Extracting road networks from satellite images with neural networks to contribute to risk mitigation Machine Learning project - Douglas Stumpp

Abstract

Geological hazards can cause significant damage to populations and infrastructure. In order to effectively respond to these hazards, it is crucial to have access to accurate maps and accessibility information. This information can help emergency stakeholders to quickly establish safe evacuation routes and provide aid to affected areas.

In this work, we are exploring the use of convolutional neural networks (CNNs) to automatically extract roads and street networks from satellite images. By leveraging the power of CNNs, we aim to develop a fast and reliable method for producing detailed maps of disaster-affected areas, which can be used to support emergency response and risk management efforts. This work is still in progress, but CNNs seems to be a promising approach for this task.

1. Introduction

Geological hazards such as volcanic eruptions, landslides, earthquakes, and floods pose a significant threat to populations, particularly in developing countries. To mitigate the risks associated with these hazards, it is essential to have access to maps and accessibility information during risk assessments, management, and crisis response.

Remote sensing data, such as satellite images, can be used to gather information about the state of infrastructure in areas that may be affected by geological hazards. However, manually extracting roads and street networks from these images can be time-consuming and error-prone.

To address this challenge, this project proposes to use a convolutional neural network (CNN) to automate the extraction of roads and street networks from satellite images. By treating this task as a computer vision problem, the proposed CNN is able to quickly and accurately extract the necessary information from the images, providing valuable data for risk assessment and crisis response.

2. Data

This work focuses on the DeepGlobe Road Extraction Dataset (Demir et al. 2018), available online[1].

It consists of a total of 8'570 satellite images captured over Thailand, Indonesia, and India. The original dataset has already been split into a training set, a validation set, and a test set. However, the validation set provided on kaggle is missing its labels. Therefore, we recreated another one from the training set (20%), leading to this final split: Training set: 4'981 images (68%); Validation set: 1245 images (17%); Test set: 1'101 images (15%).

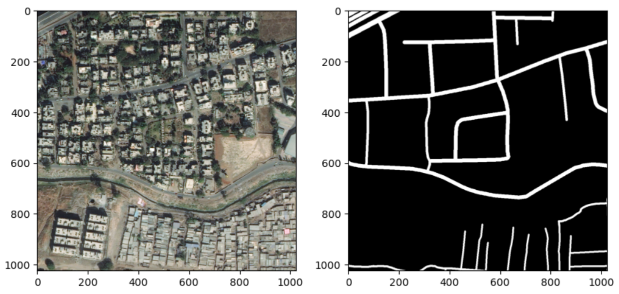


Figure 1. Example of a training instance with satellite image (left) and its associated mask (right).

The original images consist of 3 channels (Red-Green-Blue) made of 19'584 $\times$ 19'584 pixels, that have been downsampled to 1'024 $\times$ 1'204 pixels on the dataset provided on kaggle. Each training instance is associated with a labeled mask, which is an image of the same dimensions with road and non-road pixels (*Figure 1.*). Note that both the satellite images and the masks have the same shape (1024, 1024, 3), where the mask channels are identical for each colorband. Therefore, the task of road extraction is formulated as a binary classification problem, where the neural network must predict (one channel of) the mask, depicting the road network from the satellite image.

[1] https://www.kaggle.com/datasets/balraj98/deepglobe-road-extraction-dataset

3. Methodology

3.1. Data preprocessing

The dataset consists of images and masks with a resolution of 1024 × 1204 pixels. These are organized into folders for the training, validation, and testing sets. The dataset also includes CSV files that provide the pixel classification information and the paths to the images and masks. We used the pandas library to manipulate the data as DataFrames to prepare the association of each pixel's RGB values with its corresponding binary class (*i.e.,* **1 for road pixels and 0 for background pixels**), and to generate the training, validation, and testing sets as described earlier. To prevent RAM overcharge, we downsampled the size of the instances to 256 × 256 pixels before feeding them to the CNN. While we are still working on finding a more effective solution to this issue, the current method has produced reliable and performant results.

