search for Regularization

As a second approach to tackling our baseline VGG16's overfitting, we performed a random search on two regularizing hyperparameters. We added L2 regularization to the last layer with a variable λ and a Dropout layer between our 1st and 2nd fully-connected layers with a variable drop rate. For λ, we logarithmically randomly sampled 16 values on the log scale between $[1\text{e-}5, 1]$. Similarly for drop rate we uniformly randomly sampled 16 values in $[0.1, 0.8]$. We choose these ranges based on our understanding of commonly used values chosen to create meaningful search results.

3.4 ResNet50 From Scratch

To gain insight into how hard it is to construct and train a Tensorflow model from solely a paper, we chose to implement a ResNet50. The 50 comes from the model being 50-layers deep. This meant combing through the ResNet paper and cross checking with many online articles to figure out the following set up in appendix section 7 and the residual block equations:

- Identity block (empty shortcut): $\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$
- Convolutional block (convolution in shortcut): $\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$

Where $\mathcal{F}$ refers to the residual block (either identity or convolutional) and W's refer to weights used within. Please see the ResNet paper or our GitHub repo[2] for further details.

3.5 Data Augmentation: Merging Datasets

Here, we exploit the full dataset being a superset of the small dataset. We first preprocessed the full dataset to filter for common labels with the small dataset, and updated all full dataset label indices to match the small dataset's label indices. Then, we merged the small dataset's training portion with this preprocessed full dataset, leaving the small dataset's validation and test sets untouched. With a batch size of 32 images, this merge increases the training batch count from 96 to 238 (about 2.5X multiple).

On this dataset we trained a pre-trained VGG16 + ResNet50 and our untrained DIY ResNet50.

3.6 Few-Shot Learning

We decided to measure our trained model's performance on novel classes. We used the shirts dataset (six classes), splitting out a training set such that there are four batches of four images. The logic here is sixteen images approximately matches a realistic count of six-types of shirts.

Using the VGG16 trained on the merged dataset (pre-trained twice: once on ImageNet, then once on our data), we froze that model and removed the top three layers for transfer learning. Then adding on three fully-connected layers of 4096 (ReLU), 4096 (ReLU), and 6 (softmax) units, we trained our few-shot learning model. One note: the shirts dataset only has training and validation subsets.

4 Results

4.1 Training Procedure

Training was completed with an Adam optimizer (learning rate of 0.001) with a categorical crossentropy loss function on batches of 32-images. Our default max number of epochs was 64, and we integrated Keras' EarlyStopping training callback to stop training if validation set accuracy didn't increase for 8 epochs. Depending on the experiment, training would usually early stop between 10 - 25 epochs, and run up to 40 epochs for our DIY ResNet50 on the small + full "merged" dataset.

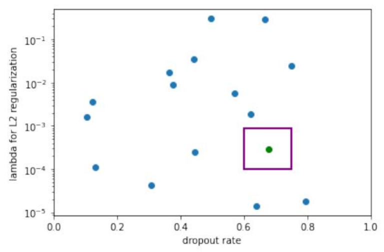
We supplied the same random seed of 42 to our dataset factory functions to reliably produce the same datasets, enabling comparability across training runs. We did not specify an initialization for model layers, instead using Tensorflow's default initialization method. Thus, we did not seed any randomness involved in weight initialization, and each training run would produce a model with different performance.

4.2 Analysis of Experimental Results

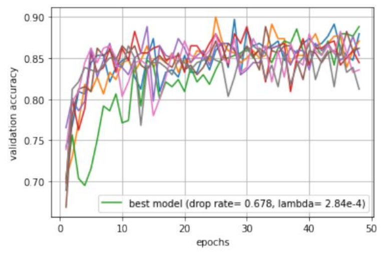
Firstly we would like to assert that Bayes error is 0% (in other words, 100% accuracy). If clothing classes are clearly disjoint, than we believe an average adult can attain a 0% error rate. However, the datasets available use categories that leave room for interpretation (shirt vs t-shirt, outwear vs longsleeve), so in practice the average adult may not always be correct.

Baseline Overfitting. Please see the train/validation/test set accuracies in table 2. It's evident that both baseline models have a substantial overfitting issue (0.5 - 4% bias, 8 - 15% variance). As stated in section 3, this shaped our experimental campaign for the remainder of the project.

