בס"ד

מבוא לראייה ממוחשבת: מטלת מנחה (ממ"ן) 13 - נדב כחלון
דו"ח פרויקט

פרויקט הסיום בקורס שלנו מטפל בבעיה של זיהוי פונטים בתמונות מהעולם. נתבקשנו לבנות מודל שיצליח לסווג טקסט טבעי ל-7 מחלקות פונטים שונות:

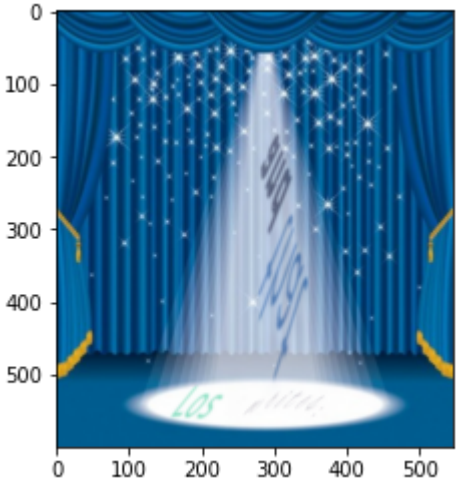
Open Sans, Raleway, *Alex Brush*, Roboto, **Russo One**, Ubuntu Mono, Michroma

כאשר טקסט מהפונטים השונים הושתל לתוך תמונות מהעולם האמיתי בעזרת הטכניקה בה השתמשו Gupta et. al ב-[1] ליצירת סט נתונים של טקסט סינתטי.
בדו"ח זה אפרט על ההתמודדות שלי עם הבעיה: תהליך העבודה, התוכניות השונות, המודל הסופי והתוצאות. העבודה מבוססת על מודלים של רשתות נוירונים, עם מספר "טריקים" מקוריים משלי.

תהליך העבודה ותיאור המערכת

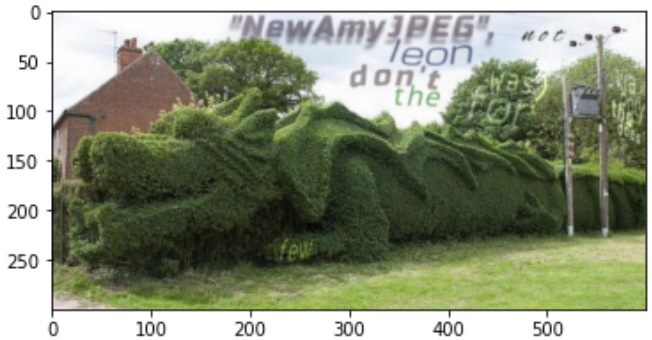
<u>חלק ראשון</u>: הכנת הנתונים

כדי להתחיל את העבודה ניסיתי להבין איך סט הנתונים שלנו בנוי, ואיך אוכל להביא אותו לצורה נוחה כקלט לרשת נוירונים. השתמשתי בסט הנתונים שקיבלנו בפורמט HDF5, והתחלתי לשחק איתו ולנסות להבין איך הוא בנוי. בהתחלה לא חקרתי לעומק על פורמט הקובץ, אלא השתמשתי בקוד שקיבלנו לייבוא הנתונים. למדתי על האופן בו מסודרים הנתונים - הטקסט, הפונטים, התמונות, וה-bounding boxes. למדתי שה-bounding boxes מיוצגים כמטריצה שעמודותיה אלו פינות ה"קופסה".



מהתבוננות ראשונית על התמונות הבנתי שהטקסט מושתל לתוך המציאות בעזרת מיפוי מסוים, ש"מעוות" את התווים כדי להתאים למשטחים בסצנה. העיוות הזה יוצר קלט בצורות מאוד לא הומוגניות. דוגמה משמעותית ניתן לראות משמאל. לי - כבן אנוש, מאוד קשה להבין מה קורה שם. מכיוון שרשת נוירונים היא מודל בהשראת המוח האנושי - כנראה גם לה לא יהיה קל. האינטואיציה הזו הובילה אותי לחפש טרנספורמציה מתאימה למיפוי התווים לאוריינטציה אחידה, תוך שימוש באוריינטציה של ה-bounding boxes.

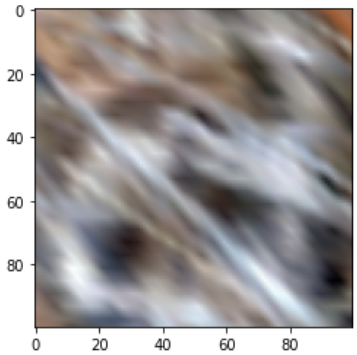
האינטואיציה הראשונה שלי הייתה כמובן - הומוגרפיה, אותה טרנספורמציה פרויקטיבית שלמדנו בקורס. לפי מה שלמדנו, משטחים מישוריים ממופים בעזרת הומוגרפיה אל תוך מרחב התמונה, ואם גם הטקסט הושתל אל תוך משטחים מישוריים - הומוגרפיה היא הפתרון. מהתבוננות בתמונות נראה שמילים שלמות לא ממופות למשטחים ישרים - למשל המילה don't בתמונה שלמטה (נראה כאילו הישר עליו כתובות האותיות ממופה למעין קשת קלה), מה שגרם לי להטיל ספק בהומוגרפיה. עדיין קיימת האפשרות שתווים אינדיבידואלים ממופים בעזרת הומוגרפיה, אך לא הייתי בטוח.

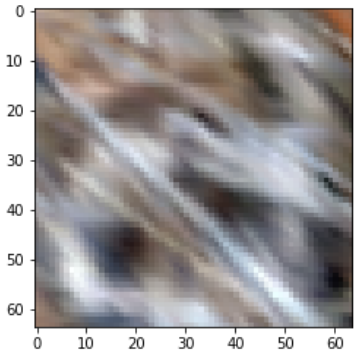


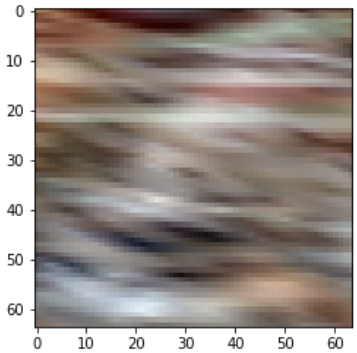
כדי לאמת את ההשערה התחלתי לקרוא את המאמר של Gupta et. al ב-[1] - לפיו נוצר בסיס הנתונים, ואת המאמרים אליהם הוא מפנה. עדות טובה לכך שמדובר בהומוגרפיה קיבלתי בסעיף 2 במאמר הנ"ל. החלטתי גם לצלול לעומק לקוד המצורף, בעזרתו בסיס הנתונים נוצר, וגם שם קיבלתי רושם טוב שה-warping נעשה ע"י הומוגרפיה.
בין היתר גם ראיתי שם את המקום בו נוצר סט הנתונים עצמו - וקיבלתי קצת יותר הבנה לגביי הפורמט.

