

מימוש מודל למידה בנושא

Human Pose Estimation

אריאל ורבין, לירן מנצורי, ניתאי דلغושן



מבוא

לחיזוי חלקי הגוף של אדם (ידיים, רגליים, מותניים, ראש, כתפיים וכדומה) הנתנו על ידי תמונה. Human Pose Estimation (בקיצור HPE, בעברית: הַעֲרָכָת תָּנוּמַת אָדָם) זאת משימה בראייה ממוחשבת

המשימה שלנו, אותה נתאר, תהיה לכתוב אלגוריתם שבහינתו תמונה המכילה בני אדם, יחויר לנו קווארדינאות (דו ממדיות) של היקן נמצאים איברים שונים בכל אחד מהאנשים בתמונה.

נתחיל עם הצגת הבעה בפירות.

- קיימים שלושה סוגים עיקריים של ¹Human Pose Estimation (HPE) :
- מודל מבוסט שלד, הנקרא גם מודל קינטטי, שמטרתו ליזות key points (מפרקים) כמו קרסוליים, ברכיים, כתפיים, גפיים וכדומה. באופן המאפשר לנו לחבר את key points הלאו ליצירת שלד המתאר את הפיזיציה של האדם.
 - מודל מבוסט קובי מתאר, שמטרתו לתאר את הפיזיציה של האדם בתמונה על ידי מערכת של קובי מתאר אשר מקיפים את האדם וחלקי גופו.
 - מודל מבוסט נפח, שמטרתו לתאר את פיזיצית האדם באופן תלת ממדי על ידי ייצוג ריאליסטי של הפיזיציה על ידי צורות גיאומטריות תלת ממדיות כמו גלילים ותיבות.

אנו נתמקד בעבודה על מודל מבוסט שלד, שכן אנו מאמינים כי קל להשוות 2 פיזיציות של בני אדם הנתונות על ידי שלד (זהה דרישת נctrax בהמשך הפרויקט).

- בין המודלים מבוסטי השلد, ניתן לחלק את אופו החיזוי שלהם לשתי שיטות עיקריות²:
- מלמעלה למטה. לפי שיטה זו, קודם נשתמש במודל ליזהוי בני האדם בתמונה ונמסגר כל בן אדם במלבן יהודי. ולאחר מכן נשתמש במודל ליזהוי המפרקים של כל אדם בנפרד על סמך המלבן שלו.
 - מלמטה למעלה. לפי שיטה זו, קודם נזהה את כל המפרקים בתמונה (הברכיים, המפרקים, המותניים וכדומה), ולאחר מכן נctrax להבין איזה מפרק שייך לאיזה אדם ולבנות את הפיזיציות הנפרדות של כל אדם.

אנו נתמקד בשיטה הראשונה, שכן היא נחשבת פשוטה יותר. נציין כי היא איטית יותר מהשיטה השנייה כיון שעליינו להפעיל את מודל זיהוי הפיזיציה עבור כל אדם בתמונה בנפרד. קיימים שלל ארכיטקטורות של רשתות ניורונים המתאימים למשימה זו. בחירת הארכיטקטורה שנשתמש בה היא שלב קרייטי במיוחד. לאחר מחקר ובדיקה של שלל ארכיטקטורות נפוצות – החלטנו להשתמש בארQUITקטורה הנקראת *ViTPose*.

¹ Human Pose Estimation: Ultimate Guide (2023 edition), Natassha Selvaraj. Kili.

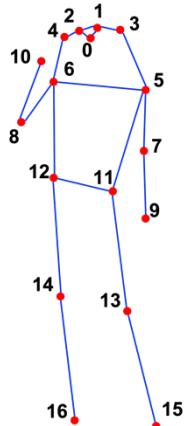
² A Comprehensive Guide on Human Pose Estimation (2022), Mrinal Singh Walia. Analytics Vidhya.

שלב ראשון – איסוף המידע והכנה

איסוף המידע הוא שלב קריטי בכל פרויקט של למידה מוגנה. ההצלחה של המודל תלולה באיכות, בכמות ובשונות של המידע בו משתמשים בשלב האימון. באימון נדרש מאות אלפי תמונות כדי להגיע למודל בעל דיוק גבוה.

נשתמש בDataset הנקרא COCO (Common Objects in Context) [ה קישור הבא](#).

COCO הוא Dataset נפוץ במיוחד למשימות של ראייה ממוחשבת, הוא מכיל מעל 200 אלף תמונות מתיוגות עבור מושגים של זהות, סגנון, ותמלול של תמונות. התוצאות של התמונות מכילות בין היתר bounding boxes עבור בני אדם (כלומר מבנים התוחמים את בני האדם בתמונה) ועבור כל בנאדם התיאוג מכיל גם 17 נקודות key points (דו ממדיות) המתארות את הпозיציה שלו.



ה-key points שהDataset מתייג הם: אוזניים, עיניים, פה, כתפיים, מרפקים, ידים, מותניים, ברכיים ורגליים.

על מנת שהמודל יוכל ללמידה בצורה טובה מהDataset, עליו להיות מאוד מגוון בסוג התמונות. ככלمر, עליו להכיל מגוון רחב של תמונות שונות בפרשtkיבת, בתאורה, ברקע, בצבעים, וכמוון בפוזיציות. באופן זה נמנע מהמודל לשנן מאפיינים לא נכונים מהתמונות ונאפשר לו להכליל בצורה טובה לתמונות חדשות. למשל, אם הדataset יכיל רק תמונות של אנשים עומדים, ייתכן מאוד שהמודל יקשה למושג פוזיציות של אנשים יושבים. לשמהחנה, COCO מציע מגוון רחב במיוחד של תמונות שונות.

כפי שציינו, אנו נעבד בשיטת Top-Down, ככלmr, התמונות שנאמדו בעזרתן את המודל יהיו של כל אדם בנפרד (בעזרת bounding box שנtauן בתיאוג התמונות – או *bbox* בקיצור – נחתוך כל אדם בתמונה למולבן נפרד עבורה). מעתה והלאה נניח שככל תמונה מכילה אדם אחד בלבד.

טרנספורמציה ואוגמנטציה

שלב הכרחי בעיבוד המידע לפני השימוש בו במהלך האימון, הוא טרנספורמציה ואוגמנטציה של התמונות. שלב זה מתייחס לעיבוד מוקדים של התמונות לפי שימוש בהם לאימון המודל. עיבוד התמונות יכול טרנספורמציה בה נקבע את גודל התמונות הנכונות למודל ונורמל את ערכיהם, ואוגמנטציה בה נשנה באופן אקרים מאפיינים בתמונה במטרה להגדיל את הגיוון והשונות בין התמונות.

עבור הטרנספורמציה, בחרנו שכל התמונות יכנסו למודל בגודל קבוע של 256×192 זאת על מנת להקל על הגדרת המודל ולשמור על התמונות קטנות יחסית (אך לא קטנות מדי כדי לא לפגוע ביכולת לפרש אותן). ובנוסף הפעלנו נורמליזציה על התמונות כדי לשמר על ההתפלגות של התמונות שנכנסות למודל אחידה, דבר המאיץ את תהליך האימון.

```
self.transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])
```

המטרה היא בעצם להביא את הממוצע וסטיית התקן של ערכי הצבעים ל-0 ו-1, בהתאם. ערך הממוצע מוחסן מכל פיקסל, וערך std מוחסן בכל פיקסל. בחרנו דווקא ערכים אלו כיון שראינו שמים חזרים על עצם במקומות רבים, וממלץ להשתמש בהם כאשר מאמנים על דאטאסט עם תמונות "רגילהות" – אנשים, מבנים, חיות, ועודומה (ערכים אלו ככל הנראה קרובים לתוחלת וסטיית התקן האמיטיים של התמונות).

