ראייה ממוחשבת – פרויקט

עמית בסקין 312259013

1 איך התכוננתי לפרויקט

עשיתי Specialization של deeplearning ב־ coursera הכולל 5 קורסים (קישור מספר 16 בסעיף הקישורים בתחתית המסמך), מתוכם הספקתי לעשות 4 לפני שקיבלנו את הפרויקט ובמסגרתם עשיתי את כל התרגילים הכלולים. צירפתי בקישורים בתחתית המסמך את תעודות הסיום שקיבלתי.

2 עיבוד הדאטאסט

בתהליך של חיתוך האותיות ניכר שחלק גדול מהקואורדינטות של התיחומים של האותיות, אינן מדוייקות. בחלק גדול מהמקרים התמונה שנחתכת אינה מכילה טקסט כלל. אחד מהדפוסים הבולטים של הבעיה הזאת היה חיתוכים במימדים קטנים מדי שלא מאפשרים הצגה של שום תוכן שהוא. מבדיקה מדגמית בעת הרצת החיתוכים היה הדפסתי את כל התמונות שהיו במימדים קטנים מ־ 20×20 וגיליתי שאכן תמונות חיתוכים בגודל כזה מהוות טעויות בתיחומים ומציגות תמונות ריקות מטקסט. בשביל תהליך האימון חשבתי לנסות לסנן את התמונות הללו ולהתאמן רק על תמונות "מדוייקות", אך בניסיון הזה לסנן תמונות במימדים קטנים או כאלה שבכלל כללו קואורדינטות שליליות או כאלה שלא בתחומי התמונה, התברר שהסינון גורם לאיבוד של כשליש מהתמונות.

מכאן עלה לי לרעיון נוסף שהוא הגדלת החיתוך לגודל מינימלי כלשהו שהגדרתי בניסוי ותעייה (בסופו של דבר הגדרתי 30×30), במידה והגודל הנתון הינו קטן מכך, וזאת בתקווה שה"טעות" בתיחום לפחות נעשתה באזור המיקום ה"אמיתי" של האות. מכאן עלה רעיון נוסף להגדיל את התיחום באופן כללי לכל אות לכדי שוליים שמהווים רבע ממימדי החיתוך הנתון. בצורה הזאת מתקבלים חיתוכים שמכילים את האות הרצויה יחד עם האותיות הסמוכות לה. עתה, מאחר שנתון לנו שהאותיות באותה המילה הן מאותו הפונט, דבר זה לכאורה לא אמור להזיק לזיהוי הפונט במילה ואף אולי עשוי

להואיל. הרציונאל הוא שתמונה שמכילה יותר מאות אחת כביכול מכילה יותר מידע על הפונט הנתון מאשר תמונה שמכילה אות אחת בלבד.

באותה המידה היה לי רעיון בכלל להפוך את המשימה לזיהוי פונט במילה בניגוד לזיהוי פונט באות. אך החיתוך של המילים התברר אף כפחות מדוייק מהחיתוך של האותיות ובנסיונות הלמידה שאתאר בהמשך המסמך, כאשר ניסיתי את הלמידה על מילים, קיבלתי תוצאות טובות פחות, מה גם שחיתוך מילים משאיר הרבה פחות תמונות ללמוד מהן.

שאלה נוספת שעלתה בהקשר של חיתוך אותיות היה חיתוך של מרובע מדוייק סביב המטרה או שמא עדיף לחתוך מלבן גרידא. ההבדל הוא שחיתוך של מרובע משאיר פחות רקע סביב המטרה שנחתכת ואילו המלבן מכיל את המטרה עצמה ואת הרגע שסביבה שמשלים למלבן תוחם. האינטואיציה בצידוד בעד מרובע חוסם היא שהתמונה יותר ממוקדת במטרה שאותה רוצים ללמוד, שהרי ההשלמה למלבן של תמונה נעשתה באמצעות רקע שחור שכביכול יותר ברור להתעלם ממנו. מהניסיונות שביצעתי לפי הכיוונים שאפרט בהמשך התברר שהשימוש במלבן חוסם לעומת מרובע כללי חוסם נתן תוצאה טובה יותר, היה ניתן להיווכח בכך בהפרשים משמעותיים בדיוק כבר בשלב מוקדם של האימון.

אם כן, בסופו של דבר החלטתי לנקוט בחיתוך של אותיות וסביבתן כך שבפועל לכל מילה מתקבל אוסף תמונות של חלקים ממנה. דבר זה השתלב היטב עם הכיוון שאתאר בסעיף 11 של הרחבת הדאטאסט.

3 Text Recognition

בשביל לפתור את הבעיה בחוסר הדיוק של תיחום האותיות ניסיתי להשתמש במודל קיים של Text Recognition (קישור מספר 15 בסעיף הקישורים בתחתית המסמך) בשאיפה שכך אוכל גם למצוא את הטקסט בתמונה ולתחם אותו באופן מדוייק, וגם לדעת מהו הטקסט שנמצא על מנת שאוכל להתאים לתיחום את הפונט שנתון לי.

שוב, שלא יותר מדי במפתיע, הניסיון לא עבד טוב במיוחד:

ניסיון זה מתועד בקובץ text_recognition_try.py.

