Abstract

I present a simple, highly modularized network architecture for the image classification of bird datasets. This network architecture consists of repeating a block that aggregates a set of transformations with the same topology, which results in a multi-branch architecture that contains only a few hyperparameters to set. The size of the set of transformations, or cardinality, is the factor that greatly improves classification accuracy, in addition to width and depth. This model architecture is named ResNeXt [1], and with this model accuracy on small, fine grained classification problems is greatly improved compared to other architectures.

1 Introduction

In the past, convolutional neural networks have been the most frequent and standard model used for image recognition and object detection, and with rapid improvement and research the quality and effectiveness of these models have progressed tremendously. Along with more powerful hardware, improved models, and larger datasets the greatest improvements have been through new algorithms and network architectures. In relation to visual recognition and object detection, neural networks have been more traditionally designed to learn features from large-scale data and be able to perform a variety of recognition tasks, but improvements in network architecture can serve to enhance the capability of these neural networks.

With an increasing number of hyper-parameters that a model needs to process, architectures for deep networks to effectively learn these parameters have diversified and contain a vast variety of different approaches. The ResNeXt [1] architecture consists of repeating layers of hyper-parameters while splitting the input into fewer lower-dimensional embeddings then transforming it by a set of specialized filters and then merged by concatenation. A module in the network performs a set of transformations, each on a low-dimensional embedding whose outputs are aggregated by summation. The transformations to be aggregated are all of the same topology. Through this method, a new dimension called “cardinality”, which is the size of the set of transformations is exposed, a major factor in the model performance.

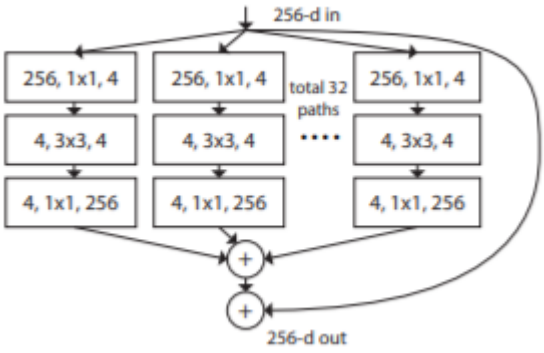

Figure 1. A block of ResNeXt with cardinality = 32. A layer is shown as (# in channels, filter size, # out channels) [1].

In visual object recognition, category-level classification is mostly used, but it is only effective in rough distinctions between object classes and fails to make distinctions with smaller interclass differences and finer details. Fine-grained recognition [4] is a method in recent years that aims to combat this problem by discriminating between object classes and categories with minor differences. With Convolutional Neural

Networks (CNNs) one needs to train networks with large quantities of supervised data, but labeled fine-grained datasets are small, usually containing around 10,000 labeled training images. In scenarios like these, fine-tuning the networks that are pre-trained on large scale datasets such as ImageNet [5] are often used.

In this paper, a training scheme for large scale fine-grained categorization with high performance on ImageNet will be applied to a small-scale fine-grained dataset such as the CUB_200_2011 birds dataset through transfer learning. The ResNeXt 101_32x8d CNN model will be used in building the model. After the model is trained, analysis on the accuracy of classification will be conducted on the model by feeding in new bird images foreign to the model. The goal of this project is to perform analysis on pictures of birds and successfully return the classification of those birds, with a model accuracy of greater than or equal to 80%.

2 Previous Work

Convolutional Neural Networks

Convolutional neural networks (CNN) typically have a standard structure with stacked convolutional layers followed by one or more fully-connected layers. This design along with its variants are frequent among image classification problems and have provided the best results on large image datasets such as CIFAR and ImageNet [5]. For these larger datasets, the common trend has been to increase the number of layers and layer size of the CNN while using dropout to address the problem of overfitting. CNN network architecture has been used successfully for various object detection problems [2]. One of the most successful previous methods of object detection have been Regions with Convolutional Neural Networks (R-CNN). R-CNNs utilize low-level cues such as color and superpixel consistency for potential object proposals in a category-agnostic fashion, and to then use CNN classifiers to identify object categories at those locations. This two stage approach leverages the accuracy of bounding box segmentation with low-level cues, as well as the highly powerful classification power of the best CNNs.

