Assignment 4 - Report

Shahar Oded, 208388918
Nir Rahav, 316275437

הקמה של ה-TabularDataset:

אנחנו מתעסקים פה עם דאטה טבלאי, פשוט יחסית, עבורו אנחנו יודעים מראש מה הערכים הייחודיים בכל עמודה ומה ה-data type של כל עמודה. קצת מידע סיכומי:

```
[Info]: Number of rows: 32561
[Info]: Number of columns: 14

[Info]: Target class distribution:
income
b'<=50K'    0.75919
b'>50K'     0.24081

[Info]: Categorical columns: 8
[Info]: Continuous columns: 6
[Info]: Number of columns after one-hot encoding: 93 == Embedding Vector Dimension (with 1 label column)
```

ערכים ייחודיים שלא מופיעים בדאטה עצמו אינם מיוצגים בווקטור (נניח ערכים המופיעים ב-meta אך לא בפועל). מכיוון שאנו עוסקים בהתפלגות הנתונים בפועל, ומידע שלא יהיה קיים לא יג'ונרט ע"י G, זה לא מפריע לנו. כמו כן, הסרנו את עמודת education המופיעה פעם אחת כערך קטגוריאלי ופעם אחת כנומרי. דאגנו לסמן get_dummies(...,drop_first=False על מנת שגם עמודות קטגוריאליות בוליאניות יהפכו ל2 עמודות ולא ל-1. זה משפיע לאחר מכן על האופן בו נייצג מידע קטגוריאלי מג'ונרט (אנחנו רוצים שכל ערך True יהיה בעמודה משלו).

אנחנו רואים imbalance בעמודת ה-target אבל לא משהו מאוד חריג. נרצה לבצע stratified split כשנעסוק בחלקים מהדאטה הזה. כמו כן, הגדרנו אוגמנטציה באמצעות SMOTE כך שנייצר ייצוג שווה בדאטה לפי ערך עמודת ה-target, תוך גיוון בנתונים, מחשש ש-G יילמד יותר טוב את ייצוג label הרוב (ניסיון למנוע mode collapse בהמשך). בפועל, שיטה זו לא עזרה באימון והשארנו הדאטה בהתפלגותו הרגילה. כמו כן, הערכים הנומריים נורמלו באמצעות MinMaxScaler. את עמודת ה-target נציג בתור כניסה בודדת עם "1" עבור 'b'>50K' ו"0" עבור 'b'<=50K'.

יצרנו בנוסף **TabularDatasetFromArrays** משני שתפקידו לקחת אובייקט מהדאטה סט הראשי אחרי ביצוע Stratified Split ולהפוך חזרה לאובייקט אותו ניתן להפוך ל-dataloader. זו מחלקה לוגיסטית בלבד שנועדה לאפשר split על אובייקט dataset מבלי להשתמש באובייקט Subset שמגביל אותנו בהמשך.

היות ונשתמש בפונקציית TanH כדי ליצור הוקטור היוצא מה-generator באקטיבציה הסופית, העברנו את כל הנתונים להיות בטווח של [-1,1]. אחרי שימוש בפונקציית החלוקה אנחנו רואים שימור של היחסים בתוך כל label:

```
[Main]: Class distributions after split:
Train (26048 labels): 0 -> 0.759175
                      1  -> 0.240825

Test (6513 labels): 0  ->  0.759251
                    1  ->  0.240749
```

ואם אנחנו בוחרים להשתמש באוגמנטציה:

```
[Main]: Class distributions after split:
Train (39550 labels):  1  ->  0.5
                       0  -> 0.5
Test (6513 labels): 0  ->  0.759251
                    1  ->  0.240749
```

הוספנו האפשרות לחלוקה גם ל-val_set מתוך ה-train (לפני אוגמנטציה). בפועל, נדרשנו בזה רק כאשר אימנו AE. על מנת לטפל בבעיה עתידית, בה המודל לא יוכל לתת ערכים קטגוריאליים (אלא רק רציפים), שמרנו כ-attribute גם מיפוי של כל האינדקסים של עמודות one-hot לעמודות הרלוונטיות שלהם, כך שפונקציית generate תוכל לעשות להם עיגול לערך 1 או -1 בהתאמה, עבור הערך המקסימלי בכל קבוצת עמודות (עמודה קטגוריאלית לשעבר). המיפוי מועבר למודל באמצעות ה-dataloader, שהוגדר במחלקה משלו כדי להחזיק את ה-attribute הזה (CustomDataLoader). כך נוכל לג'ונרט רשומות אמינות יותר, לפחות בהיבט הזה. על מנת שיעבוד, מיפינו ערכים חסרים ('?') לערך 'Unknown', ואלו מהווים עוד אפשרות לערך בקטגוריות הרלוונטיות, שהמודל לומד לג'ונרט.

הגדרת מודל GAN:

בדומה לעבודות קודמות, רצינו להשאיר את הקונפיגורציה של השכבות בתור פרמטר חיצוני וברור המגדיר לכל שכבה גדלים, אקטיבציות, batch norm ו-dropout. ניתן לראות את הקונפיגורציות בהן בחרנו ב-config.py:

```
# Model Config
DATA_DIM = 92 # The size of the feature vector
NOISE_DIM = 32  # The size of the initial noise vector

## Generator Configuration
GENERATOR_CONFIG = [
    {"input_dim": NOISE_DIM, "output_dim": 128, "activation": nn.LeakyReLU(0.2), "norm": 'batch', "dropout": 0.1},
    {"input_dim": 128, "output_dim": 256, "activation": nn.LeakyReLU(0.2), "norm": 'batch', "dropout": 0.1},
    {"input_dim": 256, "output_dim": 512, "activation": nn.LeakyReLU(0.2), "norm": 'batch', "dropout": 0.1},
    {"input_dim": 512, "output_dim": 1024, "activation": nn.LeakyReLU(0.2), "norm": 'batch', "dropout": 0.1},
    {"input_dim": 1024, "output_dim": DATA_DIM, "activation": nn.Tanh(), "dropout": 0.0},
]

DISCRIMINATOR_CONFIG = [
    {"input_dim": DATA_DIM, "output_dim": 128, "activation": nn.LeakyReLU(0.2), "norm": 'layer', "dropout": 0.2},
    {"input_dim": 128, "output_dim": 32, "activation": nn.LeakyReLU(0.2), "norm": 'layer', "dropout": 0.2},
    {"input_dim": 32, "output_dim": 1, "activation": nn.Sigmoid(), "dropout": 0.0},
]
```

בדומה לתרגול, בחרנו להשתמש בפונקציות אקטיבציה LeakyRelu, כך שה-output של ה-generator יהיה באמצעות TanH, ועבור ה-discriminator בחרנו להשתמש ב-output של sigmoid הרי שהבעיה היא בינארית.

רצינו ליצור ארכיטקטורה מורכבת יותר עבור ה-generator על מנת שיוכל להיות חזק מספיק מול ה-descriminator שעבורו הארכיטקטורה פשוטה יותר ואמורה לשרת משימת קלאסיפיקציה יחסית "קלה". הארכיטקטורה המצורפת היא זו שהתכנסה הכי טוב.

המודל מקבל וקטור רעש בגודל NOISE_DIM בגודל 32 וממנו מג'נרט את הדגימה. בחרנו ברעש הקטן מ-DATA_DIM לאור המלצת קלוד, על מנת שזה יתרחב בהדרגה לאורך שכבות המודל.

