Attentive DenseNet for Image Classification (DenseAtt)

Carson Yu Tian Zhao, Mengyu Li, Jiushuang Guo
Statistics Department
Stanford University
czhao333@stanford.edu, lmy18@stanford.edu, jguo18@stanford.edu

Abstract

In this paper, we apply various classification models to the iWildCam 2019 dataset. We experiment with transfer learning on various network architectures including Inception v3, ResNet, ResNext, SENet, and DenseNet with weights pre-trained on ImageNet, to expedite the training process. Then we insert attention layers into the ResNet and DenseNet architectures to improve model performance. We also perform image preprocessing and data augmentation techniques such as rotation, flipping, and color jitter. The best single model is DenseNet with self-attention layers (DenseAtt), with a validation F1 score of 0.93641. After ensembling several versions of DenseNet and DenseAtt, we are able to achieve a F1 score of 0.94112 on the test dataset.

1. Introduction

Due to continuing climate change, degradation of the environment, and destruction of natural habitats, many species of wild animals are in grave danger. Camera traps enable fast monitoring of the biodiversity and population density of various animal species by automatically collecting vast amounts of images in the wild.

Our goal is to be able to accurately classify the animals in the images collected by the camera traps. The input for our classification model are colored RBG channel images. The output of the model are the class scores for each of the animal classes for each image. This image classification problem is made more interesting by a series of data challenges. First of all, not all images have animals in them. Problems include occlusion (e.g. grass covering up part of a rabbit), illumination (nighttime images have low brightness and mainly dark pixels), motion blur (animals in the image appear blurry), camera malfunctions, weather conditions (e.g. rainwater on the camera lens), perspective (animals are too close to the camera and appear distorted), and temporal changes (landscape changes over time).

Our project experiments with a few of the state-of-art image classification models. We improve on pre-trained models with fine-tuning and data augmentation. The transfer learned models are further improved with the insertion of attention layers and modules into the pre-existing architecture. Details of these modifications to the models can be found in Section 3. Information about the dataset can be found in Section 4. Section 5 includes a discussion of the experiments, hyperparameters, quantitative results, and qualitative results of the project.

2. Related Work

The task of image classification exploded into popularity with the rise of convolutional networks in the early part of the decade. Currently, Inception-v3 [13] is widely used for image classification. It uses an efficient "Inception" module to emphasize computational efficiency while having a deeper architecture than its predecessors. Inception uses "bottleneck" layers comprised of 1x1 convolutions to alleviate compute and decrease the number of parameters [13]. It also uses a global average pooling layer to reduce the computational cost of the fully-connected classifier layer, which we will see in many later models.

ResNet trains even deeper networks and adds residual connections. There are 34, 50, 101, and 152 layer versions of ResNet. ResNet-101 and ResNet-152 also use the "bottleneck" layer described above in Inception v3 [7]. To achieve better performance, ResNext uses a new dimension "cardinality", the size of the set of transformations, in addition to depth and width [16]. This is helpful when going deeper and wider yields diminishing returns. A similarity between ResNext and Inception is that both include parallel pathways in their respective modules. The actual splitting of the input to the ResNext module is performed by grouped convolution, where the input channels are divided into groups [16].

To improve on ResNet, Squeeze-and-Excitation Networks (SENet) adaptively recalibrates channel-wise feature maps. The SE-ResNet model includes a feature recalibration module comprised of the ResNet residual block and a gating mechanism with sigmoid activation [9]. The gating mechanism is parameterized by a "bottleneck" with two fully-connected layers separated by a ReLU activation. A channel-wise multiplication between the feature map and the output of the activated gating mechanism results in the final output of the SE-ResNet module [9]. Please see Section 3.2 for more details. SENet is able to improve on state-of-the-art ImageNet models and achieves a top-5 error of 2.251%.

We also examine DenseNet [8], which provides a more efficient architecture for deeper networks. DenseNet will be discussed in more detail in section 3.1.

