INSECT IMAGE CLASSIFICATION AND DETECTION USING CUSTOM CNN & PRE-TRAINED MODELS

Nicole Koshy, Siddhi Jadhav & Tina Joseph

INTRODUCTION

Insects are a diverse group with over 120 Taxonomic orders and about 1.3 million species of Arthropods. They are an important part of the ecosystem and play diverse roles as pollinators, pests, and food sources. Several repositories like iNaturalist and Project Noah use citizen observations to track wildlife movements from birds to insects and animals. There are some insect identification applications, but these are not as widespread or well. Google Lens is often the only application that can offer species or family-level identification of insects for researchers. Hence, there is a need for better insect identification programs. The aim of our project is to build an insect classification algorithm that can correctly classify insects into their taxonomic order.

PROBLEM DESCRIPTION

Insect image identification is a complex problem. Despite the diversity found in any given location, species of different taxonomic orders share several external characteristics in terms of structure and color. This makes identification a challenge for researchers and algorithms alike. The most common models used for image classification include Convolutional Neural Networks (CNN) and Vision Transformers. CNNs are popular for their diversity in handling different architectures which makes them suitable for diverse types of computational problems. ResNet can handle deep networks and prevent vanishing gradients. MobileNet is lightweight and efficient and uses fewer resources. DenseNet was built for improved information flow, feature reuse, and better performance[1]. InceptionV3 uses multi-scale processing for diverse image features to produce better accuracy with reduced computational cost[2]. The disadvantage of these models is that they are computationally expensive with large datasets. Our aim in this project was to test several custom CNN as well as pre-trained models with minimal modifications, to determine which would perform best, before proceeding to fine-tune this model to improve the accuracy of classifying insect images into taxonomic orders. We also performed object detection along with classification using one model to evaluate whether this would produce better results than classification alone.

DATA DESCRIPTION

For the classification part of our project, we used two main datasets:

1. **ArTaxOr** [4] – This is a Kaggle dataset that contains about 15000 annotated images evenly distributed between 7 insect Taxonomic orders: Araneae, Coleoptera, Hemiptera, Hymenoptera, Odonata, Diptera and Lepidoptera. The corpus is approximately 11GB in size.
2. **Mobile Corpus** – This is a corpus of mobile phone images originally distributed among 10 taxonomic orders. For the purposes of this project, we will only be using the 7 orders that are found in the ArTaxOr so that we can use the same models to train and evaluate the performance with both corpuses. This corpus contains about 14000 images of a little over 390 species of insects. It is slightly imbalanced since the orders of Odonata, Araneae and Lepidoptera contain the most images. It is about 44.5 GB in size. This dataset is referred to as “DL Testing Set”.

To test the performance of the models we also used a smaller subset containing an even mix of images from both datasets that is organized in the same manner as the larger datasets. This is referred to as “DL Small Artax_Mobl Mix”.

For object detection, we only used the ArTaxOr dataset since these images were annotated and contained bounding box information which was required for training the object detection model.

METHODOLOGY

Image Classification:

Image Size Analysis and Pre-processing:

We began image analysis by studying the sizes and distribution of all images in both datasets. Distribution analysis showed us that ArTaxOr was primarily composed of smaller images within the range of 100-500 Mpix while the Mobile

Corpus primarily had larger images between 1100-1600 Mpix in size (Results and code in Appendix II – 1). This helped inform our decision on image resizing and normalization to prepare them for models.

