Final Report: Dino Image

STAT 495: Data Science Consulting

Dr. Mena Whalen

Adrian Reuschel, Leah Boger, Kevin Rabinak

10 December 2025

Abstract

This project aims to assist Dr. Whitney's lab in their hands-on-process of differentiating fossils from rocks visually. The long term goal for the lab is to create a detection model. Based on available data, this project explores different classification models that could eventually be fed into a detection model. In determining the most promising classification model type, thorough image preprocessing was done and then three popular classification models were trained and compared using accuracy and fossil recall metrics.

Project Introduction

This project aims to assist Dr. Whitney's lab in their process of identifying fossils in a mix of fossils and rocks. Currently this process is completed by-hand in The Whitney Lab. Lab members go through piles of rocks and fossils (all about the size of a dime or smaller) and visually decide if they are fossils. Our goal was to find a way to leverage statistical modelling techniques in order to classify images so the process could be completed more efficiently.

In our initial meeting with Dr. Whitney, we were able to narrow down deliverables and what the end-goal of the longer-term project would be. Eventually, The Whitney Lab hopes to employ a detection model in conjunction with a current machine they can use to separate rocks and fossils. The aim of the detection model would be to be able to view a flat image of a mix of both rocks and fossils and identify which are fossils. In this meeting, we also discussed what data the lab already has and what would be reasonable to collect. The lab already had a couple hundred images of individual fossils, and informed us they could take a few more hundred images. With time constraints and the desire to have as much data as possible, we felt it would be prudent to use these images the lab already had. In addition, most detection models are built on classifier models, so we felt that beginning with a classifier model made the most sense. For these reasons, we decided to make our end goal creating a classification model that is able to classify an image of a single object as either rock or fossil. To achieve this, we asked for the lab to provide a couple hundred images of rocks that they go through in similar formatting to the fossil images that they already had. Our hope was to create a full classification model that performed with metrics we felt comfortable passing on to the lab. At the least, we agreed our deliverable would be detailed recommendations on further research and implementations based on what we could learn this semester.

To achieve our goal of finding an optimal classification model for the goal of fossil classification and eventual detection, we first completed detailed image preprocessing on the data we were provided as well as found additional images online to support our analysis. After image preprocessing, we tested three

popular image classification models (EfficientNet, ResNet50, and InceptionV3) to compare model metrics.

Data

We received fossil images from the lab around week 5 of the semester. The photos received came from three folders: "TBD Microfossils", "Completed Microfossils", and "Burke Microfossils". The "Completed Microfossils" folder had the cleanest images and most consistent format. There was a wide variety in resolution, background, aspect ratio, and miscellaneous objects. We used the following selection criteria to select images for our modeling dataset: white or black background, the image could be cropped to contain only the fossil, and no strict guidelines on resolution or distance from fossil as long as the fossil's features could be seen reasonably. Once the selection process was complete, we cropped each image to get rid of miscellaneous objects or image annotations.

We did not receive rock photos from the lab until around week 10, so we opted to use online rock images so we could begin model testing. The rock dataset we found comes from Kaggle, and contained 53 folders of different rock types. The rock types selected were those that would also be found in a Wyoming dig site. These rocks included: chert, sandstone, limestone, and conglomerate. Several other images were selected at random from the remaining rock folders to diversify the dataset further. To create the training, testing, and validation sets, we used Roboflow. Roboflow has a simple upload feature that allows you to upload a folder with two classes, select your desired preprocessing steps, and it will output the splits.

After receiving rock photos from The Whitney Lab, we combined these with the existing online rock sample. The big challenge with curating this dataset was balancing our need for a high sample size with the best practice of having balanced classes for model training. Ultimately, we decided to sacrifice the class balance for higher sample size. Computer vision models typically need 1,000+ images for a

robust model, and we were dealing with a couple hundred. Discussion on how this imbalance impacted model performance is discussed in the methods section. Our final dataset used in all three models had the following splits: training (189 fossils, 336 rocks), validation (16 fossils, 24 rocks), and testing (18 fossils, 36 rocks).

