ML Class Assignment

Mô tả bài toán

Một số khu vực trên Trái Đất với có trữ lượng khổng lồ dầu, khí và muối ở dưới bề mặt. Hình ảnh địa chấn được tạo ra từ chụp ảnh phản xạ từ các ranh giới đá. Hình ảnh địa chấn cho thấy ranh giới giữa các loại đá khác nhau. Về lý thuyết, cường độ phản xạ tỷ lệ thuận với sự khác biệt về đặc tính vật lý ở hai bên của giao diện. Trong khi các hình ảnh địa chấn cho thấy ranh giới của đá, chúng không nói nhiều về bản thân tảng đá; một số loại đá rất dễ xác định trong khi một số loại rất khó. Có một số khu vực trên thế giới có một lượng lớn muối ở dưới bề mặt. Một trong những thách thức của chụp ảnh địa chấn là xác định phần của bề mặt dưới bề mặt là muối. Ta vẫn phải cần đến các chuyên gia phân tích hình ảnh địa chấn để phân tích lượng muối. Tuy nhiên, kết quả có được vẫn rất chủ quan. Vì vậy, chúng ta cần một hệ thống tự động xác định xem khu vực dưới bề mặt đó có muối hay không.

Bài toán cần giải quyết là: Đưa ra mask dự đoán khu vực có muối từ ảnh địa chấn đầu vào.

- Đầu vào: Ảnh địa chấn khu vực cần phần tích.
- Đầu ra: Mask dự đoán các khu vực trong ảnh.

Dữ liệu thô

Dữ liệu là một tập hợp các hình ảnh địa chấn được chọn tại các vị trí khác nhau. Hình ảnh có kích thước 101×101 pixel được phân loại là muối hoặc trầm tích. Ngoài các hình ảnh địa chấn, độ sâu của vị trí được chụp ảnh được cung cấp cho mỗi hình ảnh.

File “train.csv”:

- Cột “id”: ID cũng như là tên của ảnh
- Cột “rle_mask”: mã hóa loạt dài được mã hóa từ mask của ảnh nhằm giảm dung lượng lưu trữ. Các mask sẽ được làm phẳng theo thứ tự từ trên xuống dưới, từ trái qua phải. Sau đó sẽ được mã hóa run-length theo vị trí pixel có giá trị bằng 1 và số pixel có giá trị bằng 1 kể từ nó. Ví dụ mask = [1 0 1 1 1 0 0 1 1] → rle = 1 1 3 3 8 2.

File “depths.csv”: Tên (ID) của ảnh và độ sâu của vị trí chụp ảnh

File “sample_submission.csv”: Tên (ID) của ảnh và rle_mask cần dự đoán của ảnh đó

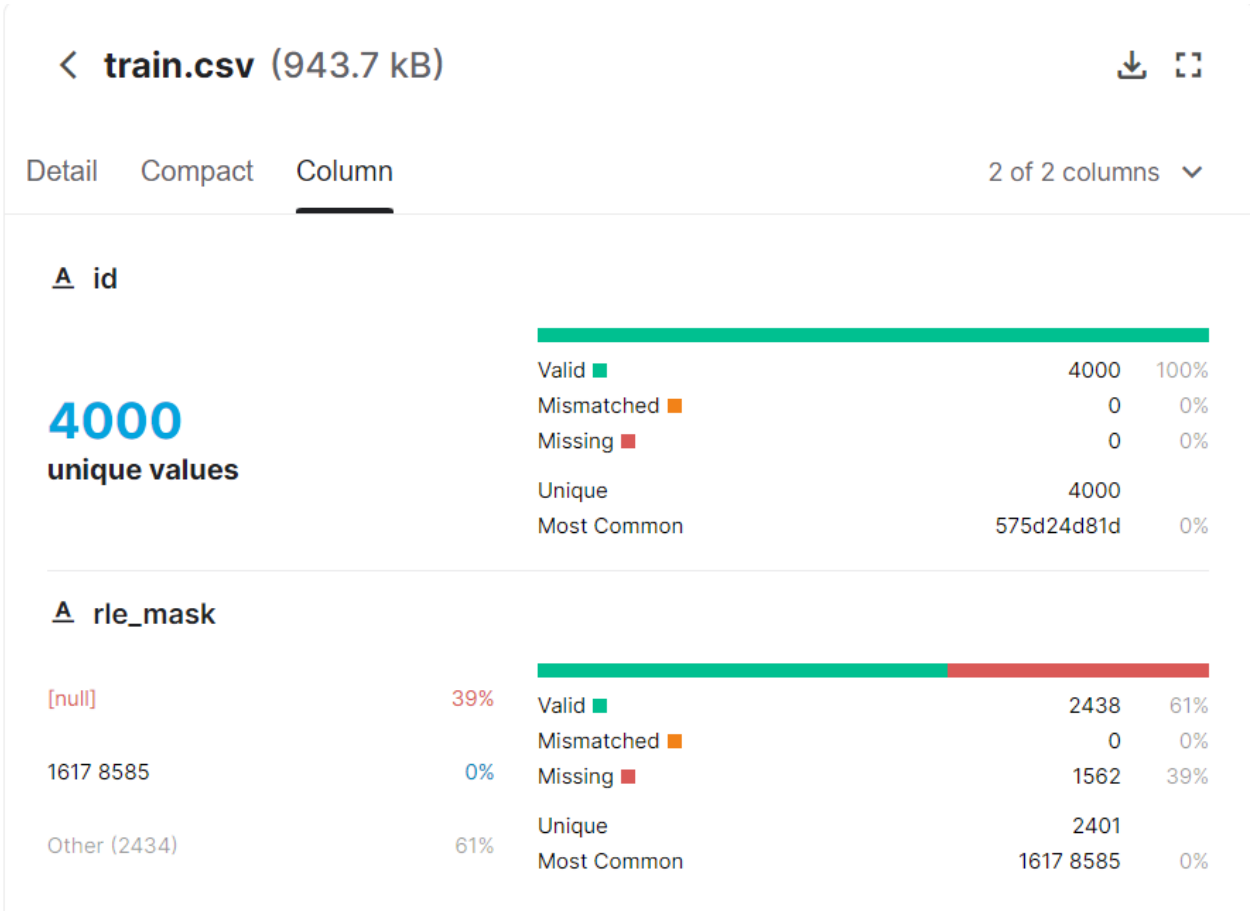
File “train.zip” → Giải nén thành folder “train”:

- Folder “images” chứa 4000 ảnh huấn luyện
- Folder “masks” chứa 4000 ảnh mask tương ứng

File “test.zip” → Giải nén thành folder “test”: chứa 18000 ảnh địa chấn cần dự đoán mask.

Xử lý dữ liệu

Đầu tiên, em load file “train.csv” vào bảng rồi xem tập dữ liệu huấn luyện và thấy rằng tại cột “rle-mask”, có đến 1562 (~39%) bản ghi có giá trị là null - tức là có đến 1562 bức ảnh địa chấn không có muối. 2438 bức ảnh còn lại là có muối, tuy nhiên, có khả năng trong số đó có rất nhiều bức ảnh có muối nhưng rất ít - khó xác định khu vực có muối.



	id	rle_mask
0	575d24d81d	NaN
1	a266a2a9df	5051 5151
2	75efad62c1	9 93 109 94 210 94 310 95 411 95 511 96 612 96...
3	34e51dba6a	48 54 149 54 251 53 353 52 455 51 557 50 659 4...
4	4875705fb0	1111 1 1212 1 1313 1 1414 1 1514 2 1615 2 1716...

