



**Introduction to AI
Project**

Title: Satellite Image Deep Learning

Submitted by:

Laraib Khalid (210201001)

Fibha Ayaz (210201006)

Batch:

BS CS 02 B

Semester:

Spring 2024

Submitted to:

Ma'am Reeda Saeed

Summary

```
import tensorflow as tf
tf.keras.backend.clear_session()
model.compile(optimizer="adam", loss=total_loss, metrics=metrics)
print(model.summary())
```

WARNING:tensorflow:From C:\Users\PMYLS\anaconda3\Lib\site-packages\keras\src\backend\common\global_state.py:82: The et_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 256, 256, 3)	0	-
conv2d (Conv2D)	(None, 256, 256, 16)	448	input_layer[0][0]
dropout (Dropout)	(None, 256, 256, 16)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 16)	2,320	dropout[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 32)	4,640	max_pooling2d[0][0]
dropout_1 (Dropout)	(None, 128, 128, 32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 32)	9,248	dropout_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 64)	18,496	max_pooling2d_1[0][0]
dropout_2 (Dropout)	(None, 64, 64, 64)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 64)	36,928	dropout_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 128)	73,856	max_pooling2d_2[0][0]
dropout_3 (Dropout)	(None, 32, 32, 128)	0	conv2d_6[0][0]

conv2d_7 (Conv2D)	(None, 32, 32, 128)	147,584	dropout_3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 256)	295,168	max_pooling2d_3[0][0]
dropout_4 (Dropout)	(None, 16, 16, 256)	0	conv2d_8[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 256)	590,080	dropout_4[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 32, 32, 128)	131,200	conv2d_9[0][0]
concatenate (Concatenate)	(None, 32, 32, 256)	0	conv2d_transpose[0][0], conv2d_7[0][0]
conv2d_10 (Conv2D)	(None, 32, 32, 128)	295,040	concatenate[0][0]
dropout_5 (Dropout)	(None, 32, 32, 128)	0	conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 32, 32, 128)	147,584	dropout_5[0][0]

conv2d_transpose_1 (Conv2DTranspose)	(None, 64, 64, 64)	32,832	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, 64, 64, 128)	0	conv2d_transpose_1[0][0], conv2d_5[0][0]
conv2d_12 (Conv2D)	(None, 64, 64, 64)	73,792	concatenate_1[0][0]
dropout_6 (Dropout)	(None, 64, 64, 64)	0	conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, 64, 64, 64)	36,928	dropout_6[0][0]
conv2d_transpose_2 (Conv2DTranspose)	(None, 128, 128, 32)	8,224	conv2d_13[0][0]
concatenate_2 (Concatenate)	(None, 128, 128, 64)	0	conv2d_transpose_2[0][0], conv2d_3[0][0]
conv2d_14 (Conv2D)	(None, 128, 128, 32)	18,464	concatenate_2[0][0]
dropout_7 (Dropout)	(None, 128, 128, 32)	0	conv2d_14[0][0]
conv2d_15 (Conv2D)	(None, 128, 128, 32)	9,248	dropout_7[0][0]

conv2d_15 (Conv2D)	(None, 128, 128, 32)	9,248	dropout_7[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 256, 256, 16)	2,064	conv2d_15[0][0]
concatenate_3 (Concatenate)	(None, 256, 256, 32)	0	conv2d_transpose_3[0][0], conv2d_1[0][0]
conv2d_16 (Conv2D)	(None, 256, 256, 16)	4,624	concatenate_3[0][0]
dropout_8 (Dropout)	(None, 256, 256, 16)	0	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 256, 256, 16)	2,320	dropout_8[0][0]
conv2d_18 (Conv2D)	(None, 256, 256, 6)	102	conv2d_17[0][0]

Total params: 1,941,190 (7.41 MB)

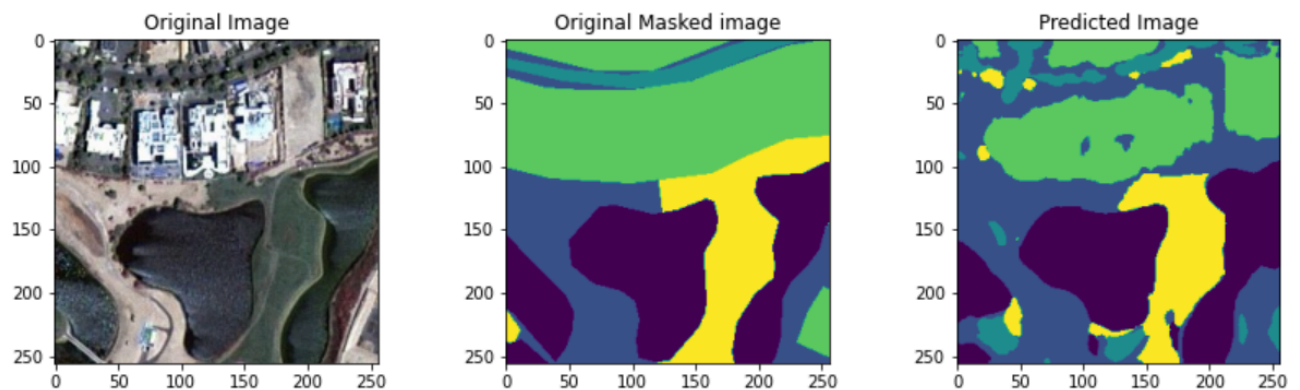
Trainable params: 1,941,190 (7.41 MB)

