Capstone Project II

Plane Vision

Final Report

Overview

The military has multiple uses for satellites. The most common missions are intelligence gathering, navigation, and military communications. When it comes to target recognition, enhancing the accuracy during combat environments would be ideal. These techniques will allow defense forces to understand potential operation areas by analyzing reports, documents, news feeds, and other forms of unstructured information. Additionally, AI in target recognition systems improves these systems' ability to identify the position of their

targets. Being able to track and identify potential threats would be beneficial to the military. The goal of this project is to detect an aircraft given a satellite image.

Problem Statement

How well can the model detect an aircraft given a satellite image?

Dataset

The image dataset will be from Kaggle. The data contains two classes, plane or no-plane. The plane class consists of 8,000 satellite images with planes. The No-plane class consists of 24,000 satellite images. A third of these are a random sampling of different landcover features - water, vegetation, bare earth, buildings, etc. - that do not include any portion of an airplane. The next third is "partial planes" containing a portion of an airplane, but not enough to meet the full definition of the "plane" class. The last third is "confusers" - chips with bright objects or strong linear features that resemble a plane.

Dataset: https://www.kaggle.com/rhammell/planesnet

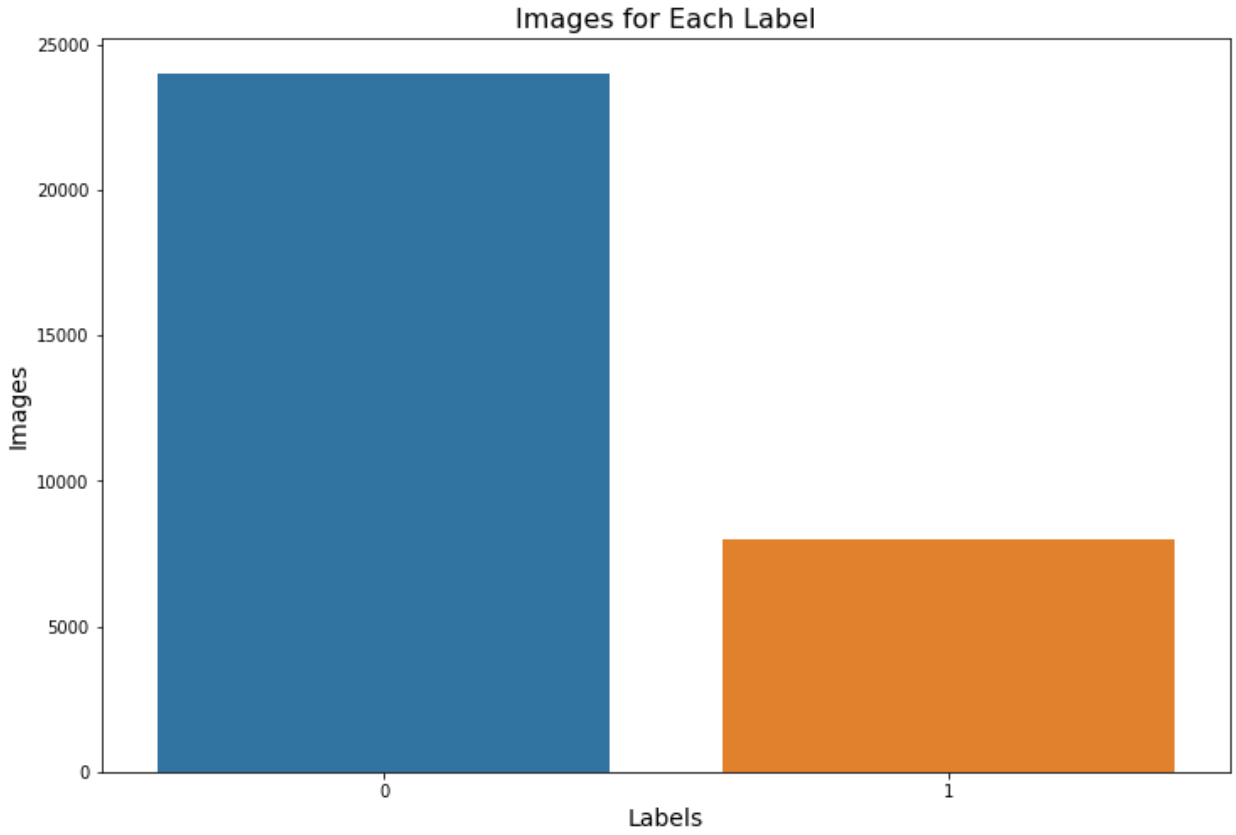
Data Wrangling

The data contained all 32000 images in one folder. I wanted to be able to used ImageDataGenerator, and in order to do that, the directory has to be set up a certain way. I created a train directory, test directory, and a validation directory where each folder contained a folder for plane and a folder for no plane. Here are a few images of what Plane and No Plane looks like.