Image Randomization. This regularizing technique increased validation accuracy from 86.2% to 88.3%, while decreasing training accuracy from 99.6% to 94.6%. Hence, we reduced the model's variance, or the gap between the training and the validation accuracy, from 13.4% to 6.3%. Thus, this experiment decreased the model variance by more than half, proving this is an effective variance reduction technique. The drop in training set performance is expected due to the regularizing effect of random image data generation.



(a) Hyperparameter search



(b) Validation set accuracy across epochs

Figure 1: Scatter plots of hyperparameter search and corresponding validation accuracy curves.

Hyperparameter Search for Regularization. Using random hyperparameter pairings found in figure 1a, we generate the validation curves shown in figure 1b. The best performing model (green curve) yielded a validation accuracy of 88.9% with roughly drop rate$= 0.678$ and $\lambda = 2.84$e-4, which corresponds to the green coordinate. This improved over our baseline model's validation accuracy of 86.2%, reducing variance from 13.4% to 8.5%. Like Image Randomization, this experiment was also effective at generalizing our model and reducing overfitting to the training data.

To further refine our hyperparameter pairing, we ran another random sampling local to the current best pairing, as show in the purple box of figure 1a. Although on average this 2$^{\text{nd}}$ pass gave better model accuracies than the 1$^{\text{st}}$ sampling, we did not outperform the best pairing from the original sampling. The deeper scatter plot and tabular results from figure 1a can be found in appendix section 10.

Value of Transfer Learning. Both the DIY ResNet50 and baseline transfer-learned ResNet50 were trained on the small dataset. The 25% accuracy increase and 30% F1 score increase on the test set from ImageNet pretraining underscores the value of transfer learning.

Benefits of Larger Training Set. Comparing the baseline models' performance when trained on the small dataset vs the merged dataset (2.5X bigger), performance increased drastically across the board. Test set accuracy for the baseline VGG16 increased from 84.7% to 99.5%, and for the baseline ResNet50 it increased from 89.3% to 98.4%.

VGG16 over ResNet50. Let's look at the baseline VGG16 trained on the merged dataset and compare it with both the baseline ResNet50 and the DIY ResNet50 trained on the same dataset. We can see the baseline VGG16's validation and test set accuracies are at least 1% higher, which means regardless of ResNet pre-training on ImageNet, the ResNet model is not as performant.

Few-Shot Learning. Our training set here was 16 images, and we attain a decent 86.7% accuracy on the training set. However, given the validation set accuracy of 43.8% and F1 score of 35.9%, it's evident our few-shot learning model did not generalize. Looking at the confusion matrix in appendix section 8 one can observe the shirts dataset is quite challenging, as the labels are very close in meaning. Furthermore, we can see our model is confusing t-shirt, polo-shirt, and tank-top quite a bit. We can conclude we need a fundamental change for few-shot learning to function well.

4.3 Scrutinizing Labels

In figure 2 we have the small dataset's test set confusion matrix for our DIY ResNet50 trained on the merged dataset. Looking at the longsleeve class, there are 67 true positives, 5 false negatives (4 from outwear), and 0 false positives. Now looking at the shirt class, there are 22 true positives, 4 false negatives (3 from outwear), and 0 false positives. This reveals to us that our ResNet is struggling exactly where a human would struggle: on the overlapping distinction between longsleeves, outerwear, and shirts (button downs).

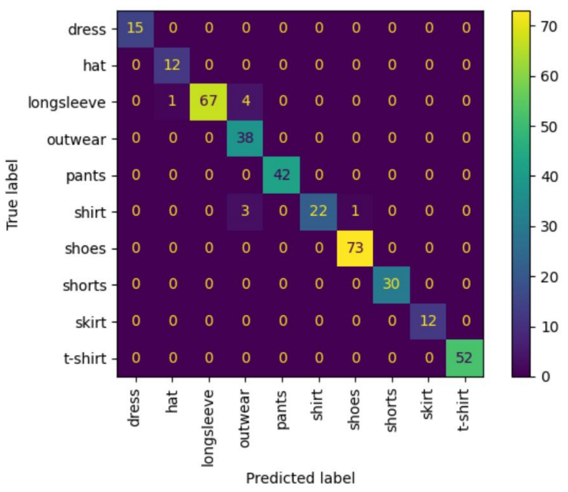


Figure 2: DIY ResNet50 confusion matrix

5 Insights and Future Steps

At the end of the project it was fun to create a "home dataset" (personal clothing items) to actually field-test our model. Using our DIY ResNet50 model, we created a visualization of the softmax layer in appendix section 9. Our model performed well on these novel real-world images, and the bar graphs yield insights into the model's thought process.

