תיכון עירוני ד׳ ע״ש פרופ׳ אהרון קציר

חלופת למידת מכונה בהיקף 5 יח״ל

סמל שאלון 883589

פרויקט בלמידת מכונה:

מתמונה למשפט בעברית – אפליקציה מבוססת NLP לתיאור תמונות בשפה העברית עם התמחות מיוחדת ברכבים

כותב העבודה: נדב סרגוסי

ת.ז: *********

מנחה: שי פרח

שנה״ל תשפ״ב

מאי 2022

תוכן עניינים

מבוא

רקע כללי

התחלתי ללמוד את תחום הבינה המלאכותית לפני כשנה וחצי, בתחילת כתה יא׳. מאז, עיקר הלמידה הייתה של תחום הקונבולוציה, במסגרתו ביצעתי מספר פרויקטים מעניינים. על כן, כשנדרשתי לבחור נושא לפרויקט הגמר, רציתי לגוון ולהתעמק בנושא מרתק בעיניי בלמידה עמוקה, והוא ה-NLP, הנמצא ברובו מחוץ לתחום הלימודים.

ובכן, הפרויקט שבחרתי לבצע הינו בנושא ה-Image Captioning, כלומר קבלת תמונה כלשהי כקלט ופליטת משפט בעברית המתאר את התמונה. בנוסף, בזכות חיבתי הרבה לתחום הרכב, רציתי לתת דגש מיוחד אליו. הדבר בא לידי ביטוי בכך שבמידה ויש רכב בתמונה, המודל ינסה לאתר מידע נוסף אודותיו דוגמת שם החברה, הדגם וכן המחיר. החידושים העיקריים בפרויקט הם שהמשפט מוצג בעברית וכן המידע הנוסף עבור הרכבים.

הפרויקט מסתכם באפליקציית אנדרואיד נוחה, ובה יכול המשתמש לצלם תמונה או לבחור אחת מהגלריה, ולאחר כמה שניות יופיע משפט בעברית המנסה לתאר בצורה מיטבית את המופיע בה.

לדעתי, הפרויקט יכול לתרום רבות ללקויי ראייה בעיקר. כל שהם צריכים לעשות זה לפתוח את האפליקציה ולצלם תמונה (ראשית יש להתרגל וללמוד את סדר הפעולות כמובן), והם יקבלו תיאור מדויק להפליא של העומד מולם בצורה קולית בעזרת TTS (Text-to-Speech), תוספת שקל מאוד לממש בעתיד באפליקציה. בנוסף, הפרויקט יכול לתרום לחובבי רכבים כמוני, שעשויים לעבור ברחוב, לראות דגם מסוים של רכב ולתהות באשר לו. גם כאן, כל שנדרש הינו צילום הרכב מכמה זוויות והסיכוי לקבל מידע מפורט עליו גבוה.

אתגרים מרכזיים

האתגר המרכזי איתו אני מתמודד הוא העובדה כי מרבית הנושאים בהם עוסק בפרויקט אינם נכללים בתוכנית הלימודים, מה שיצריך ממני למידה והשקעה מרובים בנושאים מאתגרים. בנוסף, צפויים לי אתגרים בפיתוח האפליקציה ובהטמעת המודל בה, כיוון שמדובר בתחום שזר לי לחלוטין, אין לי בו ניסיון מקדים והוא כמובן אינו נלמד בביה״ס.

ניתן לחלק את תהליך הפרויקט שלי לשישה שלבים מרכזיים:

1 **למידה ומחקר**

בשלב הראשון, התמקדתי בהרבה למידה ומחקר אודות תחום ה-NLP והרחבת הידע שלי בלמידה עמוקה באופן כללי.

2 **בנייה ואימון של מודל ראשוני**

3 **הערכה וניתוח התוצאות**

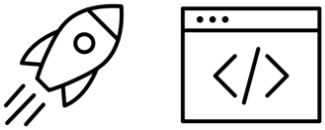
בשלב השני עמלתי על בניית מודל ראשון אשר מטרתו היא לבצע Image Captioning, כלומר בהינתן תמונה, על המודל לפלוט משפט המתאר את הנמצא בתמונה.

בשלב השלישי הערכתי וניתחתי את התוצאות של המודל לאחר האימון ובעקבות כך ניסיתי לשפר אותו ככל שיכולתי. שלבים 2 ו-3 חזרו על עצמם כשלוש פעמים.

4 **למידה ומחקר נוספים לצורך שיפור**

בשלב הרביעי המשכתי את הלמידה והמחקר בכדי לשפר את המודל עוד יותר, ובין השאר פניתי לכיוונים חדשים.

5 **פיתוח והטמעת המודל החדש והאפליקציה**



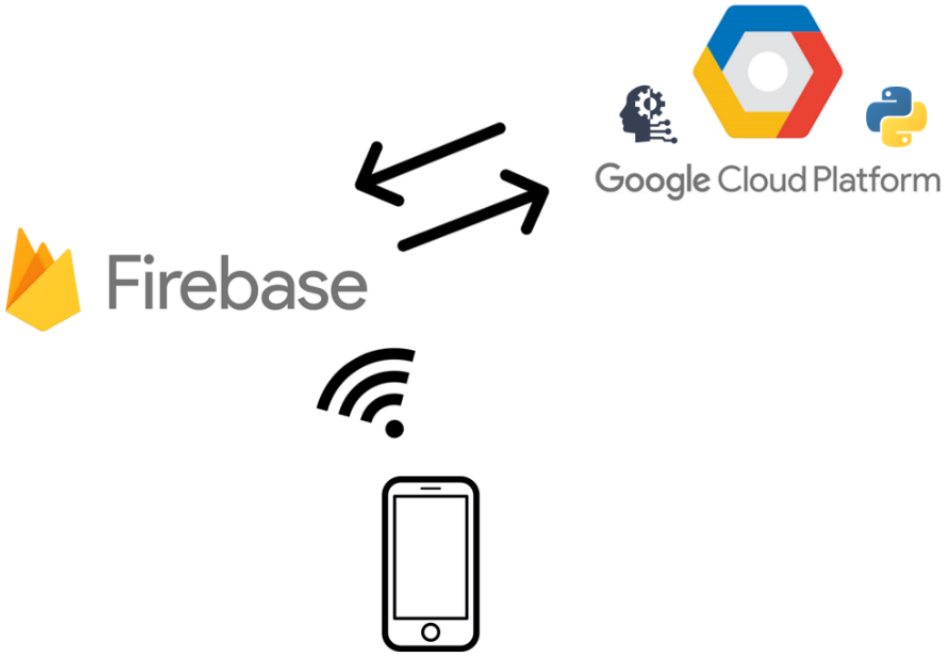
בשלב החמישי, לאחר שהגעתי למודל שעובד טוב מאוד, עמלתי על פיתוח אפליקציית אנדרואיד לצורך מימוש הפרויקט. בנוסף, הטמעתי את המודל באפליקציה ע"י חיבור לענן.

6 **אפליקציה מוכנה לשימוש**

בשלב השישי והאחרון שיפרתי ושכללתי את האפליקציה ודאגתי שהיא תהיה מוכנה להורדה ולשימוש באופן סופי. לבסוף, שכללתי את האפליקציה בכך שהוספתי התמחות מיוחדת ברכבים.

ארכיטקטורה

ארכיטקטורת הפרויקט כולו ברמת המאקרו



התרשים מעלה מציג את מבנה הפרויקט במלואו.

ראשית, אופן היישום הוא באפליקציית אנדרואיד נוחה וברורה. המשתמש מצלם תמונה או בוחר אחת מהגלריה, והיא משוגרת למסד נתונים מקוון בשם Firebase (מבית גוגל), אשר שולח את התמונה ופרטים רלוונטיים נוספים לפלטפורמת הענן של גוגל, שם נמצא קוד הפרויקט ובעצם המודל המעשי. התמונה מועברת כקלט למודל, הוא פולט את הכיתובית העברית לתמונה ושולח אותה בחזרה למסד. לבסוף, המסד מחזיר למכשיר הקצה את הפלט והוא מוצג למשתמש על המסך בסמוך לתמונה.

(הסברים מפורטים נוספים – בהמשך)

אפיון המודל

אופי המודל

מודל של Image Captioning בראש ובראשונה צריך "לדעת" מה קורה בתמונה, בדומה למשימות בתחום הראייה הממוחשבת. כלומר, לכל מודל שכזה, יש בסיס באופי של קונבולוציה שמטרתו לזהות את הפיצ'רים החשובים בתמונה. מכיוון שמטרת הפרויקט אינה לייצר מודל בראייה ממוחשבת, וכן המשאבים שעומדים לרשותי מוגבלים, אבצע Transfer Learning בעזרת מודל CNN (קונבולוציה) מוכר ומאומן מראש. כלומר למעשה, ארכיטקטורת המודל השלם מורכבת ממודל קונבולוציה המהווה את הבסיס, ומודל NLP אשר מתלווה לו ונעזר בפלטים שלו על מנת לייצר את תיאור התמונה.
בחרתי במודל ה-EfficientNet שאומן ע"י מערך הנתונים המפורסם ImageNet. הבחירה דווקא במודל זה נובעת מהעובדה שהוא כללי יחסית בתפקודו (Image Classification באופן כללי, ולא שימוש ספציפי אחר) וביכולתו להסתגל להרבה צרכים. בנוסף, ה-EfficientNet ידוע בהיותו יעיל מאוד (ומכאן השם Efficient) ביחס למגבלות כוח חישוב וזיכרון גם במכשירי קצה דוגמת הטלפון הנייד, והוא הרי היעד הסופי לפרויקט בו יימצא המודל. (עוד על EfficientNet בהמשך)

מערך הנתונים עבור המודל - Flickr8K

מערך הנתונים בו השתמשתי לאימון המודל הראשוני הוא מערך בשם Flickr8K Dataset. מערך זה הינו מהמוכרים בעולם בתחום התיאורים המילוליים של תמונות, והוא מכיל 8,000 תמונות שלכל אחת מהן 5 תיאורים שונים המתארים את המופיע בתמונה בשפה האנגלית. את כל התיאורים נתנו בני אדם, והתמונות עצמן נבחרו בקפידה כך שיהיה מגוון רחב של מצבים ונופים, ושכמעט ולא יהיו אנשים ו/או מקומות מפורסמים.
השתמשתי במערך זה מכיוון שהוא יחסית קטן ונוח להרצה. הוא מסודר, מאורגן וערוך בצורה ידידותית וקל לשימוש. לאחר ההורדה, ישנם שני קבצים: הראשון מכיל את כל 8,092 התמונות בפורמט JPEG ובגדלים שונים ומגוונים, והשני הוא קובץ טקסט מסודר המכיל את כל תיאורי התמונות המקושרים אוטומטית לתמונות.
תיקיית התמונה מחולקת 20% לטובת ה-validation set, חלוקה סטנדרטית ונפוצה ביחס לכמות הנתונים (בפועל – 6,473 תמונות עבור סט האימון ו-1,619 עבור ה-validation).

השינויים היחידים שנבצע הם במשפטים, ע"י הוספת "<start>" בתחילת כל משפט וכן "<end>" בסופו על מנת שהמודל ידע זאת (עוד על כך בהמשך).

רעיונות, יסודות ומודלים בתחום ה-NLP

מודל ה-Image Captioning

ברמת העיקרון, מכיוון שפלט המודל הוא משפט, ומשפט הוא רצף (Sequence), ברור שמדובר כאן ברשתות RNN. ואמנם, השתמשתי במודל מסוג Transformers. לשימוש זה יש כמה שיקולים.
בתור התחלה, ל-RNN's יש בעיה ידועה והיא ה-Vanishing/Exploding Gradients, אשר נפוצה ברשתות מסובכות יותר ומקשה על יצירת קשרים מורכבים בין כל חלקי המשפט ובין מילים רחוקות.
ה-LSTM ו-GRU אכן נועדו לפתור בעיות אלה, אך היבט נוסף הוא שהן עדיין מבצעות את הפעולות על המילים בזו אחר זו, מה שמעיק מאוד על תהליך האימון ומצריך זמן וכוח חישוב רבים.
לעומת זאת, מודל ה-Transformers, אשר היווה פריצת דרך בתחום ה-NLP, מאפשר התמודדות אף עם טקסטים שלמים במהירות ובאופן מדויק יותר. בנוסף, כפי שנראה בהמשך, בעקבות מבנהו של המודל, ניתן לבצע בו את מרבית החישובים בו-זמנית, מה שחוסך הרבה מאוד זמן ביחס ל-RNN's, GRU's ו-LSTM's. לבסוף, ה-Transformers מסוגל להסתגל לשלל צרכים ומטרות שונות בתחום ה-NLP, כמו תרגום משפטים, הסקה מטקסטים, וכמובן Image Captioning. כלומר, מדובר במודל שנוסף לכל מצטיין ב-Transfer Learning, בו גם אני נעזר.

בחלק זה אציג כמה רעיונות ויסודות העומדים בבסיס תחום ה-NLP ובעיקר את מודל ה-Transformers.

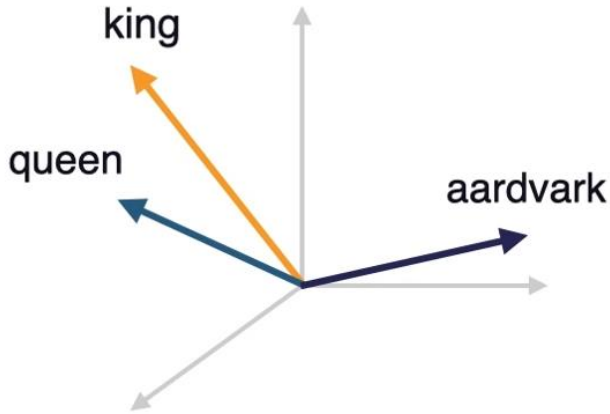
Sentence Tokenization

כל משימה ב-NLP מתחילה עם Tokenizer כלשהו. Tokenizer היא בעצם פונקציה מותאמת אישית שלוקחת את המידע (במקרה הזה המשפטים), וממירה אותו לפורמט כזה שיתאים למודל.
שלב ראשון ב-Tokenization אצלי כבר הוזכר, והוא הוספת <start> בתחילת כל משפט ו-<end> בסופו.
השלב השני הוא להפריד את המשפטים לרשימות של מילים, משום שהמודל עובד עם מילים ולא משפטים שלמים כיחידות. לאחר מכן, ממירים את המילים לוקטורים בעלי משמעות מבחינת המודל, כפי שנראה מיד.

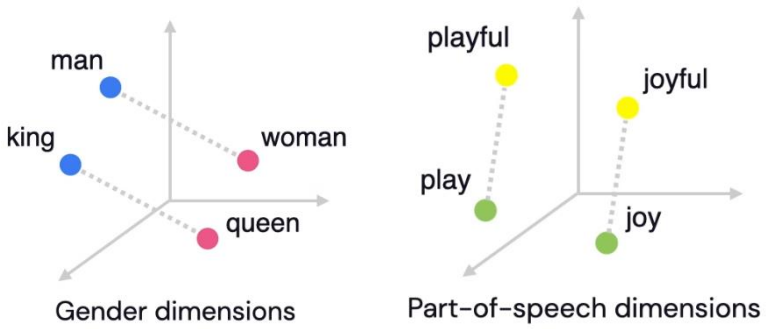
מיפוי מילים במרחב הוקטורי

לפני שניכנס לפרטים נרחבים יותר ב-NLP, נשאלת השאלה כיצד המחשב מתייחס למילים?
ובכן, המחשב כמובן אינו מבין מילים כפי שאנו בני האדם מבינים, אלא מבין מספרים, וקטורים, מטריצות וכו׳. לכן, עלינו קודם כל לייצג מילים בצורה שהמחשב יבין. זה נקרא Word Embedding, כלומר המרת טקסט למספרים (למעשה וקטורים).
דרך בסיסית לעשות זאת היא פשוט לקחת את המילון ולכל מילה להתאים מספר כאינדקס שייצג את המילה. ואולם, לא קשה לזהות את הבעיות בדרך זו. ראשית, רשתות נוירונים לא נוחות לעבודה עם מספרים גדולים מאוד שכאלה. שנית, המטרה ב-NLP היא לייצג גם קשר של דמיון ושוני בין מילים, מה שלא מתאפשר לנוכח השיטה. במקום זאת, אפשר לנסות לבצע One-Hot Encoding, כלומר עבור כל מילה יש וקטור באורך המילון, והמילה המתאימה מיוצגת ע״י הספרה 1 והשאר ע״י 0. כמו כן, מדובר בדרך לא יעילה שתופסת הרבה מקום וזמן, וגם דרכה לא ניתן לייצג קשר אמיתי בין המילים.
במקום זאת, הדרך הטובה ביותר הנוכחית היא ליצור מרחב d מימדי, ובו מייצגים כל מילה עם ערכים לכל אחד מהמימדים. כך מתגברים על סיבוכיות האחסון והזמן, שכן מדובר בוקטורים לא ארוכים עם מספרים לא גדולים, וכן ניתן ליצור תחומים במרחב שיצביעו על קשרים בין המילים.
הן מס׳ המימדים והן הערכים שכל מילה מקבלת הן תהליך למידה ארוך הניזון מכמויות אדירות של מידע, ועל כן בדרך כלל משתמשים במודל מיפוי מוכן, כמו גם בפרויקט שלי. דוגמא למיפוי מילים במרחב תלת-מימדי:

Token	Continuous vector
aardvark	[0.3, 1.9, -0.4]
king	[2.1, -0.7, 0.2]
queen	[0.5, 1.3, 0.9]



מה מייצג כל מימד?
ובכן, הכוח בוקטור מרובה מימדים הוא שכל מימד יכול לייצג אספקט שונה של המידע על המילים.
למשל, n מימדים כלשהם מתוך d יכולים לייצג את המין הדקדוקי של המילה, k אחרים את חלק הדיבור וכו׳. כוח זה בא לידי ביטוי במודלים של המיפוי, בכך שהם כוללים קשרים ברורים וסימטריים בין מילים.
למשל:
אנו רואים כי ביחס למימדי חלק הדיבור, המרחק בין המילים play ו-playful זהה למרחק בין joy ל-joyful, שכן אכן מדובר באותם יחסים. בנוסף, ניתן לראות שהמרחק בין man ו-woman זהה למרחק בין king ו-queen, שכן למעשה מדובר באותה משמעות רק במין דקדוקי הפוך.



Self-Attention

אבן היסוד של מודל ה-Transformers, ומה שבעצם הקנתה לו את כוחו האדיר היא הקונספט של Attention. לשם הדגמה אינטואיטיבית ופשוטה יותר, נתייחס לקלט ה-Attention כאל משפט בצרפתית שנרצה לתרגם לאנגלית.

המשפט לדוגמא הינו: **"Jane visite l'Afrique en septembre"**.

בתרגום לאנגלית: Jane is visiting Africa in September.

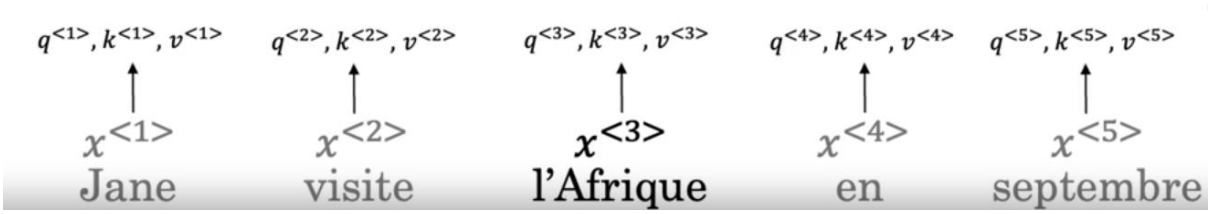
הרעיון הכללי של ה-Attention הינו, בעבור כל מילה במשפט, לייצר וקטור המאפיין בצורה הטובה ביותר את אותה מילה. מה הכוונה ל"בצורה הטובה ביותר?" – לקחת בחשבון את המשמעות של המילה כפי שהיא במרחב הוקטורי, למצוא את הקשרים שלה עם המילים האחרות במשפט, את הקונטקסט של המילה, את המשמעות הספציפית שלה מתוך כמה שיכולות להיות וכו'.