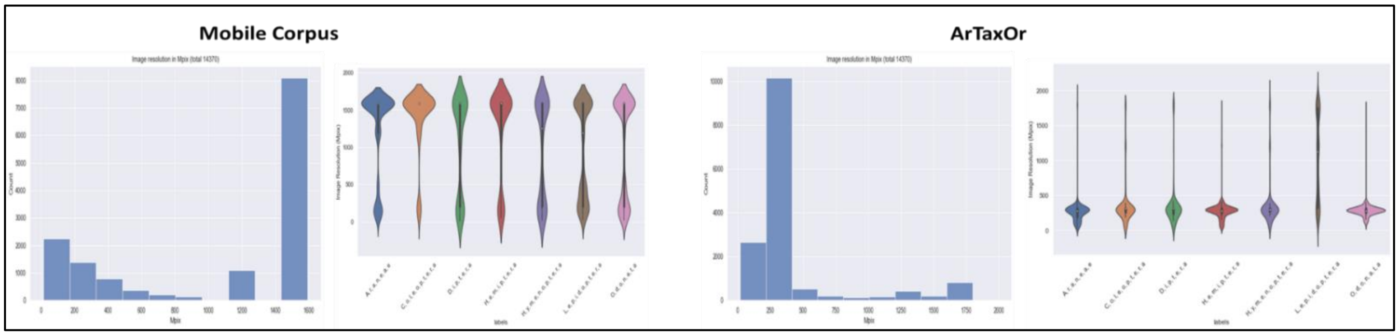


Figure 1: Image size distribution comparison of Mobile Corpus (left) and ArTaxOr (right) using histogram and Violin plots.

To preprocess images, they were first converted to PyTorch tensors and resized to between 220-300 pixels depending on the model. Pixel values were then normalized. In some cases, with the pre-trained models, we also attempted Keras based image augmentation using rescaling, zoom, geometric transformation and changes in dimensions and brightness to observe whether this would improve the accuracy of the model. In these cases, image augmentation was only used for the training and validation sets, while only pixel resizing was performed for the test sets.

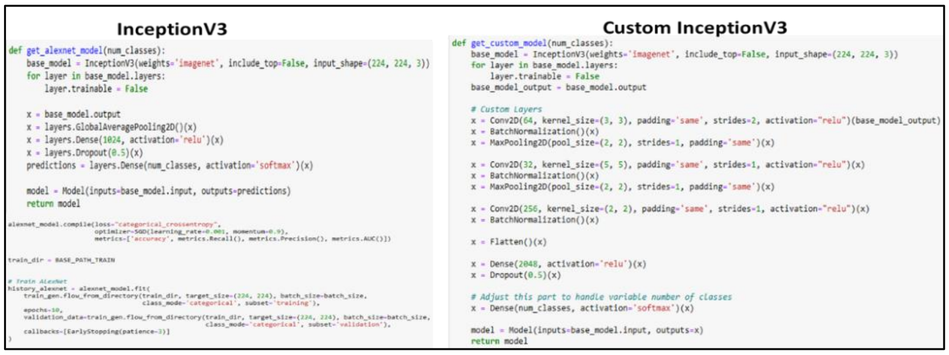
Model Architecture:

We began classification by trying several different custom CNN models with varying architectures and 4 insect classes. Common features among these models were presence of convolution, batch normalization, max pooling, flattening, dense connection and dropout layers. All the models employed adam optimization, a combination of relu and softmax activation along with either sparse categorical or categorical cross entropy loss functions depending on initial Image preprocessing steps. We tuned several hyperparameters along with adding and removing layers from our models. In all cases, while the models were able to achieve upwards of 90% train accuracy, no model could achieve more than 40% validation accuracy. This overfitting prompted us to change our strategy and try pre-trained CNN models.

We tested several pre-trained models including DenseNet, ResNet, InceptionV3, and MobileNet[4]. For every model, we began by using the base model and only added single layers for global average pooling to improve robustness, dense with relu activation to connect all layers, dropout generalization to reduce overfitting, and a final dense layer with SoftMax activation for predictions. We used categorical cross entropy as the loss function and SGD for optimization.

Each model was trained on ArTaxOr and the Mobile corpus and stored in a .h5 file. We evaluated the performance of the models based on training and validation accuracy as well as the model's performance in predicting labels for both datasets and the mixed dataset (DL Small Artax_Mobl Mix). We experimented with changing learning rates, number of epochs and early stopping to see if this would improve the performance of some models. Results of these models are included in the Jupyter notebooks with all other results for each models. We chose the model (InceptionV3) that had the most balanced performance for further modification.

The InceptionV3 model was then further modified to add three convolutional blocks with relu activation and batch normalization and max pooling along with flattening, dense connections, dropout and a dense layer with softmax activation for output. Loss and optimization parameters were kept the same but early stopping was removed. In an initial result that was presented for this model when trained on ArTaxOr, early stopping had been retained which led to lower accuracies. This has been rectified so that the same parameters are applied while training the models on both corpora. In addition to evaluating performance based on accuracy, we also looked at confusion matrices for at least 2 of 3 datasets to ascertain whether the overall test accuracy was affected by lower predictions for certain insect classes.


