סיבוב תמונות באמצעות רשת נוירונים

מודל חישובי

מאת

נדב הלחמי

עבודה זו מוגשת כעבודה בהיקף של 2 יח' כמילוי חלקי של הדרישות לקראת קבלת ציון במדע חישובי

עבודה זו בוצעה בהדרכת <u>שלמה רוזנפלד</u>

יוני 2016

תמצית

בפרויקט "סיבוב תמונות באמצעות רשת נוירונים" אפתור את אחת הבעיות המשמעותיות ביותר עמן מתמודד כמעט כל צלם יום-יום: סיבוב תמונות אוטומטי בעת העברת התמונות מהמצלמה למחשב. הרשת תזהה כפי שבן אדם מזהה שהתמונה הפוכה, ותסובב אותה בהתאם. השיטה תתבסס על רשת נוירונים שלימדתי שמגיעה ל-80% ולעתים אף ל-90% אחוזי הצלחה בזיהוי מצב התמונה.

תוכן עניינים

מבוא

הפרויקט "סיבוב תמונות באמצעות רשת נוירונים" עוסק בסיבוב תמונות אוטומטי במחשב באמצעות רשת נוירונים. עד היום היה על כמעט כל צלם לסובב את התמונות שצילם באופן ידני בעת העברתן מהתמונה למחשב. כלומר, כשהצלם רצה לצלם תמונה אנכית, הוא היה מחזיק את המצלמה במצב אנכי, אך במצלמות רבות אין חיישן תנועה שמזהה את מצב המצלמה ובעת העברת התמונות למחשב, התמונה הייתה נראית "על הצד". אני נתקלתי פעמים רבות, ועדיין נתקל יום-יום בתמונות שמוצגות על מסכים כאשר הן מסובבות ב-°90 עם או נגד כיוון השעון, ופעמים רבות אף מסובבות ב-°180. עבודת סיבוב התמונות היא עבודה קשה ומייגעת שכוללת זיהוי מצב התמונה הרצוי וסיבובה בהתאם.

את זיהוי מצב התמונה ניתן לעשות באמצעות רשת נוירונים. הרשת מקבלת כל תמונה ומחזירה את מצבה: זווית הסיבוב שלה.

העבודה נחלקת לשני חלקים עיקריים: יצירת בסיס הנתונים ולימוד הרשת ותוצאותיה, כאשר התוצר הסופי הוא תוכנה שמקבלת תיקיית תמונות ומחזירה אותן מסובבות כראוי. בחלק של יצירת בסיס הנתונים, הפכתי כל תמונה לוקטור, כאשר אורכו נקבע לפי הגודל שבחרתי לכל תמונה. בדקתי במהלך העבודה את השפעת גודל התמונה על התוצאות, במגבלות הזמן כמובן מכיוון שכל מחזור למידה של רשת לוקח זמן רב. בנוסף, ניסיתי גם לבסס את הרשת על תמונות שעברו מיצוע של הצבעים ועל הפרשי הצבעים בין הפיקסלים.

תיאוריה

רשת עצבית מלאכותית (ANN-Artificial Neural Network) או רשת קשרית הוא כינוי למודל מתמטי חישובי המאפשר לבצע הדמיה של תהליכים מוחיים או קוגניטיביים, ושל תהליכים המתרחשים ברשת עצבית טבעית. רשת מסוג זה מכילה בדרך כלל מספר רב של יחידות מידע (קלט ופלט) המקושרות זו לזו. צורת הקישור בין היחידות, המכילה מידע על חוזק הקשר, מדמה את אופן חיבור הנוירונים במוח. (ויקיפדיה)

הרשת מורכבת מ-3 שכבות: שכבה הכניסה, השכבה הנסתרת ושכבת מוצא. כל נוירון בשכבת הכניסה מקבל ערך קלט (במקרה הזה אחד משלושת ערכי RGB של כל פיקסל) ואותו הוא מכפיל במשקל מסוים עבור כל נוירון בשכבה הנסתרת. את ערכי הקלט יצרתי באמצעות הפונקציה reshape של ספריית numpy, שאפשרה לי לסדר את ערכי ה-RGB כוקטור אחד לכל תמונה. הנוירונים בשכבה הנסתרת מקבלים את הסכום של כל המכפלות, ועליו מפעילים את פונקציית האקטיבציה, במקרה הזה פונקציית "סיגמואיד" $f(x) = \frac{1}{1+e^{-x}}$. ככל שהערך קרוב יותר ל-1, כך הסיכוי שהתמונה מסובבת בהתאם למה שהנוירון מייצג גבוה יותר. כלומר, לפי הגדרתי, אם תתקבל תוצאה של [0.01,0.03,0.99,0.005] מכאן שיש סיכוי גבוה מאוד (לפי הרשת) שהתמונה מסובבת ב-°180 ואפשר להגיד שבמקרה זה הרשת קבעה שהתמונה מסובבת ב-°180. את מבנה הרשת בחרתי על סמך ניסוי וטעייה: לימדתי רשתות רבות לזהות תמונות וזו עם השגיאה הנמוכה ביותר, נבחרה. ההבדל בין הרשתות היה במספר הנוירונים בשכבה הנסתרת.

