זיהוי וקיטלוג ציורים בעזרת רשת נוירונים עמוקה

מגיש: בן רופא-אורן

ת.ז: 215525502

מנחה: אסנת אנגלמן

שם חלופה: מערכות מומחה בתחום למידת מכונה

תאריך הגשה: מרץ 2023

תוכן עניינים

מבוא

הרקע לפרויקט

בפרויקט זה בניתי רשת נוירונים עמוקה כדי להבדיל בין 30 קטגוריות שונות של שרבוטים (doodles). מטרת הפרויקט הולכת מעבר לזיהוי בין שרבוטים שונים, בעזרת בניית מודל שמבדיל בין ציורים שונים אנו מקבלים "proof of concept" שמעיד על כך שניתן לזהות כתיבה ואת מה שאנשים מציירים וכותבים בעזרת כלים שזמינים לרוב האנשים (מחשב ביתי ומצלמה). אני מאמין שפרויקט זה בצורתו הנוכחית פונה לקהל יעד צעיר יחסית שכן כרגע היישום של הפרויקט הוא משחק שבו שחקן מצייר קטגוריה מסוימת והמחשב מנסה לזהות את הקטגוריה של הציור.

בחרתי את הנושא הנ"ל לפרויקט שלי ראשית מכיוון שמעניין אותי לראות האם מחשב מסוגל לחקות את האינטואיציה של בני אדם בעזרת מידע מאוד מוגבל, הרי פעמים רבות כאשר רוצים שמחשב יזהה פרטים בתמונה צבעונית עם פרטים רבים שאפשר למצוא (כמו בזיהוי פנים או קטלוג בין סוגי חיות שונים) אבל אצלי מדובר בציור קטן יחסית בשחור על לבן שצויר בזריזות בלי הרבה מחשבה.

שנית, רציתי שהנושא שלי יהיה בעל data טוב שאוכל לעבוד עליו בלי להרגיש שהפוטנציאל האמיתי של הרשת לא ממומש בגלל מחסור בנתונים ולכן בחרתי בזיהוי וקטלוג שרבוטים שכן המאגר של גוגל רחב מאוד ומאפשר כמות כמעט אינסופית של נתונים לעבוד איתם. לכן במהלך הפרויקט לא הרגשתי שטיב הפרויקט מוגבל על ידי גורמים שלא בשליטתי.

שלישית, אני זוכר את הרגע שבו הניסוי של "Quick Draw" התחיל ובתור ילד מאוד התלהבתי ותמיד רציתי לעשות משהו שקשור לניסוי הזה ופרויקט זה בעיני היה ההזדמנות המושלמת לעשות זאת.

תהליך המחקר

בתחילת הפרויקט למרות שהייתי די בטוח שאשתמש במאגר הציורים של גוגל עקב גודלו והייחודיות שלו (אין כמעט בכלל מאגרים של שרבוטים בשחור על לבן) חיפשתי מאגרים אחרים ולמרות שלא מצאתי שום דבר שהיה טוב יותר מאשר המאגר של גוגל כן מצאתי בדרך מגוון מאמרים ופרויקטים אישיים של אחרים אשר השתמשו בנתונים של גוגל ולכן יכולתי להשיג מהם השראה וגם לראות מה כבר נעשה והיכן אני יכול לנסות לחדש.

מכיוון שמאגר זה נחקר רבות ורציתי לחדש, היה עלי לחפש משהו שלא נעשה הרבה. לאחר חיפושים רבים מצאתי שלמרות שניסו להשתמש ב-Transfer Learning עם רשתות שאומנו על ImageNet לא ניסו לבצע fine-tuning על הרשתות הללו ולכן בפרויקט שלי הוספתי זאת לפרויקט שלי.

מקורות המידע בהם השתמשתי לאורך הפרויקט היו תחילה כדי להבין כיצד הנתונים נראים וכיצד הם שמורים השתמשתי בעמוד ה-GitHub של Quick Draw[6] אשר שם מוסבר כיצד להשיג את התמונות, כיצד הן שמורות ועוד. לאחר מכן נעזרתי בפרויקטים אישיים של אנשים אחרים אשר גם ניסו לבנות רשתות שיזהו את הקטגוריות של הציורים. בנוסף על מאמרים שעסקו בנושא וניסו להשוות בין מספר מודלים שונים כדי לראות איזה מודל הכי טוב כדי לראות מה עבד לאחרים ואיפה ניתן לחדש (קישורים למקורות ימצאו בביבליוגרפיה).

אתגרים מרכזיים

במהלך כתיבת הפרויקט נתקלתי בכמה קשיים מרכזיים:

- בשלב השגת הנתונים והכנתם לקראת אימון במודל היה עליי לעבור בין מספר רב של רזולוציות תמונות והדרך בה צריך לטעון את התמונה (בשחור לבן או rgb לדוגמא). הייתי צריך לעשות זאת מכיוון שכל שינוי של פרמטר כזה מוביל ליתרונות וחסרונות (אם אני בוחר ברזולוציה גבוהה יותר לתמונה לדוגמא ייקח לי הרבה יותר זמן לאמן את התמונות ואולי יהיה צורך גם במודל גדול יותר כדי לעבד את כל הנתונים בצורה יעילה). בנוסף על כך במהלך הוספת Transfer Learning למודל היה עליי לשנות את הדרך בה נטענים הנתונים כדי שהרשת המוכנה תוכל לעבוד איתם (היה עליי לטעון את התמונות בתור rgb למרות שהן בשחור לבן מכיוון שהרשתות המוכנות אומנו על ImageNet שהוא מאגר צבעוני). כל השינויים הללו גזלו הרבה זמן ועבודה ובהחלט היוו קושי מרכזי.
- בשלב בניית ואימון המודל ניסיתי לבנות את הרשת הטובה ביותר שאני יכול אך מכיוון שאימון כל רשת לקח זמן רב יחסית לא יכולתי למצוא את כל הפרמטרים הטובים ביותר בזמן סביר. כדי לפתור בעיה זו נעזרתי באינטרנט כדי לראות לאיזה הצלחות אחרים הגיעו וכיצד עשו זאת ומשם להמשיך עם מה שהם השיגו[1] ולנסות לשפר ולהתאים למטרות שלי (לדוגמא אחת הרשתות שמצאתי עבדה על יותר קטגוריות עם רזולוציה נמוכה יותר זה היה צורך בכמה שינויים כדי שתתאים לצרכים שלי). בנוסף על כך היה עליי להחליט איזו Loss Function היא המתאימה למודל שלי, התלבטתי בין categorical_crossentropy ל-sparse_categorical_crossentropy.
- בשלב היישום הקושי המרכזי היה להעביר בצורה נכונה את התמונות שמצוירות על המסך אל הרשת כך שהתמונה שהועברה והתמונות שעליה אומנה הרשת יהיו בעלות גודל זהה, סוג ערכים זהה (חלק מהפונקציות לטעינת תמונה שמורה שמים עבור ערכי הפיקסלים ערכים בין 0 ל-1 בשונה ממה שהמודל אומן עליו שזה תמונות עם ערכים בין 0 ל-255) ועוד. במידה ופרמטרים אלו לא יהיו זהים התוכנה תקרוס או שהרשת לא תצליח ליישם את הידע שלה על התמונה שכן היא תיראה לרשת אחרת לחלוטין. פתרתי בעיה זו בעזרת הבנה של הנתונים אשר עליהם הרשת אומנה ומספר רב יחסית של פקודות אשר מתאימות את הציור של המשתמש לצורה הנכונה.

מבנה / ארכיטקטורה

שלב איסוף, הכנה וניתוח הנתונים - Collect, prepare and analyze data

מבנה הנתונים ממנו נלקחו הנתונים לפרויקט הוא מאגר התמונות של גוגל "Quick, Draw" אשר כולל למעלה מ-50 מיליון ציורים בשחור על לבן המתפרשים על פני 345 קטגוריות. המאגר נוצר בכדי לראות האם קיימים דפוסים בדרך שבה אנשים מציירים ואם כן אז למצוא אותם. הפרויקט התחיל בשנת 2016 ומאז המשיך לאגור ציורים. מגוון של מאמרים ופרויקטים אישיים שניתן למצוא באינטרנט נכתבו על מאגר המידע הנ"ל.

כעת אסביר פרטים על התמונות במאגר:

כל תמונה היא מסוג קובץ ndjson בגודל 256x256 בשחור על לבן (ערכים בין 0 ל-255) ובנוסף לכך מצורפת גם הקטגוריה אליה אמור להשתייך הציור והאם רשת הנוירונים של Google (שאומנה על ה-data של הציורים שהוכנסו עד כה) הצליחה לזהות בצורה נכונה את הקטגוריה של התמונה. קיימים גם פורמטים אחרים בהם התמונות שמורות (במקום מערך של 256 על 256 שומרים ווקטורים עבור כל קו שמכיל קורדינאטות שהקו עבר בהן והזמן שבו הקו צויר).