4 בדיקת דיוק

נקודה חשובה שהבנתי לגבי בדיקה של דיוק המודל: בהפרדה לאימון, ואלידציה ומבחן יש קודם כול לחלק את התמונות עצמן לפי הקבוצות הללו ורק אז לחלץ את האותיות מהן ולשים בקבוצה המתאימה. הנקודה היא שאם מחלצים את כל האותיות ורק אז מחלקים לקבוצות אז יהיו אותיות מאותה המילה בשלוש הקבוצות מה שבעצם "מזהם" את הבדיקה, שהרי יש קשר בין האותיות באותה המילה ואם המודל מתאמן על חלק מהאותיות במילה ונבחן על חלק אחר בה אזי שהדיוק שלו לא יהיה אמין.

5 ResNet50

היתרון ברשתות עמוקות הוא שהן מסוגלות ללמוד פונקציות מורכבות. בתור אמירה רחבה ניתן לומר שככל שהרשת עמוקה יותר כך היא מסוגלת ללמוד פונקציות מורכבות יותר, אף כי מובן שהמורכבות הנלמדת תלויה בארכיטקטורה של השכבות.

הבעיה ברשתות עמוקות הוא ה־ Vanishing Gradient. הרי, כזכור, החישוב של הגרדיאנט לשם עדכון המשקלים ברשת, נעשה על ידי Backpropagation עם כלל הנגזרת, קרי מכפלה של נגזרות. אם כן, ככל שהרשת עמוקה יותר כך המכפלה קטנה אקספוננציאלית לאפס או גדלה אקספוננציאלית לערכים גדולים למדי. מכאן שככל שהשכבה מוקדמת יותר ברשת, כך הגרדיאנט של המשקלים שלה מתאפסים או "מתפוצצים", והלמידה של המשקלים משתבשת/נעצרת.

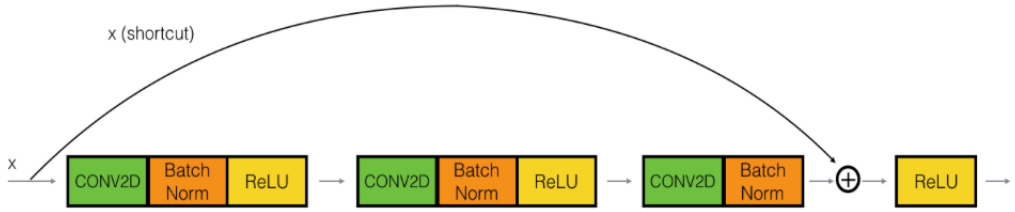
כאן ה־ ResNet באה לעזור: על מנת שהשכבות המוקדמות לא "יאבדו משמעות" בשל ה־ Vanishing Gradient שגורם

לשיבוש למידת המשקלים שלהן, ב־ ResNet, הפלט של שכבה מוקדמת "מדלג" לשכבה מאוחרת יותר. הדילוגים הללו נמשכים לאורך הרשת וכך השכבות המוקדמות מקבלות ביטוי וכן "נלמדות" בשכבות העמוקות יותר.

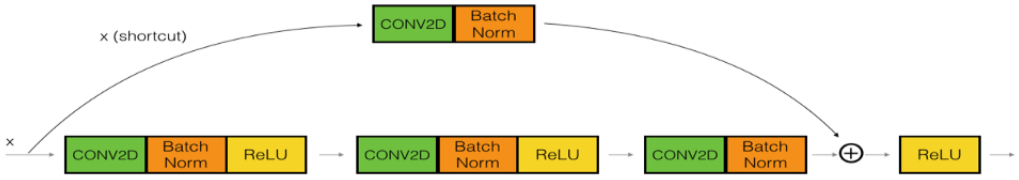
הדילוג נעשה בשתי וריאציות: אחת נקראת Identity Block והשנייה נקראת Convolutional Block.

ה־ Identity Block משמש כאשר מימדי הפלט של השכבה המדלגת, זהים למימדי הפלט של השכבה שמדלגים אליה, וה־ Convolutional Block משמש כאשר המימדים שונים (הדילוג נעשה עם קונבולוציה בשביל להתאים בין המימדים). להלן המחשה של שתי הוריאציות:

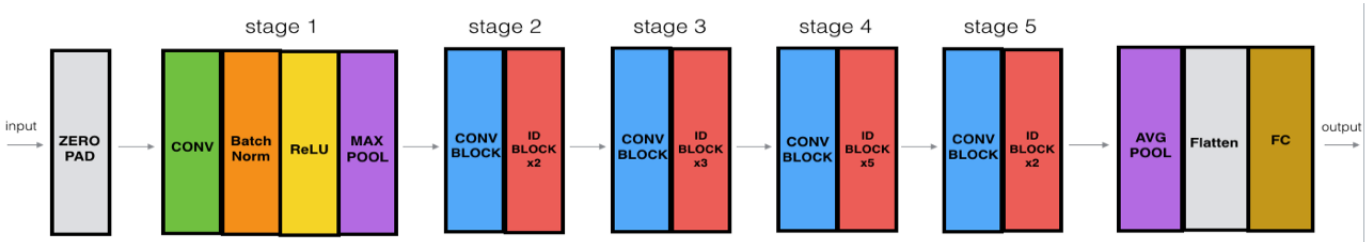
ה־ Identity Block:



ה־ Convolutional Block:



הרשת ResNet50 כשמה כן היא מכילה 50 שכבות. להלן תיאור הארכיטקטורה שלה:



בהנחיית הקורס Convolutional Neural Networks של deeplearning.ai שעשיתי ב־ coursera (קישור מספר 6 בסעיף הקישורים בתחתית המסמך), בניתי את ה־ ResNet50 בעצמי, שכבה אחר שכבה, בלוק אחר בלוק, ואימנתי אותה מאפס. המימוש שכתבתי נמצא בקובץ resnet.py.

