Stefano Di Filippo - 3079911

iNaturalist - Species Classification

Deep Learning for Computer Vision

January 31, 2023

1 Introduction

In this report, we describe the approach that has been taken to the task of classifying images of the iNaturalist data set to animal and plant species. In particular, the objective of the project has been performing image classification to the 'category' classes, focusing on a subset of the full available data (the 'split_1' and 'split_2' parts). Then, the task has been restricted to a fraction of this data, as it will be discussed. The 'supercategory' information has been discarded.

2 Data Preparation

As a first step of preparing the data for modeling, the structure of the data has been inspected. It has been observed that all images have JPEG as format, thereby allowing us to quickly ignore this piece of information in preprocessing. On the contrary, pictures present three different modes: CMYK, L, and RGB. Out of a total of 124481 images, 124390 are RGB, whereas only 89 of them have L mode, and just 2 CMYK. Since non-RGB images are less than 0.1% of the samples, it has been decided to discard them, as this was unlikely to change the performance of the models. Then, we have found that there are 1063 different image resolutions in the data. In order to make the data ready for the models, images have to be resized to a common shape. This involves destroying some information but, as long as the resizing is not particularly extreme, it should not impact performance significantly. We chose a common resolution of 200x200 pixels. Since the average resolution is 743x644 pixels, this operation is also greatly reducing the size of the input tensors, thereby allowing for faster computations.

To allow for even faster training and more experimentation with models, we have decided to subsample the total data set of images, and consider only this fraction for the task. As categories can differ greatly in the number of images they contain, this cannot be conducted by sampling images completely at random, as this risks producing classes with very few instances. Instead, the chosen approach has been to select a subset of all categories, and keep all images for those classes. This has been done by selecting 10% of all classes, starting from the most numerous and in descending order by numerosity. This operation has been conducted by discarding non-RGB images in counting the frequency of each class. Since the frequency of categories seems to exhibit a power law distribution, even taking this relatively small fraction of classes will still select a large number of instances. Indeed, starting from 1003 categories, for a total of 124390 images, the data have been reduced to 100 categories, for a total of 58226 images. Halving our instances allowed us to test more and more complex models, but also preserved enough data to hope of reaching decent performance. Since the most frequent category has around 5.6% of images in our subsample, then 5.6% is our baseline performance, at least in terms of accuracy: this is the accuracy which the simplest model, which just assigns each image to the most probable class overall, would obtain.

Then, for each of the 100 classes, the images have been assigned as training, validation, or test images, with the following spread: 60% of training data, 20% of validation data and 20% of test data. Finally, the preprocessing part was concluded by rescaling the pixels to the [0, 1] interval and by setting data generators.

3 Modeling: Optimization

Having prepared the images, the next step is start trying some models on the data. The only model architecture that has been implemented is the convolutional neural network, and, based on this architecture, different configurations have been tested. In general, for all models, we have set RMSprop as the optimizer, as it is generally a good choice, and the categorical crossentropy as the loss function, since we are concerned with a multiclass classification. Other common features are 3x3 filters, ReLU activations, stride set to 1, and padding such that convolutions do not resize their inputs. Batch size for minibatch stochastic gradient descent was chosen to be 128. Then, different models have been implemented, with the focus on the detection of a good configuration, which is capable of fitting the data with sufficient flexibility, and, for the moment, even at the cost of overfitting. For each configuration, the state of the model with best validation (categorical) accuracy was stored.

Although the objective and usefulness of a model only resides in its ability to generalize, that is, in its performance on 'new' data, our primary concern has been to find a model which is capable of representing the complexity of the problem which is to solve. Thus, in this first phase, we have not concerned ourselves with overfitting, but with the identification of a configuration which is not prone to underfitting and has sufficiently large capacity. We start by fitting a relatively simple CNN to our data. It is little deep, as it merely presents two convolutional layers, with 8 and 16 filters, and a fully connected classifier which is ultimately based on just 16 activations. This last configuration choice was adopted to forcefully downsize the parameter space of the model, which would otherwise be very big due to the few pooling layers. In the end, the model still trains over 600 thousand parameters, which are not few, though it is too shallow to build complex feature maps. The model was trained for 20 epochs.

Even though the model was not very complex nor very deep, it was already capable of overfitting, which even happens fairly early, around the 6th epoch (1). It doubles the baseline accuracy, reaching 11% of accuracy on validation data. However, even if prone to overfitting, the model is actually too simple in its specification: most of the parameters are in the fully-connected classifier, where the network overfits by memorizing non-general patterns in the training data, while the convolutional base of the network would probably benefit from being deeper, and so able to detect high-level visual features.

