Ship Detection in Satellite Imagery

Sam Markelon and Tyler Tucker

Ship Detection

- Airbus Ship Detection Challenge
  - https://www.kaggle.com/c/airbus-ship-detection/overview
- Challenge to detect ships in satellite imagery to prevent piracy, monitor trading volume, estimate environmental impact, and more
- Binary classification problem
  - Ship present or ship not present
- Rich and large data set
  - 192556 unique labeled examples
  - 0.78 are “no ship” class, 0.22 are “ship” class

Approach

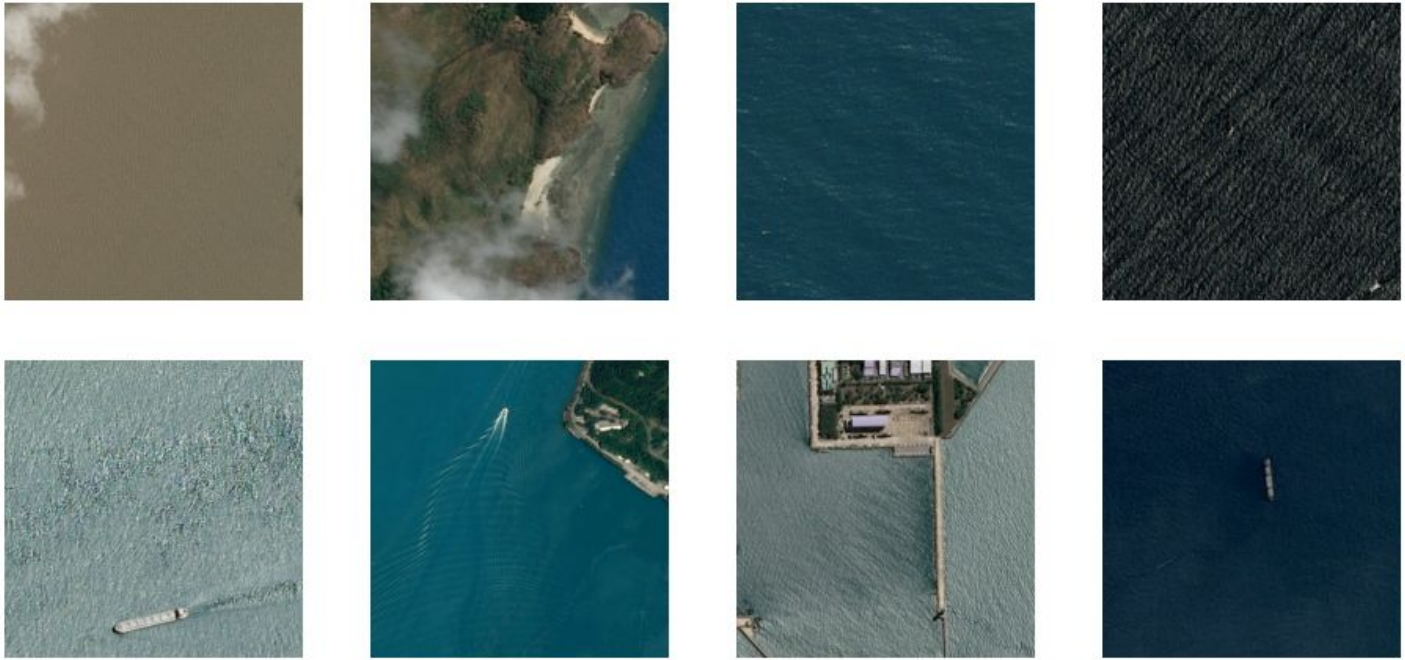
- Use a convolutional neural network (CNN) to classify the data set
  - CNN
  - CNN with dropout
  - Transfer learning on pretrained image classification model
  - Transfer learning with fine tunings
- Train on HiPerGator
  - Time constraints made it prohibitive to train on entire data set
  - Randomly sample 10000 examples for training set and 2000 for validation set
    - Representative of the entire data set
- Compare the approaches

Data Processing and Feature Engineering

- Process the labeled examples CSV to get format be a listing of the form `ImageID:classLabel`
  - process_data.py
- Sample and split the data set
- Rescale the data by 1. / 255
- Use the Keras ImageDataGenerator class to create data set generators
  - Resize the image to 150x150
  - x = ImageId, y = class
  - batch size = 20
  - Binary classification mode

	ImageId	ShipPresent
0	00003e153.jpg	0
1	0001124c7.jpg	0
2	000155de5.jpg	1
3	000194a2d.jpg	1
4	0001b1832.jpg	0
...	...	...
192551	fffedbb6b.jpg	0
192552	ffff2aa57.jpg	0
192553	ffff6e525.jpg	0
192554	ffffc50b4.jpg	0
192555	fffffe97f3.jpg	0