במהלך האימון, הדיוק של הואלידציה התקבע באזור ה־ 40% בעוד הדיוק של האימון המשיך לטפס. בשביל לנסות למנוע את ה־ overfit ולשפר את ההכללה של הרשת, ניסיתי להשתמש באוגמנטציה רנדומלית של סט האימון. האוגמנטציה שהשתמשתי בה כללה פעולות רנדומליות כגון היפוך, סיבוב, זום, הזזה וניגודיות. כמו כן ניסיתי להוסיף שכבות של Dropout בין השכבות בתוך הבלוקים ובין הבלוקים עצמם. מה ששכבת ה־ Dropout עושה זה לבטל את ההשפעה של אחוז מסויים של נוירונים (לפי השיעור שנקבע) וכך הרשת פחות "מסתמכת" על נוירונים ספיציפיים בשכבה שהופעל עליה ה־ Dropout. גם זאת עשוי לעזור לרשת ללמוד הכללה טובה יותר ולמנוע את ה־ overfitting. השיעורים של ה־ dropout שניסיתי נעו בין 0% ל־ 60%.

למרבה הצער, כל הנסיונות הללו לא צלחו. הם אמנם האטו את ה־ overfitting אך לא מנעו אותו, והדיוק על הואלידציה נשאר פחות או יותר באותו האזור.

6 FaceNet

כיוון נוסף שניסיתי היה בהשראת האתגר להתאמה בין פרצופים של אנשים. מודל ידוע במומחיותו במשימה הנ"ל הינו מודל ה־ FaceNet (קישור 13 בסעיף הקישורים בתחתית המסמך). הרעיון שהמודל הזה מבוסס עליו נקרא ה־ Triplet Loss Function. ה־ Triplet Loss כשמו כן הוא משתמש בשלושה משתנים לחישוב "הפסד": בהינתן שלוש תמונות של פרצופים – תמונה אחת שמייצגת את האדם שאנו רוצים לזהות, תמונה זו נקראת ה־ Anchor, תמונה שנייה של אותו אדם שאנחנו רוצים לזהות אך שונה מה־ Anchor, תמונה זו נקראת ה־ Positive, ותמונה שלישית של אדם אחר שאינו האדם שאנו רוצים לזהות, תמונה זו נקראת ה־ Negative. בהינתן שלוש תמונות כנ"ל אנחנו רוצים לאמן רשת לפלוט וקטור "ייצוג" של פרצוף כך שה"מרחק" בין ה־ Anchor ל־ Positive יהיה קטן משמעותית מהמרחק בין ה־ Anchor ל־ Negative. אם כן, בשביל להגדיר את ההצלחה של "ייצוג" כנ"ל, אנו מגדירים את ה־ Triplet Loss Function שמודד את המרחקים הנ"ל ואת היחס ביניהם ונותן ציון בהתאם. וזה למעשה ה־ Loss Function עבור הרשת FaceNet שלמדה את הפונקציה לייצוג של פרצוף שמאפשרת להשוות בין שני פרצופים ולפלוט אם מדובר באותו אדם.

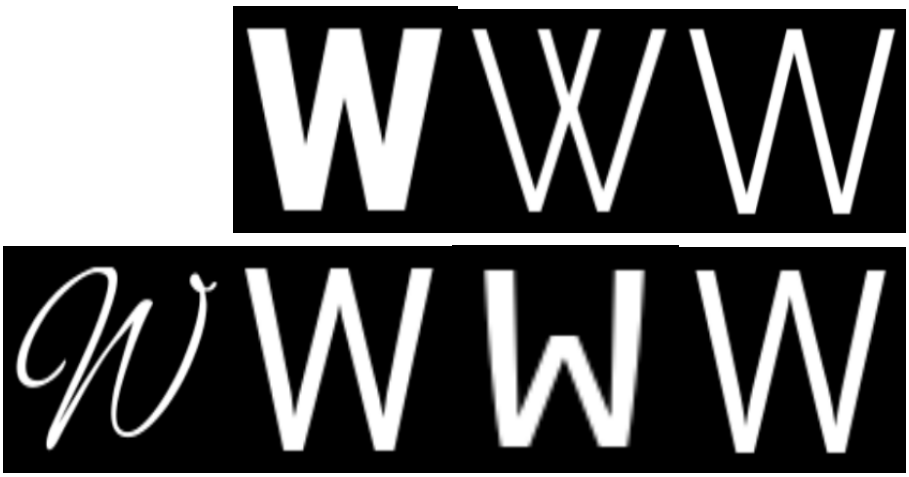
אם כן, באופן אידאלי, בשביל ליישם את הרעיון של FaceNet על הדאטאסט של הפרויקט, הייתי צריך ליצור דאטאסט נוסף של שלשות כנ"ל בשביל לאמן רשת שתלמד את ייצוג מתאים עבור אותיות במקום פרצופים. אך בשל מורכבות המשימה והקוצר בזמן, החלטתי לנסות את הרשת הקיימת של FaceNet בשביל לבדוק אם במקרה הייצוג שנלמד עבור פרצופים יתאים גם לאותיות בדאטאסט של הפרויקט.

