Practices for Best Performance of Torchvision Models for Species Classification

Zachary Pulliam
University of Kentucky
zapu222@uky.edu

Abstract

The implementation of neural networks on various tasks such as species classification via images has become exceedingly simpler as libraries, such as PyTorch, have made standard and proven architectures readily available. PyTorch's computer vision subcategory, Torchvision, offers many state-of-the-art architectures and even offers models which have been pretrained in the case of insufficient training data. Torchvision models can be loaded in just a single line of code and little to no actual knowledge of the architecture, neural networks, or image classification is needed to begin training and evaluating the model. However, this, by no means, guarantees maximizing results on the given task. To get the best performance out of Torchvision's vast model selection, there are several practices or techniques that should be followed to maximize the performance of your model. The goal of this work is to explore and evaluate several different practices which attempt to enhance the species and image classification performance of the many models offered in the Torchvision Python package which is available via the forementioned PyTorch library. Throughout this study, models will all be trained on the same BIRDS 400 species classification dataset, evaluated, and compared on the included test set that comes along with this dataset. Classifying different species correctly in an automated sense via neural networks as done here can be extremely useful in other fields such as environmental sciences. Being able to do so could help track species migration and populations. While this work mainly focuses on implementation, there are many avenues to explore with the practical uses as well. Code for this work is provided here*.

1. Introduction

Within the last decade, Python libraries such as PyTorch [1], an open-source machine learning framework that accelerates the path from research prototyping to production deployment, have made the implementation of neural networks for various tasks much simpler. Specifically, the Torchvision package, which will be explored in this work, consists of popular datasets, model architectures, and common image transformations for computer vision tasks. Although the implementation of Torchvision's built in models can be simple for even a Python beginner, there are multiple variables dependent upon the dataset itself, the hardware and resources at hand, and the desired metrics, which will influence what architecture and hyperparameters should be selected before blindly throwing a model at a given task.

1.1 Image Augmentation

Before selecting architectures or hyperparameters, one such practice that has become standard to almost all computer vision tasks, especially image classification, is that of image augmentation [2]. Neural networks are data hungry and require substantial amounts of data to properly learn and achieve good performance on a certain task. Using an image augmentation pipeline, new images, which are unseen to the model, can be created from existing images in the dataset. Depending upon the dataset, certain image augmentations are selected to add to the augmentation pipeline and are assigned probabilities as to whether they will be applied to an image before it is passed through the model. Using this probabilistic assignment of augmentations, the hope is that at each training step, a unique set of augmentations will be applied to an individual image, creating a new image each time and creating a more robust model which has been trained on more data.

1.2 Model Selection

To date, the Torchvision package has a total of 17 model architectures and 44 individual models for image classification. The model architectures provided are as follows: AlexNet [3], VGG [4], ResNet [5], SqueezeNet [6], DenseNet [7], Inception v3 [8], GoogLeNet [9], ShuffleNet v2 [10], MobileNet v2 [11], MobileNet v3 [12], ResNeXt [13], Wide ResNet [14], MNASNet [15], EfficientNet [16], RegNet [17], VisionTransformer [18], and ConvNeXt [19]. With so many models to choose from, it may be difficult to select which model is best for the task at hand. Therefore, it is important to understand different properties of models which may be available such as model depth, width, trainable parameters, etc. Each of these

**github.com/zapu222/ImageClassificationBirds400*

factors will play a significant role in the resources and time needed to train the model as well the metrics the model will produce. Given the scope of this work and the time/resources it takes to train a neural network, not all Torchvision models will be used and compared here. One model from each architecture with less than 25 million parameters (and AlexNet) will be trained and tested during the model selection phase. This leaves a total of 11 individual models: AlexNet, ResNet18, SqueezeNet 1.1, DenseNet121, GoogLeNet, ShuffleNet v2, MobileNet v2, MobileNet v3 small, MNASNet 1.0, EfficientNet B0, and RegNet y 400mf. Each of these will be discussed further.

Figure 1

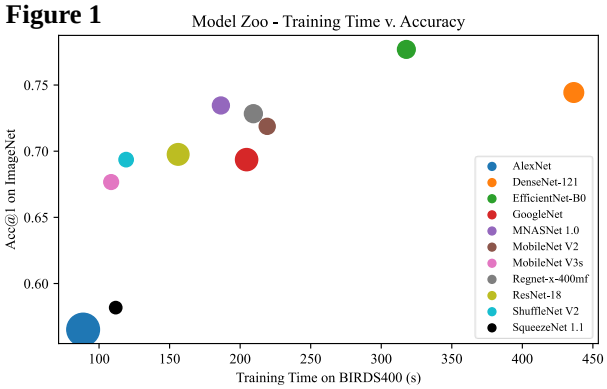


Figure 2

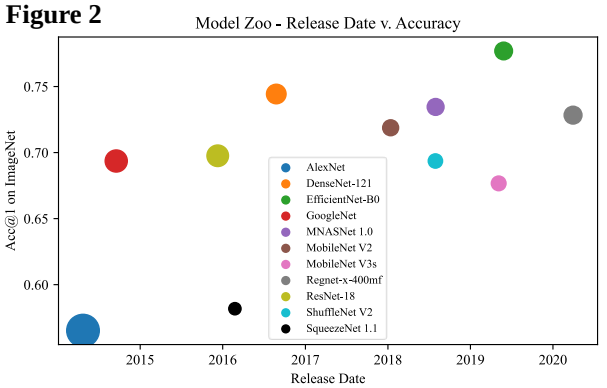


Table 1

Model	Trainable Parameters		MobileNet V2	3504872
AlexNet	61100840		MobileNet V3s	2542856
DenseNet-121	7978856		Regnet-x-400mf	5495976
EfficientNet-B0	5288548		ResNet-18	11689512
GoogleNet	13004888		ShuffleNet V2	2278604
MNASNet 1.0	4383312		SqueezeNet 1.1	1235496