מדוע השיטה של ייצוג במרחב הוקטורי בלבד אינה מספקת? ובכן, כאשר אנו רוצים שהמחשב "יבין" כל מילה בצורה מיטבית, יש להסתכל על הדרך בה אנו, בני האדם, מפרשים מילה במשפט. ניקח לדוגמא את המילה השלישית, Africa. אם נסתכל עליה "סתם ככה" בתור מילה, יכולות לעלות כמה פרשנויות אודות המשמעות המדויקת, למשל אפריקה כיבשת, אפריקה כיעד תיירותי נחשק, אפריקה כמקום היסטורי חשוב ועוד. אם כן, כיצד אנו יודעים שהכוונה במשפט היא שאפריקה מייצגת מקום ביקור? בזכות הקונטקסט – כלומר בהסתכלות על המילים האחרות במשפט, ובפרט visit. <u>תהליך זה הוא בדיוק התהליך שמנגנון ה-Attention מנסה לדמות</u>, ומכאן השם Attention – מתן תשומת לב למילים האחרות במשפט בכדי לפענח את המשמעות המדויקת של מילה נתונה.

הנוסחה הכללית ל-Selt Attention, אותה נסביר מיד, הינה:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

ראשית, נתייחס למשפט בצורת Sequence, ונתמקד בחישוב ה-Attention עבור המילה השלישית:



לכל מילה יש שלושה ערכים תואמים: Query (q), Key (k) ו-Value (v). כולם ערכים נלמדים הדרושים בכדי לחשב את $A^{<3>}$, שהוא למעשה הוקטור הרצוי בעבור המילה השלישית. לפני תחילת החישוב, יש לציין כי לכל אחד מהערכים הללו יש מטריצת משקלים שגם נכללת בחישוב, אותה המודל לומד. נסמן אותן ב-W^Q, W^K ו-W^V. בנוסף, $x^{<i>}$ הוא למעשה ערך ה-text encoding מאותו מרחב וקטורי בעבור כל מילה.

החישוב כולל שבעה שלבים בעבור כל מילה:

שלב 1 – חישוב $q^{<3>}$: $W^Q * x^{<3>}$

שלב 2 – חישוב $k^{<3>}$: $W^K * x^{<3>}$

שלב 3 – חישוב $v^{<3>}$: $W^V * x^{<3>}$

כעת נשאלת השאלה, מה הוקטורים הללו בכלל מייצגים?
ובכן, ניתן לדמות את q כאל שאלה מסוימת שאנו שואלים על המילה l'Afrique, כמו למשל "מה קורה שם?". כעת נרצה לענות על השאלה, בכדי להבין מה משמעות המילה בהקשר של המשפט. לשם כך נחלץ מהמשפט את המילה / מילים שעונות על השאלה בצורה הטובה ביותר. את k ניתן לדמות כאל "הכותרת לתשובה" לשאלה שיש בכל מילה, ואת v כאל התשובה המפורטת יותר הקשורה ל-k שלה. לכן, בעבור כל מילה, נבצע את החישוב הבא, בכדי למצוא כמה "חזקה" התשובה של כל מילה:

שלב 4 – מציאת המילים החשובות ביחס למילה הנוכחית: $q^{<3>} * k^{<1>}$, $q^{<3>} * k^{<2>}$, $q^{<3>} * k^{<3>}$ ...

מבחינת המודל, ככל שהערך גבוה יותר, כך יש קשר חזק יותר בין המילים וניתן להבין טוב יותר את משמעות האחת בעזרת השנייה. כאמור, כך מאתרים את המילים המשפיעות ביותר על המשמעות של כל מילה.

שלב 5 – לוקחים את כל הערכים שהתקבלו ומעבירים אותם דרך softmax.

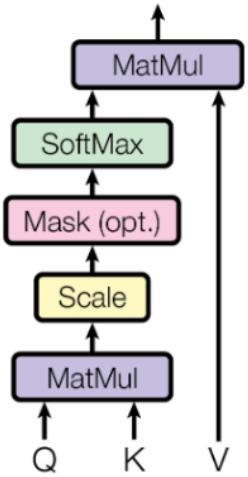
שלב 6 – לוקחים את שהתקבל עבור כל מילה ומכפילים ב-$v^{<3>}$.

שלב 7 – סוכמים את הכל וזהו למעשה הערך הרצוי שלנו, $A^{<3>}$.

את כל שבעת השלבים ניתן לסכם בנוסחה אחת המייצגת את תהליך החישוב עבור מילה ספציפית:

$$A(q,K,V)=\sum_i \frac{\exp(q\cdot k^{<i>})}{\sum_j \exp(q\cdot k^{<j>})} v^{<i>}$$

לסיכום, לכל מילה יש **q**, שהיא כמו שאלה ששואלים על אותה המילה בכדי להבין את משמעותה והקשרה במשפט טוב יותר. עבור כל מילה, ה-**k** שלה בוחן את טיב התשובה עבור **q**, וכך מצביע על אילו מילים רלוונטיות למענה על השאלה, כלומר על חוזק הקשר. לבסוף, ה-**v** הוא מעין ערך סופי שמטרתו לקבוע איך ייראה ייצוג המילה. ניתן לדמות אותו לערך המילוני המופיע בעבור כל מילה.
[אנלוגיה נוספת ל-q, k, v היא של תהליך חיפוש מידע באינטרנט. אנו מתחילים בלהקליד משהו בשורת החיפוש, וזה למעשה כמו ה-q. לאחר מכן עולות הרבה כותרות לתוצאות, שכל אחת מהן היא כמו ה-k. את התוכן המפורט יותר שכל תוצאה מכילה, ניתן לדמות כאל ה-v].



נחזור אל הנוסחה הכללית: $\mathrm{Attention}(Q,K,V)=\mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

הנוסחה היא התמצית לכל התהליך עבור <u>כל אחת ממילות המשפט</u>, ולא עבור מילה בודדת כמו בנוסחה העליונה יותר. בנוסחה זו, תפקיד המכנה שבתוך ה-softmax הוא בסך הכל בכדי שה-dot product שמבצעים לא "יתפוצץ", כלומר שלא יגיע לערכים גבוהים מאוד (למעשה scaling).

משמאל ניתן לראות תרשים ויזואלי של תהליך החישוב שהוסבר.

Multi-Head Attention

בכל פעם שמבצעים Self-Attention על המשפט, זה נקרא head (h). לכן, Multi-Head Attention זה בעצם לבצע את תהליך ה-Self-Attention כמה פעמים.

עבור כל h, יש מטריצות משקלים שונות (W^Q, W^K ו-W^V), שכל אחת מהן נותנת זווית משמעות אחרת למשפט. ניתן לדמות זאת כאל כך שכל h שואל שאלה אחרת על כל אחת ממילות המשפט, למשל "מה קורה שם?", "מתי?", "מי?" וכו'. כך ניתן ליצור אצל המודל הבנה עמוקה אף יותר של המשפט.

את הפלטים שהתקבלו מכל h אוספים לכדי וקטור אחד (concatination), אשר מוכפל במטריצת משקלים נלמדת נוספת ואחרונה W^O. ניתן לתאר את כל התהליך כך:

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

אחד היתרונות הגדולים במנגנון ה-Attention הוא שאין תלות בין אף head ברמת המאקרו ובין אף מילה במשפט ברמת המיקרו. לכן, כל החישובים מתבצעים בו-זמנית.

Positional Encoding

גם מי שלא מנוסה ב-RNN's או NLP ודאי מבין שידיעה על סדר המילים במשפט הינה הכרחית. אם כך, איך המודל יודע את סדר המילים במשפט? סדר המילים במשפט הוא כמובן דבר חשוב מאוד, הן מבחינת משמעות המשפט והן מבחינת התחביר שלו.
ברשתות ה-RNN השונות, מכיוון שהן מתבצעות שלב אחר שלב גם בחלק ה-Encoding, יש תשומת לב תמידית למיקום המילים. ובכן, במודל ה-Transformers, רגע לפני שהקלטים נכנסים אל ה-Encoder (בהמשך), "מוזרקת" אליהם פיסת מידע נוספת והיא כוללת את מיקומי האיברים בקלט, בתהליך הנקרא Positional Encoding. אופן החישוב של תהליך זה מורכב מאוד מבחינה מתמטית ולכן העדפתי שלא להעמיק בו יותר מדי, אך השורה התחתונה היא שהוא כאמור עוזר למודל לדעת את מיקומי האיברים ברצף המתקבל כקלט.

מודל ה-Transformers

בתחתית העמוד ניתן לראות את הארכיטקטורה הכללית של המודל. ישנם שני חלקים מרכזיים, ה-Encoder (הבלוק השמאלי) וה-Decoder (מימינו). ה-Encoder מקבל כקלט את ה-Embeddings של התמונה, אשר מופקים ע״י מודל ה-CNN אשר אומן מראש. ה-Embeddings זה וקטור בעל מספר נמוך של מימדים המייצג את התמונה, תוך מתן דגש על החלקים החשובים והמשמעותיים ביותר בה (הפיצ׳רים). הכיווץ לוקטור קטן יותר הכרחי משום שהתמונות בימינו עצומות בגודלן ובמימדיהן והמודל אינו יכול לשאת אותן. ואומנם, לפני שהמידע מועבר ל-Encoder, מתבצע התהליך החשוב מאוד Positional Encoding, שתפקידו כזכור להוסיף לכל פיסת מידע את המיקום שלה בקלט, שכן אין במודל ה-Transformers מעקב דומה לזה של RNN's / GRU's / LSTM's על מיקומי האיברים שבקלטים.
המידע מועבר ל-Encoder, אשר בעזרת מנגנון ה-Attention מנסה ללמוד את הקשרים בין חלקי התמונה ואת הקונטקסט שלהם. ה-Encoder חוזר על עצמו כמה פעמים, בדרך כלל 6. מידע זה מועבר לרשת נוירונים קטנה ונלמדת שמטרתה היא לחדד את הפיצ׳רים החשובים אודות הקלט שזוהו ע״י ה-Attention וגם להחזיר את מימדי המידע למימדי הקלטים. בהמשך, ה-Decoder, בשילוב המידע מה-Encoder וכן המידע ממערך הנתונים, מייצר מילה אחר מילה, כאשר בכל פעם הוא מנסה לחזות בצורה הטובה ביותר את המילה הבאה, כאשר המילים שיוצרו עד כה מצטרפות לקלט גם הן.

הקשר בין ה-Encoder ל-Decoder מתרחש בשני אופנים: בהתחלה, הקלט של ה-Decoder הוא הפלטים של ה-Encoder, ובהמשך, ב-Multi-Head Attention בלוק השני, המטריצות Q ו-K מתקבלות מה--Multi Head Attention של ה-Encoder.

ה-Decoder מתחיל בלחזות את המילה שתבוא אחרי <start>. החיזוי מתבצע מתוך וקטור באורך m (m – מספר המילים במאגר בו משתמשים), ועליו פונקציית softmax שחוזה מה ההסתברות של כל מילה להופיע במיקום הבא במשפט. כל מילה שה-Decoder מפיק מצטרפת לקלט באיטרציה הבאה שלו.

לאורך כל הארכיטקטורה יש Residual Connections. הם מופיעים בכדי לשמור על מידע חשוב משכבות קודמות ובה בעת בכדי למנוע את בעיית ה- Vanishing Gradients. בנוסף, יש שכבות המוצגות בשם ״Add & Norm״. ה-Add הוא למעשה חיבור הקלטים שנכנסו יחד אל השכבה, וה-Norm מצביע על ביצוע נורמליזציה סטנדרטית לערכים, כמו בבעיות רגרסיה (לפי ה-mean וה-std).

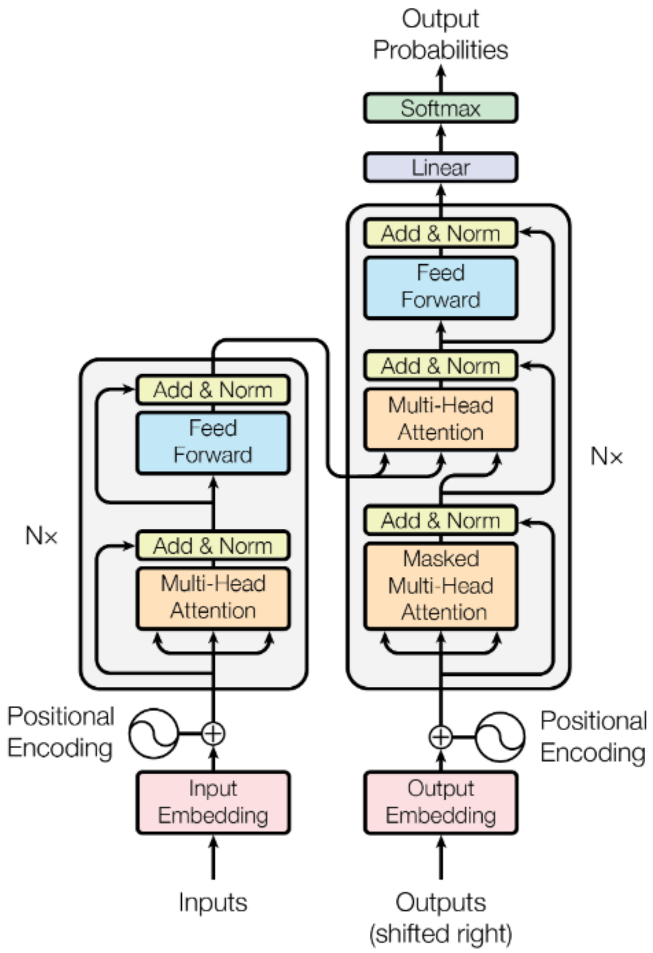


Figure 1: The Transformer - model architecture.

חישוב הדיוק של המודל עבור המשפטים

אחד האתגרים שיש בתחום ה-NLP הוא להעריך את טיב הפלטים. כאן, בתת-תחום של NLP והוא ה-Image Captioning, הפלטים הם המשפטים המתארים את התמונות. קשה להעריך אותם מכיוון שראשית, לא מדובר במספר טהור אותו המודל מנסה לחזות, או בחירה של class אחד מתוך כמה אפשריים כאשר בפועל יש רק תשובה אחת נכונה וברורה. שנית, במשימות כמו תרגום וכן Image Captioning, יכולות להיות תשובות רבות שנכונות בדיוק באותה מידה. ניתן לראות זאת למשל במערך הנתונים Flickr8K בו השתמשתי, שם לכל תמונה יש 5 תיאורים שונים הניתנו ע״י בני אדם, ושוב, כולם נכונים וטובים באותה מידה בדיוק.

לשם כך קיים אלגוריתם הערכה בשם **BLEU** (Bilingual Evaluation Understudy) שמטרתו היא, בהינתן הטקסט שמודל כלשהו הפיק לקלט מסוים, כמו גם משפטים נכונים עבור אותו הקלט, לחשב באופן מיידי הערכה מספרית לטקסט שיוצר. במקרה הנ״ל הפלט הוא משפט, ולצורך הסימון נקרא לו MT (Machine Translation), ולכל אחד ממשפטי האמת נקרא Reference 1, Reference 2 וכו׳.
נתחיל בדוגמא לשני משפטי אמת עבור תמונה מסוימת:

Reference 1 = The cat is on the mat.
Reference 2 = There is a cat on the mat.

בבסיס הרעיוני של מדידת דיוק למשפטים, נרצה לבדוק האם כל אחת ממילות המשפט שאותו בודקים מופיעה גם באחד ממשפטי האמת. ואולם, קל לזהות את הבעייתיות שבדרך זו, שכן בעבור:
MT = the the the the the the.

הציון יהיה לכאורה 6/6 כיוון שכל אחת מהמילים בפלט אכן מופיעה במשפטי האמת.

לכן, השלב הראשון ב-BLEU הוא להגדיר תגמול:
בעבור כל מילה במשפט ה-MT, נתגמל אותה <u>עד</u> מקסימום ההופעות של אותה מילה באחד ממשפטי האמת. כלומר, עבור the למשל, היא מופיעה פעמיים במשפט הראשון ופעם אחת בשני. לכן, המקסימום תגמול שה-MT יקבל בעבור מילה זו הוא 2. כך הערכת התוצאה כבר יורדת ל- 2/6. המינוח עבור ההופעות בקלט נקרא Count, והערך המקסימלי (כאן הוא היה 2) נקרא $Count_{clip}$.

ועדיין, גם דרך זו אינה מספיקה, שכן אין התייחסות למיקום המילים, ומשפט כמו:
MT = Is mat cat the on the

יקבל את מלוא הנקודות, 6/6.

כאן מגיע הקונספט של **N-grams**. מה שעשינו עד כה היה בעצם unigram (1-gram) כי השוונו כל פעם מילה אחת. כלומר, n-gram היא למעשה השוואת רצפים של n מילים מתוך משפט ה-MT. כך, ניתן לקבל הערכת משפט מדויקת ביותר כשלוקחים בחשבון גם biagram (2), triagram (3) וכן הלאה.

נראה דוגמא לחישוב הדיוק עבור biagram (2-gram):

Reference 1 = The cat is on the mat.

Reference 2 = There is a cat on the mat.

MT Output = The cat the cat on the mat.

בתור התחלה נסתכל על כל הזוגות האפשריים וכן על כמות הופעתן במשפטים:

Pair (in MT)	Count (in MT)	$Count_{clip}$
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1

ועכשיו איך מחשבים את הציון הסופי?

יש סה"כ 6 הופעות (Count) ו-4 מקסימום הופעות ($Count_{clip}$), לכן הציון הוא 4/6.

ובכן, אופן פעולה זה זהה עבור כל n שנבחר. ניתן לסכום אופי פעולה בנוסחה הבאה:

$$P_n = \frac{\sum_{n\text{-grams} \in \hat{y}} Count_{clip}(n\text{-gram})}{\sum_{n\text{-grams} \in \hat{y}} Count(n\text{-gram})}$$

זה כאמור חישוב הציון עבור n כלשהו. נרצה נוסחה נוספת לחישוב הציון הכולל של כל ה-n-grams. לשם כך מחשבים את כולם, סוכמים, ומחשבים את הממוצע. הנוסחה הסופית בעבור k n-grams היא:

$$BLEU = BP \cdot e^{\left(\frac{1}{k}\sum_{n=1}^{k} Pn\right)}$$

כפי שניתן לראות, מעלים את e בחזקת אותו ממוצע.

