III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. **Who are you (mini-bio) and what do you do professionally?**

1) Md. Awsafur Rahman
3rd Year Undergraduate Student
Department of EEE, Bangladesh University of Engineering and Technology
Chittagong, Bangladesh

2) Bishmoy Paul
3rd Year Undergraduate Student
Department of EEE, Bangladesh University of Engineering and Technology
Dhaka, Bangladesh

3) Najibul Haque Sarker
3rd Year Undergraduate Student
Department of CSE, Bangladesh University of Engineering and Technology
Dhaka, Bangladesh

4) Zaber Ibn Abdul Hakim
3rd Year Undergraduate Student
Department of CSE, Bangladesh University of Engineering and Technology
Dhaka, Bangladesh

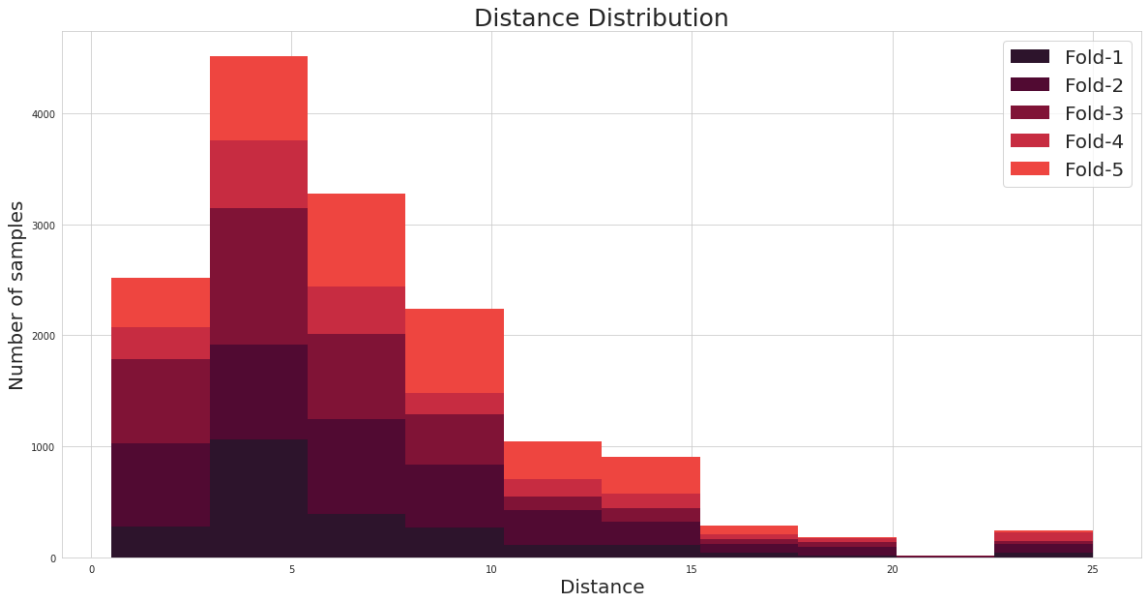
2. **What motivated you to compete in this challenge?**

We are generally interested in participating in out of the box challenges, specifically if they have a positive impact on real life scenarios. Depth Estimation is normally used in autonomous driving situations. Application of this approach in measuring distance from camera traps and animals to monitor wildlife populations sounded very interesting in the first place. Besides, the probable solution methods overlapped with our area of expertise. So, we thought this might be a great opportunity for us to apply our knowledge and learn new applications to eventually be able to help in the great cause of conservation of nature.

3. High level summary of your approach: what did you do and why?

Data Processing

The provided images had time stamps on the border which tend to change from image to image as time changes. While training, we saw through gradcam that the model tended to focus on the timestamp which shouldn't contain any useful information for this particular problem. So we moved the time stamps from the images before training. For similar reasons, we also moved a blue symbol present in the bottom left corner of some images. Furthermore, there were two images with 0 distance. We simply removed them as outliers.