את אימון הרשת ניתן להציג ב-4 שלבים:

1. הגדרת משקלים רנדומליים לכל הקשרים.
2. אם הפלט נכון, לא לשנות את המשקלים.
3. אם הפלט גבוה אך אמור להיות נמוך, להנמיך את המשקולות הקשורות לנתוני הקלט הגבוהים.
4. אם הפלט נמוך אך אמור להיות גבוה, להגביה את המשקולות הקשורות לנתוני הקלט הנמוכים.

על שלבים 2-4 יש לחזור לפי הצורך.

ככל שמאמנים רשת יותר, כך יכולת ההכללה שלה נפגעת והיא עלולה להיכנס למצב של "התאמת יתר" עבור הדוגמאות ששימשו ללמידה שלה. כדי למנוע מצב של התאמת יתר, הפרדתי את הדוגמאות שלי ל-2 קבוצות: קבוצת אימון וקבוצת אימות. ככל שאימנתי את הרשת, כך התוצאות שלה עבור קבוצת האימון השתפרו, אך עבור קבוצת האימות, התוצאות השתפרו עד שהגיעו להתאמה מקסימלית, ואז נהיו פחות טובות. בנקודת המינימום של תוצאות הרשת עבור קבוצת האימות ניתן לקבוע ששם יש להפסיק את אימון הרשת כדי לא לפגוע ביכולת ההכללה שלה.

תיאור האלגוריתם עליו מבוססת התוכנה

יצירת בסיס הנתונים

<u>יבוא התמונות:</u>

```
images=[]
location='C:/Users/nadav/Documents/WinPython-64bit-2.7.10.2/notebooks/docs/Siduri sky images from google/'
for k in range(150):
    name=str(k)+'.JPG'
    img = Image.open(location+name)
    images.append(img)
```

את התמונות שמרתי במיקום מסוים על המחשב, כך שהן מסודרות לפי מספר סידורי (שם התמונה), ומחולקות לשתי קבוצות: תמונות אנכיות ("גבוהות") ותמונות אופקיות ("רחבות"). כדי לייבא את התמונות הגדרתי את מיקום התיקייה של כל התמונות, ובאמצעות לולאה ייבאתי כל תמונה לרשימה באמצעות הפונקציה Image.open של ספריית PIL (Python Imaging Library).

<u>יצירת רשימת תמונות מוקטנות:</u>

```
small_images = copy.deepcopy(images)
k=1
for i in range(len(small_images)):
    cols,rows=small_images[i].size
    small_images[i].thumbnail((cols/320*k, rows/320*k), Image.ANTIALIAS)
```

באמצעות הפונקציה deepcopy של ספריית copy, יצרתי אובייקט חדש של רשימת תמונות בשם small_images. את כל התמונות ב-small_images הקטנתי באמצעות הפונקציה thumbnail של ספריית PIL, לרזולוציה של 4X3X3 או 3X4X3 לפי המצב המקורי של התמונה. (במקרה זה עבדתי עם תמונות ברזולוציה של 1280X960 או 960X1280 ולכן החלוקה ב-320). מטרת ההקטנה של התמונות היא לאפשר לימוד מהיר ויעיל יותר של הרשת, שלא תוכל להתמודד עם כמות רבה מידי של נתונים.

<u>המרת התמונות למערכים:</u>

```
for i in range(len(small_images)):
    small_images[i]=np.array(small_images[i])
images_arr=[]
for i in range(len(images)):
    images_arr.append(np.array(images[i]))
```

למען נוחות העבודה בהמשך, ותמיכה של ספריית numpy, המרתי כל תמונה מ- Type: PIL.JpegImagePlugin.JpegImageFile ל-Type: numpy.ndarray.