Attention has been found to be very beneficial in NLP applications. Vaswani et. al (2017) is a landmark paper on attention, demonstrating stacked self-attention. The work shows how a model architecture with only attention mechanisms can outperform those with recurrent and convolutional layers [14]. In the image recognition field, Wang, et. al (2017) created a "Residual Attention Network" comprised of stacked attention modules. The model uses CNNs and attention to generate attention-aware features, performing comparatively with ResNet-200 at only 69% of the forward FLOPs [15]. Furthermore, Zhang, et. al (2018) uses self-attention for long-range dependency modeling in image generation tasks [17]. They found that self-attention is able to capture global-dependencies and efficiently model relationships between widely separated spatial regions [17].

Boosting has been found to be very effective in many machine learning applications. The gradient boosting algorithm is an ensemble method to improve the performance of a (weak) classifier such as decision trees by increasing predictive accuracy. Boosting produces a more powerful model with a much larger function space than that of a single classifier (e.g. single tree), and the boosting function is a lot smoother [5].

Lastly, many datasets suffer from class imbalance where a subset of the classes make up a disproportionate part of the entire data (too large or too small of a ratio). This can lead to poor classification performance [2]. Buda et.al (2018) found that oversampling the minority classes can help correct class imbalance and crucially does not cause overfitting of CNNs that other balancing methods can [2].

3. Methods

3.1. DenseNet

DenseNet improves over Inception v3 by using a much deeper neural network and a connectivity pattern to avoid the vanishing gradient issue. Previous architectures like ResNet [7] applied identity mapping between layers and used a bottleneck layer to enable efficient learning of a deep neural network over 100 layers. However, this kind of architecture can have the vanishing gradient problem, and the summation of residual and layer output can impede the information flow of the network. DenseNet solves this problem by proposing dense blocks which connect all layers directly with each other to maximize the information flow. Each layer in the same block obtains additional inputs from all preceding layers and passes on its feature maps to all subsequent layers. Furthermore, DenseNet introduces a hyperparameter called growth rate to control the size of the input feature-maps. The structure of the dense block can be seen in Figure 1. In our experiment, we apply transfer learning on DenseNet-121 with pre-trained ImageNet weights to our task. DenseNet-121 includes 4 dense blocks.

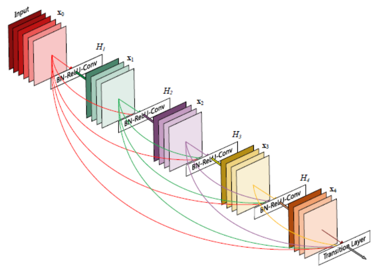

Figure 1: A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature maps as input [8].

3.2. SENet

The Squeeze-and-Excitation (SE) Networks improve the quality of representations by modeling the relationship between channels of its convolutional features. To achieve this, it builds a feature recalibration mechanism by inserting SE blocks into the residual modules and inception modules of ResNet and Inception. As mentioned before, each SE-Block gating mechanism is parameterized by a "bottleneck" with two fully-connected layers separated by a ReLU activation. The gating output can be represented by the following formula:

$$s = \sigma((W_2\delta(W_1 z)) \tag{1}$$

where δ refers to the ReLU activation, σ refers to the sigmoid function, W refers to the fully-connected weights, and z refers to the input to the gating mechanism. The gating output $s \in \mathbb{R}^{(H\times W\times C)}$ is combined with the feature map $U \in \mathbb{R}^{(H\times W\times C)}$ via channel-wise multiplication to yield the final output of the SE-ResNet module.

$$x_c = s_c u_c \tag{2}$$

where $s_c, u_c \in \mathbb{R}^{(H\times W)}$ [9]. We used SENet with both a ResNet and ResNext backbone. The architecture of the SE-ResNet module is shown in Figure 2.

3.3. Self-Attention CNN

The self-attention mechanism proposed by Vaswani, et. al [14] is widely used in Natural Language Processing applications. Self-attention allows the model decoder to learn temporal information and model dependencies regardless of

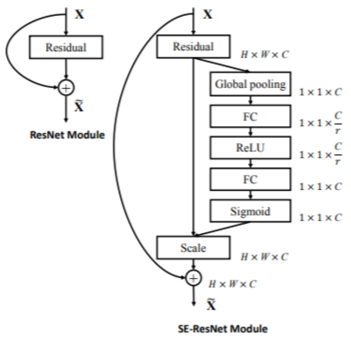

Figure 2: SE-ResNet module (right) compared to ResNet module (left) [9]

distance. In the field of image classification, images have pixels that can contribute more than other pixels to determine which class an image is. Self-attention allows the model to select more relevant image regions to focus on. In our case, as previously mentioned, there are data challenges like occlusion, illumination, motion blur, camera malfunctions, weather conditions, perspective, and temporal changes. Therefore, we believe that self-attention can improve the classification performance for images that suffer from these challenges, and also for the underrepresented classes with fewer samples.

