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

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
!nvidia-smi
```

```
Tue Apr 23 20:29:16 2024
+-----------------------------------------------------------------------
------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA
Version: 12.2     |
|-----------------------------------------+----------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A |
Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage |
GPU-Util  Compute M. |
|                                         |                      |
MIG M. |
|
=========================================+======================+======
=================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |
0 |
| N/A   38C    P8               9W /  70W |      0MiB / 15360MiB |
0%      Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------
+----------------------+


+-----------------------------------------------------------------------
------------------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|
=======================================================================
=================|
|  No running processes found
|
+-----------------------------------------------------------------------
------------------+
```

```
!pip install git+https://github.com/paulgavrikov/visualkeras --upgrade
```

```python
import os
import time
import glob
import shutil

# import data handling tools
import cv2
import PIL
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt

# import Deep learning Libraries
import tensorflow as tf
from tensorflow import keras
import tensorflow.image as tfi
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import load_img,img_to_array
from tensorflow.keras.layers import Conv2D,  MaxPool2D, UpSampling2D, 
concatenate, Activation
from tensorflow.keras.layers import Layer, Input, Add, Multiply, 
Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam, Adamax

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

print ('modules loaded')
```

```
modules loaded
```

```python
def create_data(data_dir):
    image_paths = []
    mask_paths = []

    folds = sorted(os.listdir(data_dir))
    for fold in folds:
        foldpath = os.path.join(data_dir, fold)
        if fold in ['image', 'Image', 'images', 'Images', 'IMAGES']:
            images = sorted(os.listdir(foldpath))
            for image in images:
                fpath = os.path.join(foldpath, image)
                image_paths.append(fpath)

        elif fold in ['mask', 'Mask', 'masks', 'Masks', 'MASKS']:
```

```
            masks = sorted(os.listdir(foldpath))
            for mask in masks:
                fpath = os.path.join(foldpath, mask)
                mask_paths.append(fpath)
        else:
            continue

    return image_paths, mask_paths

def load_image(image, SIZE):
    return np.round(tfi.resize(img_to_array(load_img(image)) / 255.,
(SIZE, SIZE)), 4)

# function to read multiple images
def load_images(image_paths, SIZE, mask=False, trim=None):
    if trim is not None:
        image_paths = image_paths[:trim]

    if mask:
        images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 1))
    else:
        images = np.zeros(shape=(len(image_paths), SIZE, SIZE, 3))

    for i, image in enumerate(image_paths):
        img = load_image(image, SIZE)
        if mask:
            images[i] = img[:, :, :1]
        else:
            images[i] = img

    return images

def show_image(image, title=None, cmap=None, alpha=1):
    plt.imshow(image, cmap=cmap, alpha=alpha)
    if title is not None:
        plt.title(title)
    plt.axis('off')

def show_mask(image, mask, cmap=None, alpha=0.4):
    plt.imshow(image)
    plt.imshow(tf.squeeze(mask), cmap=cmap, alpha=alpha)
    plt.axis('off')



def show_images(imgs, msks):
    plt.figure(figsize=(13,8))

    for i in range(15):
        plt.subplot(3,5,i+1)
        id = np.random.randint(len(imgs))
```

```
        show_mask(imgs[id], msks[id], cmap='binary')

    plt.tight_layout()
    plt.show()
```

ENCODER

```
class EncoderBlock(Layer):

    def __init__(self, filters, rate, pooling=True, **kwargs):
        super(EncoderBlock, self).__init__(**kwargs)

        self.filters = filters
        self.rate = rate
        self.pooling = pooling

        self.c1 = Conv2D(filters, kernel_size=3, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal')
        self.drop = Dropout(rate)
        self.c2 = Conv2D(filters, kernel_size=3, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal')
        self.pool = MaxPool2D()

    def call(self, X):
        x = self.c1(X)
        x = self.drop(x)
        x = self.c2(x)
        if self.pooling:
            y = self.pool(x)
            return y, x
        else:
            return x

    def get_config(self):
        base_config = super().get_config()
        return {
            **base_config,
            "filters":self.filters,
            'rate':self.rate,
            'pooling':self.pooling
        }
```

DECODER