הפרמטרים של המודל מוגדרים גם הם חיצונית:

```
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
BASE_LEARNING_RATE = 2e-4
WEIGHT_DECAY = 5e-4
LAMBDA_CORR = 0.1   # Level of influence of the correlation loss on global loss
GAN_EARLY_STOP = 10     # Stop after |EARLY_STOP| epochs with no improvement in the total loss
WARMUP_EPOCHS = 10  # Define a number of GAN warmup iterations in which the model won't count towards an early stop.
EPOCHS = 30    #   A high number of epochs, hoping for an early stopping
GENERATOR_UPDATE_FREQ = 3   # Number of G updates per D updates, to balance their losses.
```

המודל מאמן את שני חלקיו יחד בתוך פונקציית train, כאשר הגדרנו פרמטר GENERATOR_UPDATE_FREQ המאפשר לאמן את הגנרטור יותר מאשר את הדיסקרימינטור במקרה של שיפור מואץ שלו על פניו. בארכיטקטורה הסופית בחרנו להגדיר פרמטר זה על 3, היות וזה סיפוק לנו שיווי משקל טוב באימון.

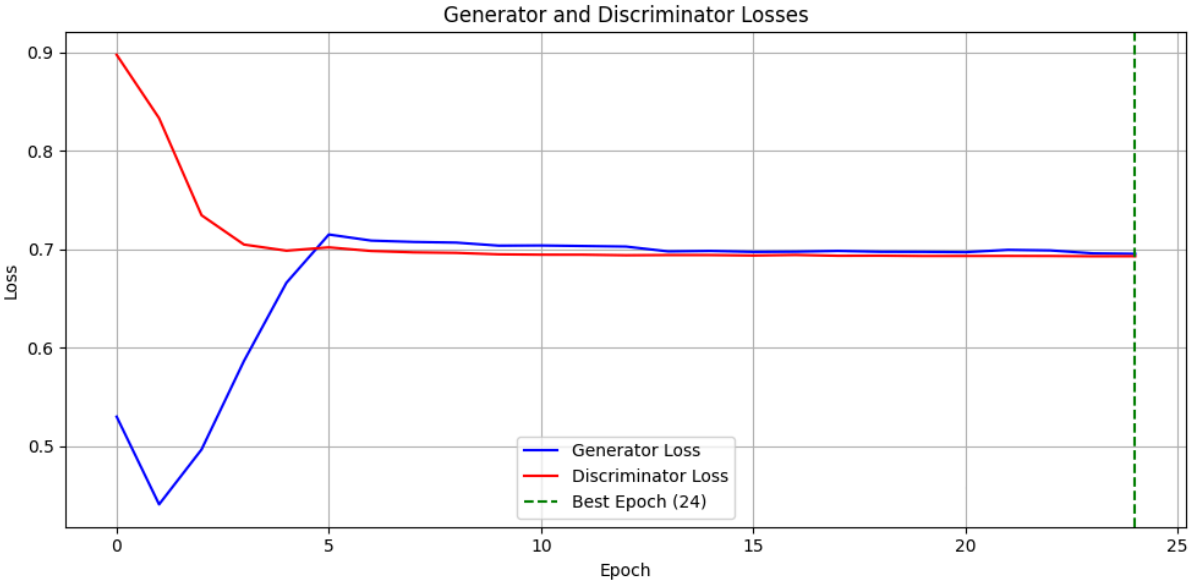
בנוסף, לאחר הרבה התנסויות עם המודלים ואחרי שראינו שקורלציה בין הפיצ'רים היא בעיה חוזרת, הוספנו loss_term נוסף באימון הבודק גם את הפרש הקורלציות הממוצע בין המידע האמיתי למידע המג'ונרט. את הערך הזה הכפלנו ב-LAMBDA_CORR וזה penalty הנוסף בכל פעפוע loss ב-generator, ומעניש אותו על רשומות פחות אמינות.

הגדרנו בתהליך האימון לשמור בכל נקודת שיפור את המשקלים של המודל הטוב ביותר על מנת שנוכל לשחזר המודלים מבלי לאמן אותם מחדש. המודל הטוב ביותר נקבע ע"י ערכי ה-loss של ה-Generator, היות והמטרה היא לייצר דגימות מזויפות משכנעות, ולכן נקבע שיפור ע"י ירידה ב-loss שלו. עם זאת, בתהליך האימון נרצה להיות ערניים לירידת ה-loss של ה-discriminator כדי לוודא שהוא בעצמו מצליח ללמוד להבחין בין הווקטורים, ונשאר יחסית מאוזן, ולכן לפני שהוא מאומן דיו, לא לקחנו את המודל הטוב ביותר כ-checkpoint (לפחות עד סוף ה-warm_up). הגדרנו גם עצירה מוקדמת אם אין שיפור בערך זה אחרי 10 אפוקים, כך שבפועל כל מודל יתאמן לפחות 20 epochs, אבל יתכן וימצא את האופטימום שלו אחרי האיטרציה ה-10, או מתישהו בהמשך. עצירה כפויה אחרי 30 אפוקים.

המודל עובד עם אופטימייזר ADAM עבור שני המודלים, עם LR ו-W2 כמופיע בתמונה ועם פונקציית loss מסוג BCELoss לאור האופי הבינארי של הבעיה. בניסיון לאזן עוד יותר בין המודלים הגדרנו W2 רק עבור הדיסקרימינטור, עם LR הקטן פי 4 מזה של הג'נרטור, מה שעזר לנו לייצב את תהליך האימון. מספר האפוקים הסופי של כל מודל יופיע בתיעוד האימון שלו (גרף). מימשנו גם scheduler עבור ה-LR בתהליך האימון, מסוג OneCycleLR, לאור המלצה של קלוד במודלים כאלו, היות ומודל זה מצמצם את ה-LR לאחר warmup ולכן רגיש לצורת האימון פה. לסיום, כדי לוודא שלא יתפוצצו לנו הגרדיאנטים בגלל השאיפה לערכים קטגוריאליים הגדרנו gradient clipping ל-G וגם ל-D, וגם label smoothing שמשאיף את הלייבלים ל-0.1 ול-0.9 במקום ל1,0, מה שגורם ל-D להיות פחות בטוח בעצמו, ולהתכנסות יציבה יותר.

תהליך האימון:

אפשר לראות שבהתחלה קל מאוד ל-generator להטל ב-discriminator ולכן ערכי ה-loss שלו טובים יותר. לאחר מספר אפוקים ה-discriminator מבין את בעיית הסיווג ומתחיל להשתפר בה. אנחנו רצינו לאפשר למודל לעצור בנקודת מינימום לוקאלית של ה-generator אחרי ששני המודלים כבר למדו להגיב אחד לשני ושה-loss יחסית יציב. למרות ההתחלה היחסית קשה, נראה כאילו אחרי כ-5 אפוקים ה-loss ההדדי שלהם נכנס למגמת התייצבות, סימן חיובי מבחינתנו.



המודל הטוב ביותר היה לאחר 24 אפוקים.

שני המודלים, אחרי שהתבדרו במשך מספר אפוקים, התייצבו על loss באזור של log2 - סימן טוב היות והמשמעות שלו היא שה-discriminator לא מצליח לקבל החלטה טובה על הדגימה האמיתית והמזויפת, אחרי שהשתפר במשימה ביחס לתחילת האימון. ייתכן שזה אומר גם ש-D חלש מידי ולכן גם G לא משתפר, אבל ברמה היוריסטית, היות ובאחד הניסיונות שלנו כשמשכנו את זה יותר אפוקים קדימה הדיסקרימינטור overpowered את הג'נרטור, אנחנו רואים במגמה הזו סימן טוב, ומעדיפים שה-loss של שניהם יהיה קרוב.