Our main strategy was to use all data training. But first we used StratifiedGroupKFold (this ensures each fold is a good representative of the whole) to split the dataset in 5 folds and measured all performances using this 5 fold cross validation score. By performing cross validation, we got to know the best epoch for our training configurations; namely the epoch when our model typically finds the global minima in optimization. Using this information, we do all data training and only take the trained model of this specific epoch. This ensures the model doesn't become overfit or underfit. We saw that the higher dimension the training images are, the more the model seemed to generalize better. In fact the test scores got the biggest boost when the model was trained using 720x1280 shape images. The higher dimensionality of the training data should help the model provide a more accurate distance mapping, so in this case the experiments also agree with the intuition. Through our experiments, we saw that a solution incorporating multiple image sizes tended to have the best scores.

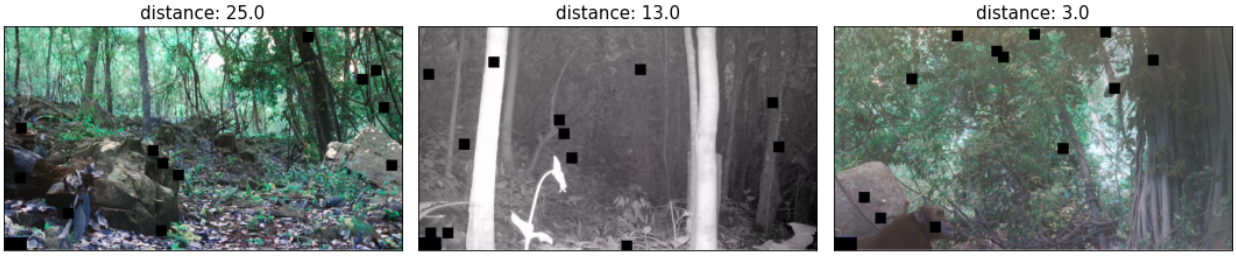
Data Augmentations

We used various data augmentations and saw the effect these had on training through the use of gradcam. The selected final augmentations are depicted below:

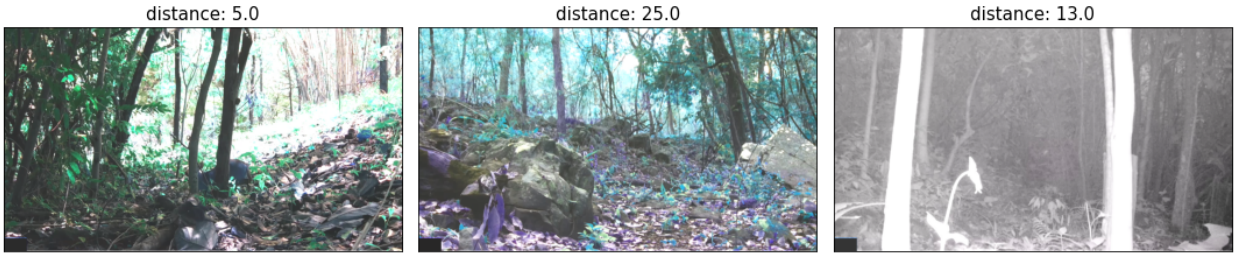
Affine Transformations: We used rotation, shear, zoom and shift to increase the adaptability of the model to calculate distance from different camera positions and angles.

Distortion: Random application of optical and grid distortion proved helpful in our case.

Cutout: Patches of small sizes were cut randomly from the training images.



Color Augmentations: Different training images contained variable amounts of light, brightness, color contrast etc. So, we tried altering these properties of images to ensure better generalization. We randomly applied Contrast, Saturation, Brightness and Hue.



ToGray: As some of the images were in grayscale, we randomly transformed a few images to grayscale.

