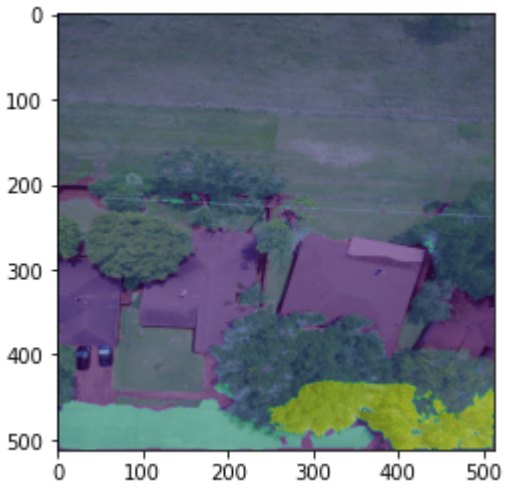
Kaggle Competition: Segment features around residential buildings in UAV images of flooded areas taken in Houston after Hurricane Harvey.

Aspram GRIGORYAN, Bryan ATOK A KIKI, Jing DAI
Name On Kaggle: We Got It Right?

Problem Statement: The task is to design and implement a deep learning model in order to perform the automatic segmentation of such images. In particular, the main objective of this challenge is to segment images acquired by a small UAV (sUAV) in the area of Houston, Texas. These images were acquired in order to assess the damages on residential and public properties after Hurricane Harvey. In total there are 25 categories of segments (e.g. roof, trees, pools etc.).

More concretely, the goal of this semantic image segmentation task is to label each pixel of an image with a corresponding class of what is being represented out of 25 possible categories. The expected output itself is an image.



In order to tackle this task we have focused on a very successful architecture, U-Net, which has proven to be performing well on similar image-segmentation tasks.

Image Pre-processing approaches:

I. Image Pre-processing approaches:

To perform the image pre-processing, we used the `torchvision.transforms` module. We decided to use different transformations on the train images compared to the validation and the test sets.

Our choice of using image augmentation techniques is motivated by the fact that we wanted to avoid overfitting and also enrich the train dataset with more images. As the images have been taken by UAVs, the horizontal and vertical axes can be reversed without losing information.

Please find below a summary the image augmentation techniques implemented:

	Train set	Validation set	Test set
`transforms.Resize`	Yes	Yes	Yes
`transforms.Normalize`	Yes	Yes	Yes
`transforms.RandomVerticalFlip`	Yes	No	No
`transforms.RandomHorizontalFlip`	Yes	No	No

During the project, we chose different image sizes on our model taking into account the GPU requirements and time constraints. The height and the width of the images ranged from 200px to 500px.

The normalization of the images was based on the statistics on the train images.

Then we applied random vertical and horizontal flips with a probability of 0.5 for image augmentation purposes

II. Modeling choosing

Over the project, we used different types of convolutional networks for image segmentation. The model that reported the best result was U-NET. The U-NET is a convolutional neural network created by Olaf Ronneberger, Philipp Fischer, Thomas Brox in 2015[1]. It stems from the fully conventional network created by Long, Shelhamer, and Darrell in 2014[2].

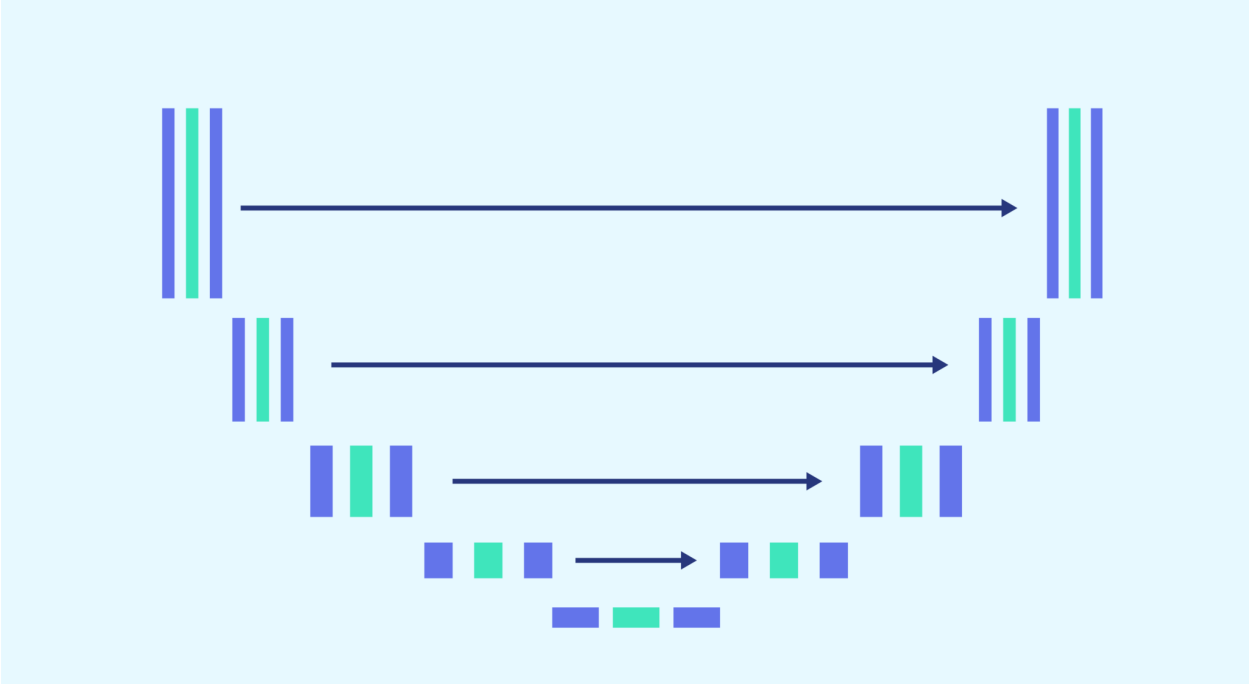
Originally developed for image segmentation in the biomedical field, it then has been implemented for various other applications which is the reason why we used this model in our project.

