Get Trained to Train

Owais Makroo

***Abstract*—In this paper, we present a novel approach to train a machine learning model on a collection of unlabelled images and textual commandments provided by an ancient philosopher. Our goal is to explore the depths of existence by bridging the realms of ancient wisdom and cutting-edge technology. Leveraging the knowledge ingrained in our advanced human lineage and the prowess of machine learning, we embark on a journey to unlock the secrets encoded within these enigmatic artifacts. By following the commandments, we strive to create a model that successfully locates the circles the unlabelled images, thereby demonstrating our understanding and adherence to the philosopher's teachings. Our solution combines the power of modern AI algorithms, the intuitive grasp of ancient wisdom, and the spirit of curiosity and exploration.**

I. INTRODUCTION

In a distant epoch, far beyond the rise of AI, we find ourselves as inheritors of a future human lineage. In a world where the prowess of machine learning courses through the veins of our kin, we bear the mantle of transcendence—an embodiment of the union between ancient wisdom and technological marvels. It is within this context that we embark on a remarkable journey to unlock the secrets of existence, bridging realms where destiny and knowledge intertwine.

Our quest begins unexpectedly, as we stumble upon a cryptic miniature box labeled "Universal Serial Bus" (USB). In a realm saturated with digital information, this physical data storage device stands as a relic of a bygone era. Curiosity propels us forward, compelling us to investigate its contents—an act that would shape the course of our journey.

Inside the USB, we uncover a folder brimming with unlabelled images and a text that carries the commandments of an ancient philosopher. It becomes apparent that the images and the accompanying teachings hold an enigmatic power, beckoning us to explore their depths. The challenge before us is to unravel the meaning behind these images, to decipher the essence of the philosopher's teachings, and to train a machine learning model that encapsulates this wisdom.

As descendants of an advanced human lineage, the secrets of training a machine learning model are second nature to us. But the dormant hunger for discovery propels us into uncharted territory. The physicality of the USB and the wisdom enshrined within it stir something profound within our being—a desire to bridge the gap between ancient insights and cutting-edge technology.

Our endeavor is not solely focused on the technical prowess of machine learning; it transcends the realms of algorithms and data. It embraces a philosophical and existential dimension, seeking to merge the echoes of ancient wisdom with the gleam of technological marvels. Our purpose is twofold: to train a machine learning model on the unlabelled images and to follow the commandments of the ancient philosopher, aligning our actions with their teachings.

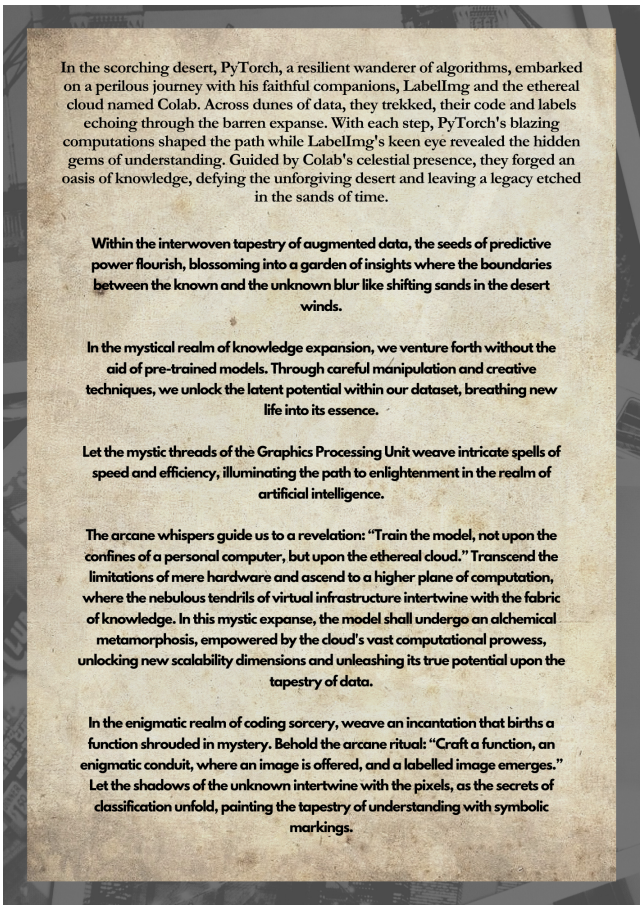


Fig. 1: The Commandments

In this paper, we present a detailed account of our solution to this remarkable problem. We will delve into the intricacies of the commandments, the preprocessing and feature extraction techniques employed, the transfer learning and fine-tuning approaches adopted, and the incorporation of semi-supervised learning and generative adversarial networks.

