Ex4 Deep Learning in Computer Vision

הערה:

את הפרויקט המלא כולל הדאטה שנוצר והמודל המאומן אפשר למצוא בקישור הזה:

https://drive.google.com/file/d/1tSvpdJqBGF0Hm5o6bfaZJ05rsPkB8qQf/view?usp=sharing , בנוסף אם רוצים להריץ את plot_results חייב את המודל המאומן או שאפשר לערוך את הקוד ולהציג בלי החלק הזה אבל עדיין צריך את תיקיית output_images.

הצגת הבעיה ופתרון אפשרי:

בפרויקט זה מימשתי מערכת שמטרת להפיק Hand Pose Estimation למספר אובייקטים תלת מימדיים בUnity.

הבעיה המרכזית שאני חוויתי בHand Pose Estimation היא שהייצוג של האובייקט שצריך לתפוס אינו ברור מאליו, מאחר שאובייקט תלת מימדי מתואר במנועי משחק באמצעות Mesh. אותו הMesh ניתן בעצם באמצעות מספר רב של נקודות. ולכל אובייקט יש מספר שונה של נקודות ולכן לתת את ה Mesh לרשת DNN רגילה זה לא ייתכן ביגלל שגודל הinput משתנה.

כמובן שבגישה אחרת אפשר לבצע preprocessing לדאטה ואז לתת את זה לרשת DNN רגילה (זה אני עושה בפרויקט אחר).

אבל, אפשר לקחת תמונה של האובייקט ולתת אותה לרשת CNN שתדע להוציא features טובים של האובייקט לפיהם הרשת תדע לתת outputs לפי אלגוריתמי אופטימיזציה שלמדנו בקורס, או באמצעות אלגוריתמי חיפוס מקומי או אלגוריתמים אבולוציונים או אלגוריתמי למידה מונחת חיזוקים.

הבעייה בשיטה הזאת היא שלא תמיד אפשר לייצג את האובייקט באמצעות תמונה דו מימדית.

ולכן הגעתי למסקנה שאפשר לקחת שני תמונות משתי זוויות שונות של צילום של האובייקט. שייתנו גם הערכה לציר הנוסף שחסר בתמונה דו מימדית פשוטה.

בחרתי בשיטה של חיפוש אבולוציוני וציפיתי לקבל שלאורך הדורות הצאצאים ילמדו לקרב את האצבעות של היד אל האובייקטים הנתונים.

פונקציית ההערכה שהשתמשתי בה לקוחה ישירות מתוך unity כאשר בהינתן רשימת outputs של המודל עבור רשימה של זוגות של תמונות, unity משנה את הסיבובי מפרקי האצבעות בunity, ובנוסף טוען את האובייקט שהoutput הזה מתאים לו, ואז מבצע מדידת מרחק בין כל אחת מהאצבעות לבין האובייקט.

אלגוריתם האימון מנסה לבצע מינימיזציה עבור התוחלת של המרחקים בין האצבעות לאובייקטים.

פירוט המערכת (סביבת האימון, מידול הפתרון, מודל הרשת, דברים שהייתי חייב לעשות כדי שזה יפעל):

מודל הרשת בנוי כך:

```python
class HandBlock(nn.Module):

    def __init__(self, in_dim, out_dim, kernel_size= 5, stride=2) -> None:
        super().__init__()
        self.in_dim = in_dim
        self.out_dim = out_dim
        self.kernel_size = kernel_size
        self.stride = stride
        self.conv = nn.Conv2d(self.in_dim, self.out_dim,
kernel_size=(self.kernel_size,self.kernel_size))
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=(kernel_size, kernel_size),
stride=stride)

    def forward(self, x):
        y = self.conv(x)
        z = self.relu(y)
        norm = nn.LayerNorm(z.shape[1:])
        d = norm(z)
        return self.pool(d)

class HandBrain(nn.Module):
    def __init__(self):
        super().__init__()

        self.cnn = nn.Sequential(
            # input (8, 447, 1002)
            HandBlock(8,16),
            # -> (16, 223, 501)
            HandBlock(16,32),
            # -> (32, 111, 250)
            HandBlock(32,16),
            # -> (16, 55, 125)
            HandBlock(16,8)
            # -> (8, 27, 62)
        )


    def forward(self, x):
        # Forward pass through the CNN
        features = self.cnn(x)

        # print(features.size())
        # Flatten the feature maps for input to the linear layer
        flattened_features = torch.flatten(features, start_dim=1)
        # -> (13x13216 neurons)

        # Define a linear layer for classification
        linear = nn.Sequential(
            nn.Linear(flattened_features.size(1), 54),
            nn.Tanh()
            )
        return (linear(flattened_features) + 1) / 2
```