האלמנט האחרון בנוסחה שטרם התייחסנו אליו הוא ה-BP. משמעות ה-BP היא Brevity Penalty, או בתרגום חופשי "עונש קיצור". הסיבה שמוסיפים את העונש הזה לציון הוא משום שלמחשב קל יותר לקבל ציון גבוה ע"י משפטים קצרים. זה די אינטואיטיבי, שהרי אם למשל נפיק את המשפט כ- "the" בלבד (עבור הדוגמאות ממקודם), נקבל ציון מקסימלי. ה-BP מחושב באופן הבא:

אם אורך ה-MT גדול מאורך המשפט המקורי (לוקחים את הקצר ביותר): $BP = 1$

בכל מקרה אחר: $BP = e^{\left(1 - \frac{Ref\ Length}{MT\ Length}\right)}$

העמקה בקונבולוציה

רשתות קונבולוציה

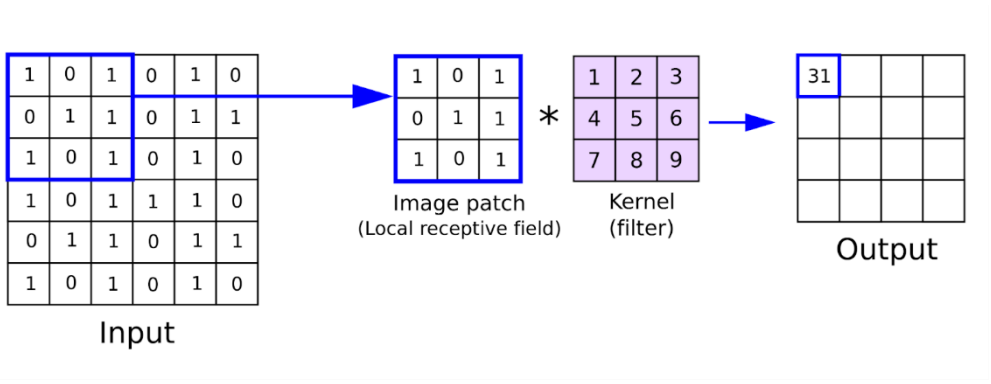
רשתות ה-CNN (Convolutional Neural Networks) הן רשתות פורצות דרך בתחום הלמידה העמוקה. רשתות אלו משתמשות בפעולת הקונבולוציה במקום כפל מטריצות בחלק רב מהשכבות. הרשתות משמשות בעיקר לעיבוד תמונות וראייה ממוחשבת (Computer Vision), אך יש בהן שימושים גם במערכות המלצה, עיבוד שפה טבעית ועוד. ליבת הרשתות מבוססת על שכבות קונבולוציה, כאשר ההשראה מגיעה מתהליכים ביולוגיים במוח המנסים להתחקות אחר דפוס חיבור הנוירונים.

שכבת קונבולוציה סטנדרטית

בכלליות, שכבות הקונבולוציה מקבלות קלט מהשכבות הקודמות, מבצעות עליו את פעולת הקונבולוציה ומעבירות את הפלט הלאה לשכבות הבאות.

הקלט והפלט של שכבה זו בעלי 4 מימדים (נקראים גם טנזור): מספר הקלטים, גובה הקלט, רוחב הקלט ומספר ערוצי הקלט. השכבה מקבלת את הקלט ומוציאה פלט חדש שישמש כקלט עבור השכבת הבאה. כל נוירון בשכבות מן הסוג הזה מעבד את המידע רק בעבור ה-Receptive Field שלו, בדומה למנגנון העצבי במוח, כאמור, בניסיון להתחקות אחריו. פעולה הקונבולוציה מתבצעת בעזרת מטריצה רבת מימדים הנקראת פילטר / kernel שערכיה נלמדים לאורך אימון הרשת, אשר מוכפלת במטריצה הנקלטת וכתוצאה מכך יוצרת פלט חדש שמטרתו לזהות דפוסים בקלט, לדוגמא קווים אנכיים ואופקיים בתמונה.

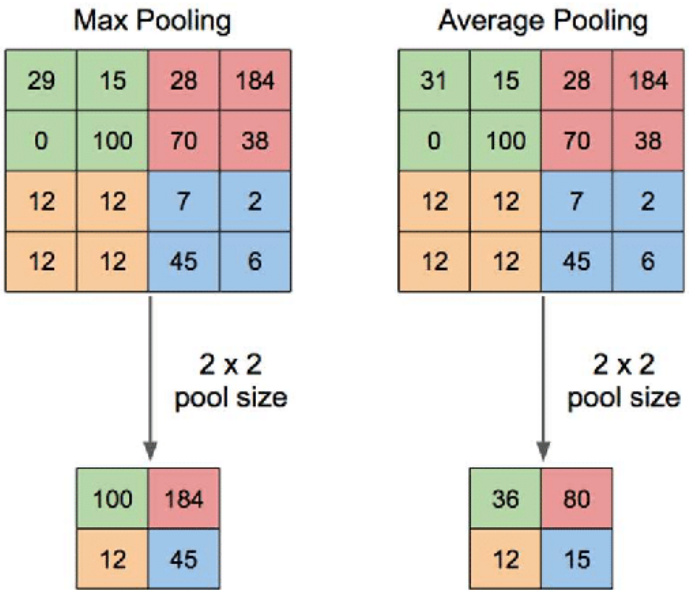
יתרון בולט שיש לשכבות הנ"ל הוא שביכולתן לשנות (השימוש הוא בעיקר לצורך הקטנה) את מספר הפרמטרים ברשת הכוללת, ובכך לאפשר לרשת להיות עמוקה בהרבה. למשל, שכבת Fully Connected לתמונה בגודל זעום של 100x100 תייצר כ-10,000 משקלים חדשים עבור כל נוירון בשכבה הבאה, מה שמהווה נטל מבחינת כוח החישוב, במיוחד כאשר הרשת סבוכה. לעומת זאת, שכבת קונבולוציה פשוטה בגודל 5x5 תייצר כ-25 משקלים נלמדים בלבד. חיסכון חישובי זה מתאפשר לנוכח העובדה שלא כל איבר באמת משפיע מבחינת חוזק הקשר שלו, למשל פיקסלים מרוחקים בתמונה הנמצאים בשוליים והשפעתם זניחה מאוד. שכבות הקונבולוציה מיישמות את מסקנה זו בכך שאינן מקשרות בין כל שני נוירונים אלא בין איברים קרובים (clusters).



פעולת הקונבולוציה.

שכבת Pooling

בכלליות, שכבות Pooling נועדות בכדי להקטין את גודל הטנזור במהלך מעבר המידע ברשת. סוגי ה-Pooling הנפוצים ביותר הם Average ו-Max. אין בשכבות אלה משקלים נלמדים, אלא רק שני פרמטרים f (גודל הפילטר) ו-s (ה"קפיצה" שעושים שלמעשה מחלקת את הקלט לאזורים השונים).
בדוגמא שמשמאל ניתן לראות דוגמא ובה f ו-s שווים 2 כ"א.



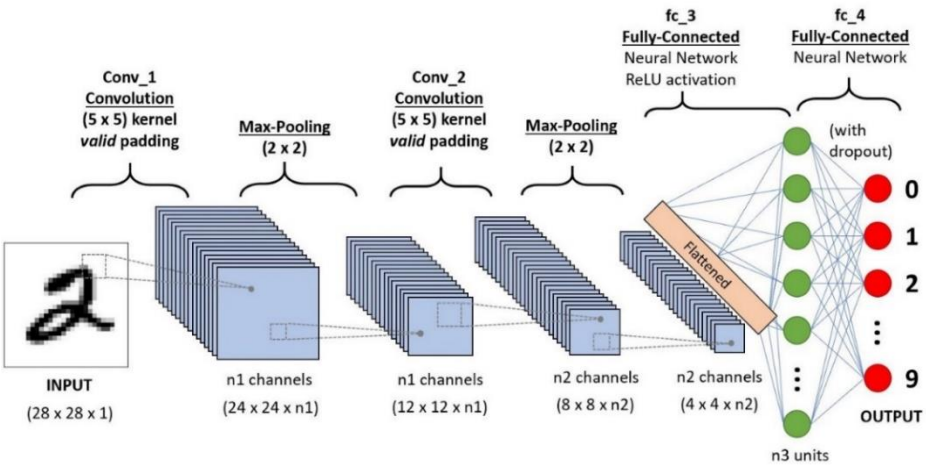
ה-Average, בהסתמך על כך שמידע מרחבי כגון תמונות מאופיין באיברים (במקרה הזה פיקסלים) קרובים הדומים אחד לשני, מאפשר "להתעלם" מכל אחד מהם בנפרד ע"י לקיחת הממוצע שלהם. לעומת זאת, ה-Max לוקח את הערך המקסימלי מכל תחום בחלוקה, במטרה להדגיש ולהעצים את האיברים הבולטים יותר בקלט.

במודל הנוכחי יש שימוש נרחב ב-Average Pooling מכיוון שמדובר בתמונות, ובהן כאמור שימוש ב-Average מקל על סיבוכיות המודל ומאפשר לו להיות עמוק יותר.

שכבת Fully Connected

בכלליות, שכבות Fully Connected מחברות כל נוירון בשכבה מסוימת לכל נוירון בשכבה אחרת, ולכן משמשות לרוב בשכבות האחרונות ברשת לצורך הפלט.

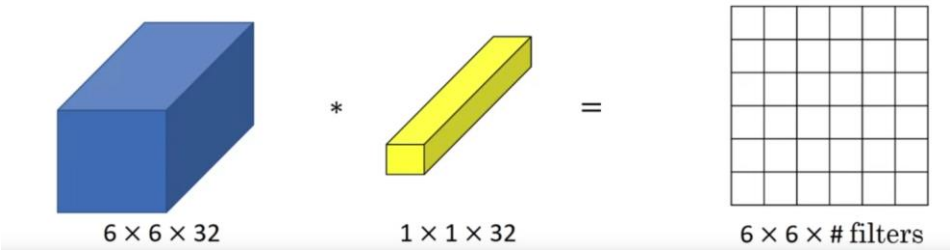
דוגמא לרשת פשוטה לזיהוי ספרות בכתב יד המכילה את כל השכבות הנ"ל.
תחילה, מתקבלת תמונה במימדים של 28x28x1 (כיוון שהתמונה בשחור לבן), ולאחריה שתי שכבות קונבולוציה מלוות ב-Max Pooling כאשר לבסוף שתי שכבות Fully Connected לצורך פליטת ניחוש הספרה. ניתן גם לראות את השינוי בגודל הטנזור לאורך הרשת.



קונבולוציה 1x1

קונבולוציה עם פילטר בגודל 1x1 נשמעת לא טריוויאלית, שהרי מדובר בהכפלה פשוטה של כל הערכים בקלט ותו לא. זה אכן המקרה כאשר מספר הערוצים הוא 1, אך המצב נעשה מעניין כאשר מספר הערוצים גבוה יותר:

נראה דוגמא למקרה בו הקלט הוא בגודל 6x6x32. מספר הערוצים בפילטר חייב להיות תואם למספר הערוצים בקלט, כלומר 32. הפילטר לוקח את כל אחת מ-36 הקומבינציות בקלט, ומבצע כפל איברים בין כל 32 המספרים המתאימים בקלט ל-32 המספרים שבו. לבסוף הוא מפעיל את פונקציית האקטיבציה ReLU על התוצאה. ניתן לדמות את הפילטר לנוירון בודד המקבל כקלט 32 מספרים, מכפיל אותם ב-32 משקלים השמורים בו ולבסוף ReLU. במידה ויש אף יותר מפילטר אחד, זו מעין שכבת נוירונים של ממש.

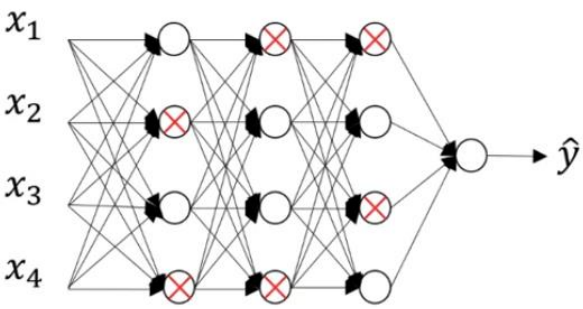


הפלט בדוגמא הוא בגודל 6x6x#filters. ה-6x6 נשמר לפי הקלט המקורי, ומספר הערוצים נקבע לפי מספר הפילטרים, שכן כל פילטר מייצר שכבה אחת כתוצאה מהחישובים.

מדוע בכלל משתמשים בקונבולצייית 1x1? ובכן, היא שימושית מאוד כאשר רוצים לשנות את גודל הקלטים, וספציפית את מספר הערוצים (את הגובה והרוחב ניתן לשנות עם שכבת Pooling) בשכבות השונות מבלי לצרוך כמעט כוח חישוב. לדוגמא, אם הגענו למצב שיש שכבה שבה הקלט הגיע למימדים של 28x28x192, וברצוננו להקטין את מספר הערוצים הרב, נבצע קונבולוציית 1x1 עם פילטר של 1x1x32, שיקטין את מספר הערוצים ל-32.

Dropout

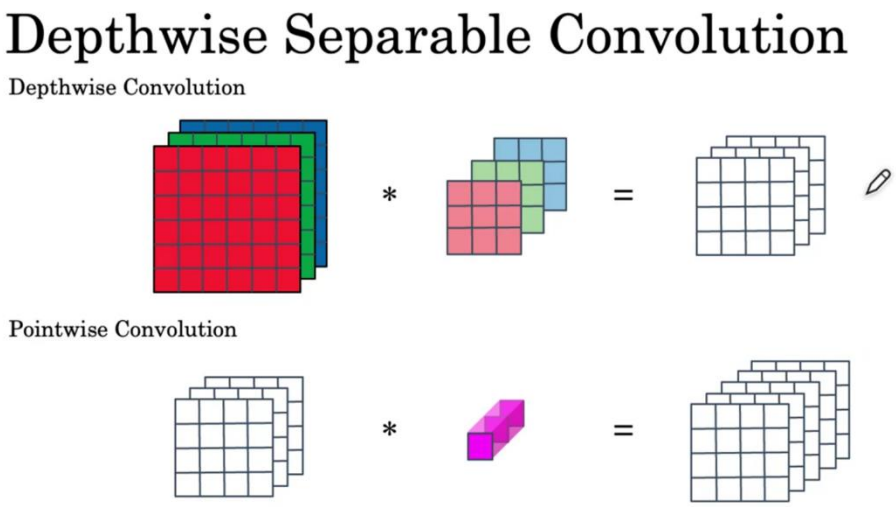
במטרה להקטין את בעיית ה-Overfit, ישנה טכניקה בשם Dropout שתפקידה היא אכן לצמצם ו/או למנוע לחלוטין את בעיה זו. מעורב בה פרמטר אחד בשם Dropout Rate, והוא קובע את עוצמת הצמצום של הרשת כלפי ה-Overfit, נקרא לו dr. אופן הפעולה של ה-Dropout מאוד פשוט: עבור כל נוירון בשכבות השונות (ניתן להגדיר שכבות בהן יהיה / לא יהיה Dropout), ישנו dr% סיכוי שהוא יימחק כליל ובאופן רנדומלי מהשכבה בה הוא נמצא. מחיקה זו מתרחשת רק בשלב האימון ולא בשלב ה-prediction.
מדוע זה עובד? ניתן להבין זאת בין השאר ע"י הסתכלות על שני מצבים אפשריים עבור נוירון כלשהו שנמחק: אם הוא "נוירון רע", כלומר לא מועיל הרבה לרשת, ולכן מחיקה שלו מועילה, מכיוון שהרשת מזהה ביתר קלות את הנוירונים החשובים. לעומת זאת, אם הוא "נוירון טוב", כלומר מוביל הרבה מידע חשוב, היעדר שלו ייאלץ את הרשת לשפר את הנוירונים האחרים ולגרום גם להם להעביר מידע חשוב.



בתמונה – דוגמא ל-dr = 0.5.

שכבת Depthwise Seperable Convolution (Dwise)

שכבה זו היא אחת מאבני היסוד לרשתות הקונבולוציה פורצות הדרך מסוג MobileNet. הרעיון המרכזי העומד מאחורי MobileNet הינו לאפשר הרצה של מודלים מורכבים גם במכשירי קצה בעלי יכולת חישובית נמוכה, דוגמת מכשירים ניידים, ומכאן השם MobileNet. קונבולוציה רגילה, על אף שפתרה בעיות סיבוכיות רבות, מאותגרת שוב בניסיון להשתמש במודלים הסבוכים באותם מכשירי קצה בעלי כוח חישוב וזיכרון לא גבוהים.



סוג זה של קונבולוציה מורכב משני חלקים. הראשון, Depthwise, במקום לקחת את כל הפילטר במימדיו המלאים ולבצע את כל החישובים על התמונה, לוקח פילטר באותם מימדים, אך מבצע את החישובים בנפרד עבור כל זוג מימדים, אחד בפילטר והשני מהתמונה. משמאל ניתן לראות הדגמה לכך – השכבה האדומה בתמונה בלבד עם השכבה האדומה בפילטר בלבד, וזו יוצרת שכבה אחת בפלט, וכך הלאה בירוק ובכחול. ואומנם, גודל הפלט עדיין קצר ב-2 מימדים במקרה הנ"ל מהגודל הרצוי המתקבל מקונבולוציה רגילה. כאן מגיע השלב השני והוא שלב ה-Pointwise. בשלב זה, מבצעים קונבולוציה על הפלט שהתקבל מהשלב הקודם, עם ה-"Point" בגודל 1x1x#channels (בצבע ורוד). ואומנם, קונבולוציה זו תייצר פלט חד מימדי בלבד, ולכן למעשה יש מספר Points (ולא אחד כמו בתמונה) השווה למספר הראשוני הקרוב ביותר הגדול ממספר הערוצים. וכך לבסוף מקבלים את הפלט הרצוי במספר נמוך בהרבה של חישובים.

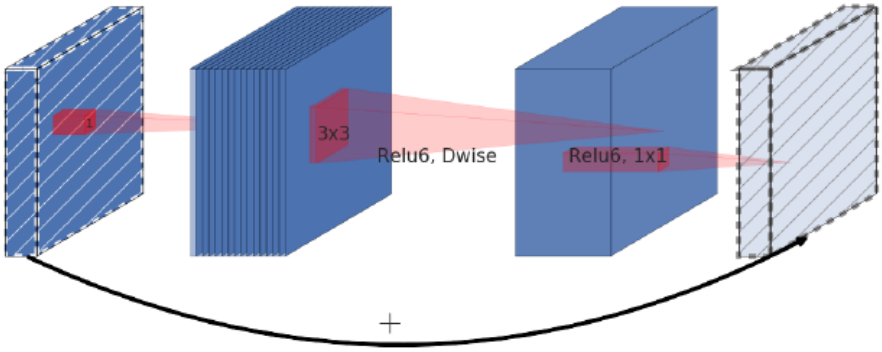
בנוגע לסיבוכיות החישובית, ניקח לדוגמא מקרה של קלט בגודל 6x6x3 עם פילטר בגודל 3x3x3.
בקונבולוציה רגילה, הדבר "יעלה לנו" 2,160 חישובים, בעוד ב-Dwise Conv 672 בלבד, כלומר שיפור של כ-70% ביעילות. במבט כללי יותר, החיסכון החישובי תלוי במספר הערוצים ובגודל הפילטר, ומחושב לפי הנוסחה הבאה:

$$\frac{1}{n_c'} + \frac{1}{f^2}$$

כך שאם לדוגמא היה לנו גודל פילטר של 3 וכן 3 ערוצים, החישוב הוא $\frac{1}{5}+\frac{1}{9}$ (5 – המספר הראשוני הקרוב ביותר הגדול מ-3), אשר שווה ל-0.311, כלומר כ-30% סך חישובים מהקונבולוציה הרגילה.

שכבת MBconv

משמעות השם MBconv היא למעשה Mobile Inverted Bottleneck Convolution. במודל הנוכחי, יש שימוש נרחב מאוד בסוג שכבה זו.



כל שכבה כזו מורכבת מבלוק מסוג Residual, כלומר קיים קשר ישיר בין האיבר הראשון והאחרון בשכבה.