Sau đó, em tính độ bao phủ của muối trên từng ảnh từ chuỗi rle_mask của ảnh đó. Như đã giải thích ở trên phần giới thiệu, trong chuỗi rle, giá trị của các phần tử có chỉ số (index) lẻ là số pixel chứa muối trong mask. Do đó em tính số pixel chứa muối bằng tổng các số có chỉ số lẻ trong chuỗi rle, rồi chia cho số pixel của ảnh. Thông tin về độ bao phủ muối cũng được đưa vào bảng thông qua cột “coverage”.

	id	rle_mask	coverage
0	575d24d81d	NaN	0.0
1	a266a2a9df	5051 5151	0.5
2	75efad62c1	9 93 109 94 210 94 310 95 411 95 511 96 612 96...	1.0
3	34e51dba6a	48 54 149 54 251 53 353 52 455 51 557 50 659 4...	0.1
4	4875705fb0	1111 1 1212 1 1313 1 1414 1 1514 2 1615 2 1716...	0.0

Từ biểu đồ và số liệu về độ bao phủ muối của các ảnh trong tập huấn luyện, chúng ta cũng thấy được rằng có số lượng lớn các ảnh địa chấn hoàn toàn không có muối hoặc có cực kì ít muối.

Tiếp theo, em load ảnh và mask vào bảng thông qua ID ~ tên của ảnh. Ảnh và mask được chuẩn hóa về khoảng [0, 1] để tính có thể tính loss khi huấn luyện mô hình.

	id	rle_mask	coverage	images
0	575d24d81d	NaN	0.0	[[[0.5254901960784314], [0.5137254901960784], ...
1	a266a2a9df	5051 5151	0.5	[[[0.3411764705882353], [0.3764705882352941], ...
2	75efad62c1	9 93 109 94 210 94 310 95 411 95 511 96 612 96...	1.0	[[[0.5686274509803921], [0.4666666666666667], ...
3	34e51dba6a	48 54 149 54 251 53 353 52 455 51 557 50 659 4...	0.1	[[[0.5411764705882353], [0.4745098039215686], ...
4	4875705fb0	1111 1 1212 1 1313 1 1414 1 1514 2 1615 2 1716...	0.0	[[[0.06666666666666667], [0.0784313725490196],...

	id	rle_mask	coverage	images	masks
0	575d24d81d	NaN	0.0	[[[0.5254901960784314], [0.5137254901960784], ...	[[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0...
1	a266a2a9df	5051 5151	0.5	[[[0.3411764705882353], [0.3764705882352941], ...	[[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0...
2	75efad62c1	9 93 109 94 210 94 310 95 411 95 511 96 612 96...	1.0	[[[0.5686274509803921], [0.4666666666666667], ...	[[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0...
3	34e51dba6a	48 54 149 54 251 53 353 52 455 51 557 50 659 4...	0.1	[[[0.5411764705882353], [0.4745098039215686], ...	[[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0...
4	4875705fb0	1111 1 1212 1 1313 1 1414 1 1514 2 1615 2 1716...	0.0	[[[0.06666666666666667], [0.0784313725490196],...	[[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0...

Cuối cùng, em chia tập “train” và “valid” theo độ bao phủ muối với tỉ lệ $8 / 2$. Và để tăng lượng dữ liệu, em lật ảnh theo hướng trái/phải.

Metric của cuộc thi

Metric được sử dụng cho cuộc thi này là **mean average precision** (MAP) tại các ngưỡng **intersection over union** (IoU) khác nhau.

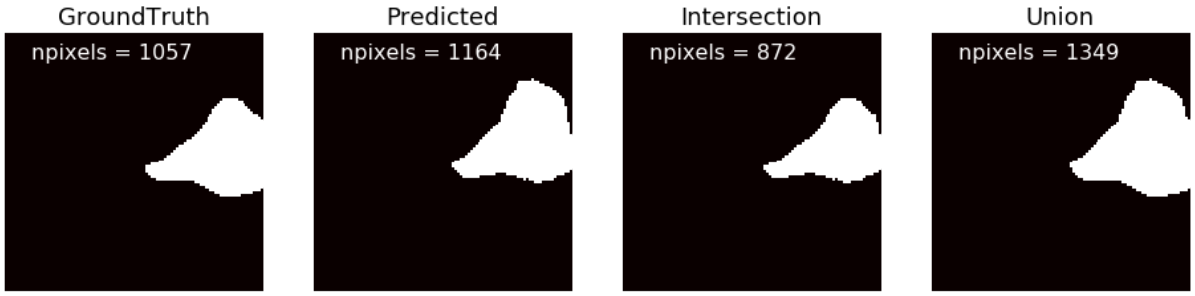
Các bước để tính điểm (Score):

1. Tính IoU:

Giả sử chúng ta có 1 bức ảnh mới mask đã cho và mask dự đoán như sau

Intersection: Phần giao: Là sự giao nhau giữa khu vực muối của 2 ảnh (Phần muối có cả ở 2 mask). Vì trên ảnh mask, 0 là trầm tích và 1 là muối, nên intersection sẽ tương đương với phép AND logic của từng pixel trên 2 ảnh

Union: Phần hợp: Là sự kết hợp giữa 2 khu vực muối của 2 ảnh (Phần muối có ở 1 trong 2 hoặc cả 2 mask). Union sẽ tương đương với phép OR logic của từng pixel trên 2 ảnh.



Công thức được định nghĩa: $IoU(A, B) = \frac{A \cap B}{A \cup B}$. Với ảnh trên, $IoU = \frac{872}{1349}$

2. So sánh IoU vừa tính bên trên với các ngưỡng.

So sánh IoU bên trên với từng ngưỡng trong tập hợp các ngưỡng sau: [0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95].

Chúng ta sẽ dùng 1 véc-tơ để lưu sự so sánh các ngưỡng. Nếu IoU > 1 ngưỡng, giá trị trong véc-tơ sẽ là 1, IoU < 1 thì giá trị trong véc-tơ sẽ là 0

3. Tính trung bình trên từng ảnh

Giá trị trung bình = số ngưỡng đạt được/ tổng số ngưỡng.