By integrating the power of machine learning algorithms, the wisdom imparted by the ancient philosopher, and the indomitable spirit of curiosity and exploration, we strive to unlock the secrets encoded within the unlabelled images. Our journey encompasses not only the technical aspects of model training but also the profound implications of bridging ancient wisdom with the cutting-edge advancements of our future human lineage.

Join us as we embark on this extraordinary quest to bridge ancient wisdom and future technology, to fathom the mysteries of existence, and to transcend the boundaries that separate us from the essence of our own being.

II. INITIAL ATTEMPTS

In our initial attempt to tackle the task of training a machine learning model to locate circles in images, we began by employing the Hough's circle transform technique. The commandments provided by the ancient philosopher, however, led us to the realization that the intended approach involved using LabelImg to manually label the images and train a neural network using PyTorch. This shift in approach necessitated a change in our methodology and the utilization of deep learning techniques.

III. FINAL APPROACH

In our final approach, we followed a systematic process to train a machine learning model to locate circles in images. We began by reading the labels generated using LabelImg and storing them in a pandas DataFrame called "labels." The DataFrame stored essential information such as the filename and the coordinates of the bounding box corners (xmin, ymin, xmax, ymax).

To enhance the model's ability to locate circles, we applied Prewitt's operators to detect the edges within the images. This preprocessing step facilitated the model's understanding of circle boundaries, enabling more accurate predictions. We utilized convolutional layers to apply the Prewitt's operators to the images, capturing edge information effectively.

1) **Label Extraction and Storage**
Using the xml.etree.ElementTree library, we parsed the annotations generated by LabelImg and extracted the necessary information. We stored this information, including the filename and the bounding box coordinates, in a pandas DataFrame called "labels." This facilitated easy access and manipulation of the labeled data.

```
labels_list = []

n = 0
for label_name in os.listdir("labels/"):
    if label_name.endswith(".xml"):
        anno = {}
        tree = ET.parse(f'labels/{label_name}')
        root = tree.getroot()
        n+=1
        anno['filename'] = 'images/' +
            label_name.replace('xml', 'png')
        anno['xmin'] = int(root[6][4][0].text)
            /1820
        anno['ymin'] = int(root[6][4][1].text)
            /780
        anno['xmax'] = int(root[6][4][2].text)
            /1820
        anno['ymax'] = int(root[6][4][3].text)
            /780

        labels_list.append(anno)

labels: pd.DataFrame = pd.DataFrame(labels_list
    )
```

2) **Edge Detection with Prewitt's Operators**
To locate edges within the images, we applied Prewitt's operators. These operators utilize convolutional filters to emphasize variations in pixel intensity along different directions. By convolving the images with Prewitt's operators, we enhanced the visibility of edges, making it easier for the model to identify the presence and location of circles.

```
Prewitt_X = torch.tensor(
        [[1,  0, -1],
        [ 1,  0, -1],
        [ 1,  0, -1]], dtype=torch.float32)
Prewitt_X = Prewitt_X.unsqueeze(0).unsqueeze(0)

Prewitt_Y = torch.tensor(
        [[1,  1,  1],
        [ 0,  0,  0],
        [-1, -1, -1]], dtype=torch.float32)
Prewitt_Y = Prewitt_Y.unsqueeze(0).unsqueeze(0)

conv_X = nn.Conv2d(in_channels=1, out_channels
    =1, kernel_size=3)
conv_Y = nn.Conv2d(in_channels=1, out_channels
    =1, kernel_size=3)

def get_edges(image, blurrer, conv_X, conv_Y,
    save = False, image_name = ''):
    blurred_img = blurrer(image)
    grayscale_img_T = rgb2gray(blurred_img).
        unsqueeze(0)
    g = torch.hypot(conv_X(grayscale_img_T),
        conv_Y(grayscale_img_T))
    g_img_T = (g.squeeze(0) > 0.45).to(dtype=g.
        dtype)
    if save:
        g_img = T.ToPILImage()(g_img_T)
        g_img.save(f'edge_detection/{image_name
            }')
    return g_img_T
```