InceptionV3

```python
def get_alexnet_model(num_classes):
    base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    for layer in base_model.layers:
        layer.trainable = False

    x = base_model.output
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(1024, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    predictions = layers.Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)
    return model

alexnet_model.compile(loss="categorical_crossentropy",
                      optimizer=SGD(learning_rate=0.001, momentum=0.9),
                      metrics=['accuracy', metrics.Recall(), metrics.Precision(), metrics.AUC()])

train_dir = BASE_PATH_TRAIN

# Train AlexNet
history_alexnet = alexnet_model.fit(
    train_gen.flow_from_directory(train_dir, target_size=(224, 224), batch_size=batch_size,
                                  class_mode='categorical', subset='training'),
    epochs=10,
    validation_data=train_gen.flow_from_directory(train_dir, target_size=(224, 224), batch_size=batch_size,
                                                  class_mode='categorical', subset='validation'),
    callbacks=[EarlyStopping(patience=3)]
)
```

Custom InceptionV3

```python
def get_custom_model(num_classes):
    base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    for layer in base_model.layers:
        layer.trainable = False
    base_model_output = base_model.output

    # Custom Layers
    x = Conv2D(64, kernel_size=(3, 3), padding='same', strides=2, activation="relu")(base_model_output)
    x = BatchNormalization()(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=1, padding='same')(x)

    x = Conv2D(32, kernel_size=(5, 5), padding='same', strides=1, activation="relu")(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=1, padding='same')(x)

    x = Conv2D(256, kernel_size=(2, 2), padding='same', strides=1, activation="relu")(x)
    x = BatchNormalization()(x)

    x = Flatten()(x)

    x = Dense(2048, activation='relu')(x)
    x = Dropout(0.5)(x)

    # Adjust this part to handle variable number of classes
    x = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=x)
    return model
```

Figure 2: Differences in architecture of the pre-trained CNN model InceptionV3 and the customized final model

All the Jupyter notebooks, model files, and the smaller image corpora have been uploaded to OneDrive and are available for review (Appendix II). It should be noted that while the custom CNN models did not produce optimal results, these models ran in under 2 hours. Conversely, the pre-trained models that produced better results took an average of 6 to 8 hours to run for each model. Generating the normalized confusion matrices for the bigger datasets also took an equally long time to compile.

Object Detection and Classification:

Next, we tried object detection along with classification to investigate if this would improve our overall results. Using YOLOv8 our objective was to precisely locate and identify morphological features within insect images to augment the efficiency and accuracy of classification using object detection. Below is the architecture for the model.