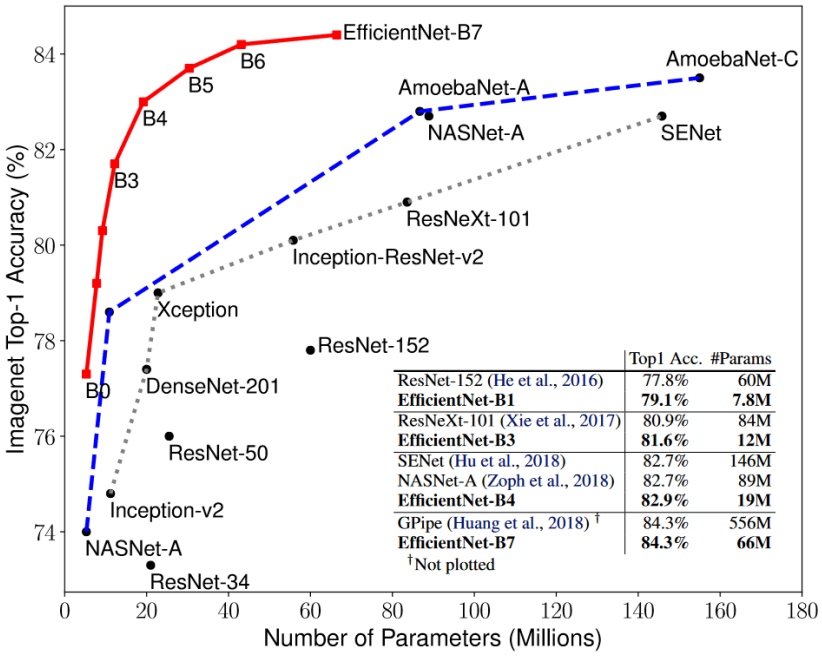
ראשית, מבצעים על הקלט קונבולוציית 1x1 בכדי להגדיל את מספר הערוצים ביחס מסוים (במודל הנוכחי ביחס של 1 ל-6).
לאחר מכן, מבצעים קונבולוציית Depthwise עם פונקציית אקטיבציה מסוימת (במודל הנוכחי השימוש הוא ב-SiLU), ולבסוף קונבולוציית 1x1 נוספת, הפעם כדי להקטין חזרה את מספר הערוצים לכפי שהיה בהתחלה מהקלט. כאמור, יש שימוש גם ב-Skip Connection בין הקלט לפלט השכבה.

מודל ה-EfficientNet

רשתות ה-MobileNet היוו פריצת דרך בשיפור הסיבוכיות ברשתות הקונבולוציה. כעת, נשאלת השאלה כיצד ניתן להתאים את הרשתות הללו כך שיעבדו בצורה מיטבית לצרכים ספציפיים ובהינתן מגבלות ספציפיות. למשל, אופן פעולת הרשתות על מכשירים ניידים מתוצרת חברות שונות היא שונה (כוח חישוב, נפח אחסון וכו'), וכן מטרות שונות דוגמת זיהוי אובייקטים, סיווג תמונות ועוד. במילים אחרות, לפרויקטים שונים יש צרכים שונים, שכולם דורשים שינויים, גם אם מינוריים, מרשתות ה-MobileNet ומרשתות באופן כללי. רשת ה-EfficientNet באה כדי לנסות ולפתור את השיקולים הללו בצורה אוטומטית.

הוגי רשת זו מציגים בתור התחלה את שלושת הפרמטרים המרכזיים שאותם ניתן לשנות בכדי "לפשט" / "לסבך" את הרשת, והם רזולוציית התמונה (מסומנת באות r), מספר השכבות ברשת (d) וגודל כל אחת מהשכבות עצמן (w). השאלה הנדונה היא, בהינתן מגבלות חישוביות מסוימות, ו/או צרכים אחרים כלשהם, מהם השינויים המיטביים שיש לעשות עבור כל אחד משלושת הפרמטרים? עד כה, לא היה עיסוק משמעותי בנושא של יחסי הגומלין בין השלושה, וכך רשתות רבות הגיעו ל-Plateau ולא רשמו התקדמות כלל לאחר שלב מסוים. הדבר יצר סוגים רבים של רשתות חדשות עם ארכיטקטורות חדשות. בכך ה-EfficientNet מייחדת את עצמה מן האחרות, בכך שהיא מנסה לשפר את הקיים בצורה מיטבית, ולא להמציא ארכיטקטורה מהפכנית.

להלן גרף המתאר הישגי רשתות על מערך הנתונים ImageNet. באדום ניתן לראות את ביצועי ה-EfficientNet.



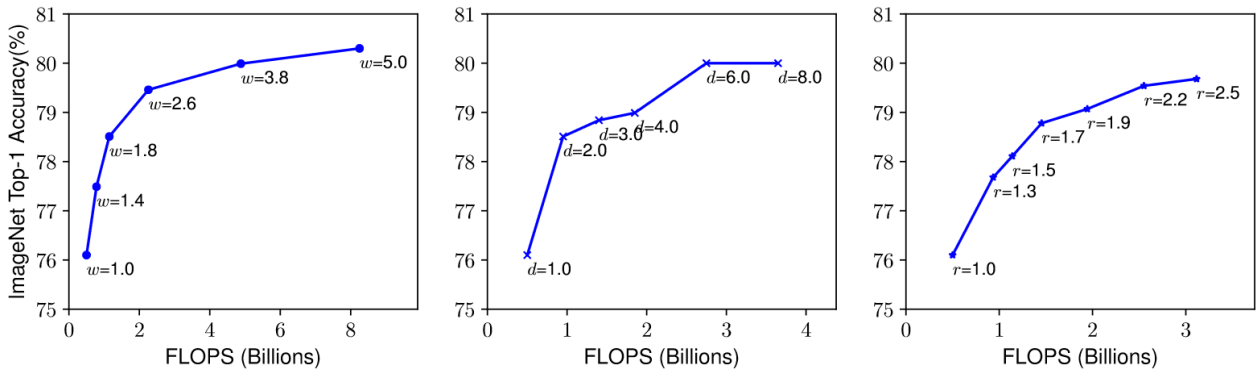
הנתונים מרשימים למדי, שכן לא רק שה-EfficientNet-B7 בעלת אחוזי הדיוק הטובי ביותר, יש לה גם משמעותית פחות משקלים מלרשתות הטובות האחרות.

מסקנת החוקרים היא שיש לשנות כל פרמטר בצורה מאוזנת ע"י יחס קבוע. למשל, אם נרצה להגביר את כמות החישובים ב-2^N כלשהו, נגדיל כל אחד מהפרמטרים במספר קבוע משלו:

α^N עבור גודל הרשת, β^N עבור גודל כל אחת מהשכבות ו- γ^N עבור הרזולוציה.

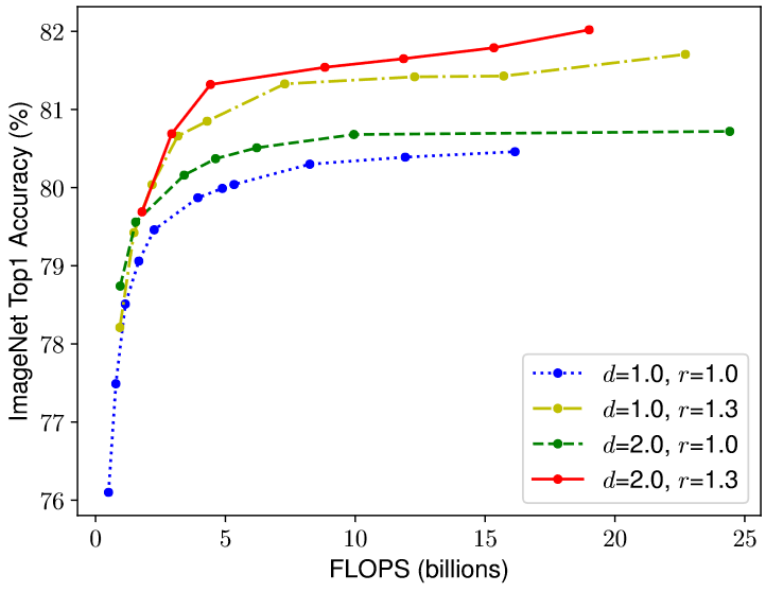
לאחר מחקר רב הגיעו החוקרים לאותם מספרים קבועים אשר עובדים בצורה מיטבית ומאפשרים כאמור להתאים את הרשת לצרכי הפרויקט בקלות ועדיין לקבל ביצועים טובים.

בתור התחלה, החוקרים בדקו בפשטות מה קורה כאשר מגדילים כל אחד מ-d, w ו-r, כפי שעושים בכל רשת על מנת לשפר את ביצועיה: (FLOPS = יחידת מידה למס' פעולות חישוב)



אין זה מפתיע שנרשמו עליות משמעותיות באחוזי הדיוק ככל שהפרמטרים גדלו, ומן הסתם גם מספר החישובים, שכן שיפורים אלה בהכרח משפרים ביצועים של מודלים עד לרמה מסוימת. ואומנם, כפי שניתן לראות בבירור בכל אחד משלושת הגרפים, בשלב מסוים מפסיק להיות שיפור בביצועים.

מסקנותיהן של החוקרים הן שלמעשה, בצורה די אינטואיטיבית, יש קשר בין שלושת הפרמטרים ולא נכון להתייחס לכל אחד בנפרד. למשל, הם ציינו שככל שהרזולוציה (r) גבוהה יותר, יש להגדיל גם את עומק הרשת (d). הם הגיעו למסקנה זו בעזרת הגרף הבא:



ניתן לראות בגרף שילובים שונים בעבור r ו-d ואת אחוז הדיוק שלהם ביחס לכמות החישובים.

בהינתן r כלשהו, בהסתמך על הגרף, ברור כי הגדלת d ביחס ל-r תורם משמעותית לשיפור אחוזי הדיוק – גם כש-d נמוך יותר!
אם נשווה לדוגמא בין הגרף הירוק והצהוב, נראה שבירוק יש d גבוה פי 2 מ-d בצהוב, ואילו r נמוך ב-0.3.
התוצאה – הגרף הצהוב מצביע על אחוזי דיוק טובים בהרבה.
נתון מרשים נוסף הוא העובדה כי דווקא הגרף האדום, עם d ו-r הכי גבוהים מבין הגרפים, דורש פחות כוח חישוב מאשר הגרף הצהוב והירוק.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

זהו המבנה הבסיסי והנפוץ ביותר המרכיב את ארכיטקטורת ה-EfficientNet (הדוגמא היא של B0).
הרשת הבסיסית ביותר, ה-B0, מכילה 238 שכבות וכ-4 מיליון משקלים. ערכים אלה עולים בהדרגה עד לרשת המורכבת ביותר והיא ה-B7, המכילה 814 שכבות וכ-64 מיליון משקלים.

כעת נשאלת שאלה חשובה – כיצד החוקרים יכולים לקבוע שהיחסים בין הפרמטרים הם הגורם המשפיע ביותר, ולא הארכיטקטורה החדשה שלו?

ובכן, בכדי לענות על שאלה זו, השוו החוקרים את כל אחת מסוגי רשתות ה-EfficientNet, מ-0 עד 7, לרשתות מובילות נוספות אשר הגיעו לאותם אחוזי דיוק של סוג רשת ה-EfficientNet, ובדקו את ביצועיהן לפני (עמודה שמאלית) ואחרי ה-tuning הייחודי בעזרת אותם יחסים בין שלושת הפרמטרים (עמודה ימנית). התוצאות:

EfficientNet-B0	**77.1%**	**93.3%**
ResNet-50 (He et al., 2016)	76.0%	93.0%
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%
EfficientNet-B1	**79.1%**	**94.4%**
ResNet-152 (He et al., 2016)	77.8%	93.8%
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%
Xception (Chollet, 2017)	79.0%	94.5%
EfficientNet-B2	**80.1%**	**94.9%**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%
EfficientNet-B3	**81.6%**	**95.7%**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%
PolyNet (Zhang et al., 2017)	81.3%	95.8%
EfficientNet-B4	**82.9%**	**96.4%**
SENet (Hu et al., 2018)	82.7%	96.2%
NASNet-A (Zoph et al., 2018)	82.7%	96.2%
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%
PNASNet (Liu et al., 2018)	82.9%	96.2%
EfficientNet-B5	**83.6%**	**96.7%**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%
EfficientNet-B6	**84.0%**	**96.8%**
EfficientNet-B7	**84.3%**	**97.0%**
GPipe (Huang et al., 2018)	84.3%	97.0%

ניתן לראות שהביצועים בכל המודלים השתפרו בצורה יוצאת דופן בזכות היישום של היחס בין הפרמטרים, מה שעונה על השאלה ומוכיח שעל אף שהארכיטקטורה טובה מאוד, השיפור כתוצאה ממנה בטל בשישים לעומת השיפור שנגרם כתוצאה מהיחס.

אז מה היחס בין שלושת הפרמטרים?

בטבלה מטה ניתן לראות דוגמא לשימוש ביחס, כאשר לקחו החוקרים את מודל ה-EfficientNetB0 הבסיסי, ובדקו אותו ארבע פעמים: בשלוש הפעמים הראשונות הגדילו משמעותית כל פרמטר בנפרד, ובפעם הרביעית הגדילו בצורה "חכמה" לפי היחס שמצאו. התוצאות הן ברורות לטובת הבדיקה הרביעית.

Model	FLOPS	Top-1 Acc.
Baseline model (EfficientNet-B0)	0.4B	77.3%
Scale model by depth (d=4)	1.8B	79.0%
Scale model by width (w=2)	1.8B	78.9%
Scale model by resolution (r=2)	1.9B	79.1%
Compound Scale (d=1.4, w=1.2, r=1.3)	**1.8B**	**81.1%**

התחלתי מלנסות את מודל ה-B0 כיוון שהוא בסיסי יותר ולוקח פחות זמן להרצה. לאחר מכן השתמשתי במודל ה-B7 כדי לקבל תוצאות משופרות יותר, אבל עוד על כך בהמשך. לשם השוואה, למודל ה-B0 יש 238 שכבות וכ-4 מיליון משקלים שנלמדו מראש, בעוד ל-B7 יש 814 שכבות וכ-64 מיליון משקלים!

בעצם, משתמשים במודל ה-CNN כ-Feature Extractor של התמונה, שלאחר מכן אותם פיצ'רים יועברו אל החלק הראשון ברשת ה-Transformers. במקרה הנוכחי, ה-Feature Extraction הוא פשוט לקחת את התמונות, למצוא בהן את החלקים החשובים והמרכזיים, וגם, כמובן, עצם העובדה שמחשבים לא מבינים תמונות אלא מספרים, להמיר אותם לפורמט מספרי שאיתו תוכל הרשת לעבוד.

פונקציית השגיאה

פונקציית השגיאה שבה השתמשתי היא SparseCategoricalCrossentropy. פונקציה זו היא אחת מהוריאציות של פונקציית הבסיס Cross Entropy Loss, שתפקידה לחשב את השגיאה המצטברת עבור Multi Class Classification. היא מחושבת באופן הבא:

$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

C מייצג את מספר ה-classes, t מייצג את הערך האמיתי, כלומר ה-ground truth אליו המודל שואף, f היא פונקציית האקטיבציה (למשל sigmoid או softmax) ו-s מייצג את הערך שפלט המודל בעבור אותו ה-class. במקרה הנוכחי אפשר להתייחס לכל מילה כאל class, שהרי המודל מפיק את התחזיות שלו בעבור כל מילה.

לאחר מכן, בכדי שהמודל יוכל ללמוד ולהשתפר, מחשבים את השיפועים (gradients) של השגיאה שהתקבלה ביחס לפלט שיצא מהמודל וכן ביחס לכל class בנפרד, ובאמצעותן מבצעים את החלחול לאחור המאפשר למודל ללמוד את המשקלים הרלוונטיים.

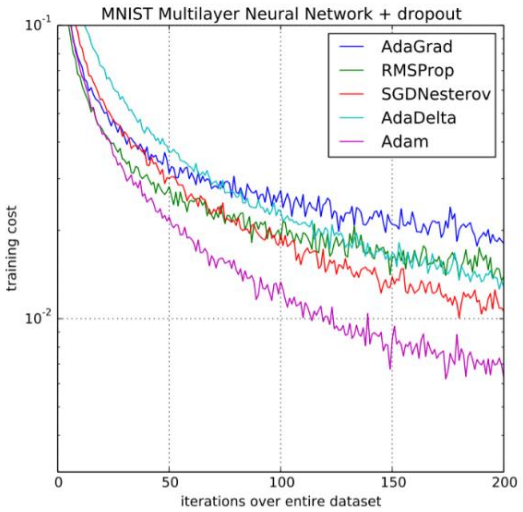
התוספת האחרונה היא השימוש ב-sparse. מה זה sparse?

sparse זה תהליך של כיווץ מטריצות והעלמת הערכים הלא-חשובים (ערכי ה-0), והוא נחוץ בעיקר כאשר יש מטריצות עצומות כמו כאן (לכל תמונה 5 ערכים תואמים, ויש אלפי תמונות). לביצוע התהליך, יוצרים שלושה וקטורים חדשים שירכיבו את המטריצה החדשה. הוקטור הראשון נקרא V והוא מכיל את כל הערכים שאינם 0 מהמטריצה המקורית. השני נקרא R והוא מייצג את מספרי השורות בהם נמצא כל ערך בהתאמה לוקטור V. השלישי נקרא C ומייצג את מספרי העמודות של אותם ערכים, גם הוא בהתאמה לוקטורים V ו-R כמובן. כך, נחסכים הרבה כוח חישוב וזיכרון.

כל אלה מרכיבים את פונקציית השגיאה, שמטרתה היא פשוט לחשב את השגיאה של הערך שהתקבל מהמודל כתוצאה מהמשפט שהפיק ביחס לכל חמשת הערכים המתקבלים מחמשת המשפטים התואמים.

אלגוריתם האופטימיזציה

אלגוריתם האופטימיזציה (Optimizer) בו השתמשתי הוא Adam (Adaptive Moment Estimation). אלגוריתם זה ידוע בתור אחד הטובים ביותר לאופטימיזציה של Gradient Descent משום שהוא יעיל, חסכוני ומהיר ביחס לאלגוריתמים אחרים, בייחוד כשהמודל מתעסק עם הרבה מידע ומכיל הרבה משקלים. Adam הוא שילוב של שני אלגוריתמים מוכרים אחרים, RMSprop ו-Momentum.