Fine-grained recognition

A way in handling fine-grained recognition problems is to apply visual bag-of-words approaches commonly used for standard object categorization. Because there can be visual similarity between classes, there is a high likelihood that there is a significant amount common visual words shared between classes that do not help in distinguishing classes from each other [3]. One method in handling fine-grained recognition is to reduce background clutter which could interfere in classification using segmentation techniques [8]. In contrast, another method entails directly using detection results to restrict feature extraction to image regions that are likely to contain object parts, along with performing global matching with subsequent ensemble learning using nonparametric part transfer and part feature ensembles [3].

Transfer Learning

CNNs trained on the ImageNet [5] database have been popularly used for transfer learning, through the use of pre-trained networks for fine-tuning or as a feature extractor [9]. Through much extensive research into pre-trained CNNs applied to transfer learning, results have been successful overall, providing motivation for more research into applying transfer learning methods. In prior work, transfer learning between two random splits is easier to perform than natural object splits in ImageNet [5], demonstrating the connection between transfer learning and domain similarity. Azizpour *et al.* [10] conducted a full study on a list of transfer learning tasks that have different similarity with the original ImageNet

classification tasks, such as image classification and instance retrieval tasks. Cui *et al.* [4] demonstrates the use of modern neural network architectures to obtain significantly higher accuracy predictions, through quantifying the similarity between the source and target domain and then choosing a more similar subset from the source domain for better transfer learning. Through the use of deep learning networks combined with high resolution image training, high accuracy for small, fine-grained classification problems can be achieved through transfer learning.

3 Method

3.1 Instructions

Libraries used: torch, torchvision, sklearn, numpy, pandas, matplotlib, seaborn, scikitplot, pathlib, imutils. Pip install sklearn, numpy, torch, torchvision, imutils, opencv-python, pandas, matplotlib, seaborn, scikit-plot and it will include these libraries

In this project, there is just one ipynb file containing the entire project

Part3ML.ipynb

The dataset (CUB_200_2011) is included in a folder along with it split between separate training and testing folders. File path /images/train and /images/test.

All the code is contained within Part3ML.ipynb, which performs the loading of the dataset and splitting it between training and testing sets, creating and training the model, and the results and analysis of the model. Run all the cells in order from the top to perform all these operations.

The original results image got corrupted, so another image with similar results is provided.

3.2 Functions

There were many functions created to encapsulate specific portions of the code for ease in implementation.

def save_pkl(pkl_object, file_name) – saves a created python model as a pkl file

def unpkl(file_name) – loads the python model from a pkl file

def train_model(model, criterion, optimizer, scheduler, device, dataloaders, dataset_sizes,

num_epochs=25, return_history=False, log_history=True, working_dir='output') – builds and trains a CNN model given a pretrained model, loss function, optimizer, scheduler, and number of epochs.

def imshow(inp, title=None, figsize=(20,8)) – displays images given as a grid

def visualize_model_grid(model, class_names, device, dataloaders, num_images=6, figsize=(20,20), images_per_row=3) – displays prediction of model against some training labels along with their respective images

def get_title_string_from_classes(classes, class_names) – helper function to display class names of bird images

def makeTransforms(img_crop_size=224, img_resize=256)— makes transforms and augments to the entire dataset, such as random resizing, flipping, resizing, and cropping. Normalization is also applied to the dataset here.

def make_predictions_proba(model, dataloaders, device) – loads in the testing dataset and runs them through the created model and produces predictions. Returns two numpy arrays of truth labels, prediction labels and class probabilities.

def softmax(x) -- Computes softmax values for each sets of scores in x.

The ResNext-101_32x8d pretrained CNN model architecture is used

Pytorch will be the framework used to implement and train the models.

