

Assignment 4 - Report

Shahar Oded, 208388918
Nir Rahav, 316275437

הקמה של ה-TabularDataset:

אנחנו מתעסקים פה עם דאטה טבלאי, פשוט יחסית, עבורו אנחנו יודעים לתכנן מראש את הערכים הייחודיים בכל עמודה ואת הסוג שלהם. נציג מידע סיכומי על הדאטה:

[Info]: Number of rows: 32561
[Info]: Number of columns: 15

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

[Info]: Categorical columns: 9
[Info]: Continuous columns: 6
[Info]: Number of columns after one-hot encoding: 93 == Embedding Vector Dimension (with 1 label column)

בסינון זה, ערכים ייחודיים בעמודות שלא מופיעים בדאטה עצמו אינם מיוצגים בווקטור. מכיוון שאנו עוסקים בהתפלגות הנתונים בפועל, 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 ל-2 עמודות ולא ל-1. זה משפיע לאחר מכן על האופן בו נייצג מידע קטגוריאלי מג'ונרט.

אנחנו רואים imbalance בעמודת ה-target אבל לא משהו מאוד חריג. נרצה לבצע stratified split כשנעסוק בחלקים מהדאטה הזו. כמו כן נרצה לאפשר ביצוע אוגמנטציה באמצעות SMOTE על מנת להוסיף קצת גיוון בערכים בזמן האוגמנטציה ולנסות למנוע מצב של mode collapse בו ה-generator לומד ייצוגים של ה-majority class בלבד. בפועל, שיטה זו לא עזרה באימון והשארנו הדאטה בהתפלגותו הרגילה. כמו כן, את הערכים הקטגוריאליים נתרגם באמצעות one-hot-encoder לכניסות נפרדות, ואת הערכים הנומריים נגמל 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. היות ונשתמש בפונקציית TanH כדי לייצר את 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 באובייקט 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). כך נוכל לג'נרט רשומות אמינות יותר, לפחות בהיבט הזה. על מנת שיעבוד, מיפוי ערכים חסרים ('?') לערך 'Unknown', ואלו מהווים עוד אפשרות לערך בקטגוריות הרלוונטיות.