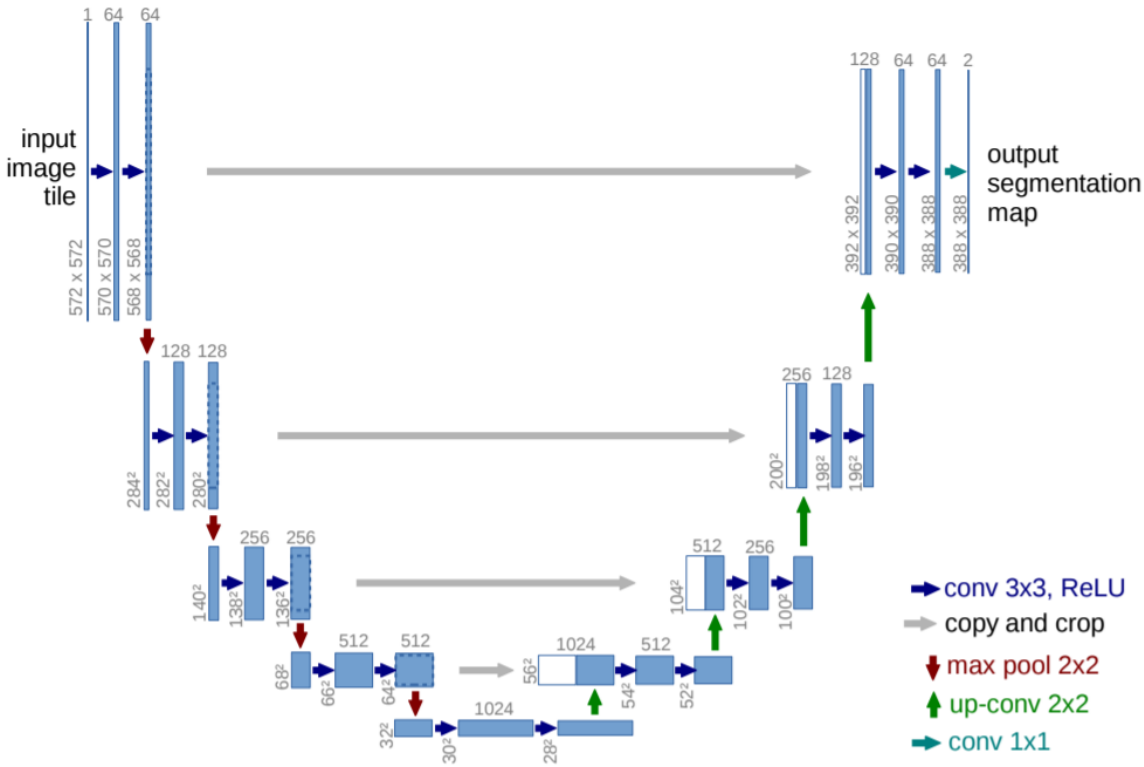
4. Tính trung bình trên toàn bộ tập dữ liệu.

Mỗi ảnh đã được tính giá trị trung bình với công thức trên. Sau đó, chúng ta sẽ tính trung bình trên toàn bộ ảnh của tập dữ liệu.

Mô hình

Do đây là bài toán phân vùng muối trong ảnh địa chấn - Salt Segmentation nên em chọn mô hình mạng U-Net. Trong bài toán này, ta cần phân loại mỗi pixel trong ảnh. Với mỗi pixel trong ảnh ta cần xác định xem nó là trầm tích hay là muối. Ảnh đầu vào (input) và đầu ra (output) có cùng kích thước.

U-Net



Mô hình này được gọi là U-Net do nó có hình dáng giống chữ U.

- Phần bên trái: Phần thu hẹp, có vai trò như 1 Encoder, làm nhiệm vụ trích lọc đặc trưng để tìm ra bối cảnh của ảnh. Phần này được thiết kế bao gồm nhiều (3-4) tầng, mỗi tầng bao gồm các lớp tích chập (Conv) 3×3. Chiều dài và chiều rộng (width & height) của các layer giảm dần trong khi depth lại tăng. Kết thúc mỗi tầng là lớp Max pool 2×2, giảm kích thước width và height đi 1 nửa.
- Phần bên trái: Phần mở rộng, có vài trò giống 1 Decoder, mục đích khôi phục lại kích thước gốc của ảnh. Trong khi phần bên trái dùng lớp Max pool để giảm kích thước, thì phần bên phải dùng lớp Up-conv để tăng kích thước. Bên cạnh đó, có các đường màu xám từ phần bên trái qua phần bên phải, nó nối layer trước với layer hiện tại để tránh Vanishing gradient cũng như mang được các thông tin cần thiết từ layer trước tới layer sau.
- Vì đây là bài toán mang tính phân loại cho mỗi pixel, nên em dùng hàm loss “binary cross entropy”.

Code

Mô hình mạng của em cũng được xây dựng dựa trên kiến trúc mạng U-Net như trên. Ở tầng đầu tiên, đầu vào là ảnh địa chấn có kích thước $101 \times 101 \times 1$ được đi qua 2 lớp Conv2D để tăng depth và padding=”same” để giữ nguyên kích thước; đi kèm sau mỗi lớp tích chập là 1 hàm activation ReLU. Sau đó, ảnh được đi qua lớp MaxPool để giảm đi 1 nửa kích thước chiều dài và rộng. Ở cuối tầng là 1 lớp Dropout để tăng hiệu suất huấn luyện.

Sau khi đi qua 4 tầng, ảnh còn lại kích thước $6 \times 6 \times 128$. Ảnh được đi qua 2 lớp Conv2D ở tầng đáy để tăng depth ($6 \times 6 \times 256$), sau đó bắt đầu vào phần bên phải của mạng U-Net để khôi phục kích thước ban đầu. Ở tầng đầu tiên của phần decoder, sau khi qua lớp Conv2DTranspose để tăng kích thước, giảm depth, ảnh được nối thêm một phần cùng kích thước ở layer tương ứng bên encoder (skip connection). Sau đó, ảnh lại được đưa qua 2 lớp Conv2D để giảm depth và giữ nguyên kích thước. Tuy nhiên, có 2 tầng trong phần decoder, để cùng kích thước với ảnh được nối, lớp Conv2DTranspose sẽ có padding=”valid”. Ở lớp cuối cùng, đầu ra sẽ được đi qua Conv 1×1 để trở về kích thước ban đầu. Hàm kích hoạt được dùng ở lớp này sẽ là sigmoid do từng pixel trong ảnh có thể là 0-trầm tích hoặc 1-muối.

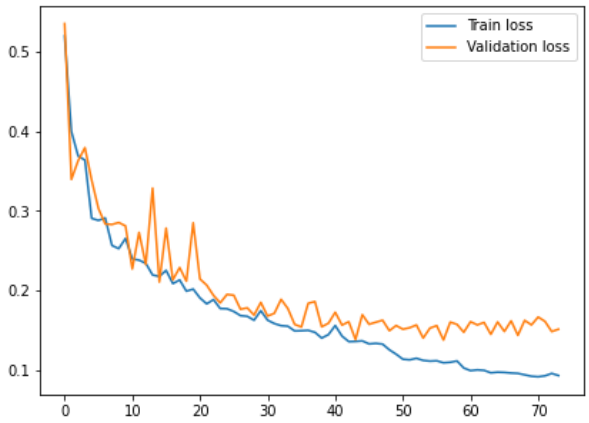
```
def build_model(input_layer):
    start_filters = 16
    # 101 -> 50
    conv1 = Conv2D(start_filters * 1, (3, 3), activation="relu", padding="same")(input_layer)
    conv1 = Conv2D(start_filters * 1, (3, 3), activation="relu", padding="same")(conv1)
    pool1 = MaxPooling2D((2, 2))(conv1)
    pool1 = Dropout(0.25)(pool1)

    # 50 -> 25
    conv2 = Conv2D(start_filters * 2, (3, 3), activation="relu", padding="same")(pool1)
    conv2 = Conv2D(start_filters * 2, (3, 3), activation="relu", padding="same")(conv2)
    pool2 = MaxPooling2D((2, 2))(conv2)
    pool2 = Dropout(0.5)(pool2)

    # 25 -> 12
    conv3 = Conv2D(start_filters * 4, (3, 3), activation="relu", padding="same")(pool2)
    conv3 = Conv2D(start_filters * 4, (3, 3), activation="relu", padding="same")(conv3)
    pool3 = MaxPooling2D((2, 2))(conv3)
    pool3 = Dropout(0.5)(pool3)

    # 12 -> 6
    conv4 = Conv2D(start_filters * 8, (3, 3), activation="relu", padding="same")(pool3)
    conv4 = Conv2D(start_filters * 8, (3, 3), activation="relu", padding="same")(conv4)
    pool4 = MaxPooling2D((2, 2))(conv4)
    pool4 = Dropout(0.5)(pool4)
```

