Plankton Classification Aided by Transfer Learning

Omer Ijaz Sheikh(20495239), Shuai Peng(20529148), Zhaokai Liu(20724504)

University Of Waterloo, ECE657A, Group 4

Abstract. We implemented an image classification experiment which uses residual convolutional neural networks(CNN) to classify on the Plankton Dataset from Woods Hole Open Access Server(WHOAS). This experiment was intended to explore the viability of transfer learning on highly dissimilar images by reutilizing a resnet50 classifier originally trained on ImageNet by retraining it to detect and classify greyscale images of plankton. We compare our results achieved with this model to those of a much smaller but similar network trained from scratch, which ultimately outperforms our transfer learning model. Overall, this project tells us that transfer learning is difficult to use when the target dataset and the original dataset have very different general features. At the same time, it is worthwhile to compare different models and parameters in order to get the best result.

1 Introduction

Plankton classification is an important problem because their population levels are the perfect measure for determining the health of an ocean ecosystem. Modern techniques involve taking a large number of underwater images - so many that manual human classification is too slow to analyze even a small subset. As a result, there is significant interest automatic image classification for plankton images.

Until recently, computers had a much harder time than human in telling apart different objects in the real world. In the past few years, however, deep learning has enabled computers to classify images with much greater precision. For images of objects than can normally only be classified by experts, deep learning is proving to be an extremely powerful tool for classification.

However, training deep learning models is extremely expensive both computationally and in terms of data required. One way to work around these limitations is transfer learning.

2 Literature Review

We reviewed a few important papers that relate to our project:

Deep Big Simple Neural Nets Excel on Hand-written Digit Recognition [1]: One of the earlier papers that demonstrated the utility of deep learning for image classification. They achieved better results than neural networks in the past through by training in parallel on GPUs, which enabled them to train much deeper networks than before. They also observed that their model had a tendency to overfit on their training data. To reduce overfitting, they used data augmentation to train on deformed images combining affine and elastic deformations. These data pre-processing techniques are still relevant today, even with much deeper and more complex neural networks. In project, we used similar affine transformations to reduce the possibility of overfitting in our model.

Deep Residual Learning for Image Recognition [3]: Popular neural network models for image classification tended to be highly complex with an extremely large number of parameters. Up until 2015, adding more layers to an already deep (up to 30 layers) convolutional neural network would generally lead to a reduction in accuracy. This paper introduced deep residual networks - "resnets" - which were capable of being much deeper - to the tune of 150 layers - without suffering from the same accuracy degradation. The paper also shows that such networks can be much less complex despite their depth and still achieve better accuracy. An ensemble of these resnets achieved 3.57% error on the ImageNet test set and won first place in ILSVRC 2015. In addition, weights trained on ImageNet for certain resnet models (resnet50 and resnet101) are publicly available. It was for these reasons that we decided on using resnet based networks for our image classification models.

How transferable are features in deep neural networks? [5]: The authors present a method to measure the generality of the features from a given layer in a neural network. They experimentally show two of the causes of accuracy degradation in transfer learning. First, transferring

weights from a different problem domain results in the loss of feature interaction between layers. Secondly, the higher layers tend to be specialized to perform well on their original task, and thus these layers do not transfer well. However, their experiments also show that with fine-tuning of the transferred features, transfer learning can even outperform training from scratch. They theorize this is due to less over-fitting in the lower layers. We similarly fine-tune our lower layers in order to improve our network's results.

Dermatologist-level classification of skin cancer with deep neural networks [2]: In this paper, a classifier for skin cancer was introduced which was able to identify skin cancer with the same level of competence as a dermatologist. The image classification model is built using transfer learning on an Inception V3 network trained on ImageNet. This was the first case in which transfer learning was successful on transferring features between such widely different data domains; ImageNet's imageset is not at all similar to images of various kinds of skin cancer this classifier was trained to detect. This paper pushed the boundary what could be accomplished with transfer learning.

3 Description

The goal of this project was to build a classifier for categorizing greyscale images of plankton. In doing so, we explored the viability of using transfer learning to utilize a model pre-trained on a highly dissimilar set of images (ImageNet) and compared the results of this model to that of one trained from scratch. We used residual networks for all models.

