Assignment 2 - Report

Shahar Oded, 208388918
Nir Rahav, 316275437

הקמה של הפרויקט ובניית ה-dataloader:

בשלב ראשון הורדנו את קובץ ה-ZIP עם התמונות, חילצנו ממנו הקבצים וכתבנו פונקציונאליות על מנת להבין באופן ראשוני את מבנה הנתונים כולל צפייה בתמונות.
שני קבצי הטקסט train.txt ו-test.txt מגדירים לנו עבור התיקיות, ששם כל תיקייה הוא שם של אדם וכל תיקייה מכילה מספר של תמונות, אילו תמונות הן של אותו אדם ואילו של אנשים שונים. למשל, עבור השורה:

Abdullah_Gul 13 14

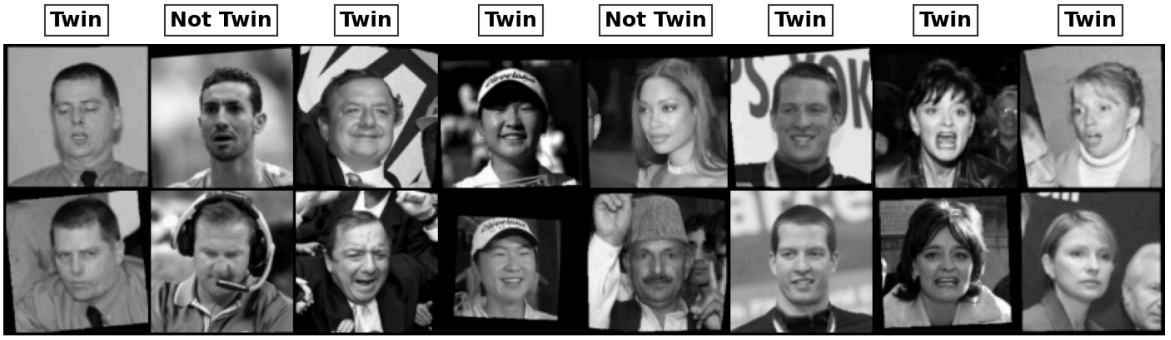
אנחנו יודעים שתמונות 13 ו-14 בתוך התיקייה בשם Abdullah_Gul הן תמונות של אותו האדם, בעוד עבור שורות כמו:

Seth_Gorney 1 Wilton_Gregory 1

אנחנו יודעים שתמונות מספר 1 בתיקיות של Wilton_Gregory ושל Seth_Gorney לא מייצגות את אותו האדם.

בשלב הראשון נדרשנו לאפיין פונקציה כללית להדפסת התמונות ומחלקת dataset המטפלת ביצירת Dataset, שגם מספק בהתאם להנחיות את התמונות כטנזור, בצמדים, עם label מתאים. אובייקט זה מקבל נתיב לתיקייה המכילה את כל תיקיות התמונות ונתיב לקבצי הטקסט המגדירים את הצמדים. פונקציית עזר נוספת במחלקה הופכת את ה-dataset לאובייקט dataloader, לצורך הזנה למודל. החלוקה היא מכיוון שאובייקט Dataset ניתן לחלק, פונקציונאליות שנצטרך. הקוד שמור במודל dataset.py המנהל את הפרסור, טעינה וחלוקה של הנתונים בהמשך. בנוסף, מודול זה מממש פונקציה לביצוע data_augmentation שלוקחת זוגות של תמונות מתוך הדאטה, מעוותת אותן באופן רנדומלי (רעש, סיבוב, טשטוש ועוד) ושומרת חזרה, בזמן ריצה (מבלי להוסיף את התמונות לתיקיות הלוקאליות). המוטיבציה לאור הבחנה ראשונית כי מדובר בדאטה סט קטן מאוד שעלול להוביל ל-overfitting.

במסגרת ה-main.py הוספנו פונקציה (visualize_data) המאפשרת הצגה של batch תמונות באופן הבא, כוידוא שאכן חלק זה הוקם כראוי:



ה-data (המקורי) המוקצה למשימה מכיל:

Train:
[Dataloader Info]: Total number of pairs (rows in file): 2200
[Dataloader Info]: Total positives detected: 1100
[Dataloader Info]: Total negatives detected: 1100

Test:
[Dataloader Info]: Total number of pairs (rows in file): 1000
[Dataloader Info]: Total positives detected: 500
[Dataloader Info]: Total negatives detected: 500

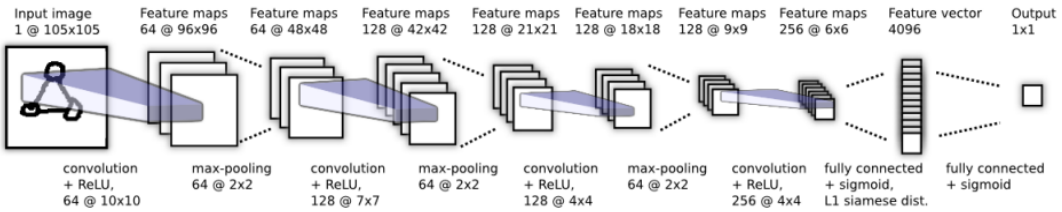
אפיון המודל:

בשלב שני בנינו את המודול model.py המכיל את המודל. המודל מורכב מצירופי בלוקים של CNN ובלוקים של Fully Connected NN, לפי דרישה. המשקלים בכל שכבת CNN נקבעים מתוך התפלגות נורמלית בעלת סטיית תקן של 0.01, המשקלים של שכבות FC נקבעים מתוך נורמלית דומה רק עם סטיית תקן 0.2, והביאסים נגזרים מתוך התפלגות נורמאלית עם ממוצע 0.5 וסטיית תקן של 0.01, לכל השכבות, בדומה למאמר. הקמנו את המודל כך שמסוגל לקבל חיצונית את הפרמטרים לכל שכבה, כך שהגדרות הבניה והריצה המלאות יוכלו להיות מרוכזות בקובץ config.py.

בתור נקודת התחלה בחרנו להתחקות אחרי הארכיטקטורה המתוארת במאמר ולכן הזנו את השכבות באופן הבא:

```
# Model Parameters:
DROPOUT = 0.0
BATCHNORM = False
# CNN Block Configurations
CNN_BLOCKS = [
    # Original architecture
    {"out_channels": 64, "kernel_size": 10, "stride": 1, "padding": 0, "use_pooling": True,
"use_batchnorm": BATCHNORM, "dropout_prob": DROPOUT},   # 105x105 -> 48x48
    {"out_channels": 128, "kernel_size": 7, "stride": 1, "padding": 0, "use_pooling": True,
"use_batchnorm": BATCHNORM, "dropout_prob": DROPOUT},   # 48x48 -> 21x21
    {"out_channels": 128, "kernel_size": 4, "stride": 1, "padding": 0, "use_pooling": True,
"use_batchnorm": BATCHNORM, "dropout_prob": DROPOUT},   # 21x21 -> 9x9
    {"out_channels": 256, "kernel_size": 4, "stride": 1, "padding": 0, "use_pooling": False,
"use_batchnorm": BATCHNORM, "dropout_prob": DROPOUT},  # 9x9 -> 6x6
]

FC_LAYERS = [
    {"in_features": 256 * 6 * 6, "out_features": 4096, "use_batchnorm": BATCHNORM, "dropout_prob":
DROPOUT}, # Fully Connected Layer, # 6x6 -> 1X4096
    {"in_features": 4096, "out_features": 1, "use_batchnorm": False, "dropout_prob": 0.0},  # Final
similarity score layer (1X4096 -> 1X1)
]
```

התמונות במאגר הן בגודל 250X250. בשלב ראשון, רצינו להיצמד לארכיטקטורה המקורית (המוצגת בשכבות מעלה), עבור תמונה בגודל 105X105, ולכן כיווצנו את הקלט. הארכיטקטורה המתוארת:



היות ומדובר בארכיטקטורה עם מספר שלבים הקיבוע של הבלוק עם גדלי pooling מסוימים ו stride=1 עבור הקונבולוציה עזרו לנו פחות להתפזר בניסויים השונים. אנחנו נניח שהיחסים בין החלקים הפנימיים טובים, ובמידת הצורך נעשה ניסויים עם כמות וגדלי הבלוקים השונים. בתוך המחלקה עבור בלוק ה-CNN הגדרנו שבמידה ובלוק זה מכיל max pooling, יוגדר לו stride=2 ו-kernel=2, בהתאם לארכיטקטורה.

בנוסף, הארכיטקטורה שלנו מאפשרת הוספת batchnorm לכל בלוק CNN ו-FC בנפרד, לפי דרישה.

כמו כן, הוספנו אפשרות בתוך המודל לרגולריזציה באמצעות dropout של אחוז יחסי מהנוירונים הפעילים בבלוק. פרמטר זה גם הוא ניתן להגדרה בתוך הקונפיגורציה. נציין שהמאמר לא מדבר על שימוש בשיטה זו, ולכן הדיפולט שהגדרנו הוא 0.0, ונתנסה עם האופציה במידת הצורך בלבד.

בהמשך הניסוי רצינו גם לנסות לעבוד עם התמונות בגודלן המקורי, בתקווה לשפר ביצועים. לשם כך עשינו שינויים בארכיטקטורה כך שתתאים לגדלים המעודכנים. נציין בדוח זה את הארכיטקטורה שהגיעה לביצועים הטובים ביותר.