Figure 1&2 and Table 1: Comparison of models used throughout this study. Trainable parameters are represented by point size on both scatterplots. Figure 1 displays a comparison of average training time to accuracy at top 1 on ImageNet. Figure 2 displays a comparison of release date to accuracy at top 1 on ImageNet. Table 1 simply lists the number of trainable parameters of each model as preloaded directly from Torchvision.

A. AlexNet

AlexNet is a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky and released in 2014. AlexNet first competed in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [20] on September 30, 2012. This challenge requires the classification of a testing subset of ImageNet with over one million images and a thousand total classes. AlexNet achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the second-place finisher. AlexNet's architecture has a total of 8 layers, 5 of which are CNN layers, and the last 3 max-pooling layers. By incorporating things such as the Rectified Linear Unit (ReLU) [21] activation function in its CNN layers, AlexNet was able to stand out compared to others using hyperbolic tangent at the time, which was standard. By using ReLU activations, AlexNet was able to offer much better performance and lower training times.

B. ResNet

Residual Neural Network (ResNet) was first introduced in 2015 by Kaiming He et al. Before ResNet, model depth had to be kept somewhat limited due to the vanishing gradient problem. Vanishing gradients occur at early layers in the network when backpropagating gradients throughout the network. Gradients are calculated first at the output and then calculated at each previous layer based on the weight update at the current layer. This leads to smaller and smaller gradients at each previous layer and leads to little to no weight update at beginning layers. ResNet was able to offer a solution to this issue by introducing a new kind of neural network layer called "residual block." It is a gateless or open-gated variant of the HighwayNet [22] in which skip connections or shortcuts are used to jump over some layers. The skip connections consist of ReLU activation function and batch normalization between. The idea behind these skip connections is to avoid vanishing gradients by utilizing gradients from layers two or three layers previous.

C. SqueezeNet

SqueezeNet was released in 2016 and was developed by researchers at DeepScale, the University of California, Berkeley, and Stanford University. The authors' goal when creating SqueezeNet was to create a smaller neural network with fewer parameters that can more easily fit into computer memory and can more easily be transmitted over a computer network

D. DenseNet

DenseNet was first released in 2017 by Gao Huang et al. It shows similarities to ResNet in that it uses shortcut connections between layers in order to address the vanishing gradient issue. DenseNet utilizes dense

connections between layers, through "dense blocks," where all layers are connected with matching feature-map sizes. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. This structure allows for use of fewer layers which subsequently is faster and requires less resources to train.

E. GoogLeNet

GoogLeNet, also known as Inception v1, was the first member of the Inception family, developed by Christian Szegedy et al. and Google. This specific architecture was released in 2015 and prior to that, GoogLeNet participated in the ILSVRC challenge and won first place and achieved a new state-of-the-art score for ImageNet in 2015. The main contribution of GoogLeNet is its solution to overfitting and high computation power for extremely deep networks. The convolutional layers consist of filters of sizes 1x1, 3x3, and 5x5 all at the same layer. This allows the model to extract both big and small features at the same layer, creating a wider network rather than a deep one. The feature map from all filters is concatenated and passed to the next layer.

F. ShuffleNet v2

ShuffleNet v2, was introduced by Ningning Ma et al. in 2018. The goal of ShuffleNet was to create a more efficient and effective network which was able to perform well while using less computational power.

G. MobileNet

Specifically, MobileNet v2 was first introduced in 2019, based upon the first MobileNet v1, released in 2017. The MobileNet family of architectures is formed by two main layers, the "depth-wise convolution" and the "point-wise convolution." The "depth-wise convolution" layer applies a single convolutional filter per input channel, while the "point-wise convolution" applies a 1x1 convolutional filter meant to combine input channels for feature extraction.

H. MNASNet

Automated mobile neural architecture search (MNASNet) was created with the idea of introducing a network which was able to perform well on mobile devices and was released in 2018. Given that mobile devices have significantly less computational power than other machines, the model had to be small in size, yet still perform well. To do so, the authors created what they called automated mobile neural architecture search (MNAS). MNASNet explicitly incorporates model latency as the main objective so that the search can identify a model that achieves a good trade-off between accuracy and latency. To further strike the right balance between flexibility and search space size, the authors proposed a novel factorized hierarchical search space that encouraged layer diversity throughout the network.

I. EfficientNet

Released in 2020 by Mingxing Tan and Quoc V. Le, EfficientNet proposed a new scaling method that uniformly scaled all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient. The results and effectiveness of this efficient scaling was demonstrated via scaling up on MobileNets and ResNet. Further, using this scaling technique, the authors created a new family of CNN architectures which they named EfficientNets.

J. RegNet

First introduced in 2021, RegNet attempts to make improvements to the forementioned ResNet by addressing the issue of the shortcut connection mechanism limiting the ability of reexploring new potentially complementary features due to the additive function by introducing a regulator module as a memory mechanism to extract complementary features. The regulator module is composed of convolutional recurrent neural networks (RNNs) [23] which are shown to be good at extracting spatio-temporal information.