עבור האוגמנטציה, נסובב, נשקף, ונמתח את התמונה בהסתברות מסוימת. זאת על מנת להגבר את הגיוון בין התמונות ולמנוע מהמודל לשנן/לנתת חשיבות למאפיינים לא רלוונטיים כמו זווית האדם, גודלו וכדומה. בחרנו את ההסתברויות הבאות עבור שלב זה :

- נגדיל/נכווץ את התמונה פי מספר המתפלג אחיד מ[0.65, 1.35].
- נסובב את התמונה בכמות מעלה המתפלגת אחיד מ[−90,90].
- נשקף את התמונה בהסתברות חצי.

כמובן נפעיל את הפעולות האלה גם על התוצאות של התמונה (למשל אם נסובב את התמונה ב 10° , נדאג להפעיל את הסיבוב הזה גם על התיאוג של התמונה כך שהPOINTS-keypoints שלה יהיו במקומות המתאים בחזרה).

שלב שני – מודל הרשת – ViTPose

מודל ViTPose הוא מודל המשמש כאמור לביצוע HPE. המודל מביא ביצועים גבוהים במיוחד (state-of-the-art) ועושה זאת אף תוך שמירה על ארכיטקטורה פשוטה באופן ייחודי המאפשרת לבצע את החיזוי ב מהירות.

נציין כי המודל שבחרנו נדרש לעבוד ב-Real-Time עבור היישום שלנו, כלומר עליו לבצע את חיזוי הпозיציה בזמן קצר במיוחד, תוך שמירה על דיקוגרף גבוה ככל הניתן. וזאת אחת הסיבות שבחרנו לעבוד דווקא עם ViTPose.

מודל זה פורסם במאמר "Simple Vision Transformer Baselines for Human Pose Estimation" בשנת 2022, על ידי Dacheng Tao, Yufei Xu, Jing Zhang, Qiming Zhang ו- Vision Transformers – שלראשונה הוצגו עבור מישימות של שימוש שפה טבעית ובשנים האחרונות דלפו למשימות ראייה ממוחשבת) במקום ארכיטקטורה "קלאסית" של רשתות קונבולוציה.

ניתן לחלק את ארכיטקטורת המודל לשני חלקים הנקראים `backbone` ו- `head`.

שלב ה `Backbone` (נקרא גם Encoder)

ה `backbone` מתייחס לחלק הראשון בארכיטקטורת המודל. חלק זה מקבל את התמונה כקלט ומהלץ ממנו פיצרים, כולם וקטורים ארכיטקטוריים מייצגים תכונות חשובות בתמונה אשר הכרחיות לביצוע משימת המודל. בארכיטקטורת ViTPose, חלק `backbone` מורכב מספר בלוקים של טרנספורמרים, המוחברים אחד אחר השני (כך שפלט של טרנספורמר אחד מוזן כקלט לבlok הטרנספורמר הבא).

אמנם, נזכר כי בлок הטרנספורמר מקבל סט של וקטורים כקלט. למרות שבמקרה שלנו הקלט הוא תמונה. לכן, לפניה שnezין את התמונה לבlok הטרנספורמר הראשון עלינו להמיר את התמונה לסט וקטורים. ההמרה צריכה להתבצע בצורה חכמה, שבני היתר תשמור בצורה טובת ככל הניתן על המבנה המרחבי של התמונה. נזכיר בוגוסף כי בлок הטרנספורמר מצפה לקבל סט של וקטורים $\{X_i\}_{i=1}^{N_X}$ שככל אחד ממימד D_X .

נשתמש ב *Patch Embedding*. על פי שיטה זו, נחלק את התמונה ל- "טלאים" (patches) בגודל קבוע (16×16 פיקסלים). נשים לב שהחלוקת לטלאים מפיקה מפיקה $H_p \cdot W_p$ טלאים.

את כל טלאי i נשים לב שטח $patch_i \in \mathbb{R}^{3 \times 16 \times 16}$ (כasher 3 זה מספר העורוצים המתאימים לRGB) שטח לווקטור X ממימד D_X על ידי טרנספורמציה לינארית, עליה נפרט מיד.

עבור הטרנספורמציה הילינארית, יהיו לנו $\{f_{W_i}\}_{i=1}^{D_X}$ פונקציות לינאריות עם פרמטרים $W_i \in \mathbb{R}^{3 \times 16 \times 16}$

כל אחת מוגדרת להיות

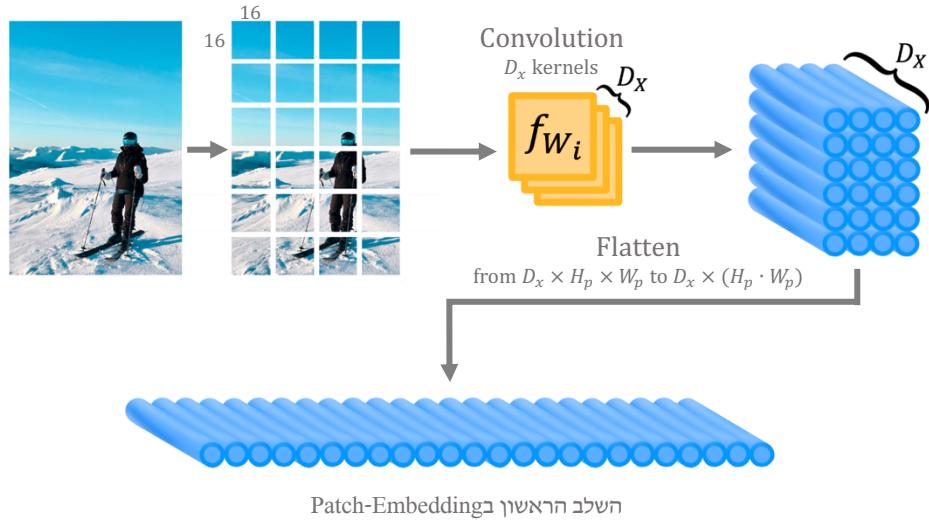
$$f_{W_i}(\text{patch}) = \sum_{n \in [3], m, k \in [16]} \text{patch}[n, m, k] \cdot W_i[n, m, k]$$

הפרמטרים W_i ילמדו חלק מתהליכי האימון. השימוש של הטליי יונדר להיות

$$\text{LinearProj}(\text{patch}) := \left(f_{W_1}(\text{patch}), f_{W_2}(\text{patch}), \dots, f_{W_{D_X}}(\text{patch}) \right)^T$$

נשים לב שאנו קיבלנו עבור הטליי וקטור ממימד D_X .

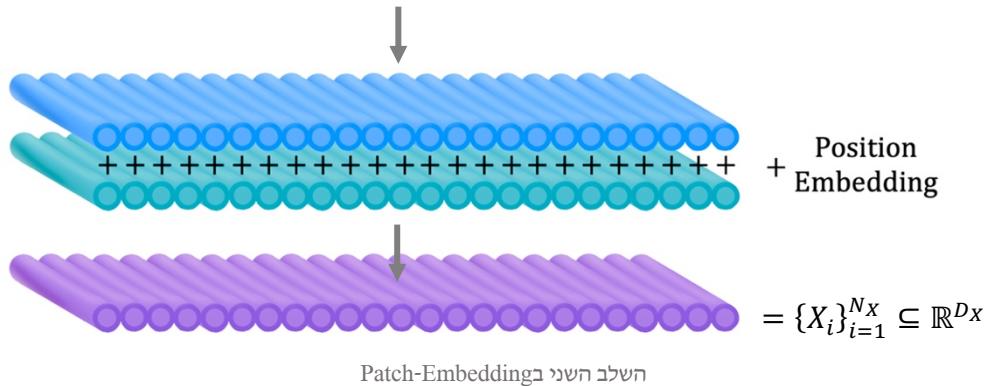
בחינת המימוש, נשים לב שתהליכי שקול לשכבות קונבולוציה יחידה, כך שכל פונקציה f_{W_i} היא בעצם פילטר יחיד, אשר זו על גבי התמונה בקפיצות של 16 ($stride = 16$) ומחשב *dot-product* עבור כל מיקום שלו בתמונה. נקבל גRID של וקטורים ממימד D_X כאשר כל וקטור מתאים לטליי בתמונה. ניקח את הGRID ונשיטה אותו לסט של וקטורים. נסכם את שלב זה עד כה על ידי שרטוט.