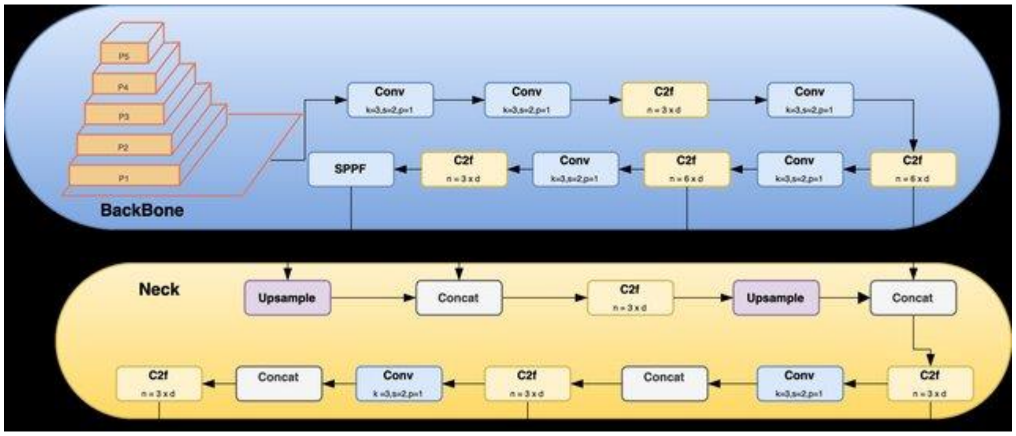


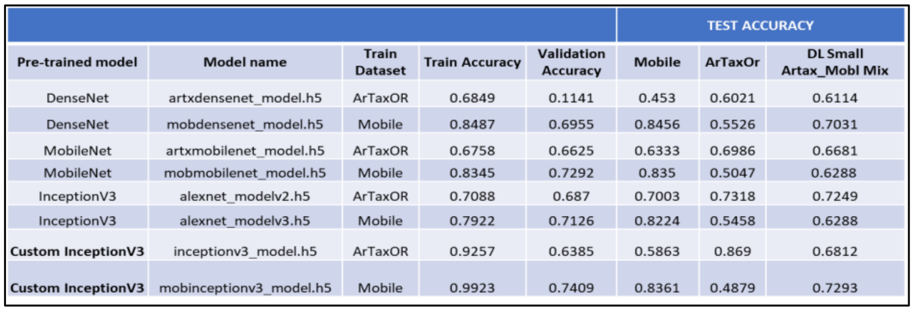
Figure 3: Architecture of the Yolov8 model [6]

To utilize Yolov8, we first imported all requirements along with the main ultralytics package [7]. We first input the dataset images and corresponding annotations which contain bounding box information for each object in the image. Through a training phase, the model learns to recognize and predict these bounding boxes, honing its ability to discern intricate features within the images. The model has 168 layers, 11,128,293 parameters and was trained for ten epochs. The best model was saved in *.onnx format. Once trained, the model was evaluated on unannotated images, where it identified and localized objects based on the learned patterns and performed predictive multi-class classification.

RESULTS

Image Classification:

Beginning with a CNN model with only 9 layers, we trained several models with each corpus and consolidated the results. For this report, we have focused on the results of the 4 models with the best results. These models contained between 158 to 431 layers depending on the pre-trained model used. However, we received our most balanced results with the Custom Inceptionv3 model which had 319 layers. Results in Figure 4 show that the mixed corpus's test accuracy was between that of the two original datasets in all cases and closest to the validation accuracy. Furthermore, models trained on the Mobile Corpus consistently performed better than those trained with ArTaxOr. This could be because the images in this corpus were bigger and retained more detail when resized for input into the models.



					TEST ACCURACY		
Pre-trained model	**Model name**	**Train Dataset**	**Train Accuracy**	**Validation Accuracy**	**Mobile**	**ArTaxOr**	**DL Small Artax_Mobl Mix**
DenseNet	artxdensenet_model.h5	ArTaxOR	0.6849	0.1141	0.453	0.6021	0.6114
DenseNet	mobdensenet_model.h5	Mobile	0.8487	0.6955	0.8456	0.5526	0.7031
MobileNet	artxmobilenet_model.h5	ArTaxOR	0.6758	0.6625	0.6333	0.6986	0.6681
MobileNet	mobmobilenet_model.h5	Mobile	0.8345	0.7292	0.835	0.5047	0.6288
InceptionV3	alexnet_modelv2.h5	ArTaxOR	0.7088	0.687	0.7003	0.7318	0.7249
InceptionV3	alexnet_modelv3.h5	Mobile	0.7922	0.7126	0.8224	0.5458	0.6288
Custom InceptionV3	inceptionv3_model.h5	ArTaxOR	0.9257	0.6385	0.5863	0.869	0.6812
Custom InceptionV3	mobinceptionv3_model.h5	Mobile	0.9923	0.7409	0.8361	0.4879	0.7293

Figure 4: The performance of the 4 best performing pre-trained models trained on either the ArTaxOr or Mobile corpus.