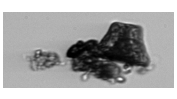
Transfer learning is a machine learning technique in which a model that is trained to do one task is re-trained to do another task. In the context of neural networks, this means re-training the final fully connected layer for the new classes and using the already trained weights for the lower layers. This can be significantly cheaper than training a network of equal complexity from scratch. As most popular modern deep learning models for image classification have millions of parameters, transfer learning is currently an extremely useful technique for building a classifier for a given problem. In the coming years, transfer learning is thus expected to gain traction and be used for a growing domain of machine learning problems.

Deep residual networks are a type of convolutional neural networks that do not suffer from the problem of vanishing gradients. This allows residual networks to be extremely deep. For their rather extreme depth, such networks are often not that complex, when first introduced even 150 layer residual networks had a much smaller number of parameters than other contemporary networks that were far less deep.

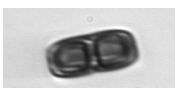
4 Dataset

This project uses images of 103 categories of plankton made available from the Woods Hole Oceanographic Institution (WHO-I). This dataset includes 1.7 million greyscale images, which are only a small subset of the 700 million images taken by the Imaging FlowCytobot (IFCB). The dataset is publicly available at `http://hdl.handle.net/10.1575/1912/7341` [4]. The data is separated by year; we used the entire dataset from 2006 to 2014.

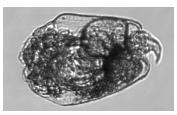
The images are of various sizes but are generally smaller than 350x350 and are not necessarily square. In addition, the category distribution is highly imbalanced; 90% of the images belong to just 5 classes. In the extreme ends, we have classes with hundreds of thousands of images and classes with less than 3 images.



(a) Bidulphia



(b) Ephemera



(c) Zooplankton

Fig. 1: Some examples images from three classes

5 Data Preprocessing

5.1 Removing Categories

The image set included a few classes with incompletely annotated data; we removed these following categories from our analysis:

- Chaetoceros_other
- diatom_flagellate
- G_delicatula_detritus
- other_interaction
- pennates_on_diatoms
- mix

Of the above, the "mix" category was the largest, with nearly 1 million images and accounted for almost half of the original dataset. After the removal of these classes, there were approximately 880000 images left.

In addition, the following pairs of classes were combined:

- Ditylum_parasite was added to Ditylum
- G_delicatula_external_parasite was added to Guinardia_delicatula

Of the remaining classes, those with less than 40 instances were also discarded, purely to avoid classes with too small a training set.

5.2 Data Splitting

Because of the sheer size of the dataset, we pre-split our images into training, validation and test sets. Because some classes still had a very small number of samples, we split the data for every class individually. As a result, our train, test and validation sets all had the same class distribution. For every class, we reserved 50% of the images for testing, 45% for training and 5% for validation. The validation set was purposely kept small in order so that we could quickly perform a validation check after every epoch in order to catch possible overfitting.

5.3 Data Augmentation

We used data augmentation to artificially inflate the size of our training dataset by randomly applying a set of affine transformations. A modified version of each input image is generated every training pass. Training a model on such artificial images can help reduce overfitting; especially for classes with only a small number of samples. The set of transformations we use with their parameters are:

- rotation of up to 360 degrees clockwise
- horizontal and vertical flipping
- horizontal translation up to 20% of the image's total width
- vertical translation up to 20% of the image's total height
- shearing with a maximum intensity of 0.2 radians
- zooming within the range [0.98, 1.02] at any point in the picture