לשם כך בהינתן אות, יצרתי ייצוג "סטנדרטי" עבורה לפי כל אחד משבעת הפונטים שנדרשנו ללמוד, כאשר הכוונה

בייצוג סטנדרטי היא תמונה שחורה עם האות כתובה בלבן ומתפרסת על פני התמונה כולה בצורה ברורה ללא כל עיוות.

לדוגמה עבור האות W: בהינתן תמונה של W מהדאטא סט:

יצרתי את התמונות הבאות:



לאחר מכן, יצרתי וקטור ייצוג עבור כל אחת מהתמונות באמצעות המודל של FaceNet, השוויתי בין הייצוגים הללו, והפונט עם הייצוג הכי קרוב לתמונת המטרה הוא הפונט הנבחר.

לאחר הבדיקה של הכיוון הזה, באופן לא כל כך מפתיע כנראה, התוצאות היו רנדומליות למדי. ניסיתי זאת עבור מספר תמונות שונות, וכמעט שלא היה הבדל ניכר במידת ההתאמה של כל אחד מהייצוגים.

יש להניח שהייצוג שמתאים לפרצופים אינו מתאים לייצוג של פונט של אותיות. הניסיון שתיארתי לעיל מתועד בקובץ facenet_try.py. כמו כן, במסגרת הלמידה שביצעתי בקורס Convolutional Neural Networks של deeplearning.ai ב־ coursera, שם למדתי על ה־ FaceNet, מימשתי בין השאר בתור תרגיל גם את פונקציית ה־ Loss שתיארתי, וצירפתי אותה לאותו הקובץ.

7 Sift

ניסיון נוסף שערכתי היה עם פונקציית ה־ Sift שלמדנו בכיתה (קישור 14 בסעיף הקישורים בתחתית המסמך). הרעיון שניסיתי די דומה לרעיון שתיארתי לעיל בניסיון של ה־ FaceNet: בהינתן תמונת מטרה, יצרתי שבע תמונות כנ"ל וניסיתי למצוא התאמות Sift בין פונקציית המטרה לכל אחת מהתמונות.

אך שוב, כאמור, בדיעבד כנראה שלא במפתיע, גם ניסיון זה לא צלח. ללא יוצא מן הכלל, לכל תמונה שבדקתי, לא נמצאו התאמות כלל. קרוב לוודאי שדבר זה נובע מכך ש־ Sift הינו כלי אפקטיבי כאשר מדובר בשינויי זווית, גודל ותאורה, אך בדאטאסט של הפרויקט נעשו שינויים דרסטיים על האותיות שהושתלו בתמונות, עיוותים שמקורם בהתאמה למפת העומק של התמונה שבה בוצעה ההשתלה.

הקוד של הניסיון הזה מתועד בקובץ sift_try.py.

8 Transfer Learning

השיטה של Transfer Learning (קישורים 7 ו־ 8 בסעיף הקישורים בתחתית המסמך) היא לקיחת מודל מאומן, החלפת שכבת הפרדיקציה בשכבות חדשות שמסתיימות בשכבת פרדיקציה מתאימה, אימון המודל החדש תוך הקפאת השכבות של המודל המקורי למספר קטן של איטרציות ועם מקדם למידה גבוה, ולאחר מכן הפשרה של מספר שכבות מהשכבות האחרונות של המודל המקורי, אימון נוסף למספר גדול יותר של איטרציות ועם מקדם למידה נמוך.

בתחום סיווג התמונות ניתן לטעון מודלים מאומנים רבים בצורה נוחה למדי, ולהשתמש במשקלים מטוייבים שנלמדו באמצעות דאטאסט עצום ובמשך מספר גדול מאוד של איטרציות. לרוב משתמשים בדאטאסט שנקרא imagenet המכיל $14,000,000$ תמונות ו־ $20,000$ מחלקות שונות.

הראציונל של השיטה הזאת הוא שבאופן כללי תמונות חולקות מאפיינים משותפים שכל רשת נוירונים לסיווג תמונות תרצה ללמוד אותם, כגון edges בתמונה. עתה, ככל שמתקדמים בשכבות כך נלמדים מאפיינים מורכבים יותר ולכן זה הגיוני להשתמש במשקלים של השכבות ה"ראשונות" (יכול להיות בפועל רוב השכבות) שכבר נלמדו תוך השקעה של משאבים רבים, ולנסות לטייב רק את המשקלים של השכבות העמוקות יותר, בשביל ללמוד את המאפיינים המורכבים הספיציפיים לדאטאסט שאותו רוצים ללמוד.

בסופו של דבר במודל שאותו הגשתי השתמשתי בשיטה זו, ואתאר את השימוש בה בתתי־הפרקים הבאים.