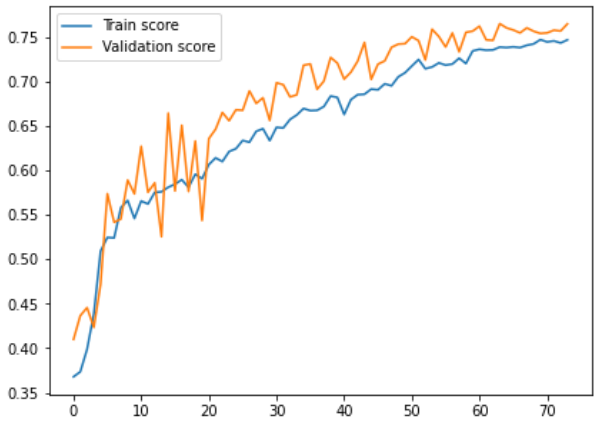
```
# Middle
convm = Conv2D(start_filters * 16, (3, 3), activation="relu", padding="same")(pool4)
convm = Conv2D(start_filters * 16, (3, 3), activation="relu", padding="same")(convm)
# 6 -> 12
deconv4 = Conv2DTranspose(start_filters * 8, (3, 3), strides=(2, 2), padding="same")(convm)
uconv4 = concatenate([deconv4, conv4])
uconv4 = Dropout(0.5)(uconv4)
uconv4 = Conv2D(start_filters * 8, (3, 3), activation="relu", padding="same")(uconv4)
uconv4 = Conv2D(start_filters * 8, (3, 3), activation="relu", padding="same")(uconv4)
# 12 -> 25
deconv3 = Conv2DTranspose(start_neurons * 4, (3, 3), strides=(2, 2), padding="valid")(uconv4)
uconv3 = concatenate([deconv3, conv3])
uconv3 = Dropout(0.5)(uconv3)
uconv3 = Conv2D(start_neurons * 4, (3, 3), activation="relu", padding="same")(uconv3)
uconv3 = Conv2D(start_neurons * 4, (3, 3), activation="relu", padding="same")(uconv3)
# 25 -> 50
deconv2 = Conv2DTranspose(start_neurons * 2, (3, 3), strides=(2, 2), padding="same")(uconv3)
uconv2 = concatenate([deconv2, conv2])
uconv2 = Dropout(0.5)(uconv2)
uconv2 = Conv2D(start_neurons * 2, (3, 3), activation="relu", padding="same")(uconv2)
uconv2 = Conv2D(start_neurons * 2, (3, 3), activation="relu", padding="same")(uconv2)
# 50 -> 101
deconv1 = Conv2DTranspose(start_neurons * 1, (3, 3), strides=(2, 2), padding="valid")(uconv2)
uconv1 = concatenate([deconv1, conv1])
uconv1 = Dropout(0.5)(uconv1)
uconv1 = Conv2D(start_neurons * 1, (3, 3), activation="relu", padding="same")(uconv1)
uconv1 = Conv2D(start_neurons * 1, (3, 3), activation="relu", padding="same")(uconv1)
#uconv1 = Dropout(0.5)(uconv1)
output_layer = Conv2D(1, (1,1), padding="same", activation="sigmoid")(uconv1)
```

Mô hình sau đó được compile với đầu vào là ảnh địa chấn $101 \times 101 \times 1$ và đầu ra là mask sau khi đi qua mạng. Hàm loss được dùng là “binary cross entropy”, optimizer là Adam và metric là metric của cuộc thi [Insert]. Cuối cùng, mô hình được huấn luyện với tập train và tập valid đã được chia. Em có dùng thêm các callback API như ReduceLROnPlateau để giảm learning rate và EarlyStopping để dừng việc huấn luyện sau khi qua nhiều epoch mà score không tăng.

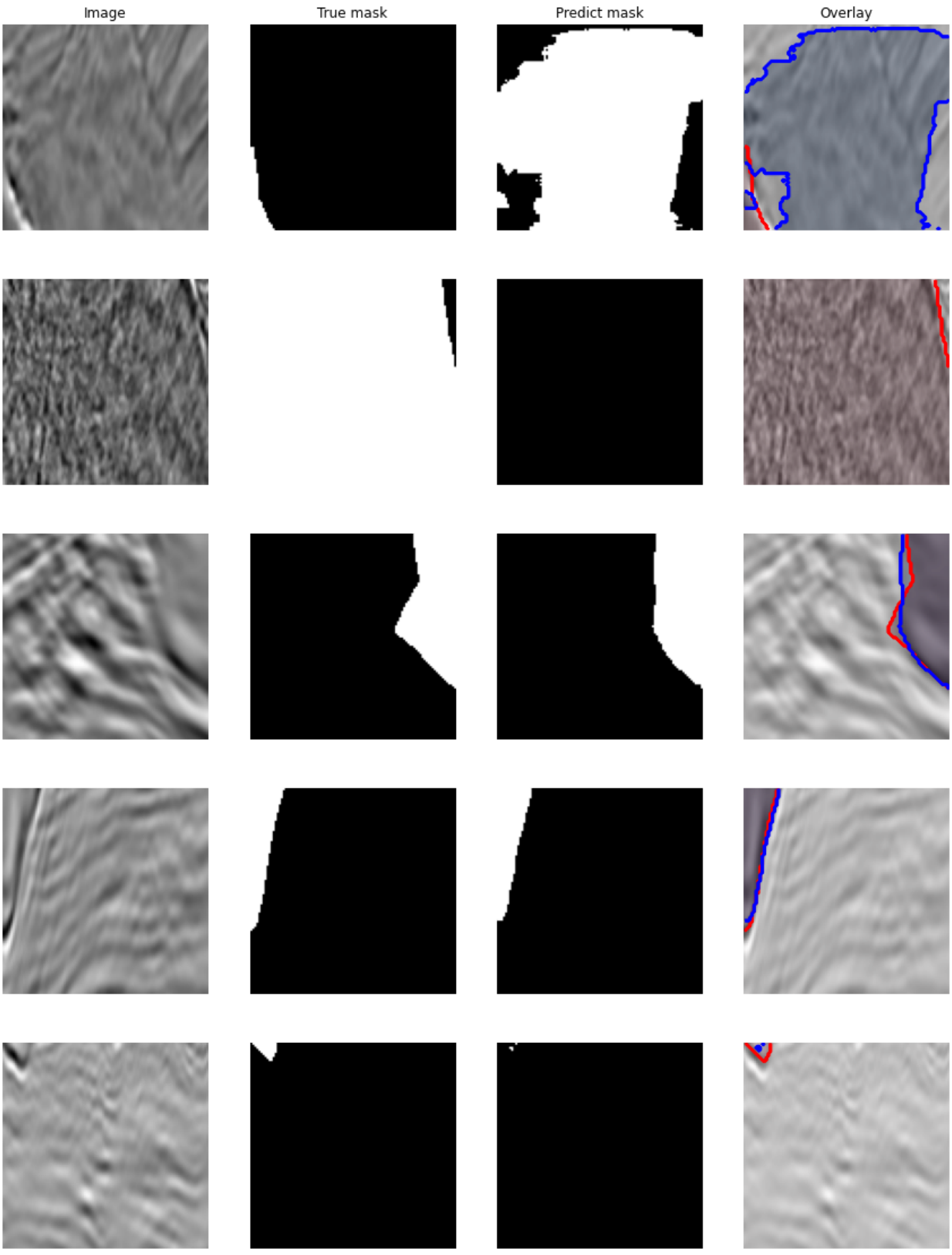
Kết quả

Trực quan hóa loss và score của mạng:

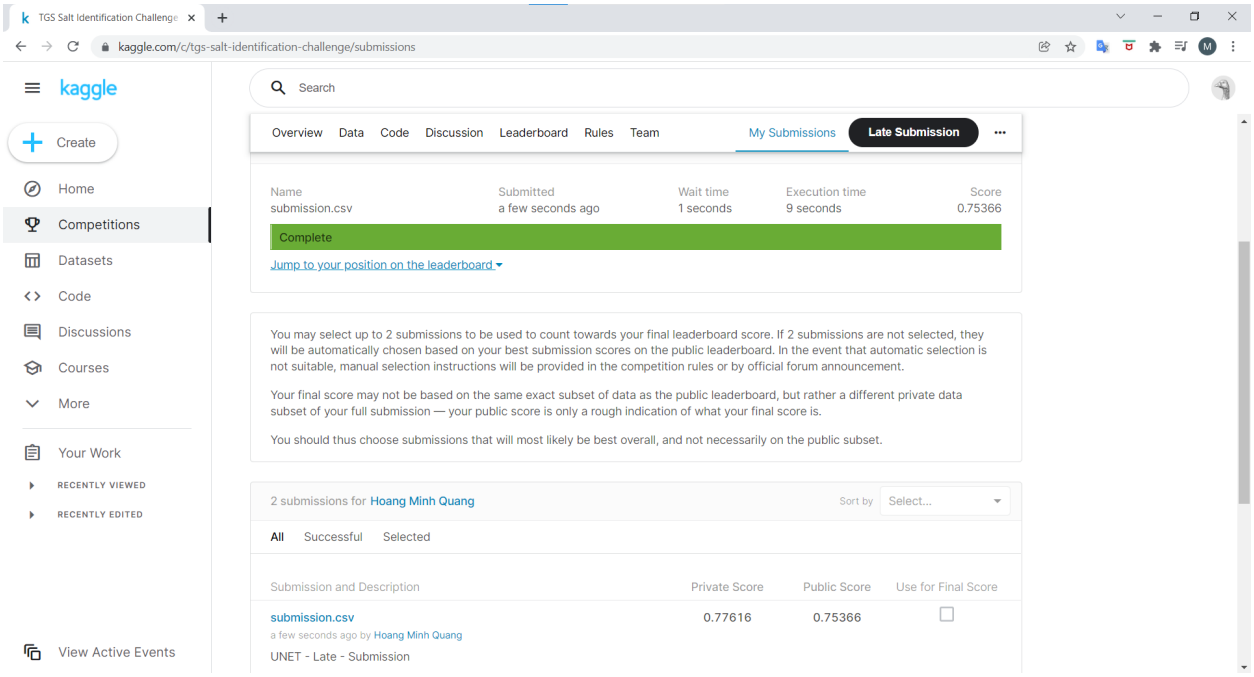




Dự đoán mask của mạng so với mask thực tế được cung cấp:



Điểm nộp trên Kaggle:



U-Net với khối residual

Ngoài ra, trong lĩnh vực thị giác máy còn có mạng ResNet với các khối residual giúp mạng tích chập trở nên “sâu” hơn và mang lại hiệu quả tốt hơn. Do đó, em có xây dựng thêm mạng U-Net và thay thế các khối tích chập thành các khối phần dư (residual).

Thông thường, đầu vào x khi đi qua 1 khối như viền nét đứt (bên trái hình dưới) sẽ được ánh xạ lý tưởng muốn học được là $f(x)$ và làm đầu vào cho hàm kích hoạt. Nhưng trong khối phần dư, đầu vào x sẽ lại được thêm vào ánh xạ $f(x)$ thành $f(x) + x$ rồi mới đi qua hàm kích hoạt

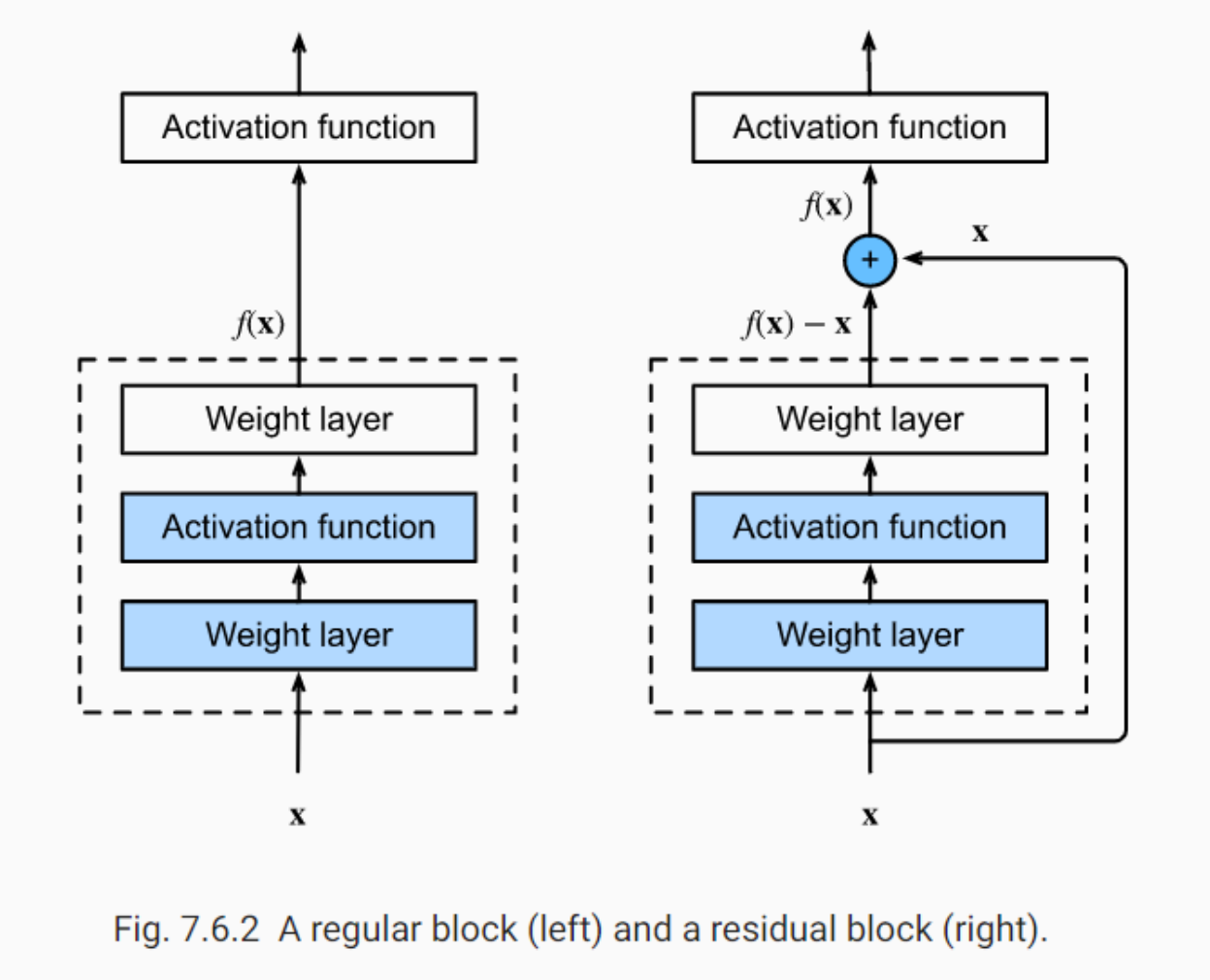


Fig. 7.6.2 A regular block (left) and a residual block (right).