בנייה, אימון וניסוי של מודל Image Captioning ראשוני

(מודל זה מבוסס על כל שנאמר עד כה, כלומר בסיס קונבולוציה בדמות EfficientNet ועליו מודל ה-Transformer. הקוד נמצא בנספחים)

אימון המודל וניתוח התוצאות

לאחר אימון של 30 אפוקים (למעשה האימון נעצר לאחר 16 בלבד, מכיוון שיש שימוש ב-early stopping, במידה והמודל לא משתפר במשך 5 אפוקים רצופים), המודל הגיע לביצועים הבאים:

loss: 11.3046 - acc: 0.4852 - val_loss: 15.0575 - val_acc: 0.4135

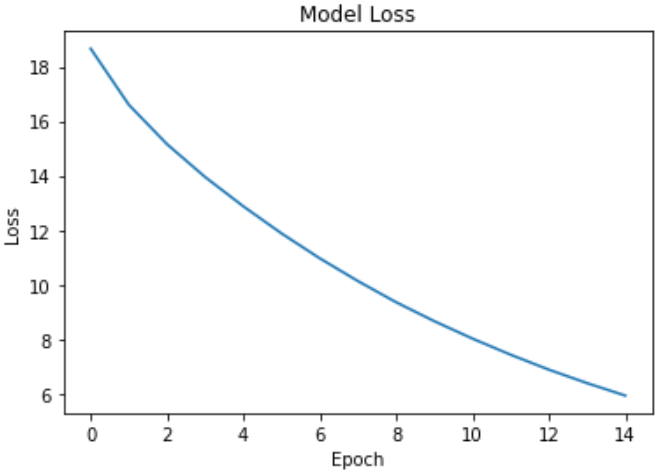
אחוזי דיוק סבירים (אם כי זה די ברור, בהתחשב בפשטות המודל).
נראה כמה דוגמאות לכיתוביות על תמונות שהבאתי:

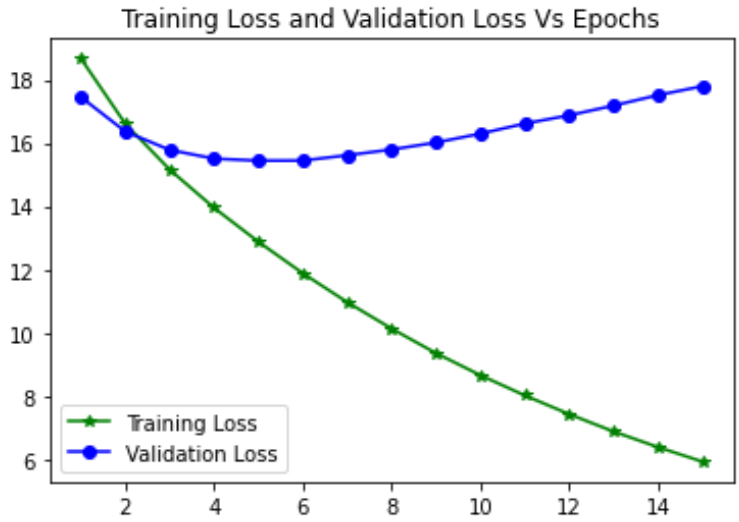
ניתן לראות שהמודל עושה ביצועים לא רעים, בעיקר בעצמים הגדולים והבולטים יותר שנמצאים בתמונות. לדוגמא, הוא מצליח לזהות כאשר מדובר באנשים, מכוניות, כלבים ואף התרחשויות בסיסיות (ככל הנראה בזכות האימון על ImageNet). אמנם, עדיין מדובר במשפטים שרובים לא תואמים את המציאות ואף לא הגיוניים מבחינה תחבירית. לא מן הנמנע יהיה לנסות וליצור מודל טוב יותר. לכן, אימנתי את אותו מודל של Image Captioning בהתבסס על EfficientNetB7, במקום B0. נשווה בין התוצאות:

B0: **loss: 11.3046 - acc: 0.4852 - val_loss: 15.0575 - val_acc: 0.4135**

B7: **loss: 5.9367 - acc: 0.6910 - val_loss: 17.7910 - val_acc: 0.3833**

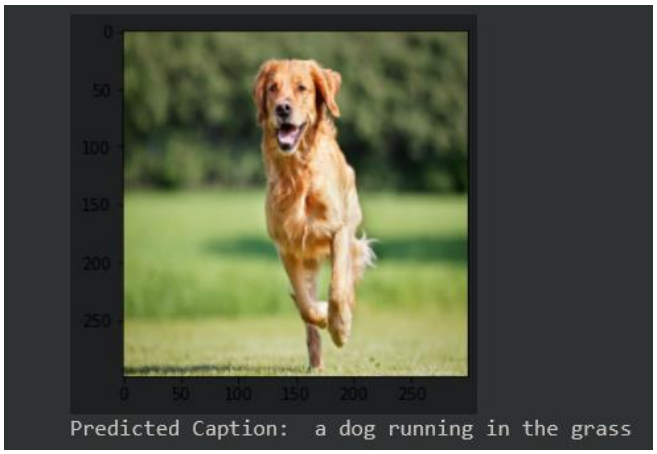
בפועל נראה שאין הרבה הבדל מבחינת מספרית. משמאל ניתן לראות את גרף השגיאה ביחס למספר האפוקים (הפעם היו 15). מה שנראה לכאורה כשיפור יפה, מתגלה כלא נכון כאשר מסתכלים על הגרף הבא, שמראה את השגיאה בסט האימון לעומת סט הבדיקה:





בעצם יש כאן Overfit גדול מאוד, ולא נרשם כמעט שיפור בדיוק על סט האימון. ההשערה המיידית שלי היא שמודל ה-B7 גדול ומסובך מדי עבור צרכי הפרויקט הנוכחי. השערה נוספת היא שיש לשכלל את מודל ה-Image Captioning ע״י הגדלת מספר הראשים (heads) ב-encoder וב-decoder בכדי לאפשר למודל מרחב תמרון רחב יותר להבין את התמונות והמשפטים.

לפני כן, נראה את המשפטים שהפיק המודל החדש על אותן תמונות ממקודם:



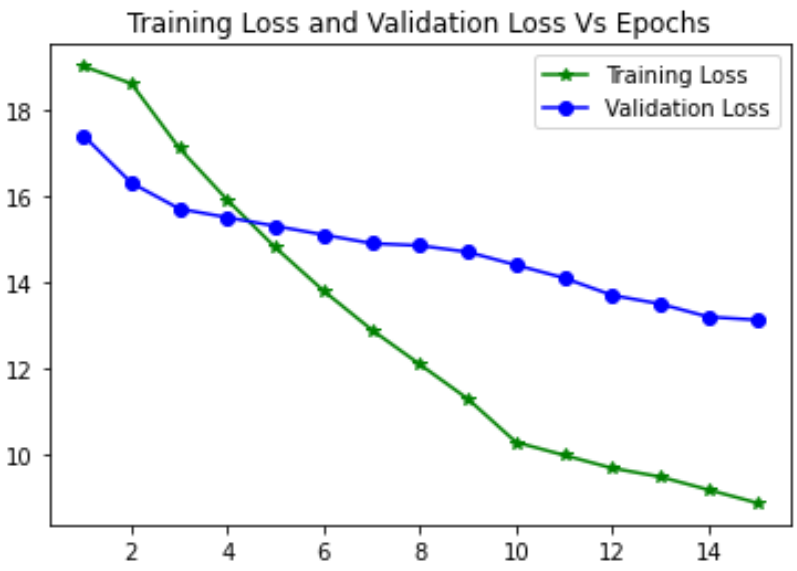
גם כאן ניתן לראות תוצאות דומות לתוצאות מהמודל הקודם מבחינת המורכבות והדיוק של התיאורים.

עבור הניסיון הנוסף בחרתי במודל ה-B3 הצנוע יותר, שמכיל 384 שכבות וכ-10 מיליון משקלים, וכן העלתי ב-10 את מספר הראשים הן ב-encoder (מ-10 ל-20) והן ב-decoder (מ-11 ל-21). התוצאות, לאחר 15 אפוקים, הן:

B0: **loss: 11.3046 - acc: 0.4852 - val_loss: 15.0575 - val_acc: 0.4135**

B7: **loss: 5.9367 - acc: 0.6910 - val_loss: 17.7910 - val_acc: 0.3833**

B3: : **loss: 8.4169 - acc: 0.6188 - val_loss: 13.1375 - val_acc: 0.4351**



ושוב גרף המציג את השיפור בשגיאה לאורך האימון. מדובר בשיפור יחסית משמעותי ביחס למודל ה-B7, אך לא טוב בהרבה מה-B0. כמו כן, עדיין יש Overfit גדול.

גם הפעם אין שינוי משמעותי במשפטים שהמודל מפיק עבור התמונות. מעט שיפור בתחביר ובזיהוי האובייקטים והסצנות שבתמונה אך לא יותר מכך.

דו"ח ריכוז ה-Hyperparameters

ה-Hyperparameters נשארו זהים גם בשינויים שביצעתי במודל*

Hyperparameter	Value
Input Shape	(299, 299, 3)
Optimizer	Adam
Learning Rate	0.001
Epochs	30
Loss Function	SparseCategoricalCrossentropy
Beta	0.9
Epsilon	1e-07
Dropout Rate	0.1
Batch Size	64
Vocabulary Size	10,000

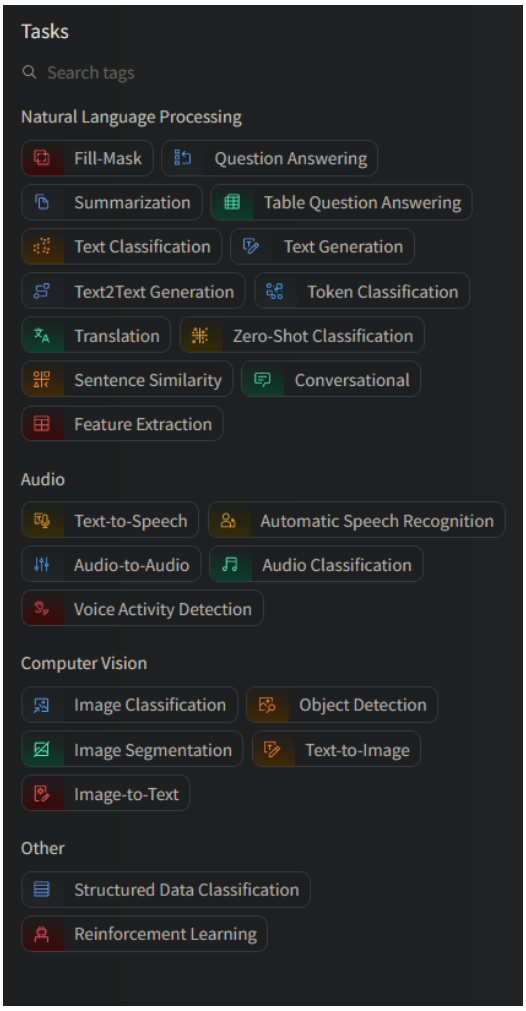
*חשוב לי לציין שבמודלים של Image Captioning שכאלה קשה יחסית למנוע Overfit, ומכיוון שהנושא אינו בתוכנית הלימודים היה לי קשה למצוא הסברים ופתרונות לעניין. פתרון אחד שכן ניסיתי הוא החלפת ה-Optimizer, מ-Adam ל-SGD, שכן Adam ידוע בנטיותיו ל-Overfit. אולם, זה לא עבד, מכיוון של-SGD לוקח זמן רב מאוד להתכנס והוא פחות התאים למסגרת המודל. בנוסף, ניסיתי להעלות את ה-Dropout Rate, אך גם זה לא הועיל לתוצאות הסופיות ול-Overfit.

בשלב זה החלטתי להמשיך במחקר ולנסות למצוא מודל טוב יותר שיענה על צרכי הפרויקט.

מודל Image Captioning מוכן ומתקדם

בהמלצת המורה המנחה, הופניתי לאתר "Hugging Face", עם הסלוגן " The AI community building the future".

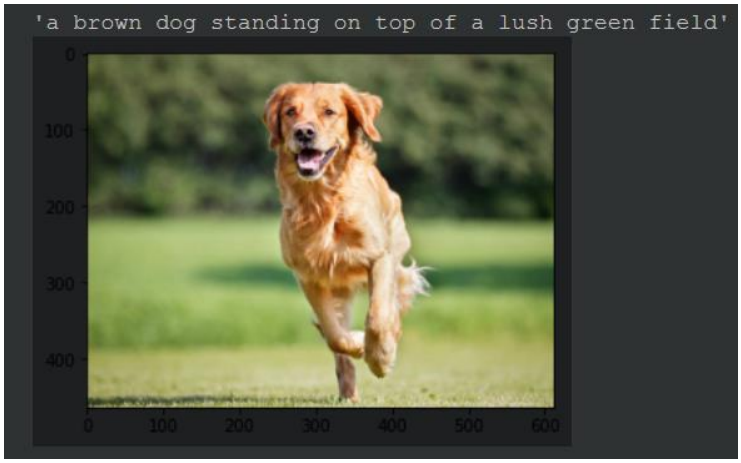
תכליתו המרכזית של האתר היא בנייה ושיתוף של מודלים מורכבים, מאגרי נתונים, קוד רלוונטי, מסמכים ולמידה, הכל אודות למידה עמוקה. משמאל ניתן לראות סרגל חיפוש ע"פ קטגוריות התחומים השונים. זהו אתר רחב מאוד, ידידותי ומעניין, המציע גם קורסים בלמידה עמוקה.



על מנת להתאים את צרכי הפרויקט הנוכחי, חיפשתי מודל הקשור ב-Image Captioning, ואכן נתקלתי באחד כזה.

המודל הנ"ל הינו מודל Image Captioning מתקדם ומורכב ביותר, השוקל קרוב ל-1GB, ומטרתו היא לפלוט משפט באנגלית המתאר כל תמונה הניתנת לו כקלט.

בכדי לבדוק את המודל וביצועיו, הורדתי אותו בעזרת ה- Google Colab וניסיתי אותו על שלל תמונות:



נראה שהמודל הנ״ל מצליח לייצר תיאורים מדויקים ביותר לתמונות וגם תחבירן מדויק. המודל נקרא "vit-gpt2-image-captioning", וכשמו מבוסס על GPT2, ראשי תיבות של Generative Pre-Trained Transformer 2. מדובר במודל למידה עמוקה מתקדם ביותר שיצא לאור בפברואר 2019 ע״י מעבדת המחקר OpenAI, ותכליתו הכללית היא עיסוק בטקסט: תרגום, מענה על שאלות, סיכום טקסטים ופליטת טקסטים ברמה הקרובה מאוד לרמה אנושית. מה שמיוחד ב-GPT2 היא בין השאר העובדה שהמודל לא אומן על אף אחד מהמשימות הנ״ל באופן ספציפי, אלא למד בצורה כללית יותר על כמויות אדירות של מידע מהאינטרנט, המקנה לו יכולות ורסטיליות ועוצמתיות להתמודדות עם טקסט מילולי. ה-GPT2 הינה למעשה רשת נוירונים במבנה של רשת Transformer יחד עם Attention, וזאת במקום GRU / RNN / LSTM הרווחים בשאר המודלים.

מכיוון שהמודל הגיע לביצועים מרשימים מאוד, החלטתי להשתמש בו באפליקציה.

מדריך למפתח

מודל ה-Image Captioning

את האפשרות להשתמש במודל כאמור לקחתי מאתר Hugging Face. כך נראה הקוד עם הסברים:

```
from PIL import Image  # ספריה לצורך עבודה עם תמונות

# ייבוא המודל ופונקציות הכרחיות נוספות
from transformers import VisionEncoderDecoderModel,
ViTFeatureExtractor, AutoTokenizer

# המודל עצמו
model =
VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-
gpt2-image-captioning")

# הפונקציה שעושה את חילוץ הפיצ'רים מהתמונה
# זאת בדומה לרשת הקונבולוציה במודל הידני
feature_extractor =
ViTFeatureExtractor.from_pretrained("nlpconnect/vit-gpt2-
image-captioning")

# אובייקט זה אחראי על לקחת את המספרים המייצגים את המשפט
# שיוצאים מהמודל כפלט ולהפוך אותם למילים
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-
gpt2-image-captioning")


max_length = 16  # הגדרת האורך המקסימלי של משפט


def predict_step(image_paths):
    # הקלט לפונקציה הוא רשימה של כתובות במחשב המפנות לתמונות
    # בקטע הקוד שמתחת עוברים כל כל האיברים ברשימה:
    # פותחים אותם כתמונות ומוודאים שהכל בפורמט RGB
    # זאת מכיוון שאלה הם מימדי הקלט של המודל
    # אם זה לא אר-ג'י-בי, הופכים את זה לכך
    images = []
    for image_path in image_paths:
        i_image = Image.open(image_path)
        if i_image.mode != "RGB":
            i_image = i_image.convert(mode="RGB")

        images.append(i_image)
```

```
    # החלק הזה הוא כמו מודל הקונבולוציה במודל הידני
    # feature extractor מעבירים את התמונות ל
    # בדומה למודל הקונבולוציה, תפקידו הוא להפוך את הפיקסלים
    # למספרים המייצגים את התמונה, וכן למצוא ולהדגיש את החשובים
    pixel_values = feature_extractor(images=images,
return_tensors="pt").pixel_values

    # כאן מעבירים את הפיצ'רים שהתקבלו למודל
    # המודל פולט רצפים של מספרים המייצגים את המשפטים
    output_ids = model.generate(pixel_values, max_length)

    # המודל כאמור פולט וקטור של מספרים שצריך להפוך למילים
    # בעזרת מתודה זו זה אכן מתבצע
    preds = tokenizer.batch_decode(output_ids,
skip_special_tokens=True)

    # ניקוי אחרון של המילים מרווחים מיותרים וכו'
    preds = [pred.strip() for pred in preds]

    # החזרת המשפט/ים
    return preds

# דוגמא לשימוש במודל ע"י הכנסת כתובת מהמחשב המפנה לתמונה
predict_step(['doctor.e16ba4e4.jpg'])
```

תרגום המשפט לעברית

לצורך הצגת המשפט בעברית, יש לתרגם אותו ישירות מפלט המודל באנגלית.

לשם כך, נעזרתי והשוויתי בין שני מקורות:

הראשון, Google Translate API בפייתון, ספרייה בשם googletrans.

השני, ביצוע Web Scraping על תרגום דרך אתר "מורפיקס", בעזרת הספריות beautifulsoup ו-requests.

להלן הקוד שכתבתי, עם הסברים:

בחלק הראשון, מבצעים את הייבואים נדרשים וכן מגדירים משתנה המכיל את המשפט באנגלית.

```
1   # pip prerequisites: beautifulsoup4, googletrans
2   from bs4 import BeautifulSoup
3   import requests
4   from googletrans import Translator
5
6   english = """
7   A white Honda car has stopped at the parking lot.
8   """
```

חלק שני: תרגום בעזרת מורפיקס. ראשית, יש לשים לב שכאשר מתרגמים משהו במורפיקס, לדוגמא "apple", כתובת ה-url נראית כך: https://www.morfix.co.il/en/apple

כלומר, כדי לקצר ולייעל את השימוש, נרצה לבצע את ה-Scraping "ישר" מתוך הכתובת המכילה את המשפט. שורות 11-13 עושות זאת. לאחר מכן, בעזרת requests.get, נקבל את כל המידע בפורמט ה-HTML, וממנו (שורות 15-17) נשלוף ונסדר את התרגום העברי.

```
10  # MORFIX
11  words = english.split()
12  exact_format = ''.join([words[i]+'%20' for i in range(len(words))])[:-3]
13  url = f'https://www.morfix.co.il/{exact_format}'
14  result = requests.get(url)
15  doc = BeautifulSoup(result.text, 'html.parser')
16  tag = doc.find_all(class_='MachineTranslation_divfootertop_enTohe')[0].decode_contents()
17  hebrew_mf = tag.strip()
```

חלק שלישי: תרגום ע"י Google Translate. זהו שימוש פשוט למדי, שכן זו המטרה היחידה והברורה.

```
# GOOGLE TRANSLATE
translator = Translator()
hebrew_gt = translator.translate(text=english, src='en', dest='he').text
```

כעת נותר לבדוק את איכות התרגומים, וכן את הזמן שלוקח לכל אופציה (בעזרת הספרייה time).

נתחיל עם מורפיקס:

```
MORFIX SAYS:
מכונית הונדה לבנה עצרה במגרש החניה.
TIME: 0.16023874282836914
```

נבצע את אותה פעולה בדיוק רק עם Google Translate:

```
GT SAYS:
מכונית הונדה לבנה נעצרה במגרש החניה.
TIME: 0.35251283645629883
```

מסקנות:

ראשית, לא ניתן להתעלם מהעובדה שהתרגום באמצעות מורפיקס היה מהיר יותר. בנוסף, התרגום היה מעט יותר מדויק. לכן, **החלטתי לבחור בתרגום דרך מורפיקס**.