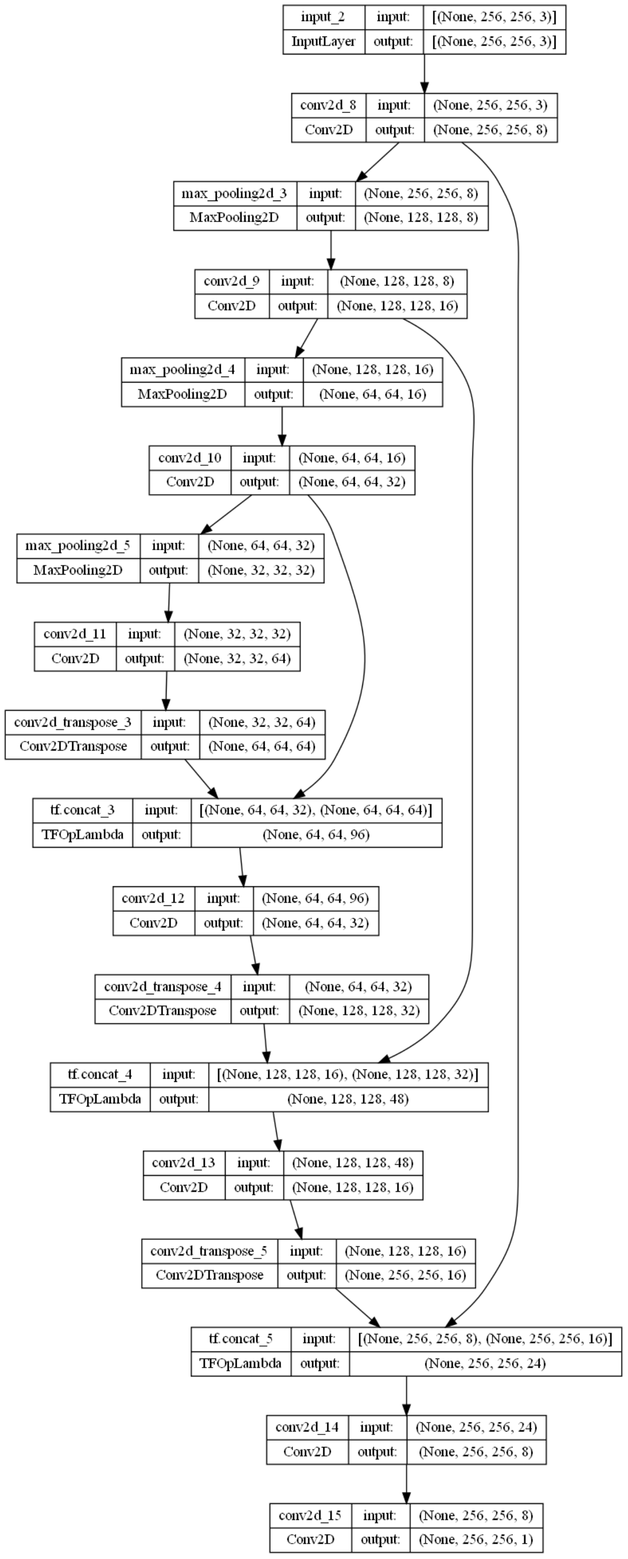
3.2. Convolutional Autoencoder

The convolutional neural network's architecture proposed for this project is taken[2] from the U-net CNN of Yang et al. 2022, who simplified the Convolutional Autoencoder (CAE) of Ronneberger, Fischer, and Brox 2015.