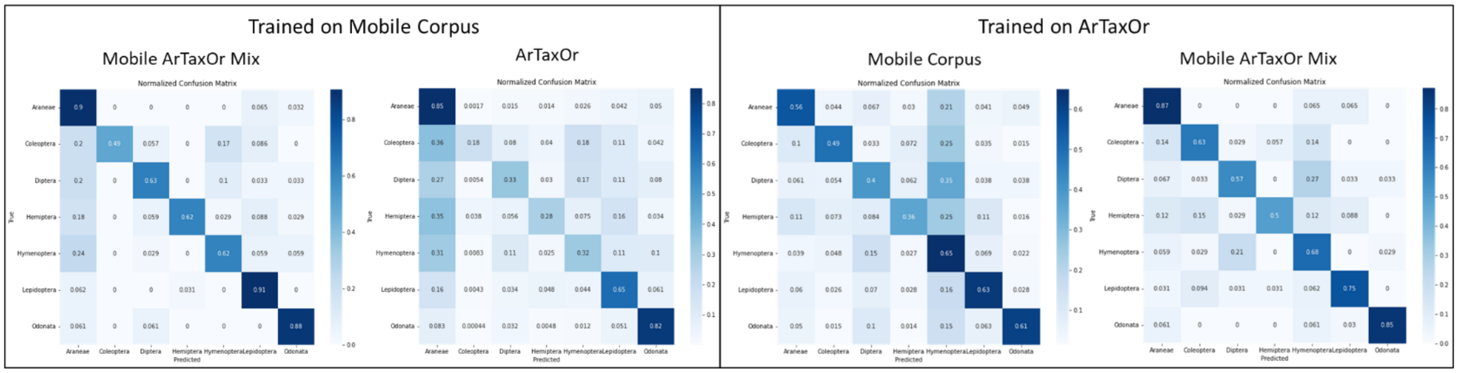


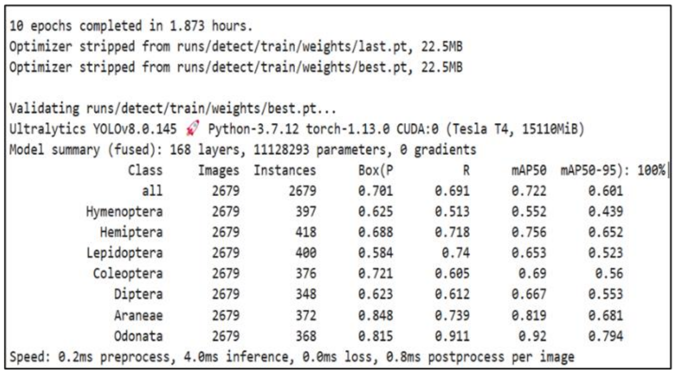
Figure 5: Confusion matrices for class predictions for Custom InceptionV3 model trained on Mobile Corpus and ArTaxOr and evaluated on the other 2 datasets.

Confusion matrix analysis confirmed this observation. Both models showed similar class level predictions for the mixed corpus with the highest accuracies seen for Aranea (spiders), Lepidoptera (butterflies and moths) and Odonata (dragonflies and damselflies). We see lower accuracies for the remaining classes, when the ArTaxOr corpus is tested with the model trained on Mobile. This could be expected in the case of the Mobile corpus since the dataset is imbalanced and contains more images for some classes. However, the model trained with ArTaxOr, while giving a similar accuracy across all classes, had a lower overall test accuracy across all datasets. This could be because the quality of the images in this corpus were affected by their smaller sizes during preprocessing.

Object Detection and Classification:

In evaluating the results, we utilized precision, recall, mAP50 and a confusion matrix to comprehensively assess the performance of our model. Figure 6 provides a detailed breakdown of correct and misclassifications across various insect species. Additionally, to illustrate the model's accuracy, we present sample images alongside confidence scores, displaying the model's confidence in predicting the correct species. This visual representation not only demonstrates the precision of our model but also emphasizes its ability to classify insects into their respective species, providing a robust and reliable outcome. Figure 7 shows four sample images with insects detected with a particular accuracy level.

The model scored 72.3 in mAP (Mean Average Precision), with recall at 69.3 and precision at 70. We prioritize mAP because it fully describes how well the model finds and identifies objects. A higher mAP means better and more consistent performance, crucial for accurate object detection in real-world situations. Performance varied across different classes, with some classes (e.g., Araneae and Odonata) showing higher precision, recall, and mAP. The model performed better in detecting Odonata instances with high precision, recall, and mAP. Some classes, like Hymenoptera, show lower precision and recall, indicating potential areas for improvement in detection for those specific classes. The overall accuracy of the model is mAP50 of 72.2% which is similar to our classification model.



```
10 epochs completed in 1.873 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 22.5MB
Optimizer stripped from runs/detect/train/weights/best.pt, 22.5MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.145 🚀 Python-3.7.12 torch-1.13.0 CUDA:0 (Tesla T4, 15110MiB)
Model summary (fused): 168 layers, 11128293 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|
                   all       2679       2679      0.701      0.691      0.722      0.601
           Hymenoptera       2679        397      0.625      0.513      0.552      0.439
             Hemiptera       2679        418      0.688      0.718      0.756      0.652
           Lepidoptera       2679        400      0.584       0.74      0.653      0.523
            Coleoptera       2679        376      0.721      0.605       0.69       0.56
               Diptera       2679        348      0.623      0.612      0.667      0.553
               Araneae       2679        372      0.848      0.739      0.819      0.681
               Odonata       2679        368      0.815      0.911       0.92      0.794
Speed: 0.2ms preprocess, 4.0ms inference, 0.0ms loss, 0.8ms postprocess per image
```

