ECE 590-05 Introduction to Machine Learning - Final Project Transfer Learning

Nathan Inkawhich
Yifan Li
Hayden Bader

Abstract

In this work, we assess the efficacy of transfer learning when training deep convolutional neural networks on new datasets. We choose to evaluate three state of the art models, AlexNet, Squeezenet, and Resnet18, on three datasets of varying difficulty. As a baseline, we train each model with random weight initializations. For the transfer learning techniques we use fine-tuning and feature extraction, and in both cases the model weights are initialized from ImageNet pre-trained models. We run each learning method on each of the models, for each dataset, and collect validation accuracy versus number of training iterations. We show that across all configurations, both transfer learning methods yield better results in terms of convergence speed and validation accuracy than the scratch method, and among the two, fine-tuning is superior. These results motivate the use of transfer learning in situations where training time is limited, resources are scarce, and accuracy is crucial.

Introduction

In today's society, Convolutional Neural Networks (CNNs) are recognized as extremely powerful tools for pattern recognition and decision making that can, if used correctly, greatly increase efficiency, productivity, and profitability for those who employ them. However, it is really the flexibility of CNNs that most accounts for this recognition. Whether they are used in self-driving cars, image classification challenges, generative adversarial networks (GANs), or other fields, each system takes in different inputs, from different domains, and has different goals, yet are built around the same base model. Still, a consistent truth across Neural Networks is that more complex systems require more resources either in the form of processing power, which costs money, or time, which is valuable in its own way. As a result, organizations seeking to formulate solutions to machine learning problems often have to limit the scope of their research to the size of the resources they have at their disposal.

Given that context, one technique that has shown promise in reducing the barrier to entry for those with more limited resources is Transfer Learning (TL). The basic idea for TL is simple: use knowledge gained from learning a previous task towards learning a new task. Figure 1, below, presents a visual representation of this idea.

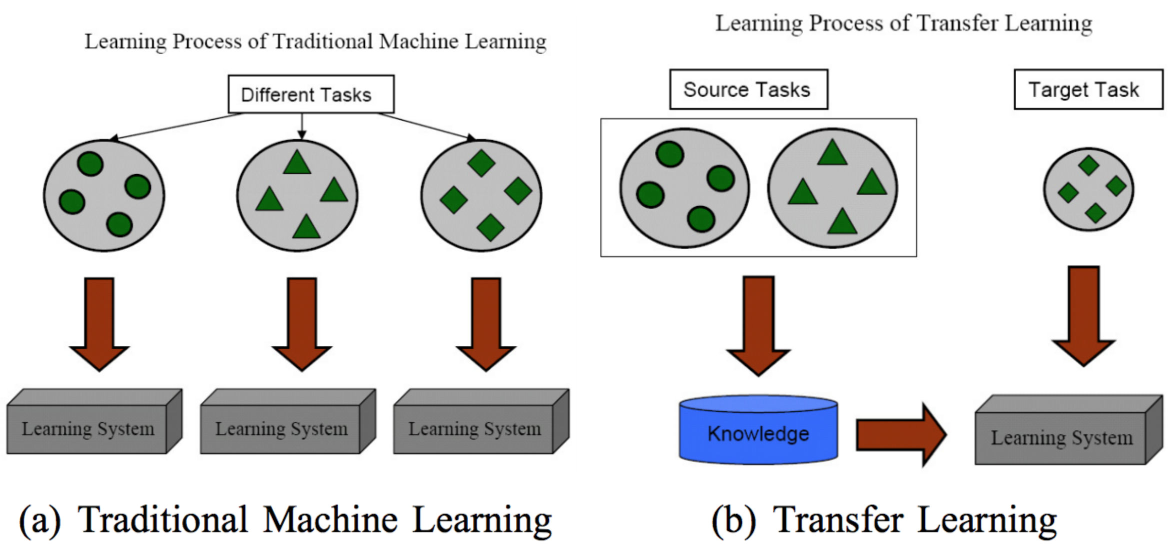


Figure 1. Visualization of Transfer Learning technique versus traditional supervised learning. Instead of training a model from scratch for each task, we can use knowledge from other (source) tasks towards learning a new (target) task. This image is from [4].

In other words, TL can be viewed as a type of mentorship. A new model lacking experience in how inputs relate to outputs is "instructed" about how certain parameters are interrelated. The Background Section, below, describes this relation in more detail. However, the important part is that a new learning system doesn't have to spend as much time gathering the basic knowledge that the "mentor" already spent time accumulating. Instead, the new system can spend its time fine-tuning the parameters that have the greatest effect on its designated task. Notably, this does not require the "mentor" system and the "mentee" system to be dedicated to the exact same task. They just have to be in the same "field of work" so that the "mentee" can take advantage of the "mentor's" experience. This means that in order to make use of TL, all one needs to do is formulate his problem in a way that mimics an architecture whose data is available for use. In addition, TL also opens up avenues for those who seek to study small datasets. Normally, to make accurate generalizations, CNNs need to operate on a large dataset. Otherwise, fitting a small set of data to a model with a huge amount of flexibility has an extreme risk of overfit. But since TL allows CNN models to not have to be trained from scratch, these risks can be largely mitigated.

However, there are several methodologies through which TL can be implemented. Moreover, there are a number of models that have been developed in order to assist in transferring knowledge from pre-existing models to new models. As a result, this paper aims to systematically compare various TL methods across several available models and datasets in order to more objectively quantify the benefits of these various approaches. Ideally, this will result in general guidelines that can better assist those looking to harness the power of Transfer Learning to generate models that converge more quickly during training and are as accurate as possible during testing.

Background

Since the primary goal of this paper is the exploration of transfer learning, it is important to first explain the basics of how Neural Networks work and how knowledge can be represented in one. Neural Networks, in general, are mathematical models that attempt to replicate the functionality of the human brain. Nodes are the fundamental building blocks for these networks. They accept input from previous layer nodes, perform a linear combination with a set of internal weights, then output a value through an activation function. These node outputs are then input into the next layer neurons. This process repeats throughout the various layers that make up the network until an output layer is reached. The values at the output layer can then be compared to a ground truth in order to determine the error produced by the network. Then, since the entire networks is a combination of inputs, weights, and activation functions, the change of the output with respect to the weights can be determined, and the weights can be updated to reduce the error. The whole process of running values through the entire network and then having weights updated constitutes one cycle of training. Often, this process must be repeated a substantial number of times before the error caused by the weights converges. Figure 2 displays this process.