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-discriminator שעבורו רצינו ליצור ארכיטקטורה פשוטה יותר שתשרת משימת קלאסיפיקציה יחסית "קלה". זו הארכיטקטורה 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 חיצוניות:

```
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.

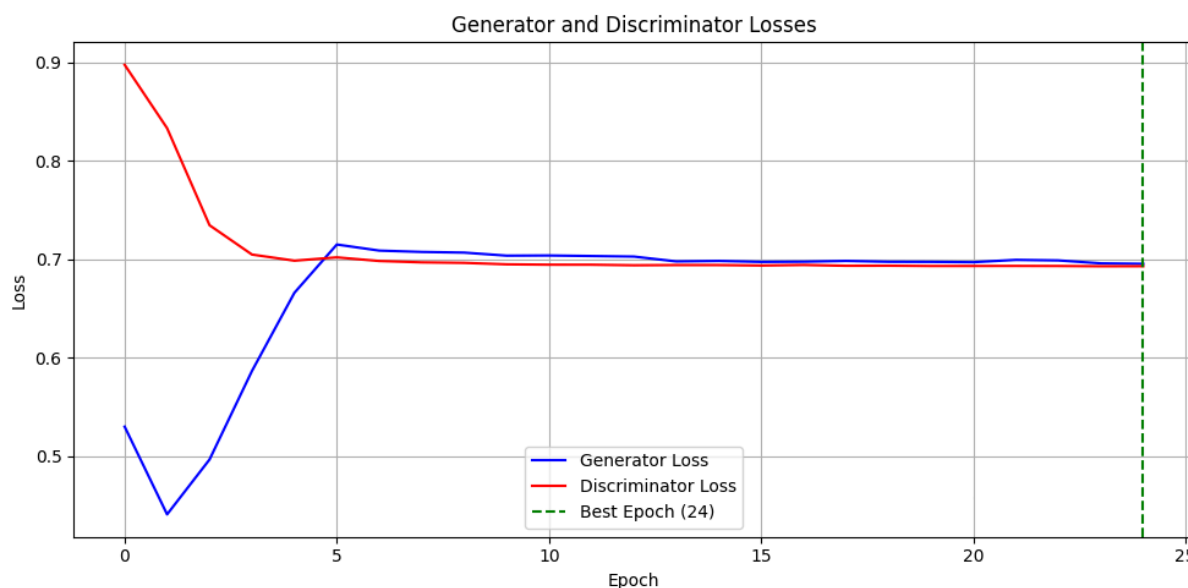
בנוסף, לאחר הרבה התנסויות עם המודלים ואחרי שראינו שקורלציה בין הפיצ'רים היא בעיה חוזרת, הוספנו loss_term נוסף באימון הבדוק גם את הפרש הקורלציות הממוצע בין המידע האמיתי למידע המג'נרט. את הערך הזה הכפלנו ב-LAMBDA_CORR וזה penalty הנוסף בכל פעפוע loss ב-generator.

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 אופטימיזצור ADAM עבור שני המודלים, עם LR ו-W2 כמופיע בתמונה ועם פונקציית loss מסוג BCELoss לאור האופי הבינארי של הבעיה. 