הגדרת מודל cGAN:

על מנת לעשות הסבה של המודל הקודם הגדרנו מחלקה יורשת בה יש פרמטר נוסף, NUM_CLASSES. האופי המודולרי של הקוד אפשר לנו באמצעות שינויים מינימליים (למשל הגדלת ה-input dimention של השכבה הראשונה) להשתמש באותו המבנה. בפונקציית ה-train היינו צריכים לדאוג להעביר לשני חלקי המודל לצד הוקטור z את ה-label האמיתי של הדגימה, זוהי למעשה instruction בגודל 2 שמקונקטת לוקטור,המועברת ע"י ה-dataloader (ומקודם פשוט לא נעשה בה שימוש).

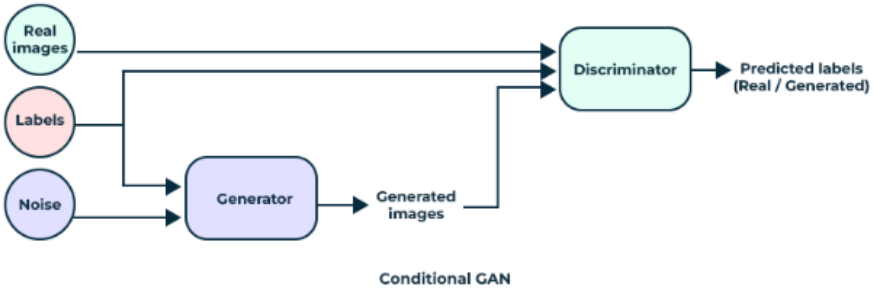
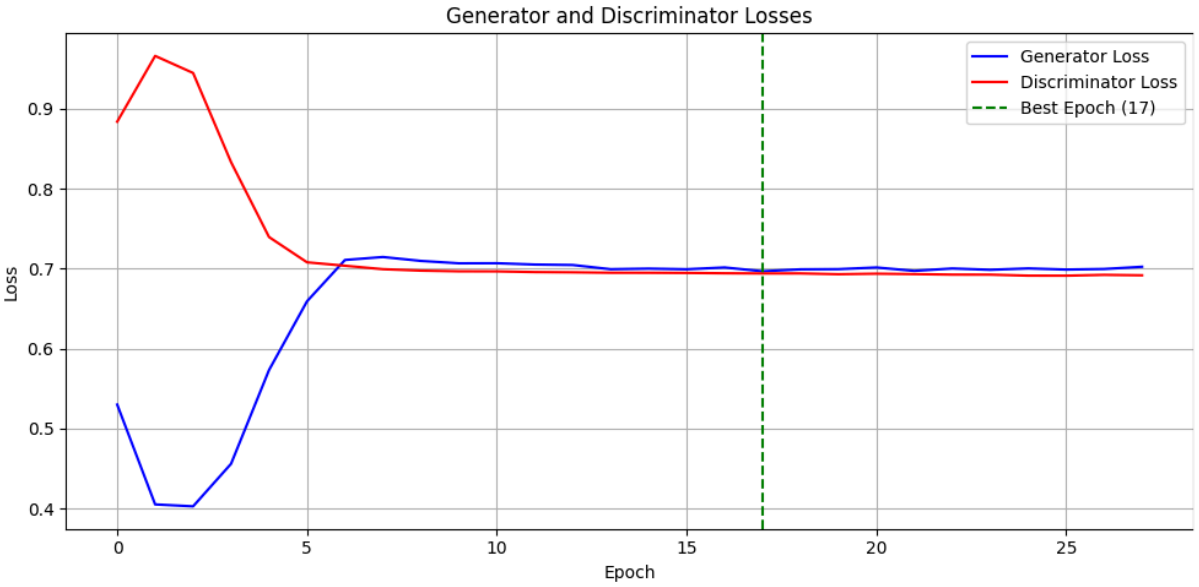


Figure 1: the conditional GAN architecture

בניגוד למודל הקודם, שאם היה מג'נרט דגימות לפי מחלקה 1 או 0 זה היה כתלות בלבד בהתפלגות הפיצ'רים שלהם, המודל הזה מנסה ללמוד את ההתפלגות של כל מחלקה בנפרד, ואז מייצר דגימות יותר מפוקסות למחלקה זו, לפי דרישה.
המודל אומן על אותם הפרמטרים מבחינת מבנה והלך אימון כמו מודל ה-GAN על מנת לייצר השוואה אמיתית ביניהם.

תהליך האימון:

גם המודל הזה הגיע לשיווי משקל לאחר כ-5 איטרציות, תוך שה-loss יורד בהדרגתיות עד לאיטרציה הטובה ביותר לאחר 17 אפוקים. בסך הכל, מבחינת תוצאות האימון, אין הבדל מהותי בינו לבין מודל ה-GAN הרגיל.

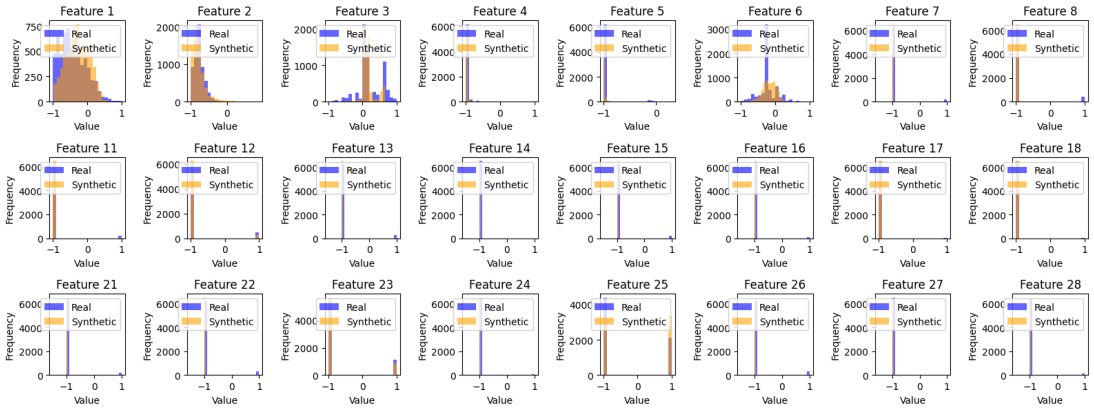


את המימוש המלא לשני המודלים ניתן למצוא במודול gan.py.

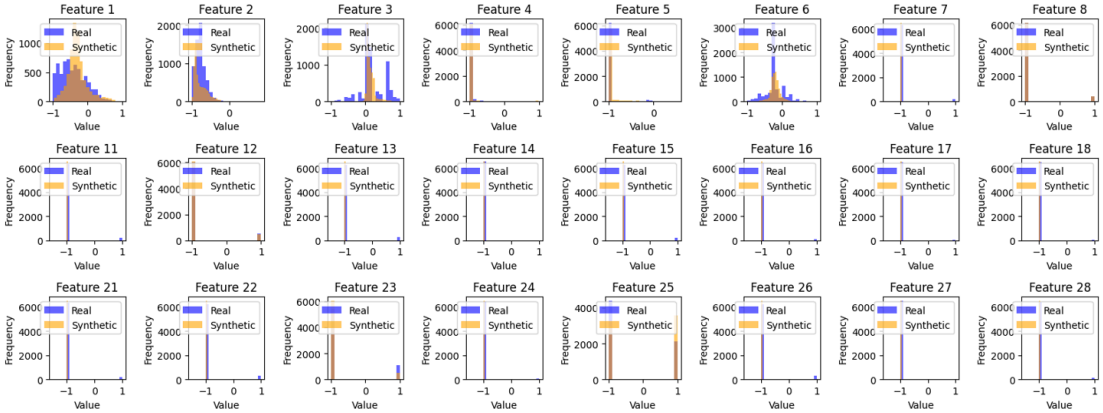
ביצוע Feature Analysis:

בשלב ראשון רצינו לבחון את התפלגויות הנתונים הנוצרים מה-GAN ומה-cGAN. בתמונות 6 הפיצ'רים הראשונים הם נומריים והיתר קטגוריאליים (מוצגים 28 מתוך 92 הפיצ'רים, מטעמי מקום וחוסר גיוון). אפשר לראות דוגמה לפילוח מלא במחברת המצורפת להגשה.