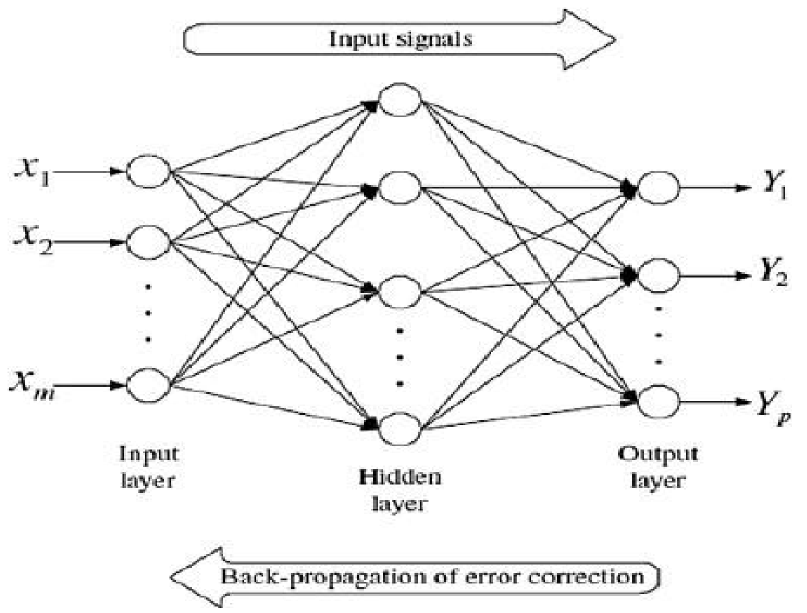


Figure 2. Visualization of Neural Network architecture. Inputs first propagate in the forward direction through the layers, then, after computing a loss, the error is backpropagated through the system to adjust weights to minimize the loss. This image is from [19].

Notably, CNNs are a specific type of neural network that have seen extensive use in image processing and are typically constructed to take advantage of pictoral inputs by having certain layers perform convolution on the features. Still, regardless of the type of network that is used, over enough iterations, the weights used to perform these calculations converge. Transfer learning, then, seeks to reuse the knowledge gathered in these weights into a new model with the same architecture so that the new model does not have to perform as many iterations to reach convergence. This works because some of the features inherent in similar tasks persist across similar models.

Still, there are several ways of utilizing pre-existing weights in training a new model. One of the more common methods involves initializing the weight from a pretrained model, then adapting *all* of the weights during the training process. This method is generally applied when the target dataset is large. Another is to use the CNN as a fixed feature extractor and only try to learn the *final* layer of the model. The idea here is that the CNN has already learned the high and low level feature extractors and the task is to just map those features to a new set of classes. This method is generally used when the target domain is similar to the source domain and when the number of training examples is limited. There are numerous examples of papers that even use both metrics in an attempt to better evaluate a specific dataset even in vastly different domains from standard image processing [1]-[3]. However, in general, contemporary work does not report on any notion of how either metric should perform against the other. Even [4] which documents and categorizes Transfer Learning techniques over the years focuses more on the procedures of the experiments and less on the general trends between methods.

Furthermore, distinct from methods of training, there are also a large variety of models that can be used in transfer learning to conduct the training of the new model. Common model architectures that appear often in literature include Inception V3, GoogLeNet, Resnet, Alexnet, and many others. And further extensions are continuously being researched [5,6]. Still, when discussing common models and methods, it is also worth discussing common datasets. ImageNet [7] is a dataset consisting of over 1.2 million RGB images from 1000 classes of everyday things. Some of the classes included are dog, kite, frilled lizard, broom, and coffee mug, among many others. In the area of computer vision and image based applications, this dataset is one of the fundamental benchmarks for how effective a model architecture is, so it is a credible place to learn from. In fact, there is an annual challenge based on the dataset called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which is where most of the new cutting edge techniques developed over the year are benchmarked.

One specific application where TL has shown to be useful is in training action recognition classifiers, which work similarly to image classifiers however they also consider temporal information. In [20], the authors use a pretrained Two-Stream Inflated 3D ConvNet (I3D) model to learn motion in a new action/video dataset. They found that pretraining led to a 5% increase in classification accuracy. The authors also analyzed that pretraining on different datasets led to different results after the knowledge transfer. Another specific TL application is image classification with sparse prototype representations. Specifically, a news-topic prediction task where the goal is to predict whether an image belongs to a particular news topic. In [21], the authors developed a method for transfer learning which exploits available unlabeled data and an arbitrary kernel function, and then formed a representation based on kernel distances to a large set of unlabeled data points. The results showed that when only few examples are available for training a target topic, leveraging knowledge learned from other topics can significantly improve performance.

In summation, it is important to use commonly accepted metrics given this paper's goal. After all, when attempting to make generalizations, the methods and procedures should be repeatable, and the genesis of the decision process should be understandable. However, further explanation about the specifics of models and why they have been selected will be relegated to the *Methods* section.

Data

The original Fruits-360 dataset [14] is a high-quality dataset of images containing 60 classes of fruits and 38,409 total images. Though it does not present an exhaustive collection of all types of fruit, it covers the most common types including several varieties of apples, pears, grapes, and bananas among others. Below are some sample data from the training set. Each individual image is 100x100 pixels.

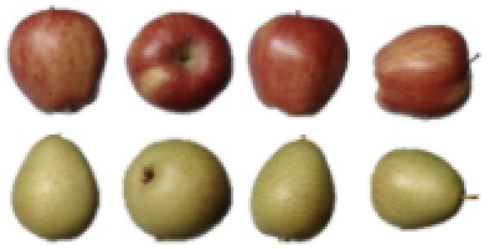


Figure 3. Sample images from Kaggle's *Fruits 360 Dataset*.

In this work we use the Fruits-360-10-class dataset, which contains all images from the first 10 classes in the original dataset. Similarly, the fruits-360-30-class dataset contains the images of the first 30 classes in the original dataset and included a greater variety of fruits. Despite the benefits TL provides, it would have taken an inordinate amount of time to train and test the various TL models on the entire Fruits-360 dataset even on GPUs, so we split the fruits-360 dataset into 10-class and 30-class portions to reduce computational cost. Incidentally, the classes from the 10-class fruit dataset are a subset of the 30-class fruits dataset, but due to the three-fold increase in labels we speculate that the fruits-360-30-class dataset would be 3 times harder to classify than its 10-class counterpart. As a result, we would also expect it to take 3 times the training iterations to converge.

The flowers dataset refers to the Flowers Recognition dataset posted on Kaggle [15]. The dataset contains 4,242 images of flowers and the pictures are divided into 5 classes: chamomile, tulip, rose, sunflower, and dandelion. The images are not high-resolution (about 320x240 pixels) and are not a uniform size. Below are some sample data from the training set. Also, Table 1 shows the number of samples in the training and test splits of the three aforementioned datasets.