Our Self-Attention CNN Network is constructed by adding self-attention modules into popular image classification networks like ResNet and DenseNet to further improve the classification performance. The convolutional feature maps from the previous hidden layer are passed into the self-attention module in the shape of $x \in R^{(C\times H\times W)}$. In order to calculate attention maps, we first transform the input feature maps into query feature maps $f(x)$ and key feature maps $g(x)$ by two separate 1 x 1 convolutional layers. Then we calculate the attention score for the i^{th} location on the j^{th} region by

$$\beta_{i,j} = \frac{exp(s_{i,j})}{\sum_{i=1}^{N} exp(s_{i,j})}, \text{where } s_{i,j} = f(x_i)^T g(x_j).$$

Finally we compute the new attentive feature maps by transforming the input feature maps into value feature maps $h(x)$ also via a separated 1 x 1 convolutional layer. Then, this is multiplied by the attentive score to get the final self-attention feature maps

$$o = (o_1, o_2, ..., o_j, ..., o_N) \in R^{(C\times H\times W)}$$

where $o_j = \sum_{i=1}^{N} \beta_{i,j} h(x_i)$. The self-attention module can be seen in Figure 3.

We construct self-attention ResNet (ResAtt) by inserting one attention module after the third ResNet block and one

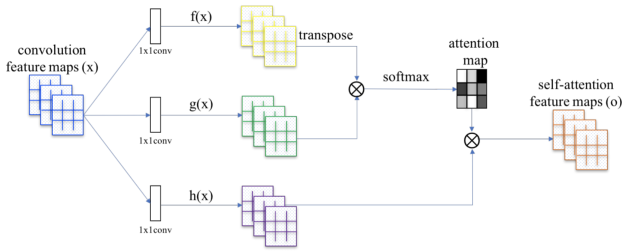

Figure 3: The proposed self-attention mechanism. The $\otimes$ denotes matrix multiplication. The softmax operation is performed on each row [17].

after the fourth ResNet block, and construct self-attention DenseNet (DenseAtt) by inserting one attention module after the transition layer of Dense Block 3 and one after the transition layer of Dense Block 4. The attention-aware features capture global-dependencies, learn pixel-wise relationships, and are able to point the model toward the most informative components of an input. The structures of the two attentive models are shown in Table 1 and Table 2.

Layers	Output Size	DenseAttNet-121
Convolution	112×112	7×7 conv, stride 2
Pooling	56×56	3×3 max pool, stride 2
Dense Block(1)	56×56	$\begin{bmatrix} 1\times 1 & \text{conv} \\ 3\times 3 & \text{conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56 28×28	1×1 conv 2×2 avg pool, stride 2
Dense Block(2)	28×28	$\begin{bmatrix} 1\times 1 & \text{conv} \\ 3\times 3 & \text{conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28 14×14	1×1 conv 2×2 avg pool, stride 2
Dense Block(3)	14×14	$\begin{bmatrix} 1\times 1 & \text{conv} \\ 3\times 3 & \text{conv} \end{bmatrix} \times 24$
Transition Layer (3)	14×14 7×7	1×1 conv 2×2 avg pool, stride 2
Attention Module(1)	7×7	1×1 conv $\times 3$ self-attention dropout, relu
Dense Block(4)	7×7	$\begin{bmatrix} 1\times 1 & \text{conv} \\ 3\times 3 & \text{conv} \end{bmatrix} \times 16$
Attention Module(2)	7×7	1×1 conv $\times 3$ self-attention dropout, relu
Classification Layer	1×1	7×7 global pool fc-13 softmax / lightGBM

Table 1: Model Architecture for Attentive DenseNet

3.4. Boosting CNNs

We also construct a classification model using Light Gradient Boosting Machine (LightGBM) based on the DenseNet features. Boosting is an effective ensemble method based on

