

# דו"ח סופי מיני פרויקט – 3D Classification

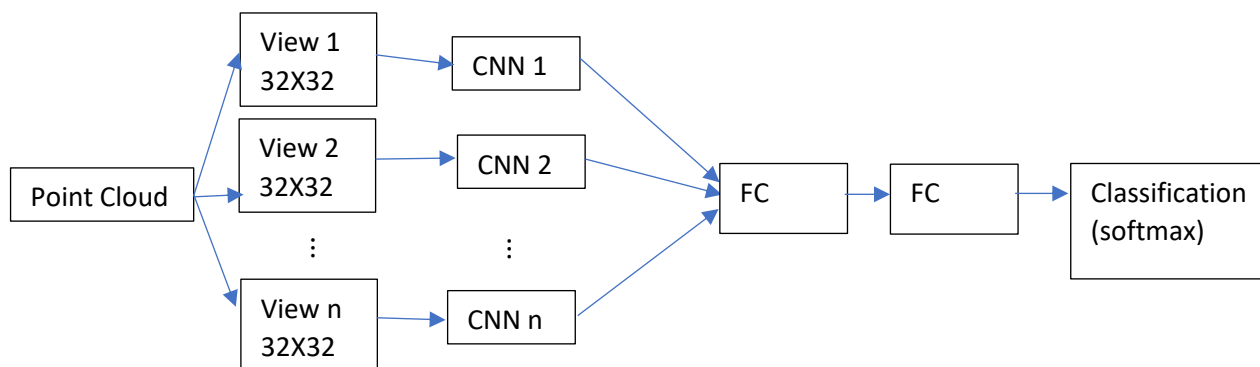
מגיש: יהונתן כהן 203372032

## Design:

התכנון והמימוש נעשה בהשראת המאמר-

H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller - Multi-view convolutional neural networks for 3d shape recognition, In Proc. ICCV, 2015

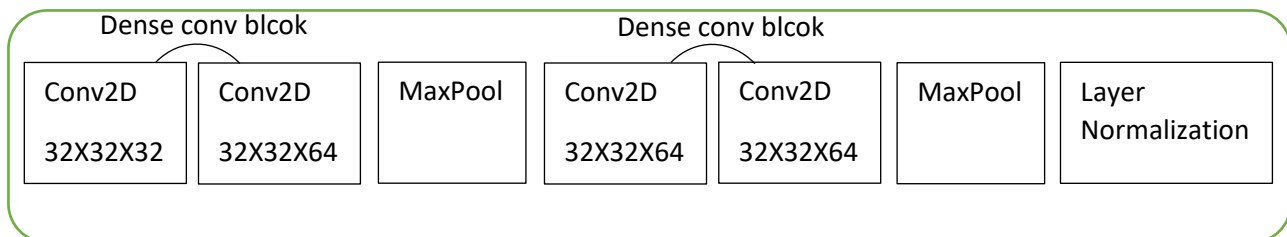
להלן הארכיטקטורה הסופית שבנית:



\* CNN = Convolutional Neuron Network

\* FC = Fully Connected

CNN i:



בקצרה:

הופכים סט של דגימות מפני השטח של אובייקט כלשהו ל- $m$  תמונות, המתארות הסתכלות על הנפח המשווער של האובייקט מ- $m$  זוויות שונות (views), כאשר  $m$  הוא היפר-פרמטר של הארכיטקטורה.

התמונות הללו הן קלט ל  $m$  רשתות קונבולוציה זהות, כל אחת מספקת בסופה מערך של 128 פיצ'רים, המתארים כל view באופן ייחודי.  $128 \times m$  הפיצ'רים ממשיכים לשתי שכבות fully connected ולאחר מכן אל 10 נירונים בשכבת הפלט, שם מתבצע הסיווג: הקטגוריה בעלת הציון הקרוב ביותר ל-1 אחרי הפעלת soft-max.

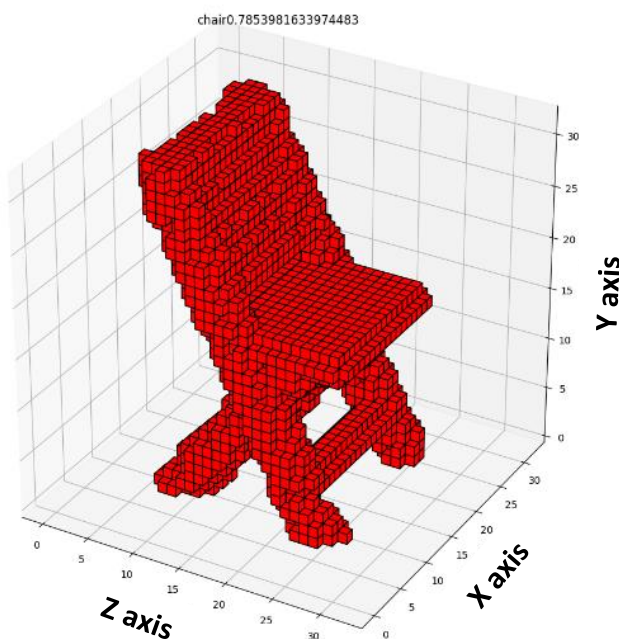
בעזרת ארכיטקטורה זו הצלחתי להגיע לרמת דיוק של 95% על סט הולידציה.

בשונה מהמאמר המקורי, כל view עובר נירמול, כך שכל הזוויות תורמות לתוצאה הסופית במידה שווה. זאת בניגוד למה שהוצע במאמר - max-pooling על פני כל ה views, דבר שמשמר את ה views שהגיבו 'חזק יותר', על גבי view שהגיבו 'פחות חזק'.

## הכנת הקלט:

מקבלים כקלט סט של 4000 דגימות לייזר שנדגמו מפני השטח של אובייקט תלת מימדי. כל דגימה היא נקודה תלת מימדית  $p = (x, y, z) \in R^3$ , כאשר  $0 \leq x, y, z \leq 1$ ,  $p$  הוא הגובה,  $z$  הוא העומק.