Non-trainable params: 0 (0.00 B)

Comparison of Predicted and Actual

```
: plt.figure(figsize=(14,8))
plt.subplot(231)
plt.title("Original Image")
plt.imshow(test_image)
plt.subplot(232)
plt.title("Original Masked image")
plt.imshow(ground_truth_image)
plt.subplot(233)
plt.title("Predicted Image")
plt.imshow(predicted_image)
```

```
: <matplotlib.image.AxesImage at 0x7fb9b42bb3d0>
```



Original Image:

This is the actual photograph or satellite image. It shows a real-world scene with buildings, water bodies, and other elements.

Original Masked Image:

This image is a labelled or segmented version of the original image. It has different colours representing different types of areas or objects (e.g., water, buildings, land, etc.). This image is used as a reference or "ground truth" to see what each part of the original image corresponds to.

Predicted Image:

This image is the result of a machine learning model or some algorithm trying to label or segment the original image, similar to the original masked image. The model tries to predict which areas of the original image correspond to different categories (the same categories as in the original masked image).

The goal of displaying these three images side-by-side is to visually compare the predicted image with the original masked image to see how well the model performed. Ideally, the predicted image should closely match the original masked image in terms of colours and patterns, indicating that the model correctly identified the different areas or objects in the original image.

Epochs

Methodology:

1. Initialization:

The model's parameters (weights and biases) are initialised, often randomly or using specific initialization techniques.

2. Forward Pass:

The training data is fed into the model. The model makes predictions based on its current parameters.

3. Loss Calculation:

The difference between the model's predictions and the actual targets (labels) is calculated using a loss function.

4. Backward Pass (Backpropagation):

The gradients of the loss with respect to the model's parameters are calculated. These gradients are used to update the model's parameters to minimise the loss.

5. Parameter Update:

The model's parameters are updated using an optimization algorithm (e.g., stochastic gradient descent, Adam).

Steps 2 to 5 are repeated for each epoch, meaning the entire training dataset is fed through the model once per epoch.




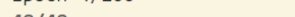
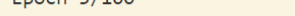
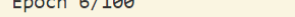
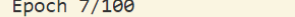
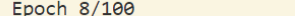
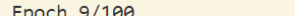

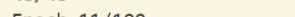

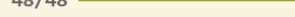
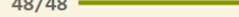
Benefits of Multiple Epochs:



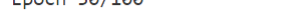
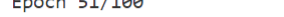




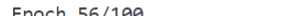







1. **Improved Accuracy:** With more epochs, the model has more opportunities to adjust its parameters, which can lead to better accuracy.
2. **Better Generalisation:** Multiple epochs help the model to generalise better to new, unseen data.

Potential Issues:





















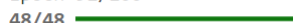


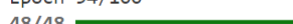
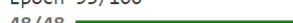
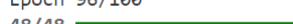
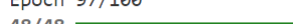
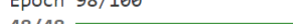
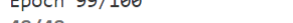
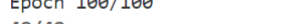

1. **Overfitting:** If the model is trained for too many epochs, it may overfit the training data, meaning it performs well on the training data but poorly on new, unseen data.
2. **Time-Consuming:** Training for many epochs can be time-consuming, especially for large datasets and complex models.