Nguồn: https://d2l.ai/chapter_convolutional-modern/resnet.html

1 khối phần dư của mạng ResNet có thiết kế như hình dưới. Nó bao gồm 2 lớp tích chập 3×3 với cùng số kênh đầu ra (channels/ filters). Lớp tích chập đầu tiên được theo sau bởi lớp chuẩn hóa theo batch (Batch Norm) và hàm kích hoạt “ReLU”, trong khi lớp tích chập thứ 2 chỉ có lớp BatchNorm ở sau. Ta đưa đầu vào x qua khối tích chập và cộng vào chính nó hoặc cộng với nó sau khi đi qua lớp Conv 1×1 để điều chỉnh số kênh rồi mới đưa qua hàm kích hoạt “ReLU” cuối cùng.

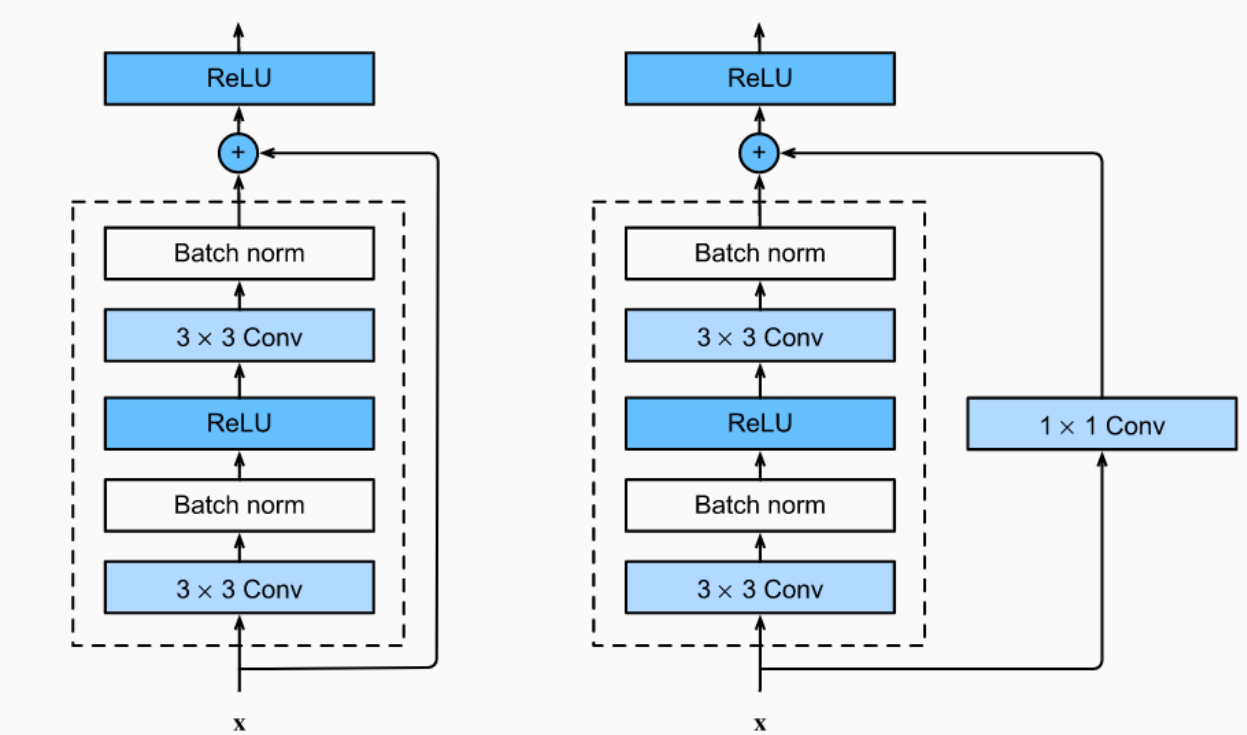
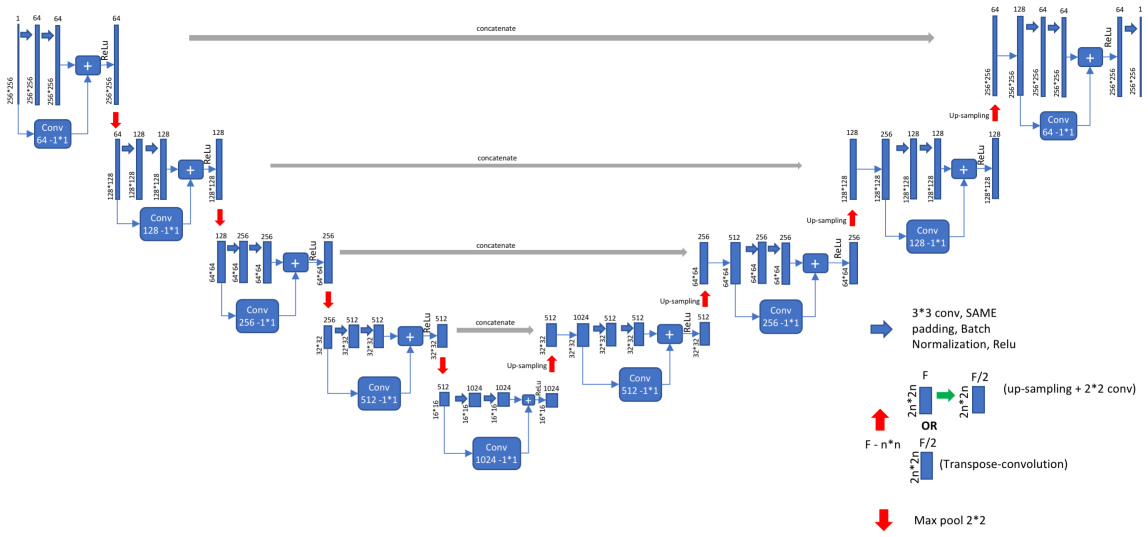


Fig. 7.6.3 ResNet block with and without 1×1 convolution.