One major insight is that "venn diagram" (non-disjoint) classes present challenges for deep learning classifiers. This challenge is best tackled simply with tons of data. Another insight is connecting up new datasets can be quite arduous, especially if the datasets are not organized in a standard fashion. When it comes to accuracy, one should not begin training from scratch, and the more data the merrier.

6 Code and Contributions

GitHub repository: jamesbraza/cs230-project.

Faizan. I carried out the image randomization experiment and random search on hyperparameters experiment. For the latter, I designed and implemented the script for random hyperparameter sampling, modeling, and plotting the accuracy results. Admittedly, I am a novice coder and had trouble adding functioning code at many stages of the project. However, James is an excellent coder/programmer and helped me with troubleshooting and understanding the coding environment throughout the course.

James. I "underperformed" on the midterm so I worked quite hard on this project. My contributions were creating the model code for VGG16 transfer learning, ResNet50 transfer learning, and all code to build a ResNet50 from scratch. I wrote the train and predict scripts used to produce our primary results table and integrated helper functions to enable accuracy, F1 score, and confusion matrix analysis. Experiments I ran were all the baselines, ResNet from scratch, data augmentation via merged datasets, and few-shot learning. I constructed all dataset utilities (downloading, cleaning, loading, and merging) and AWS infrastructure. Lastly, I compiled the "home dataset" with plots displaying the softmax classifier used in the video. It was a pleasure working with Faizan and am proud of our final product here.

References

[1] Gabriel Albertin. Clothing Dataset. https://www.kaggle.com/datasets/gabrielalbertin/clothing-dataset. [Version 2].

[2] James Braza and Faizan Ali. Image Classification for Folding Laundry. https://github.com/jamesbraza/cs230-project.

[3] FoldiMate. https://en.wikipedia.org/wiki/FoldiMate. [Online; accessed 10-April-2022].

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015.

[5] Ololo. Clothing dataset (full, high resolution). https://www.kaggle.com/datasets/agrigorev/clothing-dataset-full. [Version 1].

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[7] Abdelrahman Soltan. clothing dataset small. https://www.kaggle.com/datasets/abdelrahmansoltan98/clothing-dataset-small. [Version 2].

[8] The startup behind the world's first laundry robot has folded. https://www.digitaltrends.com/home/seven-dreamers-laundroid-bankrupt/. [Online; accessed 10-April-2022].

7 Appendix: ResNet50 Structure

Table 3: ResNet50 Structure

Category	Layer	Hyperparameters
Preprocessing	Input Lambda	n/a n/a
Conv1	ZeroPadding2D Conv2D BatchNormalization ReLU	$p = (3, 3)$ $n_C = 64, f = (7, 7), p =$ valid, $s = (2, 2)$ n/a n/a
Conv2	ZeroPadding2D MaxPool2D ConvBlock IdentityBlock (2X)	$p = (1, 1)$ pool_size $= (3, 3), p =$ valid, $s = (2, 2)$ $n_C = (64, 64, 256), f = ((1, 1), (3, 3), (1, 1)),$ $p =$ (valid, same, valid), $s = ((1, 1), (1, 1), (1, 1))$ $n_C = (64, 64, 256), f = ((1, 1), (3, 3), (1, 1)),$ $p =$ (valid, same, valid), $s = ((1, 1), (1, 1), (1, 1))$
Conv3	ConvBlock IdentityBlock (3X)	$n_C = (128, 128, 512), f = ((1, 1), (3, 3), (1, 1)),$ $p =$ (valid, same, valid), $s = ((2, 2), (2, 2), (2, 2))$ $n_C = (128, 128, 512), f = ((1, 1), (3, 3), (1, 1)),$ $p =$ (valid, same, valid), $s = ((1, 1), (1, 1), (1, 1))$
Conv4	Conv3, but with IdentityBlock (5X)	Conv3, but with $n_C = (256, 256, 1024)$
Conv5	Conv3, but with IdentityBlock (2X)	Conv3, but with $n_C = (512, 512, 2048)$
Classification	GlobalAveragePooling2D Dense	 $n_l =$ num_classes, softmax