בנוסף על כך כל התמונות נחתכות כך שהן ימלאו את המסך (לדוגמא אם צויר ציור קטן בפינה השמאלית אז ייקחו רק את הפינה השמאלית כך שכל הציור יישמר וייגע בקצוות המסך).

כדי לחסוך בזמן ריצה ומכיוון שמדובר בציורים פשוטים יחסית אני כיווצתי את התמונות בעזרת הספרייה שנוצרה עבור המאגר הנ"ל (quickdraw) לגודל של 64 על 64 וגם המרתי את סוג הקובץ להיות png שכן נוח לי יותר לעבוד איתו. נוסף על כך בחרתי רק בתמונות שהרשת של Google זיהתה באופן נכון בכדי למנוע מציורים שאינם דומים בכלל לקטגוריה (אני יוצא מנקודת הנחה שאם הרשת של Google שאומנה על כל המאגר למשך זמן רב יותר על מחשבים חזקים יותר לא הצליחה להבין מה הציור היה אז גם הרשת שלי לא תוכל). מתוך כל 50 מיליון התמונות ו-345 הקטגוריות אני בחרתי 5000 תמונות מכל אחת מתוך 30 קטגוריות כדי שאוכל להריץ את הרשת על המחשבים הזמינים לי.

הקטגוריות אותן בחרתי הן:

```
categories = ['airplane', 'apple', 'axe', 'basketball', 'bicycle', 'broccoli', 'bucket', 'butterfly',
              'crab', "diamond", 'fence', 'fish', "guitar", 'hammer', 'headphones', 'helicopter',
              'hot air balloon', 'ice cream', 'light bulb', 'lollipop', 'palm tree', 'parachute',
              'rainbow', 'sailboat', 'shoe', 'smiley face', 'star', "tennis racquet", 'traffic light',
              'wristwatch']  # categories for images in training data
```

לאחר שטענתי את התמונות הללו השתמשתי בפונקציה אשר טוענת את התמונות ויוצרת מהן מאגר מסוג tf.image.dataset אשר ניתן לאמן את הרשת עליו. הפונקציה לוקחת כל תמונה והופכת אותה לגודל 64 על 64 על 3 (כל תמונה תהיה מסוג RGB):

```
load_dataset = True  # deciding whether to create the files used for the actual training or not

if load_dataset:
    # preparing the data
    batch_size = 128

    # this function will return an image dataset where each image is paired with a label corresponding
    # to the class it belongs to (i.e. for classes a and b pictures from class a will have a label 0
    # and pictures from class b will have label 1)
    train_ds = image_dataset_from_directory(
        "dataset_30",
        validation_split=0.2,
        subset="training",
        seed=123,
        color_mode="rgb",
        image_size=image_size,
        batch_size=batch_size,
        label_mode='categorical'
    )

    val_ds = image_dataset_from_directory(
        "dataset_30",
        validation_split=0.2,
        subset="validation",
        seed=123,
        color_mode="rgb",
        image_size=image_size,
        batch_size=batch_size,
        label_mode='categorical'
    )

    train_ds.save('training_dataset')
    val_ds.save('validation_dataset')

train_ds = tf.data.Dataset.load('training_dataset')  # the training dataset
val_ds = tf.data.Dataset.load('validation_dataset')  # the validation dataset
```

נוסף על כך כדי שאוכל להשתמש ב-data יחד עם Transfer Learning טענתי את התמונות בתור תמונות צבעוניות (המודל המוכן בו השתמשתי דרש תמונות rgb מגודל 32 על 32 ומעלה).

הנרמול של הנתונים נעשה גם לפני שטענתי את התמונות (כבר במאגר עצמו התמונות אשר מושגות מהמשחק ממורכזות לקצה השמאלי העליון של התמונה) וגם בשלב בניית המודל שכן השכבה הראשונה שיש ברשת היא שכבת Rescaling אשר מחלקת ב-255 את ערכי הפיקסלים כך שנקבל ערכים בין 0 ל-1.

שלב בנייה ואימון המודל - Build and train deep learning model

המודל הסופי עליו אימנתי את המודל הוא רשת נוירונים עמוקה עם קונבולוציה ללא Transfer Learning. זה המודל שבחרתי מכיוון שרשתות עם Transfer Learning ו-Fine tuning הגיעו לתוצאות פחות טובות מאשר רשת קונבולוציה רגילה.

כעת אתאר את הארכיטקטורות של כל סוגי 3 הרשתות האחרונות אשר נבדקו לקראת הבחירה הסופית:

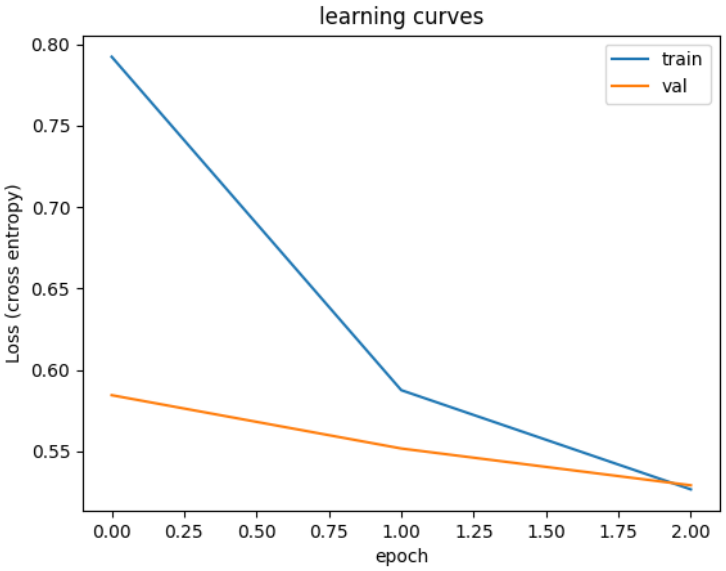
1. רשת עמוקה עם קונבולוציה ו-Transfer Learning ללא Fine Tuning:

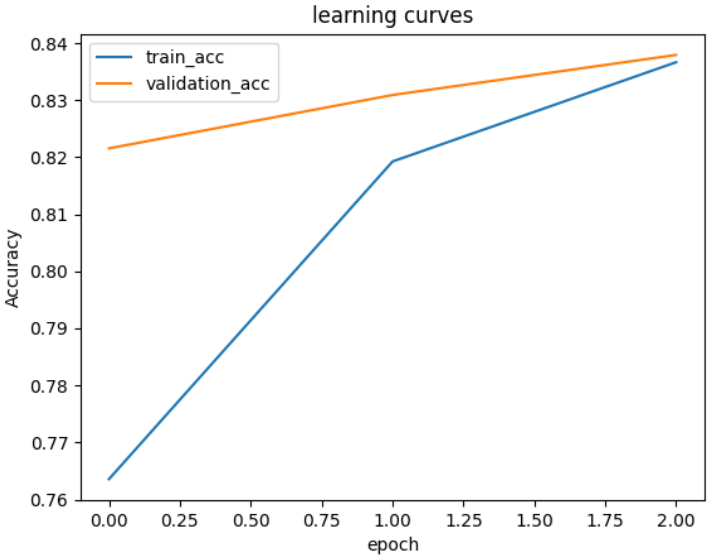
הרשת בעזרתה השתמשתי ב-Transfer Learning היא MobileNet v2 אשר זמינה בעזרת ספריית keras. מצ"ב תמונה של הקוד אשר יצר את המודל וסיכום הפרמטרים של המודל:

```
MobileNetV2_model = Sequential()

pretrained_model = MobileNetV2(
    input_shape=input_shape,
    alpha=1.0,
    include_top=False,
    weights="imagenet",
    pooling=None,
    classes=n_classes,
)

for layer in pretrained_model.layers:
    layer.trainable = False

MobileNetV2_model.add(Rescaling(1. / 255, input_shape=input_shape))
MobileNetV2_model.add(pretrained_model)
MobileNetV2_model.add(tf.keras.layers.GlobalAveragePooling2D())
MobileNetV2_model.add(Dense(1000, activation='relu'))

MobileNetV2_model.add(BatchNormalization())
MobileNetV2_model.add(Dropout(0.2))

MobileNetV2_model.add(Dense(200, activation='relu'))

MobileNetV2_model.add(BatchNormalization())
MobileNetV2_model.add(Dropout(0.2))

MobileNetV2_model.add(Dense(n_classes, activation='softmax'))

model = MobileNetV2_model
```

```
opt = tf.keras.optimizers.Adam(
    learning_rate=0.001
)

top_k_categorical_accuracy = tf.keras.metrics.TopKCategoricalAccuracy(
    k=3, name='top_k_categorical_accuracy', dtype=None
)

real_accuracy = tf.keras.metrics.TopKCategoricalAccuracy(
    k=1, name='real_accuracy', dtype=None
)

model.compile(
    optimizer=opt,
    loss='categorical_crossentropy',
    metrics=["accuracy", real_accuracy, top_k_categorical_accuracy]
)

model.summary()
epochs = 3


tic = Time.time()

history = model.fit(
            train_ds,
            validation_data=val_ds,
            epochs=epochs,
            batch_size=batch_size,
            verbose=1,
            )

toc = Time.time()
```

