Object detection in Wireframes using a pre-trained Faster R-CNN ResNet-50 FPN and conversion to a website using HTML (Wireframe to code)

Jakob Soens - A01760215

Introduction

My goal in this project is to convert a wireframe into an actual working website. I try to accomplish this by using a deep learning convolutional neural network to detect the different elements in the wireframe. These objects will be detected by use of bounding boxes with coordinates of each box (Xmin, Xmax, Ymin, Ymax). Each prediction will also feature the type of each element.

With this prediction I can convert this into a fully working website. To do this I will be using a GRID system implemented in CSS3. I will try to accurately depict each bounding box with a correct position within the grid. Accounting for the size of each element as well. With a preset CSS file I can give each element custom styles and make sure that the elements don't get in the way of each other. I will also test how far I can go with a custom JS file for animations and so on...

The main goal is to get the most accurate depiction of the wireframe. If the wireframe has mistakes it will also copy those mistakes. In the end when I completed this project, I will try to implement learning, so the model is able to fix mistakes made into a wireframe and detect faults. However for now the most accurate model is what I'm going for.

I have already tested out different models, but it seems that a pre-trained Faster R-CNN ResNet-50 FPN from PyTorch's model zoo is he most accurate I could find. I have yet to try every model, but so far it seems like I will stick with this.

I will use this pre-trained model and transfer learn it onto my custom-made dataset. I created the dataset using a pen tablet and illustrator. I made sure to use certain conventions for each element as to not confuse the model with too similar objects.

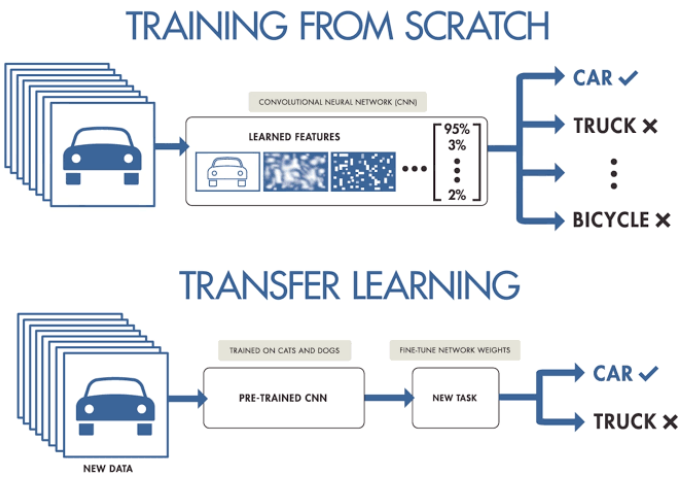
Transfer learning on a pre-trained model

Firstly I will explain in short what transfer learning is. As it is the basis of this project and how I intend to make predictions on my newly made dataset.

Transfer learning is currently the best way to go about creating a model using a custom dataset. Instead of starting from scratch. It builds accurate models in a very timesaving way. The pre-trained model has already learned from a different dataset and knows it's features. In this case our ResNet-50 was also trained for the purpose of object detection. So when we do transfer learning to our dataset it already knows what it's looking for in a way.

We give this model a small amount of data, so it knows what to look for in predictions and with as little as 100 images it's able to find somewhat accurately what we need. This is very useful as if we were to train a model from scratch, we would need to have

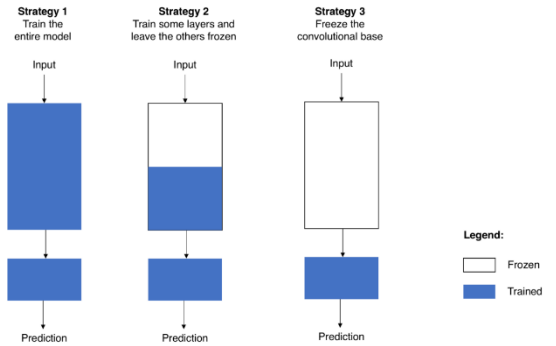
thousands upon thousands of images for a model to be able to learn where our features lie. To even find out if this would bring a better result would takes a lot of time. So, in



this case transfer learning is the best solution. A different pre-trained model can of course bring different results with it.

There are many ways to go about transfer learning the main 3 are:

- Train the entire model
- Train some of the layers and leave some frozen
- Freeze the convolutional base



In this case we will use the first method retraining the entire model to fit our dataset.