8 Appendix: Few-Shot Learning Confusion matrix

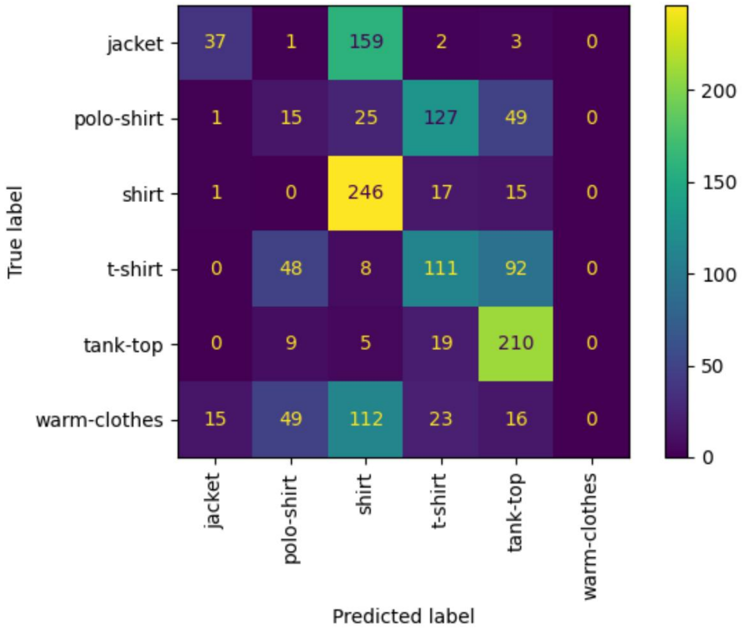
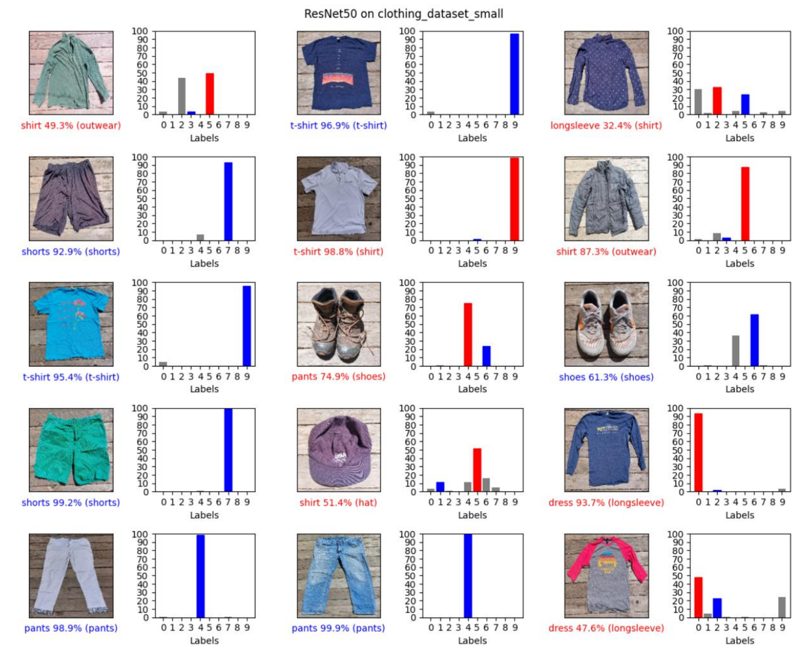


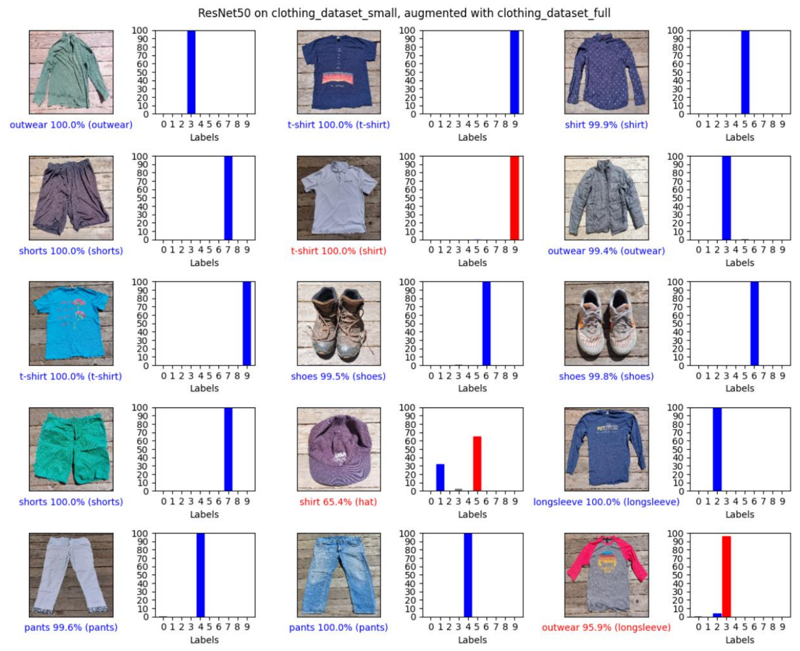
Figure 3: Confusion matrix for our few-shot learning VGG16 model.