1.3 Transfer Learning

Through the use of transfer learning, knowledge acquired by a neural network from a previous task can be stored, applied, and exploited to a novel task in order to boost model performance on the novel task [24]. This is particularly useful when attempting to train a neural network using insufficient amounts of data. With little data, a model may not be able to generalize well to a test or validation set as it has simply not seen enough examples to properly train its parameters. Therefore, in the case of insufficient data, a model can be trained on a larger, more general dataset so that the model achieves high performance on the larger dataset in hopes that this "knowledge" can be applied to the smaller dataset. After the model is pretrained on the larger dataset, model weights are saved, and the final layers of the network can be finetuned on the smaller dataset of interest. In the case of CNNs, this typically takes the form of pretraining the initial convolutional layers for adequate feature extraction, then finetuning the classification layer(s) at the output. As mentioned in section 1.2, Torchvision offers pretrained and randomly initialized variants of their architectures. The pretrained models are trained on the ImageNet image classification dataset. ImageNet contains over 14 million images to date with over 1000 categories. Each pretrained Torchvision model's metrics on the ImageNet dataset can be found here.

1.4 Learning Rate

In a broader sense than just in the species classification scenario, learning rate is a parameter of the optimization algorithm which determines how large or small step size is while attempting to minimize a loss (error) value. Selecting an appropriate learning rate can result in a much more generalized model which is better able to converge and can significantly reduce training time. In the case of image classification, the goal of the loss function is to first determine the error in the model at the output layer. The standard loss function for image classification is a cross entropy loss which calculates the difference between two probability distributions. The optimization algorithm then determines how weights and biases at the output layer should be adjusted so that the model will perform better during the next training epoch and the difference between probability distributions can be reduced. These adjustments, called deltas, are then backpropagated to the earlier layers in the network so that they can be updated as well. The size of the weight adjustments are determined by the learning rate. Having a learning rate which is too small will result in slow convergence, and the model will take a long time to perform well on the assigned task. However, having a learning rate which is too large will lead to divergence, and the error will grow exponentially or will not converge well. Therefore, selecting an appropriate learning rate is a very important task when preparing to train a model.

1.5 Ensemble Methods

While ensemble methods are not necessarily required to achieve good performance on a task, it is a method that may help boost metrics slightly during testing [25]. Ensemble learning aims to produce a strong model by harnessing the combined and complementary wisdom of multiple base models. Therefore, an ensemble is a collection of models whose predictions are combined by weighted averaging or voting. Therefore, an ensemble model consists of two or more models trained on the same task with the end goal being that each trained model will have different weights at the conclusion of training. During testing, each output other ensemble is considered; this allows the models to correct one another at the output layer in the case that an example is misclassified during testing by an individual model. In the image classification case, model outputs can be combined one of two ways; simply ranking model outputs and choosing the class with the highest rank across all models or taking into account model probabilities and selecting the class which has the highest probability across all models. The obvious downside to this method is that multiple models must be trained which consumes both time and resources. However, it is case dependent on if the time and resources spent to do so is worth the extra effort.

2. Dataset

The dataset used throughout this work is the BIRDS400 – Species Image Classification Dataset [26]. The dataset consists of 400 bird species with 58388 training images, 2000 test images (5 images per species) and 2000 validation images (5 images per species). The images within the dataset have been very well curated with only one bird in each image and with the bird typically taking up at least 50% of the pixels in the image. All images are 224 by 224 by 3 color images in jpg format. Images were gathered from internet searches by species name and checked for duplicates. One downside to this species dataset is that about 85% of the images are of the male and only 15% of the female. Males typically are far more diversely colored while the females of a species are typically bland. Consequently, males and females of an individual species may look entirely different. Therefore, if more female images were introduced in the test set, the model may perform poorly. Datasets such as these can be extremely useful in training models which can then be used in automated species classification and detection which has multiple use cases in the environmental science field such as tracking migration patterns or populations.

Figure 3: Example batch of 48 images from the BIRDS400 dataset. All images have been pulled randomly from the training set.

3. Methodology

Throughout this work, all models have been trained on the same NVIDIA 3060 GPU for 50 epochs using a batch size of 64 images. Images were kept at their original size of 224 by 224 by 3 color channels throughout all experiments. Albumentations [27] image augmentations were used exclusively to augment the training set of images. In order to correctly train and test each model on the BIRDS400 dataset, the output layer for each Torchvision model was altered to have 400 outputs from 1000 original outputs for ImageNet. Torchvision's built in Cross Entropy Loss function was used to determine loss for each model and Torchvision's Stochastic Gradient Descent optimizer was used to update each model after each epoch. The "best" weights during training were determined by the model's performance on the validation set and were then saved along with the most recent weights after each training epoch. The "best" weights were used to determine the model's performance on the test set.