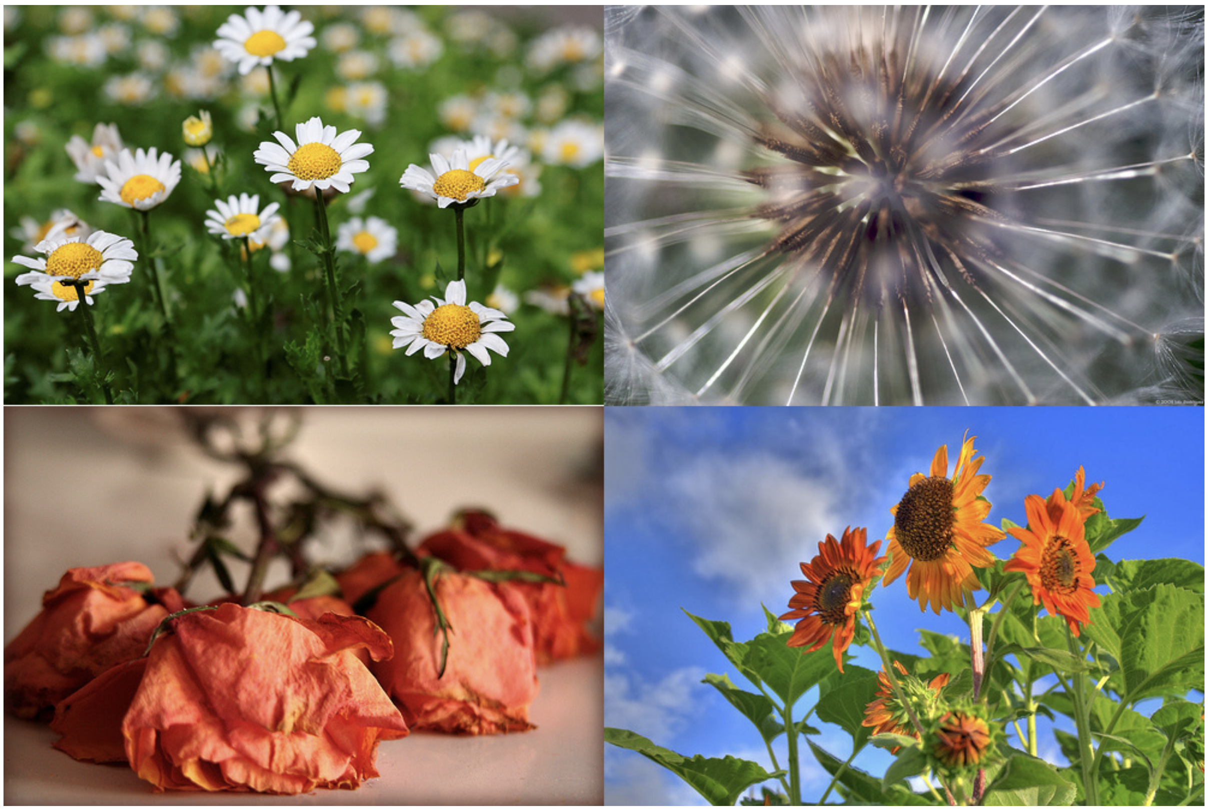


Figure 4. Sample images from Kaggle's *Flowers Recognition Dataset*

Table 1. Number of images in the train and validation splits of each dataset

Dataset Name	Number of Training Images	Number of Validation Images
Fruits-360-10-class	4,854	1,629
Fruits-360-30-class	14,620	4,928
Flowers	3,398	938

From simply visualizing the data, it would be very difficult for a human to classify the images. For instance, Figure 5 shows two red apples in the fruits dataset that look indistinguishable to the human eye, but they actually belong to two different classes. This distinction alerts us to a potential challenge inherent in our chosen datasets - relying too much on color for correct classification might not be very accurate.

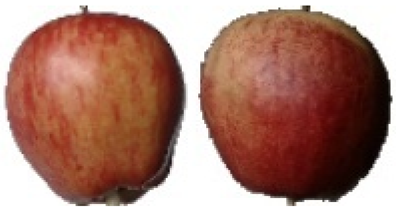


Figure 5. Two apples that belong to different classes

Moreover, as we can see from the sample images from the flowers dataset in Figure 4, unlike the fruits dataset, the images in the flowers dataset have different backgrounds that may introduce some noise into the learning models. So we could speculate that the accuracy of our models when testing on the flowers dataset should be lower than than the accuracy reported when testing on the fruits dataset.

Overall, from simply visualizing the data, we see that classifying the data in these datasets is not a trivial problem. Thus, our models must be robust enough to overcome the potential challenges inherent in the datasets while not overfitting. Some possible methods to help robustness include good data augmentation, preprocessing, and proper reshaping.

Methods

Since the purpose of TL is to use knowledge from a source task and apply it to a target task, the first questions we face are what source task should we learn from and what models should we use? Because our problem involves images and our models are CNNs, we have decided to leverage the knowledge available from the aforementioned ImageNet. The datasets we have chosen for this project are from the same general domain as the ImageNet images. In other words, all of the images we use are "natural" or "earthly" objects presented in a color format. Therefore, intuition suggests that the knowledge from ImageNet is very relevant to the target domain, and we may expect significant boosts in learning speed and overall accuracy.

For the model architectures, we will use three models designed for the ImageNet dataset, all off which are seminal works in the field. The first is Alexnet [8], and is perhaps the reason CNNs are so wildly popular today. Before Alexnet was shown to work on ImageNet, the large scale computer vision field was dominated by other machine learning algorithms [9]. However, once Alexnet was submitted to ILSVRC-2012 and achieved 62.5% top-1 test accuracy, CNN research exploded and has not slowed down since. This was also in part due to the development of GPUs. The Alexnet architecture, shown in Figure 6, is relatively simple and consists of five convolutional layers, three max pooling layers, and two fully connected layers.

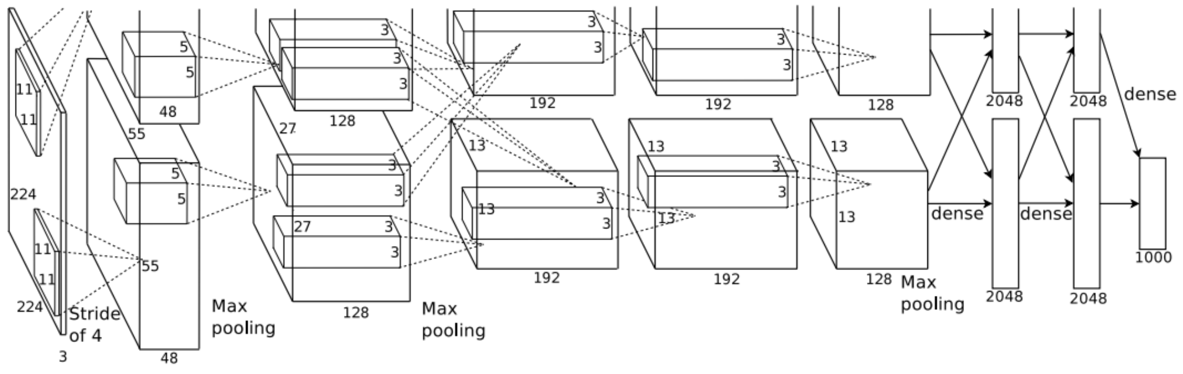


Figure 6. Alexnet CNN architecture as shown in original paper [8].