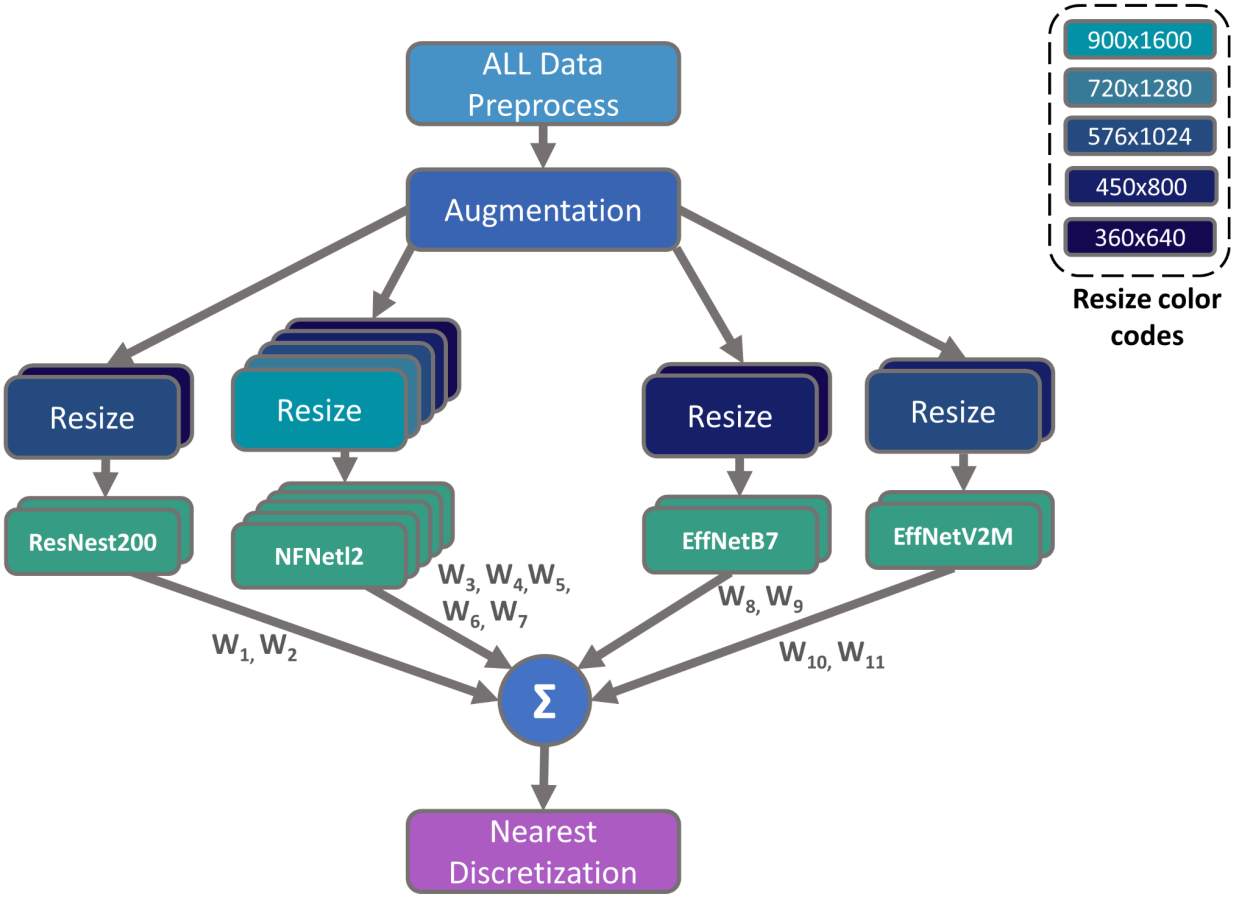
Optimizer & Learning Rate

We used Adam as our optimizer. We used batch size dependent learning rate. Cosine and Step learning rate scheduler were used with different models.

Loss Function

We used MAE and Huber Loss alternatively with different models.
Using MAE comes with some downsides. Such as slow convergence and oscillation of the metric if a larger learning rate is used. Huber loss solves this issue as described.

Our entire procedure can be summarized in this below image:



Ensemble

We trained 5 NFNetl2, 2 Resnest200, 2 EfficientNetB7 and 2 EfficientNetV2M models with varying image sizes as training input. The final collection of models is the result of countless experiments through which we selected the best models and their best training image sizes. Then the soft labels of the models were ensembled together with varying weights designed to get the optimal performance of each model. Then the ensemble labels were post processed to get the final output.