```
class DecoderBlock(Layer):

    def __init__(self, filters, rate, **kwargs):
        super(DecoderBlock, self).__init__(**kwargs)
```

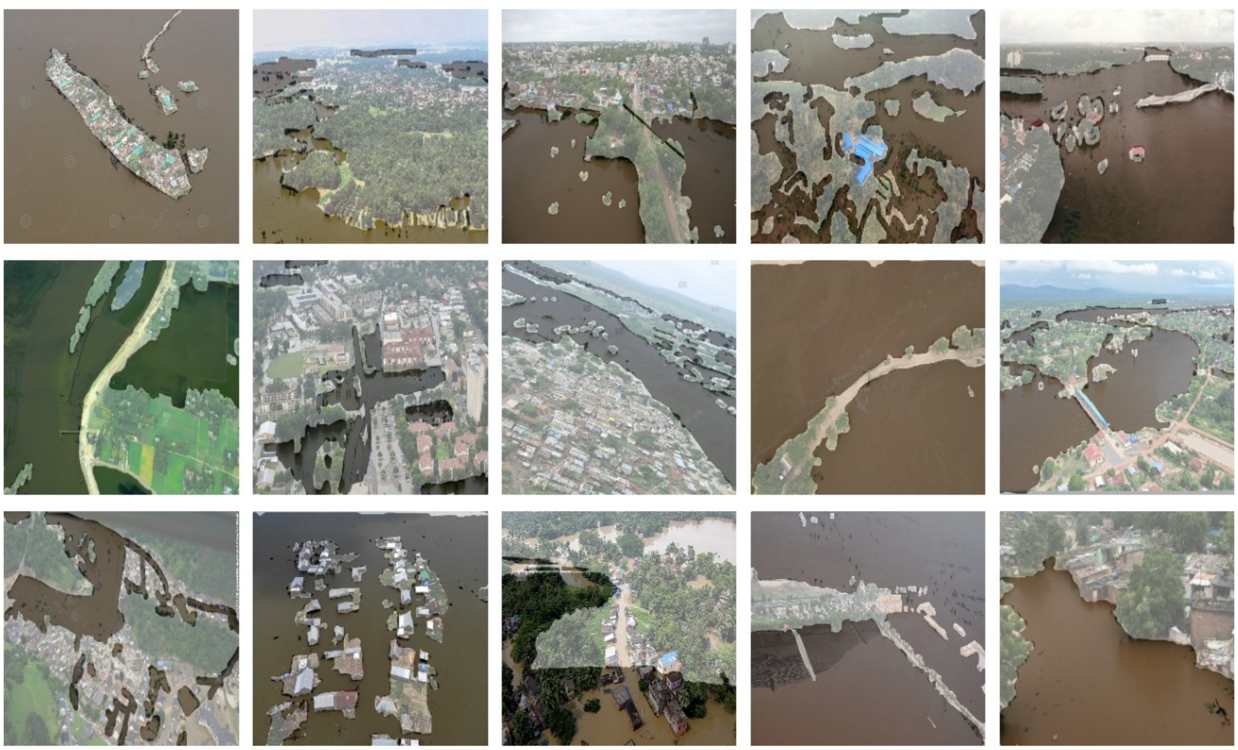
```
        self.filters = filters
        self.rate = rate

        self.up = UpSampling2D()
        self.net = EncoderBlock(filters, rate, pooling=False)

    def call(self, X):
        X, skip_X = X
        x = self.up(X)
        c_ = concatenate([x, skip_X])
        x = self.net(c_)
        return x

    def get_config(self):
        base_config = super().get_config()
        return {
            **base_config,
            "filters":self.filters,
            'rate':self.rate,
        }
```

ATTENTION GATE

```
class AttentionGate(Layer):

    def __init__(self, filters, bn, **kwargs):
        super(AttentionGate, self).__init__(**kwargs)

        self.filters = filters
        self.bn = bn

        self.normal = Conv2D(filters, kernel_size=3, padding='same',
activation='relu', kernel_initializer='he_normal')
        self.down = Conv2D(filters, kernel_size=3, strides=2,
padding='same', activation='relu', kernel_initializer='he_normal')
        self.learn = Conv2D(1, kernel_size=1, padding='same',
activation='sigmoid')
        self.resample = UpSampling2D()
        self.BN = BatchNormalization()

    def call(self, X):
        X, skip_X = X

        x = self.normal(X)
        skip = self.down(skip_X)
        x = Add()([x, skip])
        x = self.learn(x)
        x = self.resample(x)
        f = Multiply()([x, skip_X])
        if self.bn:
```

```
            return self.BN(f)
        else:
            return f
        # return f

    def get_config(self):
        base_config = super().get_config()
        return {
            **base_config,
            "filters":self.filters,
            "bn":self.bn
        }

def plot_training(hist):
    '''
    This function take training model and plot history of accuracy and
losses with the best epoch in both of them.
    '''

    # Define needed variables
    tr_acc = hist.history['accuracy']
    tr_loss = hist.history['loss']
    val_acc = hist.history['val_accuracy']
    val_loss = hist.history['val_loss']
    index_loss = np.argmin(val_loss)
    val_lowest = val_loss[index_loss]
    index_acc = np.argmax(val_acc)
    acc_highest = val_acc[index_acc]
    Epochs = [i+1 for i in range(len(tr_acc))]
    loss_label = f'best epoch= {str(index_loss + 1)}'
    acc_label = f'best epoch= {str(index_acc + 1)}'