During the model selection phase, randomly initialized models from each of the eleven models to be used were trained using the criteria above. Learning rate was set at 0.05. Image augmentations were kept the same for each model. The augmentations included during training are displayed in the code snippet 1. Within each function is a flag *p* which corresponds to the probability that the augmentation is applied. Model accuracy is then measured using the 2000 image test set that is included with the BIRDS400 dataset.

```
A.HorizontalFlip(p=0.5),

A.ShiftScaleRotate(shift_limit=0.1,
scale_limit=0.1, rotate_limit=15, p=0.1),

A.RandomBrightnessContrast(brightness_limit=0.1,
contrast_limit=0.1, brightness_by_max=True,
p=0.1),

A.HueSaturationValue(hue_shift_limit=5,
sat_shift_limit=5, val_shift_limit=5, p=0.1)
```

Code Snippet 1: Augmentations used during model selection phase

When comparing image augmentation pipelines, four different pipelines were tested. The first pipeline includes all the augmentations listed in code snippet 1. The second includes only the translational augmentations; the HorizontalFlip and the ShiftScaleRotate augmentation. The third includes only the HorizontalFlip augmentation. The last includes no image augmentations and uses only the original training set of images. For each pipeline, a randomly initialized instance of ResNet18 is used with all hyperparameters kept constant across the four models and training runs. For each of these models, loss and accuracy on both the train and validation set is recorded in order to compare model progression across epochs.

For the transfer learning stage, Torchvision's pretrained version of ResNet18 will be compared with a randomly initialized version of ResNet18. The same is also done with AlexNet. In both comparisons, all hyperparameters are kept the same to ensure a fair comparison between pertained and randomly initialized models.

When comparing how learning rate affects model training, randomly initialized instances of ResNet18 will be used. Four different learning rates are used along with Torchvision's stochastic gradient descent optimizer. The learning rates are 5.0, 0.5, 0.05, and 0.005. For each, loss and accuracy on both the train and validation set is recorded in order to compare model progression across epochs.

Lastly, in order to assess how an ensemble model compares to the individual models which create the ensemble, each individual model will be assessed on the test set and the ensemble model will also be assessed on the test set. The ensemble will consist of three random initialized instances of SqueezeNet 1.1 trained for 50 epochs, using all augmentations shown in code snippet 1, with learning rate of 0.05, image size of 224 by 224 by 3 color channels, and batch size of 64 images.

4. Results

Exploring the results for the model selection phase, most models tend to perform with over 90 percent accuracy on the test set, with 9 of the 11 models doing so. In Table 2, models are ranked from worst to best performance on the BIRDS400 test set. Keep in mind that these are only the Torchvision models with less than 25 million trainable parameters (plus AlexNet with 61 million). GoogLeNet outperforms all others on both train set and test set accuracy with accuracies of 99.6 percent and 98.1 percent, respectively. This is quite surprising given GoogLeNet's performance on ImageNet compared to the other models tested here. On ImageNet, GoogLeNet performs seventh out of the 11 models tested in this work. This could be due to the fact that the BIRDS400 dataset is much smaller than that of the ImageNet dataset and GoogLeNet is able to generalize better to a smaller test set. Similarly, despite the performances on ImageNet, AlexNet outperforms SqueezeNet in this test, quite significantly on the train set, which is likely due to the much larger size of AlexNet compared to SqueezeNet, as AlexNet is likely better able to memorize the smaller BIRDS400 train set compared to the ImageNet dataset. As AlexNet has around 60 million more parameters than the much smaller SqueezeNet, which is designed to be smaller but still have respectable performance. Lastly, the model which significantly outperforms all others on ImageNet, EfficientNet-B0, ranks fourth on the BIRDS400 test set, however, models ranking 2nd through 4th are within 0.4 percent of each other.

Model	Model Parameters	Avg. Train Time (s)	Min. Train Loss	Min. Val. Loss	Train Acc@1	Val Acc@1	Test Acc@1
SqueezeNet 1.1	1235496	111.817	0.485	0.649	0.865	0.844	**0.885**
AlexNet	61100840	88.719	0.146	0.857	0.959	0.819	**0.897**
MNASNet 1.0	4383312	186.356	0.046	0.396	0.988	0.905	**0.931**
RegNet y 400mf	5495976	209.826	0.033	0.388	0.991	0.911	**0.942**
MobileNet v3s	2542856	108.584	0.089	0.455	0.973	0.894	**0.943**
ShuffleNet v2	2278604	119.196	0.100	0.381	0.975	0.901	**0.945**
MobileNet v2	3504872	219.170	0.048	0.300	0.987	0.921	**0.961**
EfficientNet-B0	5288548	317.794	0.038	0.269	0.989	0.934	**0.964**
ResNet18	11689512	156.068	0.033	0.240	0.993	0.938	**0.965**
DenseNet-121	7978856	436.457	0.042	0.277	0.991	0.932	**0.968**
GoogLeNet	13004888	204.570	0.020	0.190	0.996	0.951	**0.981**

Table 2: Model selection results on the BIRDS400 dataset. Maximum Train accuracy over 50 training epochs is displayed along with the test accuracy using the "best" model, as determined by the highest validation accuracy. All model accuracy and loss values over all 50 epochs are visualized in the Appendix section.