9 אנסמבל

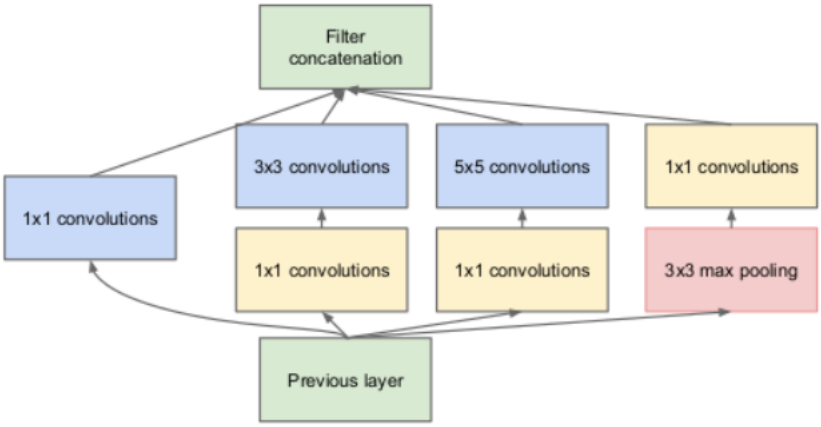
כיוון נוסף שרציתי לנסות היה לבנות אנסמבל של מודלים, קרי להשתמש במודלים מאומנים, להוסיף לאמן אותם באמצעות fine-tuining, קרי הוספה של מספר שכבות, הפשרה של מספר שכבות, ואימון על הדאטאסט של הפרויקט. לאחר מכן, בעת הפרדיקציה, הרעיון הוא לערוך הצבעה בין המודלים הללו ולבצע סיווג לפי הכרעת הרוב. לשם כך נעזרתי באתר של analyticsvidhya, שסוקר את ארבעת המודלים המובילים לכאורה בסיווג תמונות (קישור מספר 11 בסעיף הקישורים בתחתית המסמך). המודלים שהאתר הזה סוקר הם: Vgg19, ResNet50, EfficientNet ו־ Inception. אם כן, תוך מעקב אחר המדריך באתר הנ"ל וכן שימוש במדריך של keras לעריכת fine-tuning ערכתי בדיקה ראשונית להערכת ההתאמה של המודלים הללו לדאטאסט של הפרויקט. הקוד שכתבתי בשביל לערוך את הבדיקה הזאת היה מאוד דומה בין ארבעת המודלים והוא נמצא בקובץ mymodel.py.

מהבדיקה שערכתי התגלה האימון המקדים שנעשה עבור ResNet50 ו־ EfficientNet לא היטיב עם הדאטאסט של הפרויקט. לאחר כ־ 10 epochs שני המודלים לא הצליחו לעבור את ה־ 20% בעוד ש־ Vgg19 ו־ Inception הגיעו לדיוק של 30% ו־ 40% על הואלידציה.

בשל התוצאות העגומות החלטתי לזנוח את הכיוון של האנסמבל, ומאחר שהדיוק של רשת ה־ Inception הייתה הטובה מבין הרשתות שניסיתי תוך שימוש ב־ Transfer Learning, החלטתי להמשיך עם המודל הזה, ולנסות להוסיף לטייב אותו.

10 Inception

מה שמיוחד ברשת ה־ Inception (קישור 8 בסעיף הקישורים בתחתית המסמך) הוא שבמעבר משכבה לשכבה, במקום לבחור בקונבולציה מגודל מסויים בכל שלב, משתמשים בכמה קונבולוציות שונות ומשרשרים את התוצאות. כל מעבר כזה נקרא מודול, והרשת בעיקרה מורכבת ממודלים כאלה. להלן תיאור של המודול הכללי:



כאשר הקונבולוציה מסדר 1×1 היא בפרט רכיב משמעותי שכן היא מקטינה משמעותית את מספר הפרמטרים שיש ללמוד.

להלן תיאור של הארכיטקטורה של הרשת כולה:

כמו כן, רעיון נוסף שהוצג ב־ Inception הוא שכבת ה־ BatchNormalization (קישור מספר 10 בסעיף הקישורים בתחתית המסמך) אשר הוכיחה את עצמה כמשמעותית ביותר ברשתות עמוקות. שכבה זו למעשה מנרמלת את ה־ batch שעובר דרכה והסיבה שזה עוזר היא שכל שמתקדמים בשכבות, כל batch שמזינים לרשת, מופיע בהתפלגות שונה, והמשקלים של השכבות שהינם רגישים לפרמטרים של התפלגות ה־ batch שעובר דרכן, כאילו מנסות לעקוב אחר "מטרה נעה" שכן כל batch חדש שנכנס מציג התפלגות שונה לחלוטין בשכבות העמוקות יותר, וכך הרשת מתקשה ללמוד שכן כאמור, בכל צעד היא מנסה להתאים את עצמה להתפלגות שונה. לכן הוספה של שכבת נרמול עוזרת במידה רבה ללמידה של הרשת.

אם כן, השתמשתי במודל מאומן של Inception וטייבתי אותו לפי התהליך שתיארתי לעיל ב־ Transfer Learning ובעזרת מדריך של keras. הקוד שכתבתי כדי ליצור את המודל נמצא בקובץ mymodel.py וזה כאמור המודל שהגשתי בסופו של דבר.