בנוסף על כך מצורפים גרפים ומדדים אשר מתארים את למידת המודל וסיכום של הפרמטרים שלו:






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

 mobilenetv2_1.00_224 (Funct  (None, 2, 2, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 1000)              1281000

 batch_normalization (BatchN  (None, 1000)             4000
 ormalization)

 dropout (Dropout)           (None, 1000)              0

 dense_1 (Dense)             (None, 200)               200200

 batch_normalization_1 (Batc  (None, 200)              800
 hNormalization)

 dropout_1 (Dropout)         (None, 200)               0

 dense_2 (Dense)             (None, 30)                6030

=================================================================
Total params: 3750014 (14.31 MB)
Trainable params: 1489630 (5.68 MB)
Non-trainable params: 2260384 (8.62 MB)
_________________________________________________________________
```

```
Train accuracy: 0.8366488218307495
val_accuracy: 0.8379285931587219

top_k_categorical_accuracy: 0.9550
val_top_k_categorical_accuracy: 0.9545
```

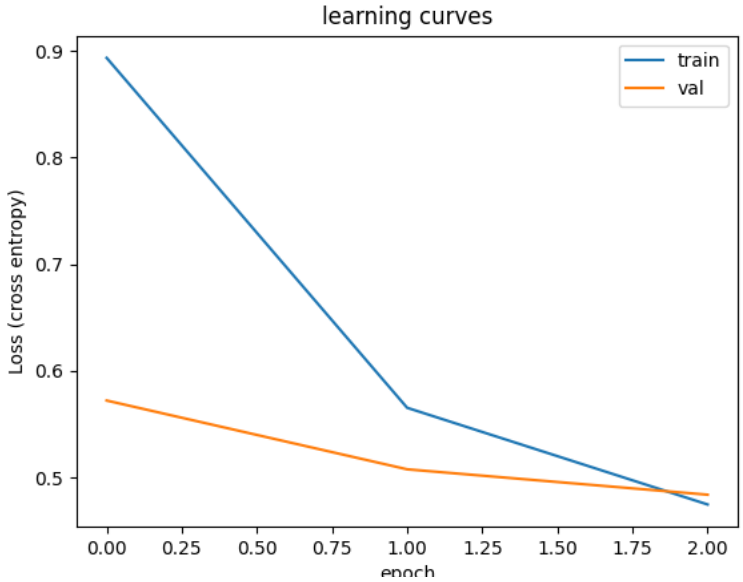
2. רשת עמוקה עם קונבולוציה ו-Transfer Learning ועם Fine Tuning:

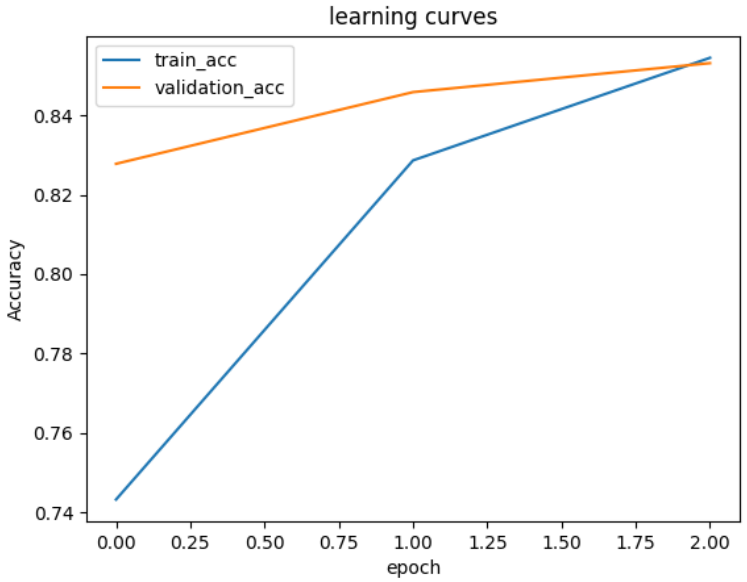
הרשת בעזרתה השתמשתי ב-Transfer Learning היא MobileNet v2 אשר זמינה בעזרת ספריית keras. בנוסף על כך ביצעתי Fine Tuning כך שאימנתי את 22 השכבות האחרונות ב-MobileNet v2. מצ"ב תמונה של הקוד אשר יצר את המודל:

```
input_shape = (image_size[0], image_size[1], 3)  # turning our photo

n_classes = len(categories)

MobileNetV2_model = Sequential()

pretrained_model = MobileNetV2(
    input_shape=input_shape,
    alpha=1.0,
    include_top=False,
    weights="imagenet",
    pooling=None,
    classes=n_classes,
)

for layer in pretrained_model.layers:
    layer.trainable = False

MobileNetV2_model.add(Rescaling(1. / 255, input_shape=input_shape))
MobileNetV2_model.add(pretrained_model)
MobileNetV2_model.add(Flatten())
MobileNetV2_model.add(Dense(1000, activation='relu'))

MobileNetV2_model.add(BatchNormalization())
MobileNetV2_model.add(Dropout(0.2))

MobileNetV2_model.add(Dense(200, activation='relu'))

MobileNetV2_model.add(BatchNormalization())
MobileNetV2_model.add(Dropout(0.2))

MobileNetV2_model.add(Dense(n_classes, activation='softmax'))

model = MobileNetV2_model

# fine-tuning the model
for layer in model.layers[:-22]:
    layer.trainable = False
```

```
opt = tf.keras.optimizers.Adam(
    learning_rate=0.0001
)

top_k_categorical_accuracy = tf.keras.metrics.TopKCategoricalAccuracy(
    k=3, name='top_k_categorical_accuracy', dtype=None
)

model.compile(
    optimizer=opt,
    loss='categorical_crossentropy',
    metrics=["accuracy", top_k_categorical_accuracy]
)

model.summary()
epochs = 3


tic = Time.time()

history = model.fit(
            train_ds,
            validation_data=val_ds,
            epochs=epochs,
            batch_size=batch_size,
            verbose=1,
            )

toc = Time.time()
```

בנוסף על כך מצורפים גרפים ומדדים אשר מתארים את למידת המודל וסיכום של הפרמטרים שלו:






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

 mobilenetv2_1.00_224 (Funct  (None, 2, 2, 1280)       2257984
 ional)

 flatten (Flatten)           (None, 5120)              0

 dense (Dense)               (None, 1000)              5121000

 batch_normalization (BatchN  (None, 1000)             4000
 ormalization)

 dropout (Dropout)           (None, 1000)              0

 dense_1 (Dense)             (None, 200)               200200

 batch_normalization_1 (Batc  (None, 200)              800
 hNormalization)

 dropout_1 (Dropout)         (None, 200)               0

 dense_2 (Dense)             (None, 30)                6030

=================================================================
Total params: 7590014 (28.95 MB)
Trainable params: 5329630 (20.33 MB)
Non-trainable params: 2260384 (8.62 MB)
_________________________________________________________________
```

```
Train accuracy: 0.854440450668335
val_accuracy: 0.8530952334403992

top_k_categorical_accuracy: 0.9640
val_top_k_categorical_accuracy: 0.9580
```

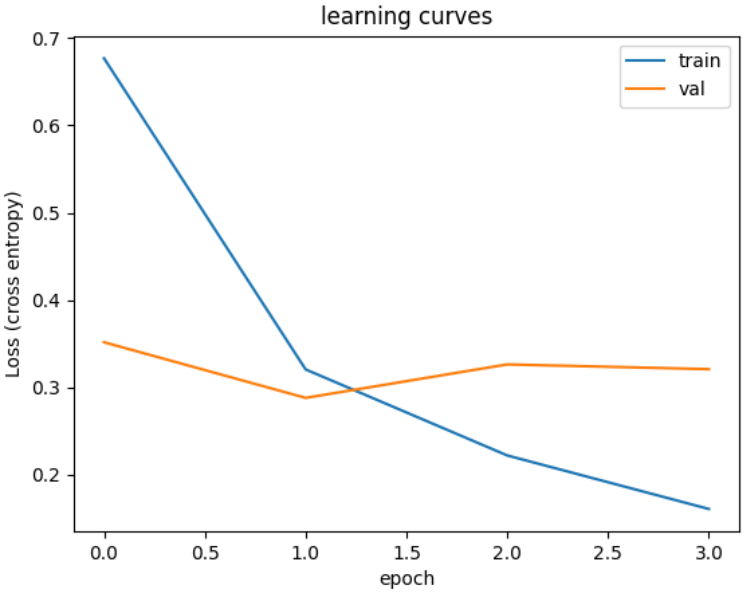
3. רשת עמוקה עם קונבולוציה ללא Transfer Learning וללא Fine Tuning:

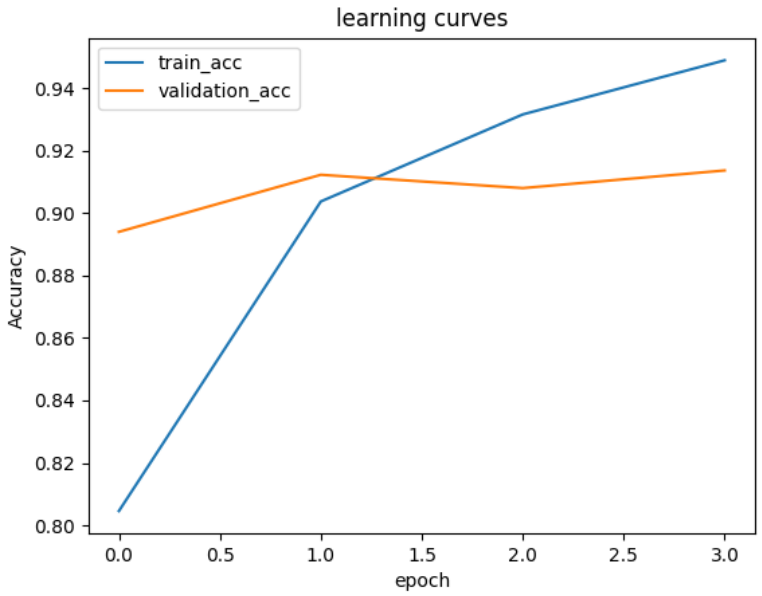
מצ"ב תמונה של הקוד אשר יצר את המודל וסיכום הפרמטרים של המודל:

```
# turning our photos into shape (image_size[0], image_size[1], 3)
input_shape = (image_size[0], image_size[1], 3)


n_classes = len(categories)  # number of categories in training set data

model = Sequential([
    Rescaling(1. / 255, input_shape=input_shape),
    BatchNormalization(),

    Conv2D(16, kernel_size=(3, 3), padding="same", activation="relu"),
    Conv2D(32, kernel_size=(3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), padding="same", activation="relu"),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),

    Dense(400, activation='relu'),
    BatchNormalization(),
    Dropout(0.2),

    Dense(200, activation='relu'),
    BatchNormalization(),
    Dropout(0.2),

    Dense(100, activation='relu'),
    Dropout(0.2),

    Dense(n_classes, activation='softmax')
])
```

```python
# Train model
opt = tf.optimizers.Adam()  # choosing the Adam optimizer

top_k_categorical_accuracy = tf.keras.metrics.TopKCategoricalAccuracy(
    k=3, name='top_k_categorical_accuracy', dtype=None
)
#  using the top k categorical accuracy which means it measures the number
#  of times the correct category of the image was in the 3 categories which
#  received the highest probability by the network and divides it by the
#  overall predictions the network made

model.compile(
    optimizer=opt,
    loss='categorical_crossentropy',  # using the categorical_crossentropy
    # loss function because we are trying to differentiate between a number
    # of categories
    metrics=["accuracy", top_k_categorical_accuracy]
)


model.summary()
epochs = 4  # number of epochs the network will run for


tic = Time.time()  # starting to measure the time the network took to run

history = model.fit(  # training the model and saving the results to history
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    batch_size=batch_size,
    verbose=1,
)

toc = Time.time()  # finishing to measure the time the network took to run
```

בנוסף על כך מצורפים גרפים ומדדים אשר מתארים את למידת המודל וסיכום של הפרמטרים שלו:



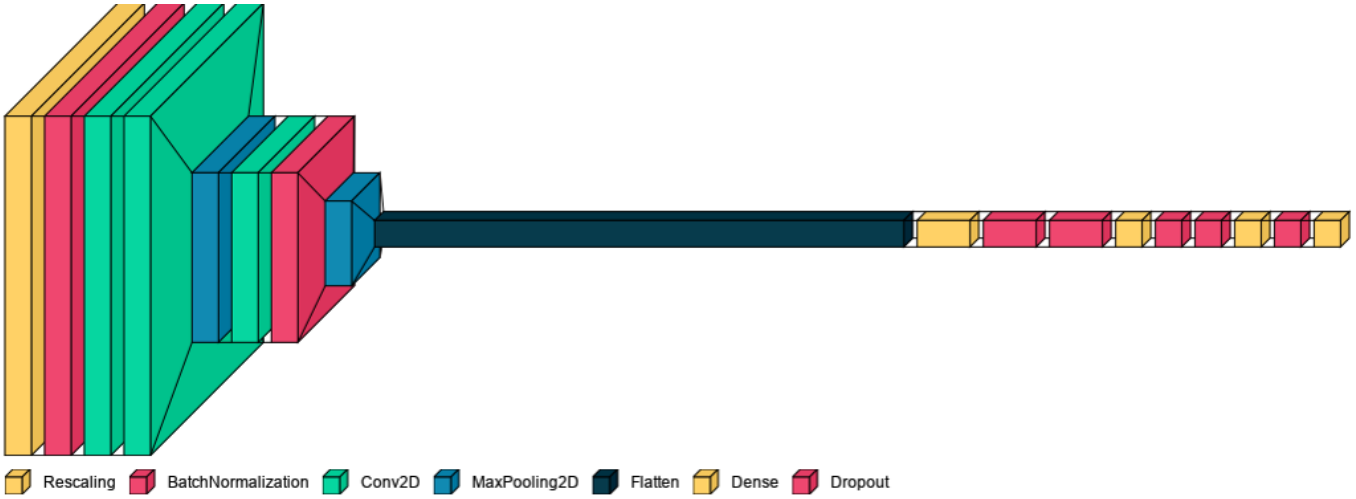


Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 64, 64, 3)	0
batch_normalization (BatchNormalization)	(None, 64, 64, 3)	12
conv2d (Conv2D)	(None, 64, 64, 16)	448
conv2d_1 (Conv2D)	(None, 64, 64, 32)	4640
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	18496
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 400)	6554000
batch_normalization_2 (BatchNormalization)	(None, 400)	1600
dropout (Dropout)	(None, 400)	0
dense_1 (Dense)	(None, 200)	80200
batch_normalization_3 (BatchNormalization)	(None, 200)	800
dropout_1 (Dropout)	(None, 200)	0
dense_2 (Dense)	(None, 100)	20100
dropout_2 (Dropout)	(None, 100)	0
dense_3 (Dense)	(None, 30)	3030

Total params: 6,683,582
Trainable params: 6,682,248
Non-trainable params: 1,334

```
Train accuracy: 0.9488452672958374
val_accuracy: 0.9135952591896057
test_accuracy: 0.9157618880271912

top_k_categorical_accuracy: 0.9918
val_top_k_categorical_accuracy: 0.9814
test_top_k_categorical_accuracy: 0.9793
```

תיאור גרפי של המודל הסופי



תהליך מציאת Hyper-Parameters מתאימים למודל

לאורך אימון המודל ניסיתי מספר רב של קונפיגורציות שונות של Hyper-Parameters ולרובם (קצב הלמידה, Batch size, optimization function ועוד) הגעתי דרך ניסוי וטעיה עד שנמצא ההרכב המתאים שהוביל לתוצאות הטובות ביותר.