יצירת מערכי תמונות מסובבות:

```
small_images=np.array(small_images)
small_images_90=[np.rot90(i,-1) for i in small_images]
small_images_90=np.array(small_images_90)
small_images_180=[np.rot90(i,-2) for i in small_images]
small_images_180=np.array(small_images_180)
small_images_270=[np.rot90(i,-3) for i in small_images]
small_images_270=np.array(small_images_270)
```

באמצעות הפונקציה rot90 של ספריית numpy ניתן לסובב תמונה מ-Type: numpy.ndarray ב-90° נגד כיוון השעון, לפי מספר הפעמים שהמשתמש בוחר. לדוגמה, אם הפונקציה תקבל את המספר 3-, תסובב הפונקציה את התמונה 3 פעמים עם כיוון השעון, כלומר ב-270° עם כיוון השעון. יצרתי 4 רשימות שאותן הפכתי למערכים של תמונות, כך שכל מערך מכיל תמונות המסובבות באותה דרך.

המרת התמונות לוקטורים:

```
small_images_list=[]
small_images_list_90=[]
small_images_list_180=[]
small_images_list_270=[]
for i in range(len(small_images)):
    small_images_list.append(small_images[i].reshape(36))
    small_images_list_90.append(small_images_90[i].reshape(36))
    small_images_list_180.append(small_images_180[i].reshape(36))
    small_images_list_270.append(small_images_180[i].reshape(36))
```

באמצעות הפונקציה reshape של ספריית numpy המרתי כל תמונה לוקטור באורך 36 (12 פיקסלים שכל פיקסל מורכב מ-3 ערכים-RGB).

יצירת ערכי מטרה:

```
small_images_target=[]
small_images_90_target=[]
small_images_180_target=[]
small_images_270_target=[]
for i in range(len(small_images)):
    small_images_target.append([1,0,0,0])
    small_images_90_target.append([0,1,0,0])
    small_images_180_target.append([0,0,1,0])
    small_images_270_target.append([0,0,0,1])
small_images_target=np.array(small_images_target)
small_images_90_target=np.array(small_images_90_target)
small_images_180_target=np.array(small_images_180_target)
small_images_270_target=np.array(small_images_270_target)
```

כדי שהרשת תוכל ללמוד לסווג את התמונות לפי מספר הסיבובים ב-90°, היא צריכה לקבל תמונות שהיא יודעת מה מצבן, כדי שתסיק מכך על תמונות חדשות. לכל תמונה יוצמד ערך המטרה שלה, שמייצג את מספר הסיבובים שהיא מסובבת עם כיוון השעון. [1,0,0,0] מייצג תמונות שאינן מסובבות, [0,1,0,0] מייצג תמונות שמסובבות ב-90° עם כיוון השעון וכו'.

הצמדת התמונות וערכי המטרה למערך אחד:

```
total_small_images=np.vstack((small_images,small_images_90,small_images_180,small_images_270))
total_small_images_targets=np.vstack((small_images_target,small_images_90_target,small_images_180_target,small_images_270_target))
small_images_inputs_targets=np.column_stack((range(len(total_small_images)),total_small_images,total_small_images_targets))
```

באמצעות קטע הקוד הנ"ל, יצרתי מערך אחד הכולל את כל התמונות בכל צורות הסיבוב, ולכל תמונה את ערך המטרה שלה. מערך זה יהיה בצורה של 600 שורות (150 תמונות ב-4 מצבים אפשריים), ו-41 עמודות (מספר סידורי, 12 פיקסלים RGB, 4 ערכי מטרה).

חלוקה לקבוצת אימון וקבוצת בחינה:

```
small_x_train,small_x_test,small_y_train,small_y_test=train_test_split
(small_images_inputs_targets[:,:-4],small_images_inputs_targets[:,-4:],test_size=1/4.0)
```

כדי שאוכל לבחון את הרשת בצורה מהימנה, הייתי צריך ללמד אותה באמצעות מספר תמונות, ובאמצעות תמונות אחרות לבדוק את תוצאותיה. כשם שתלמיד שלומד למבחן, לא מתרגל את אותם תרגילים שיהיו במבחן. את החלוקה ביצעתי באמצעות הפונקציה train_test_split של ספריית sklearn.cross_validation שגם מערבבת את התמונות, ביחס של ¼. כלומר, 450 תמונות לאימון ועוד 150 תמונות לבחינה.