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 קצב הלמידה לאחר warmup ולכן רגיש לצורת האימון פה. בניסיון נוסף לאזן עוד יותר בין המודלים חקרנו W2 רק עבור הדיסקרימינטור, עם LR הקטן פי 4 מזה של הג'נרטור, מה שעזר לנו לייצב את תהליך האימון. לסיום, כדי לוודא שלא יתפוצצו לנו הגרדיאנטים בגלל השאיפה לערכים קטגוריאליים הגדרנו gradient clipping ל-G וגם ל-D.

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 \log_2 - 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 הדיסקרימיננטור *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 dimension 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, המועברת ע"י ה-dataloader (ומקודם פשוט לא נעשה בה שימוש).

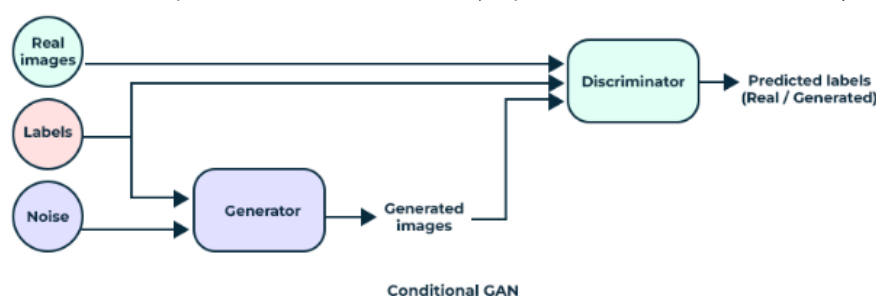
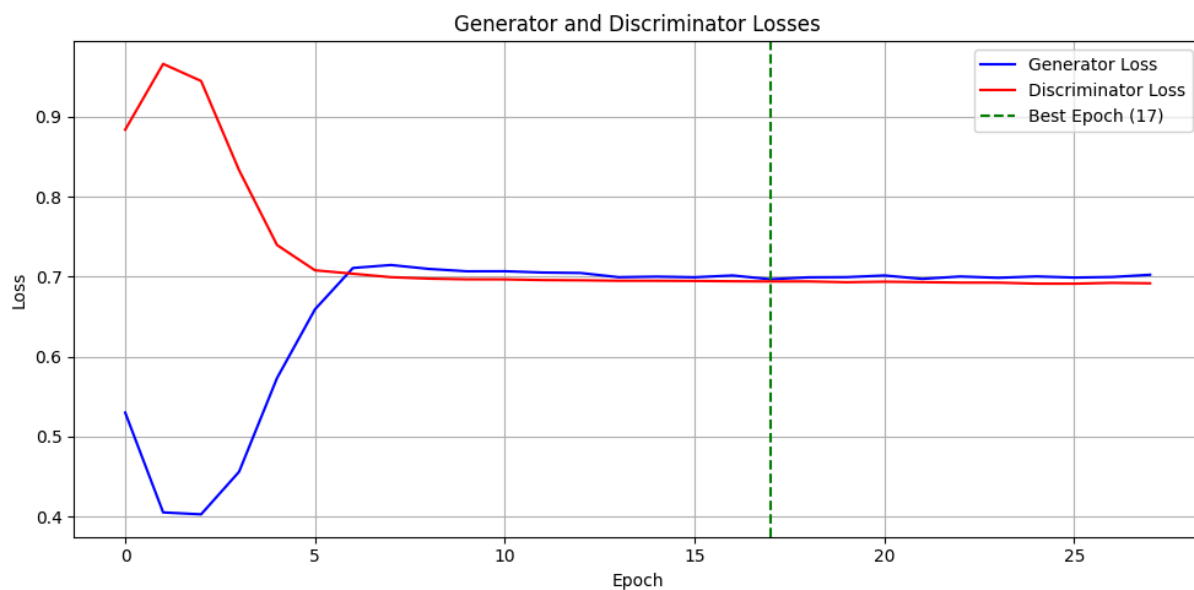


Figure 1: the conditional GAN architecture

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 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.

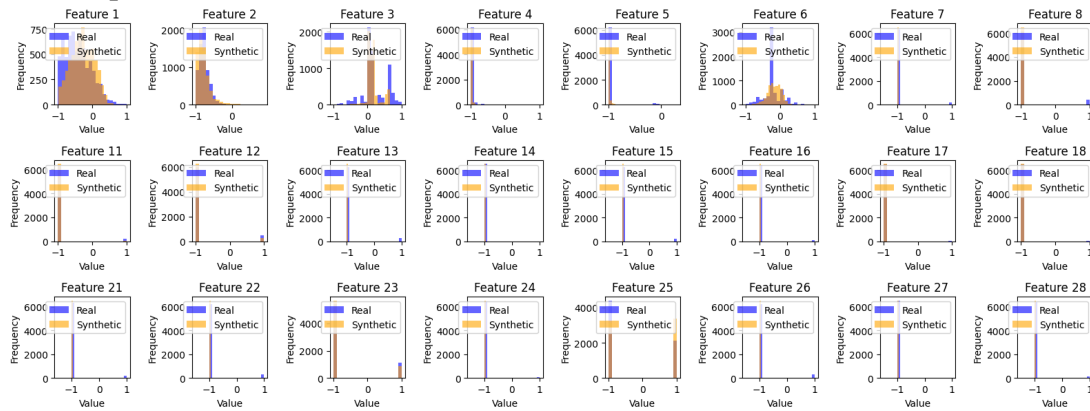