3.3 Implementation

The Caltech-UCSD Birds-200-2011 (CUB-200-2011) dataset contains 200 different classes with 11,788 images in total, with 15 part locations, 312 binary attributes, and 1 bounding box as annotations per image.

Here is an example of some of the bird images in the dataset:

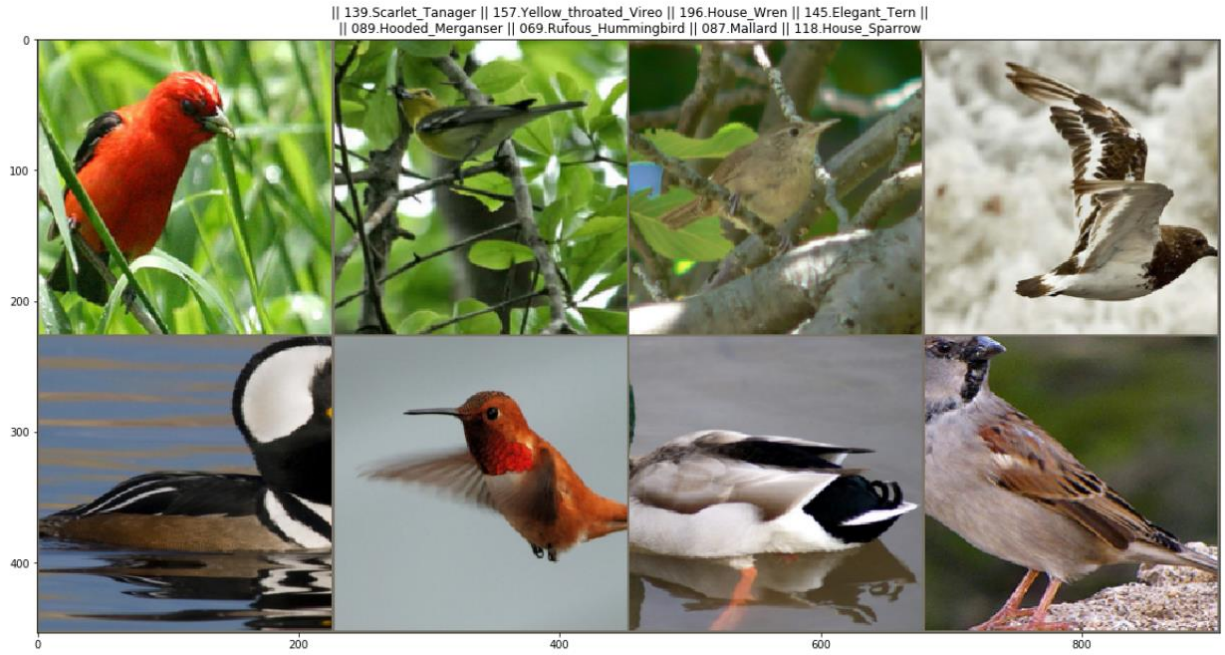

Figure 2. Sample images from the CUB-200-2011 dataset

Images are then split between training and testing sets using the supplied train_test_split.txt file included in the dataset file, which results in 5994 images in the training set and 5794 images in the testing set, each within their own files.

Images are loaded into the program as tensors, where the number of channels and height and width can be freely manipulated for each image. Mild augmentations are then applied to both datasets, with random resize cropping and random horizontal flips applied to the training set, and resizing and center cropping applied to the testing set. Normalization is also applied to both sets after the augmentations have been applied.

After the preprocessing has been completed, the device to run the training is set to the GPU, and the ResNeXt 101 32x8d[7] model with pretrained weights is loaded in. The model architecture contains 101 layers, with 32 groups and a group width (cardinality) of 8d (bottleneck width). The output layer of the model is then modified to fit the number of classes for this dataset from 1000 to 200. The model is then pushed to the GPU for training preparation.