לא הוכחנו זאת, אך בлок הטרנספורמר (ובפרט שכבת ה-*Attention*) בלתי תלוי בסדר שבו מכנים את וקטורי הקלט. אמם, כדי לשמר על המבנה המרחבי של התמונה, אנו כן מעוניינים לתת משמעות לסדר של הווקטורים (שכן כל וקטור מותאים לטליי בתמונה, ויש חשיבות לסדר שבו הטלאים מסודרים). נעשה זאת באמצעות הוספה של *Position Embedding*.

אפשר למשה להוסיף אינדקס לכל וקטור – אך גישה זו אינה מביאה ביצועים טובים בפועל. במקום, נוסיף לכל אחד מהווקטורים וקטור של ערכים שנלמדים בתהליכי האימון (לכל וקטור של טליי, נוסיף וקטור ערך ייחודי עבורו). בתהליכי האימון המודל ילמד אילו ערכים כדאי לו להוסיף לכל אחד מהווקטורים, ובמילים

אחרות הוא למד כיצד לקודד את המיקום של כל טליי – וכך בעצם אנו נותנים משמעות למיקום של הטליאים.



ומכאן, קיבנו סט של וקטורים $\{X_i\}_{i=1}^{N_X}$. אוטם נעביר דרך L בלוקים של טרנספורמרים, כאשר כל בלוק שומר על המימד של הווקטורים (כלומר מקבל N_X וקטורים ממייד D_X ומחזיר N_X וקטורים ממייד D_X). על הפלט של בלוק הטרנספורמר האחרון נעביר שכבת Layer Normalization כדי להביא לשלב Head קלטים מנוומלים.

שלב ה-Head (נקרא גם Decoder)

שלב head, מתיחס לחלק השני בארכיטקטורת המודל. חלק זה מקבל את הפיצ'רים שהתקבלו בשלב backbone, ומבצע את החישוב השפכתיי בסופו מקבל את הפלט הרצוי והסופי של המודל. מטרתו בעצם לפרש את הפיצ'רים בצורה נכונה ולהסיק מהם את הפלט הסופי.

עבור שלב זה כאמור אנו מקבלים קבלת את הסט $\{F_i\}_{i=1}^{N_X}$ שיצא מבlok הטרנספורמר האחרון ומשכבה LN. הוקטורים בסט הם (עדין) מימד D_X . וצורך כמות הוקטורים בסט הוא W_p $N_X = H_p$ כמספר הטעאים.

כלומר הפלט מה backbone הוא מימד $D_X \times (H_p \cdot W_p)$

עבור השלב הנוכחי נסתכל על הסתן שוב בתווך גריד בגודל $W_p \times H_p$ (כלומר נסדר את N_X הוקטוריים באופן שבו יהיו מסודרים הטלאים בהתחלה). נקבל טנסור בגודל $D_X \times H_p \times W_p$ (גריד של $W_p \times H_p$ וקטוריים ממימד D_X כל אחד) והוא יהיה הקלט לשלב Head.

נzieין כי בסוף השלב, אנו נרצה לקבל N_K מפות חום (*heatmaps*) עבור כל אחד מ- N_K key-points שהוא מעונייננו למצוא בתמונה. כל מפת חום צריכה להתאים בגודלה לגודל התמונה, ובעצם תהיה לכל פיקסל בקלט מעין ערך הסתברות, המציין את הסבירות שהkey-point נמצא בפיקסל זה.

שלב Head יהיה מורכב משכבות הנקראות *Deconvolution* (או *Transformed Convolution*) ופונקציית אקטיבציה *ReLU*, ופונקציית נורמליזציה *Normalization*.

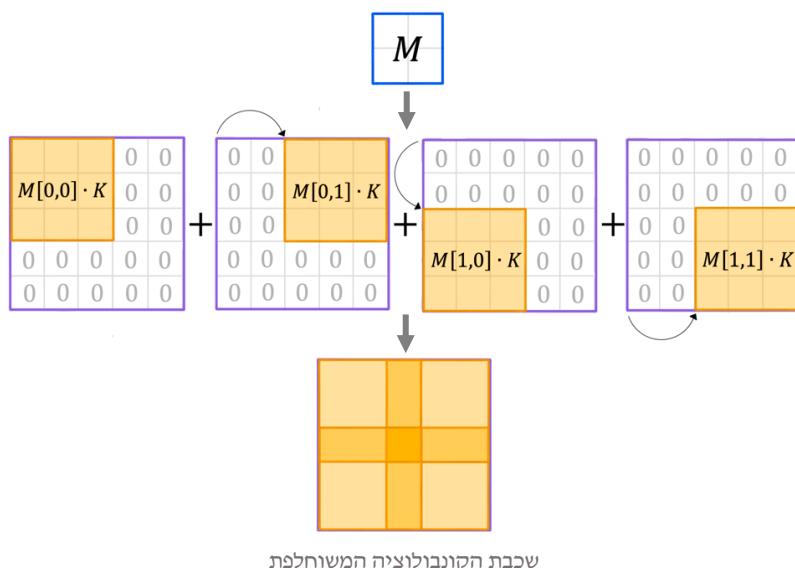
Transformed Convolution

מטרת השכבה היא בעצם להעלות את הממדים המרוחבים (אורך ורוחב) של הקלט זאת בנגדוד לשכבה הconvolutional הרגילה שזוosa בדיק את ההפוך ומקטינה את הממדים המרוחבים של הקלט. נסמן את הקלט להיות $p \in \mathbb{R}^{H_p \times W_p}$ (עבור ההסבר נתעלם מ- D_x מספר העורצים – שכן בפועל הקלט הוא $W_p \times H_p$).

נראה כי הפלט של השכבה יהיה מממד $W_p \cdot 2 \times H_p \cdot 2$ (פי 2).

בדומה לשכבת convolutional, יהיה לנו פילטר K (באורך – בגודל 3×3). בנגדוד לשכבת convolutional, הפילטר יעבור על כל איבר $M[i, j] \in \mathbb{R}^{3 \times 3}$ בקלט ויכפיל את ערך האיבר בכל הפילטר לקבלת $M[i, j] \in \mathbb{R}^{3 \times 3}$. לאחר מכן הפילטר יוזז לאיבר הבא ויבצע את אותה פעולה. נזהה מעוניינים לסקום את כל התוצאות מכל מקום של הפילטר על הקלט – אופן הסכימה תבצעו לפי stride שנקבע.

בעצם מגדיר לנו איפה למקם את $K \cdot M[i, j]$ הנוכחי במטריצת הפלט, ובעצם מגדיר לנו את גודל מטריצת הפלט. נסביר את הכוונה באמצעות האירור הבא. באירור מטריצת הפלט $M \in \mathbb{R}^{2 \times 2}$ וגודל הקרן $K \in \mathbb{R}^{3 \times 3}$. נראה כי אנו זוזים $stride = 2$ אלמנטים ימינה/למטה במטריצת הפלט כאשר אנו סוכמים את $M[i, j] \cdot K$ הבא (مبוטא באמצעות החיצים). מטריצת הפלט המתקבלת היא בגודל 5×5 . נראה למשל כי האיבר האמצעי במטריצת הפלט הוא היחיד שתלווה בכל ארבעת איברי הפלט במטריצת M .



פרמטר נוסף הנקרא *padding*, קובע כמה שורות ועמודות נמחק מטריצת הפלט (לא נרצה לכלול שורות ועמודות שתלוויות רק בפיקסלים בודדים ממטריצת הפלט). כלומר נמחק את מסגרת מטריצת הפלט בעובי המתאים לערך *padding*.