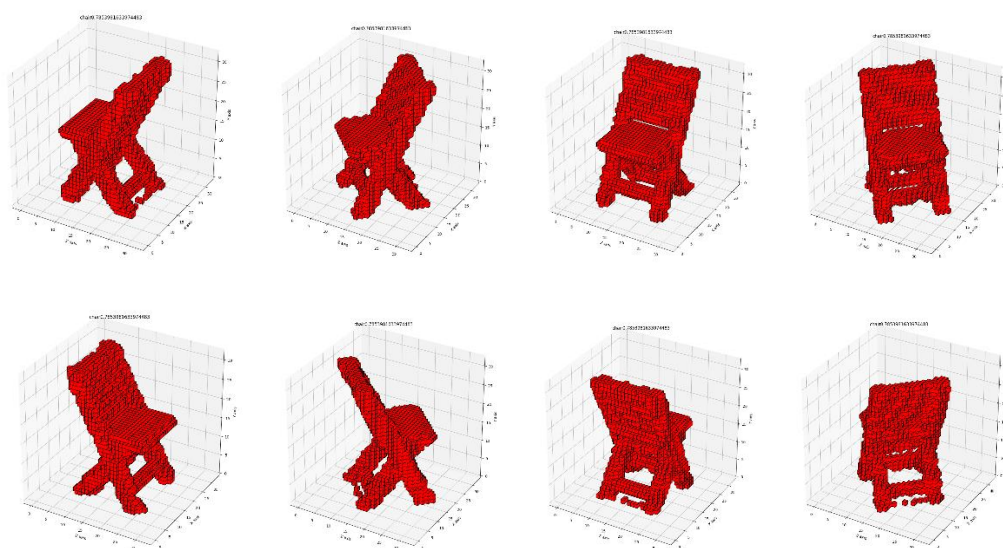
כעת נעבור ממרחב רציף למרחב בדיד -  $(x, y, z) \in R^3 \leftarrow (\hat{x}, \hat{y}, \hat{z}) \in \{0, \dots, 31\}^3$ . אנחנו רוצים להתייחס לכל אובייקט כנפח חלקי ובדיד מתוך קוביית היחידה. ישנן  $32^3$  'תתי-קוביות', כל קובייה בעלת צלע של  $\frac{1}{32}$  המרכיבות את קוביית היחידה. כל קובייה יכולה לקבל אחד משני ערכים: אם הייתה נקודה  $p$  המוכלת בנפח של הקובייה, יתקבל הערך 1, אחרת יתקבל הערך 0. להלן דוגמה של אובייקט (כיסא) עליו בוצע התהליך:



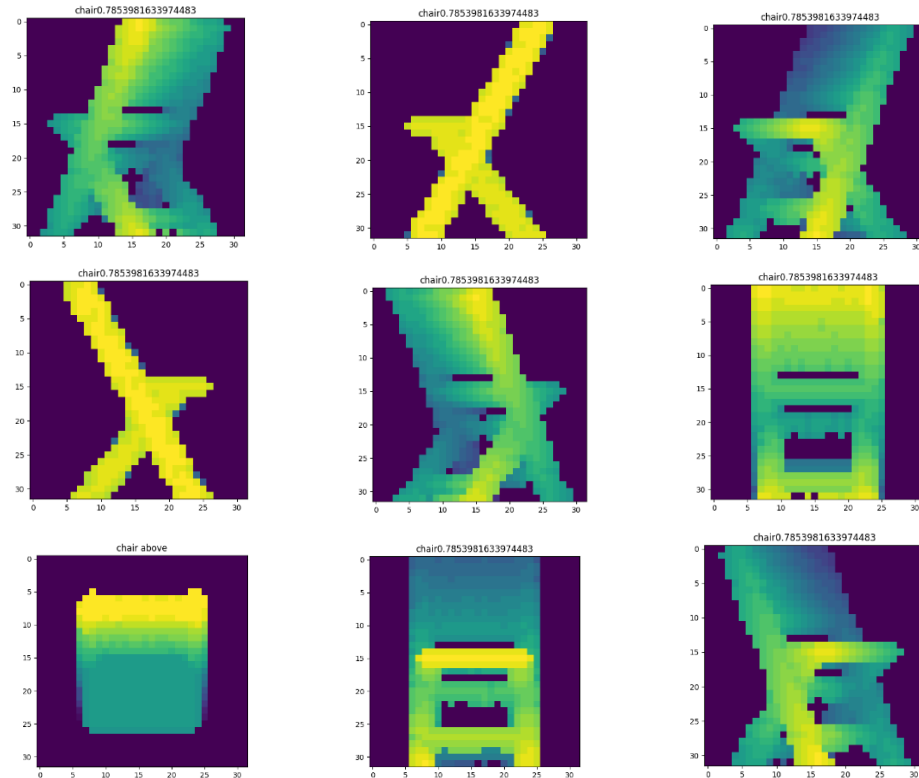
נרצה כעת לסובב את הנקודות (המקוריות) סביב ציר  $y$ ,  $n - 1$  פעמים, בכל פעם סיבוב של  $\frac{360}{n-1}$  מעלות (או לחלופין  $\frac{2\pi}{n-1}$  רדיאנים), כך שעבור צופה הנמצא על ציר  $z$ , זווית ההסתכלות על ענן הנקודות משתנה. קיבלנו  $n - 1$  זוויות הסתכלות שונות על האובייקט. נוסיף זווית הסתכלות נוספת מלמעלה (ציר  $y$ ) על חלקו העליון של האובייקט.

כעת יש בידינו  $n$  זוויות הסתכלות (views) שונות על האובייקט. נמיר כל view (מלבד ה view העליון) לנפח בדיד כפי שתיארנו, ונקבל  $n - 1$  נפחים. להלן הנפח של אותו האובייקט מסובב ע"פ  $n - 1$  הזוויות:

(לצורך המחשה בדוגמאות הנ"ל אשתמש ב  $n = 9$  כלומר 8 זוויות, כל פעם סיבוב של  $45^\circ$  או  $\frac{\pi}{4}$  רדיאנים)