The loss function chosen for training is cross-entropy ($-(y\log(p)+(1-y)\log(1-p))$), the optimizer chosen to update parameters is Stochastic Gradient Descent with a learning rate of 0.01 and momentum of 0.09, and the scheduler for the optimizer is set with a step size of 7 and a gamma (learning rate decay) of 0.1.

Once these values are set, the classifier model is trained by passing in the ResNeXt model, the loss function, the optimizer, the learning rate scheduler, the GPU device, the images in mini-batches, and the number of epochs. The model is trained with 15 epochs with mini-batch size of 8. With each epoch, the loss and accuracy for training and testing is returned.

Once the model is finished training, the best validation accuracy is returned with the best model saved in a pkl file. With the fully trained model, analysis and results can be computed through various tables and graphs.

To use the model to classify images not contained within the dataset, the model is set to evaluation mode and the new images are loaded into the model with the classification returned.

4 Results and Analysis

```
Epoch 0/14
----------
train Loss: 3.9891 Acc: 0.1638
test Loss: 1.9019 Acc: 0.4517

Epoch 1/14
----------
train Loss: 2.2666 Acc: 0.4623
test Loss: 1.2230 Acc: 0.6351

Epoch 2/14
----------
train Loss: 1.7029 Acc: 0.5822
test Loss: 1.0187 Acc: 0.7035

Epoch 3/14
----------
train Loss: 1.4177 Acc: 0.6500
test Loss: 0.8424 Acc: 0.7434

Epoch 4/14
----------
train Loss: 1.2452 Acc: 0.6909
test Loss: 0.8038 Acc: 0.7630

Epoch 5/14
----------
train Loss: 1.0835 Acc: 0.7322
test Loss: 0.8443 Acc: 0.7641

Epoch 6/14
----------
train Loss: 1.0366 Acc: 0.7421
test Loss: 0.7851 Acc: 0.7772

Epoch 7/14
----------
train Loss: 0.8052 Acc: 0.8118
test Loss: 0.6232 Acc: 0.8286

Epoch 8/14
----------
train Loss: 0.7230 Acc: 0.8368
test Loss: 0.5962 Acc: 0.8402

Epoch 9/14
----------
train Loss: 0.6866 Acc: 0.8443
test Loss: 0.5845 Acc: 0.8397

Epoch 10/14
----------
train Loss: 0.6466 Acc: 0.8570
test Loss: 0.5898 Acc: 0.8445

Epoch 11/14
----------
train Loss: 0.6155 Acc: 0.8582
test Loss: 0.5802 Acc: 0.8443

Epoch 12/14
----------
train Loss: 0.6032 Acc: 0.8634
test Loss: 0.5871 Acc: 0.8447

Epoch 13/14
----------
train Loss: 0.5723 Acc: 0.8735
test Loss: 0.5744 Acc: 0.8459

Epoch 14/14
----------
train Loss: 0.5462 Acc: 0.8814
test Loss: 0.5655 Acc: 0.8476

Training complete in 142m 41s
Best val Acc: 0.847601
Returning object of best model.
```

Figure 3. The results of model training at each epoch.

The best validation accuracy result was 0.848, which meets the standard of above 80% accuracy expected for the classification model.