על מנת לנהל יחסית בקלות את הניסויים הללו, בנינו פונקציית עזר (calculate_conv_out) המחשבת את גודל הoutput מכל שכבות הקונבולוציה שהוגדרו ב-config.py, ביחס לגודל ה-input שהוגדר (מימדי התמונה).

```
Output size after Conv + Pooling: 6X6X256=9216
```

<u>אימון המודל</u>

את הפונקציה train הגדרנו בתוך המודול model.py והיא האחראית על הלמידה. הפונקציה מקבלת את הפרמטרים שלה כ-input, ואלו יהיו הפרמטרים הגלובליים מתוך config.py אשר יועברו במסגרת ה-main. לולאת האימון שלנו מממשת את הדברים הבאים:

- חלוקה של אובייקט ה-dataset ל-train, val (שני אובייקטי dataset חדשים) באמצעות יחס 30% לוולידציה תוך שמירה על חלוקה שוויונית של מחלקות (0,1) באמצעות stratified_split הממומש במודול dataset.py. לאחר החלוקה של קובץ ה-train קיבלנו:

  [Data Distribution]: Subset Train - Positives: 880, Negatives: 880
  [Data Distribution]: Validation - Positives: 220, Negatives: 220

- ביצוע augmentation עבור ה-Train בלבד כך שיגדל פי 9 (בדומה למאמר, עם affine distortion):
  [Augmentation]: Dataset augmented. Original size: 1540, New size: 13860
  [Data Distribution]: Subset Train - Positives: 6930, Negatives: 6930
  [Data Distribution]: Validation - Positives: 330, Negatives: 330
- בניית Dataloader עבור שני הסטים.
- הגדרת אופטימייזר ADAM (למרות השימוש במאמר ב-SGD, שמצאנו שהיה משמעותית פחות מוצלח).

הפונקציה יודעת לקבל LEARNING RATE ו-L2_REG, על מנת לבצע רגולריזציה על המשקלים. בהמשך, במסגרת הניסויים, נציג את הפרמטרים שנקבעו לכל ניסוי. הגדרנו גם scheduler עבור LR, כך שיקטין את ה-LR לקראת התכנסות (בשונה מהמאמר שהגדיר ירידה הדרגתית של 1% בכל epoch). נציין כי בשלבים מוקדמים בניסויים הבחנו כי המודלים נוטה לסבול מ-overfit רציני ולכן הצבנו פרמטרים מחמירים עבור רגולריזציות, הגדרנו שימוש ב-LR גבוה יחסית, הגדרנו BATCH_SIZE קטן (להגביר השפעה של הרגולריזציה בכל צעד) ועשינו שימוש ב-augmentation שבמסגרתה אנחנו מגדילים משמעותית את מספר הדגימות תוך הוספת רעש, סיבוב של התמונה ושינויי מימדים.

- עבור כל epoch הפונקציה מחשבת val_loss, ומקיימת תנאי עצירה מוקדם אם הערך לא שיפר את best_val_loss במשך early_stop_patience איטרציות (לשם הגדרה זו, איטרציה=epoch). הגדרה זו מאפשרת לנו מראש לתת למודל לרוץ לאורך הרבה איטרציות, תוך הבטחה שלא יכנס ל-overfitting אחרי שהגיע לאופטימום טוב, ויעצור כשימקסם ביצועיו (בתקווה שאלו יהיו סבירים).
- הפונקציה שומרת ערכי loss על שני הסטים, לצורך plot בסוף האימון. כמו כן, בכל איטרציה מודפס מדד ה-accuracy על סט האימון וסט הבדיקה, לצד זמן האימון עד לנקודה זו וה-LR שנקבע ע"י ה-scheduler.
- הפונקציה שומרת state_dict של המודל הטוב ביותר (לפי val_accuracy) בתיקיית save_path בסיום האימון המאפשר שחזור המודל ותוצאותיו בקלות, ושימוש לחיזויים.

<u>חיזוי:</u>

בתוך מודול model.py הגדרנו גם פונקציית predict המחפשת state_dict בנתיב המועבר לה. הפונקציה צריכה גם למצוא את אותו מבנה רשת עליו התאמנה על מנת לטעון את המשקלים כראוי. הפונקציה מחזירה ציון accuracy על ה-test set.

הגדרנו לפונקציה זו 2 פונקציות מעטפת ב-main.py:

- הראשונה, main_predict תבצע חיזוי על כל ה-test ותחזיר ציון accuracy. לסיום, תדפיס מספר תמונות יחד עם ציון הדיוק שלהם מתוך batch רנדומי. עדיף לפני הפעלתה לכוון batch_size=4 על מנת שהתמונות יכנסו יפה במסך.
- השניה, view_mistakes מקבלת פרמטר k ותחזיר את k הדגימות שנחזו **הכי לא טוב**, בצירוף ציון החיזוי שלהן ובצירוף התיוג המקורי שלהן. פונקציה זו גם תדפיס היסטוגרמה המציגה את כלל הציונים אותם חזה המודל. בתקווה, היסטוגרמה זו תהיה מאוד מקוטבת, עם כחצי מהציונים נושקים ל-1 וחצי מהציונים נושקים ל-0, דבר שיעיד על ביטחון גבוה של הודל בחיזוי. בפועל קצת אבדה תקוותנו.