כעת נבצע 'הטלה' של כל נפח על מישור "ההסתכלות של הצופה", נגדיר אותו להיות המישור  $z = 1$ . מתוך  $n - 1$  נפחים, כעת נקבל  $n - 1$  תמונות, הטלה 3 מימדים ל 2 מימדים, על ריבוע היחידה. הריבוע מורכב מ  $32^2$  ריבועים, כל ריבוע בעל צלע של  $\frac{1}{32}$ , המרכיבים את ריבוע היחידה. בשונה מהנפח, בו כל קובייה הייתה בעלת הערך 1 או 0, כאן כל ריבוע בקואורדינטה  $x, y$  ( $0 \leq x, y \leq 31$ ) יכול להכיל את אחד מהערכים  $\{0, \dots, 31\}$ , ע"פ המרחק בין המישור  $z = 1$  לבין קוביית הנפח הקרובה ביותר למישור  $z = 1$  עם אותן קואורדינטות  $x, y$ . אם הקובייה הקרובה ביותר הייתה קרובה למישור  $z = 1$ , יתקבל ערך גבוה (צבע צהוב בתמונות). אם הקובייה הקרובה ביותר הייתה רחוקה מהמישור  $z = 1$ , יתקבל ערך נמוך (צבע כחול בתמונות). אם לא קיימת קוביית נפח עם קואורדינטות  $x, y$  הערך בריבוע  $x, y$  יהיה 0, הערך המינימלי. בנוסף ל  $n - 1$  ההטלות, נבצע הטלה נוספת (באותה דרך) של הנפח הראשון על המישור  $z = 1$ , כלומר מבט מלמעלה. בסה"כ יתקבלו  $n$  הטלות (views). להלן ההטלות המתאימות להמחשה (שוב עבור  $n = 9$ ):



\*ההטלה בפינה השמאלית התחתונה היא מבט 'מלמעלה'.

נשים לב שבתהליך זה איבדנו מידע, שכן לא ניתן (או קשה) כעת לשחזר את הנפח המקורי. למרות זאת, בזכות התפתחות מחקרית בתחום של עיבוד תמונה דו מימדית, יש בידינו כלים היכולים להפיק הרבה מידע מסדרת התמונות האלו. לדעתי ההשראה לכך נובעת ממערכת הראייה האנושית – לבני אדם יש יכולת טובה מאוד בזיהוי אובייקטים, וזאת על סמך קלט של 2 תמונות דו מימדיות בלבד, ללא מישוש או קלט תלת מימדי כלשהו.

## תיאור הרשת:

### רשת הקונבולוציה:

רשת הקונבולוציה (לצורך נוחות נרשום קונב') מקבלת כקלט תמונה בגודל  $32 \times 32$ , ומוציאה כפלט מערך חד מימדי של 128 features.

הבסיס של רשת קונב' היא שכבת קונב': בשכבה זו מתבצעות מספר רב של קונב' דו מימדיות, עם פילטרים שמשתנים ונלמדים לאורך ריצת הרשת. לאחר מספר ריצות מסוים, חלק מהפילטרים 'לומדים לזהות' (כלומר, יתנו תגובה חזקה יותר) קווים, פינות, שיפועי גווים ועוד. קונב' עם  $m$  פילטרים, מייצרת תמונה חדשה בעומק  $m$  ('ערוצי צבע') שתינתן כקלט לשכבת הקונב' הבאה (ברשת זו יש שתי שכבות קונב' המופיעות אחת אחרי השניה ברצף).

בארכיטקטורה שלי, בהשראת (DenseNet) (DenseNet) (G. Huang, Z. Liu, K. Q. Weinberger, and L. Maaten. Densely connected convolutional networks. In CVPR, 2017), כל זוג שכבות קונב' הם 'convolution dense block', כלומר הקלט של השכבה הראשונה מצורף לקלט של השכבה שאחריה. לדוגמה, אם צורת הקלט של השכבה הראשונה היא  $(32, 32, m_1)$ , והפלט של השכבה הראשונה הוא  $(32, 32, m_2)$ , אז הקלט של השכבה השנייה יהיה  $(32, 32, m_1 + m_2)$ . כלומר, הקלט מתווסף ל'עומק' של השכבה הבאה.

בקצרה, ארכיטקטורה זו מפחיתה overfitting, ומעבירה את המידע אל השכבות המאוחרות בצורה יעילה יותר, מה שמאפשר לרשת להיות קטנה יותר.

לאחר שתי שכבות קונב', יש שכבת Max Pooling עם 2 מטרות עיקריות – הפחתת רעש, והורדת מימד. כל ריבוע של 4 פיקסלים יהפוך לפיקסל יחיד, בעל הערך המקסימלי מבין הארבעה. אחרי שכבה זו ניתן לראות את feature מסוים הופיע באופן חזק יחסית באיזור כלשהו של התמונה. התמונה המתקבלת היא קטנה פי 4 (חצי אורך צלע) מהתמונה הקודמת.

לאחר שכבה זו מתבצע drop-out בהסתברות של 25%, כלומר, במהלך האימון כל נירון בשכבה זו יכול לקבל באופן אקראי את המשקל 0 במקום את המשקל שנלמד במהלך האימון. שכבה זו משפרת את יכולת ההכללה של הרשת- היות וכל נירון בממוצע רק ב  $\frac{3}{4}$  מהאימון, לא יקרה מצב בו הסיווג תלוי לחלוטין במשקל של נירון בודד. באופן זה, חלקים רחבים יותר של הרשת משתנים ולומדים לאורך האימון.

מבצעים את 3 השלבים האלו (conv-dense-block, max pool, drop out) פעמיים ברצף.

לאחר מכן 'משטחים' את הפיצ'רים למערך חד-מימדי ומקטינים את המימד ל 128 פיצ'רים ע"י שכבת fully connected. לאחר מכן שכבת layer-normalization, שדואגת לכך שהממוצע של 128 הפיצ'רים יהיה 0, וסטיית תקן 1 (בשונה מ- batch-normalization, שמבטיחה ממוצע 0 וסטיית תקן 1 על פני batch שלם, זאת על מנת לשפר את ההתכנסות). המטרה ב-layer-normalization היא לתת משקל דומה לכל view בשלב זה של הארכיטקטורה, ולעודד את הרשת להשתמש בפיצ'רים של כל הזוויות. לבסוף שכבת drop-out נוספת בסיכוי של 0.25.