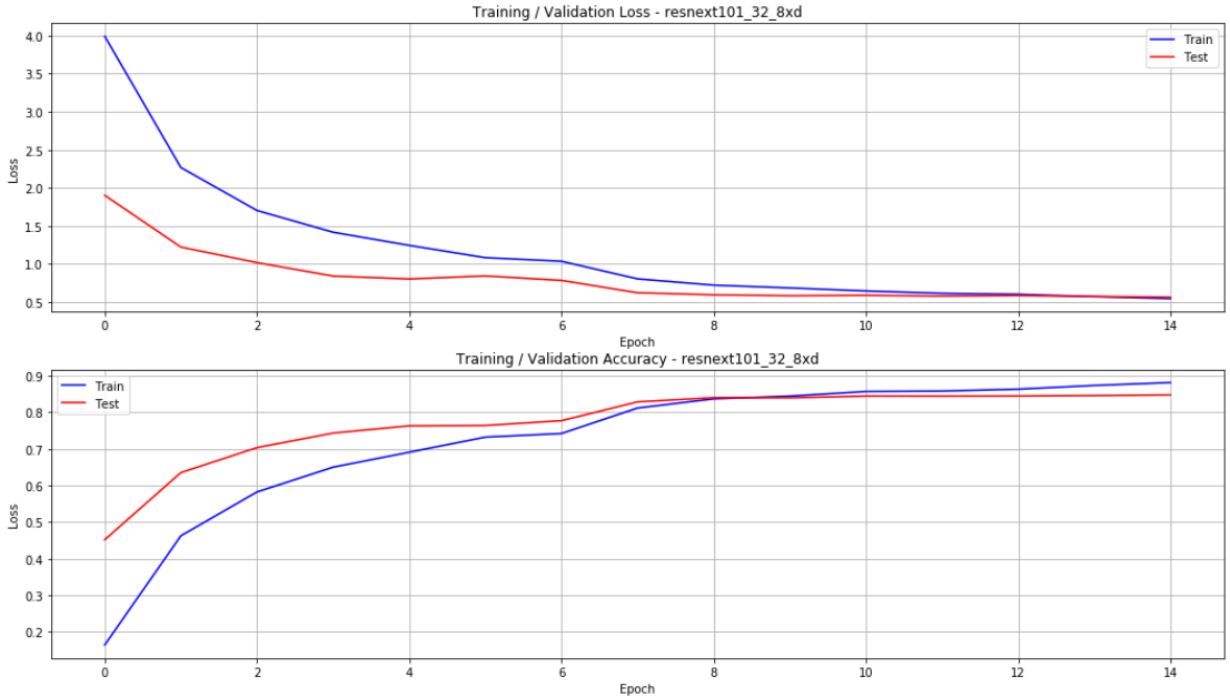

Figure 4. Graph of Loss and Accuracy for Training and Validation

In Fig. 4, the initial loss for both training and validation was high, with the training loss roughly double that of the validation loss. The same pattern goes for the accuracy, where the validation accuracy roughly doubles that of the training accuracy. At around epoch 8 is when both accuracies converge, and after that the training accuracy slightly increases faster than validation. Both graphs show the loss and accuracy converging and maintaining roughly the same value over time, which shows no signs of overfitting or underfitting, making this a strong model.

	199.Winter_Wren	200.Common_Yellowthroat	accuracy	macro avg	weighted avg
precision	0.818182	1.000000	0.847601	0.853589	0.853061
recall	0.900000	0.966667	0.847601	0.848621	0.847601
f1-score	0.857143	0.983051	0.847601	0.847746	0.847035
support	30.000000	30.000000	0.847601	5794.000000	5794.000000

Table 1. Subset of the classification report. Contains precision, recall, f1-score, and class length, along with the averages.

To evaluate the performance of the model, a variety of metrics are calculated. Along with accuracy, Precision calculates predicted true positives over total predicted positives (TP/TP+FP), Recall calculates predicted true positives over all predictions in the same class (TP/TP+FN), F1-score is the weighted average of precision and recall (2*(Recall * Precision) / (Recall + Precision)) and coveys the balance between the precision and recall. From the table above, the average values for all 4 metrics are almost equal, indicating that the model is balanced and performing consistently across all the data. However,

there still may be certain specific classes that the model is not performing well on, which raises some questions on why this is so. These classes will be investigated further.