Plane

No Plane

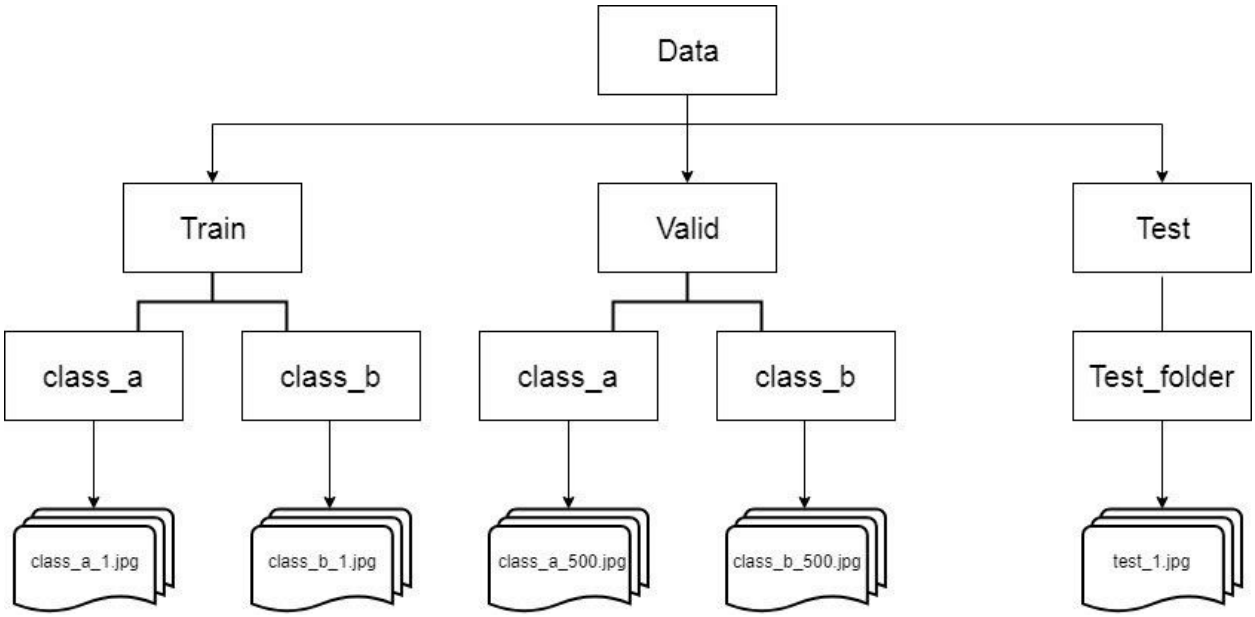
The graph below shows how many images there were for each label.



I loaded the images and created a dataframe. The dataframe contains only two columns, labels (0, 1) and image_path. The label 0 represents no-plane, and label 1 represents plane. I separated the dataframes into a no-plane dataframe and a plane dataframe. Then I dropped the labels columns in each dataframe to where only image_path was left. Then I created two filename lists from the dataframes, no-plane list, and plane list.

Next, it was time to copy the images into the three directories. Initially, I tried shutil.copy but it was taking too long to copy the images. Google collab would log out before finishing the images. I then discovered pyfastcopy. Pyfastcopy is a simple Python module that monkey patches the shutil.copyfile function of Python standard library to internally use the sendfile system call. It can provide important performance improvements for large file copy (typically 30-40%). Even with shutil.copyfile it took a long time trying to copy the files, so I settled with doing 1000 images each class.