### המשך הרשת ושילוב ה-views:

משרשרים את הפלטים של כל רשתות הקונב' למערך חד מימדי באורך  $128 \times n$  עם הפיצ'רים מכל הזוויות. מערך הפיצ'רים עובר לשתי שכבות fully connected עם 256 נירונים, עם שכבת drop out בהסתברות של 0.25 ביניהן. לבסוף יש את שכבת הפלט עם 10 נירונים (כמספר המחלקות) עליה מופעלת פונקציית האקטיבציה softmax, על מנת לקבל וקטור תחזית מנורמל (סכום הרכיבים בווקטור התחזית הוא 1). הערך הקרוב ביותר ל-1 יהיה הערך שיבחר לסיווג.

פונקציית האקטיבציה המשמשת את כל הרשת היא RELU.

## מימוש:

מימוש רשת הנורונים נעשה באמצעות הספריה Keras, בשימוש backend של הספריה Tensor Flow. זוהי אחת הספריות הנפוצות למודלים מהסוג הזה, ומקובלת בתעשייה ובאקדמיה. נעשה שימוש בשכבות מסוג: Dense, Dropout, Flatten, Conv2D, MaxPooling2D, LayerNormalization, Concatenate

## הכנת הקלט:

בדו"ח אתייחס באופן קבוע לצירים בתור: x רוחב, y גובה, z עומק, מתוך נקודות מבט של צופה הנמצא על המישור  $z = 1$ , כפי שמצוין בתרשימים. בנתונים עצמם, כל נקודה היא וקטור תלת מימדי שמסודר באופן הבא:  $(y, z, x)$ , כלומר קואורדינטת y מופיעה ראשונה.

## סיבוב הנקודות:

הפונקציה rotate מקבלת סט נקודות וזווית theta ברדיאנים, ומחזירה את סט הנקודות לאחר שהתבצע סיבוב של theta רדיאנים ביחס לציר y, כאשר ציר הסיבוב הוא מרכז המסה של הנקודות. ביתר פירוט:

- ראשית ממרכזים את הנקודות כך שמרכז המסה יהיה בראשית הצירים- מחשבים את מרכז המסה של הנקודות ואז מחסירים את מרכז המסה מכל הנקודות.
- מסובבים סביב ציר y,  $\theta$  רדיאנים, כאשר  $\theta = \frac{2\pi}{n-1}$ . יש  $n - 1$  תמונות, כך יוצא ש:  
$$2\pi \cdot (n - 1) = \frac{2\pi}{n-1}$$
  
כלומר סיבוב שלם. על מנת לסובב, כופלים (כפל מטריצוני) כל נקודה במטריצת הסיבוב  $\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$ .
- מחזירים את הנקודות לחלק החיובי של הצירים, על ידי החסרה של הערך השלילי המינימלי (חיסור של שלילי נותן מספר חיובי) בתוספת 0.0001, כדי שהערכים יהיו חיוביים ממש (גדולים מאפס).
- מנרמלים לקוביית היחידה – אם לאחר הסיבוב קיימת נקודה עם ערך z או x גדול מ1, מקטינים את הנפח כולו כך שיהיה מוכל בקוביית היחידה (תוך שמירה על הפרופורציה בין הצירים).
- בשלב זה הנפח נמצא צמוד לראשית הצירים, נרצה להזיז אותו למרכז קוביית היחידה. נעשה זאת ע"י חישוב רוחב השוליים: חישוב המרחקים המינימליים למישור  $x = 1$  ולמישור  $z = 1$ , והזזה לחצי מהמרחקים האלו.
- כעת קיבלנו את הנפח המסובב כך שהוא נמצא במרכז קוביית היחידה ואינו חורג ממנה.

## מעבר מנקודות לנפח בדיד:

הפונקציה pointcloud2volume מקבלת סט נקודות, לוקחת כל נקודה ומסמנת היכן הנקודה נמצאת בתוך קוביית היחידה, בחלוקה ל  $32^3$  קוביות (כל קוביה עם אורך צלע של  $\frac{1}{32}$ ). פעולה זו מתבצעת בעזרת הפונקציה digitize של numpy, המקבלת מספר בין 0 ל-1 ומחזירה את האינדקס של התא (bin) בו המספר נמצא, כאשר יש 32 bins:  $\left( \left(0, \frac{1}{32}\right), \left(\frac{1}{32}, \frac{2}{32}\right), \dots, \left(\frac{31}{32}, 1\right) \right)$ . נותנים את הערך 1 לקוביה שהתקבלה ול-6 הקוביות השכנות (צפון, דרום, מזרח, מערב, למעלה, למטה). פעולה זו מתבצעת לכל הנקודות, ומתקבל נפח בדיד המתאר את האובייקט.

### 'הטלה' של הנפח לתמונה דו מימדית:

מתבצעת כחלק מהפונקציה `get_photos_labels` (תפורט בהמשך). ראשית הערך של כל קוביית נפח מוכפל בערך של קואורדינטת ה  $z$  שלו, כך שקוביות עם ערכי 0 יישארו עם ערך 0, וקוביות עם ערך 1 יכילו את ערך קואורדינטת ה  $z$  שלהם. כעת מתבצעת 'הטלה' על המישור  $z = 1$ :

הערך של הפיקסל  $x, y$  ( $0 \leq x, y \leq 31$ ) יהיה הקואורדינטה של הערך המקסימלי (כלומר  $\text{argmax}$ ) מבין 31 הקוביות עם אותן קואורדינטות  $x, y$ , כלומר מבין  $\{(x, y, 0), \dots, (x, y, 31)\}$ .

התוצאה המתקבלת היא שהערך של כל פיקסל בתמונה, מתקבל לפי המרחק של הנפח מהמישור  $z = 1$ : בנקודות בהן הנפח היה קרוב למישור  $z = 1$ , יהיה ערך גבוה, בנקודות בהן הנפח היה רחוק, יהיה ערך נמוך.

### הכנת dataset חדש:

כדי לייעל את הנסיונות בבחינת ארכיטקטורות שונות, נבצע התמרה של כלל ה `point-clouds` לתמונות, ונשמור את התמונות בדיסק. הפונקציה `get_photos_labels` מבצעת את פעולה זו- עוברים על כל אחד מהסטים של הנקודות, מכינים  $n - 1$  תמונות אופקיות (כל אחת בסיבוב של  $\theta$  רדיאנים) ותמונה אחת מלמעלה.  $n$  התמונות נשמרות כאיבר במערך. בסיום, המערך נשמר כקובץ `json`.