Architecture : The implemented U-NET architecture contains two paths. First path is the contraction path (also called as the encoder) which is used to capture the context in the image. The encoder is just a

[1] Ronneberger, Olaf; Fischer, Philipp; Brox, Thomas (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation"

[2] Long, J.; Shelhamer, E.; Darrell, T. (2014). "Fully convolutional networks for semantic segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **39**(4): 640–651.

traditional stack of convolutional and max pooling layers. The second path is the symmetric expanding path (also called as the decoder) which is used to enable precise localization using transposed convolutions.



To build the U-NET architecture, we have used the PyTorch Module class and were able to inherit from it many models and sub-models.

Firstly, we initialize a `double_conv` method which involves a convolutional layer followed by a batch normalization operation, a ReLU activation, a second convolutional layer and a final ReLU activation function.

We also define a `double_conv_downs` method that takes as input a tuple (i.e., channels) of channel dimensions. The first value denotes the number of channels in our input image, and the subsequent numbers gradually double the channel dimension. Using PyTorch's ModuleList functionality we perform the `double_conv` on each input. This corresponds to the encoder procedure.

In addition, we define a list of upsampling blocks that use the ConvTranspose2d layer to upsample the spatial dimension (i.e., height and width) of the feature maps by a factor of 2. This step is performed through the `up_trans` method.

We also initialize a `max_pool_2x2` layer, which reduces the spatial dimension (i.e., height and width) of the feature maps by a factor of 2.

To get better precise locations, at every step of the decoder we use skip connections by concatenating the output of the transposed convolution layers with the feature maps from the Encoder at the same level. In the `forward` step, we concatenate the i-th intermediate feature map from the encoder with our current output x from the upsampling block. Since we need to ensure that the spatial dimensions of encoded features and x match, we use resizing.

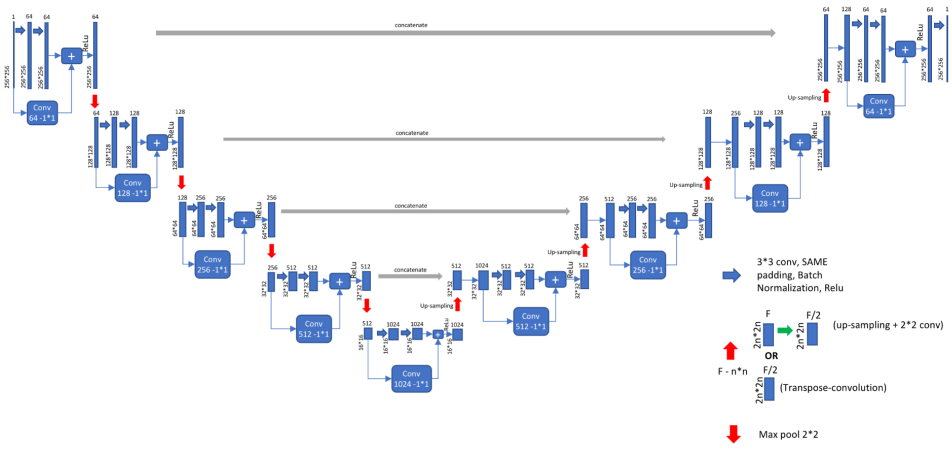
Finally, we pass the concatenated output through our i-th decoder function.