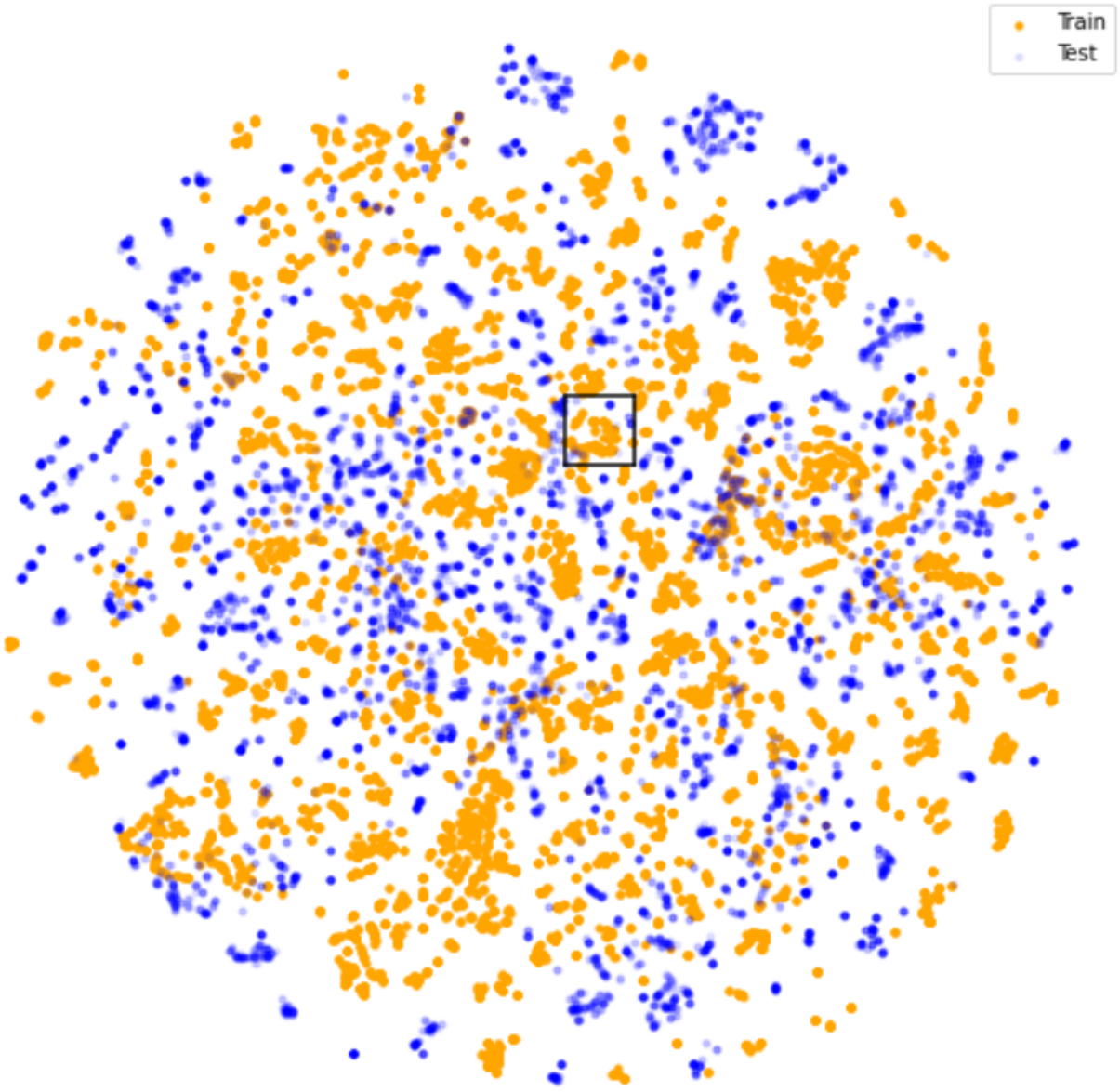
Post Processing

As the train labels and test labels were done by the same professionals, the labels should have the same distribution. In order to utilize this idea, we rounded the ensemble output to their nearest discrete values corresponding to the discrete labels of the training dataset.

4. **Do you have any useful charts, graphs, or visualizations from the process?**

We applied TSNE(t-distributed stochastic neighbour embedding) algorithm on model embeddings to view train and test images of similar distributions. One case is provided below:

TSNE Plot:



We can see the images who have quite similar distributions by taking a small area from the tsne graph and viewing the images from that space:

Train images from TSNE:

Test images from TSNE:

Furthermore we used gradcam to gain model insight and considered it as a visual feedback for our experiments.

