Project Elective

Map Analytics - Endterm Report

Vineet Priyedarshi

May 2024

Contents

Chapter 1

Classification of Maps between Colourbar and Discrete

1.1 Introduction

In this project, we aim to automate the extraction of relevant choropleth map images from PDF research papers continuing from where we had left in midterm. Choropleth maps are commonly used in research to visualize data related to geographic regions. However, manually extracting these maps from PDF documents can be time-consuming and tedious. Our solution utilizes a combination of Python libraries, image processing techniques, and machine learning models to streamline this process.

1.2 Problem Statement

Given a research paper or folder, the goal is to extract choropleth map images, classify them as discrete or colourbar and annotate the title, map and legend. This can be beneficial in various applications, such as map interpretation, data visualization, and geographic information systems.

1.3 Data and Methodology

1.3.1 Dataset

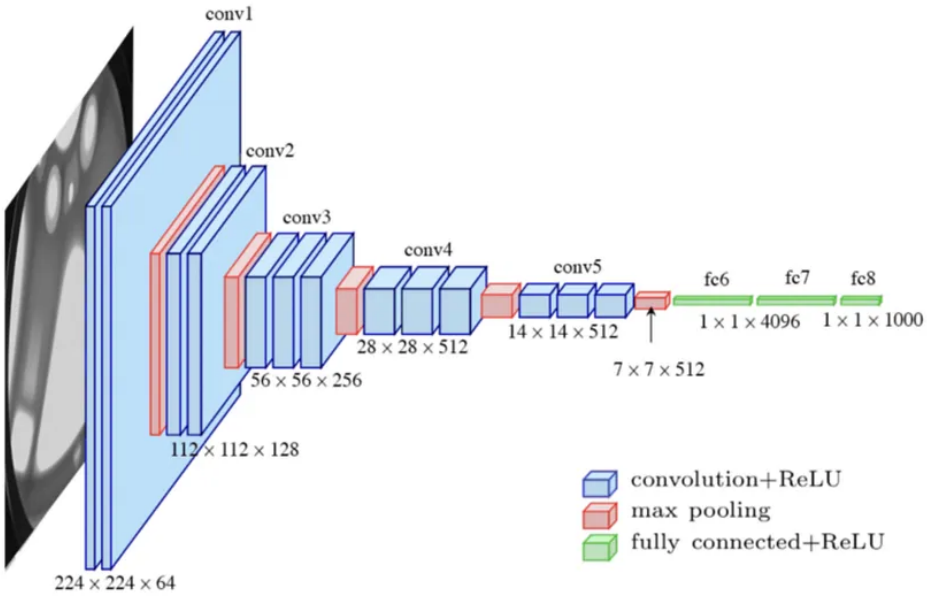
The dataset used in this project consisted of a collection of map images, divided into two classes: discrete legends and continuous (colorbar) legends. The dataset was split into training and validation sets, with the following distribution:

- Training set: 100 images (50 discrete, 50 continuous)
- Validation set: 24 images (12 discrete, 12 continuous)
- Training set: 20 images (10 discrete and 10 continuous)

The images were obtained from various sources and represented a diverse range of map styles and legend designs.

1.3.2 Transfer Learning with VGG16

1.3.3 vgg16 Image similarity



To leverage the power of pre-trained models and transfer learning, the VGG16 architecture was chosen as the base model for this project. VGG16 is a well-known CNN architecture that has been pre-trained on the ImageNet dataset, consisting of millions of images from various categories. By using a pre-trained model, we can take advantage of the rich feature representations learned by the model, which can be fine-tuned and adapted to our specific task of legend classification.

The choice of using the VGG16 model in this map legend classification task is motivated by the effectiveness of transfer learning and the model's proven performance in computer vision tasks. Transfer learning is a powerful technique in deep learning that allows us to leverage the knowledge gained from models pre-trained on large datasets for a different but related task. In this case, the VGG16 model was pre-trained on the ImageNet dataset, which consists of millions of images from various categories, including natural scenes, objects, and textures.

By using a pre-trained model like VGG16, we can take advantage of the rich feature representations learned by the model during its training on the ImageNet dataset. These features, especially the lower-level ones, are often transferable to other computer vision tasks, as they capture general patterns and characteristics of images, such as edges, textures, and shapes.