Methods

Preprocessing

The images from the lab that survived the initial filtering step had to be cropped before they could be fed into the classification models, as there was considerable variation in the features of the images external to the rock or fossil of interest. We wanted the models to learn differences in the features of the rocks and fossils themselves, rather than any differences in these external features. We cropped the images so that the rock or fossil of interest was approximately centered and constituted a sizable portion of the frame, and that all other objects were removed. This reduced the (potentially systematic) variation in the external features and made it more likely that the models would focus on features of the rocks and fossils themselves. However, differences in external features persisted, such as different colored backgrounds. If in one way or another these remaining differences were systematic between rock and fossil photos, the generalizability of the results presented in the following sections may be called into question. In hindsight, perhaps it would have been more sound to crop the photos to only include the rock or fossil of interest. Even then, non-generalizable differences might remain, in lighting, for instance.

EfficientNet

EfficientNet is a convolutional neural network that was released by Google in 2019 and is still widely used in computer vision modeling for classification tasks. Rather than just a singular model, EfficientNet contains variants B0-B7 that each increase the number of parameters used, the number of

layers in the neural network, and the input image size. These variants allow users to train models with varying inputs and complexities based on their needs. For this project, we tested B0, B1, B3, and B5 EfficientNet models.

During the middle stages of the semester, we were ready to test models, but only had fossil images from the lab. With the online rock images we sourced, we trained a B0 model on n=97 rock images and n=100 fossil images. The accuracy of this model was near 0.5, and we saw that the loss across training epochs was not decreasing. We concluded for the meantime that we needed higher sample sizes to test on, but did not rule out EfficientNet as a viable model.

Once we received rock images from the lab and increased the overall sample size, we tested B0, B1, B3, and B5 EfficientNet models to compare performance. The only hyperparameter tuned during the testing and comparison of these models was the sigmoid threshold value for classifying the images as rocks or fossils. With each increase in variant size, the sigmoid threshold inched closer back to its default state of 0.5, showing that the added parameters and layers of the neural network were finding more features in the images and creating new predictions closer to the cutoff we expect. The largest difference in the EfficientNet variants was the training time. The B0 model took only one minute to train, while the B5 model took at least 15 minutes on the exact same dataset. With our main recommendation being more data collection, we would not recommend future groups to train with B5 unless they needed the extra layers since the time to train will increase rapidly with more images.

After tuning each model, we received the same accuracy of 65.5% with fossil recall at 94.12%. This output is good in the sense that we want high fossil recall, but poor in the sense that its rock recall was 50%. The best use case of this model would be removing rocks from a large sample, resulting in a smaller sample containing a higher proportion of fossils than before. This isn't necessarily helpful for The Whitney Lab, as they can probably already do this. Despite EfficientNet's subpar performance in relation to our project goals, we still think it is a robust tool that should be considered in future work.

ResNet50

ResNet was released in 2015 by Microsoft as an image classification model that uses deep residual learning. The concept of ResNet is to create a convolution network model that does not have the usual overfitting and gradient loss issues that traditional CNN models face. ResNet50 accomplishes this by using something called “residual” or “skip” connections. When training the CNN, these connections allow more complex layers to leave output as is instead of continuing to refit if the residual is small enough. This allows the model to continue becoming more complex without overfitting.

In addition to ResNet50’s techniques to avoid overfitting, it also has pretrained weights that can be used as a base model for image classification. In our model we used Imagenet weights. These weights for the base model were trained using a dataset of a million images of over 1000 classes. The idea behind using pretrained weights is that your base model can already “see” in that it has been trained enough to identify features such as texture and edges.

In order to create the ResNet50 model, due to the size of the CNN and CPU needed to train the model, Google Colab was used so the model did not need to run locally. To train our ResNet50 model, we built a base model using the ImageNet weights and left the model without a final layer. After creating the base model, we trained the final layer using our rock and fossil training data. This data had been augmented using random zooming, rotating, and flipping. At this step, class weights were also specified with rocks getting a weight of 1 and fossils a weight of 3. This aided with the class imbalance in our dataset. Without the class weights, the model was achieving 78% accuracy, but just classifying every image as a rock. This would obviously be extremely unhelpful to the lab as they would still have to go through each image one-by-one. The final step of training was to unfreeze the last 5-10 layers of the base model and retrain them with the training data.

The final ResNet50 model had a fossil recall of 67% and accuracy of 30%. This model would be extremely unhelpful to the lab as it both misses fossils and has low overall accuracy. For future work, this model can be ruled out as both other models performed with far more success.