The next architecture we will use is Squeezenet [10], shown in Figure 7, which was originally proposed as a model with competitive accuracy to Alexnet, but with 50 times fewer parameters. Squeezenet is important because it was one of the first models to be concerned with model size and still perform very well in terms of accuracy. Before Squeezenet, people were focused on deeper models with more parameters, but this shows that acceptable accuracy does not necessarily depend on having a large number of parameters. By using Squeezenet and Alexnet, two models that have similar accuracy performance but drastically different numbers of parameters, we hope to observe how varying the number of parameters affects the learning properties.

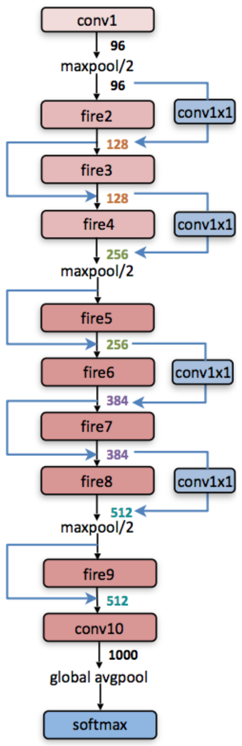


Figure 7. Squeezenet CNN architecture as shown in original paper [10].

The final model architecture we will consider is Resnet18 [11], shown in Figure 8, which is unique in that it uses the idea of residual functions and connections to boost the model capability. The residual functions also allow the Resnet architecture to have many more layers than traditional architectures, like Alexnet, which has proven to have significant positive impacts on performance. Resnet won the ILSVRC-2015 classification challenge and is considered to be one of the most capable models today. It is worth noting, Resnet has many variants which differ in the number of layers, including 18-layer, 34-layer, and 152-layer versions. Figure 8 shows the 34-layer version, however in this work we use the 18-layer version.

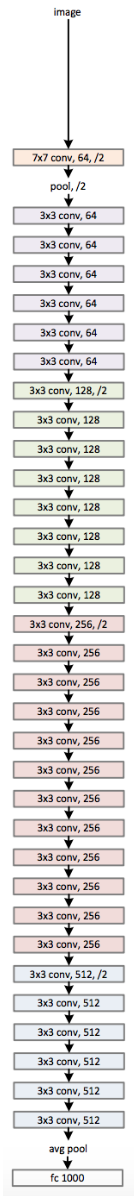


Figure 8. Resnet-34 Layer CNN architecture as shown in original paper [11]. Note, we will use the 18 layer variant of this model in this project.

The tool we use in this work is a python based deep learning framework called PyTorch [12]. One benefit of using PyTorch is that is supports the ModelZoo. In short, this means that we can download and use a variety of pre-trained models, which have all been pretrained on ImageNet at someone else's expense. Fortunately, all three of the models we use are available for download from the ModelZoo, in both pretrained and non-pretrained format. However, because the original models were trained on images of size 224x224x3 (224x224 spatial dimension with 3 color channels), this imposes a constraint that requires our images to be the same size. Fortunately, this can be easily accommodated with an image resize that preserves the aspect ratio.

One of the challenges in this project is to reshape and reinitialize the last layer of each of the models for use with our datasets. Recall, the models were originally trained on ImageNet, a 1000 class dataset, so the last layer has 1000 outputs. For our purposes, we only need 10 output nodes for the fruits-360-10-class dataset, 30 for the fruits-360-30-class dataset, and 5 for the flower dataset. This can be effectively resolved by removing the last layer of the network and re-initializing a new fully connected layer with the requisite number of inputs but our desired number of outputs. For example, in the case of Alexnet, the output layer is a 1000 node fully connected layer which is connected to a previous layer containing 4096 nodes. To use our 10 class dataset with Alexnet, we remove the final Alexnet layer, including all of the weights associated with it, and then reinitialize a new fully connected layer with 10 nodes that expects connections from the previous layer's 4096 nodes. Each of the models has a nuanced difference for how the last layer is remove and initialized, but the net result is the same in all cases which causes this difficulty to become a mere coding exercise.

We will now formally define the three methods used in our evaluation. The *scratch* method denotes the case in which we train a model from scratch and will serve as a baseline for comparison. After all, it represents a case where NO knowledge is transferred and is not considered a TL technique. In this method, we specify the model architecture (i.e. Alexnet, Squeezenet, Resnet18) and randomly initialize all weights as opposed to initializing the weights from a pretrained network. We then reshape the last layer of the network, specify that we want all parameters in the model to be adjusted by the optimizer (stochastic gradient descent algorithm), then run the training and validation. The *finetune* method follows the same process, except rather than initializing the weights randomly, we download and use the weights from the pretrained model. We then reshape the last layer, specify that *all* of the parameters are to be learned, then run the training and validation. The final method, *feature extraction*, involves using the pretrained CNN as a fixed feature extractor and only optimizes the weights in the last layer that we reinitialize. The process for feature extraction is similar to the previous two. We download the model with its weights initialized from the pretrained model, reshape the last layer, specify that we only want to update the *last layer* parameters with the optimization algorithm, then run the training and validation. Figure 9 is a flowchart of the process followed in this work.