A total of 100 epochs were chosen for this project.

```
Epoch 1/100
48/48  101s 2s/step - accuracy: 0.4176 - loss: 1.4428 - val_accuracy: 0.6012 - val_loss: 1.1038
Epoch 2/100
48/48  93s 2s/step - accuracy: 0.6474 - loss: 1.0385 - val_accuracy: 0.6705 - val_loss: 0.9346
Epoch 3/100
48/48  93s 2s/step - accuracy: 0.6983 - loss: 0.8371 - val_accuracy: 0.6860 - val_loss: 0.8433
Epoch 4/100
48/48  93s 2s/step - accuracy: 0.7070 - loss: 0.8128 - val_accuracy: 0.7267 - val_loss: 0.7757
Epoch 5/100
48/48  92s 2s/step - accuracy: 0.7198 - loss: 0.7886 - val_accuracy: 0.7373 - val_loss: 0.7562
Epoch 6/100
48/48  112s 2s/step - accuracy: 0.7412 - loss: 0.7270 - val_accuracy: 0.7569 - val_loss: 0.7034
Epoch 7/100
48/48  99s 2s/step - accuracy: 0.7689 - loss: 0.6614 - val_accuracy: 0.7512 - val_loss: 0.7183
Epoch 8/100
48/48  100s 2s/step - accuracy: 0.7705 - loss: 0.6493 - val_accuracy: 0.7624 - val_loss: 0.6651
Epoch 9/100
48/48  97s 2s/step - accuracy: 0.7625 - loss: 0.6534 - val_accuracy: 0.7771 - val_loss: 0.6524
Epoch 10/100
48/48  97s 2s/step - accuracy: 0.7905 - loss: 0.5999 - val_accuracy: 0.7416 - val_loss: 0.6905
Epoch 11/100
48/48  114s 2s/step - accuracy: 0.7792 - loss: 0.6202 - val_accuracy: 0.7403 - val_loss: 0.7210
Epoch 12/100
48/48  122s 3s/step - accuracy: 0.7932 - loss: 0.5935 - val_accuracy: 0.7626 - val_loss: 0.6661
Epoch 13/100
48/48  124s 3s/step - accuracy: 0.8017 - loss: 0.5815 - val_accuracy: 0.7518 - val_loss: 0.7104
Epoch 14/100
36/48  26s 2s/step - accuracy: 0.8030 - loss: 0.5733
```

```
48/48  99s 2s/step - accuracy: 0.8819 - loss: 0.3436 - val_accuracy: 0.8495 - val_loss: 0.5297
Epoch 49/100
48/48  99s 2s/step - accuracy: 0.8827 - loss: 0.3440 - val_accuracy: 0.8490 - val_loss: 0.5137
Epoch 50/100
48/48  101s 2s/step - accuracy: 0.8907 - loss: 0.3132 - val_accuracy: 0.8588 - val_loss: 0.5034
Epoch 51/100
48/48  99s 2s/step - accuracy: 0.8886 - loss: 0.3221 - val_accuracy: 0.8587 - val_loss: 0.4932
Epoch 52/100
48/48  98s 2s/step - accuracy: 0.8862 - loss: 0.3298 - val_accuracy: 0.8349 - val_loss: 0.5512
Epoch 53/100
48/48  100s 2s/step - accuracy: 0.8876 - loss: 0.3282 - val_accuracy: 0.8454 - val_loss: 0.5736
Epoch 54/100
48/48  101s 2s/step - accuracy: 0.8954 - loss: 0.2998 - val_accuracy: 0.8445 - val_loss: 0.5460
Epoch 55/100
48/48  102s 2s/step - accuracy: 0.9017 - loss: 0.2887 - val_accuracy: 0.8578 - val_loss: 0.5156
Epoch 56/100
48/48  231s 5s/step - accuracy: 0.9011 - loss: 0.2853 - val_accuracy: 0.8486 - val_loss: 0.5406
Epoch 57/100
48/48  627s 13s/step - accuracy: 0.8996 - loss: 0.2961 - val_accuracy: 0.8443 - val_loss: 0.5317
Epoch 58/100
48/48  197s 4s/step - accuracy: 0.8938 - loss: 0.3131 - val_accuracy: 0.8440 - val_loss: 0.5540
Epoch 59/100
48/48  104s 2s/step - accuracy: 0.9026 - loss: 0.2801 - val_accuracy: 0.8412 - val_loss: 0.5887
Epoch 60/100
48/48  96s 2s/step - accuracy: 0.9099 - loss: 0.2622 - val_accuracy: 0.8473 - val_loss: 0.5596
Epoch 61/100
48/48  97s 2s/step - accuracy: 0.9084 - loss: 0.2638 - val_accuracy: 0.8621 - val_loss: 0.5166
Epoch 62/100
48/48  97s 2s/step - accuracy: 0.9155 - loss: 0.2423 - val_accuracy: 0.8592 - val_loss: 0.5277
Epoch 63/100
48/48  97s 2s/step - accuracy: 0.9106 - loss: 0.2553 - val_accuracy: 0.8460 - val_loss: 0.5792
Epoch 64/100
-
```

```