InceptionV3

Inception is a series of convolutional neural networks designed for computer vision, developed by researchers at Google. Version 3 was introduced in 2016. We implemented it in Python using the Keras API within TensorFlow. The model has over 20 million parameters, but as is the case for the parameters of ResNet50, they come pre-trained on the Imagenet database; we retrained only a tiny fraction of them (the "head" of the model). Via a sigmoid activation function, the final layer of our model outputs a number strictly between 0 and 1 representing its confidence that a given image contains a rock, which is then converted to a binary classification according to a specified threshold value ("fossil" if the outputted number is less than the threshold; "rock" if it is greater than or equal to the threshold).

At the 0.5 threshold, InceptionV3 was able to achieve a 92% classification accuracy on the test set. This was a fairly balanced performance: it had an 89% accuracy among fossil images and a 94% accuracy among rock images. The accuracy among fossil images is especially important, considering the risk-reward matrix of the Whitney Lab. The successful detection of a fossil would likely mean more to them than the successful detection of a rock. The model classified fossil images with high accuracy without needing the threshold to be skewed and precision to be lost: 89% of the images classified as containing fossils actually contained fossils (this high precision, of course, was implied by the high accuracy among rock images).

Though these results are promising, they must be approached with caution. Because of the small size of our test set, there's a decent chance we may have just gotten lucky. Moreover, the model may be using non-generalizable external features (e.g., the color of the backgrounds) to distinguish between rock and fossil photos, rather than features of the rocks and fossils themselves. Lastly, even if InceptionV3 really does give a working classification model, it is not clear how readily this translates to a working detection model, the ultimate goal of the Whitney Lab. Further exploration of the results is due. In particular, it might be helpful to better understand why the three models' architectures yield such different results.

Future Work

Our classification models had mixed results. In particular, the InceptionV3 model shows promise, but there are serious questions surrounding whether its success is (a) generalizable and (b) translatable to a working detection model. Such a detection model was this project's (albeit dubious) end goal when it was first assigned to us in August. The semester has come and gone, and that goal is still so far out of reach.

The next step toward the goal's achievement is the verification of the soundness of the InceptionV3 model. This will likely require additional images from the lab. The code for the model is attached in the appendix and available on our team's GitHub repository. If the model holds, then the use of InceptionV3 for image detection, which it also has the capacity for, should be tested. This would require a batch of mixed images. Revisiting the other models is additionally worthwhile, since they may have untapped room for improvement, or at least be able to shed light on the seeming strength of the InceptionV3 model.

Appendix

Script to Build ResNet50 model

```
import os
import pathlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    GlobalAveragePooling2D,
    Dense,
    Flatten
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.utils import class_weight

from google.colab import drive
drive.mount('/content/drive')

train_directory = '/content/drive/MyDrive/Split_Data_Full/train'
val_directory = '/content/drive/MyDrive/Split_Data_Full/valid'
test_directory = '/content/drive/MyDrive/Split_Data_Full/test'

img_height = 128
img_width = 128
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
   train_directory,
   image_size=(img_height, img_width),
   batch_size=batch_size,
   label_mode='int'
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
   val_directory,
   image_size=(img_height, img_width),
   batch_size=batch_size,
   label_mode='int'
)
```

```
# Data augmentation

data_augmentation = tf.keras.Sequential([
   tf.keras.layers.RandomFlip("horizontal"),
   tf.keras.layers.RandomRotation(0.1),
   tf.keras.layers.RandomZoom(0.1),
])

train_ds = train_ds.map(lambda x, y: (data_augmentation(x/255.0),
y)).prefetch(tf.data.AUTOTUNE)
val_ds   = val_ds.map(lambda x, y: (x/255.0, y)).prefetch(tf.data.AUTOTUNE)


# Extract labels from dataset
train_labels = np.concatenate([y.numpy() for x, y in train_ds], axis=0)
class_weights = class_weight.compute_class_weight(
   'balanced',
   classes=np.unique(train_labels),
   y=train_labels
)
class_weights_dict = {0: 3.0, 1: 1.0}  # fossils 3x more important



# Build model

base_model = tf.keras.applications.ResNet50(
   include_top=False,
   weights='imagenet',
   input_shape=(img_height, img_width, 3)
)
base_model.trainable = False  # start with frozen base

model = Sequential([
   base_model,
   GlobalAveragePooling2D(),
   Dense(128, activation='relu'),
   Dense(2, activation='softmax')
])

model.compile(
   optimizer=Adam(),
   loss='sparse_categorical_crossentropy',
   metrics=['accuracy']
)
```

```
early_stopping = EarlyStopping(
   monitor='val_loss',
   patience=3,
   restore_best_weights=True
)


# Train frozen base first

history = model.fit(
   train_ds,
   validation_data=val_ds,
   epochs=10,
   class_weight={0: 3.0, 1: 1.0},
   callbacks=[early_stopping]
)


# Fine-tune last layers of ResNet50
base_model.trainable = True
# freeze first layers to avoid overfitting
for layer in base_model.layers[:-50]:
   layer.trainable = False

model.compile(
   optimizer=Adam(1e-5),  # smaller learning rate for fine-tuning
   loss='sparse_categorical_crossentropy',
   metrics=['accuracy']
)

history_finetune = model.fit(
   train_ds,
   validation_data=val_ds,
   epochs=5-10,
   class_weight=class_weights_dict,
   callbacks=[early_stopping]
)

test_ds  = test_ds.map(lambda x, y: (x/255.0, y)).prefetch(tf.data.AUTOTUNE)
y_true = np.concatenate([y.numpy() for x, y in test_ds], axis=0)
y_pred_probs = model.predict(test_ds)
y_pred = np.argmax(y_pred_probs, axis=1)
```

```
# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(cm)

# Classification report
cr = classification_report(y_true, y_pred, target_names=['fossil', 'rock'])
print("Classification Report:")
print(cr)
```