באופן כללי, הגודל של מטריצת הפלט המתקבלת נקבעת על פי הנוסחה

$$N' = \text{stride}(N - 1) + \text{kernelSize} - 2 \cdot \text{padding}$$

כאשר N זה האורך/רוחב לפניו/אחרי השכבה. (בדוגמה, $0 \cdot 2 - 2 = 2(2 - 1) + 3 = 5$ ובמימוש שלנו,

$$\text{kernelSize} = 4, \text{stride} = 2, \text{padding} = 1$$

ולכן מתקיים $N = 2(N - 1) + 4 - 2 \cdot 1 = 2(N - 1) + 4 - 2 = N$ כלומר השכבה מגדילה את מטריצת הפלט פי 2 באורך וברוחב.

בהסבר הנחנו שיש ערוץ אחד בקלט ופלט. אם הפלט לשכבה הוא בעל C ערוצים, נשתמש בפילטר בגודל 3×3 (ובעצם עדין הפלט שלנו יהיה מטריצה דו ממדית כלומר עם ערוץ אחד). אם מעוניינים בפלט בעל C' ערוצים, אפשר להשתמש ב- C' פילטרים שונים ועבור כל פילטר לקבל מטריצה בגודל $W' \times H'$, ולהצמיד את כלו יחד לקבל טנסור בגודל $W' \times H'$ עם C' ערוצים.

Batch Normalization

דיברנו בעמודים הקודמים על Layer Normalization המבצע חישוב דומה. גם כאן המטריה היא להביא לשכבה הבאה קלט מנורמל כדי למנוע בעיה הנקראת Internal Covariance Shift. אמנם החישוב יהיה מעט שונה.

הפלט משכבה הקונבולוציה המשוחלפת הוא טנסור בגודל $W' \times H' \times C'$. נסתכל על קלט זה כסט של $W' \cdot H'$ וקטורים כך שכל וקטור ממימד C . כמובן, אנו מקבלים לשכבה BN סט $\mathbb{R}^C \subseteq \{X_i\}_{i=1}^{W' \cdot H'}$.

בניגוד לLN, שם חישבנו ממוצע ושונות עבור כל i נפרד, לאורך כל C הפיצ'רים שלו, BN אנו נחשב ממוצע ושונות עבור כל אחד מ- C הפיצ'רים בנפרד, לאורך כל סט הווקטורים (מעין שחולוף). במלילים אחרות, במקומות לנורמל כל וקטור בסט הפלט לפי הפיצ'רים שלו, נורמל כל פיצ'ר לפי הווקטורים שבסט הפלט.

ולבשו!

פונקציית *ReLU* היא פונקציה אקטיבציה שמטרתה בין היתר לשמש כפונקציה לא לינארית המחברת בין שכבות לינאריות (למשל שכבות קונבולוציה או במרקחה שלנו, שכבות קונבולוציה משוחלפות).

הסיבה לכך הוא שרשור של פונקציות לינאריות נשאר לינארי, וכך בעצם היכולת החישובית של שרשור השכבות הלינאריות אחת אחרי השניה נשאר פונקציה לינארית וشكול מבחינה חישובית לשכבה יחידה. עם הוספה של פונקציית אקטיבציה לא לינארית בין כל שתי שכבות, אנו מונעים זאת.

הפונקציה פועלת איבר-איבר על הפלט ומוגדרת להיות

$$\text{ReLU}(x) := \max\{x, 0\}$$

שכבה Head בארQUITktורה ViTPose, מורכבת משכבה קונבולוציה משוחלת, אחרת שכבת BN ואחריה .ReLU. ואז שוב – שכבה קונבולוציה משוחלת ואחריה שכבת BN ולבסוף .ReLU

נזכיר כי הפלט Head הוא טנסור בגודל $D_X \times \frac{ImageWidth}{16} \times \frac{ImageHeight}{16}$ (ה D_X הוא מספר העורצים).

כאמור, כל אחת משכבות הקונבולוציה המשוחלת מגדילה את הממדים המרוחבים פי 2. בנוסף,anno מגדירים בכל שכבה צו 256 פילטרים נפרדים, כך שהפלט של שכבה הקונבולוציה המשוחלת הראשונה היא

$$256 \times \frac{ImageWidth}{8} \times \frac{ImageHeight}{8}$$

הBN וReLU לא משפיעות על הממדים. וכך שכבת הקונבולוציה המשוחלת השנייה מוציאה פלט במדם

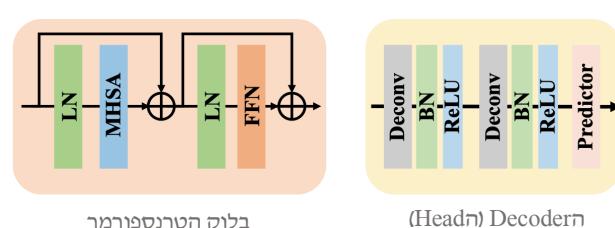
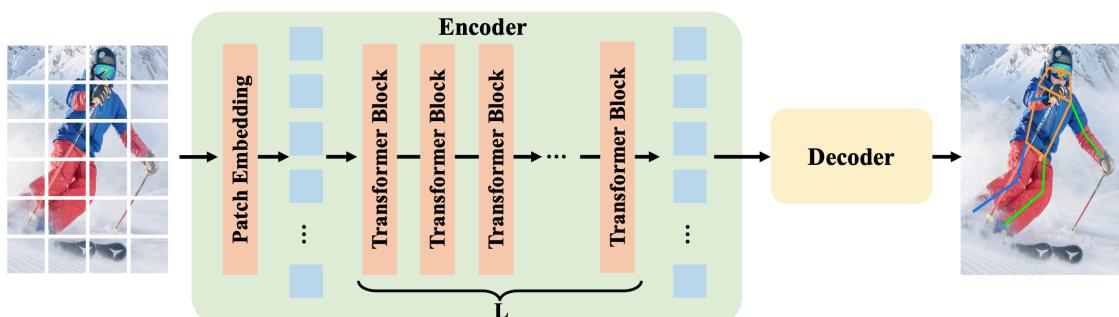
$$256 \times \frac{ImageWidth}{4} \times \frac{ImageHeight}{4}$$

נרצה לקבל בסוף N_K מפות חום, כלומר טנסור בגודל $N_K \times ImageWidth \times ImageHeight$. כך שהמפה $h[i] \in [N_K]$ תשמור עבור כל פיקסל את הסבירות שהזט-point key- נמצא בו.

ההמראת 256 עורצים N_K תבוצע באמצעות שכבה קונבולוציה (רגילה) עם N פילטרים בגודל $1 \times 1 \times 256$. כלומר נשתמש בשכבה קונבולוציה מיוחדת בעלת קרנול עם ממדים מרוחבים השווים לו. שכבה צו נועדה להגדיל את מספר העורצים מבלי לשנות את הממדים המרוחבים של הקלט.

נשאר את הממדים המרוחבים להיות רבע מגודל התמונה, ופושט "נמתח" אותם באופן נאיבי בשלב ה-Post Processing.

זהו! נסכם את הארכיטקטורה במלואה בشرطו שנמצא במאמר של הארכיטקטורה, ניתן להבחין בכל השלבים שדיברנו עליהם.



שלב שלישי – אימון המודל

לאחר שדיברנו בהרחבה על איך המודל בני, נדבר על תהליך הלמידה שלו (תהליך אימון המודל).

נשים לב שהמודל מורכב ממשקלים רבים שאנו צריכים "לבחור" (משקלי המטריצות W_V, W_Q, W_K בשכבות ה-Attention, המשקלים של הפילטרים בשכבות הקונבולוציה והקונבולוציה המשוחלפת, המשקלים בטרנספורמציות הילינאריות של שלב *Patch-Embedding* ועוד). למעשה, במודל שלנו, יש 86 מיליון (!) משקלים (נקראים פרמטרים) אותם נצטרך לבחור. הרעיון הוא למצוא את הפרמטרים האופטימליים שיתנו לנו את התוצאות הטובות ביותר.