שכלול נוסף

צורך נוסף שעלה במהלך הפרויקט הוא לכלול גם את פרטי הרכב (במידה ויש בתמונה) בתיאור. ראשית כל, עוד לפני שימוש ב-Reverse Image Search (ראו פרק נפרד), יש לבדוק את מחרוזת התרגום בהאם ישנה המילה "רכב" / "מכונית" וכו'. במידה ויש, מתבצעת קריאה לפונקציה של ה- Reverse Image Search אשר מחזירה את הפלטים מהם נרצה לקחת את המידע הרלוונטי, ולשלב אותו במשפט. הכל כאמור מתבצע בפונקצית התרגום. להלן הקוד עם הסברים בפנים:

הערה: החברה והדגם כתובים באנגלית ומשולבים במשפט העברי. אחרת שמות הדגמים וכו' לא מתורגמים כראוי.

```
for word in heb.split():  # בדיקה האם קיים "רכב" / "מכונית" וכו' במשפט שהתקבל
    if word in hebrew_terms:
        heb_car_type = word
        break

if heb_car_type:
    all_car_models = [...]
    car_model = ""   # כאן תוכנס שם החברה והדגם במידה ויש
    output = report(annotate('img.jpg'))   # Reverse Image Search

    for res in output:  # לולאה שבודקת כל תוצאה מהחיפוש - האם יש בה את אחת מחברות הרכב
        for car_model in all_car_models:
            if car_model.lower() in res.description.lower():
                car_model = res.description
                break
        if car_model:
            break

    if car_model != 'None':   # אם אכן נמצאה חברה
        # כאן מדובר בשורות טכניות לחלוטין שתפקידן להכניס את החברה במקום המתאים במשפט
        heb = heb.split()
        i = heb.index(heb_car_type)
        heb = ' '.join([heb[x] for x in range(i + 1)] + [car_model] + [heb[x] for x in range(i + 1, len(heb))])

return heb
```

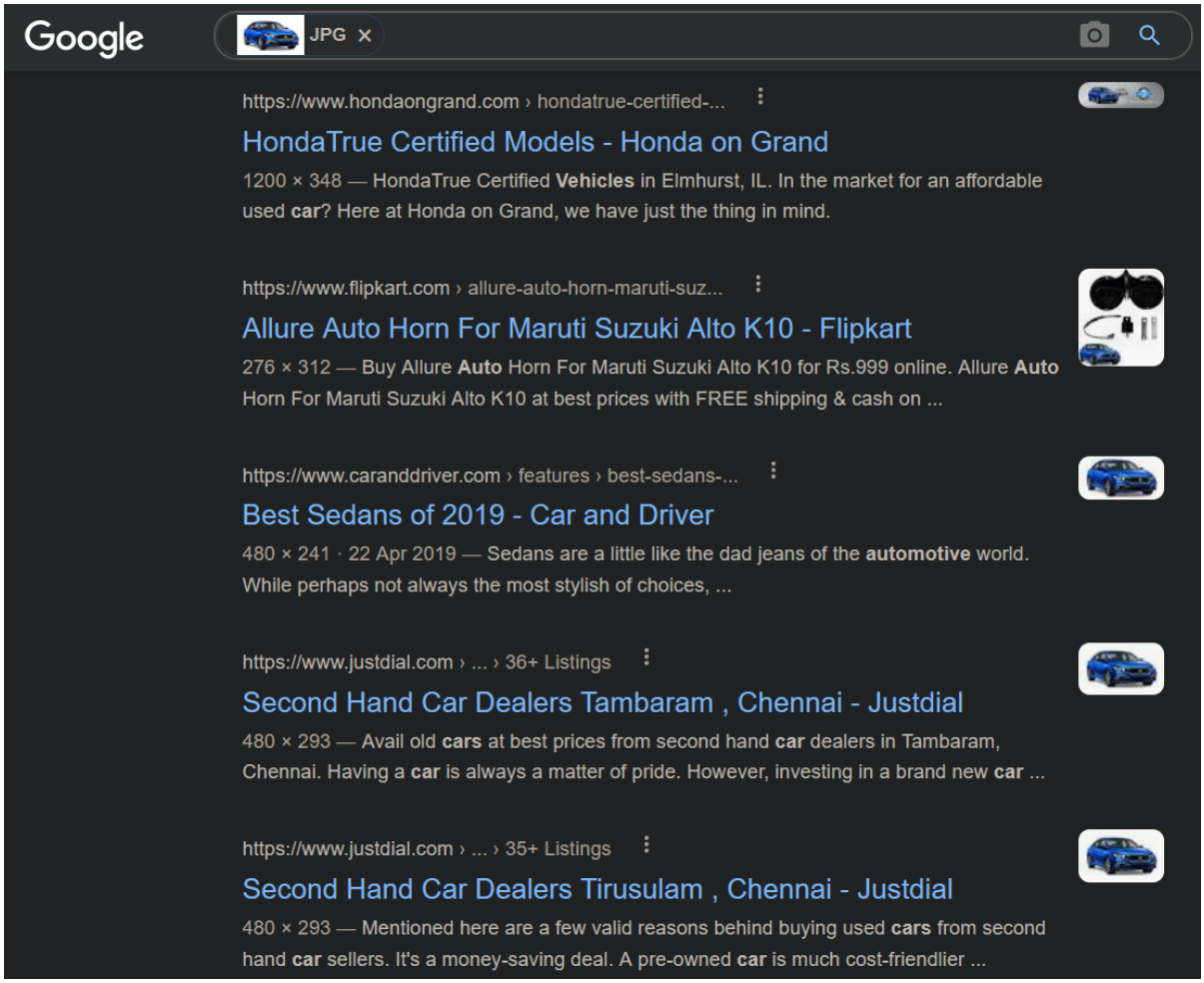
מידע נוסף עבור רכבים בעזרת Reverse Image Search ו-Web Scraping

כיום ישנן מיליארדי תמונות ברחבי האינטרנט. כל תמונה שאי פעם נתקלנו בה בעת גלישה נמצאת באינטרנט, ואפילו תמונות בחיים האמיתיים סביר שרובן נמצאות. במנועי החיפוש הקיימים, חיפוש תמונה הקשורה לטקסט זו פעולה קלה למדי, שהרי קיימת האפשרות לצפות באינספור תמונות הקשורות לחיפוש.

ואמנם, בשנים האחרונות פותחה שיטת חיפוש חדשה לתמונות, והיא "חיפוש תמונה ע"י תמונה", ובאנגלית - "Reverse Image Search" (ובקיצור "RIS"). בעזרת שיטה זו, ניתן למצוא תמונות הדומות במראה לתמונה המקורית. למעשה זהו כלי עוצמתי למדי, שביכולתו לאתר אתרי אינטרנט נוספים בהם נמצאת התמונה ולדלות מהם מידע.

למשל, נבצע RIS לתמונה הבאה:

התוצאות המתקבלות מהחיפוש נראות כך:



כפי שניתן לראות, כלי ה-RIS במנוע החיפוש של גוגל מצא מגוון אתרים בהם מופיעה אותה תמונה בדיוק, על סמך "חיפוש התמונה עצמה" בלבד.

ה-RIS של גוגל, בו אני משתמש בפרויקט, מורכב ממודל מתמטי מורכב אשר ממיר את התמונה לפורמט מספרי, אותו גוגל משווה עם מיליארדי התמונות שנמצאות באינטרנט ומחזירה את התוצאות הדומות ביותר.

באשר לפרויקט שלי, ברצוני לספק מידע נוסף למשתמשים במידה ומופיע רכב בתמונה, כגון שם החברה, הדגם והמחיר. לכאורה, ניתן ליצור מערך נתונים רחב היקף המכיל כל דגם ופרטים מדוקדקים עליו. מערך נתונים שכזה צריך להיות מתוחזק ומעודכן באופן תמידי, שכן מאות דגמים חדשים נכנסים לשוק מדי שנה, המחירים משתנים וכו'. בנוסף, איסוף הנתונים למערך זה הוא תהליך ארוך מאוד.

לכן, בחרתי כאמור להשתמש ב-RIS של גוגל, דרך ה-Google Cloud Vision API. זהו API ידידותי מטעם גוגל המאפשר לחפש תמונה דרך קישור / מיקום במחשב ולאחזר עשרות תוצאות במהירות מדהימה. האלגוריתם לפיו פעלתי מתבצע בצורה הבאה:

המשתמש מצלם תמונה —> מודל ה-NLP מחזיר את המשפט —> מתבצעת בדיקה האם מופיע אזכור של רכב —> אם כן, מתבצע RIS על התמונה —> אם ישנן תוצאות טובות*, מתבצע Web Scraping על מספר אתרי רכב רלוונטיים לקבלת פרטים על הרכב בהתבסס על זיהוי ה-RIS —> המידע מתווסף לפלט המוצג באפליקציה.

*תוצאות טובות = כאשר בלפחות תוצאת RIS אחת מוזכר שם של חברת רכב כלשהי.

אלה הספריות בהן השתמשתי לצורך שלב זה.

```
import requests  # לביצוע ה-Web Scraping
from bs4 import BeautifulSoup  # לביצוע ה-Web Scraping
import io  # לעבודה עם תמונות
from google.cloud import vision  # ה-API עצמו
from google.oauth2 import service_account  # לאפשר את השימוש ב-API
```

השלבים הראשונים מוצגים בפרקים האחרים, וכאן נתמקד ב-RIS. נתחיל בשימוש ב-API.
ראשית יש לייבא את ה-API וליצור חשבון ב-Google Cloud Project על מנת לקבל רישיון לשימוש ב-API.
API: https://cloud.google.com/vision/docs/internet-detection

ה-API עצמו מורכב משתי פונקציות. הראשונה, והמרכזית, מבצעת את החיפוש, והפונקציה השנייה מקבלת את הפלט של הראשונה ומחזירה את כל הפלטים בצורה מסודרת. הקריאה הראשונית עם התמונה כפרמטר נמצאת בפונקציית תרגום המשפט לעברית, ראו פרק "תרגום המשפט לעברית".

הפונקציה הראשונה.

```python
def annotate(path):
    """Returns web annotations given the path to an image."""
    creds = service_account.Credentials.from_service_account_file('google_cloud.json')
    client = vision.ImageAnnotatorClient(
        credentials=creds,
    )

    if path.startswith('http') or path.startswith('gs:'):
        image = vision.Image()
        image.source.image_uri = path

    else:
        with io.open(path, 'rb') as image_file:
            content = image_file.read()

        image = vision.Image(content=content)

    web_detection = client.web_detection(image=image).web_detection

    return web_detection
```

והשנייה.

```python
def report(annotations):
    if annotations.web_entities:
        return annotations.web_entities
```

הקריאה והפלט בעבור תמונה מסויימת נראה כך:

```python
report(annotate('img.jpg'))
```

```
10 Pages with matching images retrieved
Url   : http://www.carquotes.com/new-prices/audi/a4-sedan/
Url   : https://www.carfolio.com/audi-a4-45-tfsi-quattro-s-tronic-602030
Url   : https://www.guideautoweb.com/en/makes/audi/a4/2020/specifications/45-tfsi-komfort/
Url   : https://carcostcanada.com/Canada/StandardFeatures/411733
Url   : https://www.mychevysparkev.com/audi-a4-45-tfsi-quattro-k.html
Url   : https://www.mychevysparkev.com/audi-a4-quattro-tfsi-k.html
Url   : https://www.carfolio.com/audi-a4-40-tfsi-709236
Url   : https://www.mychevysparkev.com/a4-45-tfsi-quattro-s-line-k.html
Url   : https://www.carfolio.com/audi-a4-40-tfsi-s-line-709203
Url   : https://www.mychevysparkev.com/audi-a4-2.0-tfsi-quattro-2019-k.html
```

```
10 Full Matches found:
Url  : https://www.carpixel.net/w/4171d5a8c343c078be7f4624fb04bc3f/audi-a4-sedan-s-line-wallpaper-hd-92411.jpg
Url  : https://static.carindigo.com/images/news/featured_2022-audi-a4-renderings-hint-at-massive-design-changes-for-the-compact-sedan_1595401826.jpg
Url  : https://wheelz.me/wp-content/uploads/2019/05/Audi-a4-1.jpg
Url  : https://www.autorevue.cz/getthumbnail.aspx?w=20000&h=20000&q=100&id_file=762176932
Url  : https://www.autorevue.cz/getfile.aspx?id_file=762176932
Url  : https://static.toiimg.com/thumb/msid-79837866,width-1200,height-900,resizemode-4/.jpg
Url  : https://www.autonomous.gr/wp-content/uploads/bfi_thumb/audi-a4-p280jj01er6h32tob0lgos65cjoz1q2y3i15iw06xe.jpg
Url  : https://www.otomobilir.com/wp-content/uploads/2022/04/16506375988.jpg
Url  : https://static.toiimg.com/photo/msid-79837866/79837866.jpg
Url  : http://auto123channel.com/wp-content/uploads/2021/06/Untitled-2-3.jpg
```

```
10 Partial Matches found:
Url  : https://www.carsmagazine.com.ar/wp-content/uploads/2020/06/Audi-A4-02.jpg
Url  : https://image-prod.iol.co.za/16x9/800/Static-photo-Colour-Terra-gray?source=https://xlibris.public.prod.oc.inl.infomaker.io:8443/opencontent/objects/9ee440e
Url  : https://rcs.cdn.publieditor.it/w640/M1360_02.jpg
Url  : https://cdn.shopify.com/s/files/1/0287/2878/6988/articles/2020-audi-a4-sedan-wheel-bolt-tightening-torque-specs-125830_large.jpg?v=1618653868
Url  : https://autos.yahoo.com.tw/p/r/w880/car-trim/July2020/00ab88a0a8521900131896719883c21d.jpeg
Url  : https://phantom-elmundo.unidadeditorial.es/11885170b91d4e33471fc82f215add1b/f/jpg/assets/multimedia/imagenes/2020/01/31/15804588341977.jpg
Url  : https://autoenaccion.com.ar/E7523FqucY6naW4rG9JTqLBC8/uploads/2020/06/Audi_A4-Argentina-11.jpg
Url  : https://img3.stcrm.it/images/22099495/HOR_STD/400x/audi-a4-2020-20.jpg
Url  : https://static.ekskluziva.ba/upload/attachments/audi_a4_2020_1_1068x768_5UG.jpg
Url  : https://phantom-elmundo.unidadeditorial.es/cebe0ddba73598c85d64be624c0dd606/f/webp/assets/multimedia/imagenes/2020/01/31/15804588341977.jpg
```

```
10 Web entities found:
Score       : 1.3265836238861084
Description: 2021 Audi A4
Score       : 1.322046160697937
Description: 2019 Audi A4
Score       : 1.2790305614471436
Description: 2022 Audi A4 allroad
Score       : 1.1564850807189941
Description: アウディ A4 45 TFSI Quattro S Line
Score       : 1.0858500003814697
Description: Audi Quattro
Score       : 1.063349962234497
Description: Audi
Score       : 1.0571999549865723
Description: Car
Score       : 0.8478469848632812
Description: 2020 Audi A4 45 Premium
Score       : 0.708899974822998
Description:
```

כפי שניתן לראות, ה-RIS מחזיר מגוון עצום של תוצאות - קישורים רלוונטים לאתרים בהם מופיעה התמונה ואפילו מעין prediction למה יש בתמונה, כמוצג משמאל.

לצרכי הפרויקט, אני לוקח את אותם פלטים משמאל ובודק האם באחד מהם יש את שם החברה / מודל הרכב, ובעזרתם מבצע את ה-Scraping למידע ספציפי יותר.

אם יש זיהוי של חברה בלבד, אין זה מספיק כדי לחלץ פרטים נוספים, ולכן פלט זה בלבד יעבור בחזרה לפונקציית התרגום ולא יבוצעו חיפושים נוספים. במידה ואכן זוהה הדגם, נחפש מידע נוסף בעזרת ה-Scraping.

לצורך ה-Scraping, יש לחפש במגוון אתרי אינטרנט רלוונטים לרכבים, דוגמת cars.com, kbb.com ועוד. מכיוון שלא רציתי ליצור מודל חיפוש עבור כל אתר בנפרד, יצרתי מחלקה גנרית בשם PriceWebsiteSearch שמקבלת את הקישור לאתר, שם ה-class ב-HTML המפנה להיכן שנמצא המחיר, האם מדובר באתר kbb והאם מדובר באתר carsguide (שכן הם דורשים שינויים מינוריים קצרים). כך נראית המחלקה:

```
class PriceWebsiteSource:

    def __init__(self, url, price_html_class, kbb=False, carsguide=False):
        # default is cars.com
        self.url = url
        self.price_html_class = price_html_class
        self.kbb = kbb  # requires different url formatting
        self.carsguide = carsguide  # requires additional "/price" for url
        self.all_car_models = ['Alfa Romeo', 'Audi', 'BMW', 'Chevrolet', 'Citroen', 'Chevrolet', 'Dacia', 'Daewoo',
                               'Dodge', 'Ferrari', 'Fiat', 'Ford', 'Honda', 'Hyundai', 'Jaguar', 'Jeep', 'Kia', 'Lada',
                               'Lancia', 'Land Rover', 'Lexus', 'Maserati', 'Mazda', 'Mercedes', 'Mitsubishi',
                               'Nissan', 'Opel', 'Peugeot', 'Porsche', 'Renault', 'Rover', 'Saab', 'Seat',
                               'Skoda', 'Subaru', 'Suzuki', 'Tata', 'Tesla', 'Toyota', 'Volkswagen', 'Volvo']
        self.all_car_models = [car.lower() for car in self.all_car_models]
```

```python
def get_price(self, car_model_lst):
    # (method is called only if car_model_lst is longer than 1)

    # For Example:
    # ['2021', 'audi', 'a4'], ['2020', 'audi', 'a4', 'allroad' ...]
    # ['toyota', 'corolla']

    for i in range(len(car_model_lst)):
        if car_model_lst[i].lower() in self.all_car_models:
            if i != len(car_model_lst) - 1:  # confirm there's a model, not just a company

                # Tesla (cars.com) ends with: /tesla-model_3
                # (and we usually take only 2 items)
                # Goal: ["tesla", "model", "3"] -> ["tesla", "model_3"]
                if car_model_lst[i].lower() == "tesla":
                    car_model_lst[i + 1] = car_model_lst[i + 1] + "_" + car_model_lst[i + 2]

                if self.kbb:
                    url = f"{self.url}{car_model_lst[i]}/{car_model_lst[i + 1]}"  # "kbb.com/toyota/prius"

                elif self.carsguide:
                    url = f"{self.url}{car_model_lst[i]}/{car_model_lst[i + 1]}/price"
                    # www.carsguide.com/toyota/prius/price
                else:  # default, cars.com
                    url = f"{self.url}{car_model_lst[i]}-{car_model_lst[i + 1]}"
                    # https://www.cars.com/research/toyota-prius

                html_result = requests.get(url)
                doc = BeautifulSoup(html_result.text, 'html.parser')
                try:
                    tag = doc.find_all(class_=self.price_html_class)[0].decode_contents()
                    tag_data = tag.split()

                    price = "-1"
                    for word in tag_data:
                        if '$' in word:
                            price = word
                            break

                    digits = "$,--1234567890"
                    price = ''.join([char for char in price if char in digits])
                    return price, url

                except IndexError:
                    return "-1", ""
            else:
                return "-1", ""
    return "-1", ""
```

כפי שניתן לראות, הפעולה הבונה מקבלת את כל המשתנים הנדרשים וכן כוללת רשימה של כל חברות הרכב המוכרות. המתודה get_price, כשמה, תפקידה לאחזר את המחיר. בעזרת Web Scraping שמבוצע ע"י Beautiful Soup ו-Requests. בנוסף, במידה ואכן נמצא מחיר, המתודה מחזירה את הקישור ממנו נלקח המידע. (ברירת המחדל של החיפוש היא באתר cars.com, שכן זה אתר הרכבים הגדול והמקיף ביותר בעולם).