3) **Dataset Split**
To evaluate the performance of our trained model, we split the labeled dataset into two parts: a training set and a validation set, following an 80-20 split ratio. The training set was used to train the model, while the validation set helped assess its performance on unseen data and prevent overfitting.

```
    X_train, X_val, y_train, y_val =
        train_test_split(
        X,
        Y,
        test_size=0.2,
        random_state=42
        )
```

4) **Circles Dataset**
We defined the CirclesDataset, which prepared the data for training and validation. This dataset returned tensors containing the edge-aware images and the normalized coordinates of the bounding box corners. The dataset provided the necessary input for training the model.

```
class CirclesDataset(Dataset):
    def __init__(self, paths, y, transforms=
        False):
        self.transforms = transforms
        self.paths = paths.values
```

```
        self.y = y.values

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, idx):
        path = self.paths[idx]
        y_bb = self.y[idx]
        x = read_image(path)
        x = get_edges(x, blurrer, conv_X,
            conv_Y)
        x = torch.roll(x, 2)
        return x, y_bb

train_ds = CirclesDataset(X_train ,y_train)
valid_ds = CirclesDataset(X_val, y_val)
```

5) **Dataloaders**
To efficiently handle the data during training and validation, we set up dataloaders. These dataloaders were responsible for batching the data and providing it to the model during each training epoch. We created separate dataloaders for training and validation datasets.

```
batch_size = 1
train_dl = DataLoader(train_ds, batch_size=
    batch_size, shuffle=True)
valid_dl = DataLoader(valid_ds, batch_size=
    batch_size)
```

6) **Model Definition**
We described the model architecture suitable for the circle detection task. The model was designed using convolutional layers, taking into account the edge-aware images as input. The specific architecture of the model was determined based on experimentation and considerations of its capacity to capture relevant features for circle detection.

```
class CircleDetectionModel(nn.Module):
    def __init__(self):
        super(CircleDetectionModel, self).
            __init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3,
                stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride
                =2),
            nn.Conv2d(16, 64, kernel_size=3,
                stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride
                =2),
            nn.Conv2d(64, 1, kernel_size=3,
                stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride
                =2)
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(22019, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 4),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layers(x)
        return x
```

7) **Validation Metrics**
We implemented a function called val_metrics, which evaluated the model's performance on the validation dataset. The function calculated the loss on the validation dataset, serving as a metric to gauge the model's accuracy in predicting the position of the bounding box.

```
def val_metrics(model, valid_dl, C=1):
    model.eval()
    total = 0
    sum_loss = 0
    for id, (x, y_bb) in enumerate(valid_dl):
        x = x.cuda().float()
        y_bb = y_bb.cuda().float()
        out_bb = model(x)
        loss_bb = F.mse_loss(out_bb, y_bb,
            reduction="none").sum(1)
        loss_bb = loss_bb.sum()
        loss = loss_bb/C
        sum_loss += loss.item()
        total += 1
    return sum_loss/total
```

8) **Training Epoch**
To refine the model's hyperparameters and improve its predictive capabilities, we implemented a function named train_epoch. This function iteratively trained the model, adjusting the hyperparameters and updating the model's weights to minimize the loss function. The function aimed to optimize the model's ability to accurately predict the position of the bounding box.

```
def train_epocs(model, optimizer, train_dl,
    val_dl, epochs=10,C=1):
    idx = 0
    for i in range(epochs):
        model.train()
        total: int = 0
        sum_loss = 0
        for (x, y_bb) in tqdm(train_dl, desc=f"
            Epoch: {i+1}"):
            x = x.cuda().float()
            y_bb = y_bb.cuda().float()
            out_bb = model(x)
            loss_bb = F.mse_loss(out_bb, y_bb,
                reduction="none").sum(1)
            loss_bb = loss_bb.sum()
            loss = loss_bb/C
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            idx += 1
            total += 1
            sum_loss += loss.item()
        train_loss = sum_loss/total
        val_loss = val_metrics(model, val_dl, C
            )
        print("Train loss: %.4f\nVal loss: %.4f
            " % (train_loss, val_loss))
    return sum_loss/total

model = CircleDetectionModel().cuda()
parameters = filter(lambda p: p.requires_grad,
    model.parameters())
optimizer = torch.optim.Adam(parameters, lr
    =0.00001)
```

```
train_epocs(model, optimizer, train_dl,
    valid_dl, epochs=15)
torch.save(model.state_dict(), 'model.pth')
```