The U-net is composed of a encoding path followed by a decoding path (*Figure 2.*). The encoder consists of a repeated downsampling application using one 2 × 2 convolution kernel with 1 × 1 strides, zero padding, and L1 regularization penalty on the layer's outputs to avoid overfitting. Then follows a Rectified Linear Unit (ReLU) activation function and a 2 × 2 max pooling operation with 2 × 2 stride and valid padding. After each application of a convolutional layer, the results are copied for later skip connections. Such skip connections should offer the integration of local and global features during decoding, such that the output is more reliable. Then, the decoder consists of a repeated upsampling of the feature map using a 2 × 2 deconvolution with 2 × 2 strides, valid padding and L1 regularization, a concatenation with the corresponding cropped feature map from the encoding path (*i.e.,* the copied layers), and a 2 × 2 convolution with 1 × 1 strides and zero padding, followed by a ReLU. Finally, the last layer making the prediction is once again a convolutional layer only using the sigmoïd function to normalize the output between 0 and 1.

Because the sigmoïd function maps any real-valued number to a value between 0 and 1, it can be easily interpreted as a probability. So, for each pixel constituting the output of the sigmoïd function, if it's value is close to 0, it means that the input pixel is more likely to belong to the road class, while if the output is close to 1, it means that the input pixel is more likely to belong to the background class.

[2] related to my master thesis.



input_2 | InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
conv2d_8 | Conv2D | input: (None, 256, 256, 3) | output: (None, 256, 256, 8)
max_pooling2d_3 | MaxPooling2D | input: (None, 256, 256, 8) | output: (None, 128, 128, 8)
conv2d_9 | Conv2D | input: (None, 128, 128, 8) | output: (None, 128, 128, 16)
max_pooling2d_4 | MaxPooling2D | input: (None, 128, 128, 16) | output: (None, 64, 64, 16)
conv2d_10 | Conv2D | input: (None, 64, 64, 16) | output: (None, 64, 64, 32)
max_pooling2d_5 | MaxPooling2D | input: (None, 64, 64, 32) | output: (None, 32, 32, 32)
conv2d_11 | Conv2D | input: (None, 32, 32, 32) | output: (None, 32, 32, 64)
conv2d_transpose_3 | Conv2DTranspose | input: (None, 32, 32, 64) | output: (None, 64, 64, 64)
tf.concat_3 | TFOpLambda | input: [(None, 64, 64, 32), (None, 64, 64, 64)] | output: (None, 64, 64, 96)
conv2d_12 | Conv2D | input: (None, 64, 64, 96) | output: (None, 64, 64, 32)
conv2d_transpose_4 | Conv2DTranspose | input: (None, 64, 64, 32) | output: (None, 128, 128, 32)
tf.concat_4 | TFOpLambda | input: [(None, 128, 128, 16), (None, 128, 128, 32)] | output: (None, 128, 128, 48)
conv2d_13 | Conv2D | input: (None, 128, 128, 48) | output: (None, 128, 128, 16)
conv2d_transpose_5 | Conv2DTranspose | input: (None, 128, 128, 16) | output: (None, 256, 256, 16)
tf.concat_5 | TFOpLambda | input: [(None, 256, 256, 8), (None, 256, 256, 16)] | output: (None, 256, 256, 24)
conv2d_14 | Conv2D | input: (None, 256, 256, 24) | output: (None, 256, 256, 8)
conv2d_15 | Conv2D | input: (None, 256, 256, 8) | output: (None, 256, 256, 1)

Figure 2. Schematic U-net architecture.

3.3. Loss Function

We chose the binary cross-entropy loss function for this binary classification problem. Defined as the negative log-likelihood of the true labels given the predicted probabilities, it directly measures the difference between the predicted probabilities (output of the sigmoïd activation function) and the true labels (0 or 1) of the pixels in the image. Mathematically, it can be written as:

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i)\cdot log(1-p(y_i)), \tag{1}$$

where N is the number of samples, y_i is the true label (0 or 1), and $p(y_i)$ is the predicted probability (output of the sigmoid function). θ being the usual model parameters vector.

The binary cross-entropy loss function has the following properties:

- It is a continuous and differentiable function, which makes it suitable for training a neural network using gradient descent.
- It has a global minimum, which means that it is possible to find the optimal set of model parameters that minimize the loss.
- It penalizes large errors more than small errors, which encourages the model to be more confident in its predictions.
- It is sensitive to the relative order of the predicted probabilities and the true labels, which means that it can accurately measure the difference between the two.