During this training on the new dataset it's going to calculate a loss per epoch. So we can see and update our new model accordingly. The loss function that is used is the following.

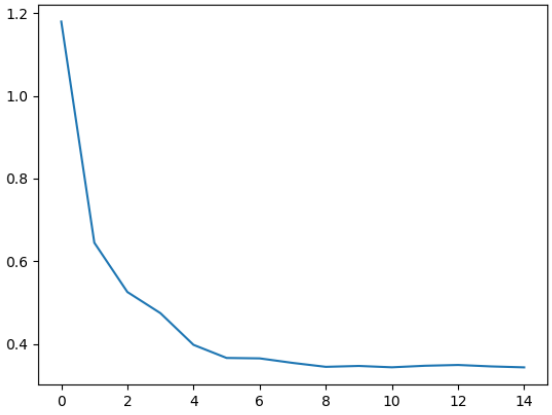
$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*).$$

This contains 2 terms:

1. Classification loss over the classes
2. Regression loss over the predicted bounding boxes.

Of course this is obvious since we are predicting 2 things not only are we trying to predict the class of the found element, but also the bounding box. Below is the loss graph of the training of my model.

We can observe that training this model is very fast and quite efficient. I used 15 epochs and a learning rate of 0.001. This seems more than enough to reach decent results

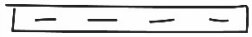
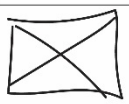


Now I will explain step by step on how I made this project.

Dataset

Firstly, we need to create new data to do transfer learning on a pre-trained model. I created a dataset (200 samples) from scratch. I simply used a pen tablet and illustrator to draw new wireframes. Of course I used a certain convention for each element as to make sure not to make it to difficult for our model.

Conventions

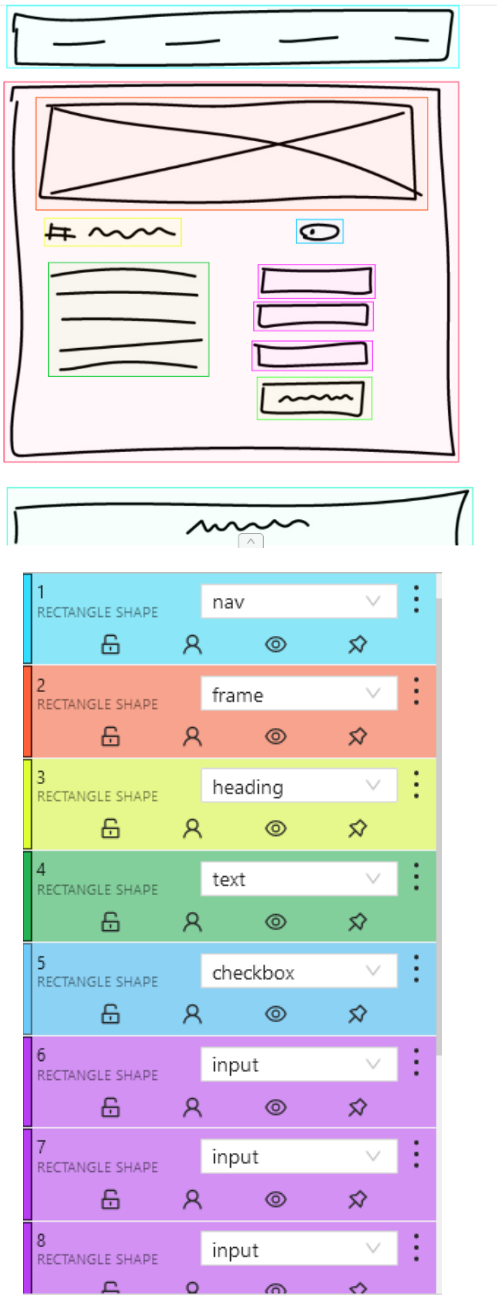
VISUAL ELEMENTS	CORRECT EXAMPLES
NAVIGATION	
CONTAINER	
HEADING	
CHECKBOX	
BUTTON	
FOOTER	
TEXT	
INPUT	
FRAME	

I used these nine elements to start with as to give the model somewhat of a challenge to predict the different classes. Some elements are also made in a way that they can change in size and shape, so the model will have to learn this in training.