הסבר פייטורץ בקוד:

המודל מקבל שני תמונות בפורמט RGBA כלומר 4 channels לתמונה. סך הכל אחרי שירשור שתי התמונות יחד קיבלנו 8 channels. גודל כל תמונה הוא 447 1002 X.

המודל מבצע 4 בלוקים של קונבולוציה כאשר כל בלוק מורכב מ קונבולוציה ואז RELU ואז LAYER NORMALIZATION ואז MAXPOOL.

בחירת ה Layer Norm באה כדי למנוע Saturation של tanh שהשתמשתי בו בסוף. מאחר שהoutputs עשויים להיות גדולים מאוד או שליליים מאוד אז tanh יתן תוצאה 0 או 1 (Saturation) ולכן זה לא אידיאלי בישבילי כאשר אני רוצה מנחים יותר מגוונים בין 0 ל 1.

הבלוקים מתבצעים בשירשור אחד אחרי השני, כאשר גודל הקרנל הקונבולוציה הוא 5 כמות הסטרייד זה 2 .

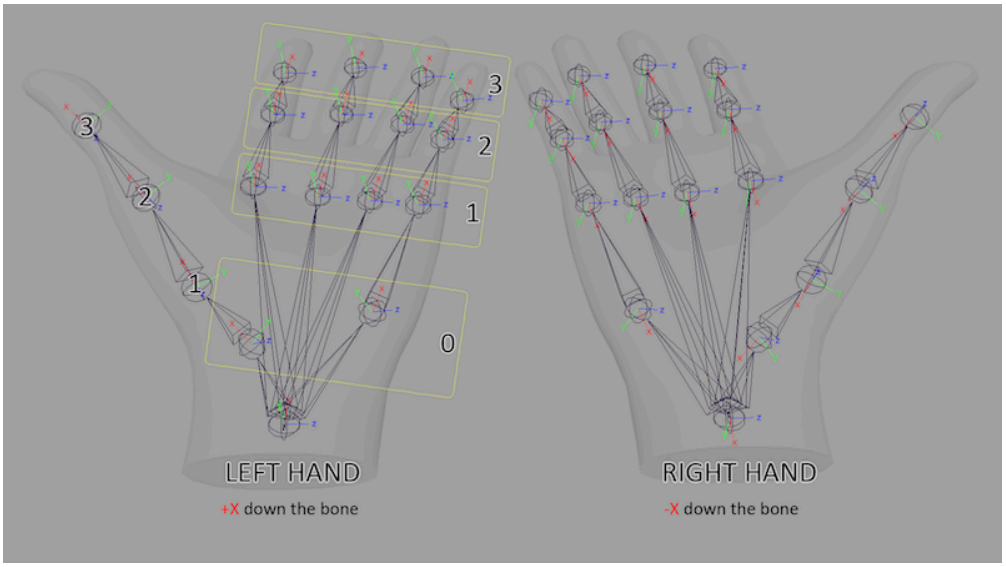
כמות הchannels בשירשור הבלוקים הוא 8,16,32,16,8.

לאחר הרצה של הקונבולוציות את התוצאה מריצים על ראש עם 54 נוירוני פלט. מריצים על התוצאות Tanh כדי לקבל טווח בין 0 ל 1.

הסבר פלט:

התוצאה מורכבת מ 54 נוירוני פלט מאחר שהיד שאיתה אני עובד, יש לה 18 מפרקים אפשריים ולכל מפרק יש 3 כיווני סיבוב אפשריים כלומר yaw, pitch, roll. סך הכל: $18 * 3 = 54$.