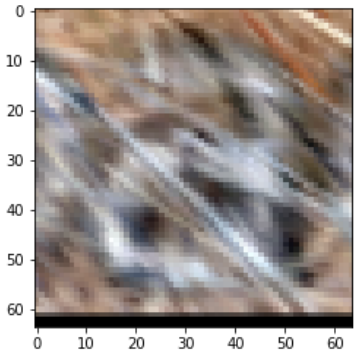
בסופו של דבר - החלטתי לעבוד עם הומוגרפיה כדי למפות חזרה את התווים למבנה אחיד (כידוע הרי - הטרנספורמציה ההפוכה להומוגרפיה היא הומוגרפיה).

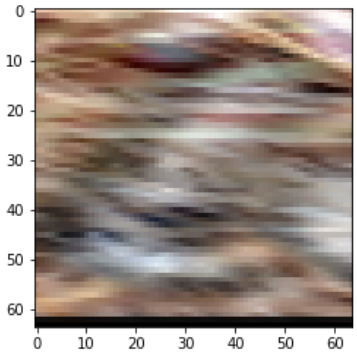
כעת ניסיתי להחליט לאן המיפוי יתבצע. חקרתי קצת בניסיון להבין האם קיימים מודלים של רשתות נוירונים שמסוגלים לקבל קלט בגדלים שונים. גיליתי שכן ואפילו התחלתי לחקור עליהם (ספציפית: על DCN ו-SPN) - אך הבנתי שהם די מורכבים ואין מימוש טוב שלהם בספריה נוחה. לפיכך החלטתי להתרכז במיפוי למלבן בגודל קבוע, ולנסות אחרת אם יישאר לי זמן.











אך באיזה גודל מלבן אבחר? הרעיון הראשון שלי היה לבחור את הרוחב של המלבן כמרחק המקסימלי בין ערכי x של הפינות של ה-bounding boxes בקלט שלנו, ודבר דומה לגביי האורך. הרוחב המקסימלי נמצא בתמונה road_112.jpg_0 והוא היה 166.26 פיקסלים, והאורך המקסימלי ב-stage_104.jpg_0 והוא היה 157.75 פיקסלים. הסתכלתי בתמונות הללו וראיתי שהן מאוד "מתוחות" (זו עם האורך המקסימלי, למשל, היא התמונה הראשונה בעמוד הקודם), ולבחור במימדים כאלו יהיה בזבזני.

מצד אחד - אני רוצה קלט קטן בשביל אימון מהיר של המודל, אבל מצד שני - אני רוצה קלט גדול כדי לא לבזבז מידע. התחלתי לשחק עם ממדים שונים של תמונות לנסות לראות מה יהיה אידיאלי. ראיתי שהומוגרפיה למלבן של 100X100 פיקסלים שומרת די הרבה מידע - ראה/י תמונה ראשונה משמאל. אולי זו כמות מיותרת של מידע? ניסיתי גודל אחר - 64X64 פיקסלים - ראה/י תמונה שניה משמאל. קיבלתי תמונה קצת פחות איכותית, אבל עדיין מציגה את עיקר המידע הרלוונטי. התלבטתי הרבה בניסיון לבחור רזולוציה טובה, והחלטתי להמשיך הלאה עם 64X64 פיקסלים.

עדיין נראה שקשה מאוד להבין מה קורה בתמונות, לא משנה באיזה רזולוציה בחרתי. אפשר להבין, איכשהו, שמדובר בתו a, אבל דוגמה אחרת בילבלה אותי מאוד - ראה/י תמונה שלישית משמאל. בהתחלה זה היה נראה לי כסתם רעש, ורק אחרי שבדקתי וראיתי שמדובר באות l הצלחתי לזהות אותה. גם פה - האינטואיציה לפיה רשת נוירונים בנויה בהשראת המוח האנושי, הובילה אותי לחפש דרך להקל על הרשת, מה שהוביל אותי לטריק מקורי, עליו אפרט מיד.

הרעיון נבע בהשראת האופן בו אנחנו, כבני אנוש, מסיקים מידע מהסביבה. אנחנו לא מסתכלים נקודתית על המידע - אלא על ההקשר כולו! אולי, ההקשר יוכל לעזור להבין מה קורה בתמונות. אינטואיציה זו הובילה אותי לרעיון של הוספת **שוליים מהסביבה** לתמונה - עוד כמה פיקסלים שיציגו את מה שקורה מסביב לתו. למשל - בתמונה הרביעית משמאל אפשר לראות את התו a המופיע בשתי התמונות הראשונות, כאשר 10 פיקסלים בכל צד הוקצו כשוליים להציג את הסביבה של התו (רזולוציית התמונה כולה היא עדיין 64X64 פיקסלים). בבירור - קל יותר להבין במה מדובר! אפילו אותה l משובשת נראתה קצת יותר הגיונית (ראה/י תמונה חמישית משמאל). שיחקתי עם השוליים בניסיון למצוא גודל טוב מספיק, והחלטתי להמשיך הלאה עם שוליים של 10 פיקסלים בכל צד.

השלב הבא היה עיבוד כל סט הנתונים, ויצירת סט "מרוכז" חדש, המכיל רק את התמונות ה"מרוכזות" של תווים (אחרי הומוגרפיה לפי ה-bounding box). חשבתי על דרכים שונות לעשות את זה: האינטואיציה הראשונה הייתה לארגן את המידע בתיקיות במערכת הקבצים, אבל אז נזכרתי בפורמט HDF5 בו קיבלנו את

סט הנתונים. תהיתי אם אוכל לנצל אותו לארגון הנתונים מחדש, בלי להסתרבל עם המון תיקיות וקבצים. יצאתי לחקור על הפורמט, ועל הAPI שמספקת הספרייה h5py לפייתון. צפיתי במדריכים, קראתי את הדוקומנטציה של הספריה, ויצרתי קבצי "צעצוע" כדי ללמוד את הממשק. כשהרגשתי שאני שולט מספיק בפורמט - ניגשתי לכתוב את המודול הראשון של הפרויקט: buildup.py.

המודול buildup.py נועד "לבנות" את בסיס הנתונים ה"מרוכז" תוך שימוש בכל הרעיונות שהצגתי לעיל. בהתחלה חשבתי להתעלם מכל המידע של "אילו תמונות-מרוכזות מייצגות אילו תווים ששייכים לאילו מילים ואילו תמונות", וליצור "ערימה" שלמה של כל התווים והפונטים. מספר שיקולים הובילו אותי לחשוב על דרך אחרת לסידור המידע - דרך שתהיה יותר מועילה בהמשך (למען האמת, חלק מהשיקולים אפילו עלו לי רק בשלבי האימון, והפעלתי אותם רטרואקטיבית).