The positions of these elements are somewhat similar each time I threw in some random things like footer at the top, nav at the bottom and container within container. Though for the most part every element is the same throughout the training set. Nav at the top, footer and the bottom and the rest of the elements inside a main container.

Bounding boxes

After drawing all the wireframes I must label each element individually. This is called creating bounding boxes. There are many applications you can use to do this I used CVAT (Computer Vision Annotation Tool)



After each element has a label and a bounding box we have to export the dataset in a structure that our model can read and process there are many ways to do this (XML, Text, JSON, …) for each different kind of model. A ResNet-50 model uses a Pascal VOC annotation format. This creates an xml file for each image with the information about the image and the bounding boxes.

```
<?xml version="1.0"?>
- <annotation>
    <folder/>
    <filename>0_jpg.rf.a22d17ea8dc7d10a24212a00a4fa0fac.jpg</f
    <path>0_jpg.rf.a22d17ea8dc7d10a24212a00a4fa0fac.jpg</path
  - <source>
      <database>roboflow.ai</database>
    </source>
  - <size>
      <width>640</width>
      <height>640</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>nav</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
      <occluded>0</occluded>
    - <bndbox>
        <xmin>46</xmin>
        <xmax>582</xmax>
        <ymin>1</ymin>
        <ymax>74</ymax>
      </bndbox>
    </object>
  - <object>
      <name>frame</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
      <occluded>0</occluded>
    - <bndbox>
        <xmin>80</xmin>
        <xmax>544</xmax>
        <ymin>109</ymin>
        <ymax>242</ymax>
      </bndbox>
    </object>
```

Creating new model weights

After the dataset has been made, we can start to create new model weights by transfer learning our new dataset onto the pre-trained model.

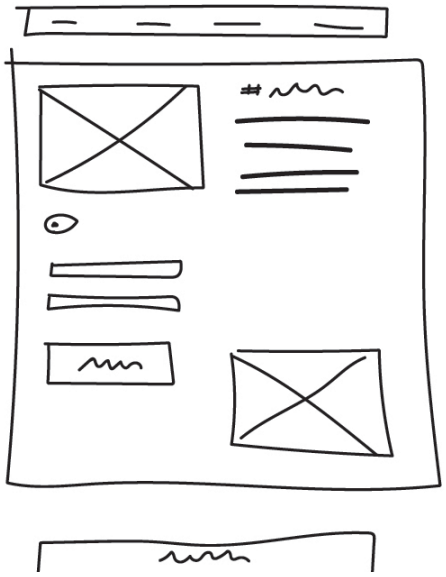
Firstly, I created our training set with some custom transformations this will basically augment the data into new images as to give the model more chances to learn from what limited data we give it.

After this we can retrain the model to fit our custom dataset and save the model for future purpose. As said before we use 15 epochs with a learning rate of 0.001 and step size of 5 to ensure a leisurely training cycle. This process takes about 10-20min depending on your hardware. We can plot the losses using the function explained above. From this we can say that our model will be very good at predicting new wireframe using the same convention.

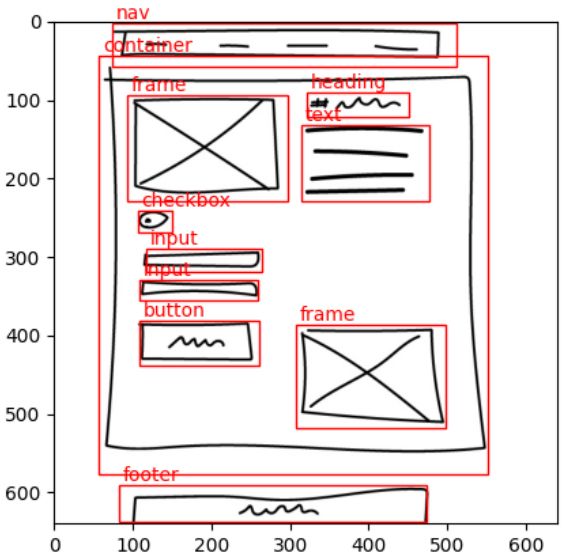
Predicting a wireframe

Now that we have a working model with a decent loss. We can start to test this out.