תמונה של מפרקי היד (יש להם צירים) שאיתה אני עובד בunity:



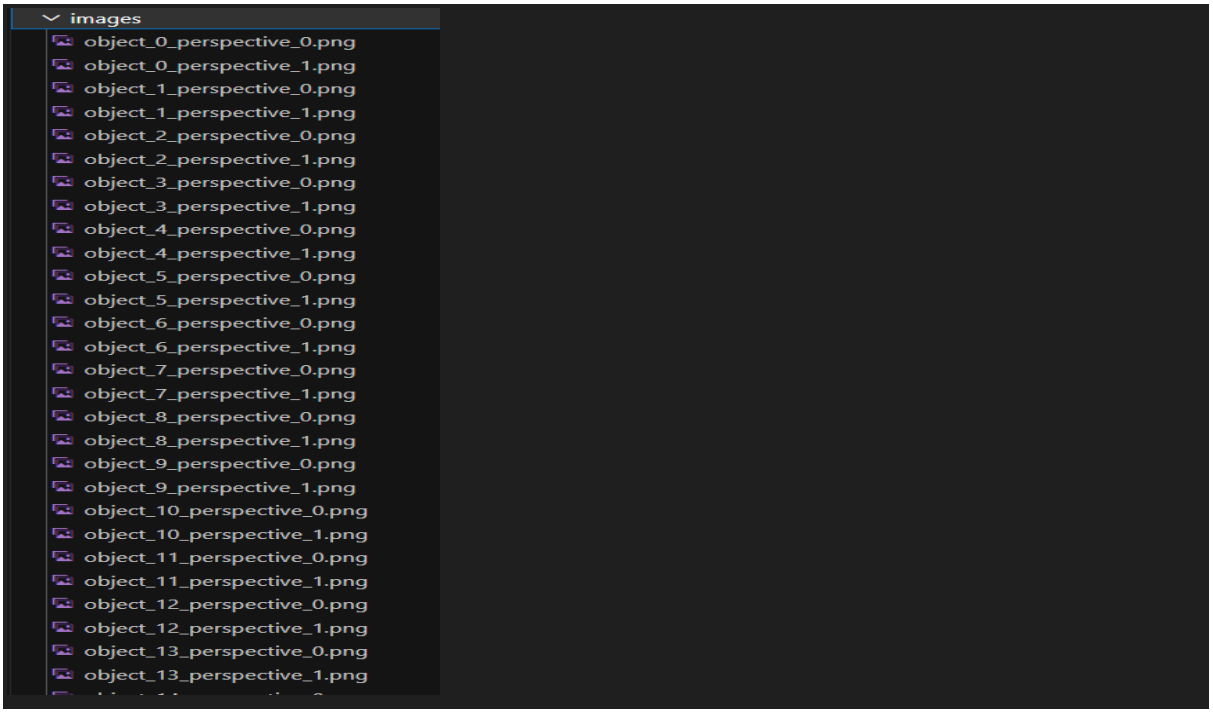
למה בין 0 ל1:

את התוצאה אני רציתי לקבל בין 0 ל1 אז הוספתי 1 לפלט וחילקתי ב 2.

בשביל שאוכל לעבוד בסימולטור עם הוזזה וטווחים הגיוניים של הסיבובים שאספתי מלפני כן. (באמצעות סיבובים את הפרקים והקלטה של הטווחים המינימליים והמקסימליים שאיתם אני רוצה לעבוד).

דאטה סט:

```
class HandDataset(Dataset):
    def __init__(self, paths):
        self.paths = paths

    def __len__(self):
        # returns the number of items in the dataset
        return len(self.paths.keys())

    def __getitem__(self, index):
        # gets an item at index
        imgs = []
        for j in range(2):
            img = plt.imread(self.paths['object_' + str(index)]['perspective_' + str(j)])
            imgs.append(img)
        cat = np.concatenate(imgs, axis=2)
        s = cat.shape
        cat = cat.reshape(s[2],s[0],s[1])
        tens = torch.from_numpy(cat)
        return tens

def CreateDatasets(paths):
    n = len(paths.keys())
    train_dataset = HandDataset(dict(list(paths.items())[:int(n * (9/10))])) # 90%
    test_dataset = HandDataset(dict(list(paths.items())[int(n * (9/10)):])) # 10 %
    return train_dataset,test_dataset

def CreateDataloder(dataset):
    return DataLoader(dataset, len(dataset),True)
```



הסבר דאטהסט:

בתוך הדאטה סט אני שומר תמונות משתי זוויות צילום שונות של אובייקטים מגוונים בunity.