Thus, the model is clearly sub-optimal, and serves as a starting point to improve on. We have extended it, by increasing the number and the size of its layers, thereby producing deeper, more hierarchical, more complex models. Two different configurations have been presented. In a first trial, the model configuration presents 4 convolutional layers, and the number of filters ranges from 16 to 128. The model accomodates almost 5 million parameters. Training lasts for at maximum 30 epochs, but for computational reasons it is stopped once there is enough evidence of overfitting. Compared to the previous model, this configuration overfits much faster (2). This is a consequence of the greater complexity. However, the same higher complexity leads to a large increase in accuracy, which is above 17% for the best parameters state. Then, we implement a second model, which is even deeper than the previous one. It has almost 6 million parameters, and a much deeper convolutional base. Also for this configuration we observe fast overfitting (3), but here the performance is damaged by the greater flexibility (it is maximized at 13%). As a consequence, we choose the first one of these two bigger configurations, and use it as a base for further improvements.

4 Modeling: Regularization

Having obtained a model which is adequate for the complexity of the problem and is capable of overfitting, then we have proceeded to regularize it to limit training data memorization and improve generalization. Before doing so, we have decided to increase the batch size, from 128 to 256, as this will lead to more informative, less noisy gradients, thereby additionally improving the fit to the training data before restraining it.

To regularize our model, we have made use of two tools: data augmentation and dropout. Through data augmentation, we feed our CNN with additional images derived from transforming the original instances thorough a series of operations, such as rotations, shifts, and flips. Through dropout, we switch off a random set of activations during training, thereby limiting the impact of noise on learned patterns. Then, we have trained the network which proved to be the best in the previous section, now regularized. Dropout is applied after each densely connected layer, and the proportion of dropped-out activations is set to 0.5. The model is trained for 50 epochs.

By inspecting loss and accuracy plots (4), we observe that this regularization prevents the model from overfitting, thereby making training and validation metrics move together, in a very different fashion from the behaviour of the unregularized model. This produces a considerable increase in performance, which stabilizes between the 30th and 40th epochs, with a peak validation accuracy of around 24%.

5 Modeling: Transfer Learning

After having obtained a model that, trained from scratch on 60% of the full data, has a classification accuracy on hold-out data of around 1/4, we have tried to go beyond this performance by leveraging a pretrained model, specifically VGG16, trained on ImageNet. We have considered and implemented 3 different approaches to transfer learning: feature extraction without data augmentation, feature extraction with data augmentation, and fine-tuning of the pretrained model.

The first of these approaches to exploit VGG16 for our classification task involves passing each image to VGG16 convolutional base, obtaining feature maps. Thus, we make use of VGG16 learned features to transform our data to a more easily usable representation: presence maps of VGG16's 'visual concepts' learned on ImageNet. This presence maps are then fed to the densely-connected classifier. This type of transfer learning has the advantage of being computationally very fast: once the features have been extracted, we just need to train fully connected layers, which are trained faster than convolutional layers. Through this approach, we have reached a validation accuracy of around 33% (5). However, it seems that, despite dropout, the model is prone to overfitting, which is particularly clear when looking at the dynamics of validation loss; indeed, not only the validation metrics stop improving in a few epochs during training, but the validation loss increases, and does so while validation accuracy is more or less constant. Despite the better accuracy, validation loss rapidly overtakes that of the regularized model (trained from scratch). Thus it seems that overfitting may be a serious problem in this model.

One way to approach the overfitting problem of this model is through data augmentation.

Clearly, this prevents the computational speed-up of the previous method, which passed each image just once through the convolutional base of VGG16. Instead, in this second approach we have extended the convolutional base of VGG16 by adding the dense layers on top, and trained the model with the base frozen. With this model, we have obtained a validation accuracy of 26% (6), but with a considerably smaller validation loss, both with respect to feature extraction without data augmentation and to the regularized model (trained from scratch).