לאחר שהפונקציה נקראה בפעם הראשונה, ניתן לטעון את ה `dataset` החדש במקום לבצע את כל ההטלות והסיבובים, שזו פעולה יקרה מבחינת זמן.

במידה ונרצה לעשות שינוי בתמונות, ניתן להעביר את הארגומנט `force_reload=True`, ואז ההטלות והסיבובים יחושבו מחדש (וישמרו בדיסק).

### סידור מחדש של ה dataset למבנה של Keras:

ראשית מנרמלים את התמונות, כל שכל פיקסל יהיה בין 0 ל-1, ומסדרים את התמונות כך שיהיו במבנה המתאים ל Keras - במקום `32X32`, התמונות יהיו במבנה `32X32X1`.

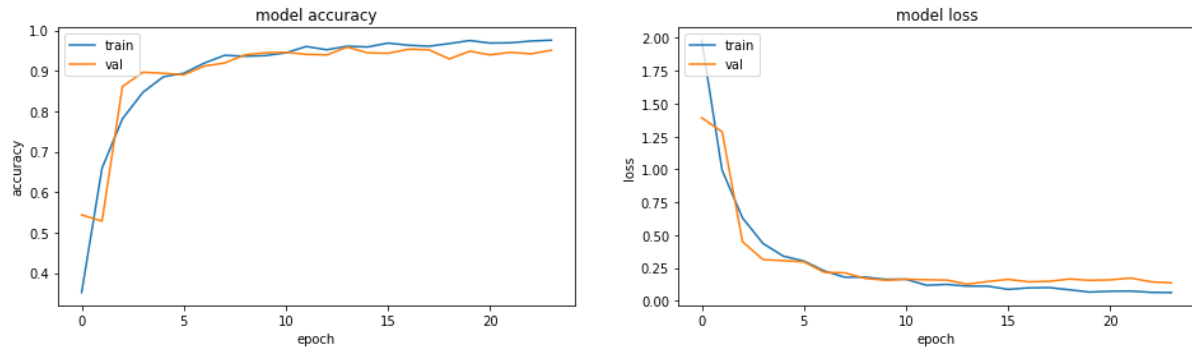
כעת מחלקים את התמונות ל  $n$  רשימות, כך שהאינדקס ה- $i$  בכל רשימה יכיל את אותו האובייקט, מזווית שונה. בצורה כזו ה `label` ה- $i$  מתאים לכל אחת מ- $n$  הזוויות של אותה התמונה ה- $i$ .

## תוצאות:

ה dataset מחולק בחלוקה של 80% מדגם אימון, 20% מדגם ולידציה ובדיקה.

התוצאה הטובה ביותר שקיבלתי היא עבור  $n=7$  (6 זוויות ומבט מלמעלה):

**95% דיוק על מדגם האימון אחרי 18 epochs**, 94% אחרי 11 epochs



מה הכוונה לתוצאה ה'טובה ביותר'? בחרתי למדוד את איכות התוצאה ע"פ:

- אחוז הדיוק
- זמן חישוב ההטלות
- מהירות ההתכנסות

### אחוז הדיוק:

אחוז דיוק גבוה הוא המטרה הראשית שהמודל צריך לענות עליה, לכן זהו הקריטריון החשוב ביותר.

### זמן חישוב ההטלות:

ככל ש  $n$  גדול יותר, כך זמן חישוב ההטלות גדל (לינארי ביחס ל- $n$ ). בשלב האימון אפשר לטעון שזהו דבר שניתן להתעלם ממנו, שכן זהו צעד שנעשה פעם אחת בלבד. לעומת זאת, בכדי להשתמש במודל על מנת לסווג point-cloud חדש כלשהו, יש לבצע את  $n$  ההטלות, מה שהופך את פעולת הסיווג ליקרה יותר ככל ש- $n$  גדל. בימינו הרבה אפליקציות דורשות סיווג ב real-time ולכן נרצה שזו תהיה פעולה מהירה כמה שיותר.

### מהירות ההתכנסות:

ניתן לשפוט את מהירות ההתכנסות של הרשת על סמך מספר ה epochs עד להתכנסות. חשוב לשים לב, שעבור ערכי  $n$  שונים, מספר ה epochs עד להתכנסות יכול להיות זהה, אך כל epoch לוקח יותר זמן (בקירוב לינארי ב- $n$ ).

אשתמש בשני קריטריונים לבחינת מהירות ההתכנסות: מספר ה epoch בו הרשת הגיעה לאחוז דיוק של 94%, ומספר ה epoch בו הרשת הגיעה לאחוז דיוק של 95% (יפורט בהמשך).

שימוש ב-  $n > 7$  הוביל לתוצאות דומות מבחינת רמת הדיוק (95%), אך שני הקריטריונים האחרים נפגעו (זמן חישוב ההטלות וגם מהירות ההתכנסות, תוך התחשבות באורך ה epoch).

שימוש ב-  $n < 7$  הוביל לירידה ברמת הדיוק (פחות מ 95%).

$n = 7$  הוא מספר הזוויות הנמוך ביותר שעדיין מספק רמת דיוק של 95%.

נמשיך לבחון את טיב הסיווג ע"י confusion matrix (ע"פ classification של סט הולידציה):

	bathtub	bed	chair	desk	dresser	monitor	night-stand	sofa	table	toilet
bathtub	19	0	0	0	0	0	0	0	0	0
bed	1	99	2	2	0	0	0	0	0	0
chair	0	0	180	0	0	0	0	0	0	0
desk	0	0	0	28	0	0	1	0	4	0
dresser	0	0	0	1	37	0	7	0	0	0
monitor	0	0	0	0	0	77	1	0	0	0
night-stand	0	1	0	0	5	1	32	0	6	0
sofa	0	1	0	0	0	0	0	139	0	0
table	0	0	0	3	0	0	1	0	73	0
toilet	0	0	1	0	0	0	0	0	0	69

ניתן לראות ש 38% מכלל השגיאות של הרשת נובעות מסיווג הפוך של night-stand לעומת dresser, ו 22% נובעות מסיווג הפוך של table לעומת desk.