מתוך הunity אני כותב לתיקיית images, את התמונות שאני מקבל מן המצלמות בunity, ושולח הודעה לפייתון כשזה מוכן, ואז הוא טוען את הדאטה משם ומקבל אותו לדאטהסט.

בכל איטרציה של האלגוריתם הגנטי הדאטה סט מתעדכן בהתאם למה שיש בunity scene, כשהמודל מתאמן.

פונקציית ההערכה של המודל:

מריצים את המודל על זוגות של תמונות. כל זוג זה שני תמונות של אותו האובייקט משתי זוויות שונות. ומקבלים פלטים לכל אובייקט זוג צילומים של אובייקט קלט אחד.

ואז לכל מנח ואובייקט ששמים אותם יחד בunity ואז מקליטים כמה רחוקה כל tip של אצבע מהאובייקט באמצעות מתיחה של Rays מכל tip.

ורציתי עוד מדד של כיוון האצבעות לאובייקט, לשם כך ייצגתי זאת באמצעות מכפלה פנימית בין וקטור שמתחיל מהtip של האצבע אל האובייקט לבין הוקטור כיוון של ה-ray.

זה נכון מאחר ש:

$$\cos cos\ (\alpha)\ \ * |A| * |B| = AB$$

כאשר α זו הזווית בין שני הוקטורים אזי כאשר $\alpha = 0$ נקבל ערך מקסימלי. ולכן ניקח את זה ביחס הפוך כי אני רוצה להגיד שהמרחק יהיה גדול יותר אם הוא לא בכיוון. כלומר במקום לחבר את המכפלה הפנימית, חיסרתי אותה מהערך שיצא מהdistance.

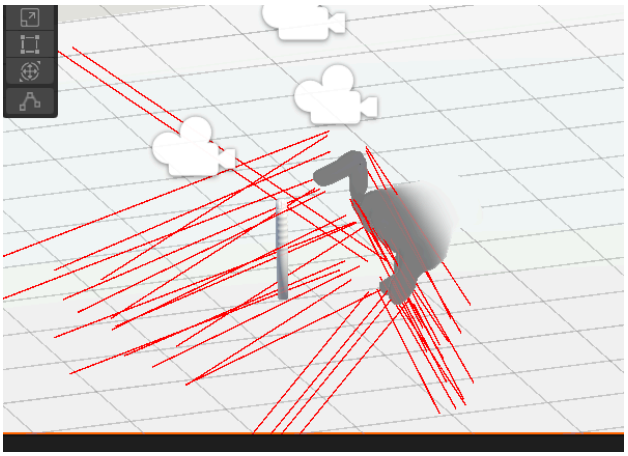
$$h(state) = \frac{\Sigma\left(dist_{finger_i}(object) - finger \cdot ray\right)}{num\ objects}$$

בקוד C# (כתוב בקוד איך לקבל את המרחקים מהאובייקט פלוס לצייר את הקרניים):

```
float[] distances = new float[5];
int i = 0;
foreach (string key in hand.Keys)
{
    if (key == "wrist")
        continue;
    int last = hand[key].Count;
    GameObject last_joint = hand[key][last - 1];
    GameObject beforeLast_joint = hand[key][last - 2];
    Vector3 direction = last_joint.transform.position - beforeLast_joint.transform.position;
    Vector3 ideal_direction = (obj.transform.position - last_joint.transform.position);
    float eps = 0.0001f;

    float correlation = Vector3.Dot(direction, ideal_direction);

    correlation = correlation < 0 ? correlation * 2 : correlation; // give extra weight to oposite side correlation

    if (Physics.Raycast(last_joint.transform.position, direction, out RaycastHit hitinfo, 20f))
    {
        Debug.Log(key + " Hit " + hitinfo.collider.gameObject.name + " in distance: " + hitinfo.distance);

        Debug.DrawRay(last_joint.transform.position, direction * hitinfo.distance, Color.red);

        // take distance with oposite relation to ray distance
        distances[i++] = hitinfo.distance - correlation;
    }
    else
    {
        Debug.Log(key + " Hit Nothing");

        Debug.DrawRay(last_joint.transform.position, last_joint.transform.TransformDirection(direction) * hitinfo.distance, Color.black);

        distances[i++] = 100 - correlation;
    }
}
```