הגדרת בסיס הנתונים כ-dataset:

```
small_train_ds=SupervisedDataSet(36,4)
small_test_ds=SupervisedDataSet(36,4)
small_train_ds.setField('input',small_x_train[:,1:])
small_train_ds.setField('target',small_y_train)
small_test_ds.setField('input',small_x_test[:,1:])
small_test_ds.setField('target',small_y_test)
small_train_indexes=small_x_train[:,0]
small_test_indexes=small_x_test[:,0]
small_train_ds.saveToFile('small_train_google_ds_4X3X3')
small_test_ds.saveToFile('small_test_google_ds_4X3X3')
np.save('small_train_google_indexes_4X3X3.npy',small_train_indexes)
np.save('small_test_google_indexes_4X3X3.npy',small_test_indexes)
```

כדי שאוכל להשתמש בספריית Pybrain ובאמצעותה לבנות רשת וללמד אותה לזהות את מצב התמונה, הייתי צריך ליצור בסיס נתונים שמורכב מהתמונות ומערכי המטרה מסוג SupervisedDataSet. יצרתי שני בסיסי נתונים: בסיס נתוני אימון לצורך לימוד הרשת ובסיס נתוני בחינה לצורך בחינת הרשת. את בסיסי הנתונים שמרתי באמצעות הפונקציה saveToFile של ספריית Pybrain.

לימוד הרשת ותוצאות

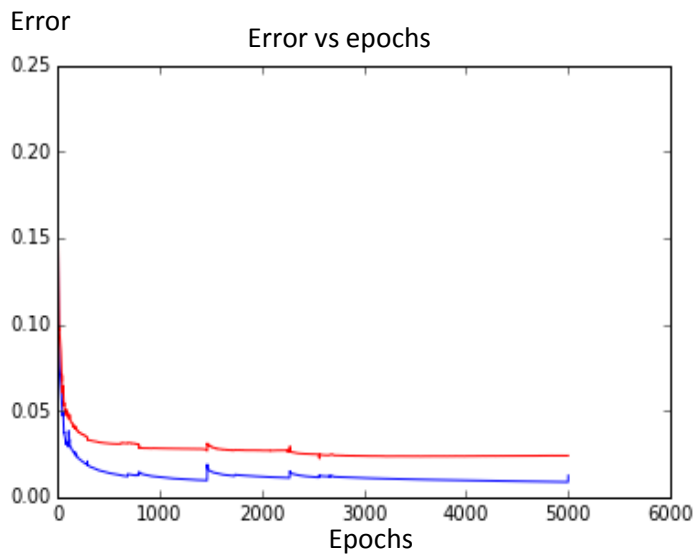
מציאת רשת אופטימלית:

```
small_nets=[]
small_err=[]
min_small_err=[]
for k in range(1,101):
    print k
    small_net=buildNetwork(36,k,4,bias=True,hiddenclass=SigmoidLayer,outclass=SigmoidLayer)
    small_trainer=BackpropTrainer(small_net,small_train_ds)
    small_err.append(small_trainer.trainUntilConvergence(maxEpochs=5000,continueEpochs=500,validationProportion=1/3.0))
    small_nets.append(small_net)

for e in small_err:
    min_small_err.append(min(e[1]))
small_idx=min_small_err.index(min(min_small_err))
```

כדי למצוא את מבנה הרשת האופטימלי, השתמשתי בלולאה שבונה 300 רשתות (בצילום המסך ניתן לראות רק 100 כי הרצתי במקביל על 3 ליבות), ושומרת עבור כל רשת את השגיאה המינימלית שלה. בסופו של דבר הרשת עם השגיאה המינימלית הנמוכה ביותר, היא זאת שנבחרת. הרשת האופטימלית שקיבלתי הייתה בעלת מבנה של 36 נוירונים בשכבת הכניסה, 69 נוירונים בשכבה הנסתרת ו-4 נוירונים בשכבת המוצא. את הרשת הטובה ביותר שמרתי באמצעות ספריית Pickle. ללימוד כל רשת השתמשתי בפונקציה trainUntilConvergence של ספריית Pybrain. הפונקציה מקבלת מספר מחזורי למידה מקסימלי, מספר מחזורי למידה לאחר שנמצא מינימום ויחס ערכי בדיקה צולבת/סך כל הערכים. לימוד הרשת מתבצע לפי האלגוריתם שתיארתי בחלק התיאוריה, ובכל מחזור למידה נבדקת תוצאת הרשת על ערכי הבדיקה הצולבת. אם נמצא מינימום מקומי בתוצאות שגיאת הרשת עבור ערכי הבדיקה, הלימוד ימשיך עוד 500 מחזורי למידה עד אשר ימצא מינימום חדש וקטן יותר. אם לא ימצא, ייפסק תהליך הלמידה ותיבחר הרשת עבורה מתאים המינימום. את הפרמטרים השונים של הפונקציה בחרתי