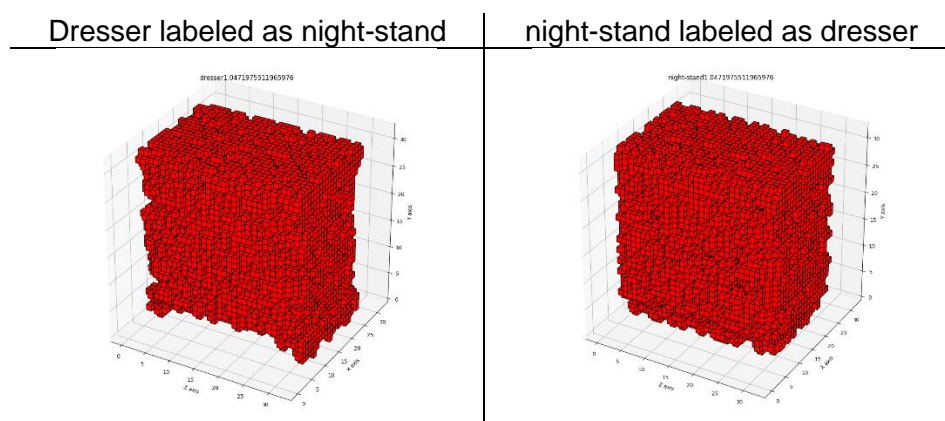
הסבר אפשרי לתופעה זו הוא, שיש מעט מדי אובייקטים בקטגוריות אלו - נבחן את טענה זו אל מול הנתונים:

בטבלה זו מתואר כל סוג אובייקט ומספר המופעים שלו ב dataset השלם:

bathtub	bed	chair	Desk	Dresser	monitor	night-stand	sofa	table	toilet	Average
106	515	889	200	200	465	200	680	392	344	400

אכן ניתן לראות שיש פחות אובייקטים מסוג desk, dresser, night-stand לעומת זאת, יש אפילו פחות (חצי) אובייקטים מסוג bathtub, ולא נראה שהייתה בעיה בסיווג של קטגוריה זו. מכאן ניתן להסיק שמיעוט האובייקטים בקטגוריות הנ"ל כנראה אינו הסיבה לתופעה.

נבחן שני מקרים שסווגו הפוך בקטגוריות dresser ו- night-stand:



כפי שניתן לראות, גם למתבונן אנושי קשה לסווג בין האובייקטים (כנ"ל עבור table לעומת desk).

במציאות ניתן לרוב להבדיל בין night-stand לבין dresser על ידי הגודל האבסולוטי שלהם, אבל ב dataset הנוכחי האובייקטים מתקבלים כאשר הם מנומרים לפי קוביית היחידה, מה שמקשה על הסיווג.

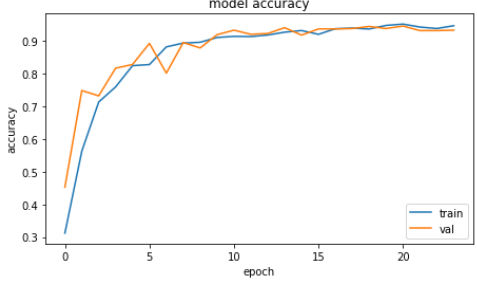
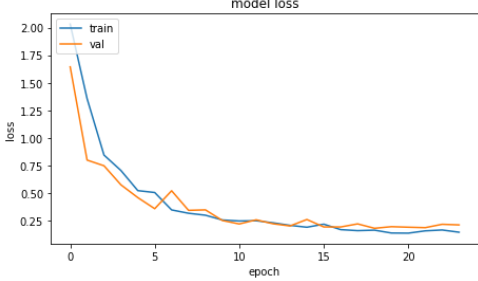
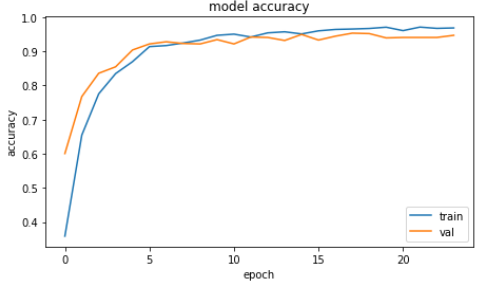
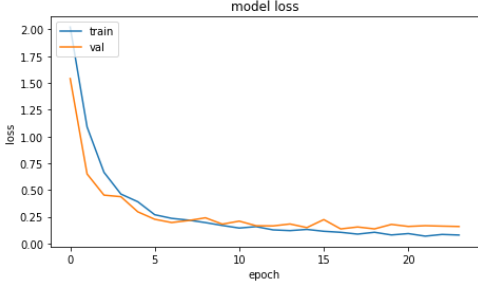
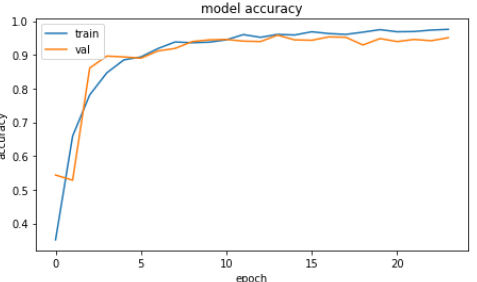
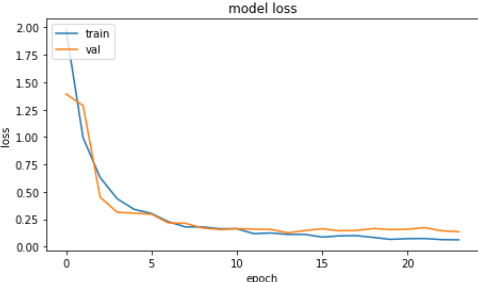
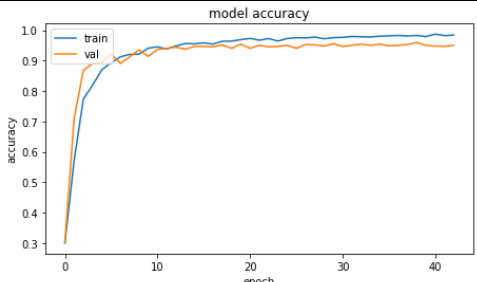
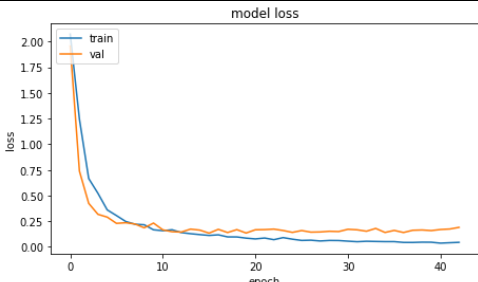
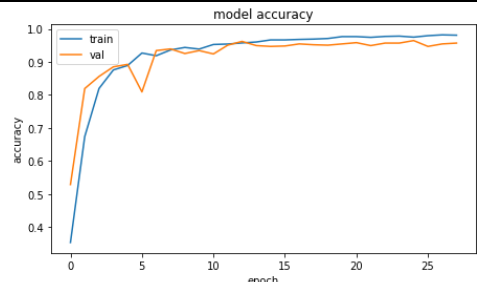
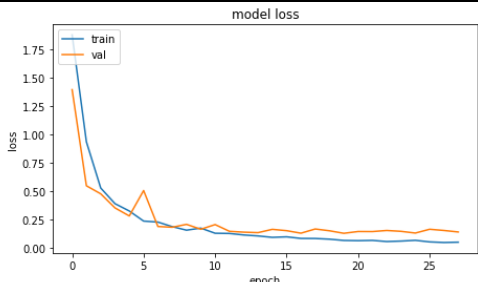
במקרה של desk לעומת table, אין הבדל בגודל האבסולוטי, אבל גם אלו אובייקטים דומים מרחבית- זו כנראה הסיבה שהמודל התקשה לסווג ביניהם, בניגוד ל bathtub, אובייקט עם תכונות מרחביות ייחודיות יחסית.