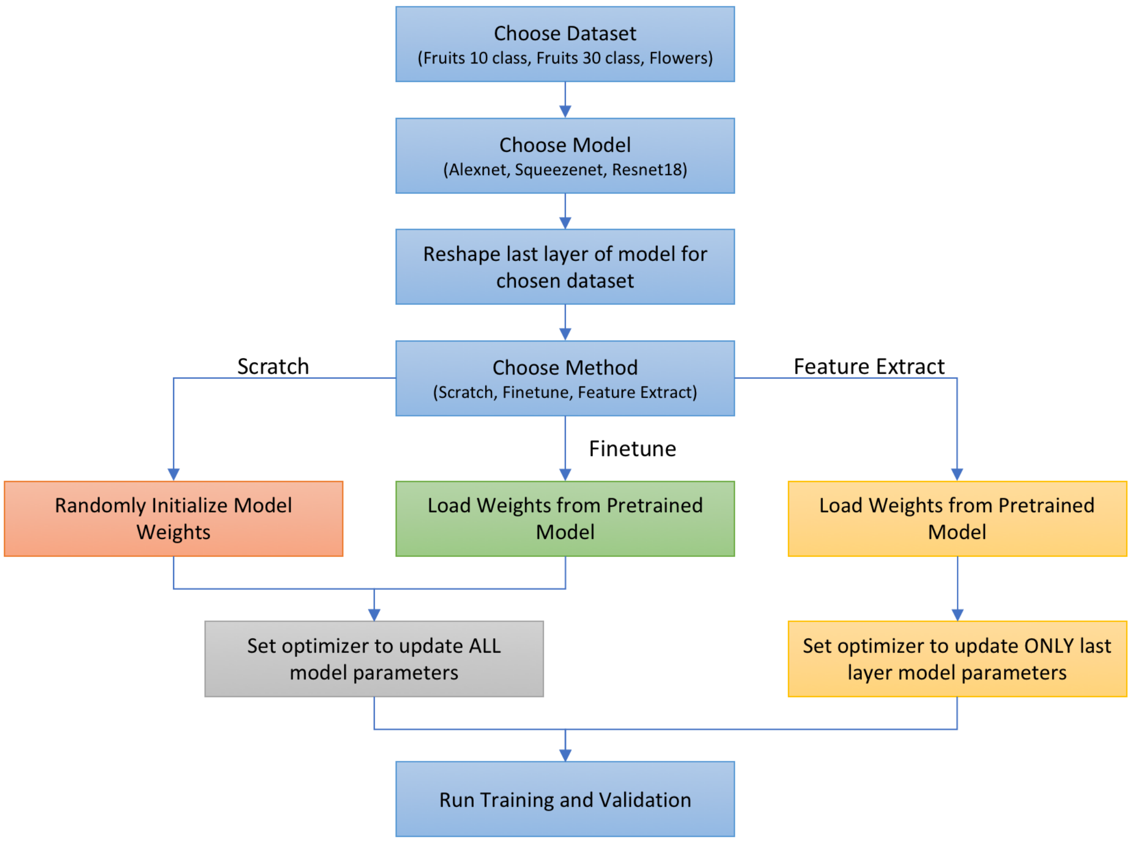
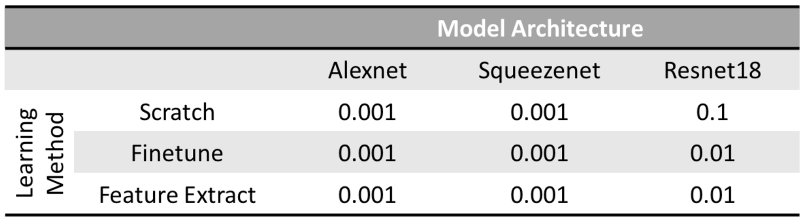

Choose Dataset
(Fruits 10 class, Fruits 30 class, Flowers)
Choose Model
(Alexnet, Squeezenet, Resnet18)
Reshape last layer of model for chosen dataset
Scratch
Choose Method
(Scratch, Finetune, Feature Extract)
Feature Extract
Finetune
Randomly Initialize Model Weights
Load Weights from Pretrained Model
Load Weights from Pretrained Model
Set optimizer to update ALL model parameters
Set optimizer to update ONLY last layer model parameters
Run Training and Validation

Figure 9. Flowchart of method used in this paper. This plot describes the process of choosing dataset, model, and method. It also shows how each method of learning is handled differently.

In an attempt to not change too many variables at once, we also follow a convention when selecting learning rates and the optimization algorithm. For all cases, the loss function is cross entropy loss, and the optimization algorithm is a stochastic gradient descent (SGD) algorithm with momentum of 0.9 and no weight decay. These values were taken from an existing TL tutorial [13]. Besides the model architecture, the only thing changing between runs is the learning rate; however, all runs have a fixed learning rate through the entire training period (i.e. no learning rate decay). The learning rate for each configuration tested is shown in Table 2.

Table 2. Table of learning rates used in the experiment. There is one entry for each of the learning-method/model configurations tested.



		Model Architecture		
		Alexnet	Squeezenet	Resnet18
Learning Method	Scratch	0.001	0.001	0.1
	Finetune	0.001	0.001	0.01
	Feature Extract	0.001	0.001	0.01

The reason not all learning rates are the same is because we have empirically obsesrved that if the learning rate is too high (i.e. 0.1 or 0.01) for the Alexnet and Squeezenet models, the learning curves are more irregular and do not always converge as smoothly in the allotted training window. However, that is not the focus of this work, so we will leave the effect of learning rate tuning to future work. As shown in the table, the Resnet model can handle a higher learning rate, especially when training from scratch. Also, the learning rates do fall within the suggested window of learning rates used for training in each of the papers describing the models [8][10][11].

With our models constructed and our methodology described, we will now describe the data augmentation scheme used as a part of training and testing. All of the data augmentation described is part of a standard practice for creating models that generalize well, and is originally described in an existing TL tutorial [13]. We mainly adopt 4 different data augmentation and preprocessing methods for both our training datasets and test datasets: resizing/scaling, cropping, flipping, and normalizing. Specifically, resizing/scaling ensures that our data is the necessary size (i.e. 224x244) and helps our network to accommodate more use cases in real life. Random cropping randomly eliminates some portions of the background, and by doing so, helps promote good generalization. Flipping preserves the object we are trying to recognize in the images but forces the CNN to learn the class from different viewpoints. All of these data augmentation methods help to increase the diversity of available data and attempt to help the classifier generalize better by not overfitting to the training set.

For the specific augmentations used on the training dataset, we first randomly crop the images and resize them to 224x224. Next, we horizontally flip each image randomly with 50% probability. Then, we convert the images to torch images/tensors (for use with the model) and normalize the tensors with the specified means and standard deviations for each RGB channel. The means of normalization are 0.485, 0.456, and 0.406 for red, green, and blue channels respectively (as shown in [12]).

For the augmentations to the validation dataset, we first resize the images to be 256x256 and then crop the images at the center to 224x224 to keep the sizes consistent. After that, we convert the images to torch images/tensors and normalize the tensors with the same means and standard deviations as those used in training. Notice, the augmentations differ depending on if we are training or validating.