However, the fact that the previous approaches could not lead to significant improvements in both validation loss and accuracy with respect to the regularized model trained from scratch, but at most in just one of the metrics, and that controlling overfitting did not completely improved on feature extraction, suggests that the high-level features which are learned by VGG16 may not be so adequate for our task. Whereas very hierarchical features could be little relevant to our specific classes, more general, basic concepts, that is, the features learned in the first convolutional layers, may be more applicable. Thus, a way to improve on the previous results would be to train not only the fully-connected classifier, but also the top convolutional layers, in order to adapt the most specific features to our problem. Following this third approach, we have then trained the last four layers of the convolutional base of VGG16 and the dense layers at the top. The dense layers have not been trained from scratch, otherwise the error signal would have been too large and could have destroyed the learned features that we wanted to fine-tune. Thus, the dense layers used were those trained when doing feature extraction with data augmentation. Again for the reason of limiting the modifications to the high-level features, we fine-tuned the model using the RMSprop optimizer with a very low learning rate. We have fine-tuned for 50 epochs, and the improvements are evident with respect to the previous methods (7). We obtained the best performance so far in both monitored metrics, and, in particular, we reached a validation accuracy of 37%. Moreover, the model had not converged yet, thus we could have easily achieved further improvements in performance, which however were not pursued for computational reasons.

6 Evaluation

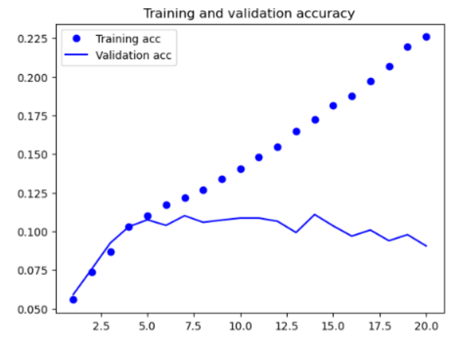
The report is concluded by evaluating and comparing the produced models on the test set, in particular: the best unregularized one, the best regularized one, and the three models based on VGG16. For computational constraints, we have not tried to fit again the final configurations using both training and validation data, which would have improved performance. Thus, the following results are based on 60% of the available data for the classes, by which it is clear that the same models, if they were deployed in a novel context after training on validation and test data too, could easily perform better.

In Fig 8, each model is mapped as its pair of accuracy and loss evaluated on the test data set. We observe that the fine-tuned model has the best performance both in terms of accuracy, which only considers matches of the predictions with the true labels, and loss, which is more general, and it is also influenced by the noise in the model's prediction. For 4 of the models, the two plotted metrics exhibit a tight relationship. Instead, the model of feature extraction without data augmentation behaves like an outlier, as it couples second-best accuracy with worst loss. As for the configurations trained from scratch, they are the most underperforming ones in terms of top-1 accuracy, but regularization clearly improves on the first model.

In Fig 9, each model is mapped as its pair of top5 accuracy and AUC score, averaged across classes. Again, there is a tight relationship between the more general metric, the AUC score, and the most task-specific one, top-5 accuracy. Also in this case, the fine-tuned model strictly dominates all other configurations. The fact that, similarly to what we have seen before, 'unregularized' feature extraction exhibits a good accuracy performance, together with an AUC score which does not fit into the general relationship we observe, makes this model somewhat less reliable than its data-augmented counterpart, which indeed exhibits higher AUC. This coexistence of relatively high accuracy with underperforming general metrics is likely due to overfitting: less regularization made the model more receptive to generalizable patterns, but also introduced considerable noise in the class predictions.

In Fig 10, the average Precision-Recall curves are plotted. Again we observe the difference in behaviour of the unregularized F-EX model, whose P-R curve exhibits a somewhat different shape from the other networks. The plot shows that, considering the model performance on binary classification for each of its classes, if we want a very precise model, fine-tuned VGG16 can offer greater recall, and is therefore superior, whereas, if we are more interested in recall, F-EX without DA can offer greater precision, hence it is preferable. All others are dominated by the fine-tuned VGG16. We end by extracting a random sample of 10 classes, and plotting the PR curves for each of them, in Fig 11. Consistently with previous results, by inspecting the Precision-Recall curves for this sample, we see that the best model, on this metric, is in some cases the fine-tuned pretrained model, and, in other cases, feature extraction without data augmentation.