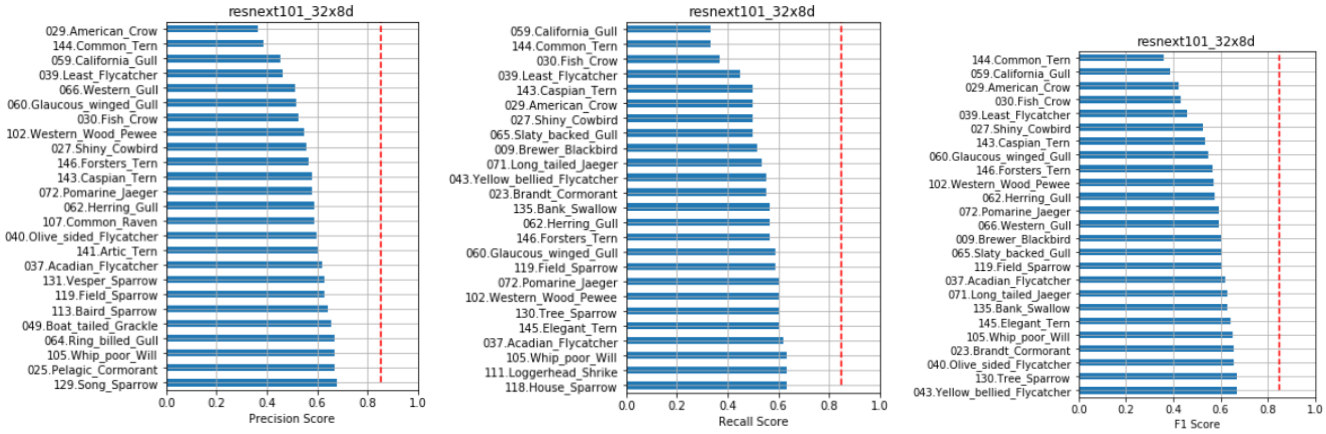

Figure 5. Chart of Precision, Recall, and F1 score of the 25 worst class performers