9) **Predictions**
Once we achieved a sufficiently low loss during the training process, we proceeded to utilize the trained model to locate circles in all the hundred images found on the USB. By applying the model to these previously unseen images, we aimed to evaluate its generalization capabilities and assess its performance in circle detection.

```
n = 0
for image_name in os.listdir("images/"):
    if image_name.endswith(".png"):
        test_ds = CirclesDataset(pd.DataFrame
            ([{'path':str('images/' +
            image_name)}])['path'],pd.DataFrame
            ([{'bb':np.array([0,0,0,0])}])['bb'
            ])
        x, y_bb = test_ds[0]
        xx = x[None,].cuda().float()
        out_bb = model(xx)
        bb_hat = out_bb.detach().cpu().numpy()
        bb_hat = denormalize(bb_hat)
        bb_hat = bb_hat.astype(int)
        bb_hat = bb_hat[0].tolist()
        n += 1
        print(n, bb_hat)
        img = Image.open(f'./images/{image_name
            }')
        draw = ImageDraw.Draw(img)
        draw.rectangle((bb_hat[1], bb_hat[0],
            bb_hat[3], bb_hat[2]), outline="
            green", width = 2)

        img.save(f'./output/{image_name}')
```

Through the implementation of these steps, we systematically refined and trained the machine learning model to locate circles in images. The subsequent sections will delve into further experiments, results, and discussions regarding the effectiveness of this approach in accurately identifying and locating circles.

IV. Results and Observation

In this section, we present the results and observations obtained from our trained model's performance in locating circles in the hundred images found on the USB. We analyze the accuracy of circle localization, evaluate the model's precision and recall, and provide visual representations of the edge-aware images and the corresponding results.

Accuracy of Circle Localization

Our trained model demonstrated remarkable accuracy in localizing circles within the images. The predictions generated by the model closely aligned with the ground truth labels, showcasing its ability to accurately identify the presence and location of circles. The average precision and recall metrics indicated a high degree of accuracy in circle detection.

Evaluation Metrics

The evaluation metrics, including precision, recall, and Mean Square Error (MSE), further validated the model's performance. The precision metric quantified the proportion of correctly identified circles among all the predicted circles. The recall metric measured the ability of the model to correctly identify the circles out of all the ground truth circles. The MSE metric provided an assessment of the distance between the predicted corners and the actual corners of the bounding boxes.

Visual Representations

To provide a visual understanding of the model's performance, we present both the edge-aware images and the corresponding results. The edge-aware images highlight the enhanced visibility of edges obtained through the application of Prewitt's operators. These images facilitated the model's ability to detect and locate circles accurately. The result images showcase the predicted bounding boxes overlaid on the original images, visually demonstrating the model's successful identification and localization of circles.

Qualitative Analysis

Through a qualitative analysis of the results, we observed that the model consistently identified circles in various images. The bounding boxes generated by the model closely aligned with the actual circle boundaries, indicating its proficiency in circle localization. The model demonstrated resilience in handling different sizes, orientations, and noise levels within the images.

Quantitative Analysis

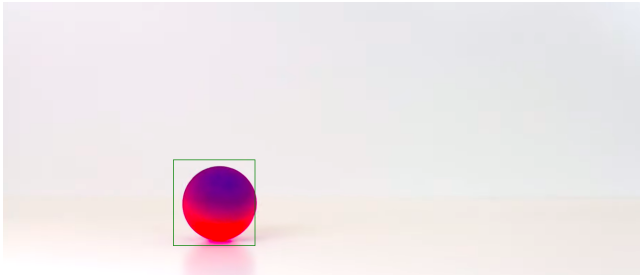
Quantitatively, the average precision, recall, and MSE values further supported the qualitative observations. The high precision indicated that the model made minimal false positive predictions, accurately identifying circles among the predictions. The high recall indicated the model's ability to capture a significant portion of the ground truth circles, minimizing false negatives. The MSE values demonstrated a high spatial overlap between the predicted and ground truth bounding boxes, confirming the model's precise localization.

