# CAI 4104: Machine Learning Engineering

Project Report: The Art of Transfer Learning

Benjamin Simonson
*(Point of Contact)*
bsimonson@ufl.edu

Haadi Gill
h.gill@ufl.edu

Alexander Calvo
acalvo1@ufl.edu

Nathaniel Austin-Clarke
n.austinclarke@ufl.edu

Lysandra Belnavis-Walters
lbelnaviswalters@ufl.edu

April 25, 2025

## 1 Introduction

Machine learning as a field continues to grow and expand every day, encompassing new breadths of applications and utilizations. As a result, the need to understand and implement models for individual scenarios has risen alongside it [5]. In this project, we explore the construction, testing, evaluation, and adjustment of an image classification model to separate between 12 different categories of images: backpack, book, calculator, chair, clock, desk, keychain, laptop, paper, pen, phone, and water bottle. We accomplish this task using the TensorFlow [6] and PyTorch [2] libraries available within the Python collection. All data has been amassed through a collection of user-driven images, resulting in the possibility and potential of outlier labels, complex or indistinguishable data, and other challenges. By taking advantage of transfer learning and marginal fine-tuning, we were able to create a model with an accuracy above 90%.

## 2 Approach

In order to begin our project, the first step involves identifying the end goal and constructing what is not the most optimal solution to ever possibly exist, but a pathway of analysis and operations to lead to a level of reliability and accuracy that our team feels is acceptable. Since we are handling image data, the first thought was to use a convolution neural network (CNN) as they are much more adept at handling such a format of information. Then, the next step is to define the rest of the architecture and approach for processing the data. The main considerations overlapped with general machine learning complexities: make sure the model does well, but does not overfit. Our images needed to be learned, not memorized.

As this field is a collection of continuously growing and expanding ideas and applications, not all work needs to be orchestrated alone. Progress takes time, and there are tools to use to ease that process. We utilized an existing model, ResNet18 [7], to use as a basis for our own. After exploring different options [3], we decided on a network that takes in "mini-batches of 3-channel RGB images of shape (3 x H x W)" [1] and is modified to output twelve logits wherein each logit represents the probability of an image being a certain class. In particular, we have added two layers [4] to the trained DenseNet model:

1. a fully-connected layer with 128 units, ReLU activation, and 50% dropout
2. a fully-connected output layer with 12 units

Furthermore, other than shuffling the images after initially loading it, we do not preprocess or augment the data. While these methods are often beneficial in helping the model converge or generalize, we did not see any added performance in the model.

Now that our model had its architecture and the data had been prepared for training, the next step was to isolate individual training and validation data sets to analyze the neural network's performance and further reduce overfitting. The validation and training datasets had 375 images reserved and the training dataset size included all the remaining image data. A random split created the two non-overlapping training and validation datasets. PyTorch DataLoaders funneled the data in batches into the model for training and validation. The model was then trained for 200 epochs.

1. Cross-entropy loss function

2. Adam optimizer with a learning rate of 0.00001 and weight decay of 0.00001

The weight decay adds a regularization to further reduce overfitting

# 3 Evaluation Methodology

To evaluate the model, we observed the loss and accuracy of each set (training, validation, test). These sets were split 80-10-10 in which 80% of the data is reserved for training while the remaining 20% is split across the validation and test sets. Since this is a classification task, the metric we are using to gauge performance is accuracy, linking the outputted labels to the expected test lables. Folowing this metric, per random guessing 1 class out of the 12, the baseline accuracy is 8.33%.

# 4 Results

After 200 epochs with an 80-10-10 training, validation, testing dataset split:

| Dataset | Loss | Accuracy (%) | Precision (%) | Recall (%) |
|---|---|---|---|---|
| Training | 0.023 | 99.1 | 99.1 | 99.0 |
| Validation | 0.375 | 93.1 | 94.0 | 93.9 |
| Test | 0.221 | 93.7 | 92.6 | 92.6 |

There is a noticeable difference in the performance of the model with the training data ( 99%) and with the validation ( 93%) and test data ( 92%). This likely indicates that our model is slightly overfitting the training data. However, a roughly 92% test accuracy indicates that the model generalizes well: reliably and correctly classifies unseen images to their correct labels. The similar precision and recall values within each dataset are somewhat expected as each class in the dataset has about the same number of examples. Additionally, this likely reinforces that the model is performing well. Generally, the model has a 92% accuracy. The baseline accuracy is 8% as the dataset is effectively balanced.

# 5 Conclusions

We developed a convolutional neural network capable of correctly identifying the object depicted in images. We used a pretrained ResNet18 model and modified the fully connected classifier head to better suit our problem. The model reached an accuracy of 92.3% on the testing data set, alongside 92.6% precision and recall. Equally high precision and recall indicate that the model neither frequently misidentifies classes nor fails to identify most instances of a given class—the model displays good predictive output and coverage. We observed model overfitting based on the difference between the training accuracy and validation and test accuracies; but data augmentation, dropout, and optimizer regularization did help to reduce it. Ultimately, the model performed well. But the image dataset is relatively small compared to the dataset ResNet18 was trained with, leading to at least some inevitable overfitting issues.

# References


[1] PyTorch. Densenet.

[2] PyTorch. Library overview.

[3] PyTorch. Models.

[4] T. Ridnik, G. Sharir, A. Ben-Cohen, E. Ben-Baruch, and A. Noy. Ml-decoder: Scalable and versatile classification head. *Computer Vision and Pattern Recognition*, 2023.

[5] I. H. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2021.

[6] TensorFlow. Tensorflow.

[7] Torchvision. Resnet.