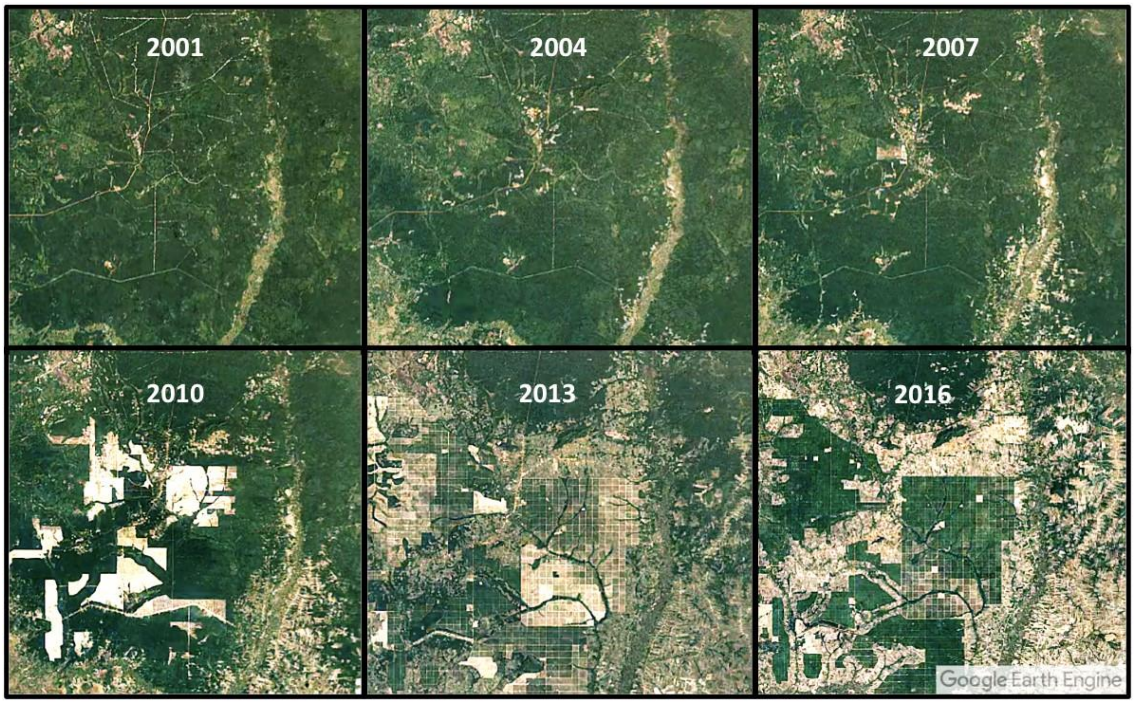
Elizabeth Hardwick, Capstone Project Final Report:

Forest Image Classification

with Convolutional Neural Networks



Synopsis

I created a model that classifies satellite images as either containing trees/forested or no trees/not-forested. I used convolutional neural networks with the Keras and TensorFlow libraries in Python. I tried scratch-built models and customized pre-trained transfer learning models using MobileNetV2 and ResNet50. In the end, the model with the best combination of accuracy, training time, and reliability proved to be the transfer learning model using MobileNetV2 with the top 14 layers fine-tuned; this model produced about 98% validation accuracy and about 97% accuracy on a smaller test set.