באופן כללי האימון מורכב מהזנה של התמונות בדאטאסט לתוך המודל, וчисוב *loss* (כלומר – הפסד) בין התוצאות של המודל עבור התמונות לבין התוצאות המקוריים. ככל שה*loss* נמוך יותר, כך ביצועי המודל מדויקים יותר. אם ננסה לפרמל,

$$loss_W(image, keypoints) = d(ViTpose(image), keypoints)$$

כאשר *keypoints* הוא התיאוג המקורי של התמונה, ו*d* זו מטריקה כלשהי. נשים לב שפונקציה זו תלויות בפרמטרים רבים שמנדרירים את המודל. היינו רוצים להיות מסוגלים לגזר את הפונקציה לפי W ולמצוא מינימום. אולם, W מכיל כאמור עשרות מיליון פרמטרים, ומשימה זו היא בלתי אפשרית מבחינה חישובית.

לכן, השיטה הכללית לקירוב המינימום של הפונקציה נקראת *Gradient Descent*. לפי שיטה זו, נתחל עם פרמטרים ההתחלתיים כלשהם θ_0 ובכל איטרציה נזין מספר של תמונות למודל, ונחשב את *loss*. בעזרתו, נחשב את הגרדיינט (הנגזרת) של *loss* לפי הפרמטרים ונתקדם בכך לכיוון ההפוך לגרדיינט (כלומר במודד הפונקציה ביחס למצב הפרמטרים הנוכחי שלנו). זאת בתקופה שאם נעשה מספר רב של איטרציות כלו אנו נשיך לזרת ולהגיע לנקודה המזערת את *loss* כמה שניתן. למרות שבשיטה זו עדין צריך לחשב את הגרדיינט, לא צריך להשווות אותו לו ולמצוא מינימום. פורמלית, נוסחת עדכון הפרמטרים תהיה

$$\theta_t \leftarrow \theta_{t-1} - \gamma \cdot \nabla_{\theta} loss(\theta_{t-1})$$

כאשר γ נקרא קצב הלמידה והוא קובע את גודל הצעד שנבצע.

עבור שלב האימון, נצטרך לבחור פונקציית *loss*, והיפר פרמטרים נוספים עליהם נפרט.

בחירה פונקציית הפסד

כאמור עליינו לבחור דרך אמפירית לחשב את *loss* (או בעברית, פונקציית הפסד), פונקציה אותה נצטרך לחשב ולגזר ובאמצעותה לבצע אופטימיזציה למודל ולשפר אותו בהתאם לדאטאסט. פונקציית הפסד שבחרנו היא פונקציית *Joints MSE Loss*.

נזכר כי המודל מוציא עבורהנו $17 = N_K$ מפות חום (אחת עבור כל key-point) כאשר כל אחת בגודל $\frac{ImageWidth}{4} \times \frac{ImageHeight}{4}$. כל ערך במונה מציג רמת ביטחון שהkey-point נמצא דזוקא שם.

ניתן למשל להפיק ממפת החום את key-points שהמודל ניבא, ולהשכבר מרחק אוקלידי בין התוצאות שהמודל ניבא לבין תיוג התמונה ("התשובות הנכונות"). אמנם, שיטה זו אינה לוקחת בחשבון את רמת הביטחון של המודל ומונעת את הפלט שלו.

במקרים, ניקח את תיוג של התמונה ונפיק ממנה מפת חום בעזרת פונקציית גaus, כאשר סביר התשובה הנכונה יש אזור "חסם". פונקציית גaus הדו ממדית מוגדרת להיות

$$Guassian(x, y) := \exp\left(-\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma^2}\right)$$

כאשר (x_0, y_0) הוא key-point הנכון על סמך תיוג. אנו בחרנו $\sigma = 3$.

נפיק מפת חום כזו בגודל $\frac{ImageWidth}{4} \times \frac{ImageHeight}{4}$ אשר מתאימה לגודל פלט המודל³ ונחשב את ממוצע הריבועים בין כל ערך בModelProperty החום שהמודל ניבא, לבין מפת החום שייצרנו בעזרת פונקציית גaus והתיוגים הנכונים.

$$JointsMSELoss := \frac{1}{N_K} \cdot \sum_{k=1}^{N_K} \left(\frac{1}{W \cdot H} \cdot \sum_{x, y \in W \times H} (b^k(x, y) - b_*^k(x, y))^2 \right)$$

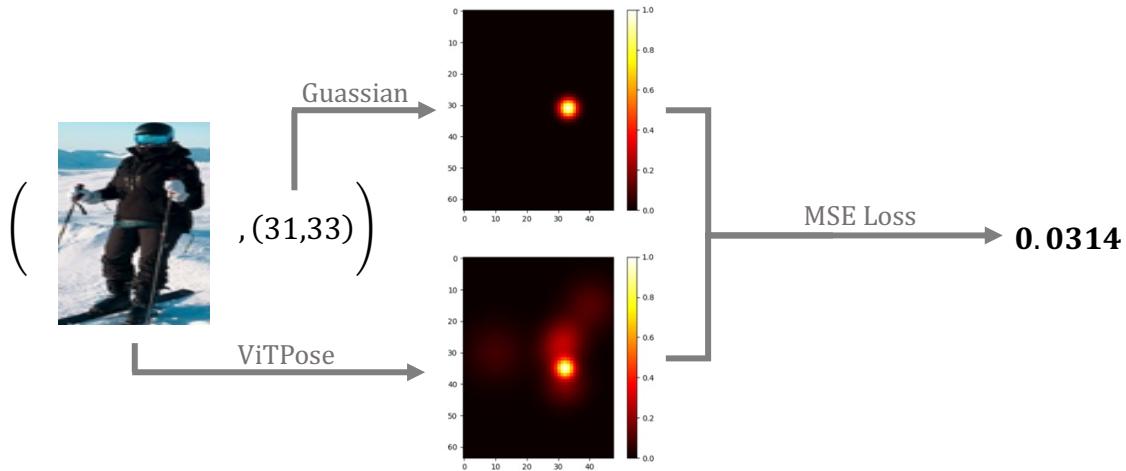
כאשר

- $N_K = 17$ - key-points
- W, H הם אורך ורוחב מפות החום (רבע מגודל התמונה המקורית).
- b^k זו מפת החום שהתקבל מהמודל עבור key-point ה- k .
- b_*^k זו מפת החום שהתקבל בעזרת פונקציית גaus והתיוג הנכון של key-point ה- k .

נבהיר, כי ככל שModelProperty החום שהמודל הפיק קרוביה יותר למפת גaus, כך loss נמוך יותר. נשים לב גם שModelProperty גaus מכילה ערכים גבוהים גם סביר key-point הנכון ולא רק בנקודה המדויקת, כך שאם המודל יניבא מפת חום עם "אזור חסם" שהמרכזו שלו מעט סוטה מן התשובה הנכונה – נוכל "לסלוח" על כך וה-loss עדין יצא נמוך באופן ייחסי.

נשים לב בנוסח שאנו נعنيש את המודל אם הינו לו מספר "אזורים חמימים" או אזור חסם רחב מידוי/בעל ערכים נמוכים (המעידים על חוסר ביטחון של המודל).

³ משום שהkey-points מתאימים לגודל התמונה המקורית, נחלק את הקואורדינטות ב-4 כדי להתאים אותן למפת החום.



чисוב loss. כאשר מצד שמאל נתון התמונה והתיוג שלה עבר אחד מהטורים.key-points loss. נפיק מפה חום עבור התיאוג הנוכחי ונשווה אותה למפה החום שהמודל ניבא.

בחירה שיטת אופטימיזציה – AdamW

דיברנו על הרעיון הכללי של שיטת Gradient Descent שמטרתה למצוא את הפרמטרים של המודל המזערים את פונקציית loss, תוך גזירה של הפונקציה והתקדמות בצעדים בכוון החוף לגרדיאנט (במורד הפונקציה בתקווה להגיע למינימום שלה). שיטה זו יש ווריאציות רבות. אנו בחרנו להשתמש בשיטה הנקראת *AdamW*. שיטה זו בעצם משלבת את החזוקות של מומנטום - *RMSProp*.