The directory structure for a binary classification problem

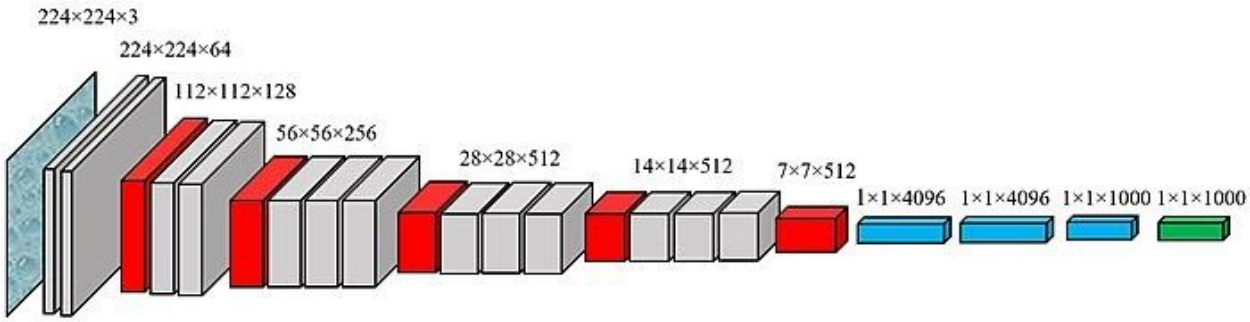


Transfer Learning

Transfer learning is a deep learning method for taking a model used for a task and reusing for another task. The most common ways to use transfer learning is to create a model or to use a pre-trained model. I chose to use a pre-trained model, specifically Oxford University's VGG16.

VGG Net is the name of a convolutional neural network model proposed by K. Simonyan and A. Zisserman. VGG Net has learned to extract the features that can distinguish the objects and classify unseen objects. VGG Net was invented to enhance classification accuracy by increasing the depth of the CNNs. VGG16 has 16 weighted layers and has been used for object recognition. VGG Net takes an input of 224×224 RGB images and passes them through a stack of convolutional layers with a fixed filter size of 3×3 and the stride of 1. There are five max-pooling filters embedded between convolutional layers to down-sample the input representation [1]. Three fully connected layers follow the stacked convolutional layers. The fully connected layers have 4096, 4096, and 1000 channels, respectively, and the last layer is a soft-max layer [2].

VGG Network Structure



After selecting my model, I will reuse the model. The pre-trained VGG16 model is used as the starting point for a model on the second task of interest, which will be classifying our satellite images of whether they are a plane or not. This may involve using all or parts of the model, depending on the modeling technique used. After reusing the model, you can choose to tune the model. The model may need to be adapted or refined on the input-output pair data available for the task of interest.

Exploratory Data Analysis (EDA)

Now it's time to use the pre-trained model. I initialized VGG16 with ImageNet as the weights.

```
from tensorflow.keras.applications import vgg16

# Init the VGG model
vgg_conv = vgg16.VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
```

In Keras, each layer has a parameter called "trainable". For freezing the weights of a particular layer, this sets the parameter to False, indicating that this layer should not be trained.

```
# Freeze all the layers
for layer in vgg_conv.layers[:]:
    layer.trainable = False

# Check the trainable status of the individual layers
for layer in vgg_conv.layers:
    print(layer, layer.trainable)
```

```
<tensorflow.python.keras.engine.input_layer.InputLayer object at 0x7fdb65799e80> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb32cde2b0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb32cde5c0> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7fdb32cde9e8> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb324436d8> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb3245d1d0> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7fdb3245d9b0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb20180160> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb20180f98> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb2018a438> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7fdb2018ac88> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb201927b8> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb2019d630> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb2019d6d8> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7fdb201a2630> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb201a2e10> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb201acc88> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7fdb201acbe0> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7fdb201b5978> False
```

So I created a new layer and added the layers I want to train.

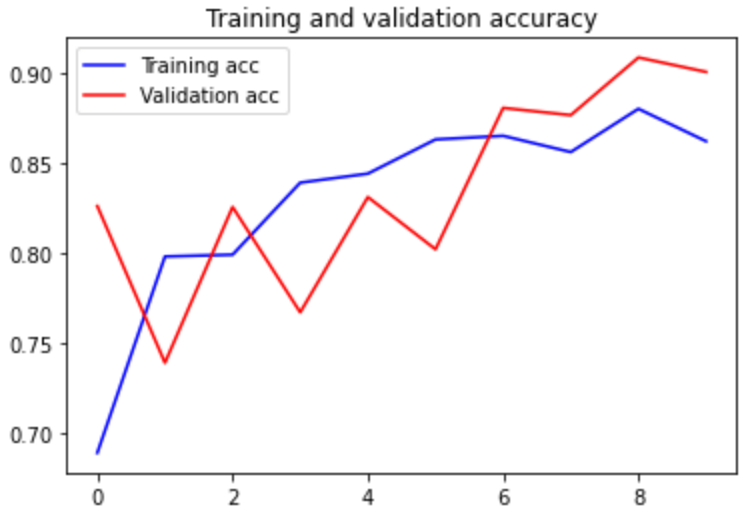
```
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from keras.utils import to_categorical
from sklearn.utils import shuffle

# Create the model
model = Sequential()

# Add the vgg convolutional base model
model.add(vgg_conv)

# Add new layers
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

# Show a summary of the model. Check the number of trainable parameters
model.summary()
```