לצערי הרב גם ניסיון זה נחל כישלון ולא הצליח לעבור בהרבה את ה־ 40% דיוק על הואלידציה.

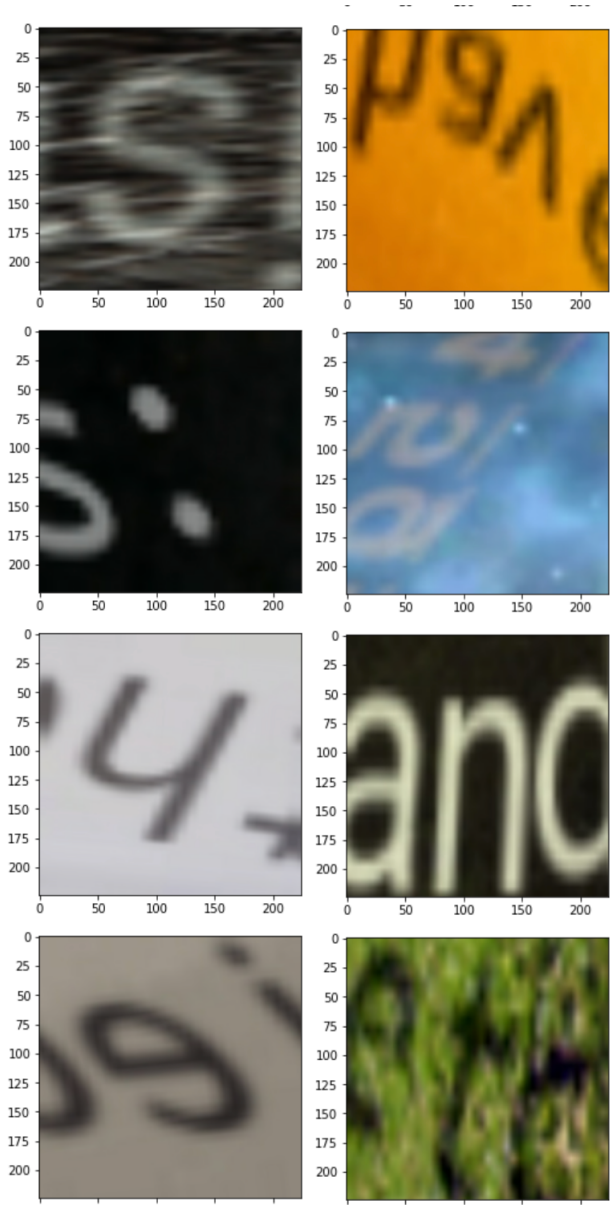
11 הרחבת הדאטאסט

לאחר הכישלון של ההרצות הראשוניות עם ה־ ResNet, הייתי משוכנע שהבעיה טמונה בכמות המועטה יחסית של תמונות שקיבלנו בדאטאסט, שהרי אוברפיט גבוה פירושו על פי רוב קושי בהכללת הפונקציה הנלמדת, מה שבתיאוריה אמור להיות מושפע באופן משמעותי מכמות המידע שהלמידה מתבצעת עליו – ככל שיש יותר מידע כך למודל יש מידע מקיף יותר הקשר בין הקלט לפלט, וכך הוא אמור לדעת טוב יותר להכליל את הקשר שהוא מנסה ללמוד.

לכן השקעתי מאמצים רבים ואף את מירב הזמן והמאמצים שלי בעבודה על הפרויקט להרחבת הדאטאסט. לשם כך ניגשתי לפרויקט של SynthText שממנו יוצר הדאטאסט שקיבלנו. עקבתי אחר ההוראות – הורדתי את הפונטים המתאימים מ־ google fonts והרצתי את הקוד של invert_font_size.py בשביל לטעון את הפונטים. הורדתי לפי ההדרכה כ־ 8000 תמונות בשביל לשבץ בהם פונטים. השלמתי את הקוד החסר בקובץ use_pre_proc.py בשביל להריץ את התכנית על התמונות שהורדתי. זיהיתי את הקבצים synthgen.py ו־ text_utils.py. מצאתי ב־ synthgen את הפונקציה place_text שעוטפת את כל התהליך של ייצור הטקסט בפונט נתון ושיבוצו בתמונה, וממנה חילצתי את הפונט שהוגרל כדי להחזיר אותו יחד עם שאר הפרמטרים, אל הפונקציה render_text שאוספת את כל תוצאות השיבוץ במילה נתונה וחוזרת אל התהליך הראשי להכנסת המידע לדאטאסט שמיועד כפלט. הרצתי את התכנית וקיבלתי קובץ SynthText.h5 באותו הפורמט שקיבלנו בדאטאסט של האימון. לאחר סקירה של הדאטא, נראה היה שהאי־דיוקים בתיחומים של האותיות אף יותר חמורים מאשר בדאטאסט המקורי שקיבלנו ולכן החלטתי להסתכל עמוק יותר בתוך הקוד עצמו בשביל לנסות לפתור את הבעיה. אם כן, לפני ההחזרה של תוצאת השיבוץ ב־ place_text, שמרתי את החיתוך של האות הראשונה במילה ובחנתי את התוצאה. גיליתי שבדרך כלל אין טעויות בחיתוך של האות הראשונה ולכן יצרתי דאטא סט נוסף שמורכב מחיתוכים כנ"ל. אך גם לאחר אימון על הדאטאסט הזה, לא קיבלתי שיפור כלל. אז בחנתי שוב את החיתוכים שהשתמשתי בהם וראיתי שבדומה לדאטאסט המקורי שקיבלנו, רבים מהטקסטים ששובצו מאוד קשים לזיהוי אפילו לעין אנושית. אז בחנתי שוב את הקוד והסתכלתי על המחלקה RenderFont שבקובץ synthgen.py. שם ראיתי את הפרמטרים שמוגדרים להגרלת הטקסט לשיבוץ בתמונה, על כל מאפייניו. אם כן, ניסיתי לשנות את הפרמטרים בשביל לקבל דאטא ברור יותר. למשל הגדלתי את self.p-flat בשביל לקבל טקסט שטוח יותר. והקטנתי את self.p-curved כדי לקבל פונט מעוות פחות. כמו כן שיניתי את הערכים של self.min-font-h ו־ self.max-font-h בשביל לקבל פונטים בגדלים סבירים יותר – שלא יהיה זעום עד כדי קושי לזיהוי ושלא יהיה ענק עד כדי כך שלא מופיע בשלמותו. אך כרגיל גם כל הנסיונות הללו לא עזרו. לא ויתרתי. חשבתי שאולי בכל זאת החיתוכים לא מדוייקים מספיק. אז הסתכלתי אפילו עמוק יותר לתוך הקוד וזיהיתי את הפונקציה render_curved