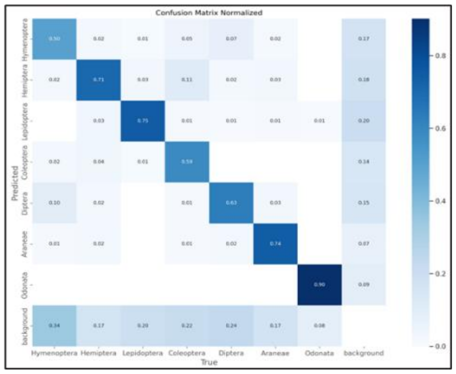


Figure 6: Performance metrics for the Yolov8 model. Validation scores (left) and confusion matrix for multi-class image classification (right).

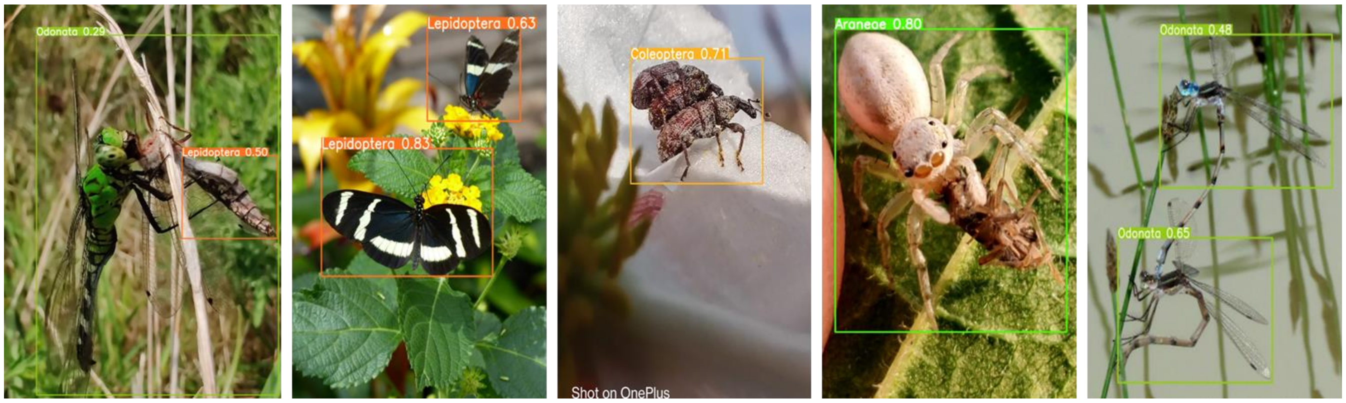


Figure 7: Results of multi-object detection and multi-class classification with the Yolov8 model.

CONCLUSION & FUTURE WORK

Our project aimed to address the challenging task of insect classification through the development and evaluation of image classification algorithms. We utilized two main datasets, ArTaxOr and Mobile Corpus, to train and test our models, considering the diversity in image sizes and insect species distribution.

Our initial attempts with custom CNN models, while efficient in terms of runtime, faced challenges of overfitting, limiting their validation accuracy to around 40%. Recognizing this limitation, we transitioned to pre-trained models, including DenseNet, MobileNet, and InceptionV3. These models exhibited improved performance, achieving higher accuracy, particularly with the Mobile Corpus. The choice of model and its modifications significantly impacted performance, with the Mobile Corpus consistently outperforming ArTaxOr.

Notably, our custom InceptionV3 model, after modifications, emerged as a promising candidate, achieving high accuracy across multiple datasets. Confusion matrix analysis revealed nuanced class-level predictions, with certain insect orders consistently showing higher accuracies than others. The challenges associated with smaller image sizes in the ArTaxOr corpus were evident in lower overall test accuracy.