192556 rows × 2 columns



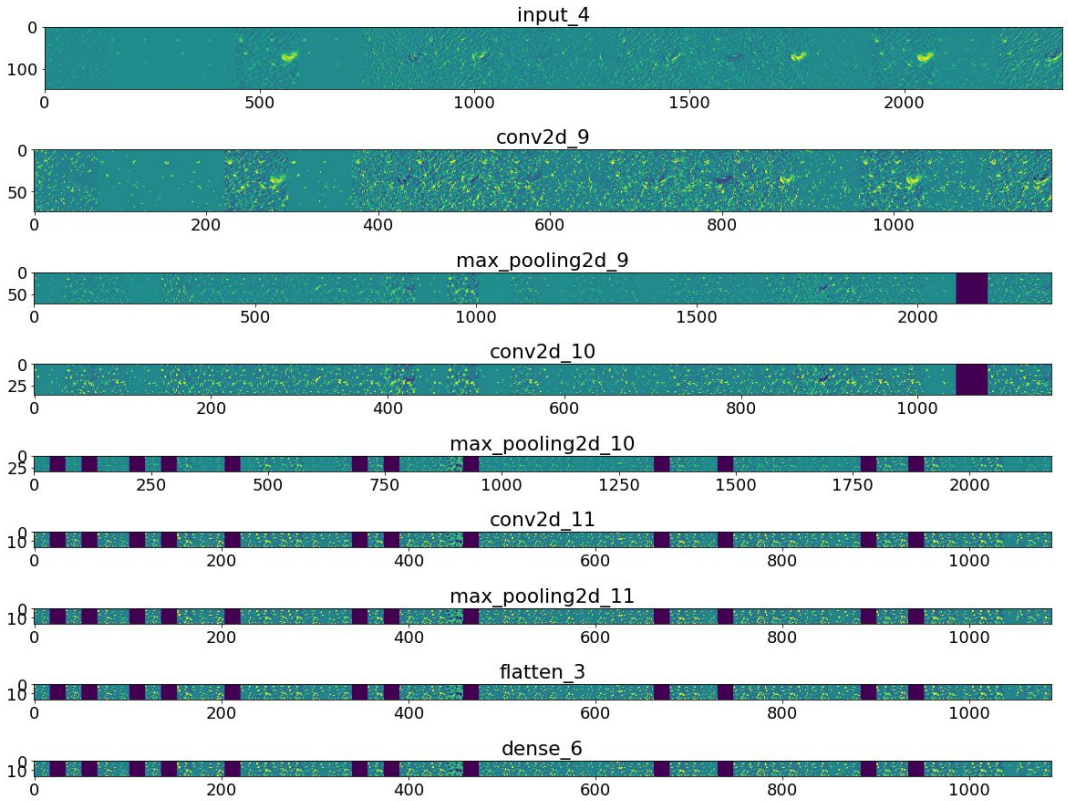
CNN Model

- Three convolution layers using relu activation
- Output layer used sigmoid activation function to encode probability of ship present as scalar in range [0,1].
- Train the model using binary cross-entropy as a loss function with a learning rate of 0.001 over 15 epochs with a batch size of 20.
- Steps per epoch determined by:

steps = ceil(data set size / batch size)

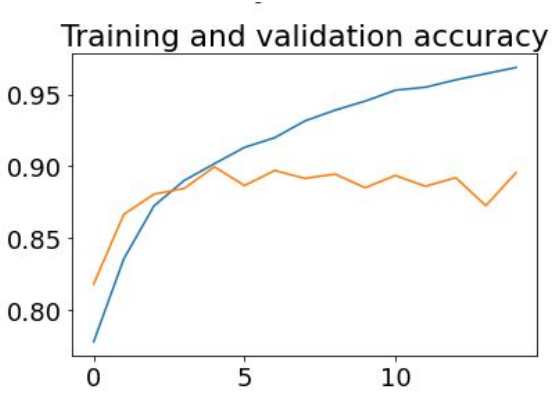
```
Model: "functional_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 72, 72, 32)        4640
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 17, 17, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               9470464
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 513
=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
```

Intermediate Image Representation by Layer



CNN Results

- Measure model using accuracy metric
- Training accuracy = 0.9687
- Validation accuracy = 0.8955
- Good model, but indicative of overfitting