48/48  107s 2s/step - accuracy: 0.9289 - loss: 0.1991 - val_accuracy: 0.8684 - val_loss: 0.5188
Epoch 80/100
48/48  108s 2s/step - accuracy: 0.9303 - loss: 0.1956 - val_accuracy: 0.8679 - val_loss: 0.5573
Epoch 81/100
48/48  106s 2s/step - accuracy: 0.9331 - loss: 0.1853 - val_accuracy: 0.8456 - val_loss: 0.6622
Epoch 82/100
48/48  106s 2s/step - accuracy: 0.9370 - loss: 0.1756 - val_accuracy: 0.8582 - val_loss: 0.5794
Epoch 83/100
48/48  110s 2s/step - accuracy: 0.9238 - loss: 0.2165 - val_accuracy: 0.8565 - val_loss: 0.5753
Epoch 84/100
48/48  112s 2s/step - accuracy: 0.9366 - loss: 0.1799 - val_accuracy: 0.8711 - val_loss: 0.5433
Epoch 85/100
48/48  112s 2s/step - accuracy: 0.9326 - loss: 0.1851 - val_accuracy: 0.8549 - val_loss: 0.6622
Epoch 86/100
48/48  112s 2s/step - accuracy: 0.9371 - loss: 0.1770 - val_accuracy: 0.8636 - val_loss: 0.5096
Epoch 87/100
48/48  112s 2s/step - accuracy: 0.9165 - loss: 0.2416 - val_accuracy: 0.8679 - val_loss: 0.4958
Epoch 88/100
48/48  112s 2s/step - accuracy: 0.9153 - loss: 0.2530 - val_accuracy: 0.8539 - val_loss: 0.5365
Epoch 89/100
48/48  113s 2s/step - accuracy: 0.9285 - loss: 0.2085 - val_accuracy: 0.8595 - val_loss: 0.5428
Epoch 90/100
48/48  112s 2s/step - accuracy: 0.9316 - loss: 0.1899 - val_accuracy: 0.8640 - val_loss: 0.5382
Epoch 91/100
48/48  112s 2s/step - accuracy: 0.9373 - loss: 0.1726 - val_accuracy: 0.8704 - val_loss: 0.5607
Epoch 92/100
48/48  115s 2s/step - accuracy: 0.9417 - loss: 0.1578 - val_accuracy: 0.8697 - val_loss: 0.5687
Epoch 93/100
48/48  114s 2s/step - accuracy: 0.9440 - loss: 0.1521 - val_accuracy: 0.8695 - val_loss: 0.5548
Epoch 94/100
48/48  113s 2s/step - accuracy: 0.9436 - loss: 0.1535 - val_accuracy: 0.8708 - val_loss: 0.5609
Epoch 95/100
Epoch 87/100
48/48  112s 2s/step - accuracy: 0.9165 - loss: 0.2416 - val_accuracy: 0.8679 - val_loss: 0.4958
Epoch 88/100
48/48  112s 2s/step - accuracy: 0.9153 - loss: 0.2530 - val_accuracy: 0.8539 - val_loss: 0.5365
Epoch 89/100
48/48  113s 2s/step - accuracy: 0.9285 - loss: 0.2085 - val_accuracy: 0.8595 - val_loss: 0.5428
Epoch 90/100
48/48  112s 2s/step - accuracy: 0.9316 - loss: 0.1899 - val_accuracy: 0.8640 - val_loss: 0.5382
Epoch 91/100
48/48  112s 2s/step - accuracy: 0.9373 - loss: 0.1726 - val_accuracy: 0.8704 - val_loss: 0.5607
Epoch 92/100
48/48  115s 2s/step - accuracy: 0.9417 - loss: 0.1578 - val_accuracy: 0.8697 - val_loss: 0.5687
Epoch 93/100
48/48  114s 2s/step - accuracy: 0.9440 - loss: 0.1521 - val_accuracy: 0.8695 - val_loss: 0.5548
Epoch 94/100
48/48  113s 2s/step - accuracy: 0.9436 - loss: 0.1535 - val_accuracy: 0.8708 - val_loss: 0.5609
Epoch 95/100
48/48  100s 2s/step - accuracy: 0.9491 - loss: 0.1371 - val_accuracy: 0.8630 - val_loss: 0.6331
Epoch 96/100
48/48  101s 2s/step - accuracy: 0.9490 - loss: 0.1376 - val_accuracy: 0.8697 - val_loss: 0.5884
Epoch 97/100
48/48  96s 2s/step - accuracy: 0.9482 - loss: 0.1398 - val_accuracy: 0.8660 - val_loss: 0.6351
Epoch 98/100
48/48  108s 2s/step - accuracy: 0.9470 - loss: 0.1416 - val_accuracy: 0.8622 - val_loss: 0.6111
Epoch 99/100
48/48  97s 2s/step - accuracy: 0.9447 - loss: 0.1478 - val_accuracy: 0.8693 - val_loss: 0.6046
Epoch 100/100
48/48  97s 2s/step - accuracy: 0.9459 - loss: 0.1434 - val_accuracy: 0.8656 - val_loss: 0.6106
6/6  6s 945ms/step - accuracy: 0.8597 - loss: 0.6260