GAN (top few):



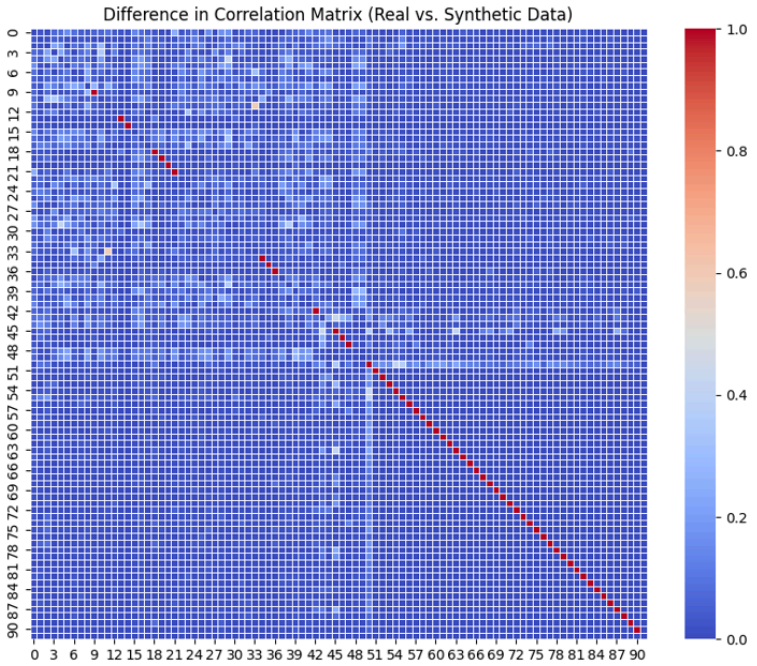
cGAN (top few):



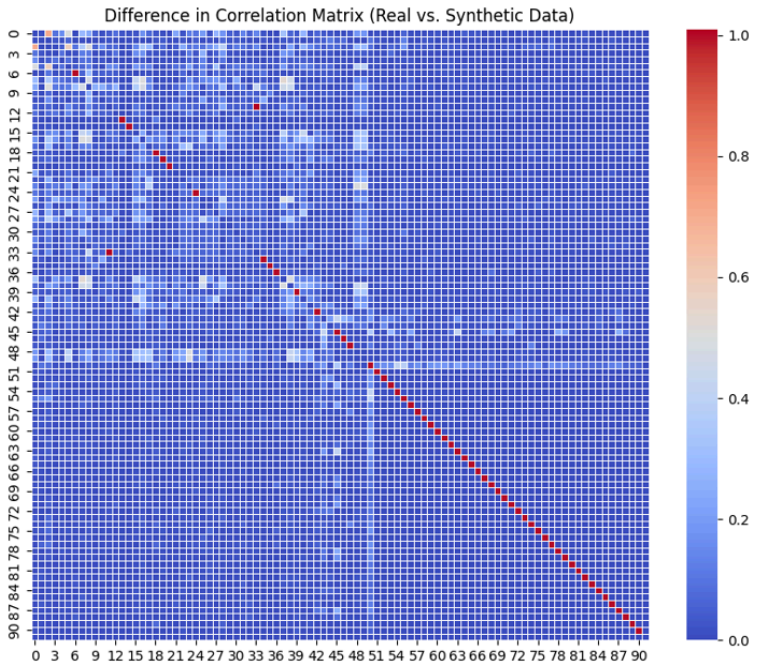
בבדיקה זו, לדעתנו, מודל ה-GAN מציג התפלגויות מעט יותר קרובות למציאות, עם פחות מרכוז סביב ערכים ספציפיים. ז"א, אנחנו רוצים לראות חפיפה כמה שיותר משמעותית בין ההיסטוגרמה הצהובה לבין הכחולה, כך שאם קיימת התפלגות מפוזרת יחסית בנתונים האמיתיים, היינו רוצים שגם מודל ה-GAN יידע לחקות אותה.
התרשימים מראים לנו שהמודלים מצליחים בסך הכל לייצר גם את הערכים הנומריים וגם הערכים הקטגוריאליים, בהתפלגויות יחסית דומות למקור. אם למשל היינו רואים "פעמון" באיזור שונה בסקאלה מזה בו הנתונים האמיתיים נמצאים - היינו דואגים. זה לא ממש המצב, למרות שכן ניתן לראות סטייה מסוימת עבור feature 1, למשל, בעיקר במודל ה-cGAN. בדיקה זו היא ראשונית ונחמדה, אבל בשלבים הבאים נרצה לבחון האם הנתונים אמינים. רשומה אמינה היא רשומה בה נשמרים היחסים הפנימיים שהיינו מצפים ביחס לדאטה המקורי, ולכן הצגת ההתפלגות של הפיצ'רים היא לא מספיקה (למרות שעוזרת לראות האם יש פיצ'רים שפשוט לא נלמדו טוב ולא מקבלים ייצוג בדאטה הסינטטי).

לשם כך בשלב השני נבדוק את הקורלציות בין הפיצ'רים בתוך הדגימות עצמן, והאם דומות לקורלציות בין הפיצ'רים בנתונים האמיתיים. על מנת לפרשן יותר בקלות, יצרנו מטריצת קורלציות המראה את ההפרשים בקורלציות בתוך הנתונים המזויפים למול בתוך הנתונים האמיתיים.
בפשטות יחסית, תמונה בה הצבע הכחול הכהה שולט יותר מציגה לנו למעשה שההבדלים בקורלציות בין הפיצ'רים הנוצרים מהמודל יותר קטנים, ושהמודל מצליח לייצג טוב יחסית את היחסים הפנימיים בין הפיצ'רים בתוך הדגימות. ערכים אדומים מוחלטים - כנראה ערכים שהמודל לא הצליח לייצר כלל, תוצאה שלילית מבחינתנו. לדעתנו מודל ה-GAN עשה זאת מעט יותר טוב:

GAN:



cGAN:



לסיכום של בדיקה זו - הנתונים המיוצרים חד משמעית אינם מושלמים, ויש איזורי חולשה משמעותיים בהם נראה שגם אם התפלגויות הנתונים יחסית נשמרות, הקורלציות בין הנתונים לא. סביר להניח שכשננסה להשתמש בנתונים אלו לאימון מודל, הוא יוכל להבחין שאינם קוהרנטיים עם עצמם, ושהקורלציות לא נשמרות, ולהצביע על זה באמצעות עץ החלטה.

מהאיור נראה שמודל ה-cGAN יצר קורלציות פחות משכנעות (יותר "מוכתמות" משמע פחות מקורלצות ביחס לדאטה האמיתי). כמו כן אנחנו רואים חולשה יותר משמעותית ברבע העליון השמאלי של המטריצה. כנראה שהנתונים שם יותר קשים לג'נרוט אמין, והמודלים שניהם מתקשים לעשות זאת. קיימת הבעיה של הערכים הקטגוריאליים, בהם נראה ששני המודלים מתקשים (ריבוי ערכים אדומים מובהקים). ייתכן שהפתרון שלנו של עיגול הערכים עבור עמודות מסומנות מראש הוא אגרסיבי מידי, ומייצר "הרעבה" יחסית של ערכי פיצ'רים שנמצאים בייצוג נמוך יותר בנתונים האמיתיים.