- רעיון ראשון היה להשאיר מידע לגביי זהות התו בתמונה המרוכזת, ולא רק זהות הפונט שלו. האינטואיציה לכך עלתה לי כבר קודם, כשבחנתי דרכים "לרכז" את התמונות על התווים. שמתי לב, שרק אחרי שבדקתי שמדובר בתו l בתמונות שלעיל, באמת הצלחתי להבחין במשהו בעל משמעות. שם קיבלתי את הרעיון **ליצור רשתות נוירונים שונות לתווים שונים**, רעיון שיש לו משמעות אדירה בהמשך הדרך, ואפרט עליו בשלב האימון.
- רעיון שני היה להשאיר מידע לגביי השייכות של תווים מסוימים למילים מסוימות. אם נתון לנו שתווים באותה מילה שייכים לאותו פונט - ללא ספק אפשר להשתמש במידע הזה לנסות למצוא "קונצנזוס" על זהות הפונט בין כל התווים באותה מילה.
- רעיון שלישי היה לא "לזרוק" לגמריי את המידע לגביי אילו תווים שייכים לאילו תמונות, לערבב הכל ביחד, ורק אז לפצל לסט אימון, ולידציה, וטסט. לעומת זאת - לעשות את הפיצול לסט אימון, ולידציה, וטסט, **על התמונות** (ולא על התווים). האינטואיציה לכך הייתה שנתוני הבחינה הסופיים שנקבל יהיו מתמונות חיצוניות, שלא הופיעו בכלל בסט האימון. אם סט האימון וגם סט הטסט יכילו תווים מאותן תמונות, הערכת הדיוק על סט הטסט עשויה להיות מוטית.

כל זה הוביל אותי ליצירת ההיררכיה עליה אחראית התכנית buildup.py בקובץ הפלט NeatSynthText.h5. המבנה של קובץ הפלט מפורט היטב היטב בראש הקובץ buildup.py, ויהיה מיותר (וארוך) לחזור עליו פה. נקודה חשובה שדווקא כן אציין פה היא השימוש במה שקראתי לו "fontis": במקום שנצטרך לדבר בהמשך הדרך במונחים של "שמות של פונטים", החלטתי למפות אינדקס לכל פונט. לאינדקסים כאלו קראתי "fontis". דבר זה יפשט מאוד את התהליך ויעזור בהמשך. המיפוי הזה מופיע ברשימה 'fonti_map' בתחילת הקוד ב-buildup.py.

מספר נקודות ובעיות נוספות שהתמודדתי איתן בשלב הזה:

- חשבתי על כך שהיחס בין הצלעות של ה-bounding boxes הוא חשוב, ובשימוש בהומוגרפיה למלבן קבוע אנחנו מאבדים אותו. למשל עבור התו l (שהוא יחסית "ארוך") - האורך לרוב גדול יותר מהרוחב. אבל - מכיוון שהחלטתי ליצור רשתות נוירונים שונות לתווים שונים, זו לא אמורה להיות בעיה, אז לא התקדמתי לנסות לפתח רעיונות לטפל בזה.
- נתקלתי גם בבעיה מעניינת ביצירת הקובץ NeatSynthText.h5: הייתי צריך לאפיין בעזרת מחרוזת מסוימת את ה-datasets המתייחסים למידע לגביי דוגמאות מאותו תו. בהתחלה חשבתי לאפיין אותם בעזרת התו עצמו (למשל - ה-dataset שמתייחס לתו a ייקרא 'a'), אך קיבלתי שגיאה בניסיון ליצור dataset שיקרא '.'. לפיכך, החלטתי לאפיין אותם בעזרת ערך הASCII שלהם: למשל, ה-dataset המתייחס לתו n יקרא '110'.

הפרטים הקטנים בתוכנית buildup.py מפורטים היטב בקובץ עצמו, ויהיה מיותר (וארוך) לחזור עליהם פה. בזאת מסתיים ארגון המידע להמשך הדרך.

<u>הערה טרמינולוגית קטנה לפני שנמשיך לחלק הבא</u>: מעתה והלאה אשתמש במונח "מודל" לציון רשת נוירונים, ולא לציון מערכת הסיווג כולה. מערכת הסיווג תשתמש בכמה רשתות נוירונים, כלומר בכמה מודלים. התוצר הסופי, בטרמינולוגיה הזו, הוא לפיכך לא "מודל", אלא "מערכת סיווג" המשתמשת ב"מודלים".

<u>חלק שני</u>: מודלים ואימון

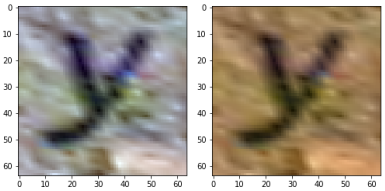
עכשיו, נצטרך לאמן מודלים לזיהוי הפונטים של התווים בתמונות המרוכזות.

כפי שכבר ציינתי קודם החלטתי על הכיוון של אימון **רשתות נוירונים שונות לתווים שונים**. למשל - רשת שתסווג פונטים של התו "l", רשת אחרת שתסווג פונטים של התו "a", וכו'. האינטואיציה לכך עלתה לי בשלב הקודם, כאשר בחנתי דרכים לרכז את התמונות על התווים האינדיבידואלים, כקלט לרשתות הנוירונים. שמתי לב שעבורי, כבן אדם, היה מאוד קשה להבחין במשהו בעל משמעות בתמונות עמוסות בטקסטורה, והתחלתי להבחין במידע משמעותי רק אחרי שידעתי באיזה תו מדובר. לפיכך - אולי גם רשת הנוירונים, שמבוססת על המוח האנושי, יכולה למצוא תועלת במידע לגביי זהות התו שבקלט, ובניית רשתות שונות לתווים שונים היה נשמע לי רעיון מעניין.
הרי באופן כללי - תווים שונים בנויים שונה, ולא בהכרח תהיה חפיפה מלאה בין הפיצ'רים שמאפיינים תווים שונים באותו פונט. ניצול המידע הזה והתאמת רשת נוירונים עבור כל תו אינדיבידואלי (ולמידת הפיצ'רים שמייחדים אותו בכל פונט), היה נראה לי כמו דרך טבעית וטובה להתקדם.
אמנם - יש כמה בעיות עם הרעיון הזה (בעיות שכמובן אתייחס אליהן בהמשך), אך גם אותן הצלחתי לפתור בהצלחה (בהמשך).

בשלב הזה החלטתי לגשת וללמוד על ה-API שמספקת הספרייה tensorflow (ולמעשה - על כל תהליך העבודה עם רשתות נוירונים). הרצתי כמה דוגמאות "צעצוע" כדי לוודא שאני מבין טוב איך הדברים עובדים.

החלטתי להתחיל בבניית ארכיטקטורה שתשמש את כל רשתות הנוירונים בפרויקט, כזו שתתאים טוב ללמידת סיווג פונטים של תווים. כדי לשפר אותה, הייתי צריך לבחור תו ספציפי עליו אני אבחן את הרשת. בחרתי בתו עבורו יש לי לא מעט דוגמאות אימון, והוא גם "מעניין" במבנה שלו - התו h. היה נראה שעם ארכיטקטורה די פשוטה אני מגיע לתוצאות הכי מוצלחות, אבל עדיין התוצאות על סט האימון היו טובות יותר מהתוצאות על סט הוולידציה - התרחש overfitting. ניסיתי למצוא דרכים להתמודד עם זה, ונתקלתי ב**טכניקת האוגמנטציה**.