- מומנטום (או תנופה) מחשב את הצעד הנוכחי שיש לבצע על סמך ממוצע משוקל של הגרדיאנטים שהיו (ולא רק על פי הגרדיאנט האחרון). הדבר מאפשר לנו בעצם לצבור "תנופה" ובעזרתיה לעקוף ולא להיתקע במינימום מקומיים או באזוריים שטוחים של פונקציית ההפסד. ממוצע זה נקרא **מומנט ראשון**.

- *RMSProp* בעצם מחשב את גודל הצעד הנוכחי על סמך ממוצע משוקל של ריבועי הגרדיאנטים שהיו (ולא רק על פי הגרדיאנט האחרון). וככל שמדובר זה יותר גדול, כך הצעד שנעשה יהיה קטן יותר. כך, אנו נתקדם בצעדים "עדינים" יותר עבור משקלים שונים להשתנות בצורה דרסטית יותר, ונתקדם בצעדים יותר "גדולים" עבור משקלים קבועים להישאר קבועים. כך אנו מאפשרים התכנסות יציבה ומהירה של כל המשקלים. ממוצע זה נקרא **מומנט שני**.

чисוב הממוצע המשוקל בשני המקרים דורש בחירה של פרמטר β , לפיו נחשב

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \text{grad}_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\text{grad}_t)^2$$

נשים לב ש m_t מחושב עבור התנופה, ו v_t מחושב עבור *RMSProp* (מעלים בו את הגרדיאנט בריבוע). ככלומר, הגרדיאנט הנוכחי קיבל משקל מקסימלי β והמשקל ידוע בכך אפסוננטיאלי ככל שהגרדיאנט ישן יותר (הגרדיאנט grad_{t-1} קיבל משקל β^2 , אחריו grad_{t-2} קיבל משקל β^3 וכדומה).

נתאר פסאודו קוד של AdamW באופן הבא⁴

AdamW($\gamma, \beta_1, \beta_2, \theta_0, \lambda$):

```

 $(m_0, v_0) \leftarrow (0,0)$                                 # First and second moments
for  $t = 1$  to ... do:
     $grad_t \leftarrow \nabla_{\theta} Loss(\theta_{t-1})$           # Calculate current gradient
     $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) grad_t$       # Update first moment
     $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) (grad_t)^2$     # Update second moment
     $\hat{m}_t \leftarrow m_t / (1 - (\beta_1)^t)$                 # Bias correction
     $\hat{v}_t \leftarrow v_t / (1 - (\beta_2)^t)$                 # Bias correction
     $\theta_t \leftarrow \theta_{t-1} - \gamma \cdot (\hat{m}_t / (\sqrt{\hat{v}_t + \epsilon}) + \lambda \cdot \theta_{t-1})$  # The step

```

כאשר המשתנה γ הוא קצב הלמידה (סקלר) הקובע את גודל הצעד הסופי שנבצע.

המשתנים β_1, β_2 משמשים עבור חישוב המומנט הראשון והשני.

המשנה θ_0 הוא המצב ההתחלתי של משקלים המודל.

נשים לב למשל שהצעד מתבסס על \hat{m}_t שמורכב מממוצע משוקל של הגרדיאנטס הקודמים, ולא רק של האחרון. בהתאם למומנטום. בנוסף, עבור פרמטרים בעלי גרדיאנט גבוה (כלומר נוטים להשתנות בצורה דרסטית יותר), הערך \hat{m}_t יהיה גדול ולכן נתקדם בצעד קטן יותר (שכן אנו מחלקים את גודל הצעד ב- \hat{v}_t),

בהתאם ל-*RMSProp*

והמשנה λ נקרא *weight decay* והוא משמש עבור רגולרייזציה למניעת ערכאים גבוהים מידי עבור הפרמטרים. הסבר פשוט לכך הוא שאנו בעצם מחסירים את $\theta_{t-1} \cdot \gamma \cdot \lambda$ מהפרמטרים ובכך שומרים עליהם נמוכים. הסיבה שנרצה ערכאים נמוכים היא שערך גבוהים לרוב מעדים על מצבים של התאמת יתר.

⁴ הפסאודו-קוד מתבסס על המימוש של AdamW בספריית PyTorch

בחירה היפר-פרמטרים

מלבד פונקציית ההפסד שגמ נחשבת היפר פרמטר, הגדרת המודל ותהליכי האימון דורש בחירה של פרמטרים שאינם יכולים להילמד בתהליכי האימון שכן הם מגדירים את הארכיטקטורה המדויקת או את אופן האימון.

נסכם את היפר-פרמטרים בטבלה.

טבלה עבורה היפר-פרמטרים המגדירים את ארכיטקטורת המודל המדויקת.

הערך שבחרנו	תיאור תפקידו	שם היפר-פרמטר
768	מימד סט הוקטורים שנכנס ויוצא דרך הטרנספורמים. אנו קראנו לפרמטר זה D_X במהלך ההסבירים (ראו עמוד 5). אך הוא נקרא גם d_{model} במאמר ⁵ Attention Is All You Need.	$d_{model} = D_X$
192×256	בחרנו כזכור בשלב הטרנספורמציה של התמונות יכנסו למודל בגודל קבוע ושווה 192×256 .	<i>Image Size</i>
16×16	גודל כל טליי בעת חישוב ה-Patch Embedding. בהסבירים כבר הנקנו גודל זה להיות 16×16 .	<i>Patch Size</i>
48×64	כפי שהסבירנו כבר, נשים לב שגודל זה הוא בדיקות רבע מגודל התמונה המקורי (אורך ורוחב).	<i>Heatmap Size</i>
12	מספר בלוקי הטרנספורמים שנצמיד אחד לשני בשלב ה- <i>backbone</i> .	<i>Depth</i>
12	כמות הראשים בשכבה ה-Attention של כל טרנספורמר.	<i>Number of Heads</i>
2	כמות שכבות הconvolutional המשוחלפות ב- <i>decoder</i> .	<i>Deconvolution Layers</i>
4×4	גודל הפילטרים בשכבה הconvolutional המשוחלפות.	<i>Decoder Filter Size</i>
0.3	הסתרות לדלג על שכבות שלמות (בניגוד ל- <i>dropout</i> , שמלג על נוירונים ספציפיים) בטרנספורמים, חלק משיטות רגולרייזציה במטרה לעזרה למודל להקליל בצורה טובה יותר.	<i>Drop Path Rate</i>

⁵ Vaswani, A., et al. (2017). Attention is all you need.

טבלה עבורה ההיפר פרמטרים שגדירים את תהליך האימון.

שם ההיפר-פרמטר	תיאור תפקידו	הערך שבחרנו
Total Epochs	מספר האיטרציות הכולל על כל הדאטאסט.	210
Learning Rate	קצב הלמידה של המודל – גודל הצעדים שניקח	$5e - 4^*$
Optimizier	שיטת האופטימיזציה – פירטנו עלייה.	<i>AdamW</i>
Betas	הערכים β_1, β_2 המגדירים את האיטרציות של <i>AdamW</i>	$(\beta_1, \beta_2) = (0.9, 0.99)$
Weight Decay	עבור רגולריזציה (הסבירנו עליו).	$\lambda = 0.1$
Loss Function	פונקציית ההפסד עלייה הסברנו.	<i>Joints MSE</i>
Batch Size	כמויות התמונות שניקח בכל צעד של האופטימיזציה לחישוב loss – כל מעבר שלם על הדאטאסט (epoch אחד) מורכב מ- $\frac{ Dataset }{ Batch Size }$ צעדים.	64

* במספר epoch 170 ו-200 נעדכו את קצב הלמידה להיות פי 1.0 כדי לעדן את הצעדים לקראת סוף האימון.

אתחול משקלים המודל

גישה אחת לתחילת האימון היא לבחור משקלים אקראים התחלתיים עבור המודל ומשם להתחיל לעשות להם אופטימיזציה. אמנים, פעמים רבות שימוש במשקלים מוכנים שהתקבלו עבור מודל אחר (שעוצב למשימה אחרת) מהווים אתחול טוב יותר.