```

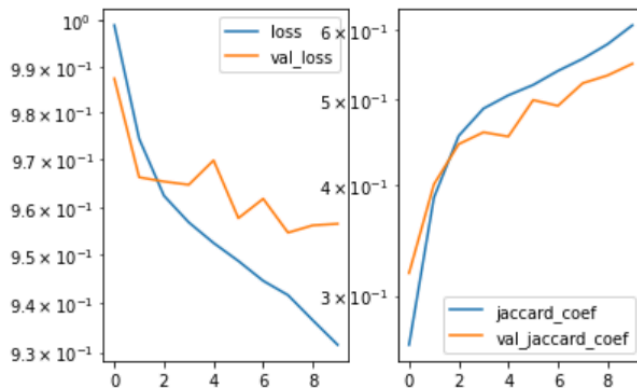
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

Visualisation

Visualisation Whilst Training

```
model_history = model.fit(X_train, y_train,
                           batch_size=16,
                           verbose=1,
                           epochs=10,
                           validation_data=(X_test, y_test),
                           callbacks=[plot_loss],
                           shuffle=False)
```

<Figure size 1008x576 with 0 Axes>



51/51 [=====] - 10s 191ms/step - loss: 0.9315 - accuracy: 0.8036 - jaccard_coef: 0.6075 - val_loss: 0.9565 - val_accuracy: 0.7427 - val_jaccard_coef: 0.5495

Left Plot

- **Training Loss (Blue Line):** Shows how the model's error on the training data decreases as it learns.
- **Validation Loss (Orange Line):** Shows how the model's error on the validation data decreases. This helps in understanding how well the model might perform on unseen data.
- **Observation:** Both losses decrease, indicating the model is learning. However, the validation loss fluctuates, which might suggest the model is starting to overfit (learning too much from the training data and not generalising well).

Right Plot

- **Training Jaccard Coefficient (Blue Line):** Measures the similarity between the predicted results and actual results for the training data. Higher is better.
- **Validation Jaccard Coefficient (Orange Line):** Measures the same for the validation data.
- **Observation:** Both coefficients increase, indicating improving performance. The gap between the lines suggests the model may be overfitting.