ניסיתי דרכים שונות להוסיף אוגמנטציה למודל, חלקן צלחו וחלקן לא נתנו שיפור משמעותי. הדרך המוצלחת ביותר הייתה **עיוות הצבע** של התמונות תוך הוספה רנדומלית של ניגוד, בהירות, רוויה, וגוון בתמונות הקלט, בשכבה מיוחדת ברשת הנוירונים המיועדת לבצע את העיוות הזה בכל איטרציה של תהליך האימון (ובכך לחדש את סט האימון בכל פעם). האינטואיציה מאחורי זה הייתה למנוע מהרשת להתאים את עצמה לצבעים של תמונות האימון, במקום ללמוד את המבנה של התווים המופיעים בהן. לצערי, לא היה ב-tensorflow שכבה מוכנה שעושה עיוות כזה לצבע, לכן הייתי צריך ליצור **שכבה מותאמת אישית** תוך בניית מחלקה חדשה - `RandomColorDistortion`, היורשת מ-`tf.keras.layers.Layer`, וממשת את העיוות הזה (ניתן למצוא את ההגדרה שלה בקובץ training.py עליו אפרט בהמשך). כדי לעשות את זה, למדתי על האופן בו מיוצגות שכבות ב-tensorflow, ועל הדרכים ליצירת שכבות מותאמות אישית.
דוגמה לתוצאה של העיוות הזה (לפני - מימין, אחרי - משמאל):



ניסיתי גם דרכים אחרות אוגמנטציה, ואת רובן נטשתי מחוסר בשיפור משמעותי.
כמה מהן היו:

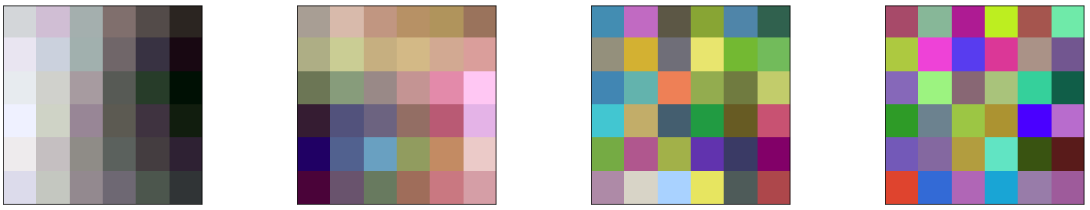
- הוספת שכבה לרשת היוצרת רעש גאוסיאני אקראי על התמונות המרוכזות.
הרעיון ננטש מחוסר בשיפור משמעותי.
- הוספת שכבה לרשת אשר מסובבת, מזיזה, ומתקרבת/מתרחקת לתמונות בצורה אקראית. את הדרך הזו נטשתי מכיוון שבכל מקרה אנחנו מרכזים את התווים באוריינטציה נכונה בעזרת הומוגרפיה מה-bounding box, ואין טעם בשכבה כזו. יתרה מזאת - לאוריינטציה עשויה להיות חשיבות בזיהוי הפונט (בעיקר אם אנחנו יוצרים רשתות שונות לתווים שונים).

- הוספת רעש גאוסיאני ועיוות הצבע של התמונות המקוריות (לפני שהפעלנו עליהם הומוגרפיות). האינטואיציה לכך הייתה להוסיף טקסטורה אקראית לתמונות, אך לא באמת ראיתי שיפור משמעותי. יתרה מזו - האוגמנטציה הזו הפכה את סט הנתונים לכבד הרבה יותר מבחינת זיכרון (כי היא קורית לפני תהליך האימון, ולא בתוך שכבות המודל).

כעת ראוי לציין שבינתיים, ביצעתי רגולריזציה ע"י שכבות Dropout בלבד - בין השכבות החבויות, והשתמשתי רק בפונקציית אקטיבציה ReLU.

בשלב הזה נזכרתי בנקודה מעניינת שנגענו בה בשיעורים: רשתות נוירונים (ורשתות קונבולוציה בפרט) בנויים בהשראת מוחות של בעלי חיים, ומחקרים הראו שבשלבים הראשונים של עיבוד המידע מוחות של בעלי חיים מזהים קודם כל צורות בסיסיות מאוד. ראינו בשיעורים גם כיצד, עם רגולריזציה מתאימה, ניתן לראות ממש תבניות בסיסיות בקונבולוציות המופיעות בשכבה החבויה הראשונה של רשתות קונבולוציה מוצלחות. כיוון מחשבה זה הוביל אותי לנסות לראות האם גם בשכבה החבויה הראשונה של הרשת שלי (שאני רוצה שתהיה מוצלחת), לפילטרים שנלמדו יש צורה מעניינת.

אבל - נחלתי אכזבה. להלן ארבעה מהפילטרים שלמדה הרשת שלי (אחרי שנירמלתי אותם):

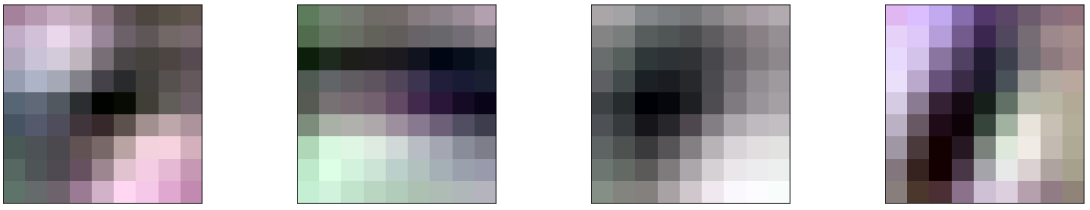


רובם המוחלט של הפילטרים נראו כמו זוג הפילטרים הימיניים - רעש מוחלט. כמה נראו כמו הפילטר השלישי מימין - בעלי צורה מסוימת, עם המון רעש. בודדים מאוד נראו כמו הפילטר השמאלי - צורה ברורה עם מעט רעש.

הבנתי שהמודל התאים את עצמו מאוד לרעשים בתמונות האימון, ורוב הזמן לא בדיוק הראה ניסיון להבין תבניות וצורות מסוימות. החלטתי, כפי שלמדנו בקורס, להוסיף **רגולריזציית weight-decy** בשכבה החבויה הראשונה (במקום רגולריזציה ע"י שכבת dropout כמו בין השכבות החבויות האחרות), בתקווה שהמודל ינצל את השכבה הראשונה ללמוד פיצ'רים מעניינים ותבניות מעניינות (ולא תבניות שנראות כמו רעש מוחלט).