If we now give the model a test image for instance:



We can see how this performs. This image is very common as most images in the training set are very similar. The way the predictions work is with a score. Each element that has been found is given a score as to how sure the model is about this prediction. If we don't put a threshold on this the model will predict all sorts of wild things. So, we need to give it a number. The score is given from 0-1 so, I picked a threshold of 0.8 as to have some faith in our model.



As you can see this shows a very great result. Every class has not only been predicted right, but also the bounding boxes are almost perfect. From this prediction I can very accurately convert this into a working code example.

The prediction is given as follows:

```
{
  'image': 'data/testset/2.jpg',
  'element': 'button',
  'Xmin': 108.13252258300781,
  'Ymin': 380.9977111816406,
  'Xmax': 261.0608825683594,
  'Ymax': 438.7176208496094
}
```

With this I can create a new element in HTML.

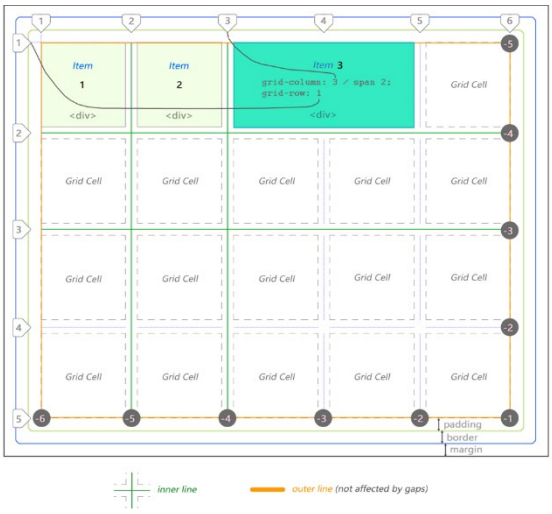
Converting to HTML

There are a lot of different ways to go about this; Even a lot of different frameworks you can use. I decided to just stick to plain old HTML and CSS to convert my website. This should be the simplest and still a very accurate way to do it. I created a placeholder folder in my project that will be copied when a prediction is made and a new index.html will be made. This way a have a template CSS file where I start from and just need to write a index.html where I add all the elements.

Positioning

So, we have some X and Y values for our bounding boxes how to we actually calculate where these elements are within our browser. We could do this statically, but this would create the problem that when you start working on different resolutions this would not work properly anymore.

For this problem I used a new implementation in CSS3 called grids. This splits the browser up in columns and rows this way we can convert our X and Y values into a certain cell within the grid where our element is. For this I used a grid of 20. So, 20 rows and 20 columns in total 400 cells where our elements can be.



Verification

One more problem we have with positioning is the Z-axis. How do we know if an element is within another one? A container/div is mostly used to store different elements but

we the prediction doesn't give any information about this.

The prediction is a JSON that shows all elements in order as they were predicted, but this doesn't mean anything in terms of position what element is first the one with lower X or lower Y? That is why when I get this list, I sort it by lowest Y this way every element is ranked from first to last and we can accurately work our way from top to bottom.

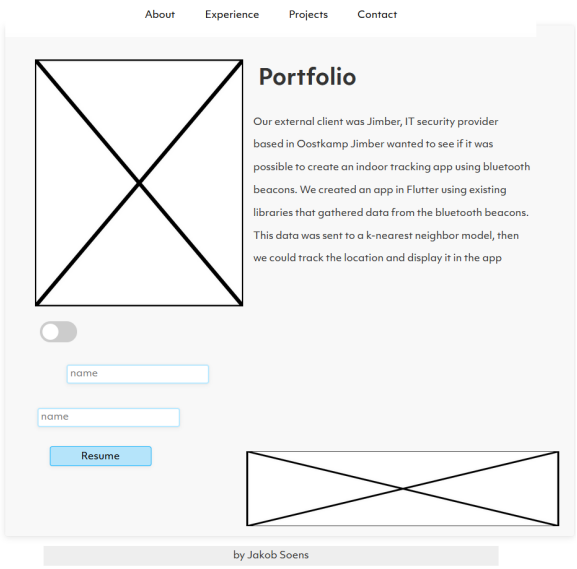
Another problem we have is multiple containers. How to we know what elements are inside what container. So again we must create a check for each element to see where they are in terms of a container element. Firstly, we check for elements labeled containers and loop all elements to check if they are positioned inside it. If the elements are inside, we remove them from the list and check the rest of the list again for elements that are outside of it.

```
15  <body>
16      <div class="o-container">
17          <div class="c-content o-row">
18  >           <div class="c-app__topbar js-na
45              </div>
46              <div class="c-container" style=
47                  <h1 class="c-heading" style
48                  <div class="c-frame"
49                      style="min-height:65.0p
50  >               <div class="c-text" ···
57                  </div>
58                  <div class="c-checkbox" sty
59                          <input type="checkb
60                  <div style="grid-column:4 /
61                          placeholder="name"
62                  <div style="grid-column:3 /
63                          placeholder="name"
64                  <a href="#" class="c-nav__l
65                      style="max-width:164px;
66                  <div class="c-frame"
67                      style="min-height:65.0p
68              </div>
69
70              <div class="c-footer" style="ma
71                  <p>by Jakob Soens</p>
72              </div>
73          </div>
```