Figure 4

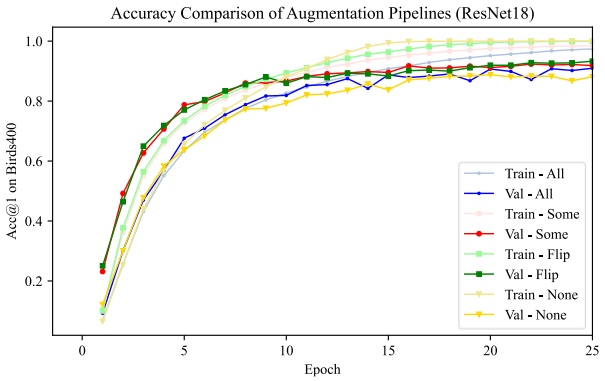


Figure 5

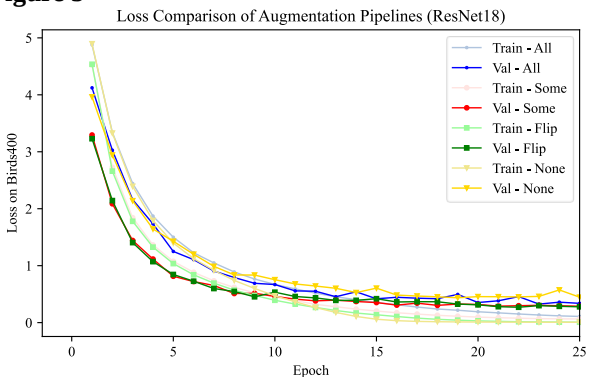


Figure 4&5: Figure 4 displays the accuracy on train and validation sets from epochs 0 to 35 of training and Figure 5 displays the train and validation losses. Corresponding models have the same color with different saturations and have the same point indicators. "All" reflects all augmentations in Code Snippet 1, "Some" just the translational augmentations, "Flip" just the HorizontalFlip augmentation, and "None" no augmentations.

The results of the different data augmentation pipelines can be seen in Figure 4 and 5. Given what is known of data augmentation, it is expected that having more augmentations to further diversify training data will yield better validation results, however, in this case, the "some" and "flip" augmentation pipelines outperform the "all" pipeline, which has augmentations not include in the "some" and "flip" pipelines. This result is likely because the BIRDS400 dataset is extremely curated; adding the two augmentations which shift brightness/contrast and hue/saturation likely creates images during training which are unlike those in the validation set. Therefore, the validation accuracy remains slightly lower. It is extremely important to have a good understanding of how curated or diverse the images within the dataset are so that only augmentations which yield representative images are included in the augmentation pipeline. Another important result to note from this phase of the experiments is that while the "none" augmentation performs best on the training set, it has the worst performance on the validation set, and the same is true of the loss. During training, with no image augmentations, the ResNet18 model is better able to memorize the training set since it is training on the exact same images during each epoch and in turn has better training metrics. However, this yields a less generalized model and in turn, yields poorer performance on the validation set. Another interesting find here is the sporadic performance of the "all" pipeline on the validation set. Approaching epoch, 10 and beyond, the validation performance increases and decreases at rather large intervals while keeping an overall increasing trend rather than remaining at a somewhat constant increase similar to the others. This is also likely caused by the unrepresentative images created by using unnecessary augmentations, as the model learns from images during training which are unrepresentative of the validation set.

Figures 6 and 7 help to visualize the results of the transfer learning phase. As expected, the Torchvision pretrained models perform substantially better than their randomly initialized counterparts. On the BIRDS400 dataset, after just a single training epoch, the pretrained instances of ResNet18 and AlexNet reach 90% and 84% accuracy, respectively on the validation set, whereas the randomly initialized models are at only 9.25% and 0.25% accuracy after the first epoch. Using these pretrained models can save tremendous amounts of time in a scenario such as this, or in cases where training data is lacking, as discussed in section 1.3. Even after 30 training epochs, the ResNet18 pretrained instance performs 7% better than its randomly initialized counterpart, and the same is true for AlexNet with an even larger boost of 15%, in favor of the pretrained model. An interesting phenomenon here is the somewhat oscillatory nature of the pretrained AlexNet model. The performance increases then decreases and seems to level out over time. This could suggest that the learning rate of 0.05 is too high for AlexNet.

Figure 6

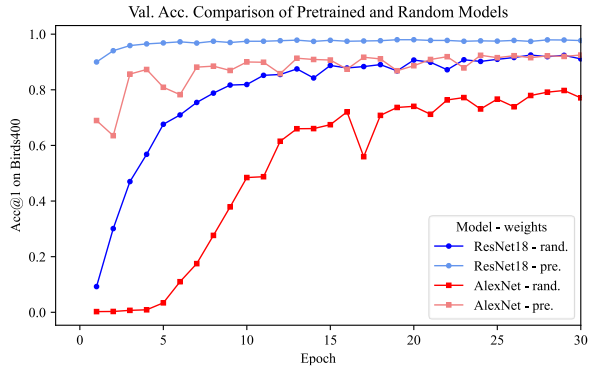


Figure 7

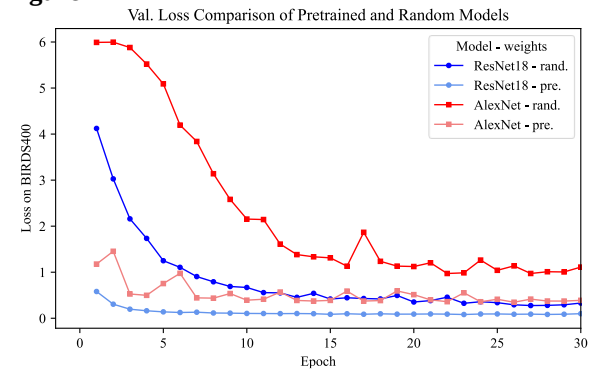


Figure 6&7: Validation accuracy and loss comparison of pretrained models and randomly initialized models up to epoch 30. Here, comparisons are made of ResNet18 and AlexNet. The pretrained instance of each model is displayed in the lighter color, with ResNet18 being represented in blue and AlexNet in red.