Sau đó, chúng ta có thể thay đổi các khối tích chập trong mạng U-Net thành các khối phần dư. Theo em, chúng ta có thể thay đổi số lượng khối phần dư ở mỗi tầng để tăng thêm độ phức tạp và khả năng biểu diễn của mạng. Mạng U-Net sau khi thay các khối tích chập thành khối phần dư có thể có kiến trúc như sau:



Nguồn tham khảo: https://medium.com/@nishanksingla/unet-with-resblock-for-semantic-segmentation-dd1766b4ff66

Code

Đầu tiên, em xây dựng khối tích chập và khối phần dư như ở Hình trên. Khối tích chập có thể xây dựng bao gồm 1 lớp tích chập 3×3 với lớp chuẩn hóa theo batch BatchNormalization theo sau và tùy chọn hàm kích hoạt ReLU . Sau khi đã xây dựng xong khối tích chập, việc xây dựng khối phần dư cũng rất đơn giản. Em cho đầu vào (input) qua khối tích chập đầu tiên (có hàm kích hoạt ReLU), tiếp tục cho qua khối tích chập thứ 2 (không có hàm kích hoạt ReLU) rồi thêm lại chính đầu vào và đưa qua hàm ReLU cuối cùng.

```python
def convolution_block(x, filters, size, strides=(1,1), padding='same', activation=True):
    x = Conv2D(filters, size, strides=strides, padding=padding)(x)
    x = BatchNormalization()(x)
    if activation == True:
        x = Activation('relu')(x)
    return x
```

```python
def residual_block(blockInput, num_filters=16):
    x = convolution_block(blockInput, num_filters, (3,3) )
    x = convolution_block(x, num_filters, (3,3), activation=False)
    x = Add()([x, blockInput])
    x = Activation('relu')(x)
    return x
```

Đến với kiến trúc mạng, tại mỗi tầng, em thay 2 lớp tích chập (ở kiến trúc mạng U-Net) thành 1 lớp tích chập 3×3 và 2 khối phần dư. Các lớp tích chập em đều dùng padding=”same” để giữ nguyên kích thước của feature map ở mỗi tầng ở cả bên phần bên trái (phần thu hẹp - encoder) và phần bên phải (phần mở rộng - decoder). Từ đó, sẽ không cần cắt feature map từ phần bên trái để ghép vào phần bên phải như ở kiến trúc được đề xuất. Tuy nhiên, cũng giống code ở phần mạng U-Net, có 2 tầng trong phần decoder, để cùng kích thước với ảnh được nối, lớp Conv2DTranspose sẽ có mode=”valid”.

```
def build_model(input_layer):
    start_filters = 16
    DropoutRatio = 0.5

    # Encoder - Layer 1: 101 -> 50
    conv1 = Conv2D(start_filters * 1, (3, 3), activation=None, padding="same")(input_layer)
    conv1 = residual_block(conv1,start_filters * 1)
    conv1 = residual_block(conv1,start_filters * 1)
    pool1 = MaxPooling2D((2, 2))(conv1)
    pool1 = Dropout(DropoutRatio/2)(pool1)

    # Encoder - Layer 2: 50 -> 25
    conv2 = Conv2D(start_filters * 2, (3, 3), activation=None, padding="same")(pool1)
    conv2 = residual_block(conv2,start_filters * 2)
    conv2 = residual_block(conv2,start_filters * 2)
    pool2 = MaxPooling2D((2, 2))(conv2)
    pool2 = Dropout(DropoutRatio)(pool2)

    # Encoder - Layer 3: 25 -> 12
    conv3 = Conv2D(start_filters * 4, (3, 3), activation=None, padding="same")(pool2)
    conv3 = residual_block(conv3,start_filters * 4)
    conv3 = residual_block(conv3,start_filters * 4)
    pool3 = MaxPooling2D((2, 2))(conv3)
    pool3 = Dropout(DropoutRatio)(pool3)

    # Encoder - Layer 4: 12 -> 6
    conv4 = Conv2D(start_filters * 8, (3, 3), activation=None, padding="same")(pool3)
    conv4 = residual_block(conv4,start_filters * 8)
    conv4 = residual_block(conv4,start_filters * 8)
    pool4 = MaxPooling2D((2, 2))(conv4)
    pool4 = Dropout(DropoutRatio)(pool4)
```

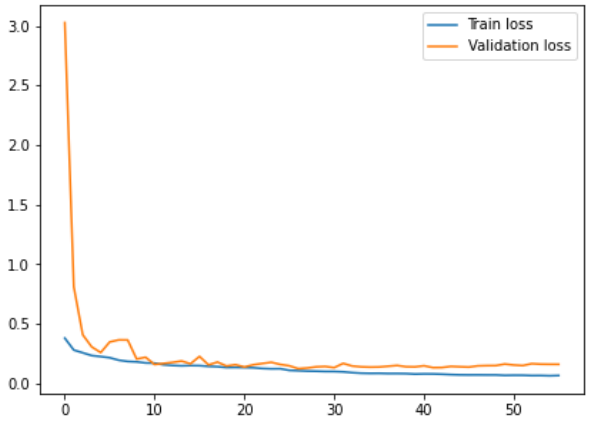
```
# Bottom
convb = Conv2D(start_filters * 16, (3, 3), activation=None, padding="same")(pool4)
convb = residual_block(convb,start_filters * 16)
convb = residual_block(convb,start_filters * 16)

# Decoder- Layer 4: 6 -> 12
deconv4 = Conv2DTranspose(start_filters * 8, (3, 3), strides=(2, 2), padding="same")(convb)
uconv4 = concatenate([deconv4, conv4])
uconv4 = Dropout(DropoutRatio)(uconv4)

uconv4 = Conv2D(start_filters * 8, (3, 3), activation=None, padding="same")(uconv4)
uconv4 = residual_block(uconv4,start_filters * 8)
uconv4 = residual_block(uconv4,start_filters * 8)

# Decoder - Layer 3: 12 -> 25
deconv3 = Conv2DTranspose(start_filters * 4, (3, 3), strides=(2, 2), padding="valid")(uconv4) # 12*2<25 --> 'valid' mode
uconv3 = concatenate([deconv3, conv3])
uconv3 = Dropout(DropoutRatio)(uconv3)

uconv3 = Conv2D(start_filters * 4, (3, 3), activation=None, padding="same")(uconv3)
uconv3 = residual_block(uconv3,start_filters * 4)
uconv3 = residual_block(uconv3,start_filters * 4)
```

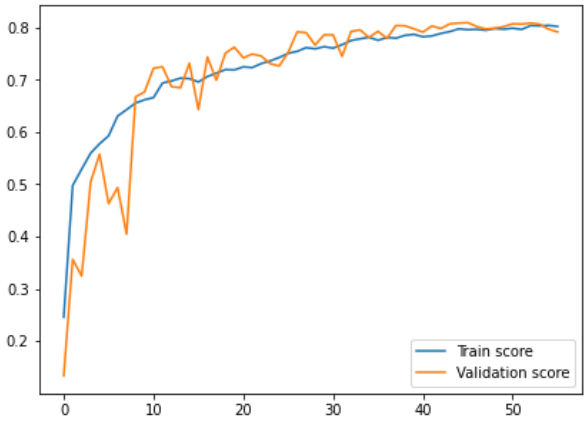
```
# Decoder - Layer 2: 25 -> 50
deconv2 = Conv2DTranspose(start_filters * 2, (3, 3), strides=(2, 2), padding="same")(uconv3)
uconv2 = concatenate([deconv2, conv2])

uconv2 = Dropout(DropoutRatio)(uconv2)
uconv2 = Conv2D(start_filters * 2, (3, 3), activation=None, padding="same")(uconv2)
uconv2 = residual_block(uconv2,start_filters * 2)
uconv2 = residual_block(uconv2,start_filters * 2)

# Decoder - Layer 1: 50 -> 101
deconv1 = Conv2DTranspose(start_filters * 1, (3, 3), strides=(2, 2), padding="valid")(uconv2) # 50*2<101 --> 'valid' mode
uconv1 = concatenate([deconv1, conv1])

uconv1 = Dropout(DropoutRatio)(uconv1)
uconv1 = Conv2D(start_filters * 1, (3, 3), activation=None, padding="same")(uconv1)
uconv1 = residual_block(uconv1,start_filters * 1)
uconv1 = residual_block(uconv1,start_filters * 1)

#output_layer = Conv2D(1, (1,1), padding="same", activation="sigmoid")(uconv1)
output_layer_noActi = Conv2D(1, (1,1), padding="same", activation=None)(uconv1)
output_layer =  Activation('sigmoid')(output_layer_noActi)

return output_layer
```

Phần code compile và huấn luyện mô hình cũng giống với mạng U-Net.