1.4 Data Preprocessing and Augmentation

Before feeding the images into the model, several preprocessing and augmentation steps were applied to enhance the model's performance and generalization capabilities:

- Rescaling: The pixel values of the images were rescaled to the range [0, 1] by dividing them by 255.
- Data Augmentation: To increase the diversity of the training data and prevent overfitting, data augmentation techniques were applied. These included random width and height shifting, which slightly translated the images horizontally and vertically during training.

1.5 Model Architecture

VGG16 Base Model: The pre-trained VGG16 model was used as the base, with all layers frozen (non-trainable) to preserve the learned feature representations.

Flatten Layer: The output of the VGG16 base model was flattened to a one-dimensional vector. Dense Layer (1000 units): A dense layer with 1000 units and ReLU activation was added. Output Layer (2 units): The final output layer had 2 units with a softmax activation function to produce the class probabilities for discrete and continuous legends.

In our map legend classification project, we are utilizing the VGG16 model as a feature extractor and adapting it for our binary classification task (discrete vs. continuous legends). Here's how

we are using the VGG16 model:

1. Loading Pre-trained Weights: We load the pre-trained weights of the VGG16 model, which were trained on the ImageNet dataset. These weights encode the learned feature representations that can be leveraged for our task.

2. Freezing Base Model Layers: We freeze all the layers of the VGG16 base model, ensuring that the learned weights and feature representations remain fixed during fine-tuning. This step allows us to leverage the powerful features learned by the pre-trained model while focusing on learning task-specific representations in the added layers.

3. Removing Original Fully Connected Layers: We remove the original fully connected layers from the VGG16 model, as these layers were specifically designed for the ImageNet classification task with 1000 classes.

4. Adding Custom Layers: After the convolutional and max-pooling layers of the VGG16 base model, we add the following custom layers:

   (a) Dense Layer (1000 units): A dense layer with 1000 units and ReLU activation is added to learn task-specific representations for our binary classification task.

   (b) Output Layer (2 units): The final output layer has 2 units with a softmax activation function to produce the class probabilities for discrete and continuous legends.

5. Fine-tuning: During the training process, the weights of the VGG16 base model remain frozen, while the weights of the added custom layers are updated and fine-tuned using our map legend dataset. This process allows the model to learn task-specific representations tailored to the legend classification task.

1.6 Training and Optimization

The model was compiled with the following settings:

- Loss function: Categorical cross-entropy
- Optimizer: Adam optimizer with a learning rate of 0.0001
- Evaluation metric: Accuracy

The training process involved feeding the data to the model using data generators, which enabled

efficient loading and preprocessing of the images during training. The model was trained for 120 epochs, with the training and validation sets being passed through the respective data generators at each epoch. During training, the model's performance was monitored on both the training and validation sets, allowing for early stopping or adjustments if necessary.

1.6.1 Training and Validation Performance

The training and validation accuracy and loss at each epoch

The model achieved good performance on both the training and validation sets, with the training accuracy reaching 100% and the validation accuracy reaching 100% after 120 epochs.

1.7 Test Set Performance

After saving the optimal model, we run it on our test data. To see the test dataset's accuracy, we compare how many of the map images were correctly classified. Across 20 test images of high quality, we got an accuracy of 100%.

1.8 Results and Evaluation

So we finally have a reliable classification model with an accuracy of 100%. We then move on to our annotation model for annotating map, title and legend.