Therefore, measuring the difference between the predicted probabilities $p(y)$ (output of the autoencoder) and the true labels y (0 or 1) of the pixels, will guide the optimization process of the autoencoder during training.

3.4. Optimizer

The optimizer used for our CNN is Adam (Adaptive Moment Estimation), which combines the benefits of the SGD (Stochastic Gradient Descent) and RMSProp optimization algorithms. One advantage of Adam is that it uses adaptive learning rates, which means that the learning rate for each parameter is updated based on past gradient information. This can help the optimization process converge faster and more accurately, making it a well-suited candidate for the use of a convolutional autoencoder for binary classification of RGB images.

4. Results

4.1. Model performance

We trained our CNN for 50 epochs (∼7h) while considering a patience of 3 on the validation loss for early stopping. We also used the validation set for cross-validation and saved the best model and weights. The resulting plots (*Figure 3, 4.*) of the training and validation loss and accuracy over the epochs demonstrate the reliability of these results and, associated with our regularization method and early stopping confirm that it did not suffer from overfitting.

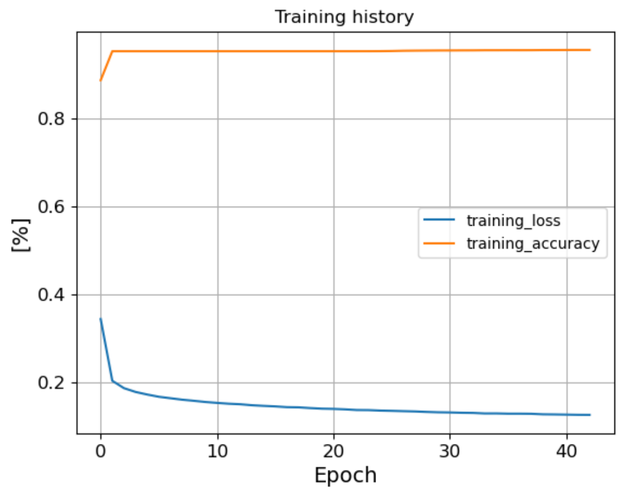


Figure 3. Training loss and accuracy over the epochs.

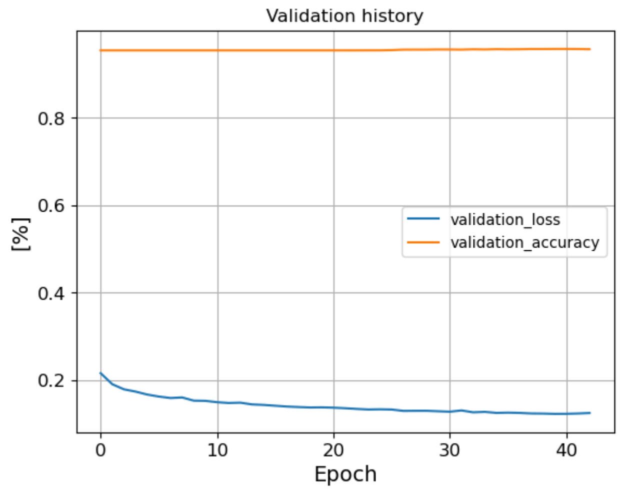


Figure 4. Validation loss and accuracy over the epochs.

4.2. Evaluation metric

To provide a more detailed breakdown of the model's performance, we generate a confusion matrix (*Figure 5.*) on the validation set by comparing the model's predictions on the validation instances and the validation masks (*Figure 6.*). To produce the confusion matrix, the outputted predictions made by the CNN had to be reshaped into a 1D array, and each pixel value (standing between 0 and 1 from the sigmoïd function) had to be associated with either 0 or 1. One would think that using a threshold of 0.5 would be straightforward (0 if < 0.5 and 1 if > 0.5). However, it appears that the outputted pixel values are not evenly distributed between 0 and 1. Therefore, the threshold was chosen subjectively based on the distribution of the values and the best *Categorical prediction*.