בהתאם להמלצת המאמר של ViTPose, אנו מתחילה את backbone של המודל עם המשקלים של מודל [Masked Autoencoders](#) (MAE).

מודל זה עוצב עבור משימה של שלמת תמונות. כלומר, הוא מקבל תמונה שמחקו חלק منها ומחזיר את התמונה השלמה. משימה זו היא דוגמה ל- "משימה מזויפת".

משימות מזויפות פופולריות בהקשר של עיבוד שפה טבעית. למשל, מאמנים מודל לקבל משפטים עם מילים חסרות ולהשלים אותם, כך שהמטרה היא לא לקבל כליה להשלמת מילים חסרות במשפטים, אלא לקבל מעין

"творר לוואי" של אימון המודל שהוא משקלים שמתמחים בעיבוד מילים והמטרה לווקטורים בצורה שמקודדת את המשמעות שלהם.

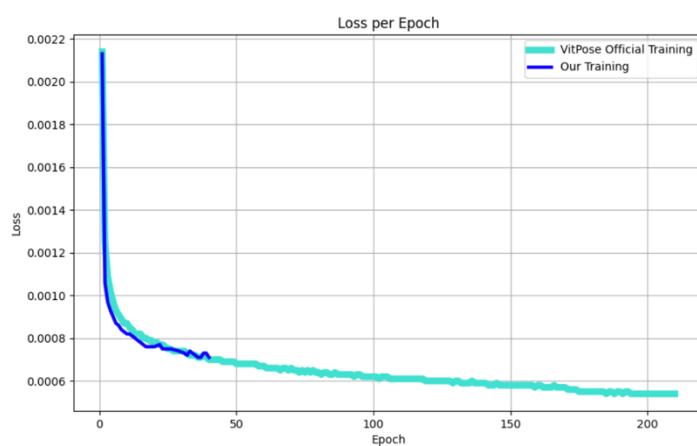
המטרה של MAE היא דומה. הוא למד משימה של השלמת תמונות וBUMIN תופעת לוואי המשקלים שהתקבלו בתהליך האימון שלו מתמחים בהוצאת מידע משמעותי מתמונות, והם פעמים רבות אידיאליים ומהווים נקודת תחילת מצוינת עבור משימות של עיבוד תמונה, כמו במקרה שלנו.

הרצת האימון

הרצת האימון הتبוצעה באמצעות מנווי Google Colab המאפשר גישה למשתמשים חזקים במיוחד. את הרציה הרצינו על גבי מעבד GPU יחיד מסוג V100 (לצערנו לא יכולנו להשתמש במספר מעבדים כחלק מהשירותים של גугл, ולכן לא יכולנו לקבל את תהליך האימון). חלק מוגבלות השירותים של Google Colab, לא יכולנו להריץ את כל האימון ברכז ולכן הרצינו בפעימות של 10 epochs בכל פעם, כאשר בכל פעם שמרנו את המצב הנוכחי של המודל ולאחר מכן המשכנו מאיפה שעצרנו.

לצערנו האימון גבה עליות כספיות גבוהות מאוד ולכן נאלצנו לעזרה באמצעות האימון ולא הייתה לנו את ההזדמנויות לנסות לשפר את המודל ע"י הוספת דאטה או ביצוע שינויים בהיפר-פרמטרים. לשם חתנו ניתן למצוא בדף GitHub הרשמי של ViTPose, אשר נמצא [בקישור הבא](#), משקלים לגרסאות השונות של המודל (השוני בגרסאות small, base, large, huge) נבדל על ידי היפר פרמטרים שונים כגון מספר הראשים בשכבה היבטי בגרף התקנסות של loss כפונקציה של מספר האיפוקים. צירפנו בגרף את התקנסות המודל לפי קבצי `log` שפורסמו בדף GitHub של ViTPose, לצד התקנסות המודל בניסיון האימון שלנו.

אימון המודל שלהם הتبוצע באמצעות 8 מעבדי GPU חזקים. מה שאפשר להם בעצם להגדיל את גודל הbatch פי שמונה ולחلك אותו בין המעבדים על מנת לבצע צעדים מדויקים יותר ולהקטין את מספר האיטרציות. ניתן להביט בגרף התקנסות של loss כפונקציה של מספר האיפוקים. צירפנו בגרף את התקנסות המודל לפי קבצי `log` שפורסמו בדף GitHub של ViTPose, לצד התקנסות המודל בניסיון האימון שלנו.



פונקציית loss כתלות במספר האיפוקים לאורך האימון.

ניתן לראות שניסיונו האימוני שלנו אכן נראה דומה לאופן ההתקנסות של האימון הרשמי של המודל.

ניתן להוריד את המודל שאימנו ולראות את ביצועיו (אך למרות שאימנו אותו למשך 40 איפוקים בלבד). שמו לב של מרבית ההפתקה המודל הצליח לזהות בקלות תווי פנים, כתפיים ומותניים, אך עדין היה לו קשה לזהות ידיים ורגליים עבורי פוזיציות מורכבות. לכן, בחרנו להשתמש במשקלים שפורסםם בדף GitHub הרשמי שבו הם הגיעו לתוצאות טובות יותר.

שלב רביעי – ניתוח ביצועי המודל

לאחר שסימנו עם תהליך האימון בעזרת סט נתונים המבחן ונתוני הולידציה, עליינו לחת את המודל ולהעריך את ביצועיו. נעשה זאת בעזרת סט נתונים המבחן אליו המודל אינו נחשף בזמן האימון. שמירה של נתונים המבחן לשלב ניתוח הביצועים יראה לנו כיצד המודל הצליח להכליל וכייזד הוא מתקף עבור תמונות שלא ראה.

סט נתונים המבחן מכיל תמונות רבות של בני אדם והטיוג (key-points) הנכון עבורן. אנו ניקח את התמונות הללו ונעביר אותם דרך המודל שאימנו, לקבלת key-points שהמודל מנבא. בעת, עליינו להשוות באופן אמפירי בין הטיוגים הנכונים לטיוגים שהמודל ניבא עבור התמונות האלה. נסביר על אופן ההשוואה⁶.

חישוב Confusion Matrix

עבור כל key-point, נחליט על כלל החלטה קשיה לקביעה האם הניבוי היה "נכון" או "לא נכון". לצורך העניין, אם key-points שהמודל ניבא עבור אדם כלשהו הוא מספיק קרובות key-points האמיטיות, נקבע שהמודל "צדק". אחרת, נקבע שהמודל "טעה". את אופן חישוב המרחק בין נקודות, וכן ערך הסף עבור המרחק (נקרא *threshold*), נבחר בהמשך.

כלומר, עבור כל תמונה של אדם בסט נתונים המבחן – נקבע כי המודל "צדק" אם הנקודות שהוא ניבח עבורו האדם מספיק קרובות לנקודות הנכונות. אם זהו המקרה, אז נקרא לתמונה זו *True Positive*.

בדומה, נקבע כי המודל "טעה" אם הנקודות שהוא ניבח עבור האדם אין מספיק קרובות לנקודות הנכונות. אם זהו המקרה, אז נקרא לתמונה זו *False Negative* וגם *False Positive*.

נשים לב שההגדירות של *False Negative* ו- *False Positive*, אמורים קיימים שני מצבים בהן ההגדרות שלחן שונות אחת מהשניה:

- אם המודל ניבח נקודה למרות שאין לה תיוג (למשל אם היד של האדם לא נמצאת בתמונה, אז אין תיוג עבור היד. אם המודל בכל זאת ניבח נקודה "מיותרת" עבור היד, כלומר חשב שהיא כן נמצאת בתמונה, נקרא False Positive).

- אם המודל לא ניבח נקודה למרות שיש לה תיוג (למשל אם המודל החליט שהיד של האדם לא נמצאת בתמונה כלומר הוא לא הוציא נקודה עבור היד, למרות שהיא כן נמצאת בתמונה והיא מותאמת, נקרא False Negative).