    # Plot training history
    plt.figure(figsize= (20, 8))
    plt.style.use('fivethirtyeight')

    plt.subplot(1, 2, 1)
    plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
    plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
    plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label=
loss_label)
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
    plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
    plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label=
```

```
acc_label)
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout
    plt.show()

SIZE = 256

# get data
data_dir = '/content/drive/MyDrive/data_flood_mapping'
image_paths, mask_paths = create_data(data_dir)

# load images and masks
imgs = load_images(image_paths, SIZE)
msks = load_images(mask_paths, SIZE, mask=True)

# show sample
show_images(imgs, msks)
```



```
# Inputs
input_layer = Input(shape= imgs.shape[-3:])

# Encoder
p1, c1 = EncoderBlock(32, 0.1, name="Encoder1")(input_layer)
```

```python
p2, c2 = EncoderBlock(64, 0.1, name="Encoder2")(p1)
p3, c3 = EncoderBlock(128, 0.2, name="Encoder3")(p2)
p4, c4 = EncoderBlock(256, 0.2, name="Encoder4")(p3)

# Encoding
encoding = EncoderBlock(512, 0.3, pooling=False, name="Encoding")(p4)

# Attention + Decoder
a1 = AttentionGate(256, bn=True, name="Attention1")([encoding, c4])
d1 = DecoderBlock(256, 0.2, name="Decoder1")([encoding, a1])

a2 = AttentionGate(128, bn=True, name="Attention2")([d1, c3])
d2 = DecoderBlock(128, 0.2, name="Decoder2")([d1, a2])

a3 = AttentionGate(64, bn=True, name="Attention3")([d2, c2])
d3 = DecoderBlock(64, 0.1, name="Decoder3")([d2, a3])

a4 = AttentionGate(32, bn=True, name="Attention4")([d3, c1])
d4 = DecoderBlock(32,0.1, name="Decoder4")([d3, a4])

# Output
output_layer = Conv2D(1, kernel_size=1, activation='sigmoid',
padding='same')(d4)

# Model
model = Model(inputs= [input_layer], outputs= [output_layer])

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

# Summary
model.summary()
```

```
Model: "model"
______________________________________________________________________________________
 Layer (type)                Output Shape                 Param #
Connected to
======================================================================
============================
 input_1 (InputLayer)        [(None, 256, 256, 3)]        0          []



 Encoder1 (EncoderBlock)     ((None, 128, 128, 32),       10144
['input_1[0][0]']
                              (None, 256, 256, 32))
```

```
 Encoder2 (EncoderBlock)     ((None, 64, 64, 64),         55424
['Encoder1[0][0]']
                              (None, 128, 128, 64))



 Encoder3 (EncoderBlock)     ((None, 32, 32, 128),        221440
['Encoder2[0][0]']
                              (None, 64, 64, 128))



 Encoder4 (EncoderBlock)     ((None, 16, 16, 256),        885248
['Encoder3[0][0]']
                              (None, 32, 32, 256))



 Encoding (EncoderBlock)     (None, 16, 16, 512)          3539968
['Encoder4[0][0]']


 Attention1 (AttentionGate)  (None, 32, 32, 256)          1771265
['Encoding[0][0]',

'Encoder4[0][1]']


 Decoder1 (DecoderBlock)     (None, 32, 32, 256)          2359808
['Encoding[0][0]',

'Attention1[0][0]']


 Attention2 (AttentionGate)  (None, 64, 64, 128)          443265
['Decoder1[0][0]',

'Encoder3[0][1]']


 Decoder2 (DecoderBlock)     (None, 64, 64, 128)          590080
['Decoder1[0][0]',

'Attention2[0][0]']


 Attention3 (AttentionGate)  (None, 128, 128, 64)         111041
['Decoder2[0][0]',

'Encoder2[0][1]']
```

```
 Decoder3 (DecoderBlock)     (None, 128, 128, 64)         147584
['Decoder2[0][0]',

'Attention3[0][0]']


 Attention4 (AttentionGate)  (None, 256, 256, 32)         27873
['Decoder3[0][0]',

'Encoder1[0][1]']


 Decoder4 (DecoderBlock)     (None, 256, 256, 32)         36928
['Decoder3[0][0]',

'Attention4[0][0]']


 conv2d_30 (Conv2D)          (None, 256, 256, 1)          33
['Decoder4[0][0]']