תוצאות:

כאמור, הבנו בתחילת הניסויים שמדובר במודל בעל נטייה משמעותית ל-overfit על המשימה, ככל הנראה לאור דגימות train מועטות ולא מספיק מגוונות. מהרגע שהבנו זאת עברנו להתמקד בשיטות שיעזרו להעניש את המודל או לגרום לו להכליל יותר טוב בשלב האימון. סביר להניח שאופטימיזציה יותר ארוכה על הפרמטרים האלו יכולה לשפר את המודל אפילו יותר. הניסויים שערכנו כוללים מספר בדיקות באמצעות הארכיטקטורה המקורית עבור תמונות בגודל 105X105, עם עונשים שונים ושימוש ב-batchnorm, המלווים בסדרת ניסויים על התמונות בגודלן המקורי (250X250) בתקווה שהשיפור באיכות (בליווי ארכיטקטורה מעט יותר מסובכת על מנת לכווץ התמונה באמצעות הקונבולוציה וללא transform) יעזור לשפר ביצועי המודל.

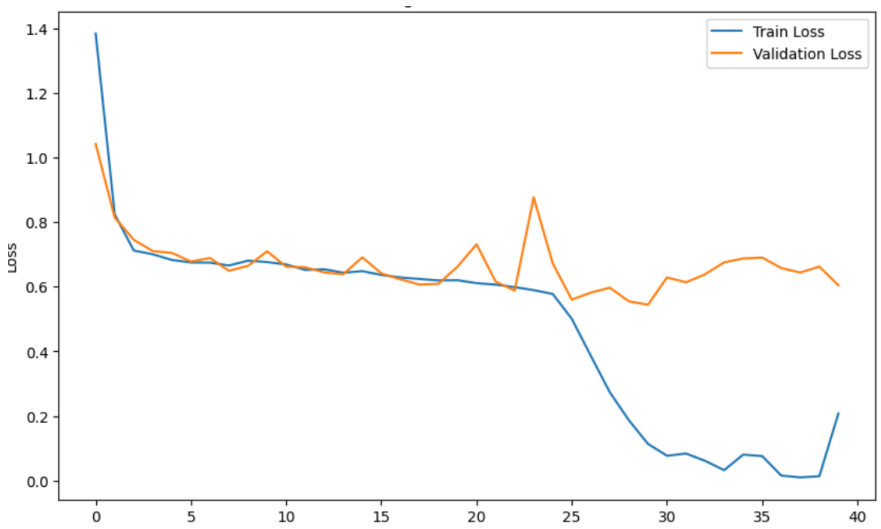
המודל הטוב ביותר שאומן הוגדר עם הפרמטרים הבאים:

```
# DataLoader Parameters
BATCH_SIZE = 16       # Batch size for training and validation
NUM_WORKERS = 2      # Number of worker threads for DataLoader

# Image Transformation Parameters
IMAGE_SIZE = (250, 250)  # Adaptation to paper's image size

# Model Parameters:
DROPOUT = 0.0
BATCHNORM = True
# Training Parameters
VAL_SPLIT = 0.2                    # Fraction of data for validation
AUGMENT_RATIO = 3                  # Augment the data X3 times
MAX_EPOCHS = 200                   # Max number of training epochs
LEARNING_RATE = 1e-3               # Learning rate for the optimizer
L2_REG = 1e-3                      # L2 regularization strength
EARLY_STOP_PATIENCE = 10           # Number of epochs for early stopping
```

עבור הארכיטקטורה הבאה (הוספת שכבת קונבולוציה וגם שינוי בחלוקת הפילטרים וגודל הארכיטקטורה):

```
CNN_BLOCKS = [
    {"out_channels": 64, "kernel_size": 10, "stride": 1, "padding": 0, "use_pooling": True,  "use_batchnorm": BATCHNORM, "dropout_prob":
DROPOUT},  # 250x250 -> 120x120
    {"out_channels": 128, "kernel_size": 7, "stride": 1, "padding": 0, "use_pooling": True,  "use_batchnorm": BATCHNORM, "dropout_prob":
DROPOUT},  # 120x120 -> 57x57
    {"out_channels": 256, "kernel_size": 5, "stride": 1, "padding": 0, "use_pooling": True,  "use_batchnorm": BATCHNORM, "dropout_prob":
DROPOUT},  # 57x57 -> 26x26
    {"out_channels": 256, "kernel_size": 4, "stride": 1, "padding": 0, "use_pooling": True,  "use_batchnorm": BATCHNORM, "dropout_prob":
DROPOUT},  # 26x26 -> 11x11
    {"out_channels": 512, "kernel_size": 3, "stride": 1, "padding": 0, "use_pooling": False, "use_batchnorm": BATCHNORM, "dropout_prob":
DROPOUT},  # 11x11 -> 9x9
]

FC_LAYERS = [
    {"in_features": 9*9*512, "out_features": 4096, "use_batchnorm": False, "dropout_prob": DROPOUT}, # Fully Connected Layer, # 9x9x512 ->
1X4096
    {"in_features": 4096, "out_features": 1, "use_batchnorm": False, "dropout_prob": 0.0},  # Final similarity score layer (1X4096 -> 1X1)
]
```

מודל זה הגיע לדיוק של 73.9% על ה-test set.