דוגמה של הקרניים יוצאות:

זה ניראה בערך כך (התמונה כאן כוללת rays מעיבוד קודם):



אבל באופן כללי, רואים בתמונה יד שממנה יוצאים 5 Rays אדומים בכיוונים שונים שאוספים מרחק של האצבע מהאובייקט, שתלויים במנח האצבע הנוכחי.

האובייקט הלבן באמצע התמונה הוא האובייקט שנתבקשה היד לתפוס.

איך זה ניראה מנקודת מבט פייתון:

לאחר שאספנו את כל המרחקים של האצבעות מהאובייקט עשיתי ממוצע בין כל הוקטורים כדי לקבל את המרחק הממוצע של כל אצבע מאובייקט.

כך נימדדים הביצועים של המודל, ואלגוריתם האימון מנסה למקסם את הפונקציה הזו משיקולים טכניים שpygad דורש תמיד למקסם את ההערכה של המודל:

```python
def fitness_func(self,rays,index):
        """
        This function should recieve rays outputs as values
        of the distance from the fingers to the object to grab
        """

        ideal = torch.from_numpy(np.array([0,0,0,0,0]))
        loss_function = torch.nn.MSELoss()
        loss = loss_function(rays, ideal)
        eps =  0.00000001

        # we got minimization loss,
        # then we need to convert it to maximization loss
        # therefore we divide 1 by the loss + epsilon to avoid division by
zero
        fitness_value = (1 / loss + eps)

        return fitness_value.item()
```

שזה בעצם שקול ללמנמם את ה MSE של ה מרחקים מהאובייקט.

אימון המודל:

כדי לאמן את המודל השתמשתי בסיפרייה של אלגוריתמים גנטיים בפייתון שניקראת pygad.

הסיפרייה נוחה לשימוש ולהרצה של אלגוריתמים גנטיים על מודלים של פייטורץ'.

הפרמטרים שהשתמשתי בהם נתונים כאן:

```
def runGenetic(self, fitness_wrapper):
        torch_ga = torchga.TorchGA(model=model, num_solutions=100)
        num_generations = 50
        num_parents_mating = 10
        initial_population = torch_ga.population_weights
        ga_instance = pygad.GA(num_generations=num_generations,
                               num_parents_mating=num_parents_mating,
                               initial_population=initial_population,
                               fitness_func=fitness_wrapper,
                               save_best_solutions=True,
                               # save_solutions=True,
                               # parallel_processing=["thread" ,7],
                               crossover_type="two_points",
                               parent_selection_type="tournament",
                               mutation_probability=0.01,
                               mutation_type="random",
                               on_generation=lambda gen: print("Generation: ",
gen.generations_completed),
                               on_crossover=lambda a,b : print("on crossover"),
                               on_fitness= lambda gen_instance,b: print("on
fitness")
                               ,on_mutation=lambda a,b: print("on mutation"),
                               on_parents= lambda a,b: print("on parents")
                               )
        ga_instance.run()

        filename2 = "ga_instance2"
        ga_instance.save(filename2)
        print(ga_instance.summary())
```