דוגמא לשימוש במחלקה:

```
car_to_test = "tesla model 3".split()
carscom_site = PriceWebsiteSource("https://www.cars.com/research/", "accordion-spec-item-value")
print("cars.com:", carscom_site.get_price(car_to_test))
```

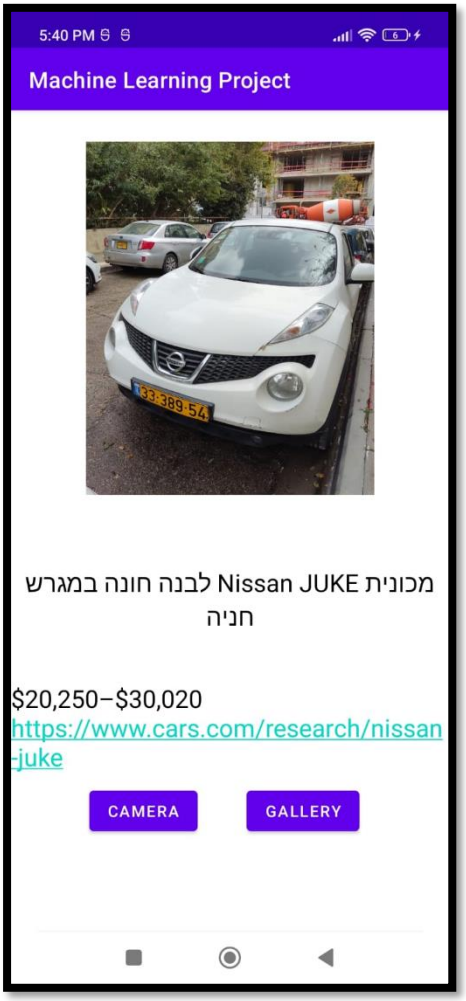
```
cars.com: ('$46,990-$62,990', 'https://www.cars.com/research/tesla-model_3')
```

דוגמא נוספת, במידה ונרצה לבדוק באתר אחר:

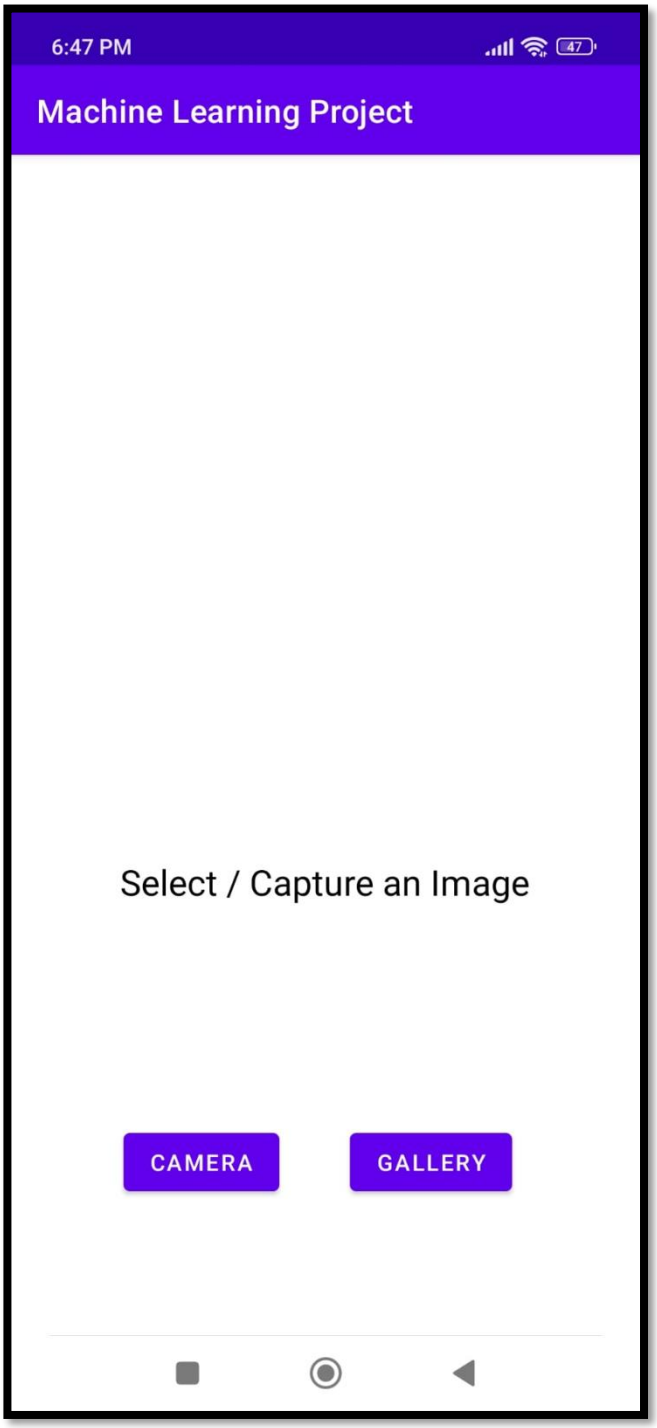
```
car_to_test = "tesla model 3".split()
kbb_site = PriceWebsiteSource("https://www.kbb.com/", "css-167zoth", kbb=True)
print("kbb:", kbb_site.get_price(car_to_test))
```

```
kbb: ('$48,490', 'https://www.kbb.com/tesla/model_3')
```

בפועל, ברירת המחדל היא כאמור cars.com, ואם לא נמצא מידע אז ישנה מעין "רשת ביטחון" של מגוון אתרים נוספים שדרכם נחפש את המחיר. המחלקה הגנרית בהחלט מאוד מקלה על כך. המחיר והקישור מועברים לפונקציית התרגום וכעת הפלט באפליקציה נראה כך:



מדריך למשתמש



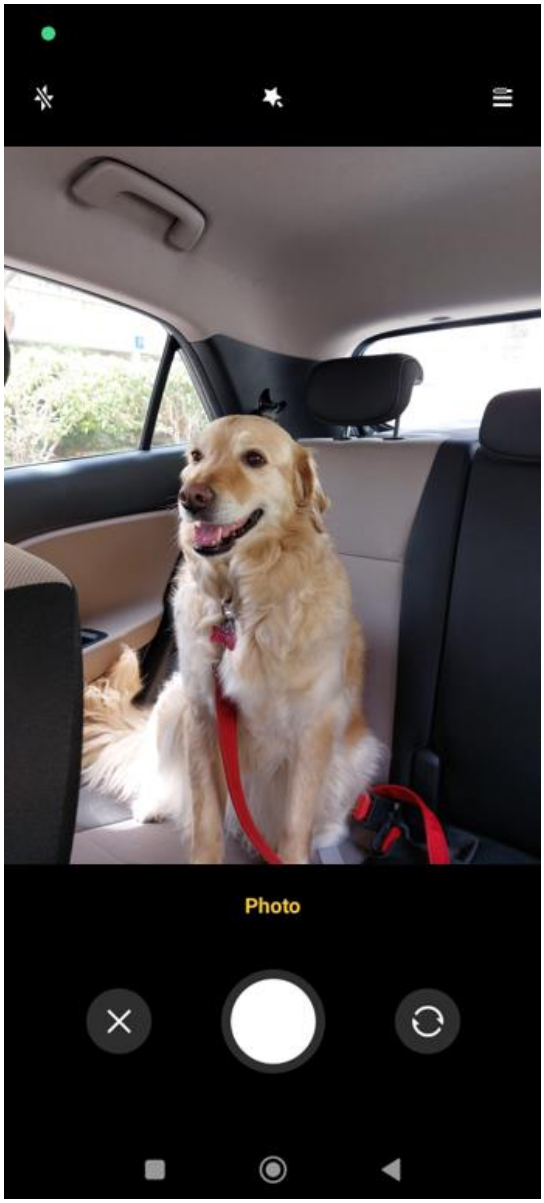
לצורך השימוש באפליקציה, כל שנדרש הוא להוריד את קובץ ה-APK.

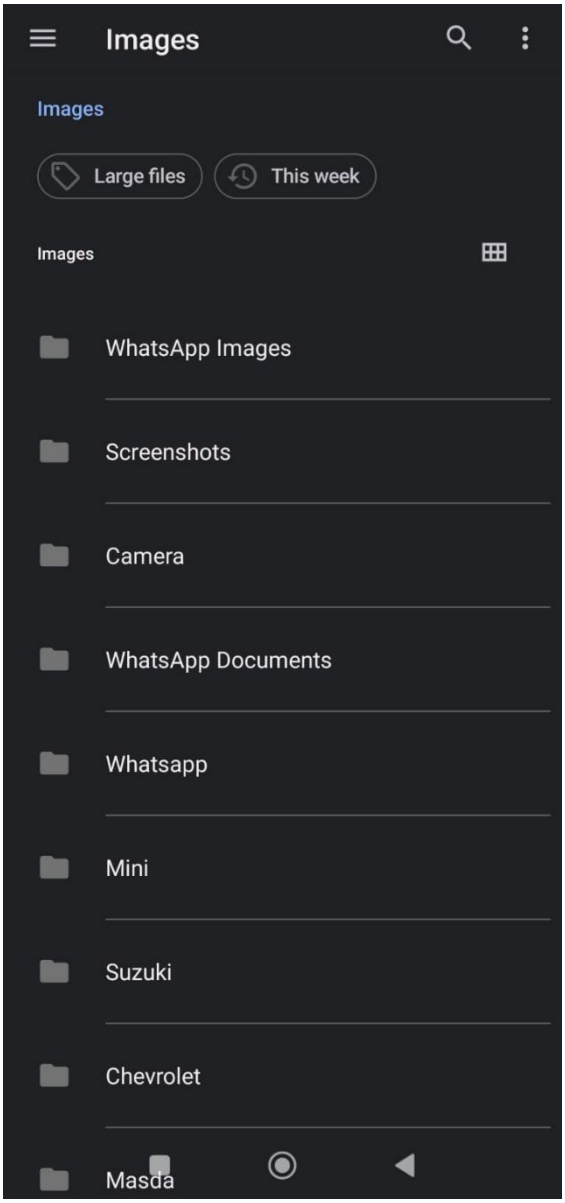
בעת כניסה לאפליקציה, המסך הראשי נראה כך:

המסך הראשי ממוקד וברור ולא ניתן להגיע ממנו למסכים אחרים מלבד בחירת / צילום התמונה.

הכיתוב "Select / Capture an Image" ושני הכפתורים "Camera" ו-"Gallery", לא מותירים ספק בדבר מה שהמשתמש צריך לעשות כדי להתנסות באפליקציה.

כאמור, למשתמש יש שתי אפשרויות לבחור את התמונה שברצונו להעלות. האפשרות הראשונה היא צילום תמונה באופן ישיר ע"י פתיחת המצלמה. לחיצה על הכפתור "Camera" תוביל למסך הצילום משמאל, ובו ניתן לראות שהמצלמה אכן נפתחה ואפשר לצלם. האפשרות השנייה היא העלאת תמונה מהגלריה, בלחיצה על הכפתור "Gallery". הלחיצה תוביל למסך הגלריה מימין, ובו ניתן לראות את הגלריה פתוחה, ובה יכול המשתמש לאתר את התמונה המבוקשת ולבחור אותה.





לאחר בחירת התמונה, יש שלושה שלבים עד לקבלת המשפט בעברית.

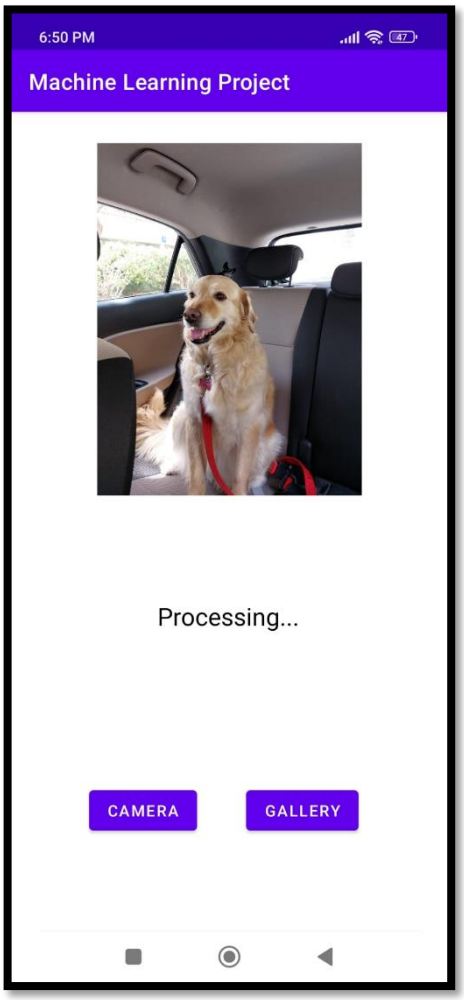
שלב 1 : התמונה נשלחה בהצלחה. בעת שלב זה, יופיע הכיתוב "Processing…".

שלב 2 : התמונה הגיעה למודל בהצלחה. כעת, יתחלף הכיתוב ל-"AI is looking at the picture…".

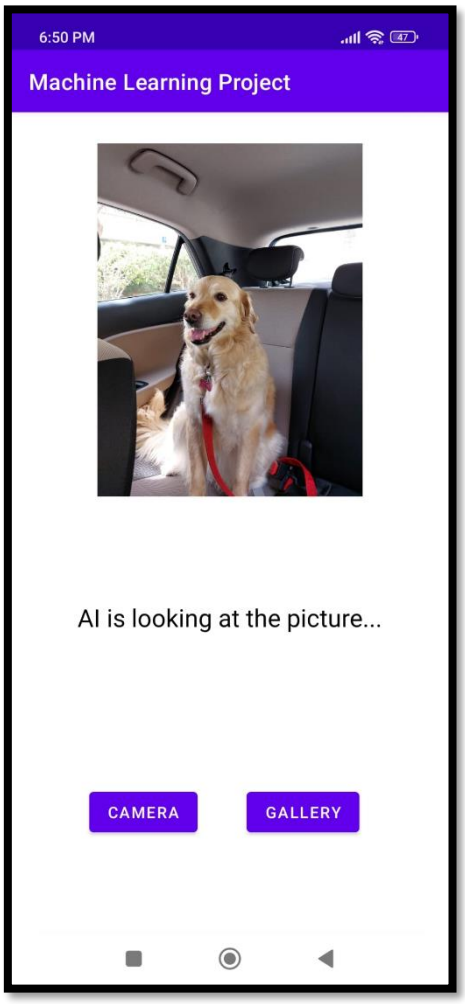
שלב 3 : המודל הוציא משפט בהצלחה והוא חזר לאפליקציה. כעת, הכיתוב יתחלף שוב למשפט עצמו.

להלן הדגמה רצופה של שלושת השלבים, משמאל לימין :

שלב 1



שלב 2



שלב 3

רפלקציה

תהליך הפרויקט עבורי היה מחד מאתגר ולא פשוט ומאידך משמעותי ומספק. באופן כללי, עבדתי על הפרויקט במשך כשלושה חודשים בעבודה אינטנסיבית, שבסופו של דבר הובילה לתוצר שאני גאה בו.

ראשית, מרבית החומר אינו נמצא בתוכנית הלימודים וזה כבר היה אתגר. ללמוד את תחום ה-NLP ועל אופן פעולת ה-Transformers בכללי וכן Image Captioning באופן ספציפי היה מעניין מאוד אך לא קל. כמו כן, שלב פיתוח האפליקציה והטמעת המודל גם היה מאתגר. אף אחד מחבריי למגמה לא ביצע פרויקט הנשען על יסודות למידה עמוקה דומים, וגם אין למנחה שלי ידע נרחב מספיק בתחום, כך שלא היה לי הרבה עם מי להתייעץ ולשאול בדברים נקודתיים הנוגעים למודל בעיקר. עוד דבר הוא שהמתמטיקה המעורבת בתחום היא ברמה גבוהה מאוד, וכאשר נתקלים בבעיות כלשהן או רוצים להעמיק את הידע, מדובר במשימה לא פשוטה בכלל לתלמיד תיכון.

אני יכול להעיד שקיבלתי הרבה כלים מהפרויקט הנוכחי ומהמגמה באופן כללי, כאשר הבולט שבהם הוא היכולת ללמוד לבד ולהתמודד עם חומר לא פשוט, לעתים ברמה אקדמית. למדתי הרבה על תהליך יצירת פרויקט בלמידה עמוקה, החל משלבי המחקר הראשונים, איסוף הנתונים, בנייה ואימון של מודלים, מימוש אפליקציה שתשלב את המודל ולבסוף תיעוד של הכל בספר פרויקט כמו זה. בנוסף קיבלתי הרבה יכולות טכנולוגיות: למדתי את הבסיס של פיתוח אפליקציות אנדרואיד בשפת Kotlin בתוכנה Android Studio, למדתי כיצד להשתמש במסד נתונים מקוון, כיצד לעבוד עם מערכת הענן של גוגל, כיצד להטמיע מודל למידת מכונה באפליקציה וכיצד להשתמש ב-API של גוגל עבור Reverse Image Search.

אם הייתי מתחיל היום את העבודה, הייתי שם דגש נרחב על למידת בסיס ה-NLP בצורה הכי מעמיקה שאפשר ולהרחיב את הידע בתחום ה-Transformers. זאת על מנת למזער אי הבנות במהלך הפרויקט ולהגיע לביצועים מקסימליים. מסקנותיי הן שאין להקל ראש בלמידת הבסיס התיאורטי והמעשי לפני שניגשים לפרויקט מן הסוג הזה, אם באמת רוצים להגיע לתוצרים טובים ומרשימים, ושחשוב לעבוד בצורה הדרגתית, מסודרת ומתוכננת מראש לאורך זמן.