ארצה לתת התייחסות מיוחדת לפונקצית ההפסד אשר נבחרה למודל. כפי שציינתי במבוא, מפאת חוסר משאבים (מחשב חזק לדוגמא) ורצון ללמוד מאחרים המודל ההתחלתי שלי היה לקוח מהאינטרנט*. במודל זה, השתמשו בפונקצית הפסד sparse_categorical_crossentropy, פונקציה זו נועדה לעבוד עבור בעיות קלסיפיקציה באופן דומה ל-categorical_crossentropy (ההבדל היחיד ביניהם הוא ש-sparse_categorical_crossentropy מחזירה ערך יחיד והוא המרחק בין הניחוש הנכון לבין ההסתברות שנתנו לו ו-categorical_crossentropy מחזירה את המרחקים בין וקטור התוצאה הרצוי, 1 בקטגוריה הנכונה ו-0 בכל האחרים, לבין וקטור התוצאה של הרשת) ולאורך רוב תהליך אימון המודל נראה שהיה שיפור והתקדמות. למרות זאת, כאשר בדקתי את המודל עצמו באופן ידני (בעזרת הממשק שיצרתי) נראה שהמודל לא הצליח בכלל למרות שעמד על מעל 80% במדד ה-accuracy. לאחר שחקרתי את הנושא לעומק ושילבתי מדד הצלחה חדש, top_k categorical accuracy (ממד זה לוקח את מספר החיזויים שבהם הקטגוריה הנכונה הייתה ב-k הסבירויות הגבוהות ביותר שהמודל יצר ומחלק אותו במספר החיזויים הכולל שהמודל עשה. אני השתמשתי ב-k=3) ראיתי שהמודל כשל לחלוטין (בערך 10% הצלחה). כאשר הבנתי שצריך לשנות את פונקצית ההפסד כדי שהרשת תלמד לא רק מהמרחק שלה מהניחוש הנכון אלא מכל החיזוי שלה הצלחתי להגיע להצלחה של מעל 90% בשני המדדים.

התמודדות עם הטיה ושונות לאורך אימון המודל:

בעיקר בתחילת אימון המודל נתקלתי בבעיה די רצינית של overfitting אשר הגיעה להבדל של מעל 10% בין נתוני האימון לנתוני הבדיקה. כדי לפתור בעיה זו, הוספתי שכבות של Dropout אשר הכריחו את המודל לבצע הכללות (הרי חלק מהנתונים, ליתר דיוק 20% מכל שכבה, "נעלמו"). לאחר ההוספה של שכבות אלו, בעיית ה-overfitting נעלמה ברובה.

בנוסף על כך לקראת סוף אימון המודל שמתי לב שככל שמתקדמים לאורך ה-epochs כך הביצוע של המודל אינו משתפר, אלא דועך במקצת. אני מסיק שזה קורה מכיוון שהמודל מתאים את עצמו

יותר מידי לנתוני האימון ולכן נוצר overfitting. כדי לפתור בעיה זו הקטנתי את מספר ה-epochs בהתאם לתוצאות של ריצות קודמות כך שהלמידה תיעצר רגע לפני שהירידה בביצועים מופיעה.

תהליך ייעול ההתכנסות של המודל

לאורך רוב הפרויקט לא מצאתי ששינוי בקצב הלמידה שלי הוביל לשיפור דרמטי בתוצאות. אני משער שזה היה המצב מכיוון שקצב הלמידה של המודל איתו התחלתי את הפרויקט היה מותאם לנתונים. יחד עם זאת כן הוספתי שכבות של batch normalization (שכבות אשר מנרמלות את הנתונים גם בין שכבות של המודל ולא רק בתחילת המודל) אשר אפשרו לי להגדיל את קצב הלמידה שלי ולהגיע לתוצאות יותר טובות, יותר מהר.

חשוב לציין שעבור המודל שעבר fine tuning היה חשוב מאוד להשתמש בקצב למידה נמוך משמעותית מזה שבו השתמשתי עבור כל שאר המודלים מכיוון שאנו יוצאים מנקודת הנחה שהמשקולות של המודל המוכן מותאמות וטובות ולכן נרצה לשנות אותן מעט מאוד (כדי "לכוונן" אותן לנתונים שלנו) ובעקבות זאת נרצה להשתמש בקצב למידה קטן הרבה יותר.

פונקצית ייעול - Adam optimizer

פונקצית הייעול שבחרתי לפרויקט זה היא adam optimizer.

פונקציה זו נועדה לצמצם את השגיאה של המודל בזמן האימון. Adam פותח מפונקצית הייעול SGD (שזה ראשי תיבות עבור stochastic gradient descent). פונקצית ייעול זו משלבת את היתרונות של שתי שיטות שונות לצמצום השגיאה: AdaGrad ו-RMSProp.

Adam משתמש בקצב למידה אחר עבור כל משקולת (ערכים אלו נקבעים לפי הממוצע של השיפועים - gradients, והממוצע של השיפועים בריבוע), ובכך הרשת יכולה להגיע אל נקודת המינימום של השגיאה בצורה מהירה ויעילה יותר.

שלב היישום - Software development

היישום אשר בחרתי לממש עבור פרויקט זה הוא תוכנה אשר מאפשרת למשתמש לראות בזמן אמת את החיזוי של המודל עבור ציורים שהוא עצמו צייר. נוסף על כך ניתן לטעון תמונות מוכנות ולראות את חיזוי המודל עבורן.

כיצד היישום משתמש במודל

בתחילת הריצה המודל (שנשמר בתור קובץ h5) נטען לתוך המשתנה ai בעזרת הפקודה load_model מספריית keras.

לאחר כל קו שהשחקן מצייר התמונה נחתכת (נפטרים מהקצוות הלבנים שלא צויר בהם כלום), מומרת ל-Tensor אשר מועבר ל-ai בעזרת הפונקציה ()__call__. המודל מבצע חיזוי עבור התמונה ומוחזר וקטור של 30 ערכים בין 0 ל-1 המסמלים את ההסתברויות השונות שהמודל חוזה עבור כל קטגוריה (כל ההסתברויות סוכמות ל-1). לאחר מכן, וקטור זה מוצמד אל רשימה של 30 הקטגוריות כך שלכל קטגוריה מוצמד החיזוי שלה וכל הזוגות הללו יוצרים מילון. המילון ממוין לפי ההסתברויות שמוצגות על המסך יחד עם הקטגוריה התואמת להם. לאחר מכן, נלקחים 5 ההסתברויות הגבוהות ביותר והן מודפסות למסך.

הטכנולוגיה שעל פיה מומש המודל

כדי לממש את המודל השתמשתי בספרית pygame אשר נועדה לבניית משחקים, ממשקים, אמנות דיגיטלית ומוזיקה ב-Python, היא מאפשרת ליצור גרפיקה ואינטראקציה עם מכשירי הקלט של המשתמש (עכבר ומקלדת). הספרייה הינה open-source ומתחזקת סביבה קהילה של מפתחים אשר מעדכנים ומשפרים אותה לעיתים קרובות.

תרשים UML של תוכנת היישום - Graphics_final.py

Class Window:

Parameter	Type	Explanation
presets_dir	str	התיקייה בה יש את התמונות המוכנות שאפשר לטעון
presets	list	רשימה שתחזיק את התמונות בpresets_dir מוכנות לטעינה
offset	int	איזה תמונה היא הראשונה ברשימה שמוצגת על המסך (על המסך אפשר להציג רק 5 תמונות כל פעם מכך offset מצביע על התמונה הראשונה ברשימה של ה5, אם מגלגלים למטה offset קטן וההפך אם מגלגלים למעלה)
background_color	tuple	צבע אזור הציור
screen	pygame.surface	החלון של התוכנה
right_bar_color	tuple	צבע הצד הימני של המסך
left_bar_color	tuple	צבע הצד השמאלי של המסך
drawing_board	pygame.rect	מלבן המייצג את אזור הציור של המסך
right_bar	pygame.rect	מלבן המייצג את האזור הימני של המסך
left_bar	pygame.rect	מלבן המייצג את האזור השמאלי של המסך

Function	Parameters	Return Type	Explanation
clear_screen			מנקה את אזור הציור של המסך
refresh_rightbar	percentage_list: list		מעדכנת את האזור הימני של המסך עם רשימת אחוזים המייצגת את הניחושים של הרשת.
refresh_leftbar	direction: int		מעדכנת את האזור השמאלי של המסך כאשר גוללים עליו כדי להזיז למעלה או למטה את רשימת התמונות שאפשר לטעון

get_screen	compress_size: tuple cut_edges: bool	PIL.Image	מחזירה תמונה של אזור הציור בגודל compress_size בשביל חיזוי עם רשת נוירונים, cut_edges נותן לחתוך קצוות לבנים כדאי שלרשת יהיה קל יותר לחזות
make_prediction			משתמשת בrefresh_rightbar ורשת הנוירונים כדי לחזות את התמונה על המסך ולהציג את האחוזים
run			מתחיל את לולאת המסך ועוקב אחרי ההקשות של שחקן

Final_model.py

תפקיד קובץ זה הוא: לטעון את הנתונים בעזרת ספרית quickdraw, להתאים אותם כך שהמודל יוכל ללמוד בעזרתם, לבנות את המודל, לאמן את המודל ולתעד את מידת ההצלחה של המודל.