Layers	Output Size	ResAttNet-50
Conv(1)	112×112	7×7 conv, stride 2
Pooling	56×56	3×3 max pool, stride 2
Conv(2)	56×56	$\begin{bmatrix} 1 \times 1 & \text{conv} \\ 3 \times 3 & \text{conv} \\ 1 \times 1 & \text{conv} \end{bmatrix} \times 3$
Conv(3)	28×28	$\begin{bmatrix} 1 \times 1 & \text{conv} \\ 3 \times 3 & \text{conv} \\ 1 \times 1 & \text{conv} \end{bmatrix} \times 4$
Conv(4)	14×14	$\begin{bmatrix} 1 \times 1 & \text{conv} \\ 3 \times 3 & \text{conv} \\ 1 \times 1 & \text{conv} \end{bmatrix} \times 6$
Attention Module(1)	28×28	1×1 conv $\times 3$ self-attention dropout, relu
Conv(5)	14×14	$\begin{bmatrix} 1 \times 1 & \text{conv} \\ 3 \times 3 & \text{conv} \\ 1 \times 1 & \text{conv} \end{bmatrix} \times 6$
Attention Module(2)	7×7	1×1 conv $\times 3$ self-attention dropout, relu
Classification Layer	1×1	1×1 avg pool fc-13 softmax

Table 2: Model Architecture for Attentive ResNet

decision trees, combining a large number of weak learners that can outperform individual stronger learners [5]. During the training process, boosting assigns more weights on the misclassified examples in the current learners to let the future learners focus more on these examples to better classify them. Gradient boosting implements gradient descent to minimize the loss function.

LightGBM is a gradient boosting framework for both regression and classification, which uses tree based learning algorithms to efficiently and quickly train on large scale data with reduced memory. This is due to LightGBM using histogram-based algorithms that bucket continuous feature values into discrete bins [6]. We construct boosting with derived DenseNet features by extracting the DenseNet features maps before the fully connected layer, and then passing this into the LightGBM classifier instead of DenseNet classifier. The output is the final prediction scores for each class.

3.5. Model Ensembling

After training various models, we use weighted average ensembling on the test dataset to improve the F1 score. This is the only time we calculate the F1 score on the test set, to avoid bias and fitting to the test set. The weights are calculated by the F1 score on each category for each model so that the model with higher F1 score gets higher weights in the ensembling. Our final predictions on the test set use the ensembled logits which are calculated by $\frac{1}{\#models} \sum_{m \in \text{models}} \text{weights} \times \text{logits}_m$.

4. Dataset and Features

The data is provided by iWildCam2019, and contains images of animals from 138 different locations in Southern California, as well as a CSV file with the image file names and their corresponding class labels [1]. We randomly sampled 100,000 images as our training data, and selected approximately the same number of images for each category from the rest to construct a balanced-class validation and test set with roughly 8,600 images in each.

There are 13 categories in our dataset, consisting of empty (0), opossum (19), raccoon (13), coyote (11), rabbit (8), deer (1), bobcat (16), cat (17), squirrel (3), dog (18), rodent (4), skunk (14) and fox (10). During exploratory data analysis, we noticed that class 0 (no animals in the image) comprises of over 66% of the dataset [1]. This makes logical sense since the cameras record an image every fixed time interval. Unfortunately, this creates a class imbalance issue. According to Buda et.al [2], class imbalance has a detrimental impact on classification performance. The paper found that oversampling the minority classes outperforms other methods for correcting class imbalance and will not cause overfitting.

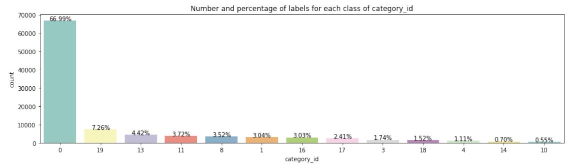

Figure 4: The distribution of classes in training set.

Thus, we apply data augmentation including horizontal and vertical flip, color jitter, and random image rotation on all the classes except the empty class according to their respective proportion of the dataset to oversample the minority classes. Since the data is still a bit imbalanced after the oversampling, a weighted loss function is used to correct for this.