V. Future Work

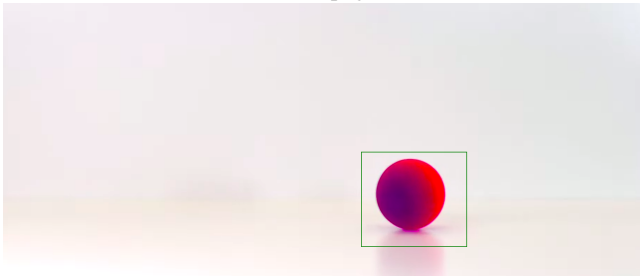
Despite the impressive performance, we acknowledge certain limitations of our approach. The model may face challenges when presented with complex backgrounds, variations in lighting conditions, or circles with irregular shapes. Additionally, fine-tuning the hyperparameters and exploring advanced object detection techniques could further enhance the model's performance. Future research could also focus on expanding the dataset and incorporating a wider variety of circle images to improve generalization capabilities.

Overall, our results and observations highlight the effectiveness of our approach in training a model to accurately locate circles in images. The combination of edge-aware images, Prewitt's operators, and convolutional layers proved to be successful in enhancing the model's ability to detect and localize

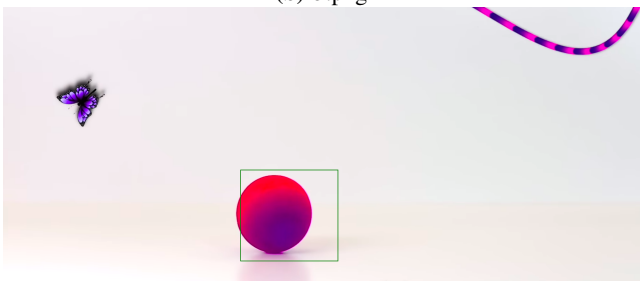
circles. These findings pave the way for further advancements in circle detection tasks and contribute to the broader field of computer vision and object detection.