Script to build InceptionV3 model

```
# Imports
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
import os

# Training, Validation, and Test Sets
train_dir = "Dino-Classify/fossil-classifier/data/processed/Split_Data_Full/train"
val_dir = "Dino-Classify/fossil-classifier/data/processed/Split_Data_Full/valid"
test_dir = "Dino-Classify/fossil-classifier/data/processed/Split_Data_Full/test"

# Pre-Processing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
```

```
    target_size=(299, 299),
    batch_size=32,
    class_mode='binary'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(299, 299),
    batch_size=32,
    class_mode='binary'
)

test_generator = val_datagen.flow_from_directory(
    test_dir,
    target_size=(299, 299),
    batch_size=54,
    class_mode='binary',
    shuffle=False
)

# Inception Model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

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

# Defining Recall (Thanks, Adrian)
def fossil_recall(y_true, y_pred):
    y_pred_bin = tf.cast(y_pred > 0.65, tf.float32)

    # fossils, positive class = 0
    true_positives = tf.reduce_sum(tf.cast((y_true == 0) & (y_pred_bin == 0), tf.float32))
    false_negatives = tf.reduce_sum(tf.cast((y_true == 0) & (y_pred_bin == 1), tf.float32))

    return true_positives / (true_positives + false_negatives + 1e-7)

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', fossil_recall])

# Training Model
```

```
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator
)

test_res = model.evaluate(test_generator)
print(test_res)

# Following Adrian's Lead
# Visualize Performance

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

# Plot Accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'b-', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot Loss
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'b-', label='Training Loss')
plt.plot(epochs, val_loss, 'r-', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# More Plots!
y_pred_probs = model.predict(test_generator, steps=len(test_generator)).ravel()
```

```
y_true = test_generator.classes  # these are already integers (0 or 1)

thresholds = np.linspace(0.2, 0.6, 41)
recalls = []
precisions = []

for t in thresholds:
    preds = (y_pred_probs > t).astype(int)

    TP = np.sum((y_true == 0) & (preds == 0))  # fossils predicted correctly
    FP = np.sum((y_true == 1) & (preds == 0))  # rocks predicted as fossils
    FN = np.sum((y_true == 0) & (preds == 1))  # fossils missed

    recall = TP / (TP + FN + 1e-7)
    precision = TP / (TP + FP + 1e-7)

    recalls.append(recall)
    precisions.append(precision)

plt.figure(figsize=(10, 6))
plt.plot(1-thresholds, recalls, label="Recall (Fossils)")
plt.plot(1-thresholds, precisions, label="Precision (Fossils)")
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title("Precision & Recall vs Threshold")
plt.legend()
plt.show()
```