Then, we preprocess the images to make them all the same size. The images in the data can vary in heights and widths but most of them are 1024 by 747. We resize every image to have the same height and width of (224, 224) or (512, 512), depending on the model. We also normalize each image to have a mean of (0.485, 0.456, 0.406) and standard deviation of (0.229, 0.224, 0.225) as instructed by the PyTorch implementation for image normalization on pretrained models. Then we wrote a subclass of the $Dataset$ superclass in $torch.utils.data$ for our dataset so that we could apply oversampling via data augmentation and use the PyTorch data loader function.

4.1. Class Visualization

The images in the dataset have large variance due to different environments and time of day. Picture quality also varies as in some images, it is difficult to spot the animal and correctly classify it by eye. A few images from the dataset are shown in Figure 5. We can see examples of background clutter (1st row, 4th image), illumination (night images), and occlusion (1st row, 3rd image).

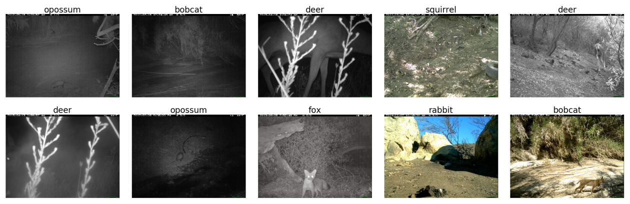

Figure 5: Example pictures of animals from the iWildCam2019 dataset

5. Experiments and Analysis

5.1. Evaluation Metric

The evaluation metric we use to measure the classification model performance are weighted BCEWithLogitsLoss and F1 score. BCEWithLogitsLoss stands for binary cross entropy with logits loss, which combines a sigmoid layer with binary cross entropy loss. For every image, the sigmoid layer turns the output scores from the model into the scores for each class (a number between 0 and 1). Also, after oversampling, the empty class still makes up twice as much as the other classes. Thus, we add class weights to adjust the loss. The resulting weighted binary cross entropy loss is represented by

$$-\frac{1}{n}\sum_{i=1}^{n} w_i(y_i \log \hat{y}_i - (1-y_i)\log(1-\hat{y}_i)). \quad (3)$$

where w_i represents the weight applied, y_i is the true label of the i-th observation, and $\hat{y}_i$ is the predicted label of the i-th observation. Our models try to minimize this loss. F1 score is used to evaluate the performance of the model on the validation and test sets. F1 score is the harmonic mean of precision and recall, which can be calculated with the formula $\frac{2 \times precision \times recall}{precision + recall}$.

5.2. Model Configuration

We randomly extract a 4,000 image subset of the full dataset to determine how various network architectures do on smaller datasets. This is useful as data collection is expensive and many real-world datasets are small. Thus, we thought it would be interesting to see which models perform well if we only had four percent of the full training set.

We use different model configurations for the small dataset and the full dataset. For the 4,000 images subset, we use image height and width of 512, number of epochs 20 (with early stopping on some models), one to four V100 GPUs, batch size of 32, 64, or 128 (depending on memory constraints), and Adam optimizer with learning rate 5×10^{-4} and beta = $(0.9, 0.999)$. The learning rate halves when the training loss does not improve by 1×10^{-4} for five epochs. The lower bound of the learning rate is 1×10^{-5}. We apply a L2 weight decay of 1×10^{-5}, chosen via validation F1 score, on ResNet with attention and DenseNet with attention. We use data augmentation techniques on the ResNet and DenseNet models. Data augmentation (see section 5.3.1) increases the total dataset size to roughly 15,000. We also use gradient clipping with max norm of five to prevent exploding gradients.

For the full dataset, we use image height and width of 224 instead of 512 for faster model training, number of epochs 30-40 (early stopping if validation F1 did not improve for 7 epochs), batch size of 64, 96, and 128 (again depending on memory constraints and number of GPUs), and L2 weight decay of 1×10^{-5} (DenseNet, ResNet), 2×10^{-5} (DenseAtt), and 5×10^{-5} (another version of DenseAtt). The L2 weight decay values were chosen by F1 score performance on the validation set. We use Adam optimizer with the same hyperparameters as described above for DenseNet, ResNet, and SENet. However for DenseNet with attention, we switched to Nesterov momentum with starting learning rate 1×10^{-3} momentum of 0.9. Nesterov SGD improves the best DenseAtt model by 0.6% compared to Adam. We also halve the learning rate whenever the training loss plateaus, as described above. The effect of reducing the learning rate on plateau is very pronounced, leading to noticeable drops in training loss. This can be seen in Figure 7. Since we have class imbalance, we apply a class balancing correction to the logits of the output. Furthermore, we found that dropout rate of 0.1 in the attention module improves the DenseAtt model by 0.2% in the full dataset.