Chapter 2

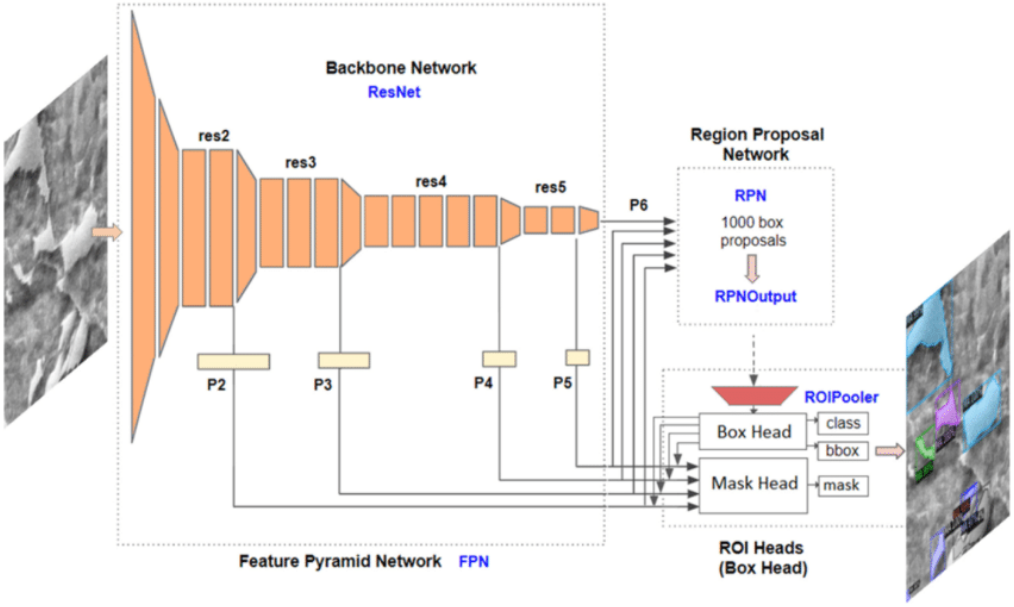
Map Annotation using Object Detection

2.1 Introduction

In addition to classifying map legends as discrete or continuous, another important task in map analysis is the accurate detection and annotation of key components such as the map itself, the title, and the legend. This annotation task can provide valuable information for interpreting and understanding the map's content and context.

The objective of this chapter is to develop a deep learning model capable of detecting and segmenting these three classes (Map, Title, and Legend) in map images. By leveraging the power of object detection and instance segmentation techniques, the model can localize and draw bounding boxes around these objects, enabling downstream applications such as automated map interpretation, data extraction, and information retrieval.

2.2 Model Architecture



For this task, we have used the Detectron2 library, which is a state-of-the-art object detection and segmentation framework developed by Facebook AI Research (FAIR). Detectron2 provides a modular and extensible codebase for building and training object detection and instance segmentation models.

The model architecture employed in this project is based on the Mask R-CNN (Region-based Convolutional Neural Network) model from the Detectron2 library, which is a powerful and widely-used approach for object detection and instance segmentation tasks.

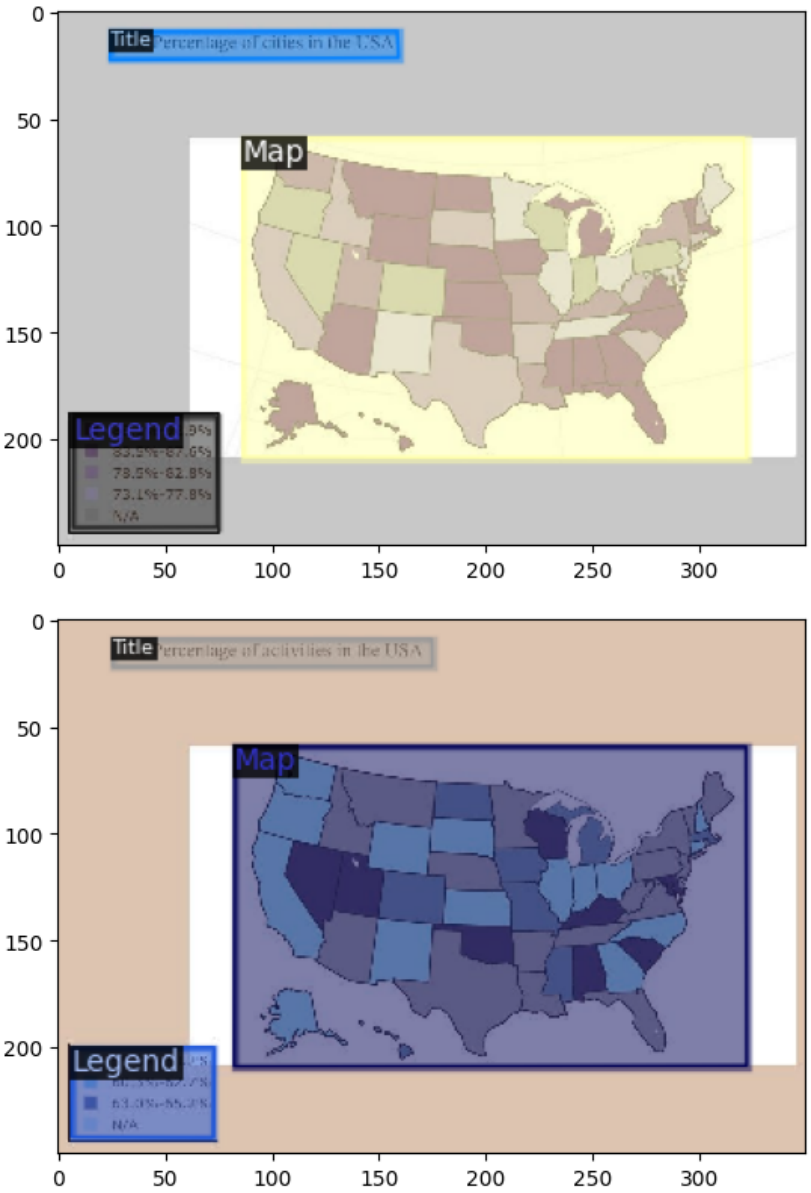
The backbone network used in this architecture is ResNet-50, which is a deep convolutional neural network with 50 layers. The backbone network is responsible for extracting rich feature representations from the input image.