A Figures



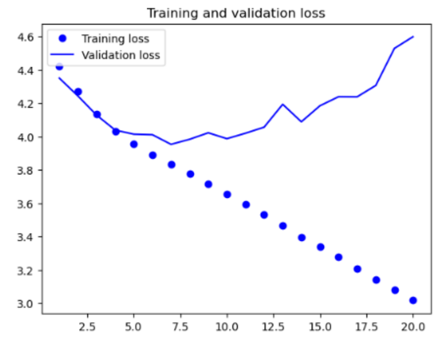
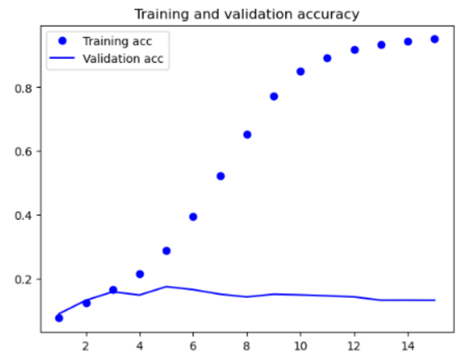


Figure 1: Shallow Model



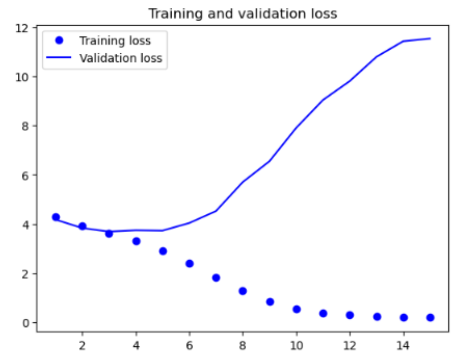


Figure 2: Deep Model 1

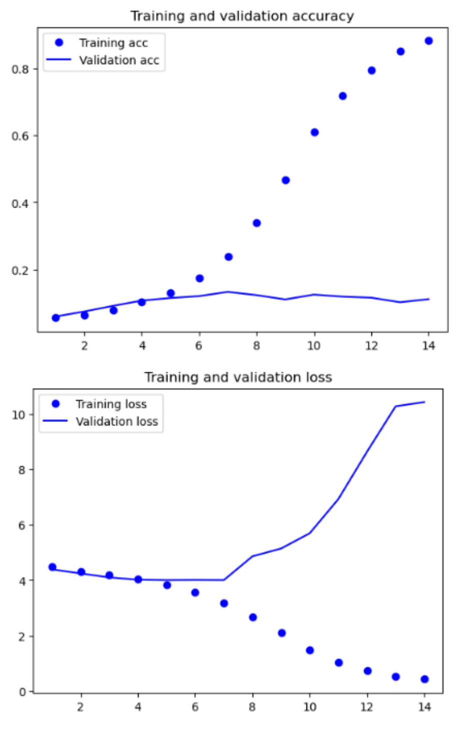


Figure 3: Deep Model 2

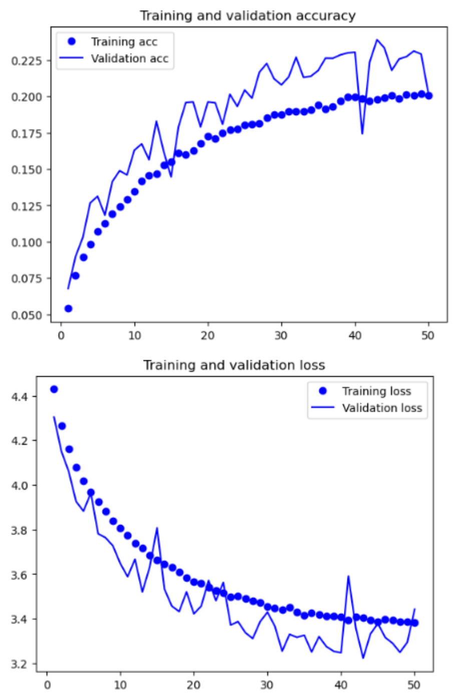