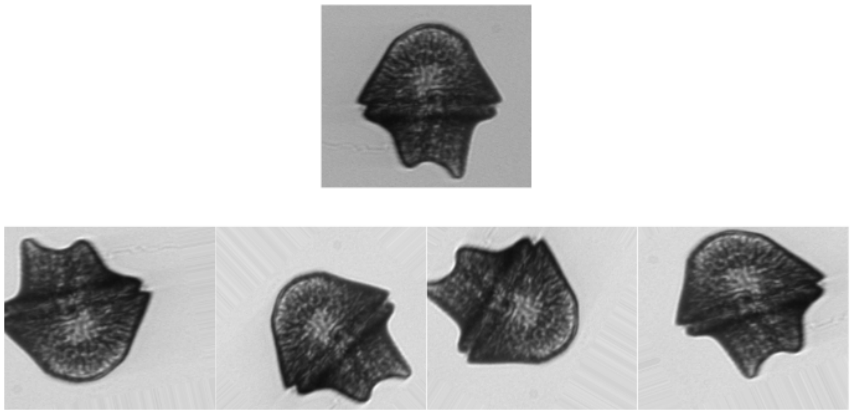


Fig. 2: Examples of image transformations

5.4 Sampling Schemes

Because of the extremely imbalanced class distribution, we experimented with a small number of sampling schemes. These are referred to as sampling schemes 1, 2 and 3. These were as follows:

1. Undersampling classes with more than 5000 samples to only 5000 samples by sampling without replacement. Classes with less than 5000 samples were used as-is.
2. Undersampling classes with more than 1000 samples to only 1000 samples by sampling without replacement. Classes with less than 1000 samples were upsampled to 1000 instances by sampling with replacement.
3. No undersampling or upsampling of any classes, use original data distribution as-is.

6 Implementation Details

6.1 Resnet

For our classifiers, we use deep residual networks as introduced and described in [3]. A short explanation for a basic residual block is included here.

A residual network consists of several residual blocks. Each block contains a certain number of convolutional layers and a shortcut. Provided that they are of the same shape, the input of the first layer is added to the output of the final layer to produce the output of the residual block. If the input is not of the same shape as the output of the final layer, then the shortcut includes a layer to increase the input dimensions. In resnet18 and resnet 34 each block has 2 convolutional layers. In resnet50 each block has 3 convolutional layers.

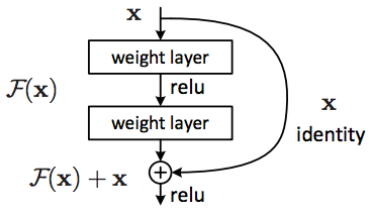


Fig. 3: A basic residual block. Image from [3]

6.2 Models

We built and compared the results of three models. These are referred hereafter as models 1, 2 and 3. The network for models 1 and 2 is "resnet50", a residual convolutional network introduced in [3]. We use pre-trained weights trained on the ImageNet dataset.

For model 1, the last fully connected layer is replaced and trained while all the lower layers are kept "frozen" - their weights are not updated while training. This reduces training time as there is only one backpropagation step required.

For model 2, once again the last fully connected layer is replaced and trained. However, the previous lower layers are not kept frozen. They are initialized with the weights trained on ImageNet and fine-tuned while training on the new classes. Since all the layers are being trained, each epoch takes as long as it would to train from scratch; though fewer epochs may be needed due to faster convergence.

Model 3 was a custom network trained from scratch. Ideally, this should have also been a resnet50 architecture. However, because of the computational resources required to train a resnet50 network from scratch, we instead opted to use a smaller 14 layer residual network. This network is very similar to the 18 layer resnet18; the only difference is that the last 2 residual blocks have been removed. Batch normalization is perfor,med after every convolutional layer.

All models used the Adam(adaptive moment estimation) optimization algorithm and used the categorical crossentropy between predictions and targets for the loss function. For models 1 and 2, these were preferred as the original weights had been trained under these parameters.

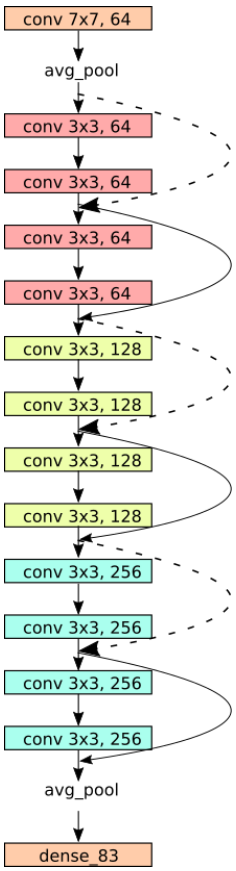


Fig. 4: The 14-layer residual network. The filters in the convolutional layer rise from 64 to 256. Dotted shortcuts represent 1x1 convolutions with appropriate strides.