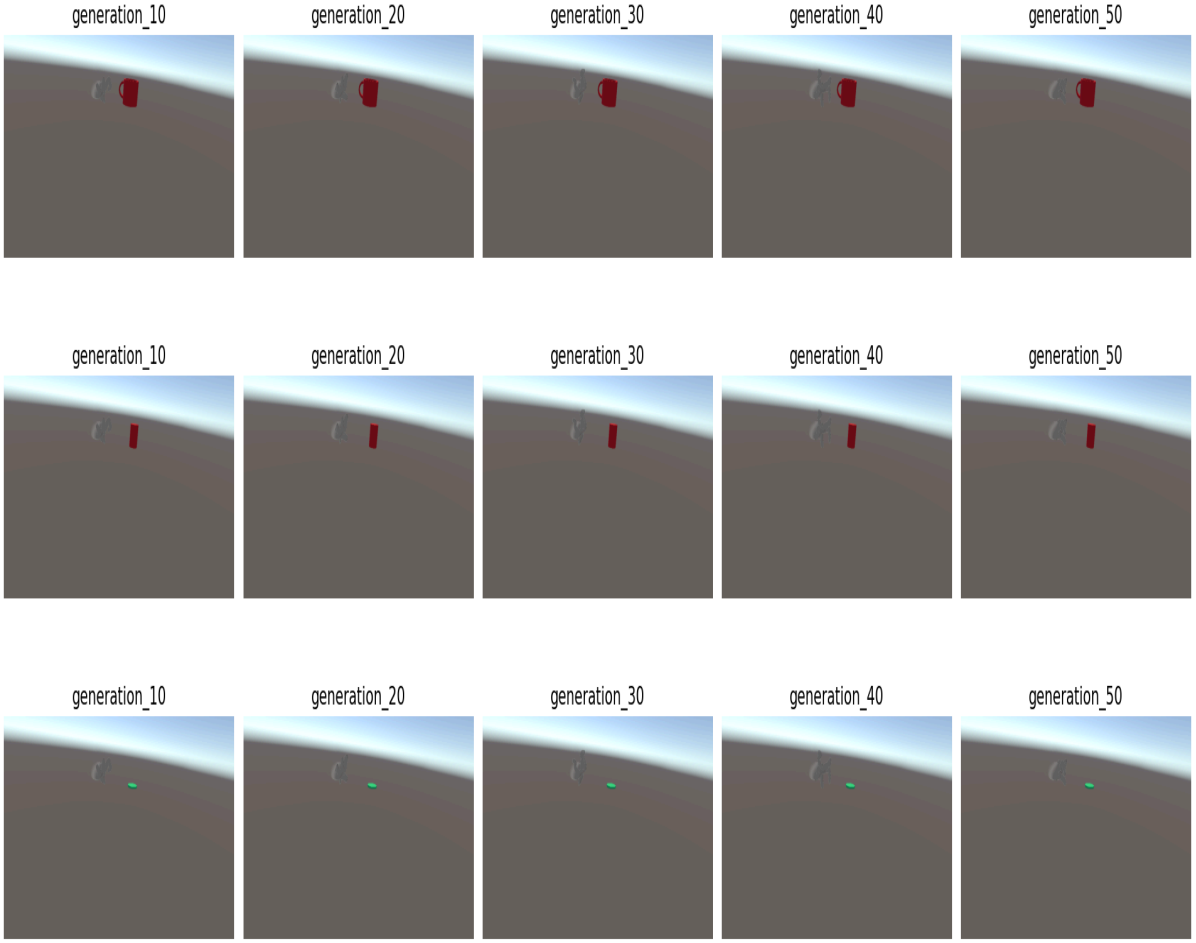
הסבר קוד pygad:

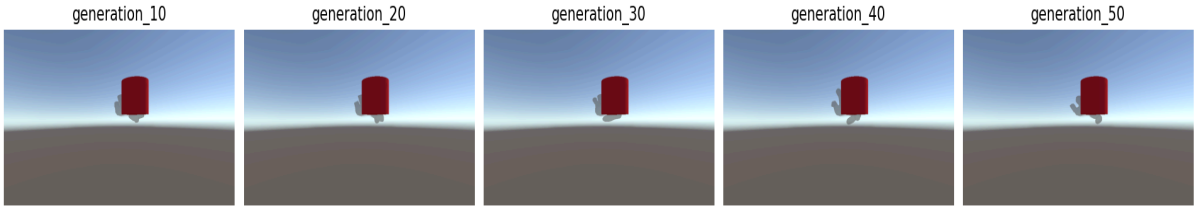
כאשר מספר הדורות הוא 50, מספר ההורים הנבחרים מתוך כל דירוג דור הוא 10, ומספר הילדים בכל דור הוא 100. שיטת בחירת ההורים זה בחירת k הורים באקראי ומתוכם ליבחור את הטובים ביותר. שיטת המיזוג DNA של ההורים היא two point crossover כלומר לבחור שני נקודות מתוך הDNA וליצור שני ילדים על ידי העתקת קטעים אלטרנטיבית בין החלוקה של הקטעים מכל הורה. זה טוב כי אז פרמטרים של ההורים שהיו טובים יכולים לעבור הלאה לדורות הבאים. כמובן שאפשר לבחור את אותו ההורה פעמיים לא נתתי מגבלה על זה. הפרמטרים הללו הם hyperparameters ובחרתי אותם מתוך ניסוי וטעייה וראיתי מה עובד יותר טוב.