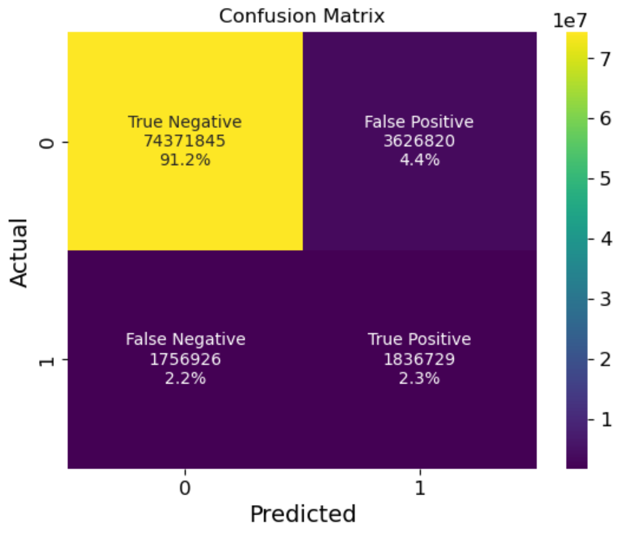


Figure 5. Confusion matrix regarding every pixel prediction on the validation set.

Then, the evaluation metric chosen for our binary classification of pixels in an image is the F1 score (**Equation (2)**). The F1 score combines precision and recall, and is defined as the harmonic mean of the two. In the context of image classification, precision would be the fraction of pixels that were correctly classified, while recall would be the fraction of pixels belonging to a particular class that were correctly identified. The F1 score is a good choice for evaluating binary classification of pixels in an image because it takes into account both the false positives and the false negatives, which are both important considerations in this task.

$$F1_i = \frac{2TP_i}{2TP_i + FP_i + FN_i}, \tag{2}$$

where TP_i is the number of pixels that are correctly predicted as road pixel, FP_i is the number of pixels that are wrongly predicted as road pixel, and FN_i is the number of pixels that are wrongly predicted as non-road pixel for image i. Therefore, for n images, a final score defined as the average of the F1 scores among all images would be:

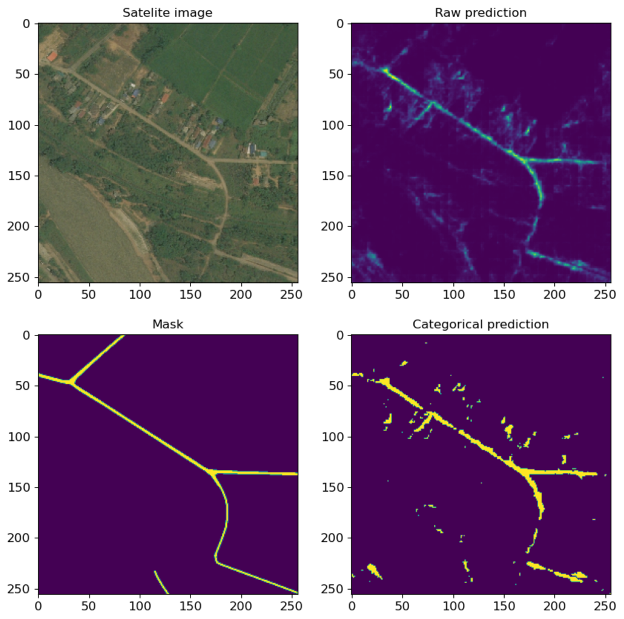


Figure 6. Model *Raw* and *Categorical* predictions over a validation instance compared to the ground truth and its mask.

$$F1 = \frac{1}{n}\sum_{i=1}^{n} F1_i \tag{3}$$

The evaluation metrics for our model on the validation set are as follows:

- Precision = 0.33%
- Recall = 0.51%
- F1 = 0.40%

4.3. Test predictions

We present here some raw predictions made by our CNN on the test set, depicting the great performance of our (complex) model for our (simple) classification problem.

