Class Project 2
CS-433 Machine Learning

Mortadha Abderrahim[1], Ana Lucia Carrizo[2], Louay Najar[3]
Swiss Federal Institute of Technology Lausanne

Abstract— **The goal of Project 2 is to apply the concepts learned in the Machine Learning course on a real-world dataset. The dataset consists of satellite images of roads, and it was provided in the ML-Road Segmentation Challenge on AIcrowd. Machine learning methods such as exploratory data analysis, feature engineering and different neural net models will be applied to understand, analyze and generate predictions.**

I. Introduction

THE objective of the project is to identify, as accurately as possible, the pixels that are part of a road for a given satellite image obtained from GoogleMaps. Our task is a classic semantic segmentation computer vision problem, from which there are many solutions found in the literature. In our case, we decided to explore two models: a classical solution using a convolutional neural network, and another model using U-Nets. Given how data intensive this task resulted, we decided to take advantage of the GPUs available with Google Colab to run our models.

II. Data Preprocessing

We started by doing a thorough exploration of our data. Secondly, we proceeded by generating new images using classic data augmentation techniques. After this, we proceeded to test different machine learning models seen in class and understand why they were under/over performing.

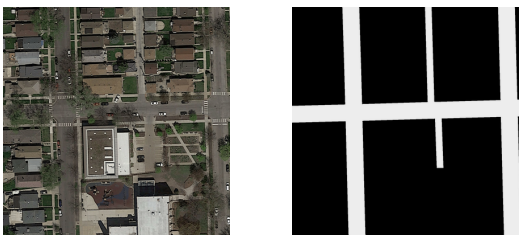
A. Exploratory Data Analysis

Our data comprises a training set consisting of 100 images of size 400x400, and their respective ground-truths. We also have a test set that consists of 50 images of size 608x608, and no groundtruths were provided. The class labels have two possible values: 1 and 0, which represent a road or the background, respectively.

The fact that the training and testing set have different sizes was one of the big challenges of this project. In order to be able to predict results compatible with the format expected to make submissions on AIcrowd, we decided to work with patches. Since 16 is the greatest common divider of 400 and 608, it makes the perfect size for our patches. We crop the images from both the training set and the test set into patches of size 16x16.

[1]Msc. Data Science, Swiss Federal Institute of Technology Lausanne, e-mail: mortadha.abderrahim@epfl.ch.
[2]Msc. Data Science, Swiss Federal Institute of Technology Lausanne, e-mail: ana.carrizodelgado@epfl.ch.
[3]Msc. Data Science, Swiss Federal Institute of Technology Lausanne, e-mail: louay.najar@epfl.ch.

In Fig. 1 we can see an example of our training data:



(a) Satellite Image (b) Ground Truth Image

Fig. 1: Example of training Dataset

B. Data Augmentation

The fact that we have such a small training set (only 100 images) is not enough to make our model accurately predict if a pixel belongs to a road or not. If we do not perform data augmentation, our model is going to overfit on the training set, and we will not obtain good results on the test set. For this reason, we need to do some intensive work preprocessing our images. We proceed to explain all the data augmentation mechanisms performed:

1. *Random Rotation*: our model chooses a random angle between 0 and π degrees, to make sure the model performs consistently no matter the orientation of the roads.
2. *Blurring*: we added some salt&pepper noise, which will add robustness to noise.
3. *Changing Brightness*: we vary the image's light levels following a uniform law.
4. *Horizontal Flipping*: we apply an horizontal mirror transformation to the image.
5. *Vertical Flipping*: we apply a vertical mirror transformation to the image.

When performing the data augmentation, all the operations are performed both on the images and their ground-truth except for the Blurring and the change of Brightness. We tried implementing other data augmentation processes such as width and height shifts, but they did not improve the performance of our models.

III. Benchmarking measures

When we analysed our data, we found out that our data set is actually not balanced (or symmetric), meaning that the average number of pixels per image being identified as background is larger than the

number of pixels identified as road. Thus, we chose to measure the performance of our model according to its F1 score as well as the accuracy.

IV. Models and Methods

Image segmentation is a very important part of Image processing, Neural Networks have proven to be very efficient in this domain, specifically Convolutional Neural Networks. We thus chose to first implement a Convolutional Neural Network architecture. Then we implemented a U-net architecture, which is essentially an improved version of CNN.

A. Implementation Methods

1. **Activation Functions**: they are used to determine the output of a neural network. In both of our models we use nonlinear activation functions that make our model better at generalization. In this project we used Leaky ReLU, ReLU and the Sigmoid function as activations. In the following sections we will detail which models used which activation functions.
2. **MaxPooling**: creates a down-sampled feature map. MaxPooling consists in calculating the maximum value for patches of a feature map.
3. **Dropout**: the dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time. It helps prevent overfitting.
4. **Batch Normalization**: adding a batch normalization layer will make our neural network faster and more stable by adding extra layers to our model.
5. **Kernel Regularizer**: it applies a penalty on the layer's kernel. These layers are summed into the loss function that the network optimizes.

B. Keras Callbacks

1. **Early Stopping**: we monitor the F1 score with a patience of 9.
2. **Reduce Learning Rate on Plateau**: we monitor the loss with a patience of 3.

C. Optimizer

An optimizer is one of the two arguments required for compiling a keras model. We chose ADAM as our optimizer with a learning rate of 1e-4. It consists of performing stochastic gradient descent based on adaptive estimation of first-order and second-order moments.