תוכן הקובץ:

```
import time as Time


from pathlib import Path

from matplotlib import pyplot as plt

from quickdraw import QuickDrawDataGroup, QuickDrawData


from tensorflow.keras.preprocessing import image_dataset_from_directory  #
remember to install tf-nightly to fix this


from tensorflow.keras.models import Sequential

from tensorflow.keras.losses import categorical_crossentropy

from tensorflow.keras.layers.experimental.preprocessing import Rescaling

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
Dropout, BatchNormalization


from keras.applications import MobileNetV2


import tensorflow as tf

import keras

from keras import layers
```

```
image_size = (64, 64)  # this is the image resolution for the data


categories = ['airplane', 'apple', 'axe', 'basketball', 'bicycle',
'broccoli', 'bucket', 'butterfly',

             'crab', "diamond", 'fence', 'fish', "guitar", 'hammer',
'headphones', 'helicopter',

             'hot air balloon', 'ice cream', 'light bulb', 'lollipop', 'palm
tree', 'parachute',

             'rainbow', 'sailboat', 'shoe', 'smiley face', 'star', "tennis
racquet", 'traffic light',

             'wristwatch']  # categories for images in training data




#  loading the data

def generate_class_images(name, max_drawings, recognized):

   # this function creates a subdirectory named <name> in dataset_30 which
has <max_drawings> images of class <name>.

   # the function only loads images that Google's network correctly
identified


   directory = Path("dataset_30/" + name)  # specifying the path of the
dataset


   if not directory.exists():

       directory.mkdir(parents=True)  # make a subdirectory for the category
in dataset_30


   images = QuickDrawDataGroup(name, max_drawings=max_drawings,
recognized=recognized)  # load the images using the


# quick draw library

   for img in images.drawings:

       filename = directory.as_posix() + "/" + str(img.key_id) + ".png"  #
name the image

       img.get_image(stroke_width=3).resize(image_size).save(filename)  #
make the image 64 by 64



c = True  # deciding whether to load the images or not
```

```
# generating 7000 images for each category in categories and putting them
in subdirectories in dataset_30

if c:

   for label in categories:

       generate_class_images(label, max_drawings=7000, recognized=True)


print("\n\nFinished loading images for
dataset_30-----------------------------------------------------------------
--")


load_dataset = True  # deciding whether to create the files used for the
actual training or not


if load_dataset:

   # preparing the data

   batch_size = 128  # because we load the images using
image_dataset_from_directory() they are

   # already saved in a format ready for training a neural network, so we
need to specify the batch size


   # this function will return an image dataset where each image is paired
with a label corresponding

   # to the class it belongs to (i.e. for classes a and b pictures from
class a will have a label 0

   # and pictures from class b will have label 1)

   train_ds = image_dataset_from_directory(  # training dataset

       "dataset_30",

       validation_split=0.2,

       subset="training",

       seed=123,

       color_mode="rgb",

       image_size=image_size,

       batch_size=batch_size,

       label_mode='categorical'

   )


   val_ds = image_dataset_from_directory(  # validation dataset
```

```
        "dataset_30",
        validation_split=0.2,
        subset="validation",
        seed=123,
        color_mode="rgb",
        image_size=image_size,
        batch_size=batch_size,
        label_mode='categorical'
    )

    train_ds.save('training_dataset')
    val_ds.save('validation_dataset')

train_ds = tf.data.Dataset.load('training_dataset')  # the training dataset
val_ds = tf.data.Dataset.load('validation_dataset') # the validation dataset

# building, compling the training the model

input_shape = (image_size[0], image_size[1], 3)  # choosing the input shape
of the model

n_classes = len(categories)  # number of categories in training set data

# creating the model we will train
model = Sequential([
    Rescaling(1. / 255, input_shape=input_shape),
    BatchNormalization(),

    Conv2D(16, kernel_size=(3, 3), padding="same", activation="relu"),
    Conv2D(32, kernel_size=(3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), padding="same", activation="relu"),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
```

```
    Flatten(),

    Dense(400, activation='relu'),
    BatchNormalization(),
    Dropout(0.2),

    Dense(200, activation='relu'),
    BatchNormalization(),
    Dropout(0.2),

    Dense(100, activation='relu'),
    Dropout(0.2),

    Dense(n_classes, activation='softmax')
])

# Train the model
opt = tf.optimizers.Adam()  # choosing the Adam optimizer


#  using the top k categorical accuracy which means it measures the number
#  of times the correct category of the image was in the 3 categories which
#  received the highest probability by the network and divides it by the
#  overall predictions the network made
top_k_categorical_accuracy = tf.keras.metrics.TopKCategoricalAccuracy(
    k=3, name='top_k_categorical_accuracy', dtype=None
)


model.compile(
    optimizer=opt,
    loss='categorical_crossentropy',  # using the categorical_crossentropy
    # loss function because we are trying to differentiate between a number
    # of categories
```

```
    metrics=["accuracy", top_k_categorical_accuracy]
)


model.summary()  # printing a summary of the model's architecture
epochs = 4  # number of epochs the network will run for

tic = Time.time()  # starting to measure the time the network took to run

history = model.fit(  # training the model and saving the results to
history
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    verbose=1,
)

toc = Time.time()  # finishing to measure the time the network took to run

# Calculate training time and format as min:sec
minutes = format((toc - tic) // 60, '.0f')
sec = format(100 * ((toc - tic) % 60) / 60, '.0f')
print(f"Total training time (min:sec): {minutes}:{sec}")

# ----------------------------------------------------------------------

# printing and graphing the results

print("Train accuracy: " + str(max(history.history['accuracy'])))
print("val_accuracy: " + str(max(history.history['val_accuracy'])))

plt.title('learning curves')
plt.xlabel('epoch')
plt.ylabel('Loss (cross entropy)')
plt.plot(history.history['loss'], label='train')
```

```
plt.plot(history.history['val_loss'], label='val')
plt.legend()
plt.show()

plt.title('learning curves')
plt.xlabel('epoch')
plt.ylabel('Accuracy')
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='validation_acc')
plt.legend()
plt.show()

# saving the model
name = "final_model"  # name of the model
model.save("models/" + name + ".h5")
print("Saved model to disk")
```

Graphics_final.py

תפקיד הקובץ הוא לממש את חלק היישום בפרויקט. קובץ זה יוצר את הגרפיקה, מאפשר ציור של תמונות, העלאת תמונות שמורות אל המסך, התאמת התמונות לחיזוי על ידי המודל וביצוע הקריאה למודל.