In an effort to collect our result data points at a reasonable resolution while operating within our computational budget/limits, we adopt the following training and validation schedule. In total, for each run we train for approximately 3,000 iterations (rounded to the nearest epoch), with a batch size of 50, and run a full validation step every 20 training iterations. This means that for every run, we collect about 150 data points for our validation accuracy versus training iterations collection. The final detail of our method is how we will measure performance in this experiment. The most important metric we will measure is validation accuracy versus number of training iterations. Since the model is not trained on data from the validation set, validation accuracy gives a good estimate of generalization performance. Training iterations, on the other hand, convey an essence of time and do not depend on dataset size. Although an iteration may be slower on a CPU than on a GPU, the number of training iterations will be used as a universal approximator of time. In the ideal case, a method would converge to 100% validation accuracy in a very small number of training iterations. Another metric we will measure performance on is overall best validation accuracy across the three methods (*scratch, finetune, feature extract*). Unlike the previous metric, this measurement is only concerned with accuracy and does not consider cost or efficiency. To obtain the results, we will run each learning method on each of the models

(Alexnet, Squeezenet, Resnet18), on each of the datasets. Effectively, this will form a 3x3 grid of plots where columns represent model architecture, rows represent the dataset, and each plot will display three trend-lines depicting the accuracy of each learning method. The next section will describe the results of this experiment.

Results

Figure 10 shows the results for the primary experiment, where we measure validation accuracy versus training iterations. Each column represents one model architecture (Alexnet, Squeezent, Resnet18), each row represents a different dataset (Fruits-360-10-class, Fruits-360-30-class, Flowers), and each of the three lines per plot represent a different learning method (scratch, finetune, feature extract). In the plots, the blue line represent the model trained from scratch, the orange represents the model with full finetuning, and the green line represents the feature extraction learning. The x-axis of each plot is the number of training iterations that have passed (i.e. time). The y-axis of each plot shows the validation accuracy as measured on a separate validation dataset that is not being trained on. Also, the axes of all plots are at the same scale, where the x-axis spans between 0 and 3,000 training iterations and the y-axis spans 0-100% validation accuracy.

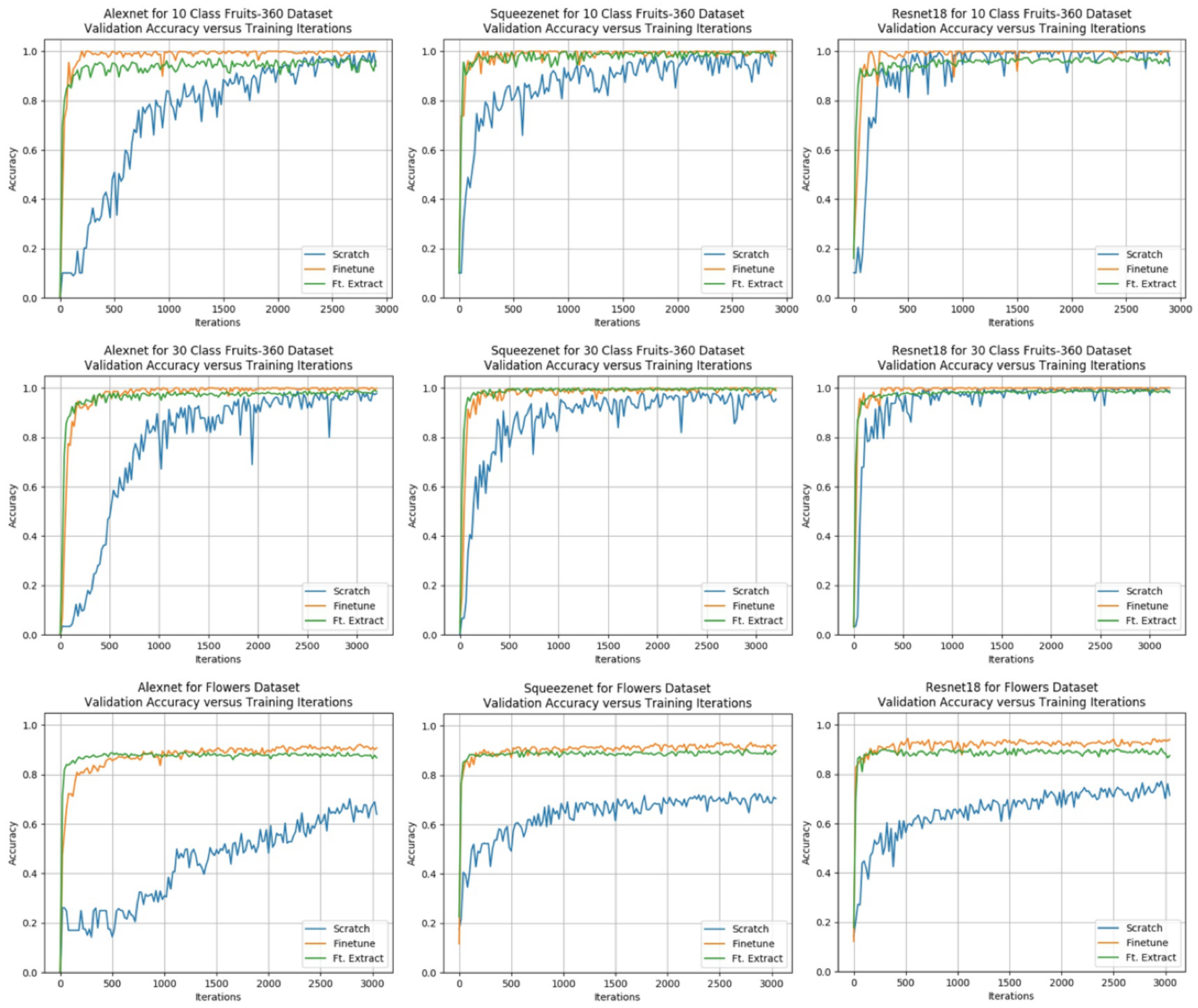


Figure 10. Plots of validation accuracy versus training iterations for the three learning methods (scratch, finetune, and feature-extract), as measured on three model architectures (Alexnet, Squeezenet, Resnet18) trained on three datasets (Fruits-360-10-class, Fruits-360-30-class, and Flowers). The blue lines represent models trained from scratch, the orange represent models that are finetuned, and the green represent models that are used as feature extractors.

There is a lot of information in this figure, so we will start by evaluating the performance down the rows. Roughly speaking, the rows are ordered by dataset complexity where the top row, of the Fruits-360-10-class, is intuitively the easiest because the images only contain the object with no background noise and there are only 10 classes. The middle row represents a seemingly 3x harder problem than the first row, as the data is of the same style but there are 3x as many classes to be learned. And the Flowers dataset in the last row presents the hardest problem because although there are only 5 classes, there is significant background noise in the images, and the object is not isolated like those in the Fruits-360 data.