בקובץ text_utils.py. ראיתי ששם מתבצע הרינדור עצמו של הטקסט עם הפונט אשר משובץ לתוך מסיכה שלאחר מכן בפונקציה place_text שבקובץ synthgen.py משולב יחד עם התמונה המקורית בהתאם לתמונת העומק שלה, בפונקציה self.colorizer.color. אז בחנתי לעומק את הקוד שבפונקציה render_curved וראיתי שהמסיכה עם הטקסט נמצאת במשתנה של surf-arr ושלאחר ההעברה של המסיכה לפונקציה crop-safe, הטקסט לעתים קרובות נחתך ולאחר מכן משובץ לתוך התמונה באופן משובש. אם כן, ביטלתי את הקריאה לפונקציה הזו וראיתי שעכשיו קיבלתי את שיבוצים מלאים לתוך התמונה וכן באמצעות הקריאה cv2.boundingRect(text-mask) אף קיבלתי את המלבן **המדויק בהחלט**! של המילה שמשובצת במסיכה. כך יצרתי בכל הרצה 35 אלף תמונות של מילים עם התיחום שלהן ללא רבב. ביצעתי 5 הרצות כאלה ובסה"כ הגדלתי את הדאטאסט לכדי כ־ $175,000$ תמונות. עתה, השתמשתי בתמונות הללו לאימון ואילו בתמונות שקיבלנו חילקתי לואלידציה ומבחן. אך כאמור סט האימון מכיל מילים ואילו הסט שקיבלנו מכיל חיתוך של אותיות, אז שוב בחנתי את הכיוון של לאמן על מילים במקום על אותיות, נסיתי זאת, אך הדיוק החמיר ולכן הייתי צריך לחשוב על משהו אחר. הרעיון שחשבתי עליו הוא כמו שתיארתי בסעיף 1 – עבור סט המבחן והואלידציה, לקחת את הסביבה של כל אות בשביל לקבל חלקי מילים, ואילו עבור סט האימון לבצע חיתוך רנדומלי של התמונה בשביל לקבל חלק מהמילה בכל דגימה. ואז היה לי סט מבחן מכובד ביותר בגודלו שדי דומה לסט המבחן והואלידציה שאני רוצה להתאים את המודל אליהם. להלן דוגמיות מהאימון והואלידציה:

דוגמיות מהאימון (מימין) ומהואלידציה (משמאל):



חזרתי לרשת ה־ Inception והתאכזבתי לגלות שגם זה לא עזר והדיוק העגום נשאר כשהיה.

המסקנה שלי היא שכנראה הדאטא שנדרשנו לעשות עליו איבאליואציה פשוט לא מדויק מספיק – קרי התיחומים לא מספיק מדוייקים ולכן בחלק גדול מהמקרים מה שנחתך בפועל הוא חלקים בתמונה שלא מכילים את האותיות שהם אמורים להכיל ולכן אין דרך לדעת מה הפונט שהאמור להיות מיוחס לחיתוכים הללו.

12 ניסיון לשפר קצת את הדיוק

אז בתור שימוש ב"קלף אחרון", בהינתן מילה, ניסיתי להשתמש בסיווגים שהאותיות שלה קיבלו ולערוך הצבעה – הסיווג שקיבל רוב ייבחר כסיווג של כל האותיות במילה. השיטה הזו סייעה לי לעלות מ־ 40% דיוק על סט המבחן ל־ 50% כך שמדובר בשיפור משמעותי, אמנם לא "מספיק" בשביל להיות מרוצים מהתוצאה, אך ביחס למה שהיה קודם זהו שיפור ניכר.