On top of the backbone network, a Feature Pyramid Network (FPN) is employed. The FPN is a top-down architecture that combines low-resolution, semantically strong features with high-resolution, semantically weak features from different layers of the backbone network. This approach allows the model to detect objects at various scales, which is particularly important for map images where the sizes of the Map, Title, and Legend objects can vary significantly

2.3 Training Process

2.3.1 Dataset

For training the model, we have prepared a custom dataset consisting of map images annotated with bounding boxes and segmentation masks for the three classes: Map, Title, and Legend. The dataset is divided into a training set and a validation set. The annotations are provided in the COCO (Common Objects in Context) format, which is a widely used format for object detection and segmentation tasks.



2.3.2 Data Preprocessing

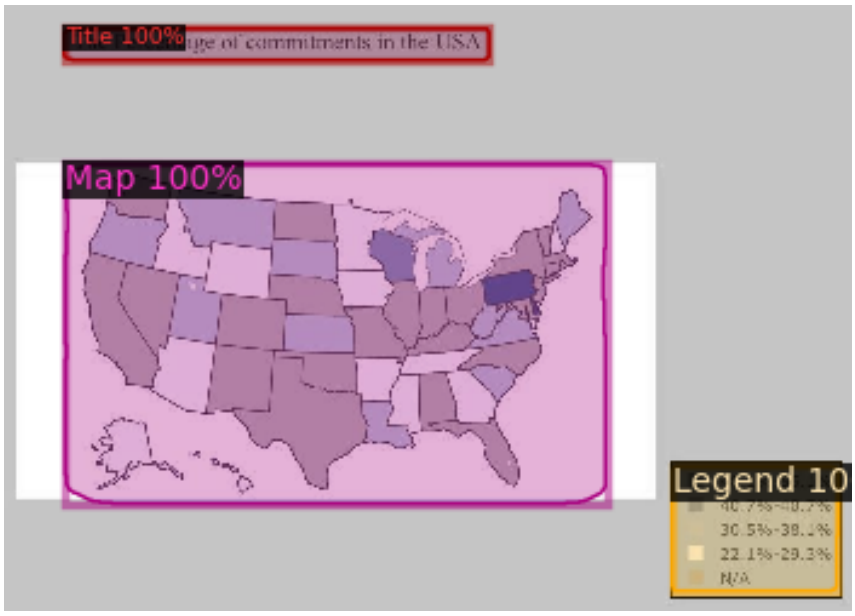
Before training, the dataset is registered with Detectron2 which associates the dataset with the appropriate metadata and annotation files. We manually annotate the legend, title and map.

2.3.3 Model Configuration

The model configuration is set up using the get_cfg function from Detectron2. We have fine-tuned the pre-trained Mask R-CNN R-50 FPN 3x model by loading the pre-trained weights from the COCO dataset. Several configuration parameters are set, such as the number of classes (3), batch size, learning rate, and the maximum number of iterations.

2.3.4 Training and evaluation

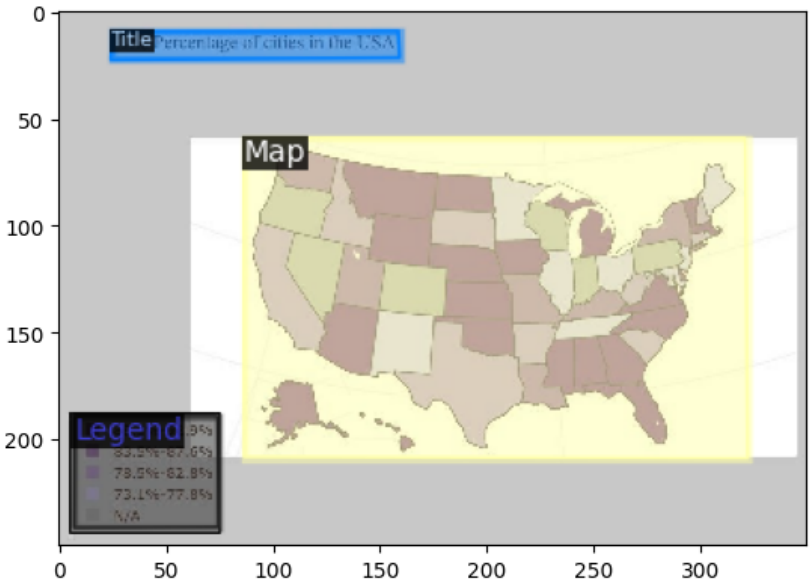
The model is trained for 7,500 iterations, with a batch size of 2 and a learning rate of 0.0001. During training, the model's performance is evaluated on the validation set using the COCO evaluation metrics.