======================================================================
============================
Total params: 10200101 (38.91 MB)
Trainable params: 10199141 (38.91 MB)
Non-trainable params: 960 (3.75 KB)
______________________________________________________________________
____________________________

batch_size = 40     # set batch size for training
epochs = 100         # number of all epochs in training
ask_epoch = 5             # number of epochs to run before asking if
you want to halt training
callbacks =  keras.callbacks.CallbackList(model= model, epochs=
epochs, ask_epoch= ask_epoch)

SPE = len(imgs)//batch_size

# Training
history = model.fit(
    imgs, msks,
    validation_split=0.2,
    epochs=epochs,
    verbose=1,
    steps_per_epoch=SPE,
    batch_size=batch_size
)
```

```
Epoch 1/100
7/7 [==============================] - 108s 8s/step - loss: 1.0328 -
accuracy: 0.5856 - val_loss: 0.6892 - val_accuracy: 0.4414
Epoch 2/100
7/7 [==============================] - 9s 1s/step - loss: 0.5125 -
accuracy: 0.7357 - val_loss: 0.7021 - val_accuracy: 0.3909
Epoch 3/100
7/7 [==============================] - 9s 1s/step - loss: 0.4724 -
accuracy: 0.7791 - val_loss: 0.6505 - val_accuracy: 0.6643
Epoch 4/100
7/7 [==============================] - 9s 1s/step - loss: 0.4365 -
accuracy: 0.7909 - val_loss: 0.6511 - val_accuracy: 0.6023
Epoch 5/100
7/7 [==============================] - 9s 1s/step - loss: 0.4128 -
accuracy: 0.8011 - val_loss: 0.6616 - val_accuracy: 0.5701
Epoch 6/100
7/7 [==============================] - 9s 1s/step - loss: 0.4053 -
accuracy: 0.8084 - val_loss: 0.6558 - val_accuracy: 0.5823
Epoch 7/100
7/7 [==============================] - 9s 1s/step - loss: 0.3782 -
accuracy: 0.8227 - val_loss: 0.6415 - val_accuracy: 0.6252
Epoch 8/100
7/7 [==============================] - 9s 1s/step - loss: 0.3781 -
accuracy: 0.8242 - val_loss: 0.6492 - val_accuracy: 0.5984
Epoch 9/100
7/7 [==============================] - 9s 1s/step - loss: 0.3921 -
accuracy: 0.8192 - val_loss: 0.6065 - val_accuracy: 0.7473
Epoch 10/100
7/7 [==============================] - 9s 1s/step - loss: 0.3866 -
accuracy: 0.8210 - val_loss: 0.6376 - val_accuracy: 0.6151
Epoch 11/100
7/7 [==============================] - 9s 1s/step - loss: 0.3867 -
accuracy: 0.8206 - val_loss: 0.6146 - val_accuracy: 0.7144
Epoch 12/100
7/7 [==============================] - 9s 1s/step - loss: 0.3755 -
accuracy: 0.8260 - val_loss: 0.6204 - val_accuracy: 0.6676
Epoch 13/100
7/7 [==============================] - 9s 1s/step - loss: 0.3681 -
accuracy: 0.8282 - val_loss: 0.6115 - val_accuracy: 0.6903
Epoch 14/100
7/7 [==============================] - 9s 1s/step - loss: 0.3480 -
accuracy: 0.8400 - val_loss: 0.6042 - val_accuracy: 0.7056
Epoch 15/100
7/7 [==============================] - 9s 1s/step - loss: 0.3579 -
accuracy: 0.8337 - val_loss: 0.6156 - val_accuracy: 0.6726
Epoch 16/100
7/7 [==============================] - 9s 1s/step - loss: 0.3315 -
accuracy: 0.8467 - val_loss: 0.6052 - val_accuracy: 0.6844
Epoch 17/100
7/7 [==============================] - 9s 1s/step - loss: 0.3432 -
```