Models 1 and 2 took input images in the same format as the original model trained on ImageNet. In other words, the input image had to have dimensions of 224x224 and have Red-Green-Blue colour channels. Since our original images are greyscale this meant duplication of data for our colour channels. In contrast, since model 3 was trained from scratch we were able to use input dimensions of our choosing. To keep the number of training parameters low we used dimensions of 150x150 with a single colour channel.

We trained models 1 and 3 on all three sampling schemes. Because of the sheer cost of training model 2, it was only trained using the sampling scheme that gave us the best results for models 1 and 3. This turned out to be sampling scheme 3, i.e. no resampling.

The top prediction accuracy and (macro-)averaged f1-score for all the classes were used as evaluation criteria.

6.3 Tools

All code for this project was written in python3 using the following libraries:

- tensorflow
- keras
- scikit-learn
- numpy
- h5py

Weights pretrained on ImageNet for resnet50 were obtained from the keras application zoo.

Models 1 and 3 were initially trained on a GTX1050 and then on a google cloud v4-machine instance with access to a tesla K80. Model 2 was trained on a google cloud v4-machine instance with access to 4 Tesla K80s. The training times per epoch on the cloud v4-machine are given below:

Model	GPUs (Tesla K80)	Approximate Training Time Per Epoch	Training Epochs
1	1	20min	30
2	4	3h 30min	10
3	1	10min	30

6.4 Code Layout

The code used for this project is included along with this report.

The library "h5py" is used to store the models in the h5df format in between training runs. The submitted code includes the script "network.py" for creating models 1 and 2 and saving them to "resnet50_tl.h5" and "resnet50.h5" respectively and the script "network_small.py" for creating the 14 layer residual network and saving it to the file "resnet14.h5". Any of these model could be trained further by running the "network_resume.py" or tested by running "network_test.py".

Instead of working with tensorflow directly, the high level library "keras" is used. Models 1 and 2 were initialized by simply loading a pretrained resnet50 model and replacing the the final dense layer (and its associated flatten layer which simply flattens the input into a 1D vector). For model 1, the previous 49 layers are also marked untrainable in order to freeze them. Model 3 is created in a similar manner to how "keras" initializes resnet50. The two types of residual blocks are first defined, blocks with input and output of the same dimensions and with different dimensions. The model is then composed of such residual blocks to create the architecture shown in figure 4.

Code for splitting the data into train/test/validation sets and sampling the training set is included in the python notebook "Data_split.ipynb".

7 Results

The table below summarizes the validation accuracy and f1-score, as well as the validation f1-score for the three models under the three sampling schemes. Model 2 (fine-tuned resnet-50) was only training on the third sampling scheme, i.e. the original distribution was used as-is without any over-sampling or under-sampling.

Model	Sampling Scheme	Training Accuracy	Validation Accuracy	Validation F-1 score
1	1	0.772	0.228	0.136
	2	0.815	0.174	0.160
	3	0.931	0.590	0.481
2	3	0.931	0.773	0.60
3	1	0.87	0.50	0.45
	2	0.91	0.52	0.5
	3	0.927	0.855	0.62

In general, using the original data distribution without re-sampling gave the best results for all models. In addition, model 1 had poor validation accuracy in all cases. We propose that is because training only the final fully-connected layer was insufficient due to the difference between the plankton images and ImageNet images.

Results on the test set were only calculated for the models 2 and 3, both using the original class distribution.

Model	Test Accuracy
Model 2	0.59
Model 3	0.85

It should be noted that due to resource constraints, model 2 was not trained for enough epochs to reach the same degree of convergence as model 3.

Given that the 14 layer resnet based convolutional network overall performed much better than our 50 layer fine-tuned one which employed transfer learning, a possible reason for this is that our image space is too different from ImageNet for transfer learning to be a viable technique here. This is not a perfect comparison however; a better one would be for the results of model 2 to be compared with those of a resnet-50 model trained from scratch. In our experiment, the difference in performance may be because of the architecture differences.