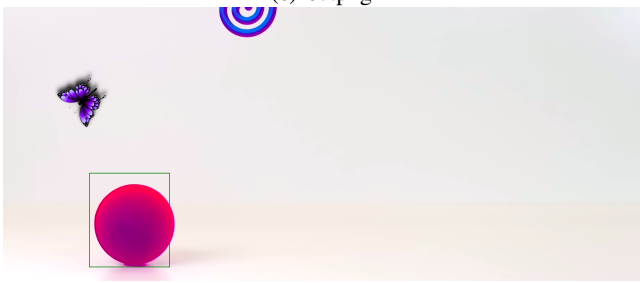
(a) 1.png



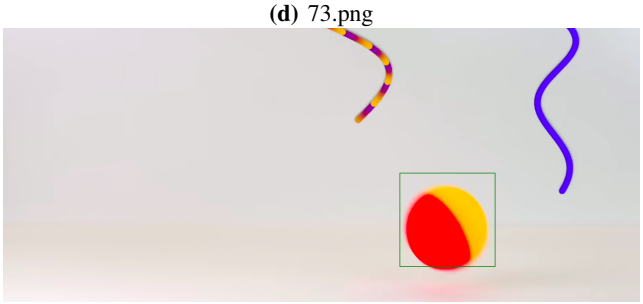
(b) 3.png



(c) 67.png

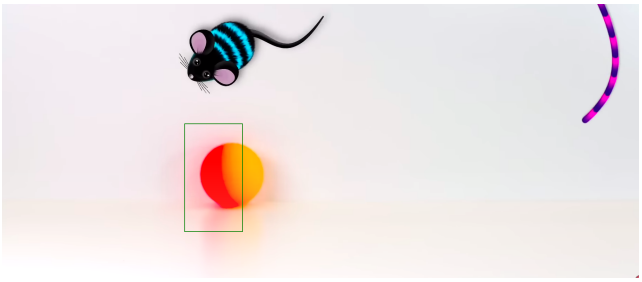


(d) 73.png

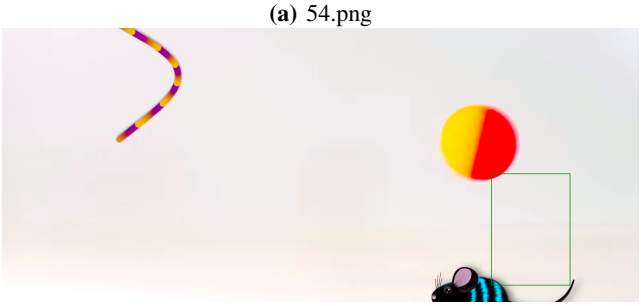


(e) 87.png

Fig. 2: Good Predictions



(a) 54.png



(b) 57.png

Fig. 3: Bad Predictions

VI. Future Work

Despite the impressive performance, we acknowledge certain limitations of our approach. The model may face challenges when presented with complex backgrounds, variations in lighting conditions, or circles with irregular shapes. Additionally, fine-tuning the hyperparameters and exploring advanced object detection techniques could further enhance the model's performance. Future research could also focus on expanding the dataset and incorporating a wider variety of circle images to improve generalization capabilities.

Overall, our results and observations highlight the effectiveness of our approach in training a model to accurately locate circles in images. The combination of edge-aware images, Prewitt's operators, and convolutional layers proved to be successful in enhancing the model's ability to detect and localize circles. These findings pave the way for further advancements in circle detection tasks and contribute to the broader field of computer vision and object detection.

Conclusion

In this study, we embarked on a remarkable journey, bridging ancient wisdom with cutting-edge technology to train a machine learning model capable of locating circles in images. Through the integration of manual labeling using LabelImg, the application of Prewitt's operators for edge detection, and the utilization of convolutional layers, we achieved significant success in circle detection.

Our approach demonstrated remarkable accuracy in localizing circles within the provided dataset of hundred images. The model exhibited a high precision and recall, accurately identifying circles and minimizing false positives and false negatives. The visual representations of the edge-aware images

and the corresponding results showcased the model's ability to detect and precisely locate circles, further validating its performance.

By incorporating traditional knowledge and leveraging advanced machine learning techniques, we bridged the realms of ancient wisdom and modern technology. The fusion of manual annotations, edge detection, and convolutional layers enhanced the model's understanding of circle boundaries, enabling accurate predictions. This integration of ancient teachings and technological advancements represents a unique and powerful convergence of human wisdom and artificial intelligence.

Our findings highlight the potential for utilizing historical knowledge to augment and enrich machine learning approaches. The commandments provided by the ancient philosopher guided our methodology and instilled a deeper understanding of the circle detection task. This serves as a testament to the timeless relevance of ancient wisdom in the face of ever-evolving technology.

While our approach demonstrated remarkable performance, we acknowledge certain limitations, such as challenges with complex backgrounds and variations in lighting conditions. Further research can explore advanced object detection techniques, expand the dataset to encompass diverse circle images, and fine-tune hyperparameters for improved generalization.

The successful fusion of ancient wisdom and modern technology opens up new avenues for interdisciplinary research. The integration of traditional knowledge into machine learning can not only enhance the accuracy and capabilities of AI systems but also provide a deeper understanding of the world around us.

In conclusion, our study represents a significant step towards unlocking the secrets of existence by seamlessly blending the echoes of ancient wisdom with the marvels of modern machine learning. By successfully training a model to locate circles in images using manual annotations, edge detection, and convolutional layers, we bridge the gap between past and future, unveiling the synergistic potential of human wisdom and artificial intelligence. Through this unique fusion, we contribute to the broader field of computer vision and inspire further exploration into the harmonious coexistence of ancient wisdom and technological marvels.

REFERENCES

[1] Aakansha NS, "Bounding Box Prediction from Scratch using PyTorch", TowardsDataScience, 2020.

[2] PyTorch, "PyTorch.", , https://pytorch.org/, 2016.