D. Metrics

1. **Loss Function**: we chose as loss function binary cross-entropy, which compares each of the predicted probabilities to our actual class output (0 or 1). It then computes the score that penalizes the probabilities based on the distance from the expected value.
2. **Accuracy**: it's the ratio of accurately predicted observations over all observations.
3. **Precision**: quantifies the number of positive class predictions that actually belong to the positive class.
4. **Recall**: quantifies the number of positive class predictions made out of all positive examples in the dataset.
5. **F1 Score**: it provides a single score that balances out the concerns of precision and recall.

E. Traditional Convolutional Neural Network

The first half of the network downsamples the input. The input consists of patches of 16x16 pixels with three channels, since we are dealing with color images. We obtain these 16x16 patches by cropping our 400x400 pixels images and feeding them into the network. The second half of the network consists on the upsampling of the inputs, and a final classification layer.

Our neural net uses the following parameters to obtain optimal results:

Table I: Parameters for the CNN

Parameter	Value
Leaky ReLU's alpha	0.01
Dropout rate	0.2
Regularization Value	1e-6

1. **First Layer**:
   - 2D Convolution: 5x5 filter with depth of 64.
   - Activation function: Leaky ReLU
   - MaxPooling
   - Dropout
2. **Second Layer**:
   - 2D Convolution: 3x3 filter with depth of 128.
   - Activation function: Leaky ReLU
   - MaxPooling
   - Dropout
3. **Third Layer**:
   - 2D Convolution: 3x3 filter with depth of 128.
   - Activation function: Leaky ReLU
   - MaxPooling
   - Dropout
   - Flatten
4. **Fourth Layer**: fully connected layer (128 nodes).
   - Dense layer
   - Activation function: Leaky ReLU
   - Dropout: with dropout rate of two times the one we were previously using.
5. **Output layer**: we add a dense layer with an L2 regularizer and a sigmoid activation function.

F. U-net

Firstly proposed for biomedical image segmentation, the U-net architecture has proven to be very efficient for my other domains. U-net is a classical fully convolutional network with a symmetrical encoder-decoder structure.

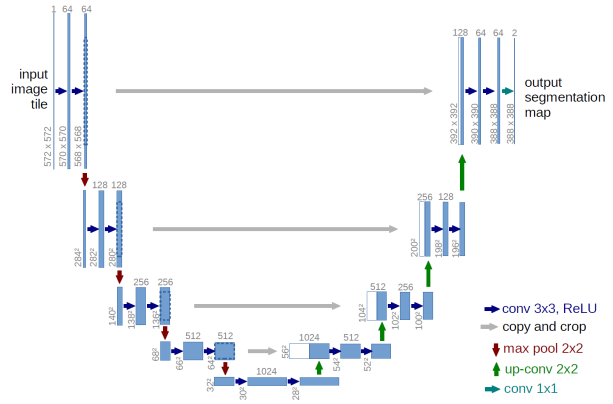


Fig. 2: U-Net Architecture for 572x572 pixels in the lowest resolution.

In Fig. 2 we can see the architecture of the original U-Net. The first path is the contraction path, which is used to capture the context in the image. It is similar to what we have previously seen with the classic CNN, it is essentially just a stack of convolutional and max pooling layers. In the second path is where we perform the upsampling of the input, which is used to enable precise location using transposed convolutions. The parameters for each transpose convolutions are such that, the height and the width of the image are doubled while the depth is halved.

We proceed to describe in detail the downsampling and upsampling paths of our network:

1. **Encoder**
   - *Entry Block*: we perform a convolution proceeded by a Batch Normalization layer and a ReLU activation function.
   - *Contraction path*: the contraction path consists of three blocks. The blocks are identical except from the feature depth. The first block will have a filter size of 64, the second one 128 and the last block 256. Each block consists of two consecutive 3x3 convolutions, followed by a ReLU activation function and Batch Normalization. Finally, each block has a MaxPooling layer with a stride of 2 that downsamples the input.
2. **Decoder**
   - *Expansion path*: the expansion path consists of four blocks that are identical except for the feature depth. The first block will have a filter size of 256, and the rest will have a filter size of 128, 64 and 32, respectively. Each block consists of two consecutive 3x3 transpose convolutions with ReLU activation functions and Batch Normalization, proceeded by an upsampling.
   - *Final layer*: we use a 1x1 convolution with a Sigmoid activation function to obtain the desired predictions.

V. Results

A summary of the performances of all models can be found in the following table:

Table II: F1 Score and Accuracy per Model

	Train F1	Test F1
CNN (without data augmentation)	0.8288	0.721
CNN (with data augmentation)	0.704	0.856
U-Net (without data augmentation)	0.87	0.78 (Validation)
U-Net (with data augmentation)	0.89	0.8(Validation)

VI. Discussion

The pipeline suggested throughout the aforementioned methodology achieves an F1 score of 80 % on the Road Segmentation Challenge. The pipeline is based on the following design choices:

- Using patches solves the problem of differences between the train and test image sizes.
- Several techniques and transformations could be considered for the preprocessing step. For our pipeline, we suggest using classic data augmentation techniques like mirror flipping, both vertically and horizontally, adding noise, and changing the brightness of the pictures. Other methods such as other color transformations or zooming, are worth exploring, but for time considerations we only implement the aforementioned techniques.

Since we get similar accuracies between the train and test set, we conclude that our model is not overfitting, and the model works as it should.

VII. Summary

In this project, we implemented a Machine Learning pipeline to identify the roads from a satellite picture. We found the model to be