13 הפיתרון האחרון שהשתמשתי בו

אז בסופו של דבר לקחתי את כל הדאטא שקיבלנו, חילקתי אותו לאימון, ואלידציה ומבחן ביחס של 0.6, 0.2 ו־ 0.2 בהתאמה, והכנסתי לסט המבחן גם חלק מסט המילים שיצרתי.

עבור סט האימון שקיבלנו השתמשתי באוגמנטציה של סיבוב, הזזה, היפוך, ניגודיות וסיבוב, ועבור סט המילים שיצרתי השתמשתי בחיתוך רנדומלי והגדלה בחזרה לגודל המקורי.

והמודל שהשתמשתי בו הוא ב־ Inception עם Transfer Learning, תוך נרמול הדאטא ב־ 1/255 לפני הכנסתו ל־ Inception, הוספת שכבת AveragePooling, שיטוח של הפלט והכנסה לשכבת Dense בגודל של 128. לאחר מכן הוספתי שכבה של BatchNormalization וכן שכבה של Dropout עם שיעור של 0.2. לבסוף הוספתי את שכבת הפרדיקציה.

עבור האופטימייזר השתמשתי ב־ Adam גם להרצה על המודל המוקפא בהתחלה וגם על ההרצה על המודל המופשר לאחר מכן. אימנתי את השכבות האחרונות ל־ 5 epochs כאשר ה־ Inception מוקפא ואז עוד 100 epochs תוך הפשרת שני בלוקים של Inception, קרי משכבה מספר 249.

כמו כן השתמשתי בתזמון של מקדם הלמידה כך שבעת ההקפאה התחלתי עם מקדם למידה של 0.01 עם הקטנה בפקטור של 0.97 כל 1000 צעדים שזה בערך כל epoch עבור batches בגודל 32 עם כ־ 32,000 דוגמאות לאימון. לאחר מכן בעת ההפשרה הקטנתי את מקדם הלמידה ל־ 0.001 עם הקטנה באותו פקטור אך כל $10,000$ צעדים שזה

בערך כל 10 epochs ביחס לגודל ה־ batches שציינתי וכן מספר הדוגמאות לאימון.

בסופו של דבר הצלחתי להגיע לדיוק של 50% על הטסט יחד עם העיקרון מהסעיף הקודם, כאשר אמנם זה לא נשמע גבוה אבל יש להזכיר שמודל רנדומלי היה משיג דיוק של כ־ 14% עם ניחוש, וכן בהתחשב באיכות הדאטאסט עם התיחומים הקשים, ייתכן שזאת תוצאה סבירה. מה גם שביחס לעיקרון בייס שמתייחס לדיוק "הטוב ביותר" שניתן להשיג, גם כן יש סיבה לאופטימיות שהרי מודל בייס בסיווג תמונות מיוצג בדרך כלל על ידי העין האנושית, והעין שלי לפחות מתקשה להבדיל בין הפונטים בתמונות שלא לדבר על לזהות שיש בכלל טקסט בתמונות כחלק בלתי מבוטל מהמקרים.

14 הגשה והוראות הרצה

בשביל להריץ את המודל שלי לקבלת התוצאות ב־ csv, יש להוריד אותו מקומית מלינק מספר 2 בסעיף הלינקים למטה ולהיכנס למחברת בגוגל קולאב בלינק 1 בסעיף הלינקים למטה. עתה יש להכניס את הנתיב המקומי למודל שלי וכן נתיב לסט המבחן הרצוי. לאחר מכן ניתן להריץ והתוצאה תישמר בשם baskin_results.csv בתוך התיקייה של סט המבחן.

```
[ ] model_path = '/content/drive/MyDrive/model.h5'
    dataset_path = '/content/drive/MyDrive/SynthText_test.h5'
```

```
[ ] main_path = '/content/drive/MyDrive/openu-cvchallenge/submission'
    results_path = f'{main_path}/baskin_results.csv'
    test_path = f'{main_path}/test/test.h5'
```

כלומר: יש להכניס את model_path למיקום המודל ואת dataset_path למיקום הדאטא שרוצים לבדוק ואת main_path למיקום התיקייה לשמירת התוצאות.

15 קישורים

1. חוברת הרצה שלי
2. המודל שלי
3. התוצאות מהמודל שלי (baskin_results.csv)

4. דוגמית מהדאטא אותיות שיצרתי
5. דוגמית מהדאטא מילים שיצרתי
6. ResNet ב־ coursera
7. Inception ב־ coursera
8. Transfer Learning ל־ Inception ב־ keras
9. fine-tuning ב־ tensorflow
10. batch normalization ברשתות עמוקות
11. אנסמבל של סיווג תמונות מ־ analyticsvidhya
12. אוגמנטציה ב־ keras
13. FaceNet ב־ coursera
14. השוואה בין תמונות עם sift
15. text-recognition ב־ keras
16. קישור לתעודת הסיום שלי בקורס Neural Networks and Deep Learning של deeplearning.ai ב־ coursera
17. קישור לתעודת הסיום שלי בקורס
    Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization
    של deeplearning.ai ב־ coursera
18. קישור לתעודת הסיום שלי בקורס Structuring Machine Learning Projects של deeplearning.ai ב־ coursera
19. קישור לתעודת הסיום שלי בקורס Convolutional Neural Networks של deeplearning.ai ב־ coursera