By doing this we make sure that each element is not only positioned correctly on X and Y, but also Z-axis. As we can see from the code example every element is nicely placed inside the container while the footer and navigation are still outside it.

Result

Below you can see a result of our previous prediction.



As we can see this is an accurate depiction of our wireframe, but as I said before the model will only predict where the elements are. It will not change it to be better. Therefore we can still see some mistakes. Such as the navigation and container overlapping and the footer being too short or the inputs not being in the same column.

Conclusion

This is of course only one way to solve a problem like this. I already know better ways to solve this problem.

1. A more accurate model.

Of course this is easier said than done. I've been looking for some time now to make my model better. I looked for different pre-trained models, writing the code myself as well (sadly to no avail) or even creating a bigger dataset.
It seems that as far as this attempt goes this is the best I can do.

2. Creating a better grid system

   This is a hard one but still doable. It would be possible to upgrade the current grid system to look at different elements, so they don't overlap or place different elements according to nearby elements. This way we don't have situation like the inputs being in different columns.

These are only some of the changes I have in mind. Of course as my main conclusion this would only be a good starting point for a front-end developer. There has to be someone to put what is created together by cleaning up the code and adding more CSS and JS. But so far this is the result.

[1] „Pix2Code Github Repository," Tony Beltramelli, 03 10 2017. [Online]. Available: https://github.com/tonybeltramelli/pix2code .

[2] „Pix2Code Research paper," Tony Beltramelli, 19 09 2017. [Online]. https://arxiv.org/abs/1705.07962 .

[3] „Convert your design mockup to html code," Emil Wallner, 05 02 2018. [Online]. Available: https://www.freecodecamp.org/news/how-you-can-train-an-ai-to-convert-your-design-mockups-into-html-and-css-cc7afd82fed4/ .

[4] „Realtime Code Generation using Machine Learning (video)," TeleportHQ, 29 09 2018. [Online]. Available: https://www.youtube.com/watch?v=ZD8Cnzc6KlY .

[5] „Generating low-fidelity designs by talking to your computer," TeleportHQ (Dimitri Fichou), 21 10 2020. [Online]. Available: https://teleporthq.io/blog/generating-low-fidelity-designs-by-talking-to-your-computer

[6] „How deep learning is transforming design: NLP and CV applications (blog)," Uizard (Javier Fuentes), 30 09 2020. [Online]. Available: https://towardsdatascience.com/how-deep-learning-is-transforming-design-cv-and-nlp-applications-4518c50690e6

[7] „Generating Design Systems using Deep Learning (blog)," Uizard (Tony Beltramelli), 30 09 2020. [Online]. Available: https://towardsdatascience.com/improving-ui-layout-understanding-with-hierarchical-positional-encodings-b19e1e9235e

[8] „R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms (blog)," Uizard (Rohith Gandhi), 30 09 2020. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[9] „ Review-faster-r-cnn-object-detection — Review faster R-CNN Object detection (article)," Sik-Ho Tsang, 30 09 2020. [Online]. Available: https://towardsdatascience.com/review-faster-r-cnn-object-detection-f5685cb30202

[10] „ Transfer learning from pre-trained models — Transfer learning from pre-trained models (article)," Pedro Marcelino, 23 10 2018 [Online]. Available: https://towardsdatascience.com/review-faster-r-cnn-object-detection-f5685cb30202

[11] „Detecto Framework — Detecto Object Detection ," Multiple users, 12 12 2019. [Online]. Available: https://detecto.readthedocs.io/en/latest/index.html