The learning rate phase of this study is the last to be explored. Figures 8 and 9 visualize how varying learning rates can affect model performance. Based on both figures, it seems that a learning rate of 0.5 performs best for ResNet18. While this is true of ResNet18, this may not necessarily be the case for all the models tested in the model selection phase of this experiment. While each of these phases can boost performance individually, it is also important to understand how each alteration of hyperparameters or models selection correspond to one another. Generally, a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights, while a smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train. Due to the smaller size of the dataset here, using a learning rate of either 0.5 or 0.05 seem to converge around the same level of accuracy, however, with a larger dataset, the difference in convergence may be more drastic and the learning rate may be more important to tune properly.

Figure 8

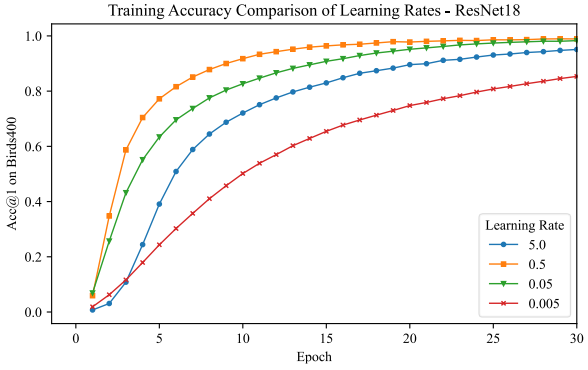


Figure 9

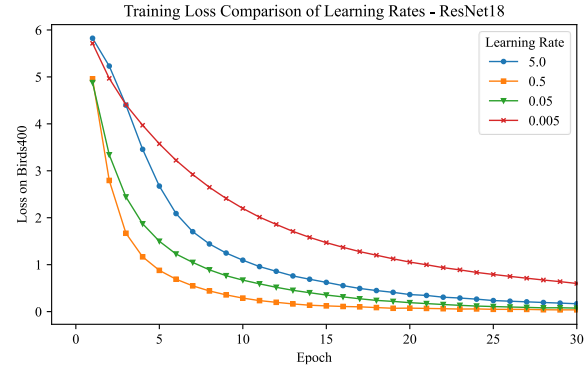


Figure 8&9: Training accuracy and loss comparison of different learning rates on randomly initialized instances of ResNet18 up to 30 epochs. Each model is trained with the exact same hyperparameters, augmentations, and with the same loss function, the only variation being learning rate.

The final phase of this study to be analyzed is the ensemble learning phase. Three instances of randomly initialized SqueezeNet 1.1 models and three instances of randomly initialized ResNet18 models were used during this phase to create two ensemble models. SqueezeNet was selected specifically for this phase primarily due to the fact that this architecture had the worst performance, and therefore was used in hopes that by creating an ensemble of multiple SqueezeNet models the performance on the test set could be boosted to a greater value than the individual models produced. Table 3 and 4 displays the results of each individual SqueezeNet model along with the ensemble model consisting of all three instances and the same for ResNet18. The "best" weights from each individual model instance were used for each as determined by the top 1 accuracy on the validation set. Each of the three individual SqueezeNet models performs around 89% top 1 accuracy on the BIRDS400 test set and around 98% accuracy at top 5. While this does not leave much room for improvement at top 5 accuracy, top 1 accuracy can be improved when considering other individual model's accuracy scores during the model selection phase, with most models performing greater than 90% at top 1 accuracy. The SqueezeNet ensemble model performs, on average, 5% better than the individual SqueezeNet models, which is quite a significant improvement in accuracy. While it may be more convenient in the case of the BIRDS400 dataset to simply use another, more complex Torchvision architecture which performs better, with no significant increase in training time or resources used, these more complex models may fail to perform well on other, larger, or more complex datasets. In cases where even the most complex models fail to produce sufficient results, creating an ensemble model will likely lead to the same result here, and have a significant boost in performance, saving the user from having to attempt to create new methods to produce better metrics, which can be extremely difficult and time consuming.

Table 3

Model	Test Acc@1	Test Acc@5
ResNet18 (1)	0.955	0.994
ResNet18 (2)	0.964	0.998
ResNet18 (3)	0.964	0.997
Ensemble (1, 2, 3)	**0.983**	**0.999**

Table 4

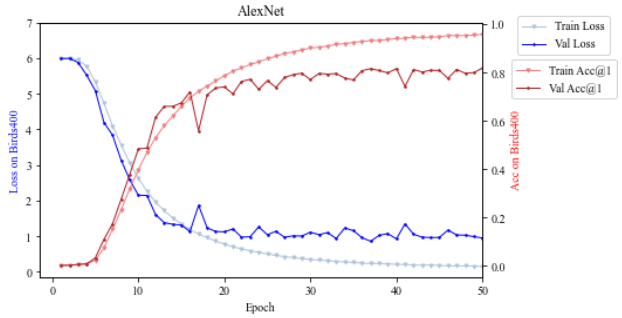
Model	Test Acc@1	Test Acc@5
SqueezeNet 1.1 (1)	0.885	0.971
SqueezeNet 1.1 (2)	0.892	0.980
SqueezeNet 1.1 (3)	0.884	0.980
Ensemble (1, 2, 3)	**0.937**	**0.987**

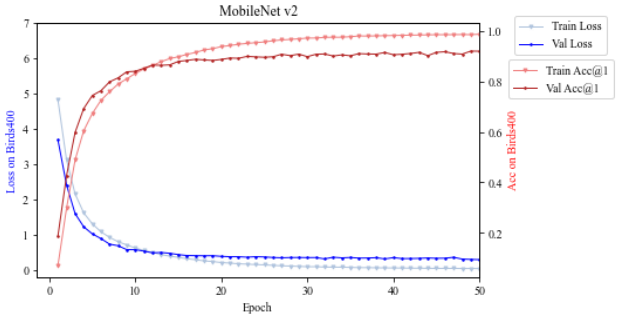
Table 3&4: Test accuracy at top 1/top 5 for three uniquely trained SqueezeNet 1.1 and ResNet18 models and their corresponding ensemble models.