לצערי - לא רק שלא ראיתי שיפור, אלא הרשת נתקעה בבערך 17.15% דיוק (על סט הוולידציה). כאן נזכרתי בנקודה שלמדתי בקורס "מבוא ללמידה חישובית": עבור קלטים שליליים, הגרדיאנט של פונקציית אקטיבציה ReLU הוא 0 - דבר שיכול לעצור לגמרי את ההתקדמות ע"י אלגוריתם gradient descent. מכיוון שרגולריזציית weight-decay מנסה להקטין את הערכים המוחלטים של המשקולות, הגיוני שהיא תוביל למצב הזה! למדנו שם, שכדי להתגבר על הבעיה, אפשר להשתמש בפונקציית אקטיבציה Leaky ReLU, שלא "הורגת" לגמרי את הגרדיאנט עבור קלטים שליליים. החלטתי להשתמש בה לכל אורך הרשת, ושיחקתי עם הפרמטרים שלה (ועם הפרמטרים של הרגולריזציה) בניסיון "לפרוץ" את מחסום 17.15% הדיוק.

הפעם - הייתה הצלחה מדהימה! לא רק שהביצועים השתפרו, אלא ממש היה אפשר לראות צורות מסוימות בפילטרים שנלמדו בשכבה הראשונה. אף אחד מהם לא היה נראה כמו רעש מוחלט. הנה כמה מהם:



טכניקה נוספת שראוי לציין שהשתמשתי בה היא **learning-rate decay** - טכניקה שמקטינה את קצב הלמידה כאשר המודל מגיע ל"פסגה" מסויימת בביצועים שלו, כדי לאפשר צעדים קטנים יותר של למידה אל עבר האופטימום. בטכניקה זו החלטתי להשתמש בעצתו של אחד החברים שלי, תלמיד תואר שני במדעי המחשב במכון ויצמן. היא הובילה לתוצאות טובות יותר - לכן גם היא נכנסה למודל הסופי.

אחרי המון ניסויים, ניסיונות, ומשחק עם המבנה של השכבות השונות, הגעתי לארכיטקטורה הבאה בה אני אשתמש בהמשך לסיווג האותיות:

```
_________________________________________________________________
# Layer (type)                 Output Shape              Param
=================================================================
randomcolordistortion       (None, 64, 64, 3)         0
(RandomColorDistortion)

conv2d_3 (Conv2D)           (None, 64, 64, 64)        15616

max_pooling2d_3             (None, 32, 32, 64)        0
(MaxPooling 2D)

conv2d_4 (Conv2D)           (None, 32, 32, 32)        73760

max_pooling2d_4             (None, 16, 16, 32)        0
(MaxPooling 2D)

conv2d_5 (Conv2D)           (None, 14, 14, 64)        18496

max_pooling2d_5             (None, 7, 7, 64)          0
(MaxPooling 2D)

dropout_3 (Dropout)         (None, 7, 7, 64)          0

flatten_1 (Flatten)         (None, 3136)              0

dense_2 (Dense)             (None, 128)               401536

dropout_4 (Dropout)         (None, 128)               0

dense_3 (Dense)             (None, 128)               16512

dropout_5 (Dropout)         (None, 128)               0

dense_4 (Dense)             (None, 7)                 903

=================================================================
Total params: 526,823
Trainable params: 526,823
Non-trainable params: 0
_________________________________________________________________
```

השלב הבא הוא לבנות ולאמן רשתות נוירונים לכל תו. ואילו, הגישה הזו מביאה איתה כמה בעיות. הבעיה הראשונה, והברורה ביותר, היא מחסור בנתוני אימון לכל תו. אמנם - יש לנו המון דוגמאות עבור תווים שכיחים כמו n, אבל מה נעשה עם תווים שיש לנו דוגמאות בודדות מהן - כמו התווים + או /? יתרה מזו - מה יקרה אם במהלך בחינת המערכת (אחרי האימון), נתקל בתו שאנחנו בכלל לא ראינו בסט האימון? הרי לא יהיה לו מודל!
הדרך הטריוויאלית שעלתה לי להתמודדות עם הבעיה היא פשוט להרחיב את סט הנתונים: ליצור המון דוגמאות של התווים הפחות שכיחים (או כאלו שבכלל לא ראינו בסט האימון). ואילו, לא התעמקתי הרבה במימוש הדרך הזו, כי עלתה לי לראש דרך הרבה יותר מעניינת עם הרבה יותר פוטנציאל.

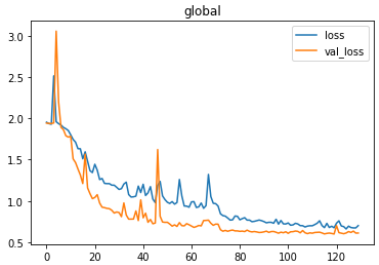
הדרך החדשה להתמודדות עם הבעיה הייתה ליצור **רשת נוירונים "גלובלית"** (להלן "המודל הגלובלי") - כזו שאנחנו נאמן על **כל הדוגמאות** בסט האימון, ולא רק על דוגמאות של תו מסויים (לרשתות שאנחנו מאמנים על דוגמאות של תו ספציפי נקרא להלן "מודלים אינדיבידואלים"). נוכל להשתמש בה לסווג תווים שלא ראינו

בסט האימון, או כאלו שלא היו לנו מספיק דוגמאות שלהם בסט האימון. הרעיון הזה הוביל אותי, עם ההתנסות והמימוש, לכמה שיפורים מעניינים שהקפיצו את הביצועים:

- שיפור ראשון הגיע מהאינטואיציה לפיה בתהליך האימון של המודל הגלובלי אנחנו לומדים המון פיצ'רים חשובים, שיכולים לשמש גם את המודלים האינדיבידואלים. מכאן עלה לי הרעיון לערוך את אימון המודלים האינדיבידואלים ע"י **fine-tuning של המודל הגלובלי**, במקום אימון מודל חדש "מאפס". כלומר - בתחילת האימון של כל מודל אינדיבידואלי, המודל יהיה העתק של המודל הגלובלי (עם כל הפרמטרים שנלמדו בתהליך האימון של המודל הגלובלי). מכאן - נריץ עוד כמה איטרציות של תהליך הלמידה, עם סט אימון המכיל דוגמאות של האות הספציפית בלבד ("fine-tuning" למקרה ספציפי של אות מסוימת).
- כאשר אימנתי את המודלים האינדיבידואלים על דוגמאות של תו ספציפי (הרצתי את תהליך ה-fine-tuning), הבחנתי בכך שלפעמים ערך ה-validation-loss דווקא עולה - כלומר מקרים בהם המודל האינדיבידואלי מוביל לביצועים פחות טובים מהמודל הגלובלי (על אותה אות). הדרך עליה חשבתי כדי "לסנן" את המקרים הללו בצורה אוטומטית היא **לשמור רק מודלים אינדיבידואלים שהראו שיפור ניכר ב-loss** על התו הספציפי, לעומת המודל הגלובלי. ככה - במהלך בחינת המערכת, נבחין שלא קיים מודל אינדיבידואלי עבור התו הזה, ונסווג בעזרת המודל הגלובלי במקום (שצפוי להניב תוצאות טובות יותר).
- השתמשתי ביוריסטיקה נוספת כדי לסנן מקרים שלא צפויים להניב תוצאות טובות עוד לפני ה-fine-tuning: מקרים בהם יש לנו פחות מ-30 דוגמאות אימון של תו מסויים. היא נועדה בעיקר כדי לזרז את תהליך האימון, ולא פגעה כלל בתוצאות.