ביצוע הערכה לדאטה הסינטטי באמצעות detection & efficiency:

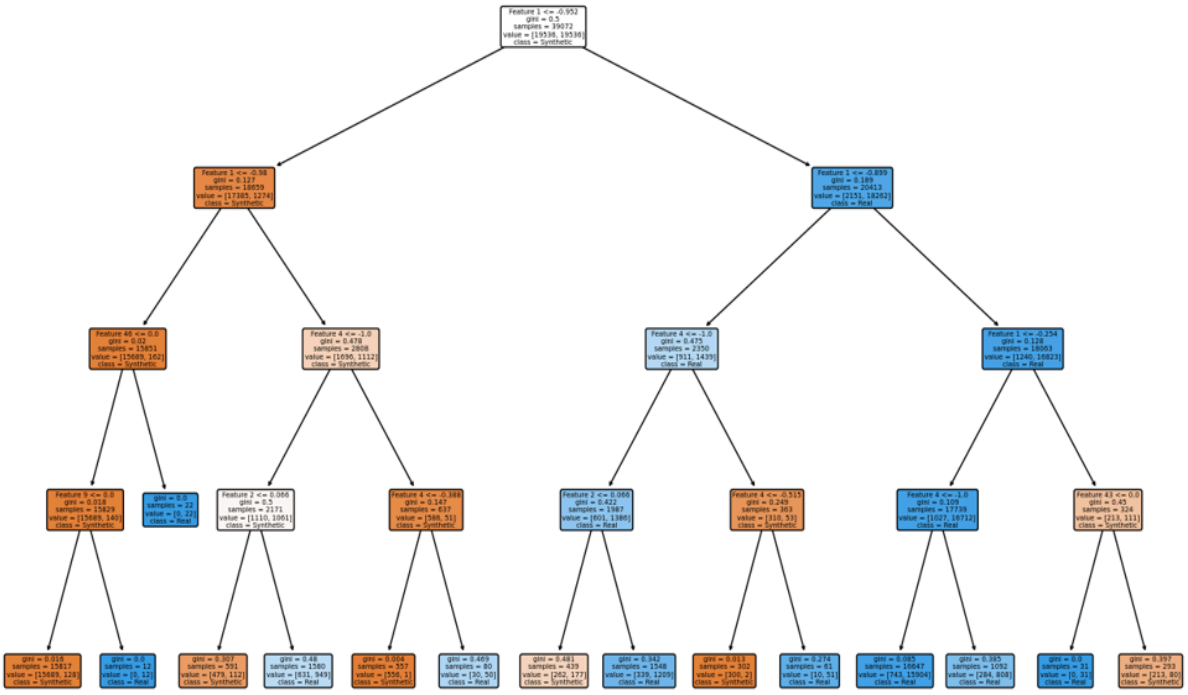
התוצאות בחלק זה מתייחסות למודלי GAN ו-cGAN הרגילים. הניסויים בוצעו על חלוקות train-test באמצעות 3 ערכי SEED שונים (42, 76, 2005) והתוצאות המופיעות פה הן ממוצע של ציוני הביצוע שהתקבלו ע"י 2 המודלים. בשלב הראשון, בהתאם לבקשה, ניקח את ה-train_set ו generated set באותו הגודל (מושפעים מערך ה-SEED) ובאמצעות CV 4 folds נאמן מודל RF על 75%, כאשר עבור ה-25% הנוספים נבדוק האם המודל מצליח להבחין בין הדגימות האמיתיות והמזויפות תוך שימוש במדד AUC. כל הדגימות האמיתיות יסומנו ב-1 וכל המזויפות ב-0. עבור מודל ה-cGAN ה-generated set מכיל דגימות ב-ratio זהה לדגימות המקוריות, היות ומשתמש בלייבלים שלהן.

עבור מודל ה-GAN ביצועי ה-detection בממוצע היו: 0.997 (תוצאות של 0.998, 0.997, 0.997).

עבור מודל ה-cGAN ביצועי ה-detection בממוצע היו: 1.000 (כל החלוקות החזירו ערך של 1.000)

הביצועים האלו לא מספקים, ומראים שמודל ה-RF הצליח בקלות להבחין בין הדגימות האמיתיות לבין המזויפות. מודל ה-GAN הצליח לשטות בו בדגימות בודדות, אך בסך הכל שני המודלים נכשלו בבדיקה.

על מנת לנסות ולהבין מה יש בדגימות המזויפות שגורם למודל לגלות אותן ניסינו לבנות מודל שניתן להסביר את בחירותיו. החשש המרכזי שלנו היה שנראה שיש פיצ'ר מסוים / ערך מסוים / כניסה מסוימת בוקטור שמסגירה בוודאות שמדובר בדגימה מזויפת. ל"מזלנו", זה לא המצב, וכדי להגיע לחיזוי גם המודל הפשוט ביותר היה צריך לעבור דרך מספר צמתי החלטה. בדיקה זו בוצעה על ה-GAN, היות והוא כן הראה ביצועים מעט יותר טובים לפי detection (אך עדיין גרועים) בחלק זה:



בהמשך לזה, ניסינו לבדוק האם יש פיצ'רים ספציפיים שאנחנו פשוט מג'נרטים ממש ממש גרוע, וקיבלנו את הרשימה הבאה של פיצ'רים בעלי חשיבות לקבלת ההחלטה של המודל (בסדר יורד). כנראה שבתחום של הפיצ'רים הנומריים דווקא המודל לא למד טוב את התפלגות הערכים המאפיינת רשומות באופן "אמין", היות ופיצ'ר 1 (שאמור להיות fnlwgt) הוא באופן משמעותי המסגיר ביותר:

Feature Importances (Top features that separate real and synthetic data):
Feature 1: 0.9416
Feature 4: 0.0366
Feature 2: 0.0156
Feature 46: 0.0028
Feature 43: 0.0019
Feature 9: 0.0015

בשלב השני נרצה לבדוק האם הדגימות המג'ונרטות מהוות חליף טוב לדגימות האמיתיות ע"י אימון מודל RF פעם אחת לדאטה האמיתי ופעם אחת למזויף. אם אותו המודל מצליח להגיע לציון AUC דומה בין שני ה-settings על ה-test set סימן שהדאטה שימושי. נבחן ע"י היחס בין שני הציונים: Real AUC\Fake AUC.
עבור בדיקה זו קיבלנו: AVG Efficacy Score: 0.59 עבור מודל ה-cGAN, לאורך 3 ניסויי SEED שונים, ציון שאינו טוב במיוחד, היות והבעיה היא בעיית סיווג בינארית. למעשה, מודל RF שניסה לחזות על נתוני המקור הגיע בקירוב לביצועים של 0.9, בעוד על הנתונים המזויפים הוא הגיע לביצועים של כ-0.55, משמע שהם לא הצליחו לקרב את הפונקציונאליות של נתוני המקור, והיות והבעיה היא בינארית, מראים גם עם כישלון כמעט מוחלט להשתמש בנתונים הסינטטיים לחיזוי (מידת דיוק רק קצת יותר גבוהה משל הטלת מטבע). ציוני ה-efficiency היו 0.68, 0.56, 0.52.

מודל ה-GAN קיבל ציון efficiency ממוצע נמוך יותר - 0.55 - פער שאינו מפתיע מבחינה תיאורטית, שכן אנחנו מצפים שמודל המג'נרט דגימות לפי לייבל יצליח לשכנע יותר טוב מודל ML. ציוני ה-efficiency שלו היו 0.62, 0.53, 0.50.