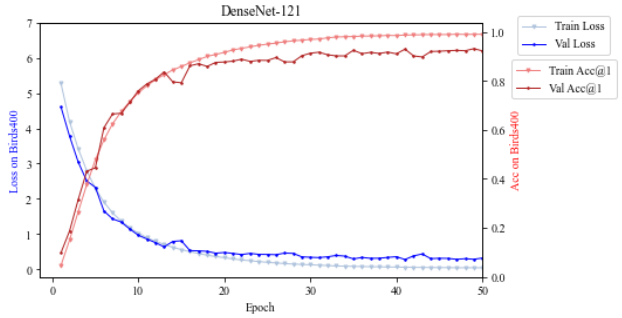
5. Conclusion

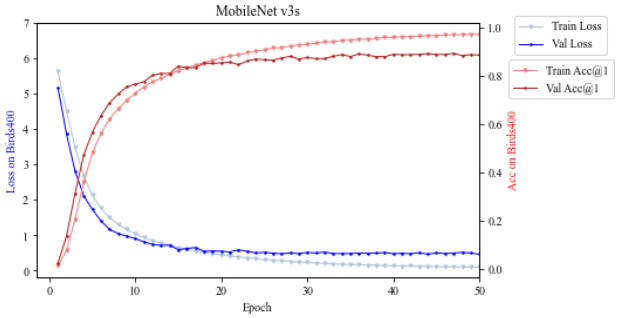
While libraries, such as PyTorch and its sub package Torchvision, have made standard computer vision architectures readily available. Computer vision beginners may be far from producing up to par metrics with out of the box methods on the species datasets which they may have at hand. Simply loading in a Torchvision model and beginning training and evaluating the model may not be enough. To get the best performance out of Torchvision's vast model selection, practices such as model selection, image augmentation, transfer learning, learning rate selection, and ensemble methods, when done correctly can go a long way in improving species classification metrics. These practices must each be considered individually based on several factors such as dataset, model, time, and resources, and each problem should be considered differently when selecting how to approach each practice. When done correctly, metrics can be boosted significantly, however, if not properly approached and considered, these practices can also harm metrics, therefore each should be approached carefully. Utilizing these techniques can therefore greatly improve practical implementations or applications where neural networks may be used to identify different species in an automated sense.