```
accuracy: 0.8418 - val_loss: 0.5838 - val_accuracy: 0.7118
Epoch 18/100
7/7 [==============================] - 9s 1s/step - loss: 0.3453 -
accuracy: 0.8425 - val_loss: 0.5930 - val_accuracy: 0.7199
Epoch 19/100
7/7 [==============================] - 9s 1s/step - loss: 0.3316 -
accuracy: 0.8469 - val_loss: 0.5853 - val_accuracy: 0.6926
Epoch 20/100
7/7 [==============================] - 9s 1s/step - loss: 0.3373 -
accuracy: 0.8461 - val_loss: 0.5928 - val_accuracy: 0.7128
Epoch 21/100
7/7 [==============================] - 9s 1s/step - loss: 0.3257 -
accuracy: 0.8508 - val_loss: 0.5761 - val_accuracy: 0.7509
Epoch 22/100
7/7 [==============================] - 9s 1s/step - loss: 0.3378 -
accuracy: 0.8455 - val_loss: 0.5763 - val_accuracy: 0.7227
Epoch 23/100
7/7 [==============================] - 9s 1s/step - loss: 0.3148 -
accuracy: 0.8544 - val_loss: 0.5666 - val_accuracy: 0.7249
Epoch 24/100
7/7 [==============================] - 9s 1s/step - loss: 0.3256 -
accuracy: 0.8511 - val_loss: 0.5577 - val_accuracy: 0.7575
Epoch 25/100
7/7 [==============================] - 9s 1s/step - loss: 0.3062 -
accuracy: 0.8592 - val_loss: 0.5773 - val_accuracy: 0.6910
Epoch 26/100
7/7 [==============================] - 9s 1s/step - loss: 0.3091 -
accuracy: 0.8581 - val_loss: 0.5571 - val_accuracy: 0.7510
Epoch 27/100
7/7 [==============================] - 9s 1s/step - loss: 0.3120 -
accuracy: 0.8582 - val_loss: 0.5811 - val_accuracy: 0.6979
Epoch 28/100
7/7 [==============================] - 9s 1s/step - loss: 0.2941 -
accuracy: 0.8656 - val_loss: 0.5367 - val_accuracy: 0.7567
Epoch 29/100
7/7 [==============================] - 9s 1s/step - loss: 0.3086 -
accuracy: 0.8613 - val_loss: 0.5412 - val_accuracy: 0.7584
Epoch 30/100
7/7 [==============================] - 9s 1s/step - loss: 0.3221 -
accuracy: 0.8521 - val_loss: 0.5575 - val_accuracy: 0.7062
Epoch 31/100
7/7 [==============================] - 9s 1s/step - loss: 0.3106 -
accuracy: 0.8566 - val_loss: 0.5362 - val_accuracy: 0.7629
Epoch 32/100
7/7 [==============================] - 9s 1s/step - loss: 0.3033 -
accuracy: 0.8614 - val_loss: 0.5833 - val_accuracy: 0.6973
Epoch 33/100
7/7 [==============================] - 9s 1s/step - loss: 0.3016 -
accuracy: 0.8621 - val_loss: 0.5084 - val_accuracy: 0.7735
Epoch 34/100
```