תוכן הקובץ:

```
import os


import pygame
from PIL import Image  # remember to install the library Pillow


import tensorflow
from tensorflow.keras.models import load_model



class Window:
   def __init__(self):
       pygame.init()
       self.presets_dir = "Presets"  # the directory in which we store the
saved images
       if not os.path.exists(self.presets_dir):   # if the directory
"Presets" does not exist, create it
           os.mkdir(self.presets_dir)
       self.presets = []  # list for the preset images
       for directory in os.listdir(self.presets_dir):
           self.presets.append((directory[:-4], Image.open(self.presets_dir +
"\\" + directory)))
       self.offset = 0  # offset from the start of the list on the left_bar
of the screen


       # colors for the three areas of the window
       self.background_color = (255, 255, 255)
       self.right_bar_color = (255, 128, 0)
       self.left_bar_color = (101, 255, 192)
       window_size = (500, 300)  # choosing the window size
       self.screen = pygame.display.set_mode(window_size)
       pygame.display.set_caption('Image recognition')
       self.screen.fill(self.background_color)
       # 3 rectangles representing the 3 areas of the window, the left bar,
the drawing_board and the right bar
       self.drawing_board = pygame.draw.rect(self.screen,
self.background_color, pygame.Rect(100, 0, 300, 300))
```

```
        self.right_bar = pygame.draw.rect(self.screen, self.right_bar_color,
pygame.Rect((400, 0, 100, 300)))

        self.left_bar = pygame.draw.rect(self.screen, self.left_bar_color,
pygame.Rect((0, 0, 100, 300)))


        pygame.display.flip()  # this is intended to update the screen


        self.font = pygame.font.SysFont('freesansbold.ttf', 15)

        self.left_bar_font = pygame.font.SysFont('freesansbold.ttf', 20)


    # clears the drawing area of the window

    def clear_screen(self):

        pygame.draw.rect(self.screen, self.background_color,
self.drawing_board)


    # refreshes the right bar of the window with percentage list, if
percentage list is

    # not given it only refreshes the rectangle with no list

    def refresh_rightbar(self, percentage_list: list = None):

        # redraws the sidebar

        pygame.draw.rect(self.screen, self.right_bar_color, self.right_bar)

        # if you give it a list to project

        if percentage_list is not None:

            # sorting from Google

            sorted_list = sorted(percentage_list, key=lambda x: x[1],
reverse=True)

            # get first 5 items (first = strong,big,great last =
weak,small,bad)

            final_list = sorted_list[:5]

            for i in range(5 * 2):

                if i % 2 == 0:

                    text_temp = str(final_list[round(i / 2)][0])

                    if final_list[round(i / 2)][0] == "The Eiffel Tower":

                        text_temp = "Eiffel Tower"

                else:

                    text_temp = str(round(final_list[round(i / 2)][1], 5))

                # make text from item and percentage
```

```
                text = self.font.render(text_temp, True, (0, 0, 0))
                # text = self.font.render(final_list[i][0] + ' ' +
str(final_list[i][1]), True, (0, 0, 0))
                # get its rect
                text_rect = text.get_rect()
                # place center where I want it to be (7/8ths of screen
width, 25 + 50 space per text)
                text_rect.center = (self.right_bar.center[0], 25 + 50 * i /
2)
                # draw text on to sidebar
                self.screen.blit(text, text_rect)


    # refreshes the left bar of the window, if the direction is not 0 it
will scroll the preset
    # list by adding or subtracting offset
    def refresh_leftbar(self, direction=0):
        if direction != 0 and 0 <= self.offset + direction <=
len(self.presets) - 5:
            self.offset += direction
        pygame.draw.rect(self.screen, self.left_bar_color, self.left_bar)
        # for i in range(self.offset, (5 + self.offset if len(self.presets)
>= 5 else len(self.presets))):
        for i in range((5 if len(self.presets) >= 5 else len(self.presets))):
            text_temp = str(self.presets[i+self.offset][0])
            # make text from item and percentage
            text = self.left_bar_font.render(text_temp, True, (0, 0, 0))
            # text = self.font.render(final_list[i][0] + ' ' +
str(final_list[i][1]), True, (0, 0, 0))
            # get its rect
            text_rect = text.get_rect()
            # place center where I want it to be (7/8ths of screen width, 25
+ 50 space per text)
            text_rect.center = (self.left_bar.center[0], 50 + 50 * i)
            # draw text on to sidebar
            self.screen.blit(text, text_rect)


    # returns PIL.Image in size compress_size in order to use the neural
network
```

```
    # if cut_edges is true it will trim away any extra pure white edges
    def get_screen(self, compress_size=(64, 64), cut_edges=False):
        # get the drawing area of the screen
        sub_surface = self.screen.subsurface(pygame.Rect(100, 0, 300, 300))
        # get the pixel data as an array from the subsurface
        pixel_data = pygame.surfarray.array3d(sub_surface)
        # convert the pixel data into a PIL Picture
        img = Image.fromarray(pixel_data)

        if cut_edges:
            nonwhite_positions = [(x, y) for x in range(img.size[0]) for y in
range(img.size[1]) if img.getdata()[x + y * img.size[0]] != (255, 255, 255)]
            if len(nonwhite_positions) != 0:
                rect = (min([x for x, y in nonwhite_positions]), min([y for x,
y in nonwhite_positions]), max([x for x, y in nonwhite_positions]), max([y
for x, y in nonwhite_positions]))
                img = img.crop(rect)

        output_data = img.resize(compress_size).transpose(Image.TRANSPOSE)
        return output_data

    # predict the image currently drawn and update the right bar
    def make_prediction(self):
        tensor = tensorflow.convert_to_tensor(self.get_screen(cut_edges=True))
        tensor = tensorflow.image.resize(tensor, [64, 64])
        input_tensor = tensorflow.expand_dims(tensor, axis=0)
        results = ai.__call__(input_tensor, training=False)
        results_list = tensorflow.Variable(results).numpy().tolist()
        results_list = results_list[0]
        categories = ['airplane', 'apple', 'axe', 'basketball', 'bicycle',
'broccoli', 'bucket', 'butterfly',
                      'crab', "diamond",
                      'fence', 'fish', "guitar", 'hammer', 'headphones',
'helicopter', 'hot air balloon',
                      'ice cream',
                      'light bulb', 'lollipop', 'palm tree', 'parachute',
'rainbow', 'sailboat', 'shoe',
```

```
                'smiley face', 'star',
                "tennis racquet", 'traffic light', 'wristwatch']
    prediction = dict(zip(results_list, categories))
    # sorted_prediction = sorted(prediction)
    prediction_to_print = []
    for key in prediction.keys():
        prediction_to_print.append((prediction[key], key))
    self.refresh_rightbar(prediction_to_print)

# window loop
def run(self):
    running = True
    last_mouse_x, last_mouse_y = None, None
    last_left, last_middle, last_right = None, None, None
    last_keys = None
    self.refresh_leftbar()
    drawing_hold = None
    history = []
    while running:
        wheel_y = 0
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEWHEEL:
                wheel_y = event.y

        mouse_x, mouse_y = pygame.mouse.get_pos()
        left, middle, right = pygame.mouse.get_pressed()
        keys = pygame.key.get_pressed()

        # if mouse was pressed but not held it saves the image
        if left and not last_left:
            drawing_hold = self.get_screen((300, 300))
```

```
            # if mouse was released it appends to history the saved image
and if history is too big trims it.
            if last_left and not left:
                history.append(drawing_hold)
                if len(history) > 15:
                    history.pop(0)
                drawing_hold = None


            # is z is pressed not held and there is something to restore it
will restore the last saved image
            if keys[pygame.K_z] and not last_keys[pygame.K_z] and
len(history) > 0:
                to_load = history[-1]
                history.pop(-1)
                pygame_surface = pygame.image.fromstring(to_load.tobytes(),
to_load.size, "RGB").convert()
                self.screen.blit(pygame_surface,
pygame_surface.get_rect(center=self.drawing_board.center))
                self.make_prediction()


            # is s is pressed but not held it will save the image currently
drawn to presets with a name given in console
            if keys[pygame.K_s] and not last_keys[pygame.K_s]:
                file_name = input("name the picture:\n")
                self.get_screen((300, 300)).save('Presets\\' + file_name +
'.png')
                self.presets = []
                for directory in os.listdir(self.presets_dir):
                    self.presets.append((directory[:-4],
Image.open(self.presets_dir + "\\" + directory)))
                self.refresh_leftbar()


            # if the mouse is over the left bar of the window and is
scrolling refresh the left bar
            if mouse_x < 100 and wheel_y != 0:
                self.refresh_leftbar(-wheel_y)
```

```
            # if left mouse was clicked but not held and is over the
left_bar load the image being clicked
            if left and not last_left and mouse_x < 100:
                row = (mouse_y - 25) // 50
                if row < 5:
                    to_load = self.presets[row + self.offset][1].resize((300,
300)).convert("RGB")
                    pygame_surface =
pygame.image.fromstring(to_load.tobytes(), to_load.size, "RGB").convert()
                    self.screen.blit(pygame_surface,
pygame_surface.get_rect(center=self.drawing_board.center))
                    self.make_prediction()


            # if the mouse is on the drawing area and left click is clicked
or held draw a line from the last location and the current one
            if last_mouse_x is not None and 100 <= last_mouse_x < 400 and
last_mouse_y is not None and left:
                pygame.draw.line(self.screen, (0, 0, 0), (mouse_x, mouse_y),
(last_mouse_x, last_mouse_y), 6)
                self.refresh_leftbar()


            # if mouse is over the drawing area of the window and released
predict the image that was drawn
            if last_mouse_x is not None and 100 <= last_mouse_x < 400 and
last_mouse_y is not None and not left and last_left:
                self.make_prediction()


            # if right click was clicked not held
            if right and not last_right:
                history.append(self.get_screen((300, 300)))
                if len(history) > 15:
                    history.pop(0)
                self.clear_screen()


            last_mouse_x, last_mouse_y = mouse_x, mouse_y
            last_left, last_middle, last_right = left, middle, right
            last_keys = keys
            pygame.display.update()
```

```python
if __name__ == '__main__':
    # load model
    # ai = load_model('transfer_dropout_batch-norm.h5')
    ai = load_model("models/final_model.h5", compile=False)
    print(['airplane', 'apple', 'axe', 'basketball', 'bicycle', 'broccoli',
           'bucket', 'butterfly', 'crab', "diamond", 'fence', 'fish',
           "guitar", 'hammer'])
    print(['headphones', 'helicopter', 'hot air balloon',
           'ice cream', 'light bulb', 'lollipop', 'palm tree', 'parachute',
           'rainbow', 'sailboat', 'shoe'])
    print(['smiley face', 'star', "tennis racquet", 'traffic light',
           'wristwatch'])
    window = Window()
    window.run()
```

מדריך למפתח

טעינת הנתונים, בניית הרשת ואימונה - final_model.py

כדי להריץ את הלמידה יש צורך ב-:

- לפתוח פרויקט חדש ב-Pycharm

- לוודא שיש גרסה מספר 3.8 של Python
- להוריד את הספריות: יש להוריד את הספריות הבאות לפרויקט: matplotlib, quickdraw, tensorflow.
- להכניס את הקובץ final_model.py אל תוך הפרויקט.
- ליצור תיקיה ריקה בשם models.
- להריץ את final_model.py.