ניתן לראות שהמודל מראה סימנים ברורים של overfit וחוסר יציבות בירידה של ה-loss על ה-validation, למרות הרגולריזציה המחמירה (שניתן לראות שעזרה למתן באיטרציות הראשונות), ובאופן כללי קשה לו להכליל על דגימות חדשות, אולי לאור המורכבות היחסית של הבעיה. נציין שהמאמר המקורי הפעיל את הארכיטקטורה על בעיה פחות מורכבת (omniglot dataset) וכנראה שעבורה ארכיטקטורה זו הייתה מספיק טובה.

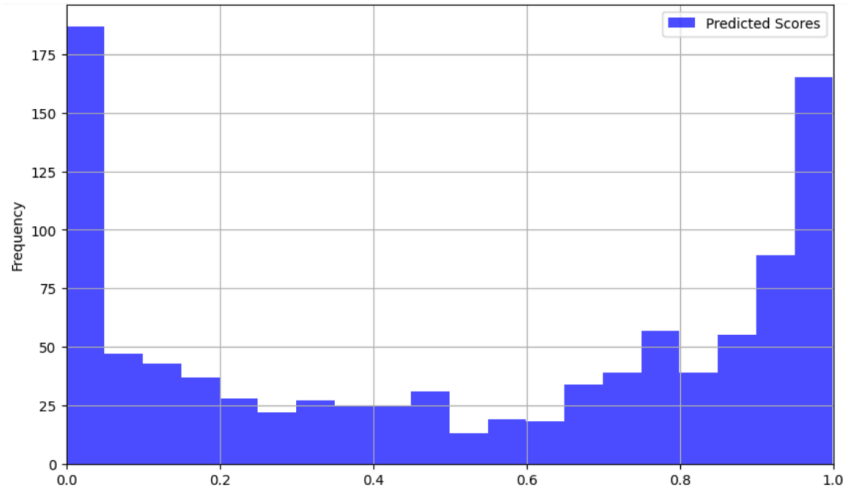
מצורף log האימון עד לנקודה האופטימאלית לעצירה (עצירה מוחלטת כ-10 איטרציות לאחר מכן):

```
[Training Status]: Epoch 1, Train Loss: 1.3834 (acc 0.579), Val Loss: 1.0417 (acc 0.548), Time: 2.4 min, LR: 0.001000
[Training Status]: Epoch 2, Train Loss: 0.8239 (acc 0.574), Val Loss: 0.8143 (acc 0.516), Time: 4.7 min, LR: 0.001000
[Training Status]: Epoch 3, Train Loss: 0.7120 (acc 0.592), Val Loss: 0.7452 (acc 0.566), Time: 7.1 min, LR: 0.001000
[Training Status]: Epoch 4, Train Loss: 0.7004 (acc 0.594), Val Loss: 0.7102 (acc 0.609), Time: 9.4 min, LR: 0.001000
[Training Status]: Epoch 5, Train Loss: 0.6828 (acc 0.599), Val Loss: 0.7046 (acc 0.586), Time: 11.8 min, LR: 0.001000
[Training Status]: Epoch 6, Train Loss: 0.6750 (acc 0.616), Val Loss: 0.6779 (acc 0.623), Time: 14.2 min, LR: 0.001000
[Training Status]: Epoch 7, Train Loss: 0.6746 (acc 0.602), Val Loss: 0.6888 (acc 0.577), Time: 16.5 min, LR: 0.001000
[Training Status]: Epoch 8, Train Loss: 0.6657 (acc 0.612), Val Loss: 0.6493 (acc 0.632), Time: 18.8 min, LR: 0.001000
[Training Status]: Epoch 9, Train Loss: 0.6806 (acc 0.612), Val Loss: 0.6647 (acc 0.627), Time: 21.2 min, LR: 0.001000
[Training Status]: Epoch 10, Train Loss: 0.6763 (acc 0.617), Val Loss: 0.7095 (acc 0.550), Time: 23.4 min, LR: 0.001000
[Training Status]: Epoch 11, Train Loss: 0.6690 (acc 0.604), Val Loss: 0.6623 (acc 0.630), Time: 25.7 min, LR: 0.001000
[Training Status]: Epoch 12, Train Loss: 0.6521 (acc 0.632), Val Loss: 0.6602 (acc 0.618), Time: 28.0 min, LR: 0.001000
[Training Status]: Epoch 13, Train Loss: 0.6540 (acc 0.625), Val Loss: 0.6444 (acc 0.641), Time: 30.2 min, LR: 0.001000
[Training Status]: Epoch 14, Train Loss: 0.6434 (acc 0.637), Val Loss: 0.6385 (acc 0.636), Time: 32.7 min, LR: 0.001000
[Training Status]: Epoch 15, Train Loss: 0.6482 (acc 0.628), Val Loss: 0.6907 (acc 0.586), Time: 35.2 min, LR: 0.001000
[Training Status]: Epoch 16, Train Loss: 0.6365 (acc 0.646), Val Loss: 0.6414 (acc 0.627), Time: 37.5 min, LR: 0.001000
[Training Status]: Epoch 17, Train Loss: 0.6287 (acc 0.650), Val Loss: 0.6229 (acc 0.643), Time: 39.7 min, LR: 0.001000
[Training Status]: Epoch 18, Train Loss: 0.6240 (acc 0.657), Val Loss: 0.6067 (acc 0.677), Time: 42.1 min, LR: 0.001000
[Training Status]: Epoch 19, Train Loss: 0.6195 (acc 0.656), Val Loss: 0.6091 (acc 0.689), Time: 44.4 min, LR: 0.001000