```
7/7 [==============================] - 9s 1s/step - loss: 0.3002 -
accuracy: 0.8636 - val_loss: 0.5101 - val_accuracy: 0.7658
Epoch 35/100
7/7 [==============================] - 9s 1s/step - loss: 0.2934 -
accuracy: 0.8647 - val_loss: 0.4968 - val_accuracy: 0.7857
Epoch 36/100
7/7 [==============================] - 9s 1s/step - loss: 0.2814 -
accuracy: 0.8709 - val_loss: 0.4861 - val_accuracy: 0.7759
Epoch 37/100
7/7 [==============================] - 9s 1s/step - loss: 0.2796 -
accuracy: 0.8707 - val_loss: 0.5035 - val_accuracy: 0.7644
Epoch 38/100
7/7 [==============================] - 9s 1s/step - loss: 0.2846 -
accuracy: 0.8702 - val_loss: 0.4626 - val_accuracy: 0.8105
Epoch 39/100
7/7 [==============================] - 9s 1s/step - loss: 0.2748 -
accuracy: 0.8730 - val_loss: 0.4701 - val_accuracy: 0.7852
Epoch 40/100
7/7 [==============================] - 9s 1s/step - loss: 0.2809 -
accuracy: 0.8724 - val_loss: 0.4739 - val_accuracy: 0.7863
Epoch 41/100
7/7 [==============================] - 9s 1s/step - loss: 0.2708 -
accuracy: 0.8774 - val_loss: 0.4840 - val_accuracy: 0.7775
Epoch 42/100
7/7 [==============================] - 9s 1s/step - loss: 0.2831 -
accuracy: 0.8692 - val_loss: 0.4651 - val_accuracy: 0.8005
Epoch 43/100
7/7 [==============================] - 9s 1s/step - loss: 0.3024 -
accuracy: 0.8628 - val_loss: 0.4993 - val_accuracy: 0.7635
Epoch 44/100
7/7 [==============================] - 9s 1s/step - loss: 0.2792 -
accuracy: 0.8730 - val_loss: 0.4894 - val_accuracy: 0.7716
Epoch 45/100
7/7 [==============================] - 9s 1s/step - loss: 0.2726 -
accuracy: 0.8758 - val_loss: 0.4675 - val_accuracy: 0.7859
Epoch 46/100
7/7 [==============================] - 9s 1s/step - loss: 0.2656 -
accuracy: 0.8795 - val_loss: 0.4646 - val_accuracy: 0.7896
Epoch 47/100
7/7 [==============================] - 9s 1s/step - loss: 0.2592 -
accuracy: 0.8806 - val_loss: 0.4515 - val_accuracy: 0.7970
Epoch 48/100
7/7 [==============================] - 9s 1s/step - loss: 0.2652 -
accuracy: 0.8768 - val_loss: 0.5467 - val_accuracy: 0.7183
Epoch 49/100
7/7 [==============================] - 9s 1s/step - loss: 0.2567 -
accuracy: 0.8821 - val_loss: 0.4436 - val_accuracy: 0.8123
Epoch 50/100
7/7 [==============================] - 9s 1s/step - loss: 0.2520 -
accuracy: 0.8843 - val_loss: 0.4889 - val_accuracy: 0.7796
```

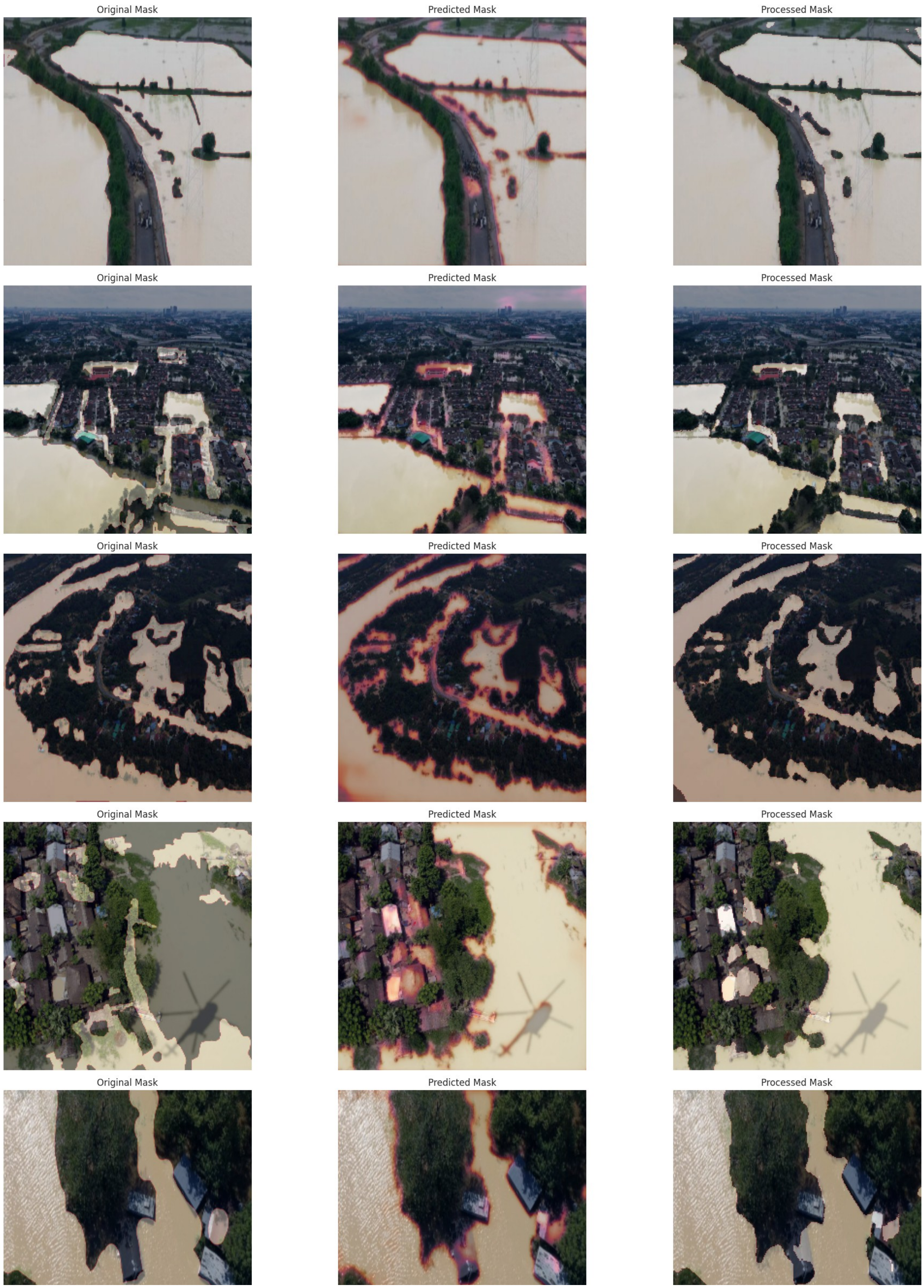
```
Epoch 51/100
7/7 [==============================] - 9s 1s/step - loss: 0.2405 -
accuracy: 0.8883 - val_loss: 0.4498 - val_accuracy: 0.8048
Epoch 52/100
7/7 [==============================] - 9s 1s/step - loss: 0.2547 -
accuracy: 0.8838 - val_loss: 0.4127 - val_accuracy: 0.8243
Epoch 53/100
7/7 [==============================] - 9s 1s/step - loss: 0.2495 -
accuracy: 0.8862 - val_loss: 0.4519 - val_accuracy: 0.8010
Epoch 54/100
7/7 [==============================] - 9s 1s/step - loss: 0.2424 -
accuracy: 0.8883 - val_loss: 0.4720 - val_accuracy: 0.7961
Epoch 55/100
7/7 [==============================] - 9s 1s/step - loss: 0.2429 -
accuracy: 0.8886 - val_loss: 0.4275 - val_accuracy: 0.8179
Epoch 56/100
7/7 [==============================] - 9s 1s/step - loss: 0.2473 -
accuracy: 0.8869 - val_loss: 0.4490 - val_accuracy: 0.8056
Epoch 57/100
7/7 [==============================] - 9s 1s/step - loss: 0.2408 -
accuracy: 0.8886 - val_loss: 0.4388 - val_accuracy: 0.8084
Epoch 58/100
7/7 [==============================] - 9s 1s/step - loss: 0.2322 -
accuracy: 0.8901 - val_loss: 0.4333 - val_accuracy: 0.8166
Epoch 59/100
7/7 [==============================] - 9s 1s/step - loss: 0.2383 -
accuracy: 0.8907 - val_loss: 0.4815 - val_accuracy: 0.7969
Epoch 60/100
7/7 [==============================] - 9s 1s/step - loss: 0.2318 -
accuracy: 0.8931 - val_loss: 0.4327 - val_accuracy: 0.8173
Epoch 61/100
7/7 [==============================] - 9s 1s/step - loss: 0.2254 -
accuracy: 0.8947 - val_loss: 0.4369 - val_accuracy: 0.8237
Epoch 62/100
7/7 [==============================] - 9s 1s/step - loss: 0.2231 -
accuracy: 0.8968 - val_loss: 0.4217 - val_accuracy: 0.8199
Epoch 63/100
7/7 [==============================] - 9s 1s/step - loss: 0.2071 -
accuracy: 0.9027 - val_loss: 0.4582 - val_accuracy: 0.8136
Epoch 64/100
7/7 [==============================] - 9s 1s/step - loss: 0.2276 -
accuracy: 0.8933 - val_loss: 0.3961 - val_accuracy: 0.8398
Epoch 65/100
7/7 [==============================] - 9s 1s/step - loss: 0.2065 -
accuracy: 0.9021 - val_loss: 0.4062 - val_accuracy: 0.8313
Epoch 66/100
7/7 [==============================] - 9s 1s/step - loss: 0.2094 -
accuracy: 0.9008 - val_loss: 0.4426 - val_accuracy: 0.8297
Epoch 67/100
7/7 [==============================] - 9s 1s/step - loss: 0.2140 -
```