## ניסויים:

מספר הזוויות (מספר ה views):

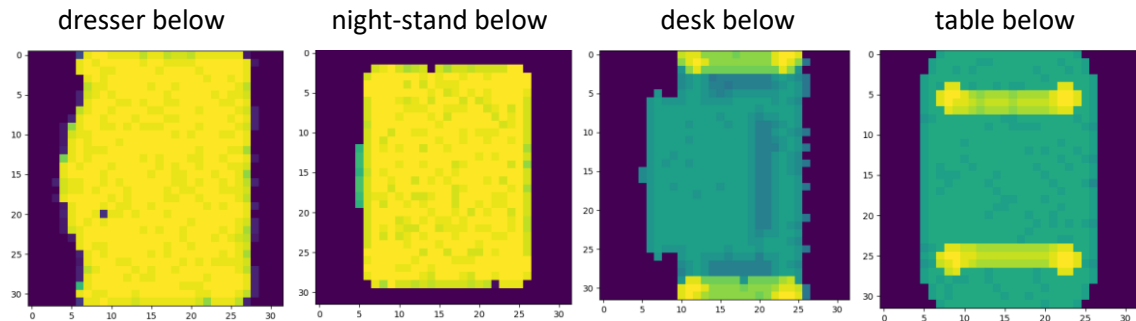
בטבלה מתוארים מספר ניסיונות שביצעתי עם גדלי  $n$  שונים. לכל  $n$  מצוין מספר ה epoch בו הרשת הגיע לרמת דיוק של 94% ו 95% (על סט הולידציה) וגרף התכנסות (אחוז דיוק ו loss):

n	94%	95%		
2	- (max 93%)	-	 	
4	18	-	 	
7	11	18	 	
9	13	21	 	
12	13	16	 	

כפי שניתן לראות, ההתכנסות המהירה ביותר תוך שמירה על אחוז דיוק של 95% התקבלה עבור  $n = 7$ .

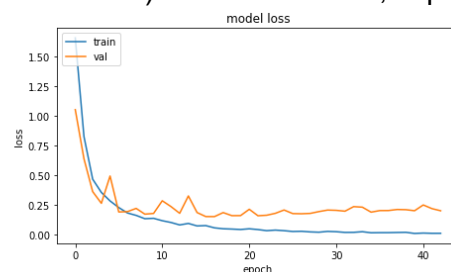
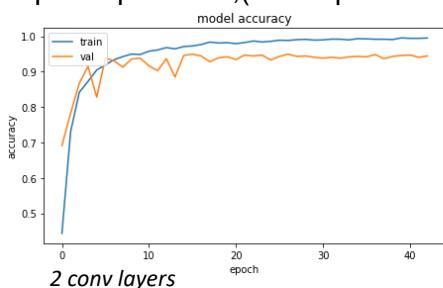
## הוספת view של מבט מלמטה:

החלטתי לבחון האם הוספה של זווית מבט נוספת תשפיע על אחוז הדיוק: בניגוד להשערתי הראשונה, לזווית המבט הנוספת לא הייתה השפעה על אחוז הדיוק. נראה שזווית זו לא מוסיפה הרבה מידע עבור האובייקטים **הקשים לסיווג**, כמו night-stand, dresser, table, desk, ולכן לא הוסיפה שיפור באחוז הדיוק. להלן מספר מבטים מלמטה. הממחישים את חוסר האינפורמטיביות של זווית זו עבור אובייקטים קשים לסיווג:

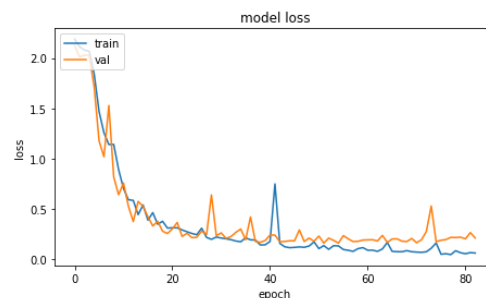
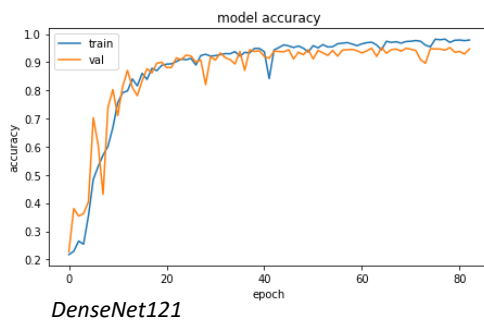


## מספר שכבות הקונבולוציה ("עומק" רשת הקונבולוציה):

רשת 'קטנה' עם 2 שכבות קונב', הייתה מהירה יותר (התכנסות אחרי 7 epochs בלבד), אבל התקבל דיוק מקסימלי של 94%. זאת לעומת רשת עם 4 שכבות קונבולוציה, עם 94% אחרי 11 epochs, ו-95% אחרי 18 epochs.