ביבליוגרפיה

Coccomini, D., Messina, N., Gennaro, C., & Falchi, F. (2022). *Combining EfficientNet and Vision Transformers for Video Deepfake Detection* [pdf]. Retrieved from https://arxiv.org/pdf/2107.02612.pdf

Cristina, S. (2021). The Transformer Model. Retrieved from https://machinelearningmastery.com/the-transformer-model/

Cuenat, S., & Couturier, R. (2022). *Convolutional Neural Network (CNN) vs Vision Transformer (ViT) for Digital Holography* [pdf]. Retrieved from https://arxiv.org/pdf/2108.09147.pdf

Gomez, R. (2018). Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. Retrieved from https://gombru.github.io/2018/05/23/cross_entropy_loss/

Guo, J., Han, K., Wu, H., Xu, C., Tang, Y., Xu, C., & Wang, Y. (2021). *CMT: Convolutional Neural Networks Meet Vision Transformers* [pdf]. Retrieved from https://arxiv.org/pdf/2107.06263.pdf

Papineni, K., Roukos, S., Ward, T. and Zhu, W. (2002). *BLEU: a Method for Automatic Evaluation of Machine Translation* [pdf]. Retrieved from https://aclanthology.org/P02-1040.pdf%20

Kumar Nain, A. (2021). Keras documentation: Image Captioning. Retrieved from https://keras.io/examples/vision/image_captioning/

Liu, W., Chen, S., Guo, L., Zhu, X., & Liu, J. (2021). *CPTR: FULL TRANSFORMER NETWORK FOR IMAGE CAPTIONING* [pdf]. Retrieved from https://arxiv.org/pdf/2101.10804.pdf

Ng, A. (2017). Convolutional Neural Networks - Week 1. Retrieved from https://www.coursera.org/learn/convolutional-neural-networks/home/week/1

Ng, A. (2017). Convolutional Neural Networks - Week 2. Retrieved from https://www.coursera.org/learn/convolutional-neural-networks/home/week/2

Ng, A. (2017). Sequence Models - Week 3. Retrieved from https://www.coursera.org/learn/nlp-sequence-models/home/week/3

Ng, A. (2017). Sequence Models - Week 4. Retrieved from https://www.coursera.org/learn/nlp-sequence-models/home/week/4

Ng, A. (2017). Improving Deep Neural Networks - Week 1. Retrieved from https://www.coursera.org/learn/deep-neural-network/home/week/1

nlpconnect/vit-gpt2-image-captioning · Hugging Face. (2020). Retrieved from https://huggingface.co/nlpconnect/vit-gpt2-image-captioning

Tan, M., & V. Le, Q. (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* [pdf]. Retrieved from https://arxiv.org/pdf/1905.11946.pdf

TensorFlow. (2021). *Transfer learning and Transformer models (ML Tech Talks)* [Video]. Retrieved from https://www.youtube.com/watch?v=LE3NfEULV6k

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., & N. Gomez, A. et al. (2017). *Attention Is All You Need* [pdf]. Retrieved from https://arxiv.org/pdf/1706.03762.pdf

Xu, K., Lei Ba, J., Kiros, R., Cho, K., Courville, A., & Salakhutdinov, R. et al. (2016). *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention* [pdf]. Retrieved from https://arxiv.org/pdf/1502.03044.pdf

נספחים

קוד המודל הראשוני:

```
import os
import re
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import efficientnet
from tensorflow.keras.layers import TextVectorization


seed = 111
np.random.seed(seed)
tf.random.set_seed(seed)

!wget -
q https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flic
kr8k_Dataset.zip
!wget -
q https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flic
kr8k_text.zip
!unzip -qq Flickr8k_Dataset.zip
!unzip -qq Flickr8k_text.zip
!rm Flickr8k_Dataset.zip Flickr8k_text.zip
!rm Flickr8k_text.zip

IMAGES_PATH = "Flicker8k_Dataset"

IMAGE_SIZE = (299, 299)

VOCAB_SIZE = 10000

SEQ_LENGTH = 25

EMBED_DIM = 512

FF_DIM = 512

BATCH_SIZE = 64
EPOCHS = 30
AUTOTUNE = tf.data.AUTOTUNE

def load_captions_data(filename):
```

```
    """Loads captions (text) data and maps them to corresponding images
.
    Args:
        filename: Path to the text file containing caption data.
    Returns:
        caption_mapping: Dictionary mapping image names and the corresp
onding captions
        text_data: List containing all the available captions
    """

    with open(filename) as caption_file:
        caption_data = caption_file.readlines()
        caption_mapping = {}
        text_data = []
        images_to_skip = set()

        for line in caption_data:
            line = line.rstrip("\n")
            img_name, caption = line.split("\t")

            img_name = img_name.split("#")[0]
            img_name = os.path.join(IMAGES_PATH, img_name.strip())

            tokens = caption.strip().split()

            if len(tokens) < 5 or len(tokens) > SEQ_LENGTH:
                images_to_skip.add(img_name)
                continue

            if img_name.endswith("jpg") and img_name not in images_to_s
kip:
                caption = "<start> " + caption.strip() + " <end>"
                text_data.append(caption)

                if img_name in caption_mapping:
                    caption_mapping[img_name].append(caption)
                else:
                    caption_mapping[img_name] = [caption]

        for img_name in images_to_skip:
            if img_name in caption_mapping:
                del caption_mapping[img_name]

        return caption_mapping, text_data


def train_val_split(caption_data, train_size=0.8, shuffle=True):
    all_images = list(caption_data.keys())
```

```
    if shuffle:
        np.random.shuffle(all_images)

    train_size = int(len(caption_data) * train_size)

    training_data = {
        img_name: caption_data[img_name] for img_name in all_images[:tr
ain_size]
    }
    validation_data = {
        img_name: caption_data[img_name] for img_name in all_images[tra
in_size:]
    }

    return training_data, validation_data


captions_mapping, text_data = load_captions_data("Flickr8k.token.txt")
train_data, valid_data = train_val_split(captions_mapping)

def custom_standardization(input_string):
    lowercase = tf.strings.lower(input_string)
    return tf.strings.regex_replace(lowercase, "[%s]" % re.escape(strip
_chars), "")


strip_chars = "!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"
strip_chars = strip_chars.replace("<", "")
strip_chars = strip_chars.replace(">", "")

vectorization = TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode="int",
    output_sequence_length=SEQ_LENGTH,
    standardize=custom_standardization,
)
vectorization.adapt(text_data)

image_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.2),
        layers.RandomContrast(0.3),
    ]
)

def decode_and_resize(img_path):
    img = tf.io.read_file(img_path)
    img = tf.image.decode_jpeg(img, channels=3)
```

```
    img = tf.image.resize(img, IMAGE_SIZE)
    img = tf.image.convert_image_dtype(img, tf.float32)
    return img


def process_input(img_path, captions):
    return decode_and_resize(img_path), vectorization(captions)


def make_dataset(images, captions):
    dataset = tf.data.Dataset.from_tensor_slices((images, captions))
    dataset = dataset.shuffle(len(images))
    dataset = dataset.map(process_input, num_parallel_calls=AUTOTUNE)
    dataset = dataset.batch(BATCH_SIZE).prefetch(AUTOTUNE)

    return dataset


train_dataset = make_dataset(list(train_data.keys()), list(train_data.v
alues()))

valid_dataset = make_dataset(list(valid_data.keys()), list(valid_data.v
alues()))


def get_cnn_model():
    base_model = efficientnet.EfficientNetB3(
        input_shape=(*IMAGE_SIZE, 3), include_top=False, weights="image
net",
    )
    base_model.trainable = False
    base_model_out = base_model.output
    base_model_out = layers.Reshape((-1, base_model_out.shape[-
1]))(base_model_out)
    cnn_model = keras.models.Model(base_model.input, base_model_out)
    return cnn_model


class TransformerEncoderBlock(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads
        self.attention_1 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim, dropout=0.0
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()
```

```
        self.dense_1 = layers.Dense(embed_dim, activation="relu")

    def call(self, inputs, training, mask=None):
        inputs = self.layernorm_1(inputs)
        inputs = self.dense_1(inputs)

        attention_output_1 = self.attention_1(
            query=inputs,
            value=inputs,
            key=inputs,
            attention_mask=None,
            training=training,
        )
        out_1 = self.layernorm_2(inputs + attention_output_1)
        return out_1


class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, vocab_size, embed_dim, **kwargs
):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=vocab_size, output_dim=embed_dim
        )
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=embed_dim
        )
        self.sequence_length = sequence_length
        self.vocab_size = vocab_size
        self.embed_dim = embed_dim
        self.embed_scale = tf.math.sqrt(tf.cast(embed_dim, tf.float32))

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_tokens = embedded_tokens * self.embed_scale
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return tf.math.not_equal(inputs, 0)


class TransformerDecoderBlock(layers.Layer):
    def __init__(self, embed_dim, ff_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.ff_dim = ff_dim
```

```
        self.num_heads = num_heads
        self.attention_1 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim, dropout=0.1
        )
        self.attention_2 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim, dropout=0.1
        )
        self.ffn_layer_1 = layers.Dense(ff_dim, activation="relu")
        self.ffn_layer_2 = layers.Dense(embed_dim)

        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()
        self.layernorm_3 = layers.LayerNormalization()

        self.embedding = PositionalEmbedding(
            embed_dim=EMBED_DIM, sequence_length=SEQ_LENGTH, vocab_size
=VOCAB_SIZE
        )
        self.out = layers.Dense(VOCAB_SIZE, activation="softmax")

        self.dropout_1 = layers.Dropout(0.3)
        self.dropout_2 = layers.Dropout(0.5)
        self.supports_masking = True

    def call(self, inputs, encoder_outputs, training, mask=None):
        inputs = self.embedding(inputs)
        causal_mask = self.get_causal_attention_mask(inputs)

        if mask is not None:
            padding_mask = tf.cast(mask[:, :, tf.newaxis], dtype=tf.int
32)
            combined_mask = tf.cast(mask[:, tf.newaxis, :], dtype=tf.in
t32)
            combined_mask = tf.minimum(combined_mask, causal_mask)

        attention_output_1 = self.attention_1(
            query=inputs,
            value=inputs,
            key=inputs,
            attention_mask=combined_mask,
            training=training,
        )
        out_1 = self.layernorm_1(inputs + attention_output_1)

        attention_output_2 = self.attention_2(
            query=out_1,
            value=encoder_outputs,
            key=encoder_outputs,
```

```
            attention_mask=padding_mask,
            training=training,
        )
        out_2 = self.layernorm_2(out_1 + attention_output_2)

        ffn_out = self.ffn_layer_1(out_2)
        ffn_out = self.dropout_1(ffn_out, training=training)
        ffn_out = self.ffn_layer_2(ffn_out)

        ffn_out = self.layernorm_3(ffn_out + out_2, training=training)
        ffn_out = self.dropout_2(ffn_out, training=training)
        preds = self.out(ffn_out)
        return preds

    def get_causal_attention_mask(self, inputs):
        input_shape = tf.shape(inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = tf.range(sequence_length)[:, tf.newaxis]
        j = tf.range(sequence_length)
        mask = tf.cast(i >= j, dtype="int32")
        mask = tf.reshape(mask, (1, input_shape[1], input_shape[1]))
        mult = tf.concat(
            [tf.expand_dims(batch_size, -
1), tf.constant([1, 1], dtype=tf.int32)],
            axis=0,
        )
        return tf.tile(mask, mult)


class ImageCaptioningModel(keras.Model):
    def __init__(
        self, cnn_model, encoder, decoder, num_captions_per_image=5, im
age_aug=None,
    ):
        super().__init__()
        self.cnn_model = cnn_model
        self.encoder = encoder
        self.decoder = decoder
        self.loss_tracker = keras.metrics.Mean(name="loss")
        self.acc_tracker = keras.metrics.Mean(name="accuracy")
        self.num_captions_per_image = num_captions_per_image
        self.image_aug = image_aug

    def calculate_loss(self, y_true, y_pred, mask):
        loss = self.loss(y_true, y_pred)
        mask = tf.cast(mask, dtype=loss.dtype)
        loss *= mask
        return tf.reduce_sum(loss) / tf.reduce_sum(mask)
```

```
    def calculate_accuracy(self, y_true, y_pred, mask):
        accuracy = tf.equal(y_true, tf.argmax(y_pred, axis=2))
        accuracy = tf.math.logical_and(mask, accuracy)
        accuracy = tf.cast(accuracy, dtype=tf.float32)
        mask = tf.cast(mask, dtype=tf.float32)
        return tf.reduce_sum(accuracy) / tf.reduce_sum(mask)

    def _compute_caption_loss_and_acc(self, img_embed, batch_seq, train
ing=True):
        encoder_out = self.encoder(img_embed, training=training)
        batch_seq_inp = batch_seq[:, :-1]
        batch_seq_true = batch_seq[:, 1:]
        mask = tf.math.not_equal(batch_seq_true, 0)
        batch_seq_pred = self.decoder(
            batch_seq_inp, encoder_out, training=training, mask=mask
        )
        loss = self.calculate_loss(batch_seq_true, batch_seq_pred, mask
)
        acc = self.calculate_accuracy(batch_seq_true, batch_seq_pred, m
ask)
        return loss, acc

    def train_step(self, batch_data):
        batch_img, batch_seq = batch_data
        batch_loss = 0
        batch_acc = 0

        if self.image_aug:
            batch_img = self.image_aug(batch_img)

        img_embed = self.cnn_model(batch_img)
        for i in range(self.num_captions_per_image):
            with tf.GradientTape() as tape:
                loss, acc = self._compute_caption_loss_and_acc(
                    img_embed, batch_seq[:, i, :], training=True
                )

                batch_loss += loss
                batch_acc += acc

            train_vars = (
                self.encoder.trainable_variables + self.decoder.trainab
le_variables
            )

            grads = tape.gradient(loss, train_vars)

            self.optimizer.apply_gradients(zip(grads, train_vars))
```

```
        batch_acc /= float(self.num_captions_per_image)
        self.loss_tracker.update_state(batch_loss)
        self.acc_tracker.update_state(batch_acc)

        return {"loss": self.loss_tracker.result(), "acc": self.acc_tra
cker.result()}

    def test_step(self, batch_data):
        batch_img, batch_seq = batch_data
        batch_loss = 0
        batch_acc = 0

        img_embed = self.cnn_model(batch_img)

        for i in range(self.num_captions_per_image):
            loss, acc = self._compute_caption_loss_and_acc(
                img_embed, batch_seq[:, i, :], training=False
            )

            batch_loss += loss
            batch_acc += acc

        batch_acc /= float(self.num_captions_per_image)

        self.loss_tracker.update_state(batch_loss)
        self.acc_tracker.update_state(batch_acc)

        return {"loss": self.loss_tracker.result(), "acc": self.acc_tra
cker.result()}

    @property
    def metrics(self):
        return [self.loss_tracker, self.acc_tracker]


cnn_model = get_cnn_model()
encoder = TransformerEncoderBlock(embed_dim=EMBED_DIM, dense_dim=FF_DIM
, num_heads=20)
decoder = TransformerDecoderBlock(embed_dim=EMBED_DIM, ff_dim=FF_DIM, n
um_heads=21)
caption_model = ImageCaptioningModel(
    cnn_model=cnn_model, encoder=encoder, decoder=decoder, image_aug=im
age_augmentation,
)

cross_entropy = keras.losses.SparseCategoricalCrossentropy(
    from_logits=False, reduction="none"
)
```

```
early_stopping = keras.callbacks.EarlyStopping(patience=10, restore_bes
t_weights=True)


num_train_steps = len(train_dataset) * EPOCHS
num_warmup_steps = num_train_steps // 15
lr_schedule = LRSchedule(post_warmup_learning_rate=1e-
4, warmup_steps=num_warmup_steps)

caption_model.compile(optimizer=keras.optimizers.Adam, loss=cross_entro
py)

history = caption_model.fit(
    train_dataset,
    epochs=EPOCHS,
    validation_data=valid_dataset,
    callbacks=[early_stopping],
)

plt.plot(history.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()

training_acc = history.history['acc']
val_acc = history.history['val_acc']
training_loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(15, training_acc, color = 'green', marker = '*', label = 'Trai
ning Accuracy')
plt.plot(15, val_acc, color = 'blue', marker = 'o', label = 'Validation
 Accuracy')
plt.title('Training Accuracy and Validation Accuracy Vs Epochs')
plt.legend()
plt.figure()

plt.plot(15, training_loss, color = 'green', marker = '*', label = 'Tra
ining Loss')
plt.plot(15, val_loss, color = 'blue', marker = 'o', label = 'Validatio
n Loss')
plt.title('Training Loss and Validation Loss Vs Epochs')
plt.legend()
plt.figure()
```

קוד המסך הראשי באפליקציה (שפת Kotlin):

```
package com.eshqol.imagepickefinal

import android.annotation.SuppressLint
import android.content.Intent
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.os.Bundle
import android.provider.MediaStore
import android.util.Base64
import android.widget.Button
import android.widget.ImageView
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.app.AppCompatDelegate
import androidx.core.graphics.drawable.toBitmap
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener
import com.google.firebase.database.ktx.database
import com.google.firebase.database.ktx.getValue
import com.google.firebase.ktx.Firebase
import com.google.mlkit.vision.label.ImageLabeler
import com.google.mlkit.vision.label.ImageLabeling
import com.google.mlkit.vision.label.defaults.ImageLabelerOptions
import java.io.ByteArrayOutputStream


@Suppress("DEPRECATION")
class MainActivity : AppCompatActivity() {
    private val PICK_IMAGE = 1500
    private val CAMERA_REQUEST = 2000
    private lateinit var imageLabeler: ImageLabeler
    private var last = ""
    private var first = true
    private val database = Firebase.database


    @SuppressLint("SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        this.title = "Machine Learning Project"


AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO)

        val image = findViewById<ImageView>(R.id.imageView2)
        imageLabeler =
ImageLabeling.getClient(ImageLabelerOptions.DEFAULT_OPTIONS)

        val gallery = findViewById<Button>(R.id.button)
        val camera = findViewById<Button>(R.id.button2)

        gallery.setOnClickListener {
            pickImage("gallery")
        }

        camera.setOnClickListener {
```

```
            pickImage("camera")
        }
    }

    private fun pickImage(kind: String) {
        if (kind == "gallery"){
            val intent = Intent(Intent.ACTION_GET_CONTENT)
            intent.type = "image/*"
            startActivityForResult(intent, PICK_IMAGE)
        }
        else{
            val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            startActivityForResult(cameraIntent, CAMERA_REQUEST)
        }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        val image = findViewById<ImageView>(R.id.imageView2)

        if (requestCode == PICK_IMAGE && resultCode == RESULT_OK) {
            if (data == null) {
                return
            }

            val inputStream = contentResolver.openInputStream(data.data!!)
            val bitmap = BitmapFactory.decodeStream(inputStream)
            image.setImageBitmap(bitmap)
            makeTheMagic()

        }

        else if (requestCode == CAMERA_REQUEST && resultCode == RESULT_OK)
{
            val bitmap = data!!.extras!!["data"] as Bitmap?

            image.setImageBitmap(bitmap)
            makeTheMagic()
        }
    }

    @SuppressLint("SetTextI18n")
    private fun makeTheMagic() {
        val image = findViewById<ImageView>(R.id.imageView2)
        val textResult = findViewById<TextView>(R.id.textView)
        val priceText = findViewById<TextView>(R.id.priceTextView)

        textResult.text = "Processing..."
        priceText.text = ""
        var bitmap = image.drawable.toBitmap()

        val oldSize = bitmap.byteCount
        if (oldSize > 10000000) {
            bitmap = getResizedBitmap(bitmap, 1000)
        }

        val baos = ByteArrayOutputStream()
        bitmap.compress(Bitmap.CompressFormat.JPEG, 80, baos)
```

```
        val byteArray = baos.toByteArray()
        val encoded = Base64.encodeToString(byteArray, Base64.DEFAULT)

        val myRef = database.getReference("message")
        myRef.setValue(encoded.toString())

        val ref = database.getReference("output")
        val refRefresh = database.getReference("refresh")
        refRefresh.setValue("True")


        ref.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                val data = dataSnapshot.getValue<String>().toString()
                if (first){
                    last = data
                    first = false
                }
                if (last != data){
                    last = data
                    if ("@" in data) {
                        val (sentence, price) = data.split('@')
                        textResult.text = sentence
                        if (price != "-1" && price != "- 1" && "-1" !in
price) {
                            val p = price.replace(' ', '\n')
                            priceText.text = p
                        }
                    }
                    else {
                        textResult.text = data
                        priceText.text = ""
                    }
                }

            }

            override fun onCancelled(error: DatabaseError) {
            }
        })

    }

    private fun getResizedBitmap(image: Bitmap, maxSize: Int): Bitmap {
        var width = image.width
        var height = image.height
        val bitmapRatio = width.toFloat() / height.toFloat()
        if (bitmapRatio > 1) {
            width = maxSize
            height = (width / bitmapRatio).toInt()
        } else {
            height = maxSize
            width = (height * bitmapRatio).toInt()
        }
        return Bitmap.createScaledBitmap(image, width, height, true)
    }
}
```