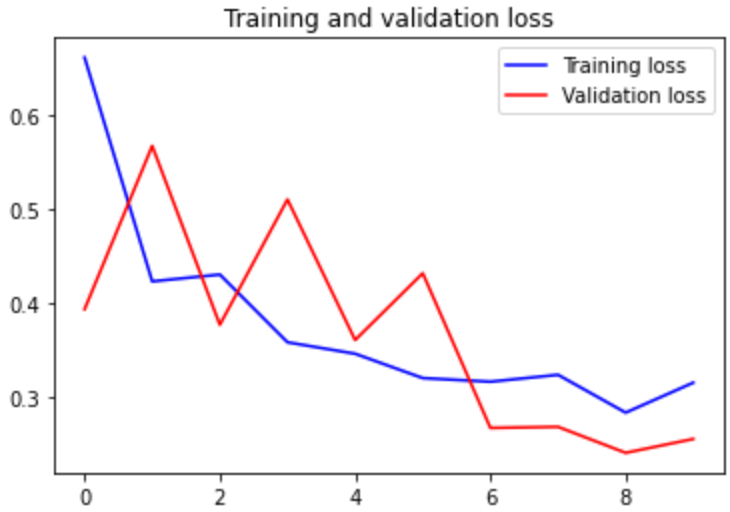
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 1, 1, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 1024)              525312
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 2050
=================================================================
Total params: 15,242,050
Trainable params: 527,362
Non-trainable params: 14,714,688
```

Next, I loaded the images using ImageDataGenerator. We already have the directory set up to be able to use ImageDataGenerator. This makes it easier for the machine to load the images and identify the two classes. After that, I train the model and check out the performance. The model performed well with 90% accuracy. The model made 198 out of 2000 errors.

```
==========================] - 5s 255ms/step - loss: 0.6605 - categorical_accuracy: 0.6895 - val_loss: 0.3937 - val_categorical_accuracy: 0.8265
==========================] - 5s 239ms/step - loss: 0.4232 - categorical_accuracy: 0.7985 - val_loss: 0.5668 - val_categorical_accuracy: 0.7395
==========================] - 5s 237ms/step - loss: 0.4304 - categorical_accuracy: 0.7995 - val_loss: 0.3772 - val_categorical_accuracy: 0.8260
==========================] - 5s 237ms/step - loss: 0.3586 - categorical_accuracy: 0.8395 - val_loss: 0.5100 - val_categorical_accuracy: 0.7675
==========================] - 5s 241ms/step - loss: 0.3465 - categorical_accuracy: 0.8445 - val_loss: 0.3611 - val_categorical_accuracy: 0.8315
==========================] - 5s 238ms/step - loss: 0.3207 - categorical_accuracy: 0.8635 - val_loss: 0.4318 - val_categorical_accuracy: 0.8025
==========================] - 5s 239ms/step - loss: 0.3169 - categorical_accuracy: 0.8655 - val_loss: 0.2679 - val_categorical_accuracy: 0.8810
==========================] - 5s 237ms/step - loss: 0.3242 - categorical_accuracy: 0.8565 - val_loss: 0.2690 - val_categorical_accuracy: 0.8770
==========================] - 5s 243ms/step - loss: 0.2841 - categorical_accuracy: 0.8805 - val_loss: 0.2415 - val_categorical_accuracy: 0.9090
==========================] - 5s 238ms/step - loss: 0.3159 - categorical_accuracy: 0.8625 - val_loss: 0.2562 - val_categorical_accuracy: 0.9010
```

Here are the Training and Validation loss and accuracy curves:





Conclusion

The workflow consists of collecting the satellite images, creating a train directory, test directory, and a validation directory where each folder contained a folder for plane and a folder for no plane. I recommend this model because it was able to detect whether an image was a plane or not accurately. This will greatly benefit the military and any other federal agencies. I could see it being used for autonomous planes where you use artificial intelligence to operate the planes. You would use the machine learning model to detect and target enemy planes. The only thing I would change about my project is trying to get to use all 32000 images. This may have increased the accuracy even more.

The overall purpose of this project was to be able to help the military with image recognition. The military could use the model to recognize targets when given a satellite image. There are already plans for a project that aims to modernize military satellites through software so they can be reprogrammed quickly -- for example, to take advantage of new artificial intelligence (AI) algorithms for threat detection.

Code

References

1. "Max-pooling / Pooling," 2018. [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling. [Accessed 2019 April 30].
2. ↑ "ImageNet: VGGNet, ResNet, Inception, and Xception with Keras," Pyimagesearch, 20 March 2017. [Online]. Available: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/. [Accessed 2019 April 30].