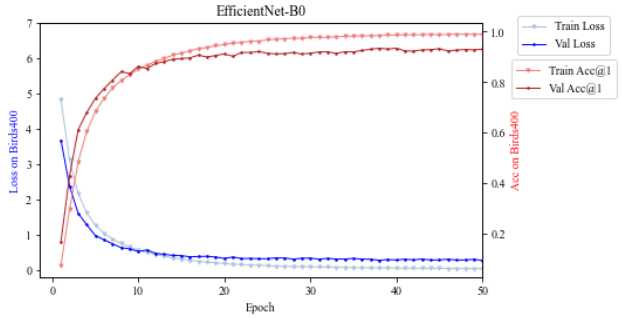
Appendix

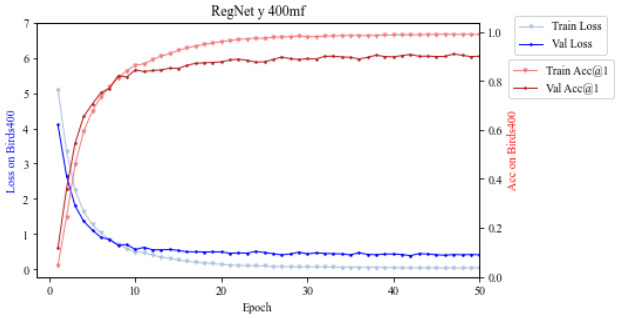

AlexNet
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch

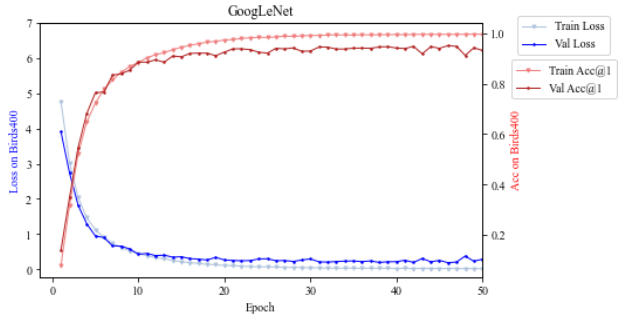

MobileNet v2
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch

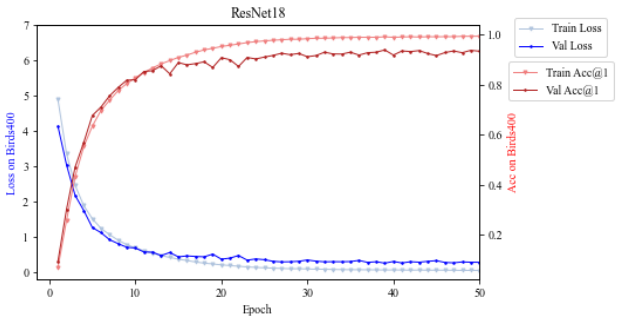

DenseNet-121
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch

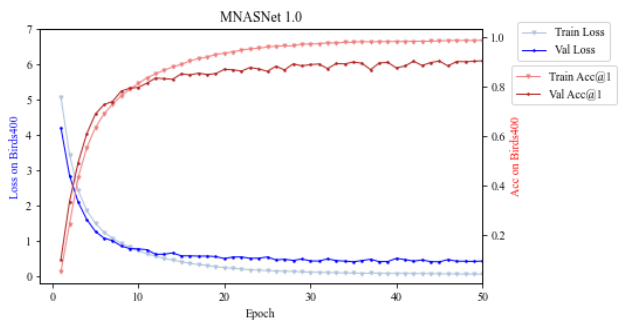

MobileNet v3s
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch

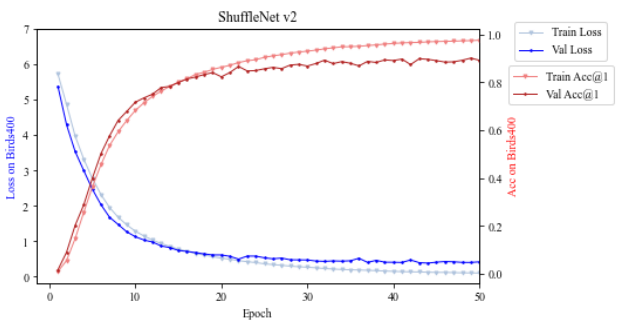

EfficientNet-B0
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch

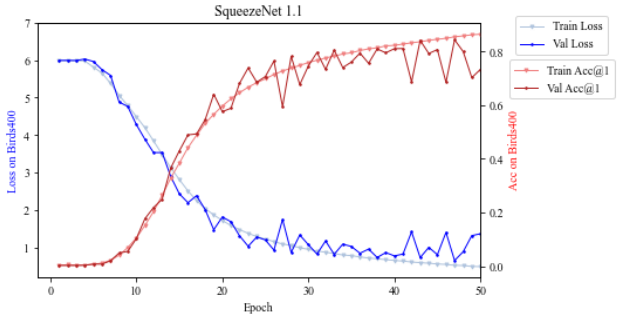

RegNet y 400mf
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch


GoogLeNet
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch


ResNet18
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch


MNASNet 1.0
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch


ShuffleNet v2
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch


SqueezeNet 1.1
Train Loss
Val Loss
Train Acc@1
Val Acc@1
Loss on Birds400
Acc on Birds400
Epoch
0
10
20
30
40
50
1
2
3
4
5
6
7
0.0
0.2
0.4
0.6
0.8

References

[1] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[2] Shorten, Connor, and Taghi M. Khoshgoftaar. "A survey on image data augmentation for deep learning." *Journal of big data* 6.1 (2019): 1-48.

[3] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," arXiv preprint arXiv:1404.5997, 2014.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[6] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," arXiv preprint arXiv:1602.07360, 2016.

[7] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[10] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 116–131.

[11] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.

[12] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. Le, H. Adam, "Searching for MobileNetV3," arXiv preprint arXiv:1905.02244, 2019.

[13] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1492– 1500.

[14] S. Zagoruyko and N. Komodakis, "Wide residual networks," arXiv preprint arXiv:1605.07146, 2016.

[15] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2820–2828.

[16] M. Tan, Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," arXiv preprint arXiv:1905.11946, 2020.

[17] I. Radosavovic, R. Kosaraju, R. Girshick, K. He, P. Dollar, "Designing Network Design Spaces," arXiv preprint arXiv:2003.13678, 2020.

[18] Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

[19] Liu, Zhuang, et al. "A ConvNet for the 2020s." arXiv preprint arXiv:2201.03545 (2022).

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al., "Imagenet large scale visual recognition challenge," International journal of computer vision, vol. 115, no. 3, pp. 211–252, 2015.

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in ICML, 2010.

[22] Srivastava, Rupesh Kumar; Greff, Klaus; Schmidhuber, Jürgen "Highway Networks". arXiv preprint arXiv:1505.00387, 2015

[23] Zaremba, Wojciech, Ilya Sutskever, and Oriol Vinyals. "Recurrent neural network regularization." arXiv preprint arXiv:1409.2329 2014.

[24] J. West, D. Ventura, s. Warnick, "Spring Research Presentation: A Theoretical Foundation for Inductive Transfer". Brigham Young University, College of Physical and Mathematical Sciences. 2007

[25] Caruana, Rich, et al. "Ensemble selection from libraries of models." Proceedings of the twenty-first international conference on Machine learning. 2004.

[26] Gerry, “BIRDS 400 - SPECIES IMAGE CLASSIFICATION,” Kaggle, 2022. [Online]. Available: https://www.kaggle.com/datasets/gpiosenka/100-bird-species

[27] Albumentations “Python library for fast and flexible image augmentations” [Online] https://albumentations.ai/

[28] F. Albardi, H. Dipu Kabir, M. Islam Bhuiyan, P. Kebria, A. Khosravi, S. Nahavandi, “A Comphrehansive Study on Torchvision Pre-trained Models for Fine-grained Inter-species Classification,” arXiv preprint arXiv:2110.07097, 2021