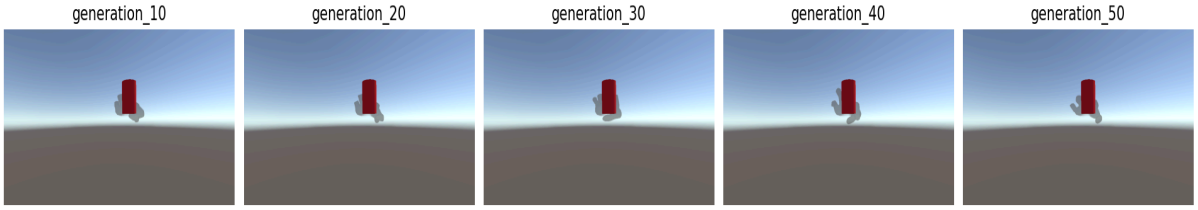
תוצאות:

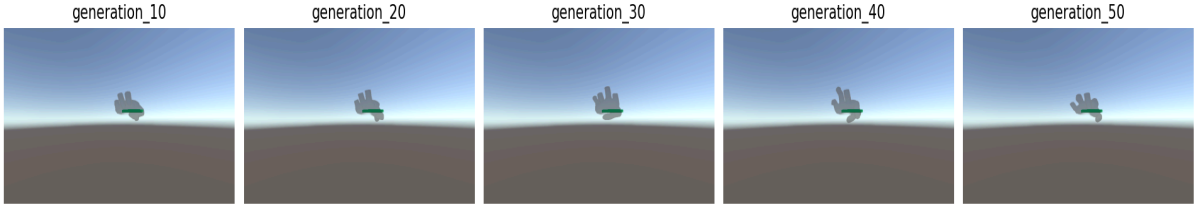
אלא התוצאות של המודלים הטובים ביותר בכל דור שניתנות משני זוויות צילום(אפשר להריץ את הסקריפט plot_results.py כדי לראות יותר טוב):

זווית 1:



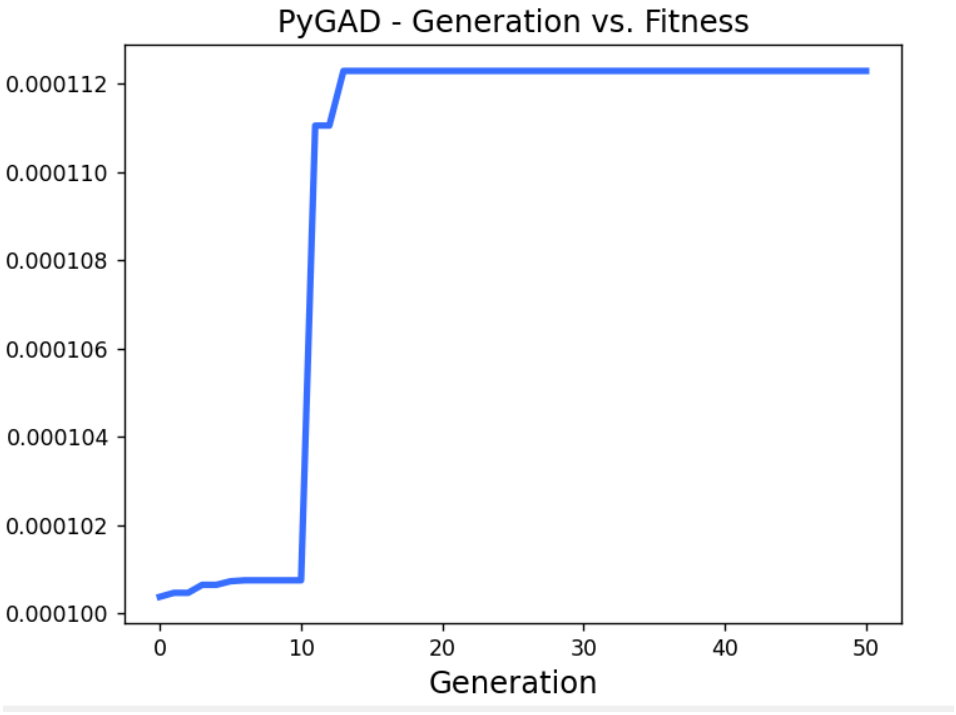






זווית 2:

תוצאות הלמידה לאורך דורות:



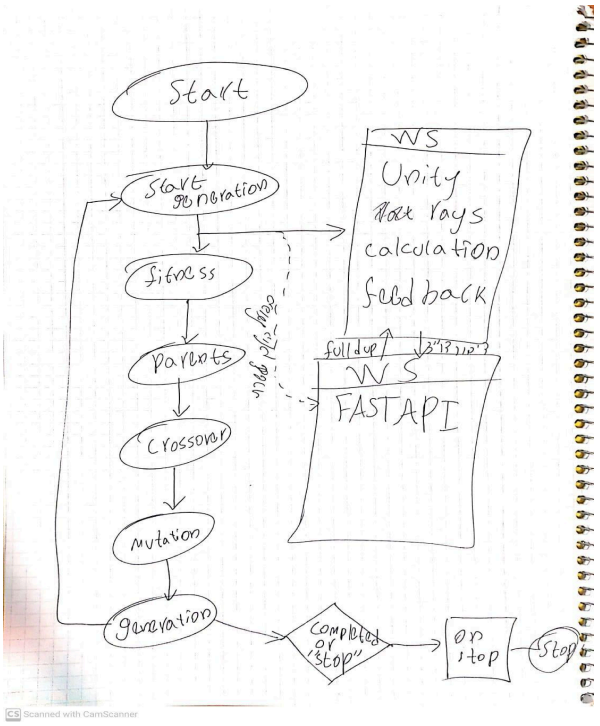
מסקנות והערות והארות שלי:

- לפי הגרף זה ניראה שהאלגוריתם לא מצליח ללמוד כמו שצריך את פונקציית ההערכה שלי, למרות שרואים פונקציה מונוטונית עולה, היא מאוד נמוכה וציפיתי לקבל ערכים יותר גבוהים בפונקציית ההערכה. מאחר שאני מחלק בסכום ריבועי המרחקים מהאובייקטים, אם פונקציית ההערכה היא בסדר גודל של 10^{-4} אזי, חילקתי ב $10,000$, ולפי החישוב שלי מאחר שנתתי לכל אצבע הערכה 100 אם אינה מצליחה לגעת באובייקט נקבל $100^2 = 10,000$. אזי התוצאות האלה לא טובות.
- קשה לאמן מודל CNN עם אלגוריתם גנטי (אולי סיבכתי אותו יותר מידי? אולי יש יותר מידי פרמטרים? אולי זה לא חכם להשתמש בתמונות בבעייה הזאת?)

דברים שהייתי צריך לעשות כדי שזה יעבוד:

- *כדי לחבר את הpython לunity בניתי שרת websockets בpython עם fastapi. ותיקשרתי איתו עם חבילת websocketSharp בunity.*
- *הייתי צריך לכתוב את כל הscenes בunity כולל הסקריפטים שמריצים אוגמנטציות ליד, ומשימים אובייקטים בהתאם, ולקחת תמונות של הscene משתי מצלמות במנוע.*
- *Fastapi שולח הודעות רק על הmain event loop אז הייתי צריך לחלק שני טרדים, אחד שיהיה השרת שישלח את ההודעות והשני שיריץ את pygad על מה שמתקבל מunity. עשיתי את זה עם ארכיטקטורה של producer consumer פשוטה עם טור חסין תנאי מירוץ.*

ארכיטקטורה:



לסיכום:

נהנתי לעשות את הפרויקט למדתי המון, לא מתחרט על כך למרות שלא הצלחתי ממש.