Figure 4: Regularized Model

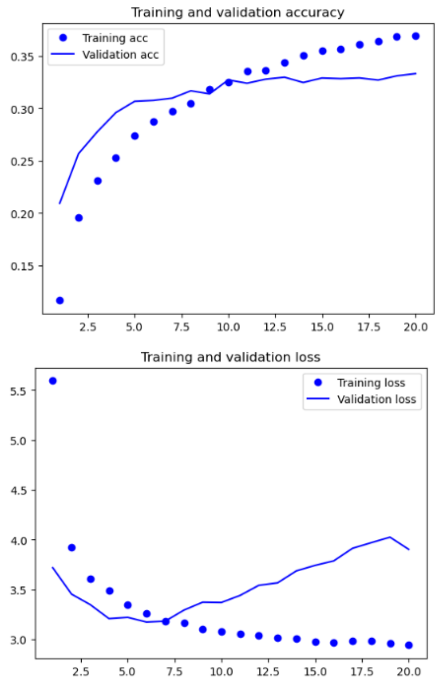


Figure 5: Feature Extraction Model without DA

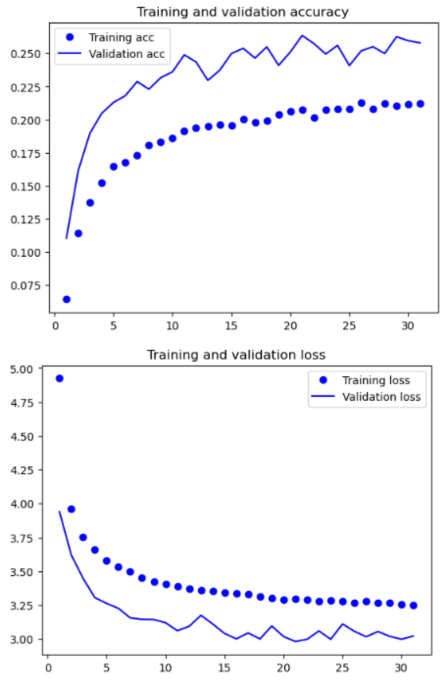


Figure 6: Feature Extraction Model with DA

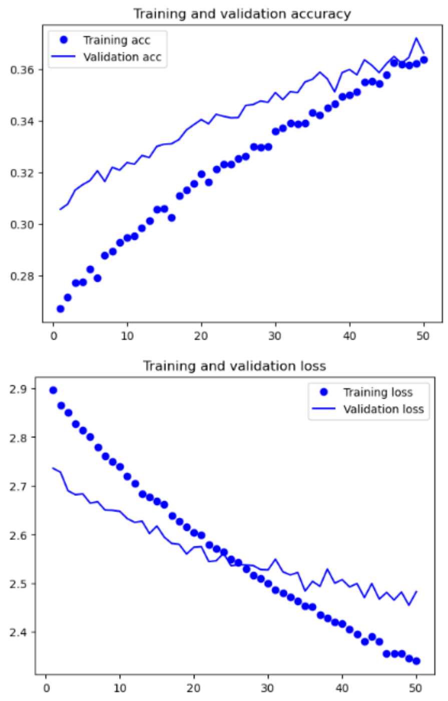


Figure 7: Fine-tuned VGG16

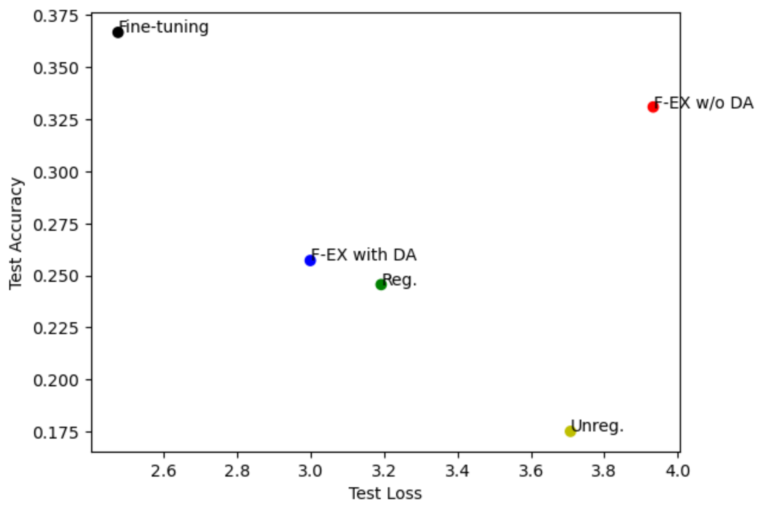