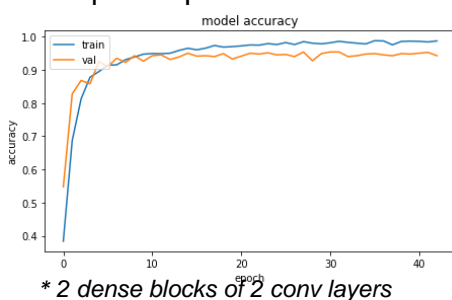
בשימוש ברשת DenseNet 'State of the Art', עם 121 שכבות בסך הכל, מתוכן 58 שכבות קונב', משך



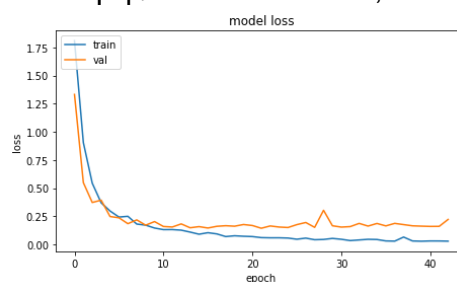
האימון היה ארוך משמעותית, גם מבחינת מספר ה epochs וגם מבחינת משך הזמן של כל epoch. לעומת זאת גם לאחר התכנסות

(50 epochs), רמת הדיוק לא הייתה שונה - 95%.

השיטה הטובה ביותר שמצאתי, הייתה להשתמש בעקרון שמאחורי רשת DenseNet – צירוף של הקלט



\* 2 dense blocks of 2 conv layers



והפלט, אבל בארכיטקטורה קטנה יותר. במקום 4 dense blocks, כשבכל בלוק יש 6, 12, 24, 16 שכבות קונב', השתמשתי בשני בלוקים, כשבכל בלוק שתי

שכבות קונב' בלבד. מצאתי שזוהי ארכיטקטורה שמצד אחד מתכנסת מהר, ומצד שני מגיעה לרמת דיוק גבוהה וכפי שנראה מהגרפים בעלת יכולת הכללה טובה.

## Data augmantation

המטרה ב data augmentation היא להגדיל את כושר ההכללה של הרשת (להקטין overfitting) ע"י טרנספורמציות של הקלט. למשל עבור תמונות- תמונת מראה של חתול היא עדיין תמונה של חתול, וכן תמונה מסובבת, מוקטנת, ועוד.

מהניסיונות שערכתי, ראיתי ש data augmentation של התמונות פוגע בדיוק הסיווג. כיוון שכל view מנומל ואין max pooling ביניהם, יש קורלציה בין המידע שהגיע מכל view, שתורמת לקבלת ההחלטה הסופית. במידה וכל view יעבור טרנספורמציה אקראית, הקורלציה הזו יכולה להיעלם, מה שיכול להשפיע לרעה על הסיווג.

ניתן לבצע ניסוי המשך שיבחן האם וכיצד כדאי לעשות data augmentation של ה point-cloud (ורק לאחר מכן להפעיל את ההטלות של ה views). בניסוי שכזה, לדעתי אין סיבה להפעיל טרנספורמציות סיבוב על הנקודות, שכן זהו בדיוק התהליך שנעשה ע"י ההטלות מזוויות שונות. ה dataset נוצר כך שהמידע מנומל (קוביית היחידה) ומיושר (ציר y מצביע למעלה), לכן לדעתי אין מקום לטרנספורמציות 'sheer' (מתיחה לא אחידה) דוגמאות לטרנספורמציות מתאימות יכולות להיות:

- א. מתיחה או כיווץ של ענן הנקודות באופן אחיד
- ב. העתקה (למעלה/למטה/הצידה) של כלל הנקודות בתוך קוביית היחידה
- ג. תמונת מראה בציר האנכי (vertical flip)

## סיכום אישי:

זה היה פרויקט לא קל, אבל מעניין מאוד. למדתי הרבה, נחשפתי לתחומים חדשים, ועדיין אני מרגיש שזה רק קצה הקרחון. אני שמח שיצא לי לעשות פרויקט מעשי בנושא של למידת מכונה / ראייה חישובית / גרפיקה- השתתפתי בקורסים תיאורטיים בנושאים אלו, ואני חושב שהפרויקט משלים את הידע התאורטי בצורה טובה.

למרות שזה 'רק' מיני פרויקט, זה משמח (ואפילו מרגש) לראות שהצלחתי לכתוב ולממש קוד שעומד בסטנדרטים של הקהילה המדעית (בטבלה המצורפת). עם זאת, עדיין יש ניסיונות שניתן לעשות על מנת לשפר את התוצאות: ליצור תמונות בגודל שונה מ- 32X32, Data augmentation, טכניקות שונות לשילוב ה views, שימוש ב SIFT (scale invariant feature transform), hyper-parameters fine tuning, וכנראה דרכים נוספות.

Algorithm	ModelNet10 Classification (Accuracy)
Ma et al. [56]	95.29%
SPNet [52]	97.25%
MHBN [51]	95.0%
Point2Sequence [49]	95.3%
SO-Net[34]	95.7%
RotationNet[32]	98.46%
PANORAMA-ENN [29]	96.85%
3DmFV-Net [24]	95.2%
VRN Ensemble [9]	97.14%
Algorithm	ModelNet40 Classification (Accuracy)
MVCNN [3]	90.1%

הטבלה לקוחה מתוך <https://modelnet.cs.princeton.edu/> ומתארת את ההישגים של אלגוריתמים שונים על ה dataset.

מתוך 63 אלגוריתמים בטבלה המלאה, כאן מתוארים 9 שעברו רמת דיוק של 95% על modelnet10.

MVCNN (ואלגוריתמים נוספים מהרשימה) הופעל על ה modelnet40 dataset, שכולל 40 קטגוריות במקום 10, ונחשב לקשה יותר (זו כנראה הסיבה להפרש באחוז הדיוק).