5.3. Quantitative Results

We choose the models with the highest validation F1 scores trained on the 4000 image dataset to train on the full dataset. The results are shown in Table 3 and Table 4.

5.3.1 Data Augmentation

We use data augmentation techniques on the 4000 image dataset to drastically improve the performance. The data augmentation techniques we apply include horizontal flip, vertical flip, color jitter, and random image rotation. This enlarges the small dataset training set to around 15,000 observations. The improvement is significant. F1 score on validation set for ResNet increases 4.6% and increases 4.4% for DenseNet.

For the full dataset, due to computational limitations, we decided to only apply data augmentation to oversam-

Architecture	Valid F1 score
Inception v3	0.7272
ResNet	0.8093
ResNet+Aug	0.8559
ResNet+Aug+Attention	0.8747
ResNet+Aug+Attention+L2 Reg	**0.8759**
SE-ResNet	0.8436
SE-ResNext	0.8610
DenseNet	0.8015
DenseNet+Aug	0.8452
DenseNet+Aug+Attention+L2 Reg	**0.8656**
DenseNet+Aug+LightGBM	0.8387
DenseNet+Aug+Highway Network Gating	0.8560

Table 3: Validation F1 scores of different architectures on 4000 images subset from iWildCam2019 dataset. Aug stands for data augmentation. L2 Reg refers to weight decay.

Architecture	F1 score
ResNet	0.9260
SE-ResNext	0.9306
DenseNet	0.9301
DenseNet + Attention	**0.9364**
Ensemble (5 DenseNets/Atts)	**0.9411**

Table 4: F1 scores of different architectures for iWild classification on 100,000 full dataset. The first 4 results are on validation set and the Ensemble result is on test set.

ple the minority classes. The empty class is already well-represented and objectively the least interesting of the classes, so we feel it is no longer necessary to apply data augmentation on the entire dataset.

5.3.2 DenseNet

DenseNet performs better than ResNet on the full dataset, but performs a little bit worse on the small dataset. This is likely due to the fact that DenseNet is overfitting on the small dataset.

From Figure 6, which shows the train loss curve on the small dataset, we can see that the training loss for DenseAtt is much smaller than the training loss of ResAtt but does worse in terms of F1 score (86.56% vs 87.59%). Thus, DenseAtt is overfitting compared to ResAtt. This argument extends to DenseNet vs ResNet as Figure 8 shows that the validation F1 score for ResNet is higher than DenseNet, which indicates that DenseNet is likely overfitting on the small dataset. This makes sense because DenseNet has a more complicated model structure than ResNet. The connectivity pattern for all layers in a dense block can lead to overfitting on a small dataset. While for the full dataset, the dense block helps alleviate vanishing gradient and strengthen feature propagation, and thus outperforms ResNet.

5.3.3 Attention

Table 3 shows that in the small dataset, DenseNet and ResNet with attention mechanism both significantly outperform the original vanilla models by about 2%. The dropout layer in the attention module acts as regularization and helps prevent overfitting, thus increasing model performance.

After experimenting on the small datasets, we also implement an self-attention mechanism in the DenseNet architecture for the full dataset. The results can be seen in Table 4. DenseNet with our proposed attention mechanism, weight decay, and dropout results in a classification F1 score that is consistently higher than the score without the attention mechanism (93.64% vs 93.01%). Due to computational, time, and budget limitations, we did not experiment with adding our attention mechanism with dropout on ResNet for the full dataset, since ResNet has lower performance on the large dataset compared to DenseNet.