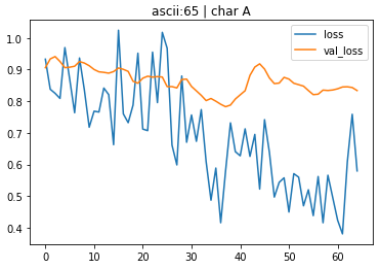
ניסוי ותהייה הובילו אותי לבחור ב-130 epochs עבור אימון המודל הגלובלי, ו-65 epochs עבור ה-fine-tuning של כל מודל אינדיבידואלי.

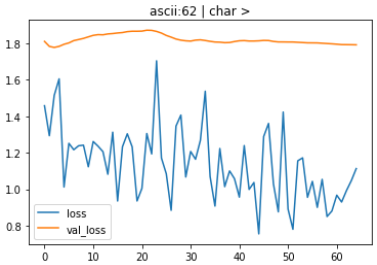
אופי האימון גם הוא היה מעניין.
קודם כל - הייתי צריך לאמן מודל גלובלי. תהליך האימון (loss כתלות ב-epoch) מופיע משמאל. ניתן לראות שעם הזמן הקפיצות ה"רועשות" בתהליך האימון לאט לאט הופכות לעדינות יותר - עקב הקטנת ה-learning rate בהגעה ל"פסגות" מסוימות.
אגב - תופעה מעניינת היא שלאורך כל התהליך, שגיאת הולידציה נמוכה יותר משגיאת האימון, בניגוד למה שהיינו מצפים. זה בגלל האוגמנטציה - נתוני האימון עוברים תהליכי עיוות שהופכים אותם לקשים יותר לסיווג (מאשר נתוני הוולידציה שנשארים "נקיים").

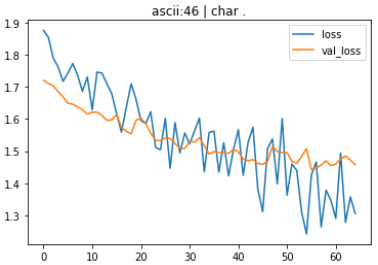


תהליך ה-fine-tuning היה שונה מאות לאות:

- לפעמים הוא שיפר את הביצועים משמעותית - כמו עבור התו [.] (ראה/י בגרף הימיני למטה).
- לפעמים הוא בקושי שינה משהו - כמו עבור התו > (ראה/י בגרף האמצעי למטה).
- ולפעמים הוא השתפר, אבל גם יצר overfitting - כמו עבור התו A (ראה/י בגרף השמאלי למטה).







אחרי שאימנתי את המודלים, הייתי צריך לשמור אותם במקום מסויים. חשבתי להשתמש גם פה בקובץ אחד בפורמט HDF5 לכל המודלים, אבל לא מצאתי דרך בה ניתן לשמור כמה מודלים באותו קובץ HDF5. לכן -

החלטתי לשמור אותם בתיקייה "models" במערכת הקבצים, כל מודל בתת-תיקייה אחרת (וגם פה השם של התיקייה בה נמצא מודל אינדיבידואלי לתו מסוים הוא ערך ה-ASCII של התו ולא התו עצמו, מכיוון שאי אפשר ליצור תיקייה ששמה ".").

על כל התהליך שפורט לעיל אחראית התוכנית training.py: התכנית מכילה את המימוש של השכבה המותאמת אישית לעיוות הצבע של התמונות - RandomColorDistortion, פונקציה createModel ליצירת מודלים מהארכיטקטורה שבחרתי. בנוסף, הקובץ מכיל תכנית ראשית שאחראית על אימון המודלים, הערכת הביצועים שלהם על סט הוולידציה, ושמירתם בתיקייה models.
יש לשים לב לעניין חשוב: התוכנית הראשית לא שומרת רק את המודלים, אלא היא גם יוצרת קובץ weights.h5 בתיקייה models. קובץ זה מכיל datasets ששמם זהה לשמות המודלים שנשמרו, והם מכילים ערך יחיד: **משקל** שהותאם לכל מודל. המשקל הזה רלוונטי לחלק האחרון - הסיווג. האופן בו הוא מחושב והמשמעות שלו, יפורטו היטב בהמשך.

הפרטים הקטנים בתוכנית training.py מפורטים היטב בקובץ עצמו, ויהיה מיותר (וארוך) לחזור עליהם פה.

בזאת מסתיים תהליך בניית המודלים.

<u>חלק שלישי</u>: סיווג

בחלק האחרון נצטרך להשתמש במודלים שיצרנו כדי לסווג תווים בקלט לפי הפונט שלהם. הפלט של כל מודל על תמונת קלט מסויימת יהיה וקטור הסתברויות בעל 7 רכיבים, עבורו הרכיב ה-i (כאשר i=0,...,6) מהווה את ה"הסתברות" שהקלט מכיל תו מהפונט ה-i ("לדעתו" של המודל).

לכל תו נבחר מודל מבין המודלים שיצרנו: אם קיים מודל אינדיבידואלי שעבר fine-tuning מוצלח לסיווג תווים מסוג הזה - נבחר בו, אחרת - נבחר במודל הגלובלי (שאמור להניב תוצאות טובות יותר עבור התו).

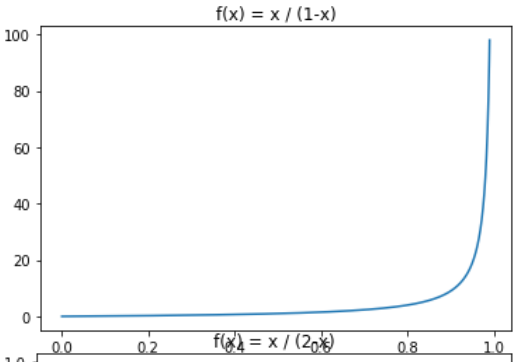
אחרי שבחרנו את המודל המתאים להערכת הפלט לתמונת קלט מסויימת, נוכל (בנאיביות) לבחור לסווג את הקלט עבור הפונט שקיבל את ה"הסתברות" הגבוהה ביותר בוקטור ההסתברויות בפלט. ואילו, נוכל לנצל את העובדה שהתווים באותה המילה שייכים לאותו פונט, ולנסות לבחור את הפונט שיש סביבו הכי הרבה "קונצנזוס" בין התווים במילה, וככה להיות יותר בטוחים בזהות הפונט.