Problem and Context

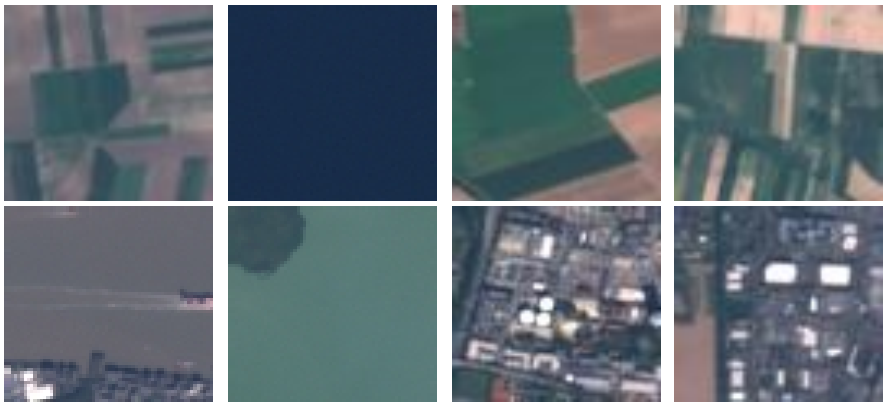
There are a number of real-world problems where this solution can be extremely useful. For instance:

- Classifying the same geographic area on a regular basis in a zone that is at risk for illegal timber harvesting or land clear-cutting to make way for farmland.
- Monitoring progress of an area where reforestation is underway following a wildfire.
- Estimating biomass extents where ground truth evidence is not available.
- Land use planning for commercial timber operations

Data Wrangling and Exploratory Data Analysis

The dataset consists of a balanced, labeled set of 10400 images total with 5200 images in each class (forested, non-forested). Each image is an RGB jpg of 64x64 pixels. The dataset is compiled and available on Kaggle. The images are sourced from the European Space Agency's Copernicus Sentinel 2 mission, which aims at monitoring variability in land surface conditions with a wide swath width (290 km) and high revisit time to support monitoring of changes in the Earth's surface. No additional wrangling or processing of the images was necessary. Exploration of the data included viewing a selection of the images from both classes to get an idea of what types of scenes were included. A selection is displayed below:

No Trees



Trees

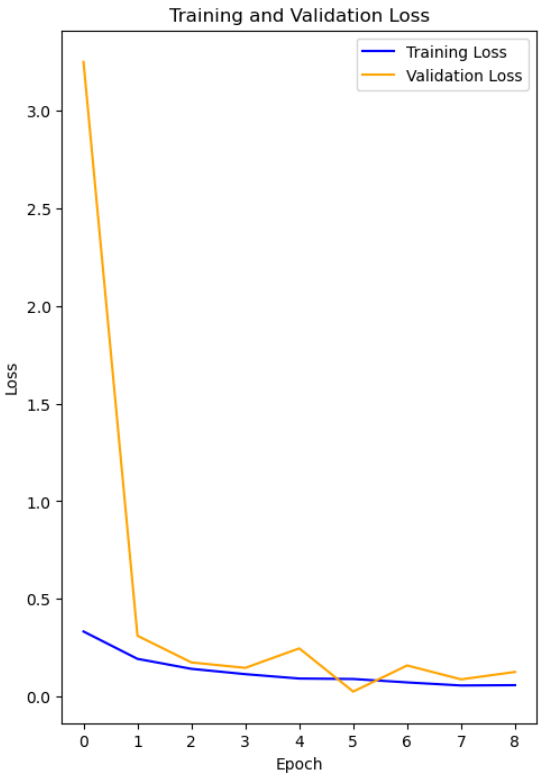
Modeling

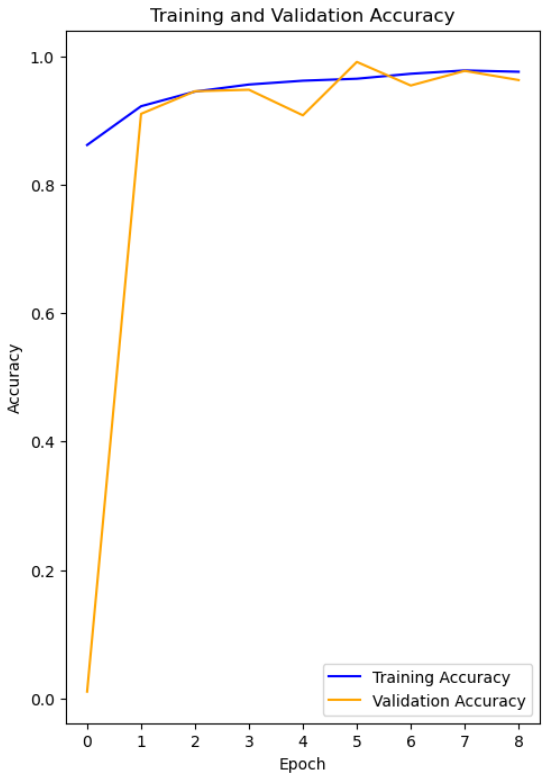
First, I explored scratch-built models with various architectures including with varying numbers of convolutional layers, numbers of neurons in the convolutional and dense layers, different activation functions, different optimizers, and different learning rates. I also tried all of the above with different sizes of validation sets including an 80/20 training/validation split which was my starting point, and then I repeated all of the architectures I had tried with that split again with a 75/25 split.

The best model with the 75/25 split had 16 neurons in the first convolutional layer, max pooling and batch normalization, 32 neurons in the second convolutional layer, max pooling and batch normalization, and 64 neurons in a dense layer, all with Relu activation, and with the Adam optimizer and a learning rate of 0.0007. This model resulted in about 96% validation accuracy and 0.1 validation loss:

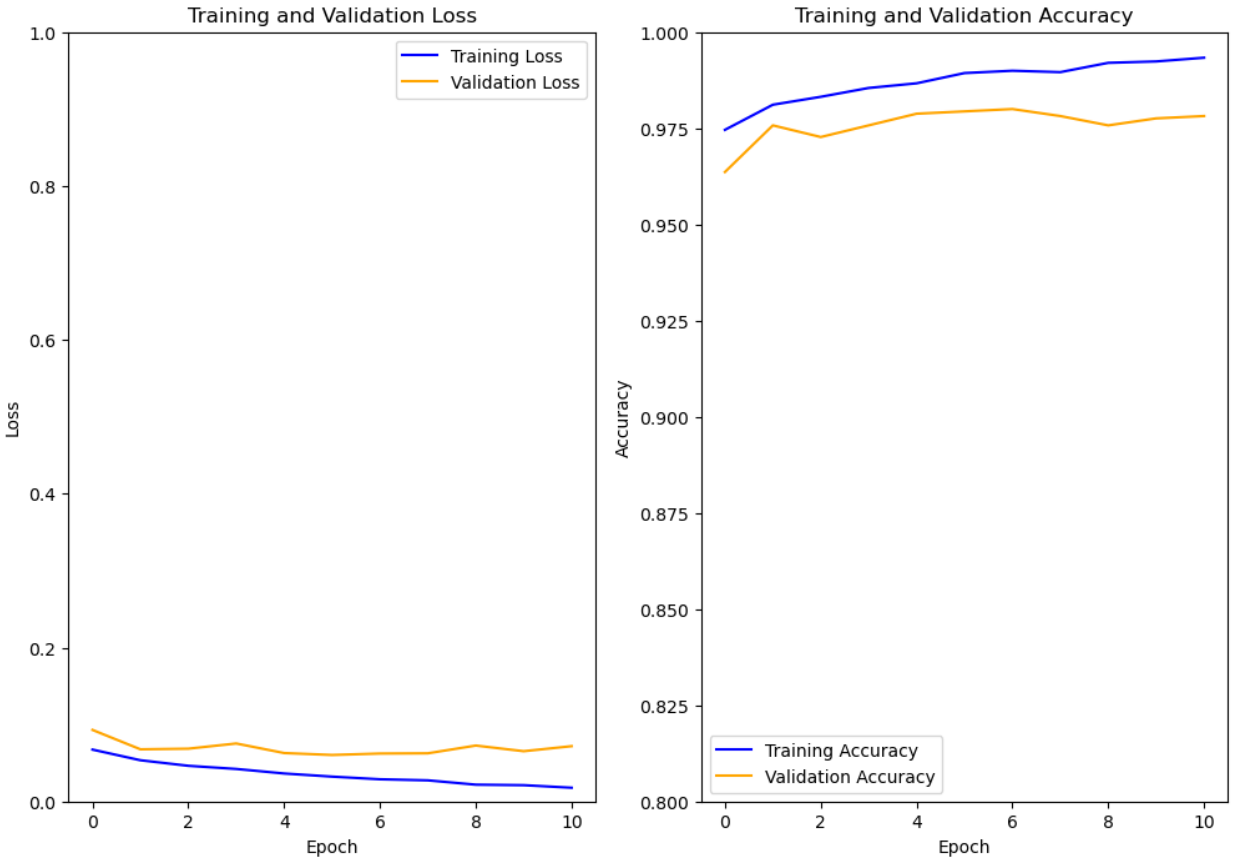
```
1 model_2.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 64, 64, 3)         0

 conv2d_2 (Conv2D)           (None, 64, 64, 16)        448

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 16)       0
 2D)

 batch_normalization_2 (Batc  (None, 32, 32, 16)       64
 hNormalization)

 conv2d_3 (Conv2D)           (None, 32, 32, 32)        4640

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 batch_normalization_3 (Batc  (None, 16, 16, 32)       128
 hNormalization)

 flatten_1 (Flatten)         (None, 8192)              0

 dense_2 (Dense)             (None, 64)                524352

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 529,697
Trainable params: 529,601
Non-trainable params: 96
_________________________________________________________________
```