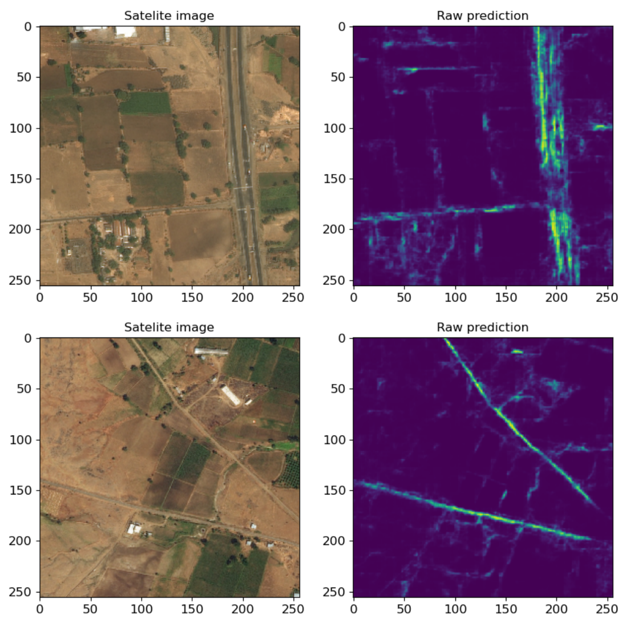


Figure 7. Predictions made on the test set.

5. Discussion

The *raw* predictions made by the CNN are highly accurate. It is able to correctly recognize both main roads and smaller paths. However, this is the main reason why our precision, recall, and F1 score are low. The ”ground truth” depicted by the mask labels only considers well-established roads, so although our model performs the task effectively, it exceeds the capabilities of the labels, which is not ideal. While this high performance on the validation and test sets may not be desirable in some contexts, in the scope of geological hazards it is actually beneficial. This is because, in emergency situations, the ability to accurately identify smaller paths can help stakeholders quickly determine if hard-to-reach areas can be accessed or not.

To further improve our model, it may be helpful to consider using other regularization techniques such as dropout layers or L2 or L1L2 penalties to prevent overfitting. Additionally, using data augmentation to generate more training data could also be helpful. Finally, finding an appropriate threshold for making *categorical* predictions remains a challenge that has not yet been fully addressed in a scientifically rigorous manner. However, implementing this strategy is only to help reduce the complexity of the CNN and improve the model's performance according to our various evaluation metrics.

6. Conclusion

Based on the results of our experiment, it appears that our autoencoder is able to make reliable predictions of road networks from RGB satellite images. The raw predictions made by the autoencoder are highly accurate, and it is able to correctly recognize both main roads and smaller paths. However, this high level of performance may not be desirable in all contexts, as it can lead to low precision, recall, and F1 scores, but are useful in the scope of geological hazards.

7. Code availability

The python code developed and used to produce this project is available at this GitHub repository: https://github.com/Goudals/RoadDeepGlobe_AutoEncoder/tree/main.

Acknowledgements

I would like to thank my peers for their precious feedback that helped me implement, improve and achieve this project. Also, I would also like to thank Tom Beucler and the TAs of the *Machine Learning for Earth and Environmental Sciences SA22* course for their availability and help during the semester.

References

Demir, I. et al. (June 2018). “DeepGlobe 2018: A Challenge to Parse the Earth through Satellite Images”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. ISSN: 2160-7516, pp. 172–17209. DOI: 10.1109/CVPRW.2018.00031.

Ronneberger, O., P. Fischer, and T. Brox (2015). “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by N. Navab et al. Cham: Springer International Publishing, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.

Yang, S. et al. (2022). “Automatically Extracting Surface-Wave Group and Phase Velocity Dispersion Curves from Dispersion Spectrograms Using a Convolutional Neural Network”. In: *Seismological Research Letters* 93.3, pp. 1549–1563. ISSN: 0895-0695. DOI: 10.1785/0220210280.