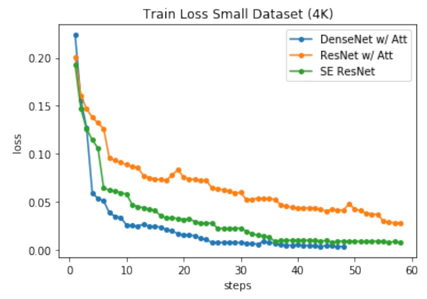

Figure 6: Train loss curves on the 4000 images dataset

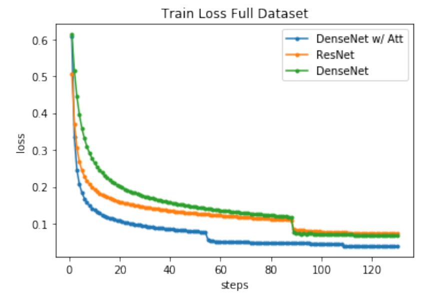

Figure 7: Train loss curves on the full dataset

5.3.4 LightGBM

Even though DenseNet with data augmentation and LightGBM has a lower overall validation F1 score compared with

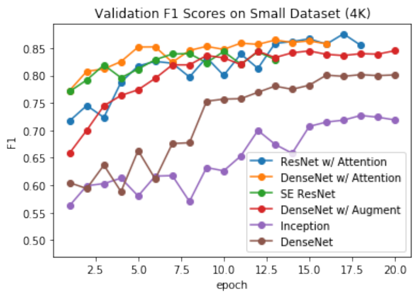

Figure 8: Validation Set F1 Scores

normal DenseNet with data augmentation, we find that LightGBM can help correctly classify classes with few examples. In our experiment on the small dataset, there are only a few images with class label skunk. Using the DenseNet with data augmentation model, the validation F1 for skunk is 0.0, but using a LightGBM classifier based on DenseNet features, the validation F1 for skunk increases dramatically to 0.737. Therefore, if one single class is very important in the classification task but has very few image samples available, then the LightGBM classifier based on DenseNet features may be beneficial.

LightGBM has a major disadvantage compared to the fully-connected classifier when implementing on the large dataset. Since we need to store the feature maps as a data frame with numerous columns, and since the dataset is large, the resulting data frame can have large numbers of rows and columns. This uses an extremely large amount of memory, and thus it is computationally infeasible to apply LightGBM on the full dataset.

5.3.5 SENet

When we apply transfer learning on SENet on both ResNet and ResNext backbones, we find that SE-ResNet outperforms ResNet by 4%, and SE-ResNext outperforms SE-ResNet by 2% in F1 score on the 4000 image dataset. SE-ResNext on the full dataset improves on ResNet by 1% in terms of F1 score and is comparable to DenseNet, but does not do as well as DenseNet with attention. As mentioned in Section 2, SENet is one of the state-of-the-art models on the ImageNet challenge so it is not surprising that vanilla SENet does so well on this image classification task.

5.3.6 Ensemble

Finally, we ensemble the five DenseNet and DenseAtt models that we trained, using their weighted logits to predict the final results on the test set. We are able to achieve an F1 score of 94.11% on the previously untouched test set. The detailed results of precision, recall, and F1-score on each class can be seen in Table 5. From the table, we can see that all F1-scores on test set are above 0.90. The highest are squirrel and rodent with F1 scores of 0.97, and the worst performance is opossum with an F1 score of 0.90. We also notice that the recall rate for opossum is 0.99, which is much higher than its respective precision of 0.82, indicating that our model can classify almost all of the images that contain an opossum correctly, but will have false positives.

Classes	Precision	Recall	F1-score
Empty	0.90	0.98	0.94
opossum	0.82	0.99	0.90
bobcat	0.96	0.91	0.94
deer	0.94	0.94	0.94
squirrel	0.99	0.96	0.97
rabbit	0.99	0.91	0.95
raccoon	0.95	0.95	0.95
coyote	0.97	0.93	0.95
fox	0.94	0.89	0.91
cat	0.97	0.92	0.94
skunk	0.98	0.86	0.91
rodent	0.99	0.96	0.97
dog	0.98	0.92	0.95
total	**0.94**	**0.93**	**0.94**

Table 5: Results of the DenseNet and DenseAtt Ensemble model for each animal class on the test set.

5.4. Qualitative Results

5.4.1 Saliency Map

To see how the classification score changes when we slightly change the image pixel, we compute saliency via backpropagation on our best performing single model, DenseAtt. We plot the saliency maps of DenseAtt in Figure 9. We compute the saliency map for each class by first computing the gradient of the correct class's unnormalized score with respect to all the pixels in an image, which results in a $(3 \times H \times W)$-shape gradient map. Then we take the absolute value of all the gradients, and apply maximization over the 3 channels to get a $(H \times W)$-shape saliency map [11].