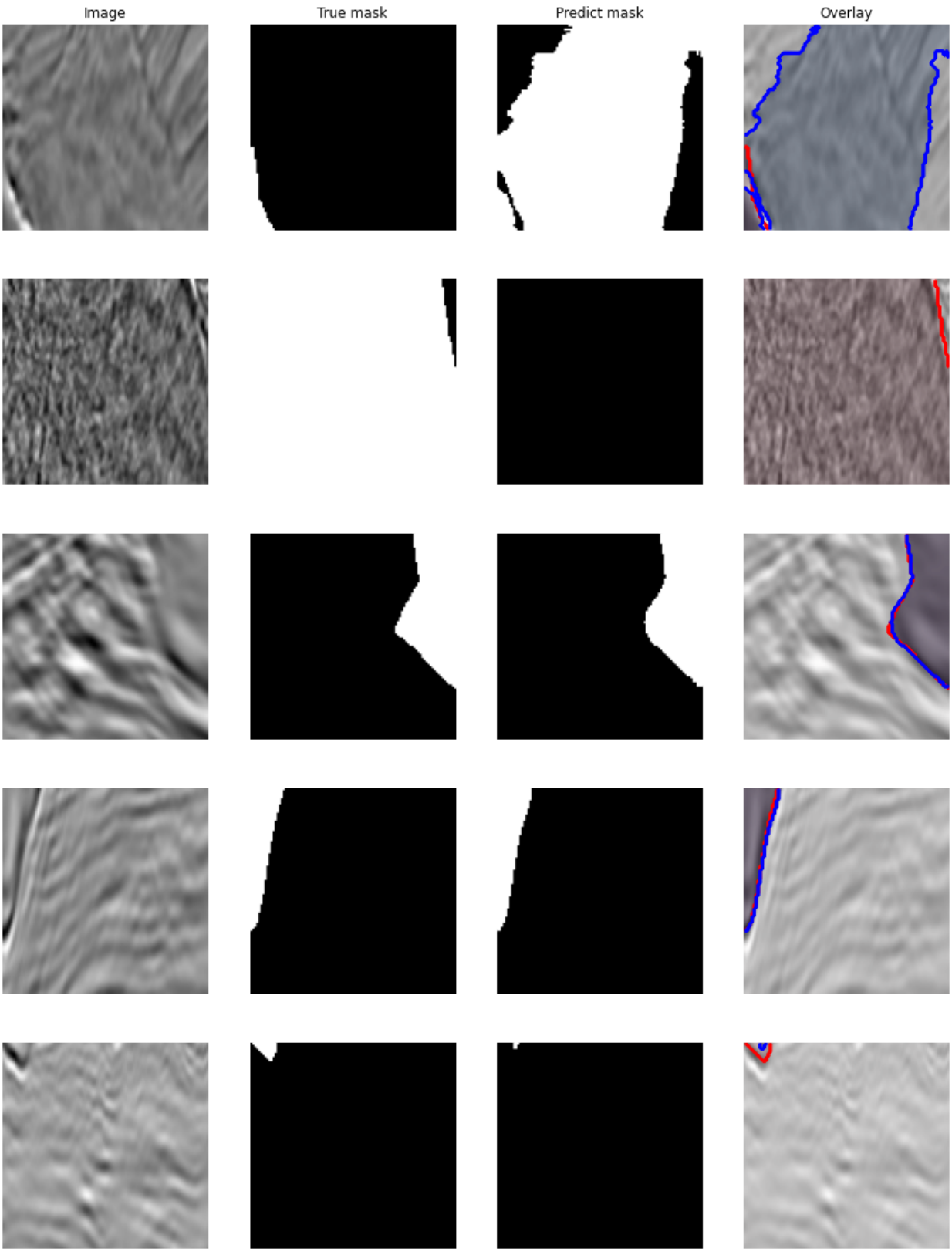
Kết quả

Trực quan hóa loss và score của mạng:

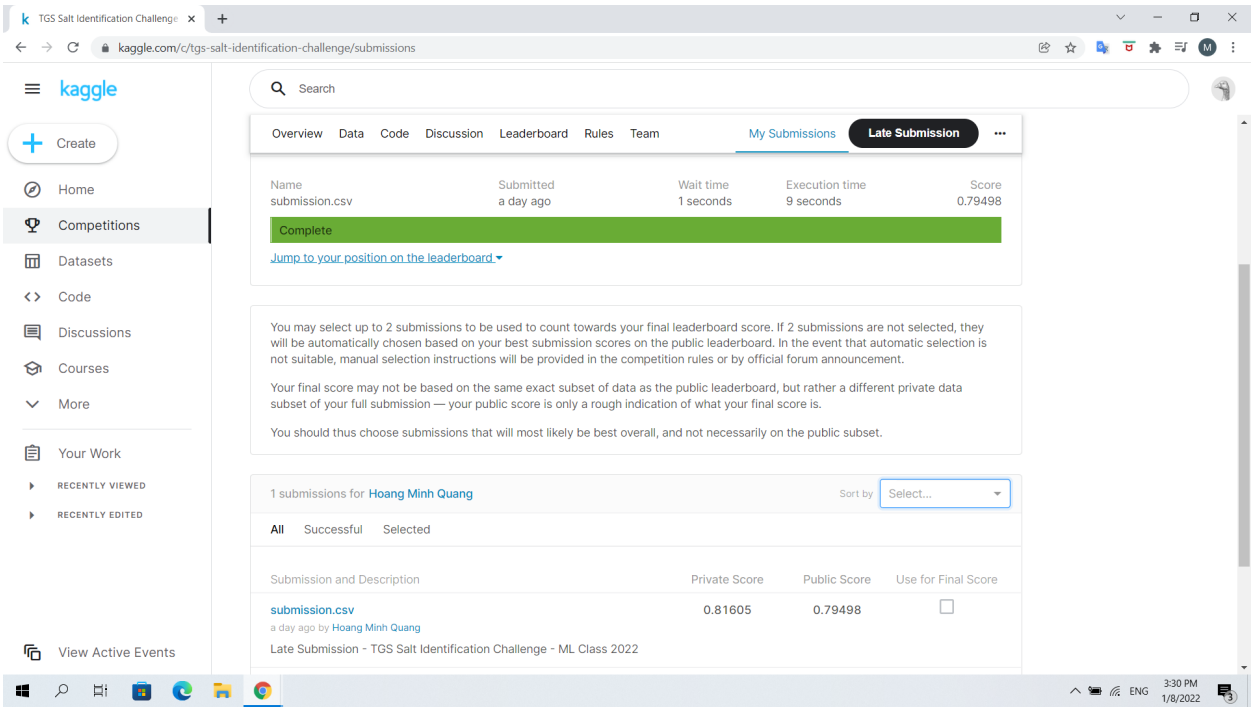




Dự đoán mask của mạng so với mask thực tế:



Điểm nộp trên Kaggle:



Link Code

https://colab.research.google.com/drive/1hKZWB_HvkSaeRNiB_kYz1yNBn_pdZkjc?usp=sharing