Despite intuition, these results indicate that the fruits-360-30-class dataset was not significantly more difficult to learn than its 10 class counterpart since the first row plots look very similar to the second row plots. Additionally, none of the models show any significant differences in the convergence patterns between the first two rows. Furthermore, a general trend can be seen across the various methods. In each plot, the finetune and feature extract methods are shown to learn very quickly, converging to a high and steady validation accuracy before the 500th training iteration in all cases. However, the models trained from scratch show a much slower convergence pattern in the case of Alexnet and Squeezenet and only a moderate change in convergence pattern in the case of Resnet. Notably when comparing finetune and feature extract, the fine-tuning method shows slightly better steady state convergence performance than the feature extract method, especially in the case of Alexnet and Resnet. This result does make sense because all of the weights are being trained in the finetuned model, which is what contributes to the overall higher accuracy. However, the third row plots (of the Flower dataset) show a significant difference in learning behavior when compared against their fruit based counterparts. The most notable aspect of this difference is that none of the models achieve near 100% validation accuracy. We suspect that this may be explained by the background noise in the dataset images. However, despite the lower overall accuracy, the trends observed by comparing between methods are still consistent with the previous findings. Both the finetune and feature extract methods converge very quickly to a high performing (>90% validation accuracy) model, with the fine-tuning method slightly outperforming the feature extract model in terms of accuracy. The model learning from random weight initialization (scratch) converges much more slowly and to a lower overall accuracy.

Overall, looking down the rows shows several important patterns. First, regardless of dataset, transfer learning in the form of fine-tuning and feature extraction is a very effective way of achieving fast convergence of validation accuracy while simultaneously achieving a higher overall accuracy than the scratch can achieve in the same amount of time. Also, regardless of dataset, fine-tuning appears to have an edge over feature extract in the steady state performance, as fine-tuned models consistently show higher validation accuracy than the feature extract methods. However, both transfer learning methods (finetune and feature extract) have favorable properties over the non transfer learning method (scratch).

Now, we will consider the differences between the columns where each column represents a different CNN model architecture. Before discussing the observed results we can develop an intuition based on the models themselves. First, the Alexnet model is the oldest and most primitive design of the three. Its simple structure has since been improved upon, but it serves as a good benchmark for comparison. The Squeezenet model was designed to have similar performance as Alexnet, but with 50x fewer parameters. Given our limited training time, we may expect that only having to learn a fraction of the parameters may lead to better learning behavior in terms of convergence speed and final accuracy. Lastly, Resnet18 has many layers and many parameters, but has shown remarkable results in ILSVRC-2015, so it is unclear how the residual functions will affect the learning process.

The first observation here is that Resnet18 converges much more quickly than the other models for the scratch method, especially in the case of the Fruits-360 datasets. Still, for each dataset, its scratch convergence is competitive with the convergence of the transfer learning methods (FT and FE). Since Resnet has a higher scratch learning rate, it is unclear how much of an impact learning rate has, but we will leave this for future work. Also, as suspected, the Squeezenet models do show an advantage over the Alexnet models in two ways. First, the Squeezenet models trained from scratch appear to learn faster than the Alexnet models trained from scratch, as the elbow in the curve comes at around 500 iterations for Squeezenet and at about 1000 iterations for Alexnet. Second, in the Alexnet model there appears to be a clear gap between the accuracy of the finetune model and the feature extract model, especially for the Fruits-360 datasets, with the accuracy of the finetune models being higher. However, there is no such gap in the Squeezenet model. Overall, from looking across the columns at the performance of the different models, the Resnet model appears to be a clear winner, outperforming the others in almost all aspects. Squeezenet also appears to have an edge over Alexnet, which is most likely due to the smaller number of parameters taking less time to learn.

Another desirable result to measure is outright performance via best validation accuracy. This result does not consider learning speed, but rather, what is the best validation performance a given model produces over the 3,000 iterations. To measure outright best validation accuracy we can use the data from the plots and grab the highest validation accuracy from each of the lines. This information is shown in Table 3.

Table 3. Table of best overall validation accuracies. There is one entry for each of the datasets/learning-methods/model-configurations tested.

	Alexnet			Squeezenet			Resnet18		
	S	FT	FE	S	FT	FE	S	FT	FE
Fruits 10 Class	99.9	100.0	97.2	100.0	100.0	100.0	100.0	100.0	97.5
Fruits 30 Class	99.0	100.0	99.0	98.9	99.9	99.9	100.0	100.0	99.2
Flowers	70.2	92.1	89.1	73.0	93.3	90.3	77.0	94.4	90.3

Table 3 provides a different way of looking at some of the same data from Figure 10 and confirms our takeaway that the finetune (FT) method produces the best results in regards to overall accuracy across all of our datasets and models. Additionally, Table 3 helps restate that the Resnet18 model is superior to the Alexnet and Squeezenet model in regards to maximum accuracy. One interesting finding here is that the Resnet trained from scratch on Fruits-360 outperforms the feature extract transfer learning method. A possible explanation may stem from the inherent limitation of the feature extract method in the sense that none of the early layer weights are being modified, so the model is "stuck" with all of the transferred weights, which may be suboptimal for the new problem. This table also quantifies the interesting performance difference between the methods on the Flowers dataset, where the transfer learning methods are shown to significantly outperform the scratch method, by about 20% validation accuracy for Alexnet and Squeezenet, and about 15% on Resnet.

For the Flowers dataset, because none of the models under any conditions converge to near 100% validation accuracy, we wish to measure the effects of training longer. We pick the most promising model, Resnet18, and run training for 3x longer, to 15,000 training iterations. The results of this run are shown in Figure 11.

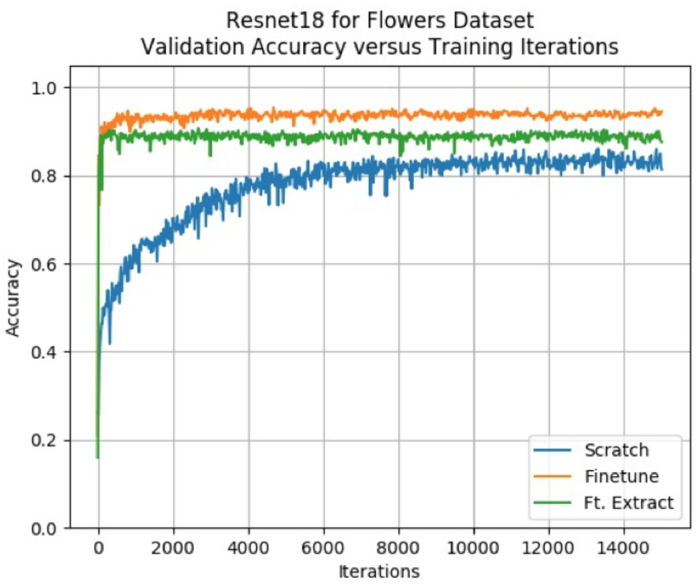


Figure 11. Plot of validation accuracy versus training iterations for the three learning methods (scratch, finetune, and feature-extract) as measured on Resnet18, trained for 15,000 iterations on the Flowers dataset.