הביצועים הגרועים בשני המדדים מראים שכנראה למידע הסינטטי יש סממנים מסגירים הקשורים ביחסים בין הפיצ'רים השונים. בגדול, זה כנראה אומר שה-GAN לא למד מספיק טוב את היחסים בין הפיצ'רים ולכן מייצר דגימות שאינן אמינות.

הטסטים וחישובי המדדים נמצאים בקובץ Analysis.ipynb.

פתרונות אפשריים יכולים להיות מודל יותר מורכב, שימוש בארכיטקטורת WGAN, שימוש בקשרים residual , יצירת תהליך אימון יותר מורכב הלומד תוך שילוב ב-loss של המסקנות על feature importance או שיטות אוגמנטציה יותר מורכבות (שיטת האוגמנטציה בה השתמשנו לא הספיקה כדי לשפר המדדים). החלטנו לנסות מודל חזק יותר שעשוי לעזור תוך שימוש ב-AE כמתווך ושופט, וגם כדי שנוכל לומר "ניסינו".

הגדרת ארכיטקטורת BEGAN:

כאמור, מתוך תקווה לשיפור בביצועים, ניסינו להעביר המשימה למודל BEGAN המשתמש ב-critic - רשת מסוג AE בתור discriminator (נקראת Critic בקוד) שמטרתה לבנות באופן טוב את הדגימות האמיתיות (reconstruct) ובאופן לא טוב את הדגימות המזויפות. ה-loss שלה נקנס על כמה טוב נבנו הדגימות המזויפות ביחס לדגימות האמיתיות. מנגד, G בארכיטקטורה זו מנסה לדאוג שהדגימות המזויפות כן ייבנו היטב, ומהווה קונטרה, כאשר דאגנו שיקנס על דגימות בהן הקורלציה בין הפיצ'רים לא טובה.

```
# === Generator Step ===
# Directive: Improve reconstruction of fake samples.
# L_G = L_D_fake = abs(reconstructed_fake_data - fake_data)
# Loss term is added with correlation loss.
```

```
# === Critic Step ===
# Directive: Build real samples properly and fake samples badly.
# L_D = L_D_real - k*L_D_fake
# L_D_real = abs(reconstructed_real_data - real_data)
# L_D_fake = abs(reconstructed_fake_data - fake_data)
```

הארכיטקטורה נחשבת שיפור לארכיטקטורה המקורית, ואמורה לאזן יותר טוב מצבים של mode collapse.
הארכיטקטורה מגדירה פרמטר k שאמור לאזן באופן אוטומטי את השיפור של G למול D. אמנם לא אבחנו בתוצאות מצב של mode collapse באופן מפורש, אבל אנחנו מקווים שמכיוון שארכיטקטורה מעבירה את הדאטה המקורי דרך encoder למרחב לטנטי קטן יותר, זה יקל על ההתמודדות עם הערכים הקטגוריאליים, ושמכיוון שהיא נשענת על פרמטר איזון k, תהליך האימון של G ושל D יהיה יותר טוב. בשלב הראשון, אימנו מודל AE אותו מודל ה-BEGAN מקבל כקלט ויעשה לו fine-tune במהלך האימון לצרכיו.

הגדרת מודל AutoEncoder:

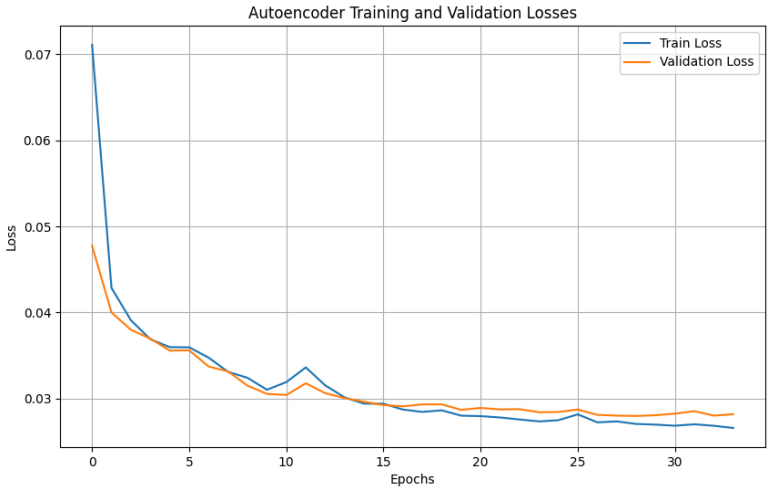
בתחילה, היה למודל קשה להתכנס והוא היה ב-underfit משמעותי, אז התאמנו את ה-learning rate שלו באמצעות scheduler לצמצום כאשר הירידה ב-val_loss יורדת (plateau), תוך שהתחלנו עם LR גדול פי 10 משל ה-GAN. הגדרנו לו גם עצירה מוקדמת, ללא warm up. ברגע שהעברנו את המודל לפונקציית loss מסוג nn.SmoothL1Loss ראינו שיפור משמעותי ואימון שהיה נראה טוב. ההשערה שלנו היא שזה נובע מכיוון שפונקציה זו מותאמת יותר לעבוד עם טווח הערכים בין [1, -1] אותו הגדרנו ב-dataset ואליו ה-reconstruct מכוון בסופו של דבר. פונקציית האקטיבציה TanH נבחרה גם פה לאור טווחי הערכים שנורמלו ב-dataset. כמו כן, פונקציית ה-loss הוגדרה בנפרד עבור הערכים הקטגוריאליים והנומריים, לאור נטייה של הקטגוריאליים להיבנות טוב ושל הנומריים להציג ביצועים פחות משכנעים, כך שה-loss של כל קומפוננטה אוזן להיות כ-50% מה-loss השלם.
הפרמטרים של המודל היו:

```
NOISE_DIM = 32  # The size of the initial noise vector
LATENT_DIM = 64 # The dimension of the latent (encoding) dimension

# Encoder Configuration
ENCODER_CONFIG = [
    {"input_dim": DATA_DIM, "output_dim": 1024, "activation": nn.ReLU(), "norm": 'batch', "dropout": 0.2},
    {"input_dim": 1024, "output_dim": 512, "activation": nn.ReLU(), "norm": 'batch', "dropout": 0.2},
    {"input_dim": 512, "output_dim": 256, "activation": nn.ReLU(), "norm": 'batch', "dropout": 0.2},
    {"input_dim": 256, "output_dim": 128, "activation": nn.ReLU(), "norm": 'batch', "dropout": 0.2},
    {"input_dim": 128, "output_dim": LATENT_DIM, "activation": None, "dropout": 0.0},  # Latent space, no activation
]

# Decoder Configuration
DECODER_CONFIG = [
    {"input_dim": LATENT_DIM, "output_dim": 128, "activation": nn.ReLU(), "norm": 'batch', "dropout": 0.2},
    {"input_dim": 128, "output_dim": 256, "activation": nn.ReLU(), "norm": 'batch', "dropout": 0.2},
    {"input_dim": 256, "output_dim": 512, "activation": nn.ReLU(), "norm": 'batch', "dropout": 0.2},
    {"input_dim": 512, "output_dim": 1024, "activation": nn.ReLU(), "norm": 'batch', "dropout": 0.2},
    {"input_dim": 1024, "output_dim": DATA_DIM, "activation": nn.Tanh(), "dropout": 0.0},  # Reconstructed output
]
```

המודל הצליח להתאמן יפה, הראה ירידה הדרגתית ב-loss והגיע ל-reconstructions דיי משכנעים כשהופעל, עם test_loss נמוך של 0.0111 לאחר האיזון בין הקומפוננטות. מתוך תהליך הלמידה שלו:



בתור POC שהמודל אכן טוב, בנינו מחדש דגימה שלו. וירא כי (דיי) טוב:

```
Original: tensor([-0.6164, -0.8565,  1.0000, -1.0000, -1.0000, -0.2041, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000,  1.0000, -1.0000, -1.0000,  1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000,  1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
     1.0000, -1.0000, -1.0000,  1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
     1.0000, -1.0000,  1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000])

Reconstructed: tensor([-0.5976, -0.8461,  0.9004, -0.9850, -0.9998, -0.1830, -0.9995, -0.9989,
    -0.9988, -1.0000, -1.0000, -1.0000,  0.9943, -1.0000, -1.0000,  1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000,  1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
     1.0000, -1.0000, -1.0000,  1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
     1.0000, -1.0000,  1.0000, -1.0000, -1.0000, -0.9999, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000, -1.0000,
    -1.0000, -1.0000, -1.0000, -1.0000])
```

הבנייה מחדש שומרת יפה על הערכים הקטגוריאליים, לרוב בסטייה של פחות מ1%, סטייה מקסימלית של 10%. הערכים הנומריים גם נבנים יחסית טוב, למרות שבהם הקושי קצת יותר גדול, ורוב הסטיות המשמעותיות מערכי המקור הן שם.

בשלב הבא ניגשנו לאימון מודל ה-BEGAN. מבחינת ארכיטקטורה, השארנו את G במבנה זהה לשל מודלי ה-GAN הקודמים, רק שכעת ה-output שלו הוא במימד LATENT_DIM, ועובר הסבה למימד הרגיל ע"י ה-Decoder במסגרת פונקציית generate. בנוסף, נדרשנו כעת להגדיר מספר פרמטרים חדשים (המתווספים לפרמטרים הקיימים במודל הקודם):

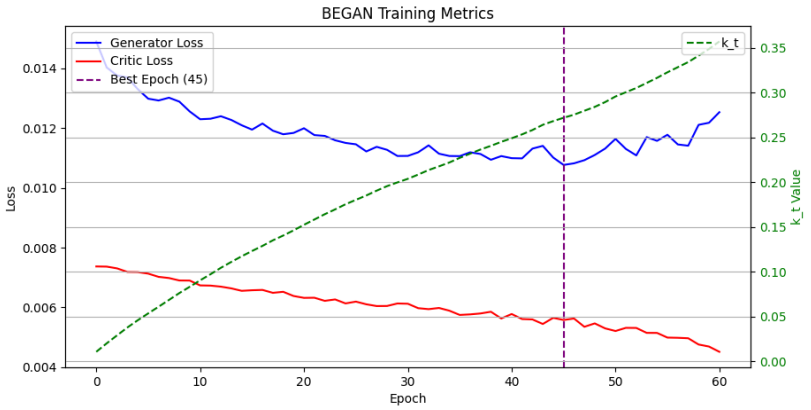
```
LAMBDA_K = 1e-2     # Learning rate for `k_t` balance term, for BEGAN.
GAMMA = 0.75        # BEGAN balance factor
```

הפרמטר Gamma אמור לאזן יחסי כוחות רצויים בין G לבין C, ומבטא את יחס ה-loss הרצוי ש-D יפיק על הדגימות המזויפות חלקי הדגימות האמיתיות. ז"א, בקונפיגורציה הזו אנחנו לא מצפים שהדגימות המזויפות יקבלו בדיוק אותו reconstruction loss, אלא נותנים להן הנחה מסוימת. באופן זה המודל מעודד ליצור דגימות מגוונות יותר, גם אם מעט פחות ריאליסטיות, ובכך לפתור את בעיית ה-mode collapse. הפרמטר LAMBDA_K אמור להוות קצב למידה של העלייה / ירידה בפרמטר K, שאמור להצביע על ההפרשים ביחסי הכוחות שלהם ולאזן בהתאם. במילים אחרות - כמה אגרסיבי צריך להיות האיזון ביניהם.

מכיוון שה-Critic הוא AE מאומן, הגדרנו כ-25 איטרציות חימום בהן G יתאמן עם קצב למידה גדול פי בערך 1e-5 מקצב הלמידה של C, היות ואנחנו לא באמת מצפים לעדכן אותו בזמן זה (קצב הלמידה המדויק הוא בעלייה ככל שמתקדמים באפוקי החימום). לאחר מכן, קצב הלמידה של C מתייצב על ערך הקטן פי 1e3 מקצב הלמידה של G, מכיוון שאנחנו לא רוצים שיאבד את יכולות הבניה שרכש ב-pretrain. כמו כן, הגדרנו שמשקל ההשפעה של ה-corr_loss יעלה כאשר ה-Critic חזק, וירד כאשר G חזק. הגדרנו nn.SmoothL1Loss בתור loss בין הווקטורים, עם התייחסות נפרדת לעמודות מספריות וקטגוריאליות (בדיוק כמו ב-AE), על מנת שהפרשים משמעותיים יודגשו. זאת מכיוון שאיכות המשימה נמדדת על הבנייה של הווקטורים (האמיתי והמזויף) ולא על ההפרשים בין האמיתי למזויף. גם פה הוספנו ל-loss של G התייחסות לקורלציה בין הפיצ'רים שהוא מג'נרט, בתקווה שילמד לייצר דאטה מציאותי. לאור המנגנונים האוטומטיים לאיזון, בחרנו שלא לתת ל-G יותר איטרציות פר איטרציה של C ולא הגדרנו פונקציונאליות זו במודל. מלבד זה יתר פרמטרי האימון זהים ל-gan.py, כולל smoothing ו-gradient clipping ואמורים לשרת את אותו הרציונל. גם פה, מתוך כוונה להשוות בין ביצועי המודלים, שמרנו על ארכיטקטורות ו-settings זהים בין מודל ה-BEGAN למודל ה-cBEGAN, כאשר ההבדל היחיד ביניהם הוא העברת ה-labels למודל ה-conditional.

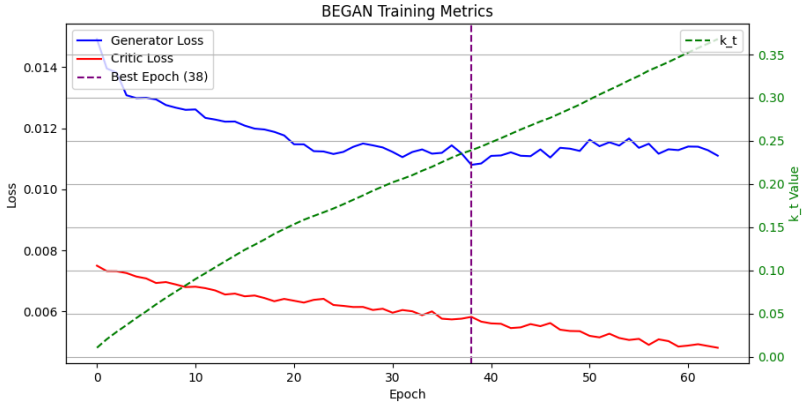
בתהליך האימון, היות ומודל ה-AE כבר מאומן, אנחנו מצפים לראות loss ללא שינוי משמעותי עבור C, כאשר G, הלומד לייצר דגימות דומות למציאות ולכן המקבלות פענוח משמעותי, אמור לשפר בהדרגה את ה-loss שלו. אחרי עלייה הדרגתית, אנחנו מצפים לראות תנודתיות מסוימת (בסקאלה של 0.2-0.7) גם בערך ה-k שתצביע על כך שהארכיטקטורה עובדת על איזון של יחסי הכוחות בין האיטרציות. כמו כן, מודל זה נדרש ביותר אפוקים באופן משמעותי, ולכן שינינו את ההגדרה שלו לחימום במשך 25 אפוקים, עצירה מוקדמת אחרי 15 אפוקים וסך הכל אימון על פני 250 אפוקים, כאשר גם פה עצירה מוקדמת תקבע ע"י ה-loss של G.