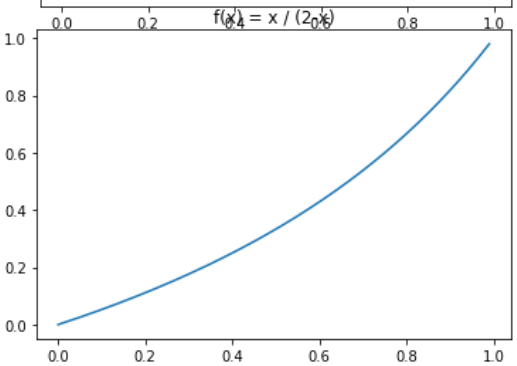
בניסיון להביא את האינטואיציה הזו לידי ביטוי, חשבתי על דרכים שונות לבחור את הפונט אליו שייכים התווים במילה מסויימת - כל אחת מתוחכמת יותר מקודמתה:

- הרעיון הראשון (והפשוט ביותר) היה לבדוק את הסיווג של כל תו במילה בנפרד, ולבחור את הפונט אליו סווגו הכי הרבה תווים.
- התבוננתי קצת בפלטים של המודלים על קלטים שונים, והבחנתי בתופעה הבאה: כאשר יש טעות בסיווג, כלומר ההסתברות לפונט שגוי גבוהה מההסתברות לפונט האמיתי, **ההסתברות לפונט האמיתי גם היא יחסית גבוהה**. למה שנתעלם מהמידע הזה ופשוט נבחר את הפונט עם ההסתברות הגבוהה ביותר לכל תו בנפרד, ורק בסוף נשקלל בין הסיווגים הסופיים? האינטואיציה הזו הובילה אותי לחשוב על רעיון אחר, רעיון שישקלל את "מידת האמונה" של הפונטים השונים בין התווים השונים במילה, ולא רק את הסיווג הסופי שלהם. הרעיון הוא **לסכום את וקטורי ההסתברות השונים** של כל התווים באותה מילה.
- אבל אז חשבתי לעצמי - למה שלתווים "קשים לסיווג" תהיה אותה השפעה על שקלול ההסתברויות כמו לתו "קל לסיווג"? בעקבות האינטואיציה הזו החלטתי **להוסיף משקל לכל מודל, בהתאם לדיוק שלו**. כל וקטור הסתברויות שחושב כפלט של מודל מסוים, יוכפל במשקל המתאים למודל, ורק אז נסכום את התוצאות. אחרי השקלול הזה - פשוט נבחר את הפונט המתאים לרכיב בעל הערך הגבוה ביותר בוקטור המשוקלל.

בעיה אחרונה היא בחירת המשקלים לכל מודל. המשקל צריך לבטא את מידת ה"ביטחון" שלנו בפלט שהמודל מפיק. דרך הגיונית להביא את זה לידי ביטוי היא לחשב את המשקל כפונקציה של **הדיוק** של המודל על סט הולידציה. לפי האינטואיציה הראשונית שלי, חשבתי על מספר "תנאים" שכדאי שהפונקציה צריכה לקיים:

1. היא צריכה להיות אי שלילית בתחום [0,1] (טריוויאלי).
2. היא צריכה להיות מונוטונית עולה: ככל שהדיוק של המודל גדול יותר, אנחנו יותר בטוחים בו.
3. היא צריכה לשאוף ל-0 ככל שהדיוק שואף ל-0: מודל שתמיד טועה לא מועיל לנו בכלל.
4. היא צריכה לשאוף לאינסוף ככל שהדיוק שואף ל-1: אנחנו לגמרי בטוחים במודל שתמיד צודק.





הפונקציה הראשונה שעלתה על הקריטריונים האלו היא $f(x) = x / (1 - x)$, אשר מופיעה משמאל. שמתי לב שהיא יוצרת יתרון גדול **מדי** עבור מודלים טובים מאוד: מודל מצויין עם 95% דיוק יקבל משקל 19, בעוד מודל "לא-רע" עם 75% דיוק יקבל משקל 3 - וכנראה יהיה חסר משמעות ביחס למודל המצויין. אנחנו לא רוצים שזה יהיה המקרה - אנחנו רוצים שגם למודלים "לא-רעים" תהיה חשיבות בהחלטה הסופית.

בעקבות כך, התחלתי לשחק עם הפונקציה במטרה למצוא פונקציה טובה יותר. לבסוף - החלטתי להמשיך עם הפונקציה $f(x) = x / (2 - x)$, המופיעה משמאל. היא קרובה מאוד לפונקציה לינארית פשוטה, אבל נותנת קצת יותר משמעות לדיוק גבוה.

ניסיתי להכניס פרמטרים אחרים לקביעת המשקל - כמו, למשל, מספר הדוגמאות עליהן אומן המודל (הרי ככל שהוא אומן על יותר דוגמאות, אנחנו יותר בטוחים בו). אבל לא קיבלתי שיפור משמעותי בתוצאות, אז השארתי את זה ככה.

לחישוב המשקלים אחראית התוכנית training.py עליה פירטתי בחלק הקודם. היא מחשבת את המשקלים בעזרת הנוסחה $Weight = Accuracy / (2 - Accuracy)$, ושומרת אותם בקובץ weights.h5 בתיקייה models. הקובץ מכיל datasets ששמם זהה לשמות המודלים. כל dataset מכיל ערך אחד - המשקל של המודל בעל אותו השם.

וזהו! האלגוריתם של מערכת הסיווג הושלם! הפונקציות evaluate ו-predict בקובץ classifying.py אחראיות על מימושו. התוכנית הראשית בקובץ הזה מעריכה את הפלט של מערכת הסיווג על קובץ קלט SynthText_test.h5 (בעל הפורמט של קובץ האימון SynthText.py רק ללא מידע על פונטים), ויוצרת קובץ פלט results.csv המכיל מידע על הפונטים אליהם סווגו התווים השונים בתמונות שונות. הממשק שמציע הקובץ classifying.py דרך הפונקציות evaluate ו-predict והפרטים הקטנים בתוכנית עצמה מפורטים היטב בקובץ עצמו, ויהיה מיותר (וארוך) לחזור עליהם פה.

אציין רק שעקב היותו של סט הבחינה (החיצוני) גדול מאוד, התוכנית הראשית בקובץ הזה מפרקת את התמונות בו ל-batches, ומטפלת בכל batch בנפרד.

תוצאות

בקובץ experimenting.py ערכתי מספר ניסויים על תת-סט הבחינה שיצרנו בחלק הראשון של הפרויקט (ע"י פיצול הנתונים בקובץ SynthText.h5 לאימון, ולידציה, ובחינה). ניסויים אלו נועדו לקבל אבחנה סופית על יכולות מערכת הסיווג, ותוצאותיהם מפורטות להלן.

בס"ד

אחוזי דיוק:

- דיוק על סט הבחינה: 95.02%.
- דיוק על סט האימון: 99.90%.