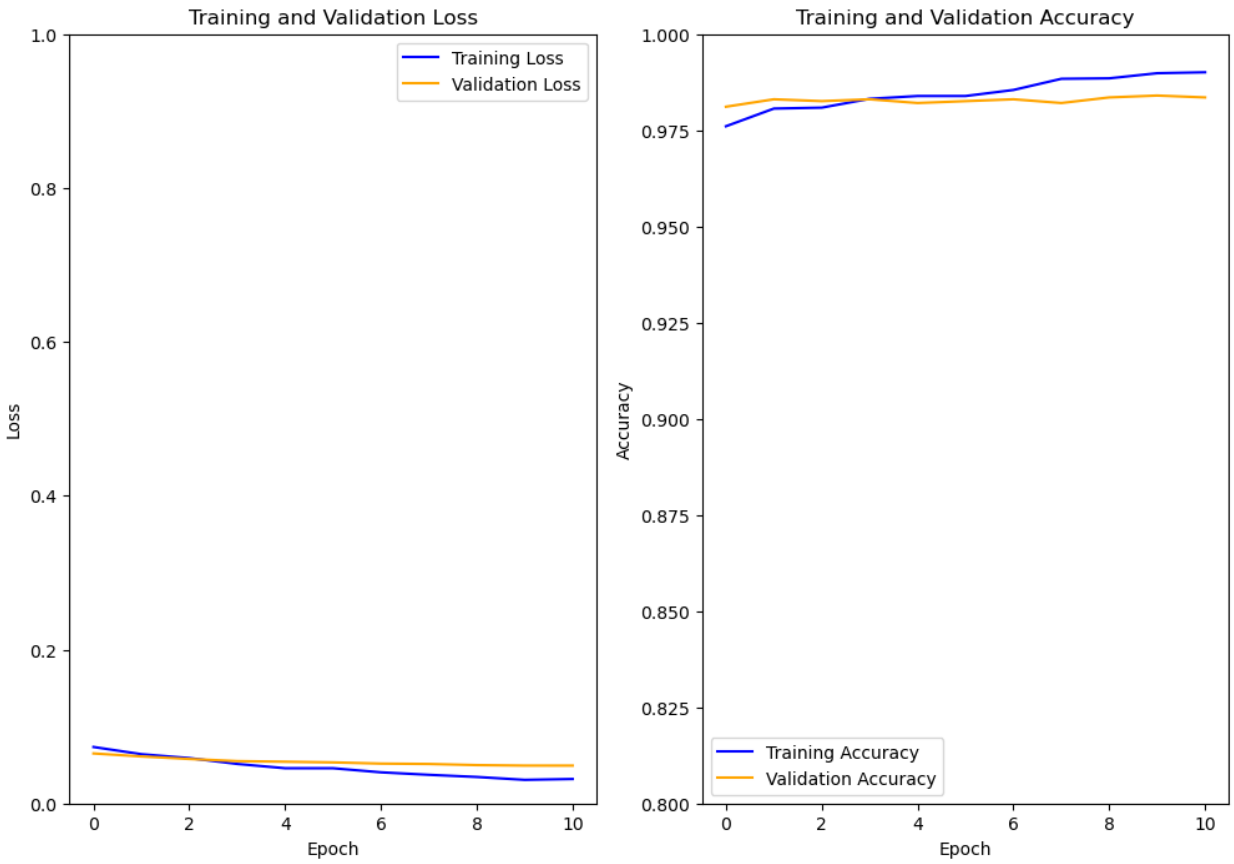




The two transfer learning options that I tried out start with the pretrained model (MobileNetV2 and ResNet50) and then I added a dense layer on top that serves as a feature extraction model, with the base model layer weights remaining frozen. I trained the feature extraction models for 10 epochs. Next I unfroze a few of the top layers of the base model to allow the weights to be updated and fine-tuned to the training data. With the MobileNetV2 model I opted to unfreeze the