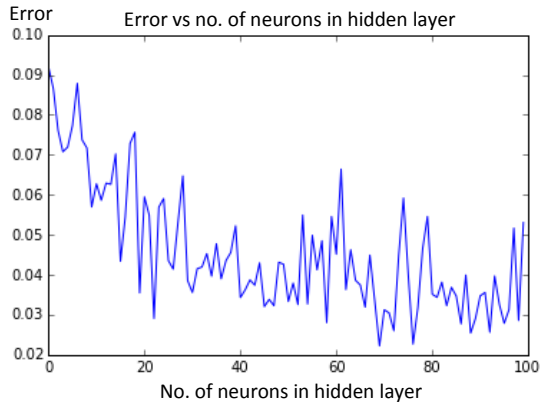
על סמך הרצות שונות בהן בדקתי כמה מחזורי למידה יש ללמד את הרשת כדי שהיא תמשיך להשתפר, מכיוון שככל שתהליך הלמידה ממשיך, השינוי בתוצאות הרשת מצטמצם. בגרף ניתן לראות את תהליך הלמידה של הרשת הטובה ביותר. הרשת משתפרת עם מחזורי הלמידה, אך השיפור קטן עמם.



<u>תוצאות האימון והבחינה:</u>

```
def get_err_in_percent(net,ds):
    h=net.activateOnDataset(ds)
    return 100-sum(h.argmax(axis=1)==ds['target'].argmax(axis=1))/float(ds['target'].shape[0])*100
```

הגרף מציג את השגיאה של הרשתות השונות כתלות במספר הנוירונים בשכבת הביניים של כל אחת מהן. ניתן לראות שהמינימום המוחלט של הגרף הזה הוא אכן ב-69. באמצעות השיטה get_err_in_percent קיבלתי את היחס (באחוזים) בין מספר התמונות שלא זוהו נכון למספר התמונות הכולל. ניתן לראות שעל קבוצת הלימוד יש 94.67% הצלחה ועל קבוצת הבחינה 90% הצלחה.




```
small_train_result=get_err_in_percent(small_net,small_train_ds)
small_train_result
```

```
5.33
```

```
small_test_result=get_err_in_percent(small_net,small_test_ds)
small_test_result
```

```
10.0
```

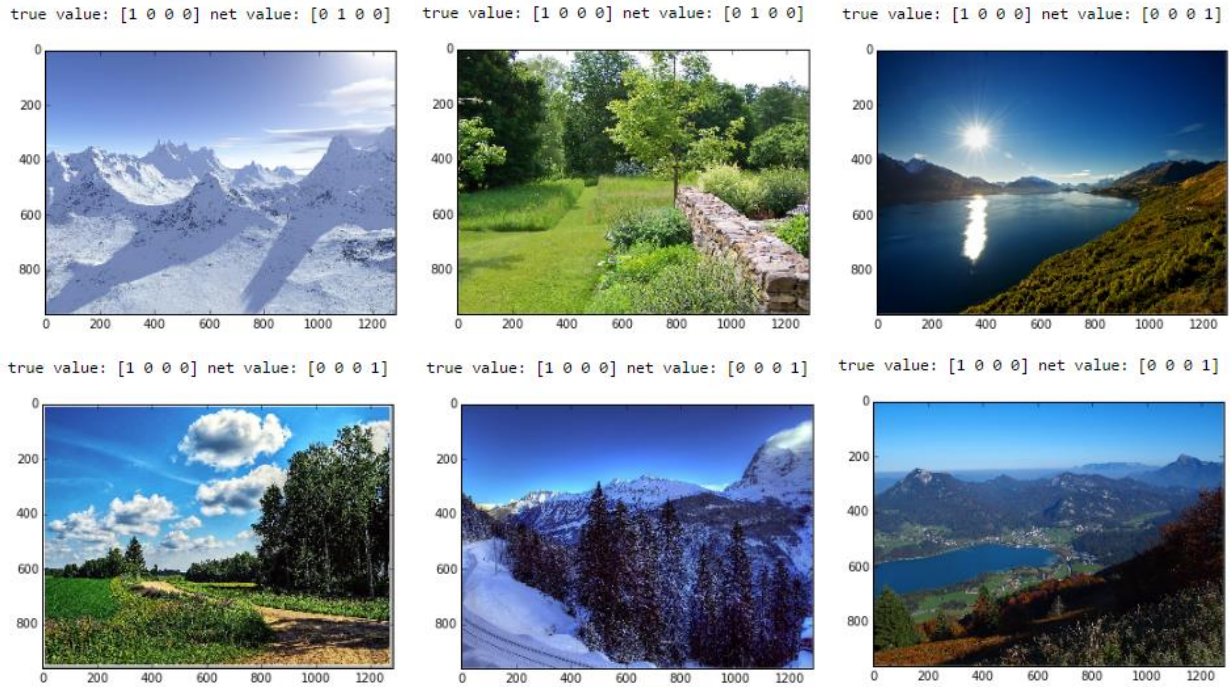
עימות המודל הממוחשב אל מול המציאות:

כדי לבחון את תוצאות הרשת מול מטרתה האמתית, היה עליה לקבל תיקיית תמונות שחלקן מסובבות, ולסובב אותן בהתאם. את תיקיית התמונות הזאת יצרתי באמצעות 150 תמונות שהורדתי מגוגל, ברזולוציה של 1280X960 (75 תמונות) ו-960X1280 (75 תמונות נוספות), כאשר מתוכן 80 תמונות (53.33%) לא סובבתי כלל, 35 תמונות (23.33%) סובבתי ב-°90 עם כיוון השעון, 25 תמונות (16.66%) סובבתי ב-°270 עם כיוון השעון ו-10 תמונות (6.66%) סובבתי ב-°180. הרשת הצליחה לזהות נכון את מצבן של **80.67%** מהתמונות, שהן 121 תמונות. חשוב לציין, שתוצאות רשת לא מאומנת, יהיו בסביבות 25% הצלחה (סיכוי של 1 ל-4 "לקלוע" למצב הנכון של התמונה), ולכן תוצאות הרשת שלי שהן טובות **יותר מפי 3** מתוצאות הניחוש, טובות במיוחד.

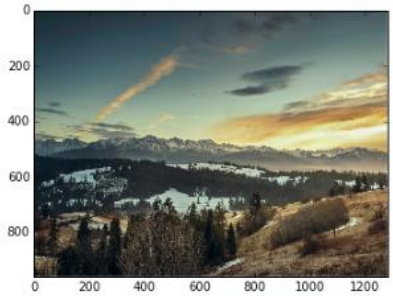
כל משתמש בעל Python יוכל להגדיר תיקיית מוצא (התמונות המקוריות) ותיקיית יעד (בה ישמרו התמונות המסובבות), זו יכולה להיות גם אותה תיקייה, ואז באמצעות הרצה אחת בלבד של הקוד ללא הגדרות נוספות, התמונות יסובבו וישמרו על המחשב באופן אוטומטי.

תמונות שהרשת נכשלה בזיהוי מצבן:

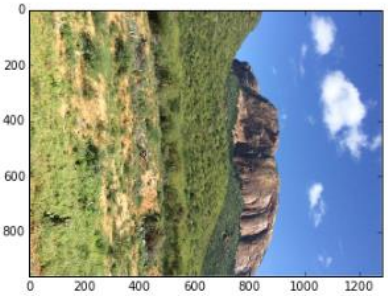
Net value- הערך שמתקבל מהרשת. True value- הערך האמיתי.



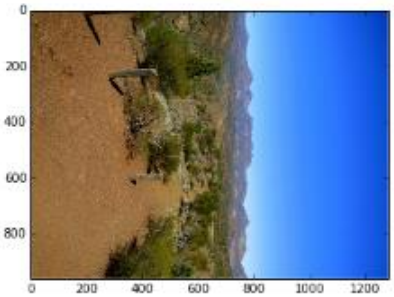
true value: [1 0 0 0] net value: [0 1 0 0]



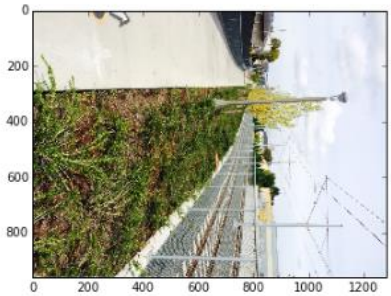
true value: [0 1 0 0] net value: [0 0 0 1]