```
accuracy: 0.8982 - val_loss: 0.4222 - val_accuracy: 0.8291
Epoch 68/100
7/7 [==============================] - 9s 1s/step - loss: 0.1961 -
accuracy: 0.9071 - val_loss: 0.4380 - val_accuracy: 0.8272
Epoch 69/100
7/7 [==============================] - 9s 1s/step - loss: 0.1967 -
accuracy: 0.9046 - val_loss: 0.4325 - val_accuracy: 0.8313
Epoch 70/100
7/7 [==============================] - 9s 1s/step - loss: 0.1795 -
accuracy: 0.9129 - val_loss: 0.4492 - val_accuracy: 0.8266
Epoch 71/100
7/7 [==============================] - 9s 1s/step - loss: 0.1957 -
accuracy: 0.9069 - val_loss: 0.4195 - val_accuracy: 0.8386
Epoch 72/100
7/7 [==============================] - 9s 1s/step - loss: 0.1851 -
accuracy: 0.9101 - val_loss: 0.3902 - val_accuracy: 0.8444
Epoch 73/100
7/7 [==============================] - 9s 1s/step - loss: 0.1823 -
accuracy: 0.9121 - val_loss: 0.4429 - val_accuracy: 0.8306
Epoch 74/100
7/7 [==============================] - 9s 1s/step - loss: 0.1765 -
accuracy: 0.9131 - val_loss: 0.4553 - val_accuracy: 0.8358
Epoch 75/100
7/7 [==============================] - 9s 1s/step - loss: 0.1789 -
accuracy: 0.9130 - val_loss: 0.4732 - val_accuracy: 0.8327
Epoch 76/100
7/7 [==============================] - 9s 1s/step - loss: 0.1911 -
accuracy: 0.9082 - val_loss: 0.5227 - val_accuracy: 0.8130
Epoch 77/100
7/7 [==============================] - 9s 1s/step - loss: 0.2072 -
accuracy: 0.9021 - val_loss: 0.4570 - val_accuracy: 0.8192
Epoch 78/100
7/7 [==============================] - 9s 1s/step - loss: 0.2043 -
accuracy: 0.9032 - val_loss: 0.4557 - val_accuracy: 0.8268
Epoch 79/100
7/7 [==============================] - 9s 1s/step - loss: 0.1987 -
accuracy: 0.9049 - val_loss: 0.4350 - val_accuracy: 0.8283
Epoch 80/100
7/7 [==============================] - 9s 1s/step - loss: 0.2068 -
accuracy: 0.9035 - val_loss: 0.5801 - val_accuracy: 0.8047
Epoch 81/100
7/7 [==============================] - 9s 1s/step - loss: 0.2037 -
accuracy: 0.9042 - val_loss: 0.5246 - val_accuracy: 0.8215
Epoch 82/100
7/7 [==============================] - 9s 1s/step - loss: 0.1876 -
accuracy: 0.9112 - val_loss: 0.4636 - val_accuracy: 0.8285
Epoch 83/100
7/7 [==============================] - 9s 1s/step - loss: 0.1766 -
accuracy: 0.9132 - val_loss: 0.4936 - val_accuracy: 0.8356
Epoch 84/100
```