Visualisation for Training Data:

```
loss = history_a.history['loss']
val_loss = history_a.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label="Training Loss")
plt.plot(epochs, val_loss, 'r', label="Validation Loss")
plt.title("Training Vs Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



- **Training Loss (Yellow Line):** The error on the training data continues to decrease.
- **Validation Loss (Red Line):** The error on the validation data decreases initially but then stabilises, showing how well the model generalises to new data.
- **Observation:** The training loss keeps decreasing, but the validation loss stabilises, indicating the model might be overfitting after a certain point.

Visualisation for Training Data (Jaccard Coeff)

```
: jaccard_coef = history_a.history['jaccard_coef']
  val_jaccard_coef = history_a.history['val_jaccard_coef']

  epochs = range(1, len(jaccard_coef) + 1)
  plt.plot(epochs, jaccard_coef, 'y', label="Training IoU")
  plt.plot(epochs, val_jaccard_coef, 'r', label="Validation IoU")
  plt.title("Training Vs Validation IoU")
  plt.xlabel("Epochs")
  plt.ylabel("Loss")
  plt.legend()
  plt.show()
```

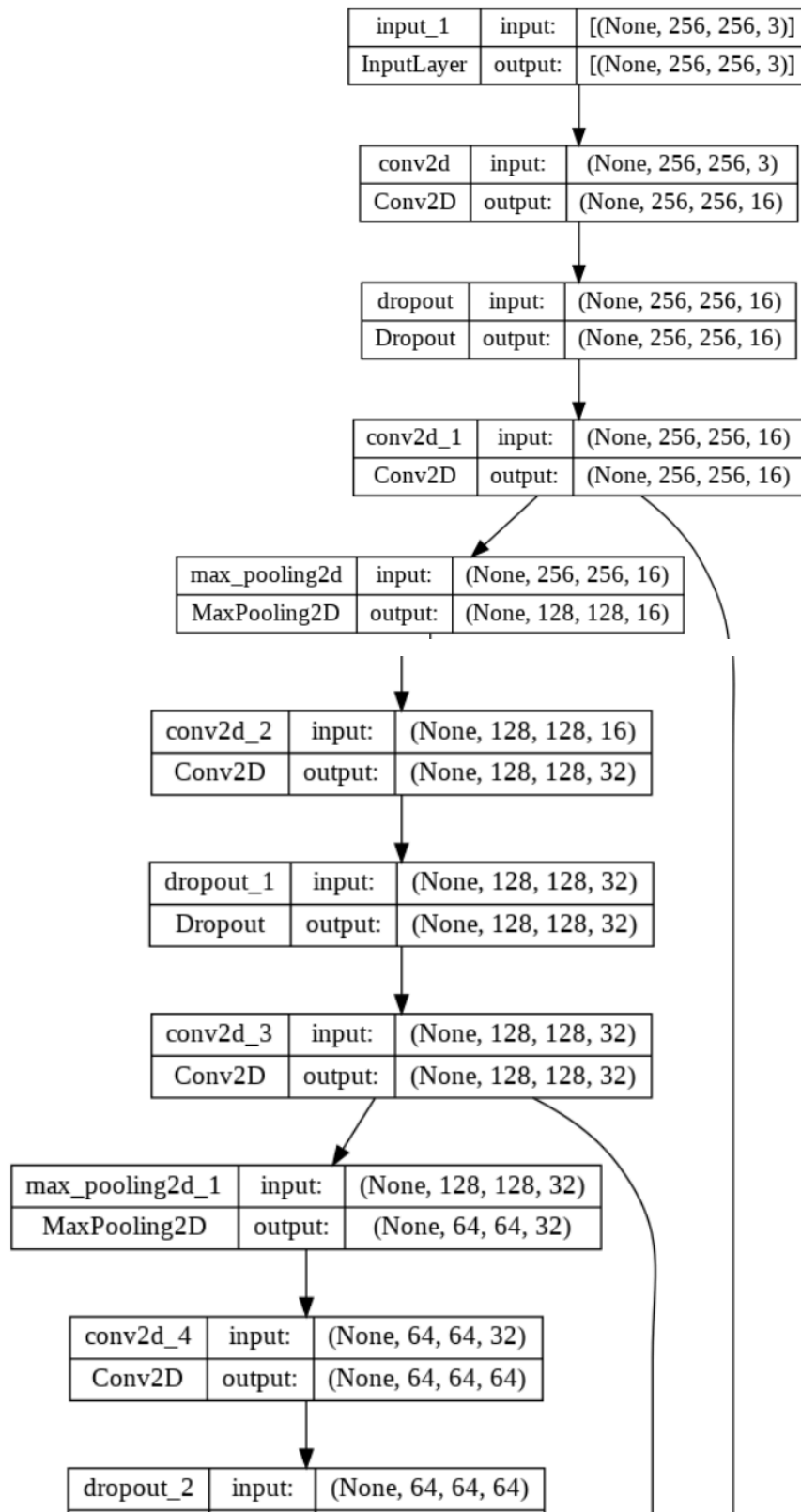


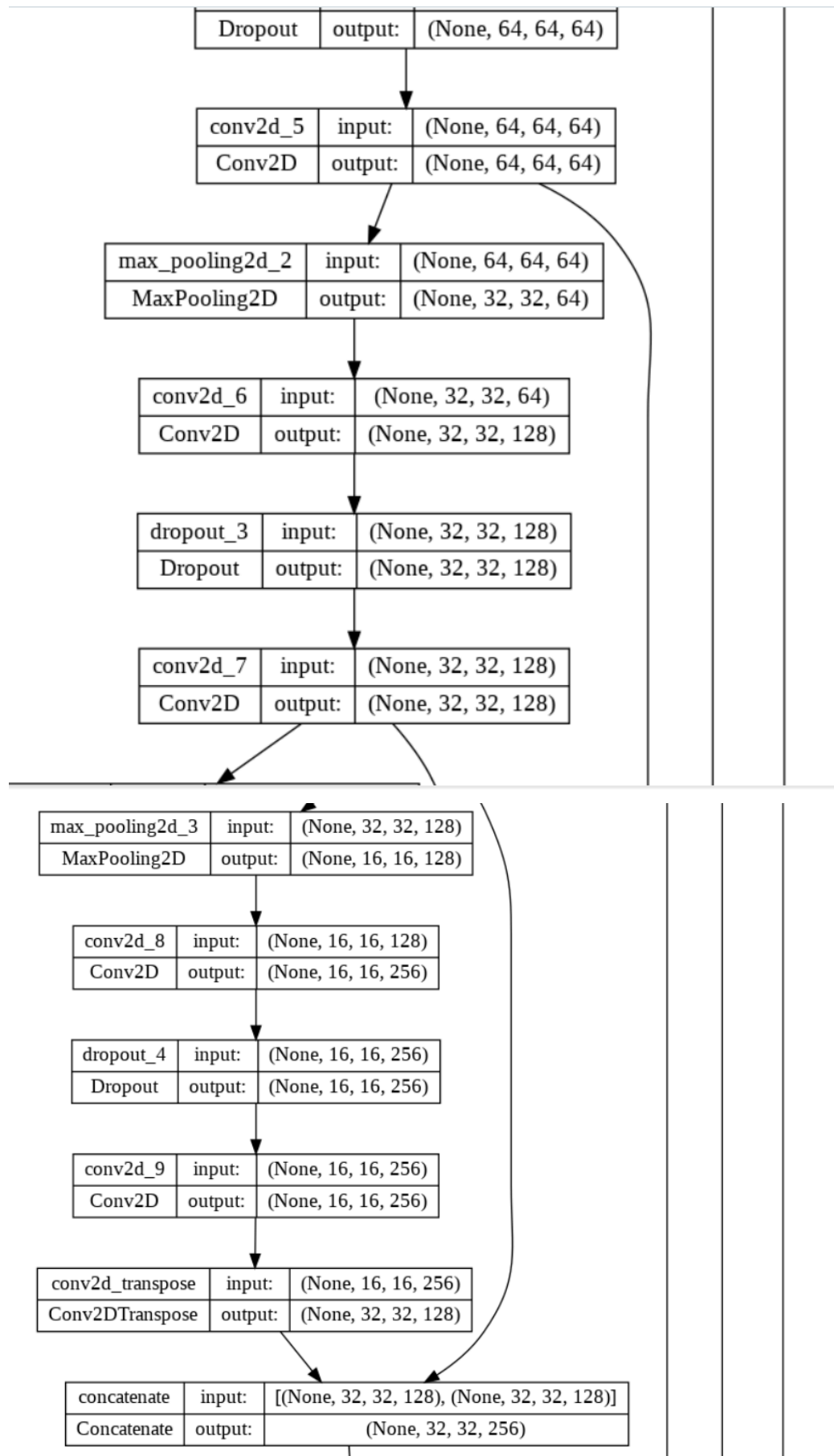
- **Training IoU (Yellow Line):** The similarity measure between predicted and actual results for the training data improves.
- **Validation IoU (Red Line):** The similarity measure for the validation data improves initially but then stabilises.