Another possible reason the fine-tuned model did not perform as well is because it was not trained for as many epochs due to resource constraints. A hint towards this is that model 2 was out-performing model 3 in terms of training accuracy. Although its validation accuracy was too erratic to make any a solid conclusion, it was performing better than model 3 for the same number of training epochs.

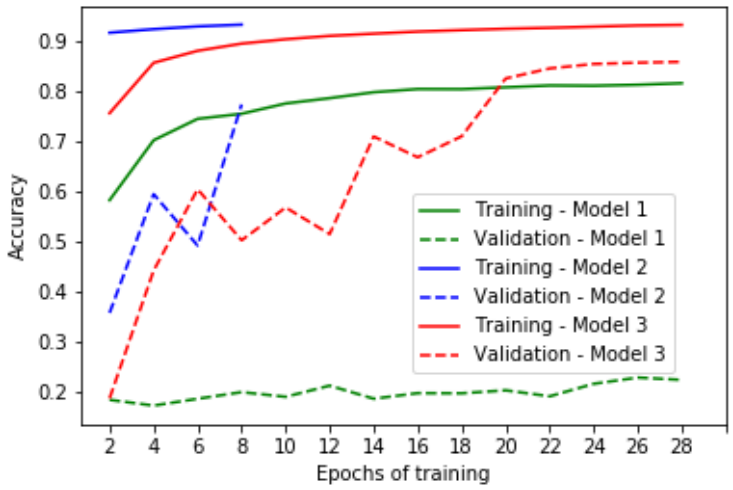


Fig. 5: Training and validation accuracy for the three models with the best sampling scheme.

Another thing that might be concluded from the above is that with transfer learning, the model does not have to be trained for as many epochs to achieve acceptable results. Indeed, by fine-tuning pretrained weights, the model was able to achieve over 90% accuracy after only the very first training epoch. This is, of course, the reason that transfer learning models do not require as many resources to train as a network of the same size and architecture trained from scratch.

In general, all of our classifiers performed better on classes with a larger number of samples. For the best performing classifier (model 3), no category had a f1-score of 0, meaning the classifier was guessing the full range of classes. That said, a good high f1-score correlated with a large number of samples. None the categories with at least 5000 training images have a f1-score below 0.6.

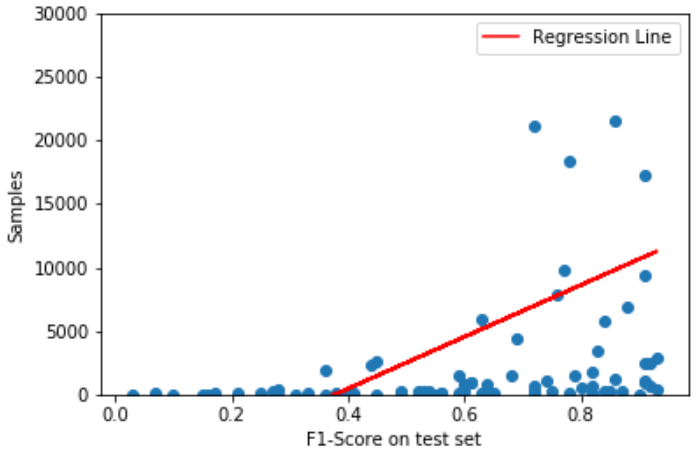


Fig. 6: Relationship between F1-Score and number of training samples per class for the best classifier. Each point represents a class of plankton. Not included in this figure are 2 classes with >60000 samples.

In order to spot overfitting, we ran validation checks after every epoch of training. We found that sampling scheme 2, which used downsampling and upsampling to force every category to 1000 samples was prone to overfitting, as evidenced by the figure below. Due to the low number of validation passes, the results on our validation set were expected to be slightly erratic, but a downward trend in accuracy over 10 epochs strongly hints towards overfitting. A possible reason behind this may the reduction in variation of the training data. While different augmented images were generated for each training pass, the re-use of images of undersampled classes greatly reduced the variation in the data the classifier was training on. Combined with the undersampling of majority classes, this may have introduced overfitting.