[Training Status]: Epoch 20, Train Loss: 0.6201 (acc 0.658), Val Loss: 0.6621 (acc 0.634), Time: 46.7 min, LR: 0.001000
[Training Status]: Epoch 21, Train Loss: 0.6111 (acc 0.670), Val Loss: 0.7312 (acc 0.577), Time: 49.0 min, LR: 0.001000
[Training Status]: Epoch 22, Train Loss: 0.6066 (acc 0.674), Val Loss: 0.6161 (acc 0.664), Time: 51.3 min, LR: 0.001000
[Training Status]: Epoch 23, Train Loss: 0.5987 (acc 0.680), Val Loss: 0.5881 (acc 0.705), Time: 53.6 min, LR: 0.001000
[Training Status]: Epoch 24, Train Loss: 0.5895 (acc 0.690), Val Loss: 0.8764 (acc 0.520), Time: 56.0 min, LR: 0.001000
[Training Status]: Epoch 25, Train Loss: 0.5776 (acc 0.702), Val Loss: 0.6722 (acc 0.643), Time: 58.2 min, LR: 0.001000
[Training Status]: Epoch 26, Train Loss: 0.5009 (acc 0.757), Val Loss: 0.5601 (acc 0.723), Time: 60.5 min, LR: 0.001000
[Training Status]: Epoch 27, Train Loss: 0.3862 (acc 0.834), Val Loss: 0.5817 (acc 0.745), Time: 62.9 min, LR: 0.001000
[Training Status]: Epoch 28, Train Loss: 0.2732 (acc 0.898), Val Loss: 0.5969 (acc 0.714), Time: 65.1 min, LR: 0.001000
[Training Status]: Epoch 29, Train Loss: 0.1868 (acc 0.939), Val Loss: 0.5549 (acc 0.745), Time: 67.4 min, LR: 0.001000
[Training Status]: Epoch 30, Train Loss: 0.1138 (acc 0.970), Val Loss: 0.5442 (acc 0.757), Time: 69.7 min, LR: 0.001000
[Training Status]: Epoch 31, Train Loss: 0.0770 (acc 0.985), Val Loss: 0.6285 (acc 0.741), Time: 72.0 min, LR: 0.001000
[Training Status]: Epoch 32, Train Loss: 0.0838 (acc 0.980), Val Loss: 0.6137 (acc 0.730), Time: 74.2 min, LR: 0.001000
[Training Status]: Epoch 33, Train Loss: 0.0613 (acc 0.986), Val Loss: 0.6384 (acc 0.743), Time: 76.3 min, LR: 0.001000
[Training Status]: Epoch 34, Train Loss: 0.0323 (acc 0.996), Val Loss: 0.6752 (acc 0.741), Time: 78.5 min, LR: 0.001000
[Training Status]: Epoch 35, Train Loss: 0.0805 (acc 0.976), Val Loss: 0.6874 (acc 0.709), Time: 80.7 min, LR: 0.001000
[Training Status]: Epoch 36, Train Loss: 0.0760 (acc 0.979), Val Loss: 0.6901 (acc 0.734), Time: 82.9 min, LR: 0.001000
[Training Status]: Epoch 37, Train Loss: 0.0156 (acc 0.999), Val Loss: 0.6576 (acc 0.745), Time: 85.0 min, LR: 0.001000
[Training Status]: Epoch 38, Train Loss: 0.0101 (acc 1.000), Val Loss: 0.6440 (acc 0.761), Time: 87.2 min, LR: 0.001000
[Training Status]: Epoch 39, Train Loss: 0.0137 (acc 1.000), Val Loss: 0.6622 (acc 0.743), Time: 89.3 min, LR: 0.001000
[Training Status]: Epoch 40, Train Loss: 0.2074 (acc 0.914), Val Loss: 0.6046 (acc 0.730), Time: 91.5 min, LR: 0.001000
```

הירידה ב-loss מצביעה לנו על דיוק + מידת הביטחון של המודל החיזויים. ייתכן מודל בעל accuracy גבוה יותר אינו בעל ה-loss המינימלי, אך בחרנו לקחת את ה-checkpoint בהתאם ל-loss הנמוך ביותר. יותר משאבי GPU כמובן היו עוזרים לנו לבחון יותר מצבים, קונפיגורציות וביצועים.