ניתן להעריך שקיבלנו מצב עם מעט overfitting, אף על פי שאחרי אוגמנטציה ראינו שהמודל עובד פחות טוב לנתוני האימון. עם זאת - הגענו לתוצאה גבוהה מאוד על סט הבחינה שלא לקח חלק כלל וכלל בתהליך ההתאמה, ואני מסופק ממנה.

רגישות מערכת הסיווג למחלקות הפונטים השונות:

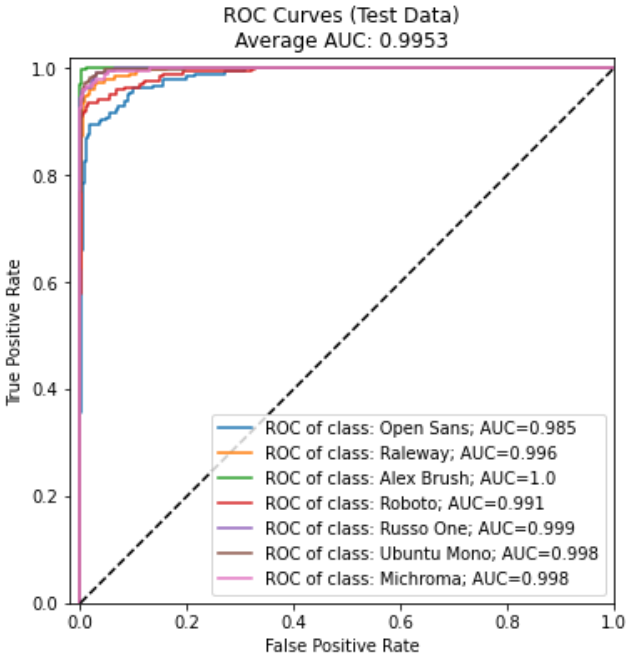
Font	Alex Brush	Ubuntu Mono	Raleway	**Russo One**	Michroma	Roboto	Open Sans
Sensitivity	99.64%	97.17%	95.64%	95.61%	94.12%	93.54	87.78

את הביצועים הטובים ביותר קיבלנו עבור מחלקת Alex Brush. תוצאה זו לא מפתיעה כל כך - לפונט הזה יש צורה מאוד ייחודית, שקל להבחין בינה לבין צורת התווים בשאר הפונטים.
אחריה, להפתעתי, הגיעה מחלקת Ubuntu Mono. לי אישית קשה להבחין במבנה מיוחד לפונט הזה, והופתעתי שהוא עקף פונטים "קלים" יותר (לדעתי) כמו Michroma או Russo One.
אחריו, מופיעים Raleway, Russo One, ו-Michroma, בהפרשים לא גדולים אחד מהשני. אני אישית מצליח להבחין במבנה מסוים לתווים במחלקות האלו, אמנם לא מבנה ברור כמו של Alex Brush. לפיכך, המיקום שלהם נראה לי הגיוני ולא כל כך מפתיע.
מחלקות Open Sans ו-Roboto אלו המחלקות הכי פחות מוצלחות. סביר לשער מדוע - הצורה של התווים בהם היא די "גנרית" - קשה להבחין בייחודיות מסויימת במבנה שלהם, והם עצמם מאוד דומים אחד לשני.

טבלת Confusion Matrix: מתוך 4564 דוגמאות של תווים בסט הבחינה:

Actual **Predicted**	Open Sans	Raleway	Alex Brush	Roboto	**Russo One**	Ubuntu Mono	Michroma
Open Sans	**632**	7	0	23	13	16	12
Raleway	28	**899**	0	17	0	3	4
Alex Brush	0	7	**838**	0	0	0	6
Roboto	21	13	0	**753**	0	7	6
Russo One	6	3	0	4	**719**	0	3
Ubuntu Mono	30	11	3	8	20	**893**	0
Michroma	3	0	0	0	0	0	**496**

כפי שראוי לצפות - יש בלבול רב בין Open Sans ו-Roboto. יש גם בלבול באופן כללי בין המחלקות Open Sans, Raleway, Roboto, Ubuntu Mono, Russo One. סביר לצפות את זה - הפונטים האלו דומים באופן כללי, לכולם יש מעין מבנה "מרובע".
התוים של Michroma סווגו להמון מחלקות שונות שאינן Michroma, ואילו הרבה מאוד תווים שסווגו כ-Michroma באמת היו שייכים ל-Michroma. מה שאומר שיש ל-Michroma שיעור False Negative גבוה, אבל שיעור False Positive נמוך - אם קלט סווג ל-Michroma אפשר להסיק ביותר ביטחון שהוא שייך ל-Michroma, אבל לא ההפך.
כמובן - Alex Brush עם ביצועים מאוד טובים, גם בשיעור ה-True Positive וגם בשיעור ה-True Negative. קל להבחין אם קלט שייך או לא שייך למחלקה הייחודית הזו.



<u>עקומות ROC</u>: מופיעות משמאל.

העקומות מתקרבות מאוד לעקומה אופטימלית (חוץ ממחלקת Open Sans, וכבר ראינו שהסיווג שלה קשה יותר).
אין הרבה מה להסיק מהן - ראינו כבר אילו מחלקות מוצלחות יותר ואילו פחות.

איך ליצור פלט עבור סט מבחן

הוראות מפורטות על איך ליצור פלט עבור סט מבחן חיצוני נמצאות בקובץ README.txt, המצורף.

רפלקציה אישית

העבודה על הפרויקט הייתה מאוד מעניינת. הייתי צריך להתמודד עם בעיות שונות, לקבל המון החלטות, להפעיל כישורי יצירתיות טובים ולהשקיע המון מחשבה. למען האמת - לא הרגשתי שזה יהיה ככה בהתחלה, ושמחתי לגלות את העניין הרב שיש בעבודה שכזו.
בפעם הראשונה בחיי התנסיתי בפרויקט הנדסי בראייה ממוחשבת. מאוד נהניתי מכל האספקטים השונים שלו, ואני מרגיש שארצה לעשות עוד כאלו בהמשך הדרך שלי.
למדתי המון טכנולוגיות, טכניקות, וממשקים נוחים לעבודה. בין היתר - נכנסתי לעומק לממשק של ספריית tensorflow ומימשתי מחלקה מותאמת אישית. בנוסף, למדתי מאפס על קבצי HDF5 והממשק של ספריית h5py. שניהם היו נוחים מאוד, שמחתי שיצא לי להכיר אותם מקרוב, ואני מאמין שאשתמש בהם בהמשך.
בסך הכל - הרגשתי שהצלחתי להפיק "מוצר" מוצלח. שמחתי לראות איך השיטות המקוריות שחשבתי עליהן "הקפיצו" את הביצועים, והכל הסתכם במערכת סיווג מוצלחת :)

בהצלחה ^_^

רשימה ביבליוגרפית

[1] Gupta Ankush, Andrea Vedaldi, and Andrew Zisserman. "Synthetic data for text localisation in natural images." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.