```
7/7 [==============================] - 9s 1s/step - loss: 0.1781 -
accuracy: 0.9127 - val_loss: 0.4846 - val_accuracy: 0.8304
Epoch 85/100
7/7 [==============================] - 9s 1s/step - loss: 0.1685 -
accuracy: 0.9172 - val_loss: 0.4927 - val_accuracy: 0.8306
Epoch 86/100
5/7 [====================>.........] - ETA: 2s - loss: 0.1651 -
accuracy: 0.9176

WARNING:tensorflow:Your input ran out of data; interrupting training.
Make sure that your dataset or generator can generate at least
`steps_per_epoch * epochs` batches (in this case, 700 batches). You
may need to use the repeat() function when building your dataset.

 7/7 [==============================] - 7s 920ms/step - loss: 0.1651 -
accuracy: 0.9176 - val_loss: 0.4758 - val_accuracy: 0.8408
```

```
plt.figure(figsize=(20,25))
n=0
for i in range(1,(5*3)+1):
    plt.subplot(5,3,i)
    if n==0:
        id = np.random.randint(len(imgs))
        image = imgs[id]
        mask = msks[id]
        pred_mask = model.predict(image[np.newaxis,...])

        plt.title("Original Mask")
        show_mask(image, mask)
        n+=1
    elif n==1:
        plt.title("Predicted Mask")
        show_mask(image, pred_mask)
        n+=1
    elif n==2:
        pred_mask = (pred_mask>0.5).astype('float')
        plt.title("Processed Mask")
        show_mask(image, pred_mask)
        n=0
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
```


Original Mask
Predicted Mask
Processed Mask
Original Mask
Predicted Mask
Processed Mask
Original Mask
Predicted Mask
Processed Mask
Original Mask
Predicted Mask
Processed Mask
Original Mask
Predicted Mask
Processed Mask

```python
image="/content/2.png"
img=load_image(image, SIZE)

show_image(img)
```

```python
pred_mask=model.predict(img[np.newaxis,...])
```

```
1/1 [==============================] - 0s 25ms/step
```

```python
pred_mask
```

```
array([[[[0.26861653],
         [0.21226007],
         [0.22648981],
         ...,
         [0.22373553],
         [0.30906814],
         [0.33539614]],

        [[0.1831808 ],
         [0.14903279],
         [0.18405148],
         ...,
         [0.16577838],
         [0.20780028],
         [0.24588585]],
```

```
        [[0.15418217],
         [0.14800848],
         [0.22503866],
         ...,
         [0.17990346],
         [0.25901395],
         [0.2017259 ]],

        ...,

        [[0.7948899 ],
         [0.8742254 ],
         [0.91562206],
         ...,
         [0.00708052],
         [0.01294582],
         [0.04032484]],

        [[0.7432741 ],
         [0.8122438 ],
         [0.8550394 ],
         ...,
         [0.01110231],
         [0.0203253 ],
         [0.06639417]],

        [[0.59902775],
         [0.72127146],
         [0.80490535],
         ...,
         [0.04503132],
         [0.07745422],
         [0.18930854]]]], dtype=float32)

show_mask(img,pred_mask)
```

```
model.save('flood_mapping_unet.h5')
```