Although it is somewhat surprising that none of the methods approached 100% accuracy, this plot does affirm (for the most part) the trends discussed above, which have been repeated throughout this section. Namely, both transfer learning methods outperform the model trained from scratch in terms of accuracy and convergence speed. Also, of the two transfer learning methods, the finetune method reaches a higher convergence accuracy than the feature extract method.

Conclusions

The problem addressed by this paper is to measure/quantify the effectiveness of transfer learning. Transfer learning has the potential to greatly speed up the training times when training models on a new dataset. It also has the potential of producing overall higher performing models. This would mean researchers would not have to spend so much time training, and a person with a great idea would not be blocked by training times if that idea involved training a CNN. Recall, transfer learning is the process of transferring knowledge learned from a source domain to learning a new target domain. In the space of machine learning, transfer learning is realized by using model weights learned from training on a source dataset, like ImageNet, as the weight initialization when trying to learn a new dataset. In this work, we measure the effect of transfer learning with three models, Alexnet, Squeezenet, and Resnet18, across three challenging datasets. As a baseline, we train the models from scratch where the weights have been initialized randomly, and evaluate the models in terms of convergence speed and overall accuracy, as measured on a validation set. We also evaluate two transfer learning techniques called fine-tuning and feature extraction. Here, fine-tuning means we used the pretrained weights in the initialization then update all of the weights in the learning process, and feature extraction means we use the pretrained CNN as a feature extractor, and only learn the last layer weights.

This work shows transfer learning techniques yield promising results. Across all configurations, both transfer learning methods show fast convergence speeds and achieve high validation accuracy, as compared to the baseline method of training from scratch. Another interesting result is that there is not a clear relationship between number of classes and convergence speed or accuracy. In the alloted training window, the transfer learning methods outperformed the scratch training method by at least 13% in final validation accuracy across all models on the Flowers dataset. Of the model architectures tested, Resnet18 was in all cases the clear best, in terms of both convergence rate and overall best validation accuracy. Also, when comparing the two transfer learning methods directly, the fine-tuning method that updates all of the model's weights consistently outperformed the feature extract method. Of the 27 configurations tested, the Resnet18 with fine-tuning shows the best results and would be a good place to start for anyone wanting to train a powerful model on a custom dataset. For future work, to further improve the performance of these models, it would be important to measure the effect of regularization, learning rate scheduling, and the optimization algorithm (e.g. Adam) on the learning process. Also, these results should be verified on larger and more difficult datasets when more computing resources are available.

Roles

Nathan Inkawhich - Nathan adapted/wrote the code to run the models with our datasets, ran the experiments, collected the data, and plotted the results. He also wrote the methods and results sections of the paper and was instrumental in production of the video and compilation of the report as a whole.

Yifan Li - Yifan did some research about the target datasets and picked our final datasets. He wrote the data section and the data augmentation/preprocessing portion of the methods section of the paper. He also helped with the video production.

Hayden Bader - Hayden primarily wrote the introduction and background section of the report, and assembled the report into this notebook.

References

1. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014
2. D. C. Cireşan, U. Meier and J. Schmidhuber, "Transfer learning for Latin and Chinese characters with Deep Neural Networks," The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, QLD, 2012, pp. 1-6.
3. A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, "Large-Scale Video Classification with Convolutional Neural Networks," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1725-1732. doi: 10.1109/CVPR.2014.223
4. S. J. Pan and Q. Yang, "A Survey on Transfer Learning," in IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, Oct. 2010.
5. Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. 2007. Boosting for transfer learning. In Proceedings of the 24th international conference on Machine learning (ICML '07), Zoubin Ghahramani (Ed.). ACM, New York, NY, USA, 193-200. DOI=http://dx.doi.org/10.1145/1273496.1273521 (http://dx.doi.org/10.1145/1273496.1273521)
6. Pan, Sinno Jilian, et al. “Transfer Learning via Dimensionality Reduction.” Proceedings of the Twenty-Third AAAI Conference on Artifical Inteligence.
7. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. Int. J. Comput. Vision 115, 3 (December 2015), 211-252. DOI=http://dx.doi.org/10.1007/s11263-015-0816-y (http://dx.doi.org/10.1007/s11263-015-0816-y)
8. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 1. Curran Associates Inc., USA, 1097-1105.
9. The 9 Deep Learning Papers You Need To Know About. https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html (https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html)
10. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. CoRR, abs/1602.07360.
11. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

12. Pytorch Transfer Learning Tutorial. http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html (http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html)
13. Paszke, A., Gross, S., & Lerer, A. (2017). Automatic differentiation in PyTorch.
14. Horea Muresan, Mihai Oltean, Fruit recognition from images using deep learning, Technical Report, Babes-Bolyai University, 2017
15. Mamaev, A. (2018). Flowers Recognition [dataset]. Retrieved from https://www.kaggle.com/alxmamaev/flowers-recognition (https://www.kaggle.com/alxmamaev/flowers-recognition)
16. Hong-Wei Ng, Viet Dung Nguyen, Vassilios Vonikakis, and Stefan Winkler. 2015. Deep Learning for Emotion Recognition on Small Datasets using Transfer Learning. In Proceedings of the 2015 ACM on International Conference on Multimodal Interaction (ICMI '15). ACM, New York, NY, USA, 443-449. DOI:http://dx.doi.org/10.1145/2818346.2830593 (http://dx.doi.org/10.1145/2818346.2830593)
17. Stanford cs231n Transfer Learning. http://cs231n.github.io/transfer-learning/ (http://cs231n.github.io/transfer-learning/)
18. Machine Learning Mastery Transfer Learning. https://machinelearningmastery.com/transfer-learning-for-deep-learning/ (https://machinelearningmastery.com/transfer-learning-for-deep-learning/)
19. Ghafari, Ehsan & Bandarabadi, M & Costa, Hugo & Júlio, Eduardo. (2014). Full Paper-BMC 2012-EG-04-1 revEJ.
20. Carreira, J., & Zisserman, A. (2017). Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4724-4733.
21. A. Quattoni, M. Collins and T. Darrell, "Transfer learning for image classification with sparse prototype representations," 2008 IEEE Conference on Computer Vision and Pattern Recognition, Anchorage, AK, 2008, pp. 1-8. doi: 10.1109/CVPR.2008.4587637

```
In [ ]:
```