אימון ה-BEGAN:



נראה שבסך הכל הצלחנו לגרום ל-G להשתפר למול C, ועצרנו כאשר C התחיל להתגבר באופן גורף על G, לאחר 45 אפוקים. נראה שפרמטר שיווי המשקל לא הגיע בפרק זמן זה למצב של איזון. יכול להיות שנדרשו יותר אפוקים על מנת להגיע לשם, על אף שמדאיג "לאבד" את האופטימום המקומי של ביצועי G.

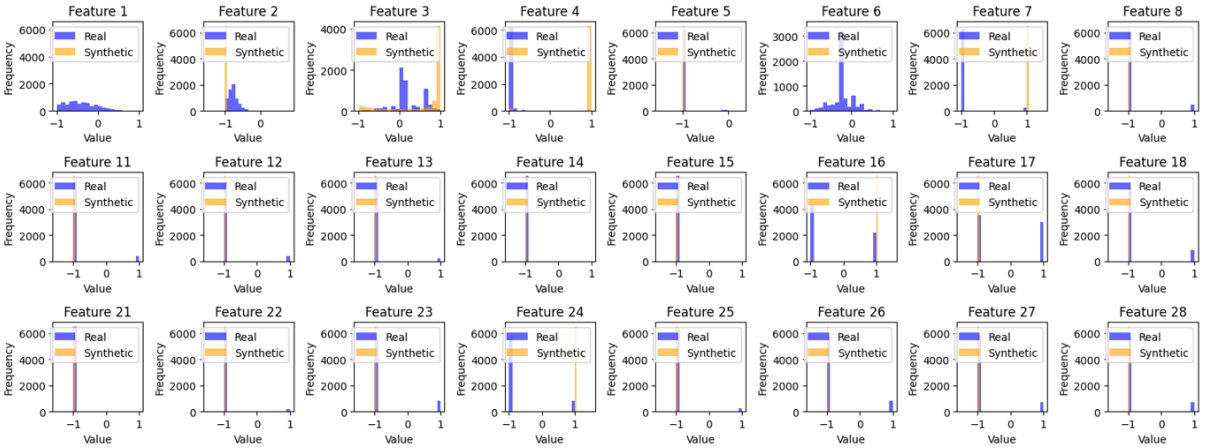
אימון ה-cBEGAN:



המודל מפגין התנהגות דיי דומה לזו של ה-BEGAN, למרות שנראה שאפילו היה פחות יציב בשיפור של G. בניסיון זה ניסינו לאפשר עצירה מוקדמת אחרי יותר אפוקים, אך לא נראה שזה עזר לנו.
המימוש של שני המודלים נמצא במודול began.py.

ביצועי המודלים:

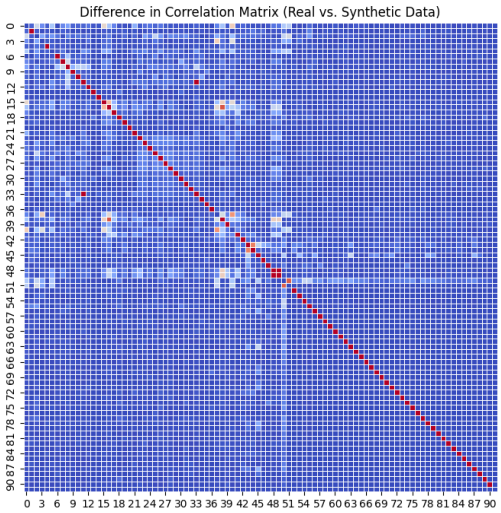
בדומה לסט המודלים הקודם, רצינו לבדוק איך מתפלגים הפיצ'רים במידע המג'ונרט היוצא ממודל זה:



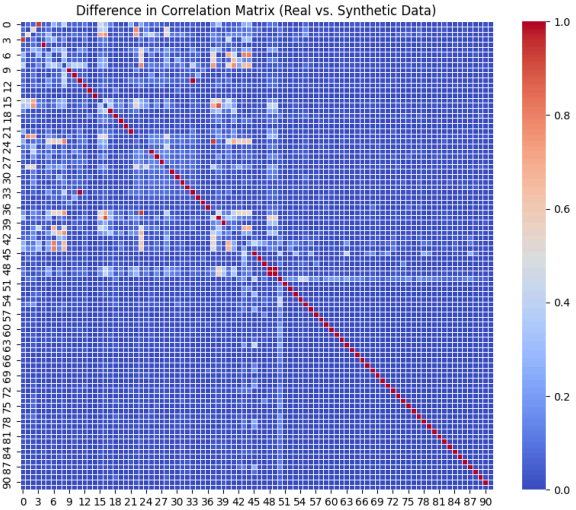
ההתפלגות של מודל ה-BEGAN נראית **לא טוב**, עם שוני דרסטי בהתפלגות בין הפיצ'רים האמיתיים לבין המג'ונרטים, שגם נראה שממוקדים על ערכי קצה באופן יחסי. ההתפלגות של מודל ה-cBEGAN נראית דיי דומה, עם פערים מאוד ברורים בייצוג של פיצ'רים שונים, בעיקר הנומריים. נראה כאילו שני המודלים יצרו התתפלגויות מאוד שונות לעומת המידע האמיתי ולא הצליחו לחקות אותו. היות וראינו שויזואליזציה זו לא מייצגת מידע מעניין, הסרנו אותה מהדוח.

בדקנו את מטריצות הקורלציה של הנתונים, ובשני המודלים נראה שהביצועים היו פחות טובים מאשר במודלי ה-GAN וה-cGAN:

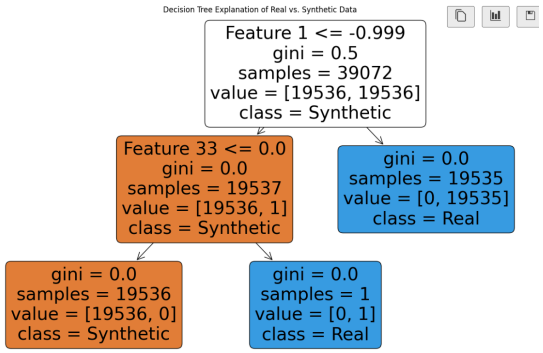
BEGAN:



cBEGAN:



כמו כן, בדקנו את ציוני ה-efficiency וה-detection של המודלים, למרות שהתפלגות הפיצ'רים שלהם הבהירה לנו שלא מפה תבוא הישועה. שני המודלים ביצעו מאוד גרוע (ציונים ממוצעים של 1.000 ב-detection וסביב 0.55 ב-efficiency, אבל היות וההתפלגות פיצ'רים בדגימות עצמן לא משכנעת, אין ממש מה להעמיק בזה). בבדיקת העץ של מודל ה-BEGAN ראינו שהפיצ'רים הנומריים מסגירים לגמרי מה דגימה אמיתית ומה מזויפת:



זו דוגמה לדגימה קטסטרופלית שלא מצליחה בכלל לדמות את הדאטה המקורי.

לסיכום, לא הצלחנו לשפר את ביצועי ה-GAN תוך שימוש בארכיטקטורה מבוססת AE. ייתכן שצורת האימון שלנו לא הייתה מספיק טובה, לא נוצלה באופן מיטבי או לא אוזנה כמו שצריך, אבל בהחלט ניסינו. קיימות ארכיטקטורות נוספות שאולי שווה לנסות, אבל את זה כבר לא נעשה במסגרת העבודה הזו.