תוכנת היישום - Graphic_final.py

כדי לטעון ולהריץ את היישום של הפרויקט יש צורך ב-:

- לפתוח פרויקט חדש ב-Pycharm
- לוודא שיש גרסה מספר 3.8 של Python
- להוריד את הספריות: יש להוריד את הספריות הבאות לפרויקט: pygame, Pillow, tensorflow.
- להכניס את הקובץ Graphics_final.py ואת התיקיות: models (שבתוכה יש רק את הקובץ final_model.h5) ו-Presets (המכילה בתוכה תמונות מכל מיני קטגוריות).
- להריץ את הקובץ Graphics_final.py.

מדריך למשתמש

בכדי להריץ את היישום של הפרויקט, יש צורך בקובץ Graphics_final.py, בתיקייה Presets ובתקייה models אשר מכילה את הקובץ final_model.h5. כאשר מריצים את Graphics_final.py יעלה למסך החלון:

חלון זה הוא החלון היחיד ודרכו ניתן לבצע את רוב הפעולות האפשריות.

בנוסף על כך יודפס ל-Console רשימה של כל הקטגוריות האפשריות לציור.

הפעולות האפשריות ביישום הן:

- החזקת מקש שמאלי ומשיכה על החלק הלבן כדי לצייר בשחור על החלק הלבן.
- לחיצת מקש ימני מוחקת את הציור הנוכחי.
- לחיצת Ctrl z כדי למחוק את הקו האחרון שצויר (זוכר עד 15 שלבים אחורה).
- לחיצת מקש שמאלי על טקסט במלבן השמאלי טוען למסך הלבן ציור באותו השם אשר שמור בתיקייה Presets.
- אם העכבר נמצא בתוך המלבן השמאלי ניתן לגלול למעלה או למעלה עם הגלגלת כדי לעבור בין שמות שונים של תמונות אשר ניתן לטעון למסך בעזרת מקש שמאלי כפי שתיארתי.
- לחיצת s מדפיס ל-Console את המשפט:

```
name the picture:
```

על המשתמש לכתוב את שם שהוא רוצה לתת לציור ואז הציור הנוכחי על המסך הלבן ישמר לתיקייה Presets בתור קובץ png עם השם שניתן כקלט. מצורפת תמונה המדגימה פעולה זו:

```
name the picture:
palm tree
```

לאחר כל שינוי במסך הלבן (הוספת קו, מחיקת קו בעזרת Ctrl z וטעינת תמונה למסך) פרט למחיקה שלו (מקש ימני) על המלבן הימני יופיעו 5 קטגוריות מתוך ה-30 הקיימות ומתחת לכל אחת מהן יופיע מספר בין 0 ל-1. מספרים אלו מייצגים את החיזוי של המודל (המספר מעיד על ההסתברות שלפיה המודל סבור שזו הקטגוריה של הציור). 5 הקטגוריות שהופיעו הן אלו בעלות ההסתברויות הגבוהות ביותר.

רפלקציה / סיכום אישי

הפרויקט הזה מאוד עזר לי להבין את החומר אותו למדנו לאורך השנה בכל הנוגע לרשתות עמוקות מכיוון שהוא דרש ממני ממש להשתמש ולהיות מסוגל להסביר כיצד המודל שלי עובד ובכך שיפר את הלמידה שלי. מעבר לכך למדתי כיצד לחפש פתרונות לבאגים ובעיות הקשורים לנושא של למידה עמוקה. בסה"כ אני חושב שקיבלתי כמה כלים שימושיים לחיים מעבר לפרויקט שאוכל להציג בעתיד.

העבודה על הפרויקט עצמו הייתה לעיתים מהנה ולעיתים סיזיפית ומתסכלת. כאשר עבדתי על הבנייה של הרשת עצמה ויותר התעסקתי בנושאים התיאורטיים של לנסות להבין כיצד הרשת עובדת ומה מיוחד בה או ברשתות אחרות שמצאתי באינטרנט נהניתי והרגשתי שאני משיג ידע חשוב ומעניין. לעומת זאת כאשר היה עליי להתמודד עם באגים ובעיקר כאלו שקשורים להעברת התמונה ביישום למבנה נתונים המתאים ליישום הרגשתי מתוסכל שכן כל המימושים של הפונקציות שהשתמשתי בהן הינם מסובכים ואינם קשורים לחומר (לדוגמא כיצד התמונה נשמרת ב-PyCharm ומה ההבדל בין לשמור ואז לטעון אותה למודל אל מול ישר לטעון אותה בלי לשמור קודם) והתיקון שלהם כלל בעיקר חיפוש באינטרנט והרבה ניסוי וטעייה אשר הרגישו סיזיפיים ומתישים.

כפי שציינתי במבוא, במהלך העבודה נתקלתי במספר קשיים מרכזיים. ביניהם נכללו העברת התמונות אשר נוצרות ביישום לפורמט מתאים לחיזוי, בניית רשת מתאימה ועוד. ההתמודדות עם הקשיים הללו בהחלט עזרה לי הן עם ההבנה של החומר והן עם יכולות התכנות שלי.

כעת כשאני בסוף הפרויקט הגעתי למספר מסקנות:

ראשית, אין צורך מהותי במחשב חזק במיוחד כדי להשיג למידת מכונה משמעותית. אני מסיק זאת מכיוון שהצלחתי בעזרת מחשב שאינו חזק במיוחד להכין מודל אשר מזהה באחוזים מאוד גבוהים ציורים ולאמן אותו בזמן סביר (בין שעתיים ל-3 שעות).

שנית, מחשב מסוגל למצוא דפוסים בציורים פשוטים של בני אדם ולכן ניתן בעתיד להרחיב את הפרויקט לזיהוי של דברים מורכבים יותר (כגון כתב יד, שרטוטים ותכנונים הכתובים ביד ועוד).

שלישית, לעיתים הפתרונות המסובכים יותר אינם המועדפים שכן בפרויקט זה לדוגמא רשת קונבולוציה רגילה הצליחה להשיג את התוצאות הטובות ביותר.

לו הייתי מתחיל היום את הפרויקט מחדש הייתי מתמקד יותר על generative neural networks מאשר על discriminative neural networks כדי לנסות ליצור ציורים פשוטים בעזרת שם של קטגוריה מכיוון שזה נושא שפחות התעמקנו בו בכיתה והוא נראה די מעניין בעיני. מעבר לכך אני מרוצה בדרך שבה עשיתי את הפרויקט ולא הייתי משנה שום דבר משמעותי.

לסיכום, פרויקט זה היה משמעותי בשבילי ועזר לי להבין וליישם את החומר הנלמד בצורה יעילה טובה. לאחר סיומו אני נשאר עם מגוון שאלות מסקרנות ומרתקות שאוכל לחקור בעצמי כגון זיהוי דברים מסובכים יותר מציורים פשוטים ויצירה של ציורים בעזרת מתן הקטגוריה כקלט.

ביבליוגרפיה

1. Vogel, A. (2020, September 3). *Image Classification using CNN - Google Quick, Draw!* Github. https://github.com/amelie-vogel/image-classification-quickdraw
2. Kothawade, D., Kazi, I., Mhatre, J., Nair, K., & Nitnaware, P. (2020). Recognition of Labels for Hand Drawn Images. *International Research Journal of Engineering and Technology*, *7*(5). https://www.irjet.net/archives/V7/i5/IRJET-V7I5351.pdf
3. Pande, K. (2018, December 15). *Doodling with Deep Learning!* Towards Data Science. https://towardsdatascience.com/doodling-with-deep-learning-1b0e11b858aa
4. Phu, N., Xiao, C., Muindi J. *Drawing: A New Way To Search (Computer Vision).* Stanford University. *https://cs229.stanford.edu/proj2018/report/33.pdf*
5. (n.d.). *Fine-Tuning MobileNet On Custom Data Set With TensorFlow's Keras API*. DEEPLIZARD. https://deeplizard.com/learn/video/Zrt76AIbeh4
6. Google (n.d.). *The Quick, Draw! Dataset*. Github. https://github.com/googlecreativelab/quickdraw-dataset
7. Ma, S. (2020, December 6). *Doodling Image Recognition: The Power of Deep Learning Models!* Towards Data Science. https://towardsdatascience.com/doodling-image-recognition-5ab25476a728
8. Wächter, L. (2022, April 2). *Recognizing hand drawn Doodles using Deep Learning*. Larswaechter.dev. https://larswaechter.dev/blog/recognizing-hand-drawn-doodles/