Our object detection and classification also gave us promising results. With accuracies similar to our better performing Inceptionv3 models, the Yolov8 model was able to successfully detect multiple objects in an image in instances where there was at least some separation between the insects in the image. In cases where insects were too close together, a single bounding box was generated around the most prominent features of the object, in the image. This is an area that we can work on to improve in our future work.

In summary, our exploration of insect classification algorithms highlights the complexity of the task and the importance of model selection and fine-tuning. Our analysis showed us that deeper models may not necessarily perform better than shallower models in all cases. The comparative analysis of datasets and models sheds light on the intricate interplay between image characteristics and algorithm performance. As we move forward, further refinement and optimization of the chosen model, considering the nuances of different insect orders, will be crucial for achieving better results. Our work contributes to the ongoing efforts in developing effective tools for insect identification, addressing the need for robust insect classification programs in ecological studies and beyond.

We classified images only up to insect taxonomic order so that each class would have more images available for training the models. While our current focus has been on taxonomic order-level classification, the potential for advancing our insect classification algorithm to operate at finer taxonomic levels, such as the family or species level [5], opens exciting avenues for future research. Progressing towards greater granularity in classification poses both opportunities and challenges.

Moving to family or species level classification would require a better understanding of how to elicit the nuances of common structural features that differentiate closely related insect groups and integrate this information into our feature extraction process [5]. Challenges arise due to the finer distinctions among species, often characterized by subtle variations in morphology and coloration. This transition presents exciting possibilities but comes with notable challenges. Fine-tuning models to capture subtle variations among closely related species, obtaining annotated datasets at finer taxonomic levels, addressing imbalances, and accommodating intra-species variability are crucial considerations. The increased computational complexity and the need for advanced techniques like transfer learning and ensemble methods further underscore the complexity of moving to finer taxonomic levels. Despite these challenges, advancing our algorithm holds promise for more precise insect identification in ecological studies, underscoring the dynamic nature of machine learning applications in biodiversity research.

References:

1. Yang Y, Zhang L, Du M, Bo J, Liu H, Ren L, Li X, Deen MJ. A comparative analysis of eleven neural networks architectures for small datasets of lung images of COVID-19 patients toward improved clinical decisions. Comput Biol Med. 2021 Dec;139:104887. doi: 10.1016/j.compbiomed.2021.104887. Epub 2021 Sep 24. PMID: 34688974; PMCID: PMC8461289.
2. Szegedy, Christian & Vanhoucke, Vincent & Ioffe, Sergey & Shlens, Jon & Wojna, ZB. (2016). Rethinking the Inception Architecture for Computer Vision. 10.1109/CVPR.2016.308.
3. ArTaxOr - https://www.kaggle.com/datasets/mistag/arthropod-taxonomy-orders-object-detection-dataset/data
4. Tomato Leaf Disease Classification - https://github.com/manibabukv/Deeplearningproject_code/tree/main
5. Hansen, OLP, Svenning, J-C, Olsen, K, et al. Species-level image classification with convolutional neural network enables insect identification from habitus images. Ecol Evol. 2020; 10: 737–747. https://doi.org/10.1002/ece3.5921
6. Sharma, Nabin & Baral, Sushish & Paing, May & Chawuthai, Rathachai. (2023). Parking Time Violation Tracking Using YOLOv8 and Tracking Algorithms. Sensors. 23. 5843. 10.3390/s23135843.
7. Roboflow platform - Roboflow: Give your software the power to see objects in images and video

Appendix I - TEAM CONTRIBUTIONS

Siddhi Jadhav - Object Detection (Yolov8 model implementation) and Classification modeling, Presentation, Report Writing, Initial Research

Nicole Koshy - Classification models (Custom CNNs, DenseNet, InceptionV3, Custom InceptionV3, MobileNet), Presentation, Report Writing, Initial Research

Tina Joseph - Classification models (Custom CNNs, ResNet, Initial MobileNet model), Presentation, Initial Research

Appendix II – One Drive Folder containing all project related files (Jupyter Notebooks, model files, Results)

Group 2 - Deep Learning Project - Insect Classification - https://indiana-my.sharepoint.com/:f:/r/personal/nickoshy_iu_edu/Documents/Group%202%20-%20Deep%20Learning%20Project%20-%20insect%20Classification?csf=1&web=1&e=vazBKv