⁶ Mean Average Precision (mAP) Explained: Everything You Need to Know. Deval Shah, 2022.

ניתן גם להגדיר *True Negative* למקרים שלא נעשו בטעות, להיות מקרה שבו המודל לא ניבח נקודת, והיא באמת אינה מתויה (למשל אם היד של האדם לא נמצאת בתמונה, אז אין תיוג עבור היד. אם המודל הבין זאת ובאמת לא ניבח נקודת עבור היד, נקרא למצב זה *True Negative*).

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

מטריצת הבלבול (confusion matrix)

כאשר הגדרנו את TP, FP, FN, נחשב את הכמות של כל אחד מהמצבים הללו בנתוני המבחן שלנו (נעבור על כל אדם והשווים ניבח עבורו לצד התיוג שלו, ונקבע האם הוא TP, FP או FN). ניתן להציג את ערכיהם אלו במטריצה הנראית מטריצת הבלבול. באופן כללי אנו נשאף שכמות TN והTP יהיה גבוהה, בעוד שכמות FP והFN יהיה נמוכה.

חישוב Recall ו-Precision

לאחר שהчисלנו את ערכי TP, FP וFN, נעבור לחישוב ערכיהם הנקראים *Precision* ו-*Recall*. המוגדרים להיוות,

$$Recall = \frac{TP}{TP + FN}, \quad Precision = \frac{TP}{TP + FP}$$

כלומר, החישוב את האחו מתוך נקודות שהמודל הוציא שהן נכונות. ואילו *Recall* מחשב את האחו מתוך נקודות האמיתיות שאכן זהה בצורה נכונה על ידי המודל.

חישוב AP – Average Precision

נשים לב שהחישוב של *Recall* וה*Precision* מושפע באופן ישיר מבחןית ערך הסף של המרחק (threshold). לכל keypoints של אדם שהמודל ניבא נחשב את המרחק שלהם מהנקודות האמיתיות. אם מרחק זה גדול מהreshold נקבע כי המודל טעה, ואם המרחק קטן נקבע כי המודל צדק. בחירת ערכים שונים עבור threshold ייתן לנו ערכים שונים ל*Recall* ול*Precision*.

לצורך חישוב ה-*Average Precision*, נרצה למדוד את הקשר בין ערכי *Recall* וה*Precision*. שכן, נשים לב שלעיתים קרובות קיימים trade-off בין ערך *Recall* לבין ערך *Precision* גובה.

- מודל בעל *Precision* גבוהה, אך *Recall* נמוך : מאוד יזהר לנבأ נקודות, וינבא מספר קטן של נקודות (רק כאשר הוא מאוד בטוח) ולכן נקודות שהוא מנבא נכונות. אולם, הוא עלול לפספס הרבה נקודות.
- מודל בעל *Precision* נמוך, אך *Recall* גבוהה : מאוד פזיז לנבא נקודות, וינבא מספר גדול של נקודות (גם כאשר הוא אינו בטוח) ולכן הוא כמעט כמעט ולא יפספס נקודות, אך הוא עלול לנבא נקודות רבות שאינן נכונות.

לכן, השאיפה שלנו היא שלמודל יהיו ערכי Precision גבוהים ככל הנתון גם עבור ערכי Recall שונים. ככלומר אם נגדיר פונקציה $Precision(recall)$ המתארת את הקשר בין Recall לprecision, נקבע כי המודל בעל ביצועים טובים אם הפונקציה הזאת מוציאה ערכים טובים עבור טווח רחב של ערכי recall. ניתן למדוד אמperfirtiy כמה המודל טוב בעזרת חישוב השטח שמתוחת לפונקציה, וזה בדיקת הגדרת $Average Precision$.

$$AP := \int_0^1 Precision(r) dr$$

נראה שאכן אם הפונקציה הזו מביאה ערכים גבוהים עבור טווח רחב של ערכי Recall, אז השטח שמתוחתיה יהיה גבוה.

בפועל, הקשר בין precision לrecall הוא איינו ישיר (כלומר לא מחשבים ישירות את precision בעזרת recall), אלא שניהם מושפעים מכמויות TP, FP, FN שהמודל הפיק עבור נתוני המבחן, וערכים אלו מושפעים מהreshold שבחרנו עבור קביעת סף המרחק.

לכן, נבחר ערכי threshold שונים ועבור כל ערך נחשב את precision ואת recall. נקרב את הפונקציה $Precision(recall)$ ע"י חיבור הנקודות $(recall, precision)$ שהיחסנו עבור כל ערך threshold.

כלומר, הAP בעצם מודד לנו כמה המודל שלנו שומר על ביצועים טובים כאשר אנו בוחרים ערכי threshold שונים (כਮובן שככל שערך זה גדול כך המודל יהיה צודק יותר פעמים שכן אנו יותר "סלחניים" לגבי המרחק בין הנקודה שהמודל ניבה לתיאוג האמתי).

אופן חישוב המרחק בין הנקודות – OKS

נשאר לנו לכנות עניין אחרון והוא אופן חישוב המרחק בין הנקודות שניבא המודל לבין הנקודות האמיתיות. ניתן כמובן להשתמש בסכום המרחקים האוקלידיים בין כל ה`key-points` של האדם. אולם, חישוב זה יעשה לפי מספר פיקסלים, ושיטה זו אינה לוקחת בחשבון את רזולוציית התמונה/גודל הבנאים (למשל אם התמונה ברזולוציה גבוהה, סטייה של 50 פיקסלים מכנה מרחק קטן מאוד). לכן, משתמשים בהגדרה הבאה,

$$OKS = \frac{\sum_j \exp\left(-\frac{d_j^2}{2s^2k_j^2}\right) \cdot \delta(v_j > 0)}{\sum_j \delta(v_j > 0)}$$

כאשר,

- d_j על כל ה`key-points` של האדם (הdataset שלנו מתייחס 17 `key-points` לכל אדם).
- v_j זה המרחק האוקלידי בין הנקודה שהמודל ניבא עבור ה`key-point` ה j , לבין התיאוג האמתי שיש עבור ה`key-point` ה j .
- זה גודל האובייקט, במקרה שלנו זה גודל ה`bbox` (bounding box) בקיצור של האדם. ככל שרזולוציית התמונה גבוהה יותר או שהאדם קרוב יותר למכלמה, כך גודל ה`bbox` גדול יותר.
- k_j זה קבוע שנבחר עבור ה`key-point` ה j . נרצה לבחור קבועים נמוכים עבור נקודות שנותות להישאר באותו המיקום (למשל הארץ), וערך גבוהים עבור נקודות שהמיקום שלהם נוטה להשתנות מהתמונה לתמונה (למשל הידיים או הרכבים). הרעיון הוא שבעור נקודות שנותות להישאר באותו מקום נרצה לתת יותר חשיבות כאשר המודל בכל זאת לא דוק בם.
- $(0 > v_j \delta)$ מסמן אם ה`key-point` ה j נמצא בתמונה ($= 1$) או לא ($= 0$), כלומר אנו בעצם מעריכים מנקודות שהן מוסתרות/חתומות. אנו בעצם מעריכים ממוצע של כל $\exp(-d_j^2/2s^2k_j^2)$ עבור כל הנקודות j שאינן מוסתרות בתמונה.

ניתוח ביצועי המודל ViTPose

חישוב ערך AP עבור מודל `ViTPose` אשר אומן למשך 210 איפוקים, הניב ערך **0.882** בהתחשב בעובדה שערך AP נע בין 0 ל-1, אנו מאמינים ש0.882 הוא מיטף.

דוגמאות חיזוי המודל

ניתן להתרשם מדוגמאות של ביצועי המודל על גבי תמונות שונות. נראה כיצד המודל מצליח להתמודד גם עם תמונות בעלות אנשים רבים וגם במצבים של פיזייקה מורכבת (כמו יוגה או סקי).