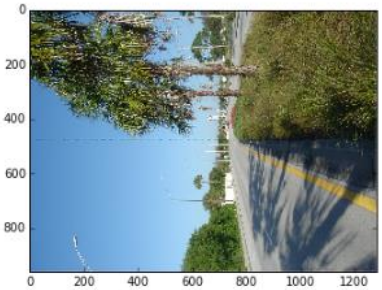
true value: [0 1 0 0] net value: [0 0 1 0]



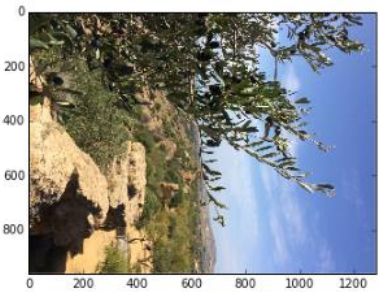
true value: [0 1 0 0] net value: [0 0 1 0]



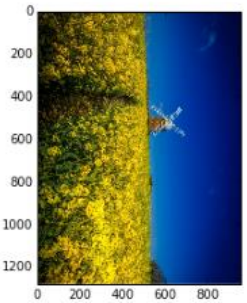
true value: [0 0 0 1] net value: [1 0 0 0]



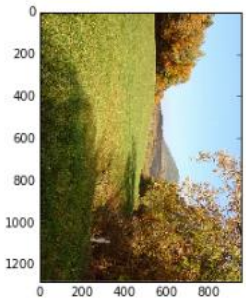
true value: [0 1 0 0] net value: [0 0 1 0]



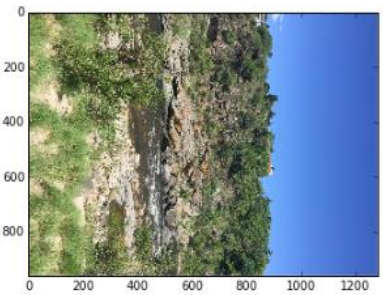
true value: [0 1 0 0] net value: [1 0 0 0]



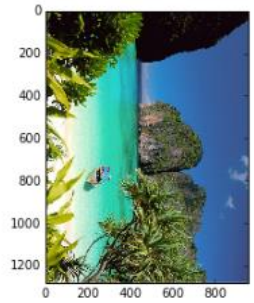
true value: [0 1 0 0] net value: [1 0 0 0]



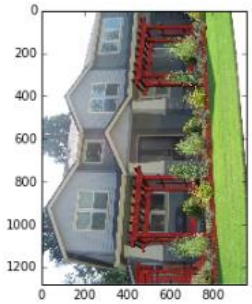
true value: [0 1 0 0] net value: [0 0 0 1]



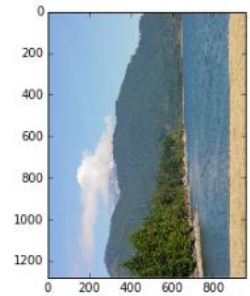
true value: [0 1 0 0] net value: [0 0 0 1]



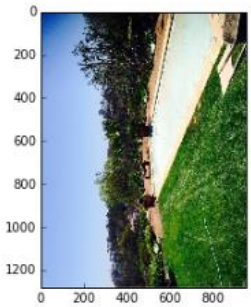
true value: [0 0 0 1] net value: [1 0 0 0]



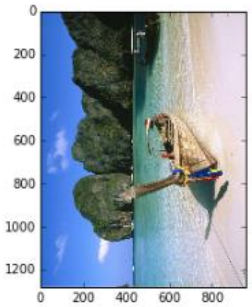
true value: [0 0 0 1] net value: [1 0 0 0]



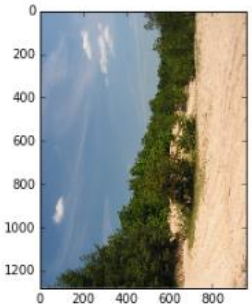
true value: [0 0 0 1] net value: [1 0 0 0]



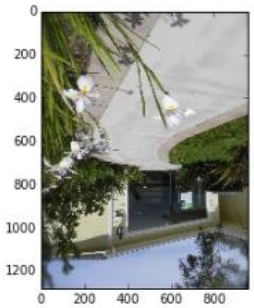
true value: [0 0 0 1] net value: [0 1 0 0]



true value: [0 0 0 1] net value: [1 0 0 0]