Besides Unet, we also attempted other learning algorithms as below:

SegNet: We also created a SegNet model which is a semantic segmentation model similar to the U-NET model. It includes an encoder and a decoder, which are both built with blocks of Convolution, Batch Normalisation, ReLU activation function and a Pooling layer.

ResUNET: We implemented a variation of the U-NET model using residual blocks. Our motivation came from the fact that increasing the number of layers or adding augmentation techniques did not improve the accuracy of our predictions due to overfitting.

Illustration of a ResUNET:



MASK R CNN: Starting by clone a repository that contains some of the building code blocks for implementing Mask RCNN, we imported COCO weights for the model and tried to set up our custom data class(instead of using the COCO dataset), however, the training result is not as satisfying.

III. Model and other parameters

A. Model

Model Training:

- Structuring the data-loading pipeline
- Initializing the model and training parameters
- Defining the training loop
- Visualizing the training and test loss curves

Model is compiled with Adam optimizer and we use the cross entropy loss function.

Results on the training set are relatively better than those on the validation set which implies the model suffers from overfitting. One obvious reason could be the small number of images used to train the model.

B. Optimizer

OPTIMIZER CHOOSING	SCORE	PLOT LOSS
Adam (with learning rate schedule)	**0.63**	**Appendix A**
Adam (without learning rate schedule)	**0.70**	**Appendix B**
Dice loss	**0.58**	**Appendix C**

IV. Final result

Our final result observed on Kaggle was 70.2%. This score has been attained with the U-Net model. Please see below the summary of this submission:

Hyperparameters	
Image height	350
Image width	350
Batch size	16

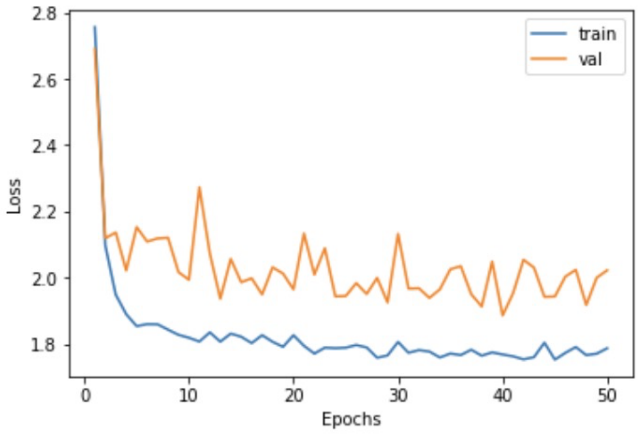
Optimizer	Adam
Learning rate	0,001
Epoch	80
Image augmentation techniques	Normalization; RandomHorizontalFlip; RandomVerticalFlip

You can see below the result for different parameters for the model implemented:

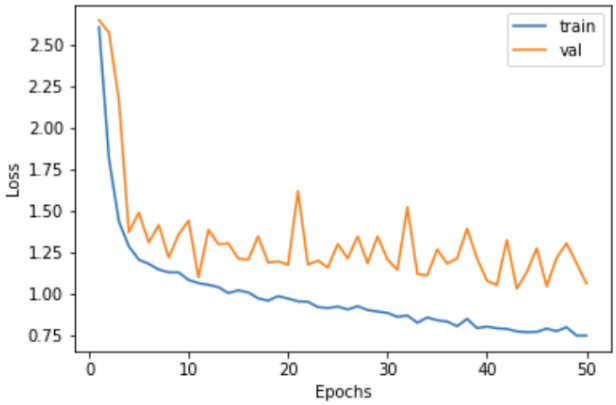
Model	**U-Net**	**U-Net**	**U-Net**	**U-Net**	**U-Net**	**U-Net**	**ResUnet**	**SegNet**
Image Height	400	310	350	350	240	350	320	350
Image Width	500	410	350	350	240	350	320	350
Batch size	4	8	16	16	32	16	10	16
Learning rate	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001
# Epochs	25	30	50	98	43	80	50	50
Accuracy								
Train set	66,65%	69,75%	75,35%	81,40%	75,13%	75,13%	69,29%	66,65%
Validation set	57,12%	61,22%	65,10%	65,66%	64,27%	64,27%	62,91%	57,12%
Kaggle Submission	**65,03%**	**67,47%**	**69,40%**	**66,51%**	**69,09%**	**70,20%**	**62,63%**	**66,58%**

V. Appendix

A. Plot of loss in train and validation set with Adam optimizer and learning rate schedule



B. Plot of loss in train and validation set with Adam optimizer



C. Plot of loss in train and validation set with dice loss function