2.4 Results

To evaluate the performance of the trained model, we have used the COCOEvaluator from Detectron2, which computes the standard COCO evaluation metrics, such as Average Precision (AP) for different IoU (Intersection over Union) thresholds.

2.4.1 Quantitative Evaluation



The performance of the trained annotation model was evaluated on the validation set using the standard COCO evaluation metrics. The overall Average Precision (AP) at an Intersection over Union (IoU) threshold of 0.50:0.95 and for all object area ranges was 0.756. This indicates that the model achieved a reasonably good trade-off between precision and recall in detecting and segmenting the Map, Title, and Legend objects.

When analyzing the performance across different object scales, the model exhibited an AP of 0.954 for large objects (area range "large"), demonstrating its effectiveness in detecting and segmenting objects occupying a significant portion of the image.

Examining the class-wise performance, the model achieved an AP of 0.91.80 for the "Map" class, indicating excellent performance in detecting and segmenting the map itself. However, the performance was lower for the "Title" class, with an AP of 0.57.299, and the "Legend" class, with an AP of 0.74.129. These results suggest that the model may benefit from further improvements, particularly in accurately detecting and segmenting titles and legends, which could be more challenging due to their smaller size, varying styles, or potential occlusions.

Overall, the quantitative evaluation results demonstrate the model's competence in annotating map images, with promising performance in detecting and segmenting the Map, Title, and Legend objects. However, opportunities for improvement exist, especially in addressing challenges related to object scale and class-specific performance.

2.5 Conclusion

The quantitative evaluation results, with an overall Average Precision (AP) of 0.756 at an IoU threshold of 0.50:0.95, highlight the model's competence in balancing precision and recall for the annotation task. The model's performance was particularly impressive in detecting and segmenting the map objects, achieving an AP of 0.91.80 for the "Map" class. However, there is room for improvement in accurately annotating titles and legends, as reflected by the lower AP values for those classes.

Despite the model's strength, several challenges and limitations were identified during the development process. These include the impact of object scale on performance, with the model struggling to detect and segment smaller objects accurately. Additionally, the potential for occlusions and overlaps between objects, as well as the inherent complexity and variations in the appearance of titles and legends, present opportunities for further refinement and advancement.

Looking ahead, the integration of this annotation model with the legend classification model developed in Chapter 1 could pave the way for a more comprehensive and robust solution for map analysis. By combining the information about the legend type (discrete or continuous) and the localization of the legend, a deeper understanding of the map's content and context can be achieved, enabling a wide range of applications in cartography, geographic information systems, and data visualization.

Bibliography

[1] Detectron2. https://github.com/facebookresearch/detectron2

[2] Choropleth Maps in Python

https://plotly.com/python/choropleth-maps/

[3] Use Python Geopandas to Make a US Map with Alaska and Hawaii

https://medium.com/@alex_44314/use-python-geopandas-to-make-a-us-map-with-alaska-and-ha

[4] Create Choropleth Map From Table Data

https://www.mathworks.com/help/map/create-choropleth-map-from-table-data.

html

[5] Python Tutorial: How to Create a Choropleth Map Using Region Mapping

https://blog.matthewgove.com/2021/07/23/python-tutorial-how-to-create-a-choropleth-map-

[6] Embed Maps and Visualizations in Python with Plotly HTML

https://www.twilio.com/en-us/blog/embed-maps-visualizations-python-plotly-html

[7] Step by Step: How to Plot a Map Slider to Represent Time Evolution

https://amaral.northwestern.edu/blog/step-step-how-plot-map-slider-represent-time-evolu

[8] Distribution of bird diversity in the US

https://www.researchgate.net/figure/Distribution-of-bird-diversity-in-the-US-Notes-The-

fig3_353731172