5. **Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.**

1. The first one is the use of RandomJitter Function. This made our model color-invariant.

```
def RandomJitter(img, hue, sat, cont, bri):
    img = tf.image.random_hue(img, hue)
    img = tf.image.random_saturation(img, sat[0], sat[1])
    img = tf.image.random_contrast(img, cont[0], cont[1])
    img = tf.image.random_brightness(img, bri)
    return img
```

2. Using Cosine LR Scheduler improved our score for most of the models.

```
elif CFG.scheduler=='cosine':
    decay_total_epochs = CFG.epochs - lr_ramp_ep - lr_sus_ep + 3
    decay_epoch_index = epoch - lr_ramp_ep - lr_sus_ep
    phase = math.pi * decay_epoch_index / decay_total_epochs
    cosine_decay = 0.5 * (1 + math.cos(phase))
    lr = (lr_max - lr_min) * cosine_decay + lr_min
```

3. We finally performed weighted ensemble to merge prediction of different models

```
# ENSEMBLE SUBMISSION
best_sub = self.sub_preds[:,self.indices[0]]
for w_idx, m_idx in tqdm(enumerate(self.indices[1:]),
                         desc='sub ',
                         total=len(self.indices[1:]),
                         bar_format=tqdm_bar):
    best_sub = self.weights[w_idx]*self.sub_preds[:,m_idx] + (1-self.weights[w_idx])*best_sub
```

6. **Please provide the machine specs and time you used to run your model.**
   - CPU (model): Intel® Xeon® Scalable (Skylake)
   - GPU (model or N/A): 8x NVIDIA Tesla V100
   - Memory (GB): 8 x 32GB
   - OS: Amazon Linux
   - Train duration: 2 weeks
   - Inference duration: 0.48s (per image)

7. **Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?**

   The learning rate scheduler we used is dependent on the batch size. So in order to properly reproduce our solution, you have to either use our designated batch size for training or mimic the learning rate according to our original learning rate scheduler graph. Otherwise we cannot guarantee that the global minima found by the models would be similar.

8. **Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?**

   We used sklearn, pandas packages for data preparation and similar tasks. For data visualization we used opencv, matplotlib and pyplot. These packages were also used in exploratory data analysis, along with t-distributed stochastic neighbour embedding (TSNE) for data distribution analysis and gradcam for model performance analysis. We also utilized wandb (a MLOps tool) to keep things organized.

9. **How did you evaluate performance of the model other than the provided metric, if at all?**

   We applied gradcam on our model outputs and compared the best performing *k* images and worst performing *k* images. Gradcam highlights the part of the image which the model deems important , so we could get an intuition on how the model is working. By storing these outputs using wandb, we could see in which particular cases each of the models performed poorly. Furthermore, we then used appropriate augmentations in the training data to mitigate those cases and make the models more robust.

10. **What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?**

    Among our experiments, we used a few techniques which we thought would greatly improve the predictions but unfortunately it didn't.

    1) A depth map is a picture where every pixel has depth information, so we thought incorporating the depth map of each image with the training data would assist the

model in figuring out the distance of the animal. But we saw no improvements from pure images as training data.

2) We were provided with boundary boxes of animals for a specific frame. We converted the boundary box information to a mask and used it as a 4th dimension for every 3 dimensional image data. This should have helped the model localize the target animal in the frame easily, but including this data showed no improvement. We thought the provided boundary box data was too noisy which might be a reason we couldn't utilize it perfectly.
3) We also directly used the provided boundary boxes in the images as metadata but again this didn't enhance the performance. We again attributed this to the noise in the provided data.

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

We couldn't properly use the object detection data this time around. That's why we might try to build on our previous related experiments regarding usage of metadata using traditional machine learning models. Specifically, using the embeddings of trained deep learning models along with the provided metadata (with additional engineered features) as training set for machine learning models is a method we would explore more deeply.

More accurate object detection data would be a feature we would find very helpful. We do have a suggestion in this regard. As each image was examined by a professional in order to get the *distance* label, the same person could also simultaneously annotate the image with the proper bounding box of the animal. This should be much more accurate than the provided data.