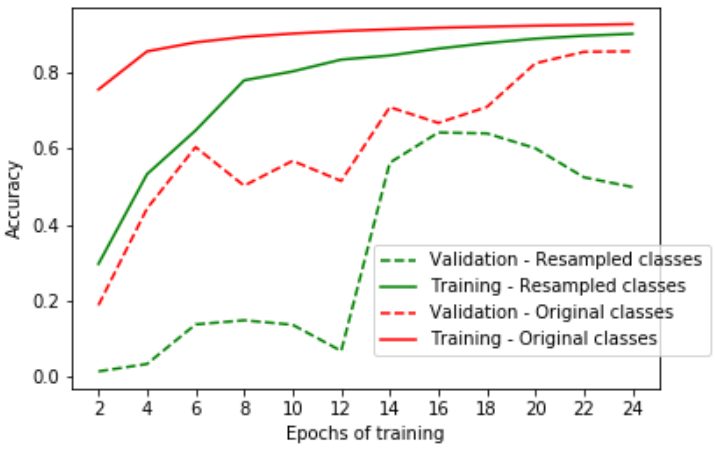


Fig. 7: Training and validation accuracy for model 3 with sampling schemes 2 and 3. Sampling scheme 3 does not show overfitting but scheme 2 does.

7.1 Conclusion

We summarize the following conclusions from our results:

– As in [5], a fine-tuned model with transferred weights performed significantly better than training only the last layer of a transferred model. This is to be expected from the extreme dissimilarities between our dataset and ImageNet.

- While not conclusive, our experiments hints towards models with transferred features being faster to converge (in terms of the number of training epochs required) in comparison to models that are trained from scratch.
- That said, the much smaller 14-layer residual network performed much better than our model with transferred and fine-tuned features. This is likely because our images have are even more dissimilar to skin cancer images used in [2]. It should be noted that our fine-tuned network was not trained for as many epochs as the 14-layer network due to the computational resources required.
- Resampling of data can have adverse effects on the performance of deep neural networks, especially if it reduces the variation of the original dataset.
- Given the results for model 3 on our test data, we suggest that residual networks, even relatively small ones are a good classification tool for classifying plankton images, even with a highly imbalanced class distribution.

8 Possible Improvements

We considered a few possible ways this experiment could be improved.

As mentioned before, one possible reason the fine-tuned transfer learning model did not perform as well could be because it had not converged to the same number of significant figures as the other to models. Due to the limited number of compute engine credits, model 2 was only trained for 10 epochs. The results may have been different had we been able to train it for longer.

Another possible reason could have been that original and target image spaces are just too different. This would imply that transfer learning is simply not the right tool for this task. However, it is also possible that the 14-layer residual network performed better due to its different architecture. For an ideal comparison, a resnet50 model should have also been trained from scratch. Of course, due to the sheer number of parameters, such a model will require exponentially more computation time than the 14-layer model.

One possible way to produce a classifier that performs better overall is to use an ensemble of neural network classifiers. This was the technique used to originally obtain a test error of only 3.59% [3] on ImageNet in ILSVRC 2015. Each classifier in the ensemble can output its class probabilities for a given input image, and the overall prediction can be made by some measure of aggregating the results of the classifiers.

In an attempt to deal with the class imbalance we attempted to use resampling schemes to modify the class distribution. That only worsened the performance. An alternate strategy would have been to utilize a penalized loss algorithm that weighs classes inversely to their frequency. By penalizing incorrect predictions for underrepresented classes more severally in the loss functions, the model may have been able to better train on those classes.

References

1. Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep, big, simple neural nets for handwritten digit recognition. Neural Computation 22(12), 3207–3220 (Dec 2010)
2. Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M., Thrun, S.: Dermatologist-level classification of skin cancer with deep neural networks. Nature 542, 115– (Jan 2017), http://dx.doi.org/10.1038/nature21056
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 770–778 (2016)
4. Sosik, H.M., Peacock, E.E., Brownlee, E.F.: Annotated Plankton Images - Data Set for Developing and Evaluating Classification Methods. http://hdl.handle.net/10.1575/1912/7341
5. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. pp. 3320–3328. NIPS'14, MIT Press, Cambridge, MA, USA (2014), http://dl.acm.org/citation.cfm?id=2969033.2969197