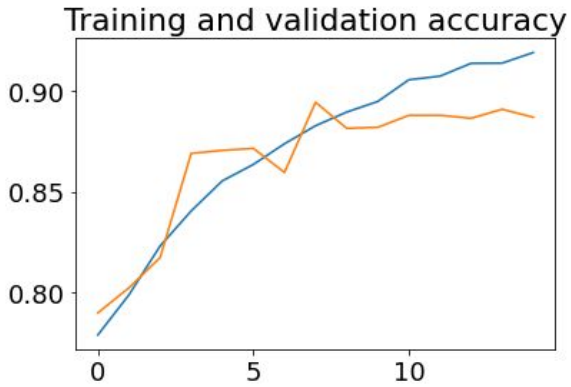
CNN with Dropout

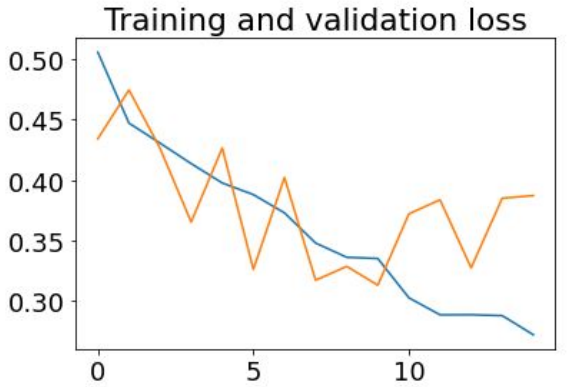
- Add a dropout layer with dropout rate of 0.5 before the output layer to combat overfitting
- Makes the problem artificially more difficult for the model
- Train the model using the same hyperparameters as before

```
Model: "functional_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 72, 72, 32)        4640
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 17, 17, 64)        0
_________________________________________________________________
flatten_7 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_14 (Dense)             (None, 512)               9470464
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 513
=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

CNN with Dropout Results

- Measure model using accuracy metric
- Training accuracy = 0.9193
- Validation accuracy = 0.8870
- Model is slightly less good than before, but we have lessened the amount of overfitting
- If we properly tune the dropout rate, we can likely achieve a model with similar accuracy that does not overfit



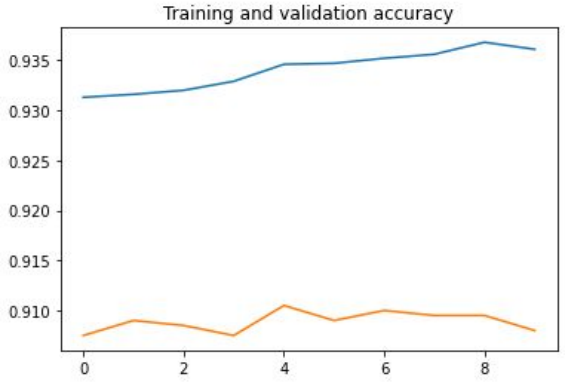


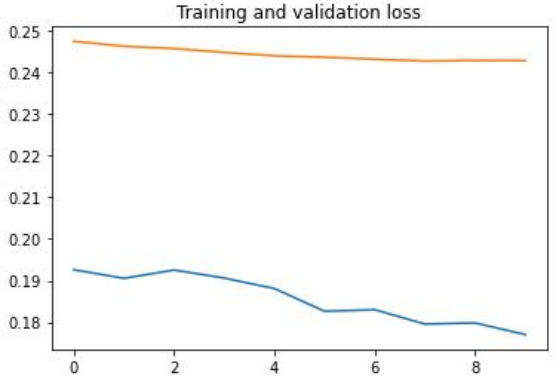
Transfer Learning

- Apply a model that is good at making predictions on a related but different dataset
- We use Google's Inception-v3 model trained on the ImageNet data set
- Freeze the model up to the mixed7 layer
    - last feature extraction layer
- Add a flatten layer, a dense layer, a drop our layer (dropout rate =0.2), and an output layer on top of the Inception-v3 layers
    - All feature extraction is done by Inception-v3 model
- Train the model with a learning rate of 0.0001 over 5 epochs
- **Achieve a training accuracy of 0.9126 and a validation accuracy of 0.9050**

Transfer Learning with Fine Tunings

- To see if we can achieve better results, we fine tune the weights of the pre-trained model's top layers on our data set
- We use stochastic gradient descent to optimize the weights using the binary cross-entropy loss function
- Trained using a learning rate of 0.00001, a momentum of 0.9, over 10 epochs.
- **Achieve a training accuracy of 0.9361 and a validation accuracy of 0.9080**
- Best model out of all the ones we tested
  - only slightly better than transfer learning without fine tunings
  - more overfit





Conclusion

- Performance on all models was similar
  - Transfer learning with fine tunings was the the the best with over 90% validation accuracy
  - Squeeze out slightly better performance with further hyperparameter tuning
  - Performed on par with the top submissions on Kaggle
    - Slightly different problem there
      - Had to identify pixels in the image where ship is present, while we did not do this due to time constraints
- Further work
  - Identify pixels where ship is present
  - Multiclass classification (small vessel, shipping vessel, etc.)
  - Explainable model