top 14 layers, and with the ResNet50 model, I first tried training on the top 10 layers, but this took over 45 minutes so I tried just training on the top 4 layers (this took about 25 minutes). This resulted in validation accuracy of just under 98% and validation loss of about 0.07 for the MobileNetV2 model and just over 98% and 0.05 for the ResNet50 model.



MobileNetV2 Learning Curves Fine-Tuned Epochs



ResNet50 Learning Curves Fine-Tuned Epochs

Conclusions

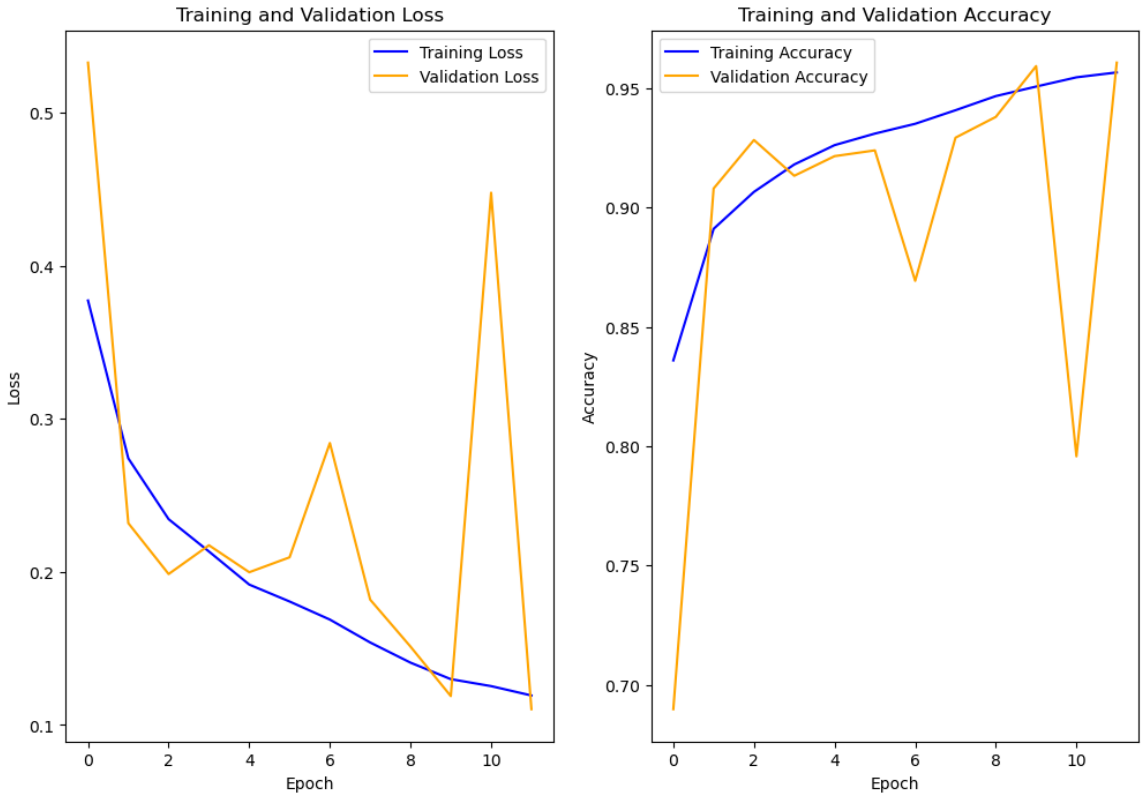
Take Home:

Both transfer learning models achieve good accuracy, around 98%, on validation and test sets. If the model will be updated with additional training data frequently, the better choice might be the MobileNetV2 model, as the training time is less than half that of the ResNet50 model.

Model	Convolutional Layers (# Neurons for Scratch-Built) Or Number of Layers in Base Model for Transfer Learning	Activation	Top Layers Unfrozen for Transfer Learning Models	Optimizer	Learning Rate	Approximate Validation Accuracy (%)	Approximate Validation Loss
Scratch-Built 80/20	2 (16,32)	Sigmoid	-	Adam	0.0007	96	0.1
Scratch-Built 75/25	2 (16,32)	Relu	-	Adam	0.0007	97	0.1
MobileNetV2 Transfer	154	-	14	RMSProp	0.00001	97.8	0.07
ResNet50 Transfer	175	-	4	RMSProp	0.00001	98.4	0.05

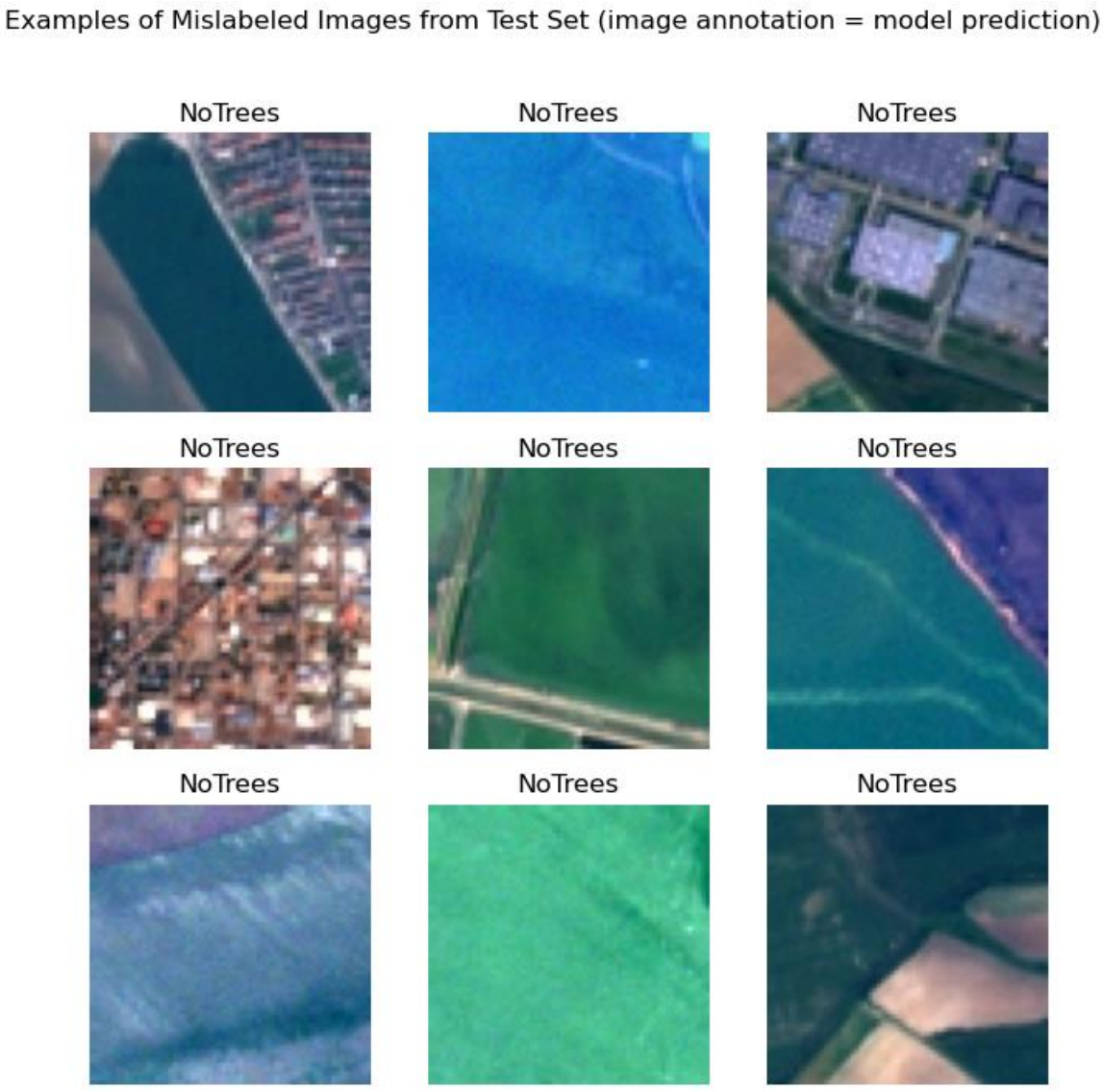
Reflections:

The learning curves of the scratch-built models tended to be very volatile in both loss and accuracy on the validation set (see example below), and generally not improving with additional epochs of training.



Some of the parameters I tried modifying to combat this including lower learning rates, increasing the number of neurons in the convolutional layers, adding an additional convolutional layer, changing the activation function, and increasing the validation set size from 20% to 25%. None of these did much to affect the volatility, but both of the transfer learning models had very smooth learning curves on all the various architectures tested, indicating that a deeper network was needed.

As a note of caution when working with Kaggle datasets, especially one of this nature in which I am relying on the labeling of a dataset for a classification model, some of the images that were 'misclassified' by the model seem to actually be mislabeled, ie images that actually do contain trees were placed in the NoTrees folder or vice versa. For example, below is a figure representing some images misclassified by the MobileNetV2 model, and the first image of the second row actually does not contain trees and should be in the NoTrees folder. The 'NoTrees' group contains images named 'AnnualCrop', 'Industrial', and 'SeaLake' while the 'Trees' group contains images named 'Forest', 'Pasture', and 'River'; the urban/industrial image in the first position of the second row is of type 'Industrial' and should be in the NoTrees group and thus would not be misclassified by the model.



Further Investigation:

This dataset consisted only of RGB images, but the Sentinel-2 satellite collects data in 13 different bands, including in the near infrared and short wave infrared spectral ranges. Certain pigments in plants strongly reflect certain wavelengths of near-infrared light (which is invisible to the human eye) and these bands are used in remote sensing for vegetation and forest mapping. It would be very interesting to obtain the additional band information from the same images and reperform the modeling with the additional information and compare results. Misclassification with bodies of water would probably not occur as frequently with near-infrared band data included.