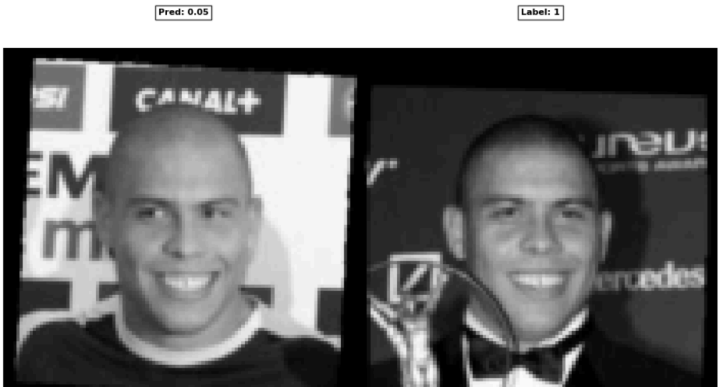
המודל מספק תחזיות בסגנון הבא (Same / Different זה הסיווג האמיתי, והחיזוי מהווה את מידת הביטחון של המודל שהתמונות **שייכות לאותו האדם**, ערך בין 0 ל1):

ניתן לראות שאפילו שהצליח בדוגמאות הרנדומליות האלו, מידת הביטחון שלו בחיזוי הייתה נמוכה (ערכים הקרובים יחסית ל-0.5). רצינו לבדוק את מידת הביטחון של המודל בחיזויים שלו ולכן הפקנו ההיסטוגרמה הבאה:

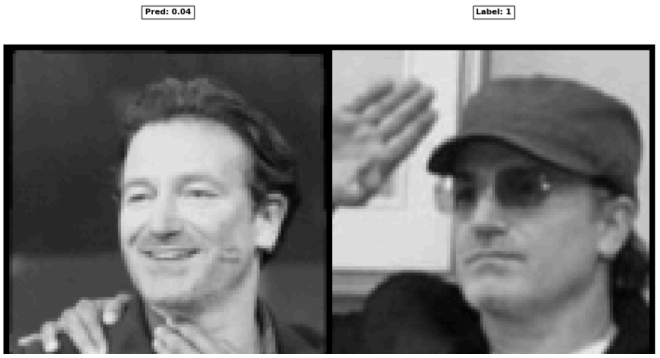


במצב אופטימלי, היינו רוצים לראות שרוב החיזויים יהיו בקרבת 1 או בקרבת 0 (באופן שווה לאור החלוקה השווה). בפועל אנחנו רואים פיזור שנראה יחסית טוב, אבל עם עדיין יחסית הרבה משקל באמצע הסקאלה. אפשר להסיק גם שהמודל מעט יותר בטוח בחיזויים שליליים מאשר חיזויים חיוביים.

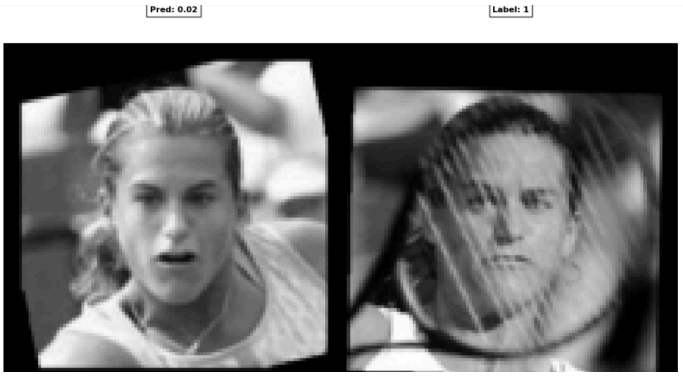
אספנו חלק מהחיזויים הכי פחות טובים בניסיון להתחקות אחר הקשיים:



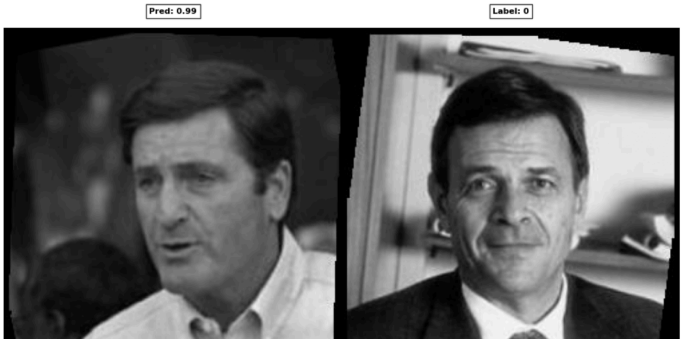
בניסיון זה אפשר לראות שה-contrast הפוך לגמרי, כאשר הרקע בהיר באחת וכהה בשניה, דבר שמשפיע גם על כהות השיער של הדמות, לצד אובייקט המסתיר את פניה באחת התמונות.



פספוס דומה קורה פה לדעתנו, כאשר הרקע כהה משמעותית בתמונה אחת מול השניה, דבר המשפיע על פיזור הצבעים הכללי. זה, לצד הבעות פנים שונות ולצד כובע באחת התמונות כנראה לא עוזרים.