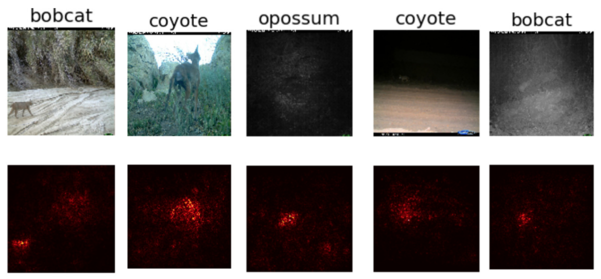

Figure 9: Saliency maps for selected samples

From the saliency maps, we notice that our proposed DenseAtt network can focus on the pixels that contain animals, even with extremely dark backgrounds (for example the opossum and second bobcat images) or if the animal in the image is small (for example the second coyote and second bobcat images).

5.4.2 Weights Visualization

Figure 10 plots the weights of the 64 filters of the first convolutional layer for our DenseNet with attention network. The filters include edges at different orientations and different frequency color features. We notice that the first-layer filter weights are very clear and smooth, which indicates that our network has converged well.

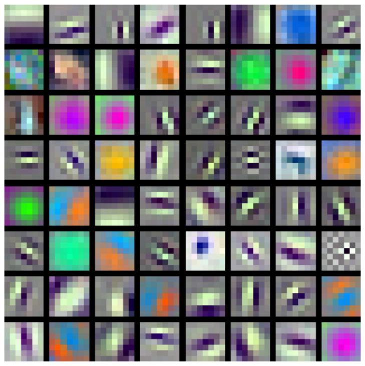

Figure 10: DenseAtt first convolutional layer weights visualization

6. Conclusion

In this project, we applied transfer learning on various classification model architectures and various model sizes to predict animal labels for each image. We experimented with Inception v3, ResNet, ResNext, SE-ResNet, SE-ResNext, and DenseNet. We also added attention layers inside the ResNet and DenseNet pre-trained models. Adding attention improves the previous model in both the small dataset and full dataset case by 2% and 0.6% respectively. For DenseAtt, the attention layers were inserted after the transition layer of the third Dense Block and after the transition layer of fourth Dense Block. We found that adding attention later in the architectures leads to boosts in performance on the validation set F1 scores. This is because attention allows the model to capture dependencies and relationships in images without regard for spatial distance. Our ensembled model of five single DenseNet and DenseAtt models leads to a test F1 score of 94.11%.

Due to lack of time and computational resources, there are multiple more experiments we wanted to try. Further work includes applying the ResNet with attention model on the full dataset, implementing visualizations such as t-SNE and Grad-CAM, exploring optimizers other than Adam and Nesterov momentum, trying to speed up the training of DenseAtt, exploring alternative attention methods like Transformers-XL, and inserting attention into more parts of the DenseNet model.

References

[1] Boydston, Erin, and Justin Brown. “IWildCam 2019 - FGVC6.” Kaggle, Microsoft AI for Earth, Apr. 2019, www.kaggle.com/c/iwildcam-2019-fgvc6.

[2] Buda, Mateusz, Atsuto Maki, and Maciej A. Mazurowski. "A systematic study of the class imbalance problem in convolutional neural networks." Neural Networks 106 (2018): 249-259.

[3] Cadine, Remi. "Cadene/pretrained-models.pytorch", Github, 2018, https://github.com/Cadene/pretrained-models.pytorch

[4] Chawla, Nitesh V., et al. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.

[5] Friedman, Jerome. “Greedy Function Approximation: A Gradient Boosting Machine.” Institute of Mathematical Statistics. The Annals of Statistics Vol. 29, No. 5. 2001

[6] Ke, Guolin, et al. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in Neural Information Processing Systems. 2017.

[7] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[8] Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[9] Hu, Jie, Li Shen, and Gang Sun. "Squeeze-and-excitation networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[10] Park, David Keetae. “Heykeetae/Self-Attention-GAN.” Heykeetae/Self-Attention-GAN, GitHub, 15 June 2018, github.com/heykeetae/Self-Attention-GAN.

[11] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[12] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[13] Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[14] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

[15] Wang, Fei, et al. "Residual attention network for image classification." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.