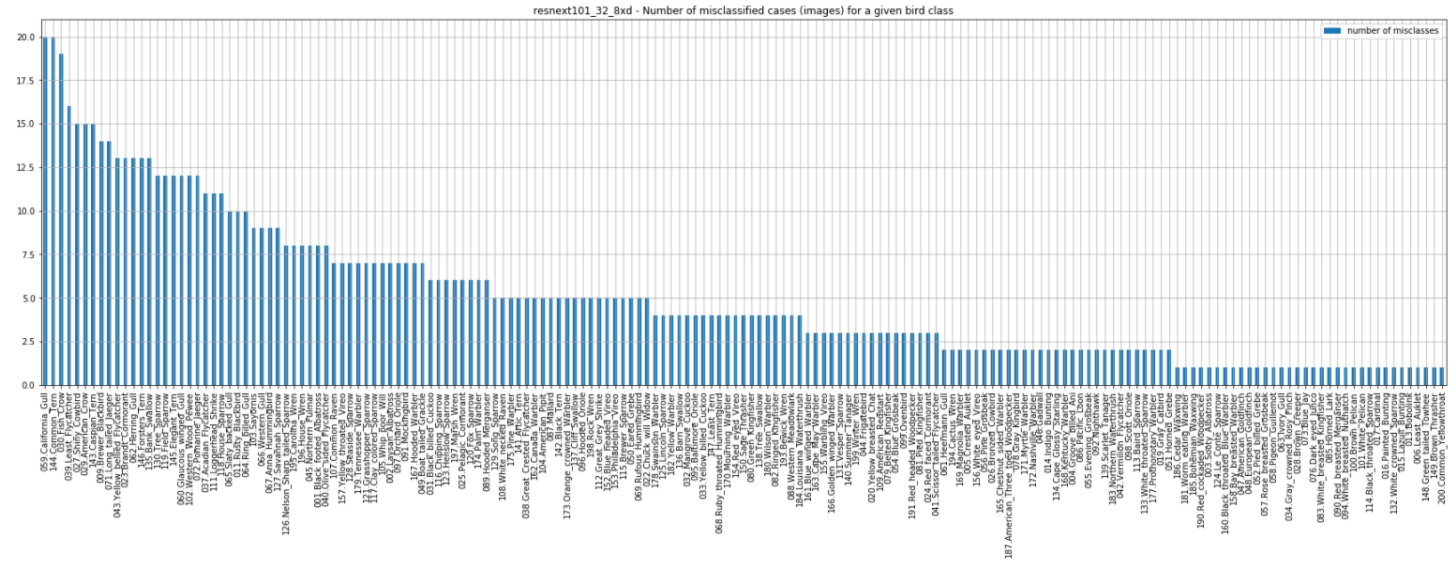

Figure 6. Graph of missclassified images for each class

In Fig. 5, the 25 classes with the worst scores are displayed. It can be seen that Gulls, Crows, Terns, and Ravens are the collective classes that perform the worst in this model. This is most likely due to the fact that different species of Gulls, Crows, Terns, and Ravens are very similar in appearance to each other within their own class. This leads to a higher frequency of misclassified images resulting in these poor scores, shown in Fig. 6 with the California Gull, Common Tern, and Fish Crow as the three most missclassified, with 20, 20, and 19 out of 30 respectively.