Figure 8: Loss / Top-1 Accuracy

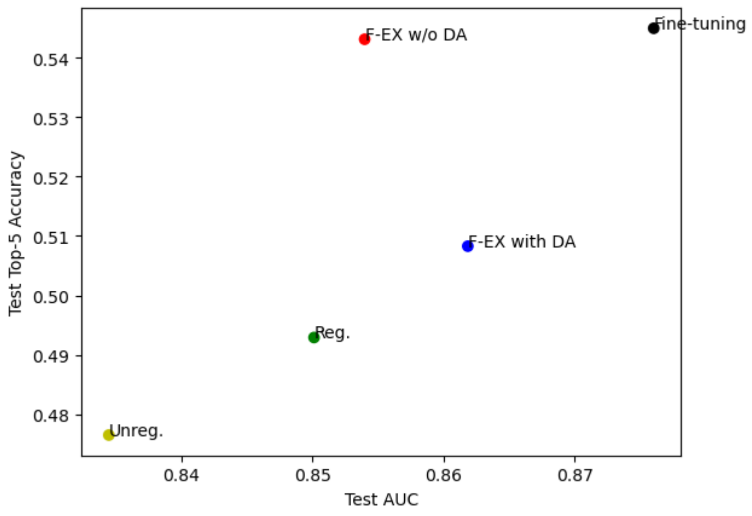


Figure 9: AUC Score / Top-5 Accuracy

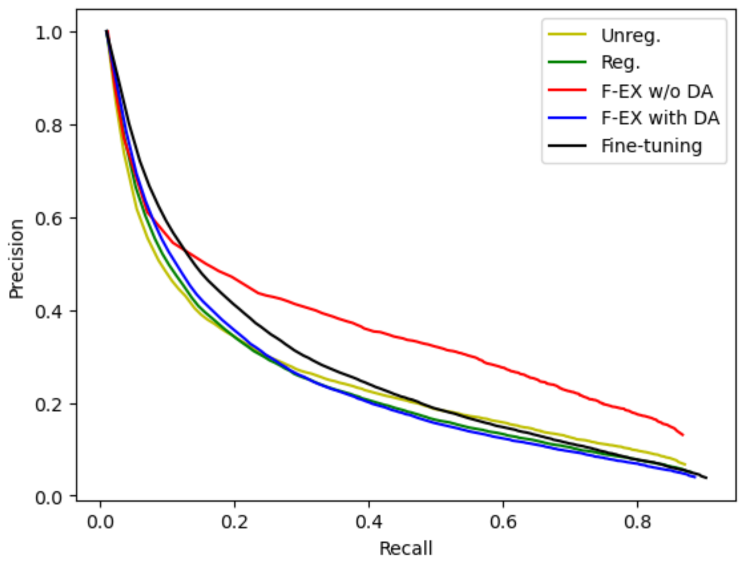


Figure 10: Average Precision-Recall Curves

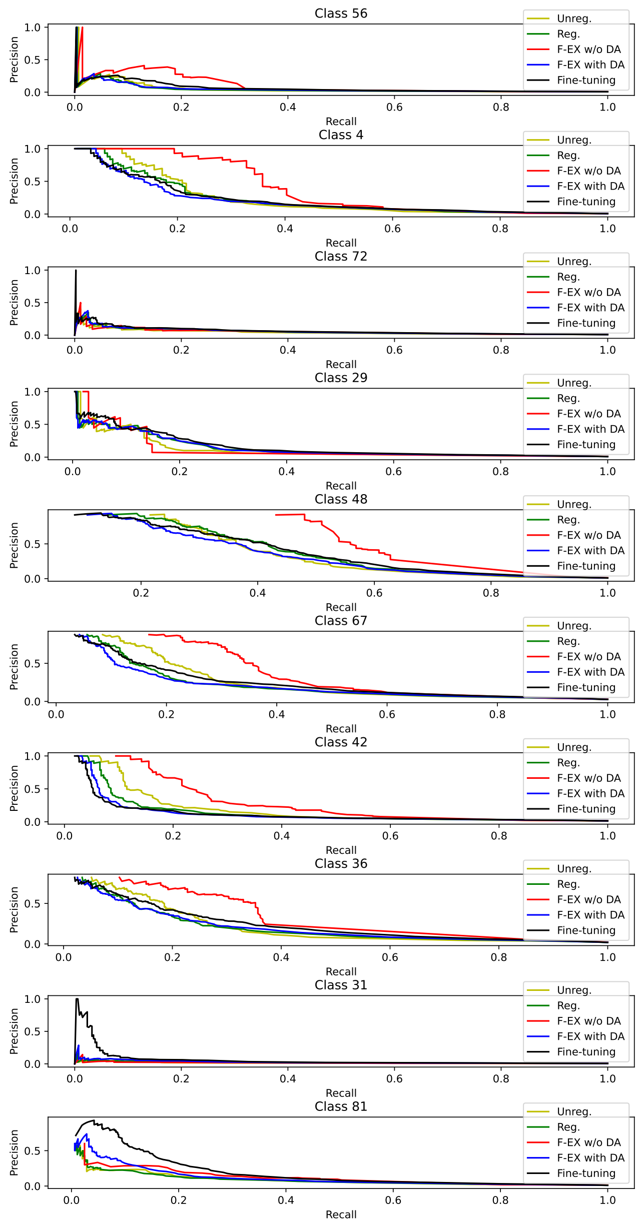


Figure 11: Precision-Recall Curves for a Sample of Classes