- **Observation:** The training IoU increases more consistently than the validation IoU, suggesting potential overfitting as the model learns the training data better than it generalises to new data.

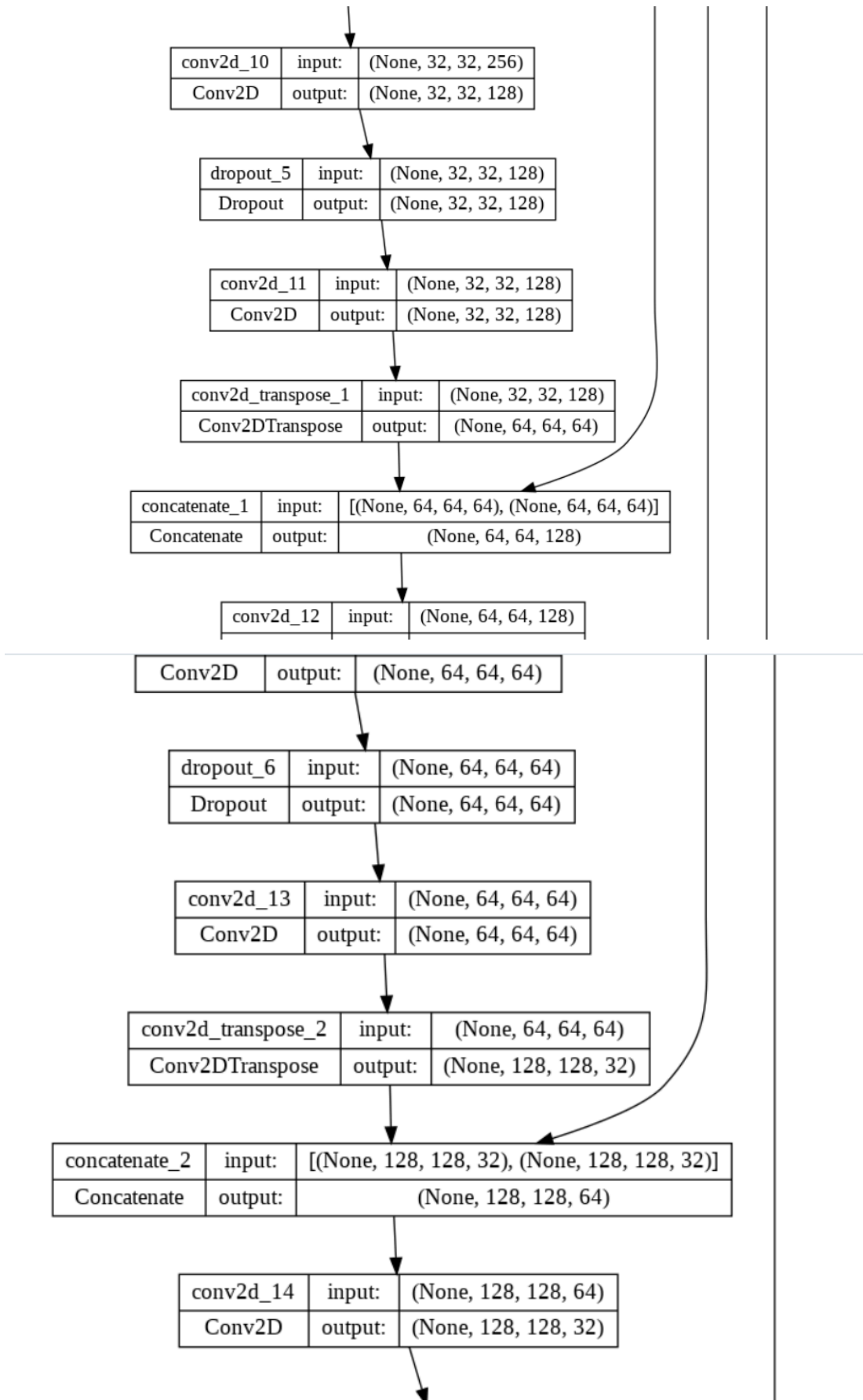
Visualization via Model:

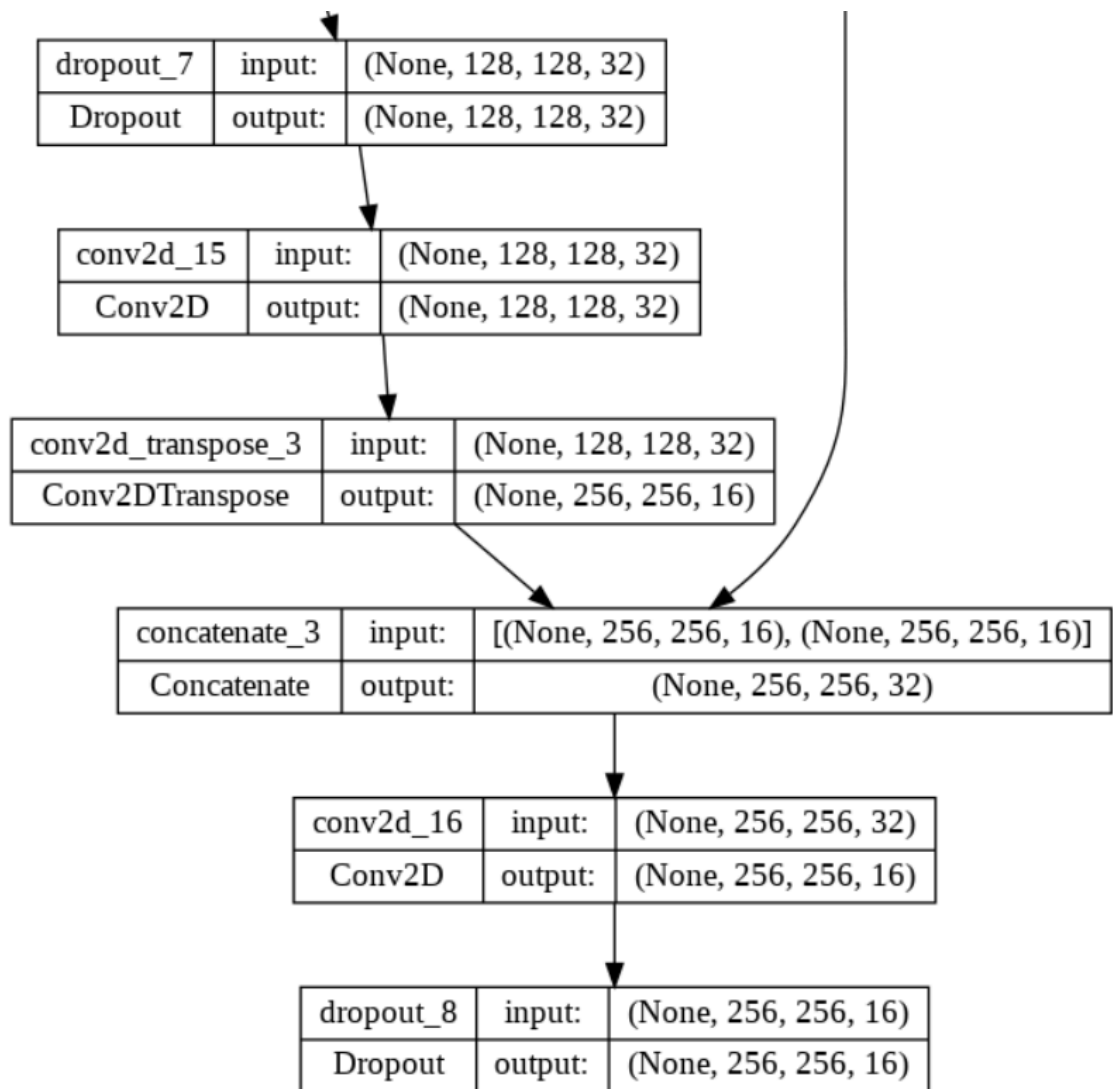
```
from keras.utils.vis_utils import plot_model
```

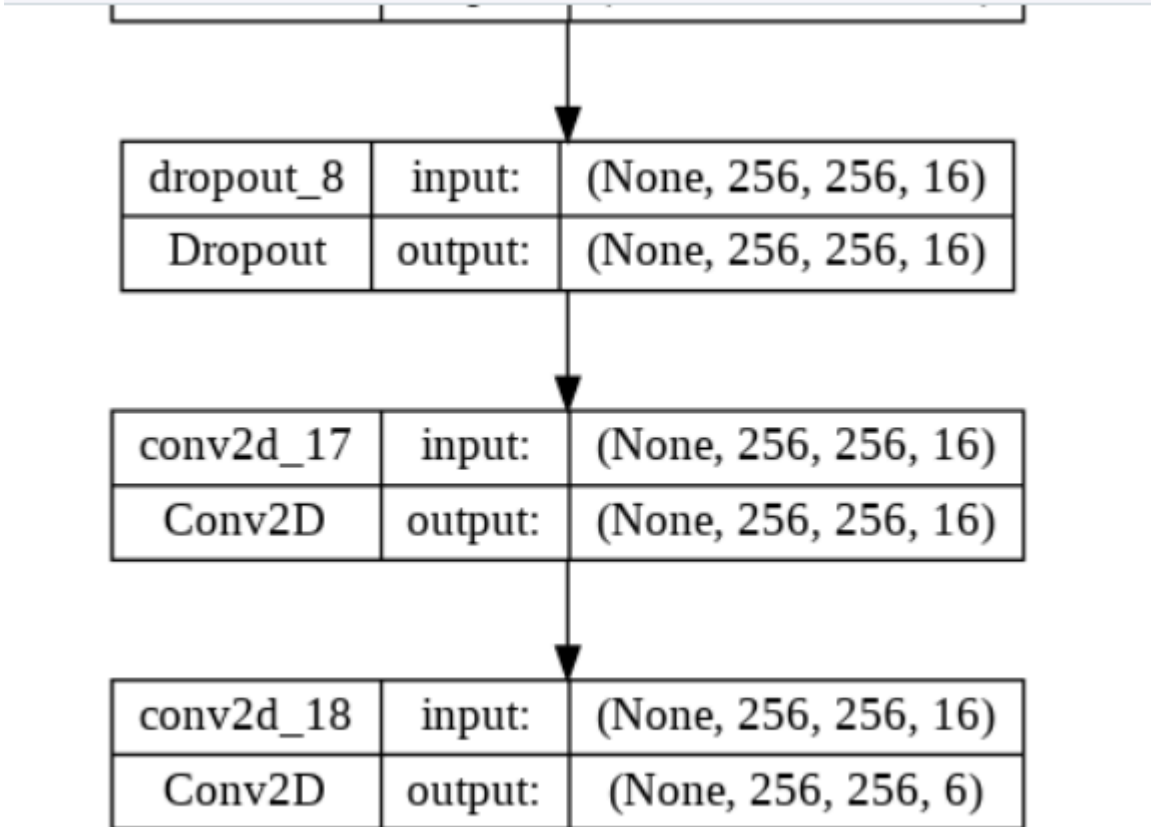
```
plot_model(model, to_file="satellite_model_plot.png", show_shapes=True, show_layer_names=True)
```











dropout_8	input:	(None, 256, 256, 16)
Dropout	output:	(None, 256, 256, 16)

conv2d_17	input:	(None, 256, 256, 16)
Conv2D	output:	(None, 256, 256, 16)

conv2d_18	input:	(None, 256, 256, 16)
Conv2D	output:	(None, 256, 256, 6)