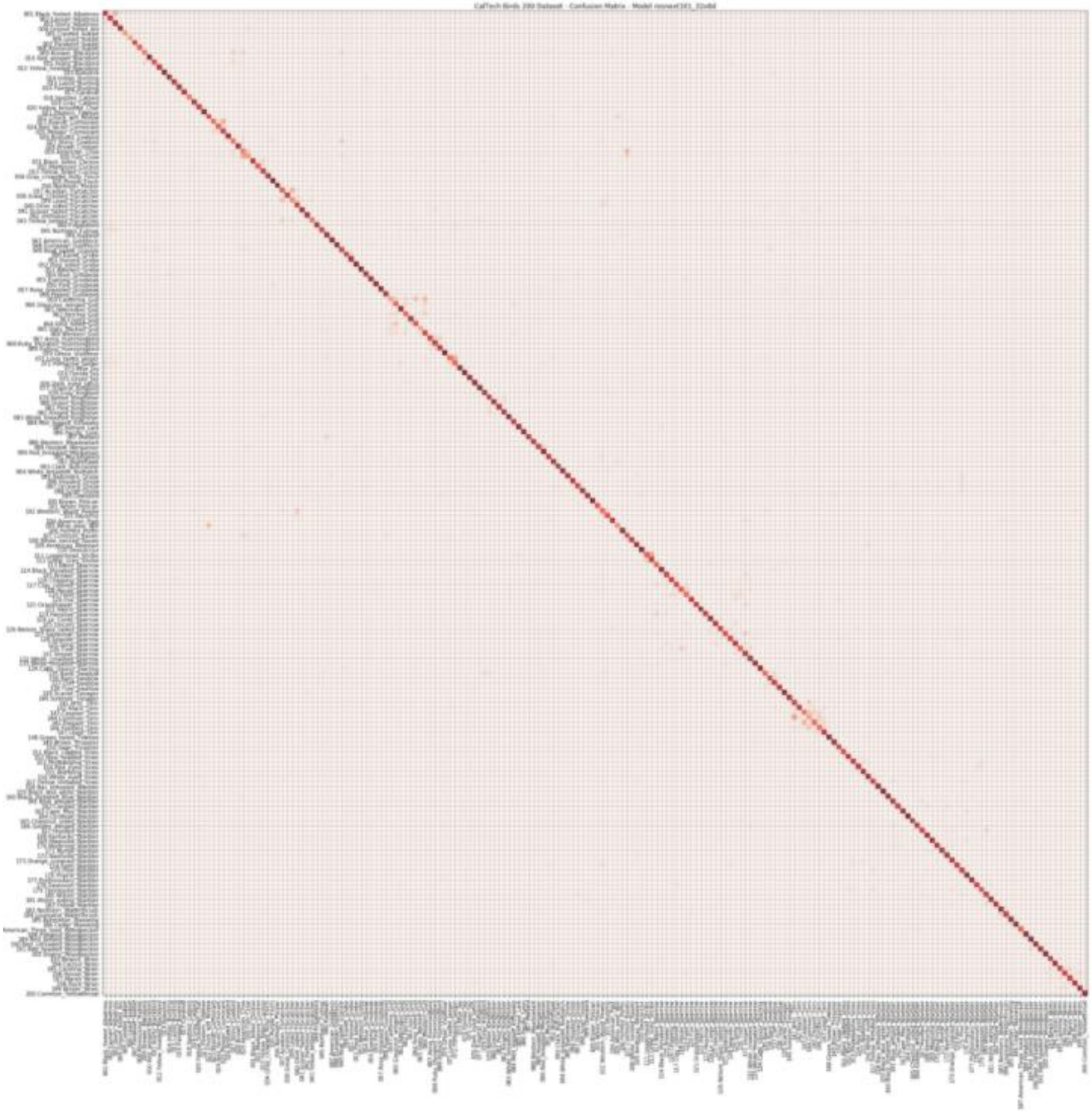

Figure 7. Confusion matrix of all bird classes

	label truth	label pred	scores
0	129	117	[1.8419261e-06, 6.152678e-07, 9.852841e-07, 6....
1	28	29	[6.9501007e-06, 7.3001206e-07, 4.5148886e-06, ...
2	70	71	[0.005031003, 0.00066887983, 0.00028944865, 2....
3	117	118	[1.8872563e-05, 9.509618e-06, 1.2648511e-05, 0...
4	71	0	[0.49189153, 0.00250331, 0.009598562, 1.401115...
...	...	...	...
878	64	58	[1.38553305e-05, 0.0018233204, 3.5272737e-05, ...
879	111	110	[5.222084e-07, 1.1897453e-05, 6.453247e-06, 7....
880	193	91	[2.018151e-05, 0.00010891093, 1.3528699e-05, 9...
881	127	128	[1.1589729e-06, 7.414111e-07, 2.0284488e-06, 2...
882	5	7	[0.001843061, 8.6976e-05, 0.00043429088, 1.589...

Table 2. Subset of misclassified images with true and predicted labels of the image and probability distribution scores.

The confusion matrix in Fig. 7 shows how all the classes performed relative to each other. The confusion matrix produced is represented as a diagonal heat map, where a darker shade of red shows higher class performance with higher accuracies and lower shades of red shows weaker class performance, indicating more mis-classifications for that class. For example, the Gulls bird class was the worst performer out of all the classes and is represented in the confusion matrix as the cluster of scattered weak heat squares around the diagonal. Similar patterns can be seen with Swallows and Terns.

5 Conclusion

The goal of this project was to perform analysis on a pictures of birds and successfully return the classification of those birds. In order to accomplish this goal, the 11,788 images collected from 200 species of birds from the first dataset were split into training and testing datasets. The training set contained 5994 images and the testing set contained 5794 images and was classified using a combination of various image augmentations, cross-entropy loss as the activation function, mini-batch stochastic gradient descent, and the ResNeXt 101 32x8d convolutional neural networks approach to produce learning models. Through the supervised learning method, backpropagation was used in training the model. The best average statistical accuracy was returned at 84%, which is successfully above the 80% benchmark proposed for the project. The model produced can now successfully be used to perform analysis on pictures of birds not included in the training or testing datasets and return the determined classification of the birds.

References

[1] https://arxiv.org/pdf/1611.05431.pdf

[2] https://arxiv.org/pdf/1409.4842.pdf

[3] https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Goring_Nonparametric_Part_Transfer_2014_CVPR_paper.pdf

[4] https://arxiv.org/pdf/1806.06193.pdf

[5] http://www.image-net.org/papers/imagenet_cvpr09.pdf

[6] https://towardsdatascience.com/adventures-in-pytorch-image-classification-with-caltech-birds-200-part-1-the-dataset-6e5433e9897c

[7] https://pytorch.org/hub/facebookresearch_WSL-Images_resnext/

[8] https://www.robots.ox.ac.uk/~vgg/publications/2011/Chai11/chai11.pdf

[9] https://arxiv.org/pdf/1403.6382.pdf

[10] https://arxiv.org/pdf/1406.5774.pdf