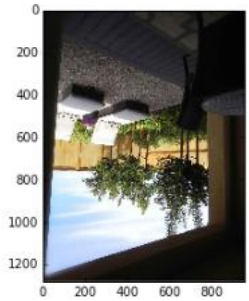


true value: [0 0 1 0] net value: [1 0 0 0]



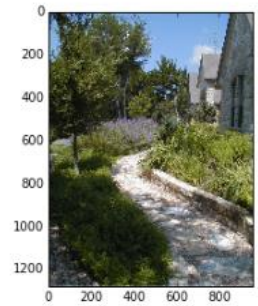
true value: [0 0 1 0] net value: [0 1 0 0]

true value: [0 0 1 0] net value: [0 0 0 1]

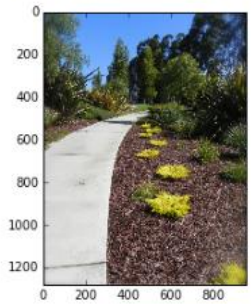


true value: [1 0 0 0] net value: [0 0 1 0]

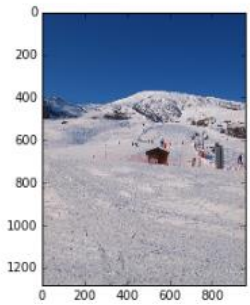
true value: [1 0 0 0] net value: [0 1 0 0]



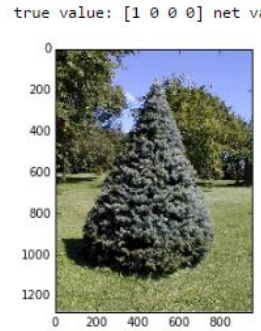
true value: [1 0 0 0] net value: [0 0 0 1]



true value: [1 0 0 0] net value: [0 0 1 0]



true value: [1 0 0 0] net value: [0 1 0 0]



מסקנות

- במהלך הכנת הפרויקט ניסיתי רשתות בגדלים שונים (16X12X3, 8X6X3, 4X3X3, 2X2X3, 28X21) ומסוגים שונים (שחור לבן וצבעוני). בנוסף, מעבר להכנסת התמונות כערכי כניסה לרשת, הוספתי בחלק מההרצות גם את ההפרשים בין הפיקסלים כערכי כניסה. הרשת שהחזירה את התוצאה הטובה ביותר בכל הפרמטרים הייתה הרשת במבנה של 4X3X3. מכך אני מסיק שכדי להשיג את התוצאות הטובות ביותר היה עלי לשמור על יחסי הגובה והרוחב של התמונה ולצמצם את היקפי הרשת.
- אדם שרואה את התמונות ברזולוציה של 4X3X3 יצליח ברוב המקרים לזהות טוב יותר מהרשת את מצב התמונות. עובדה זו הפתיעה אותי מאחר שברזולוציה כה נמוכה, גם לאדם קשה לזהות עצמים, ולכן לכאורה אין סיבה שיצליח יותר מהרשת. מכך אני מסיק שכנראה שלרשת לא היו מספיק תמונות ללמוד מהן, ושאפשר לשפר את התוצאות תוך שמירה על המבנה של 4X3X3.
- רשתות שהתבססו על תמונות צבעוניות השיגו תוצאות שהיו טובות יותר באופן משמעותי מרשתות שהתבססו על תמונות בשחור-לבן. מכך אני מסיק שכמו האדם, גם למחשב חשובה מאוד אינפורמציית הצבעים כדי לזהות את מצב התמונה.
- מבין תמונות המבחן, 5 תמונות כללו שלג וב-3 מהן הרשת לא הצליחה לזהות את מצבן. כלומר, רק 40% הצלחה עבור תמונות שכוללות שלג. ניתן להסביר זאת בכך שרוב התמונות על פי הן נלמדה הרשת צולמו בארץ, או בחו"ל בחודשי הקיץ, ולכן הרשת לא ידעה כיצד להתמודד עם שלג. בנוסף, שלג עלול להיות דומה לעתים לעננים, או לשמיים בהירים במיוחד.
- הרשת נוטה לטעות בזיהוי מצב התמונה כאשר השמיים כהים, כלומר בסביבות השקיעה והזריחה בדרך כלל.
- מדרכות בהירות גם כן מבלבלות לעיתים את הרשת, שחושבת שהן החלק העליון בתמונה, וכך גם לגבי חול ודשא לפעמים.
- תמונות בגוון כהה, או שעברו עריכה באמצעות פילטרים שמכהים אותן, גם כן מכשילות את הרשת לעיתים.