9 Appendix: Home Dataset Results

Below is a visualization of the final softmax layer. For the left images, blue text means correct prediction, red text means incorrect prediction. The naming convention is "{predicted class} {prediction probability}% ({true class})." For the right images, blue bars are the true class, grey bars are all other classes, and red bars (if present) are the incorrect prediction.



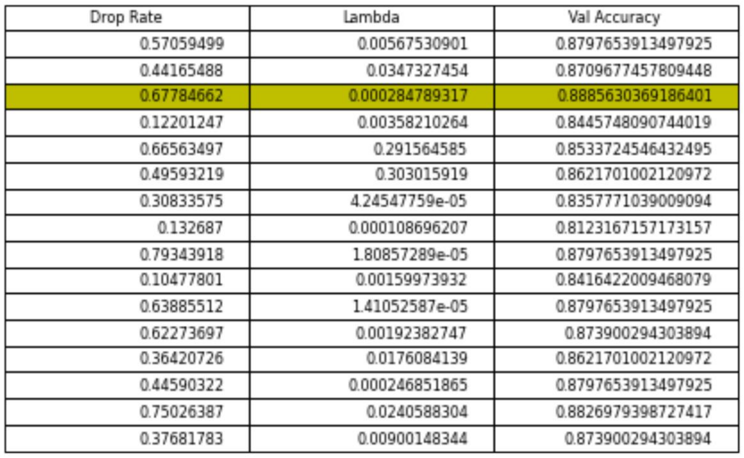
(a) DIY ResNet50 trained on just the small dataset.



(b) DIY ResNet50 trained on the merged small + full datasets.

Figure 4: DIY ResNet50 softmax visualization.

10 Appendix: Hyperparameter Search



Drop Rate	Lambda	Val Accuracy
0.57059499	0.00567530901	0.8797653913497925
0.44165488	0.0347327454	0.8709677457809448
0.67784662	0.000284789317	0.8885630369186401
0.12201247	0.00358210264	0.8445748090744019
0.66563497	0.291564585	0.8533724546432495
0.49593219	0.303015919	0.8621701002120972
0.30833575	4.24547759e-05	0.8357771039009094
0.132687	0.000108696207	0.8123167157173157
0.79343918	1.80857289e-05	0.8797653913497925
0.10477801	0.00159973932	0.8416422009468079
0.63885512	1.41052587e-05	0.8797653913497925
0.62273697	0.00192382747	0.873900294303894
0.36420726	0.0176084139	0.8621701002120972
0.44590322	0.000246851865	0.8797653913497925
0.75026387	0.0240588304	0.8826979398727417
0.37681783	0.00900148344	0.873900294303894

Figure 5: Validation accuracies from the hyperparameter search. Best pairing is highlighted.

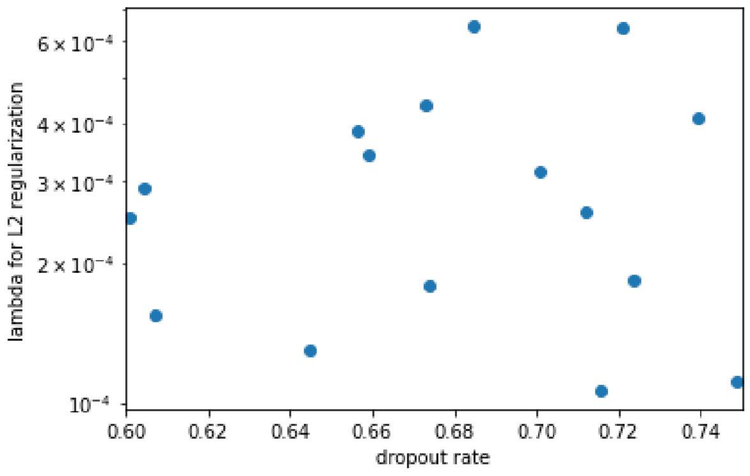


Figure 6: Deeper Random Hyperparameter Search