גם פה ההשערה שלנו היא שהמחבט על פניה של הדמות, לצד crop של התמונה ושוני בהבעות הפנים הקשו מאוד.
אפשר לראות שבכל החיזויים האלו המודל היה מאוד מאוד בטוח בתשובה שלו, רק שזו הייתה שגויה. כנראה ששינויי ניגודיות בסביבה או עצמים מסתירים / מטשטשים הם החולשה המרכזית שלו.



במקרה הזה, בכנות, אנחנו לא ממש מבדילים ביניהם אז סולחים למודל.
מנגד, חיזוי מוצלח של המודל שמצאנו משעשע:

לפחות פה הוא בטוח בעצמו. כן נציין שהבעות הפנים בשתי התמונות, צבע פריט הלבוש הבולט (הסרט) ותנועת הפה עוזרים להבין למה כנראה היה למודל יותר קל להגיע להחלטה דטרמיניסטית.

להלן טבלת הניסויים המלאה עבור הניסויים שנערכו בארכיטקטורה הסופית (עבור הארכיטקטורות השונות מהעיצוב המקורי גם שינינו את מספר הפילטרים וגודל ה-kernel. ציינו את פרטי הארכיטקטורה המלאים רק עבור האחת הטובה ביותר):

Attempt No.	Image Size	General Architecture	Batch Size	Augmentation	Validation size(of train)	BatchNorm	DropOut	Learning Rate	L2	Best Epoch	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy	Test Accuracy	Time (minutes)
1	(105X105)	Original Paper	16	X3	20%	FALSE	0	1.00E-03	1.00E-03	18	0.2682	0.5638	0.909	0.727	0.69	7.7
2	(105X105)	Original Paper	16	X3	20%	TRUE	0	1.00E-03	1.00E-03	20	0.5262	0.5785	0.761	0.689	0.71	26
3	(105X105)	Original Paper	16	X3	20%	FALSE	0.3	1.00E-03	1.00E-03	1	0.7176	0.7611	0.543	0.5	0.5	12.4
4	(105X105)	Original Paper	16	X3	20%	TRUE	0.3	1.00E-03	1.00E-03	3	0.7402	0.655	0.513	0.605	0.59	3.9
5	(105X105)	Original Paper	16	X9	30%	TRUE	0	1.00E-03	1.00E-03	18	0.0096	0.6509	1	0.726	0.7	19.1
6	(250X250)	5 CNN + 2 FC, stride=1	16	X3	20%	TRUE	0	1.00E-03	1.00E-03	24	0.5244	0.5559	0.745	0.714	0.721	36.5
7	**(250X250)**	**5 CNN + 2 FC, stride=1**	**16**	**X3**	**20%**	**TRUE**	**0**	**1.00E-03**	**1.00E-03**	**30**	**0.1138**	**0.5442**	**0.97**	**0.757**	**0.739**	**69.7**
8	(250X250)	4 CNN + 2 FC, stride=2	16	X3	20%	TRUE	0	1.00E-03	1.00E-03	27	0.6433	0.606	0.63	0.64	0.642	25.6
9	(250X250)	5 CNN + 2 FC, stride=1	16	X3	20%	TRUE	0	1.00E-03	1.00E-03	39	0.4763	0.5641	0.783	0.698	0.687	37.4

מספר תובנות:

- על אף שניסינו במספר קונפיגורציות שונות, השימוש ב-Dropout רק פגע משמעותית בתוצאות, ולכן נשארנו עם L2 לרגולריזציה, לצד LR גבוה יחסית ו-BATCH_SIZE קטן יחסית.
- ערכי ה-L2 וה-LR אליהם הגענו יחסית גבוהים והגענו אליהם אחרי נסיונות עם ערכים שונים יותר קטנים.
- האופטימייזר המוצג במאמר, SGD, פעל משמעותית פחות טוב מ-ADAM על ארכיטקטורות זהות בבעיה זו, ולכן בניסויים הסופיים לא עשינו בו שימוש.
- השימוש ב-BATCHNORM עזר בתהליך האימון ושיפר במקצת את התוצאות
- השימוש ב-Augmentation היה רעיון נחמד, שבחלק מהמקרים שיפר ביצועים, אך לא באופן משמעותי (וכמובן שהאריך את זמן הריצה משמעותית). ייתכן ששימוש יותר אגרסיבי בזה יחד עם עיוות יותר משמעותי של התמונות היה עוזר, אך בהינתן המשאבים הנוכחיים נאלצנו להישאר בערך נמוך שלו ביחס למאמר.
- כל אחד מהמודלים שבנינו היה בעל נטייה משמעותית ל-overfit. האתגר האמיתי פה היה להפעיל מספיק רגולריזציה על מנת שיצליח ללמוד מספיק טוב עד לנקודה הזו.
- שימוש בארכיטקטורה יותר גדולה על מנת לקלוט באופן תקין תמונות בגודל 250X250 עזרה קצת, ויצרה לנו את התוצאות הטובות ביותר, למרות שלא שיפרה באופן דרסטי את הביצועים על ה-test.