את המימוש המלא לשני המודלים ניתן למצוא במודול `gan.py`.

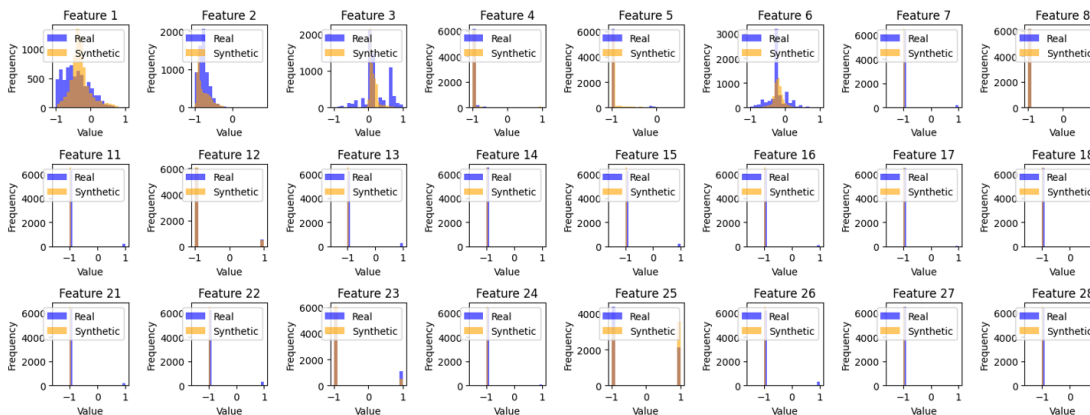
ביצוע Feature Analysis:

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). אפשר לראות דוגמה לפילוח מלא במחברת.

GAN (top few):



cGAN (top few):

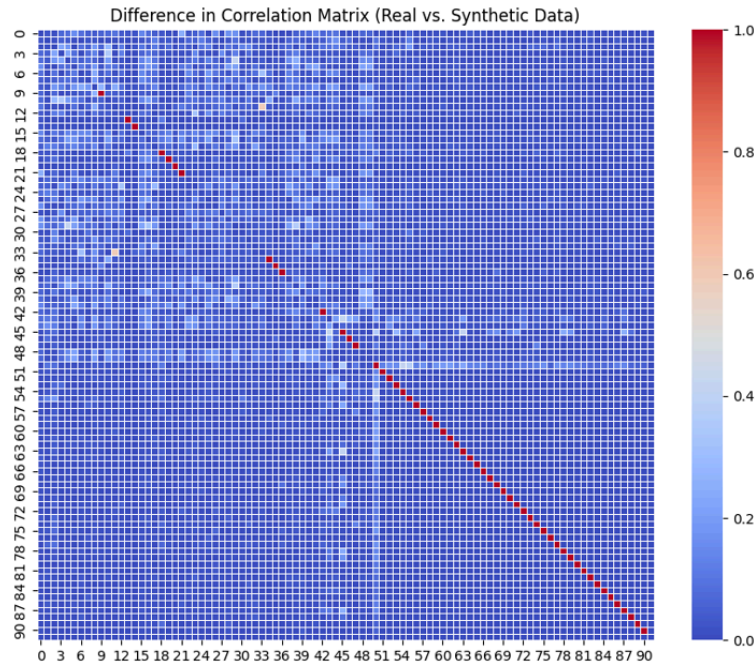


בבדיקה זו, לדעתנו, מודל ה-GAN מציג התפלגויות מעט יותר קרובות למציאות, עם פחות מרכז סביב ערכים ספציפיים. התרשימים מראים לנו שהמודלים מצליחים בסך הכל לייצר גם את הערכים הנומריים וגם הערכים הקטגוריאליים, בהתפלגויות יחסית דומות למקור. אם למשל היינו רואים "פעמון" באיזור שונה בסקאלה מזה בו הנתונים האמיתיים נמצאים - היינו דואגים. זה לא ממש המצב, למרות שכן ניתן לראות סטייה מסוימת עבור feature 1, למשל. בבדיקה זו היא ראשונית ונחמדה, אבל בשלבים הבאים נרצה לבחון האם הנתונים אמנים. האם רשומה מסוימת משכנעת אותנו שהיא אמיתית.

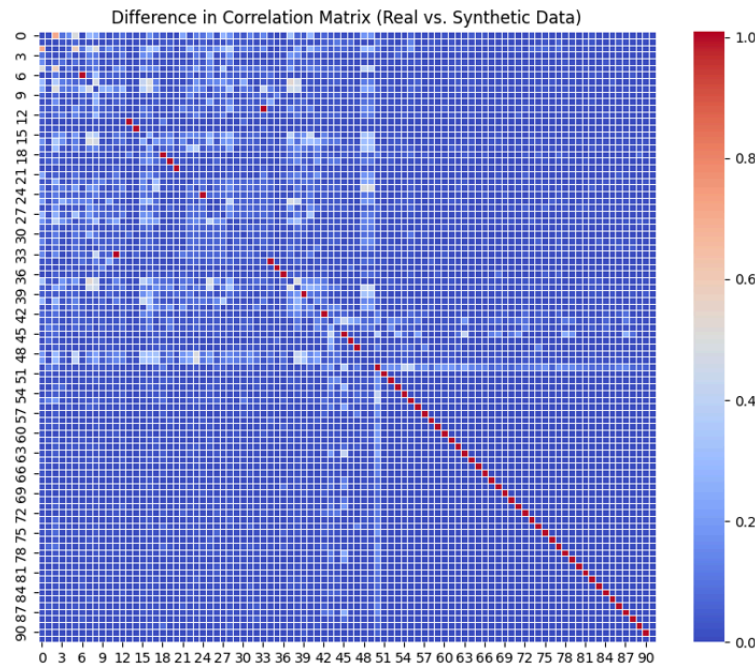
לשם כך בשלב השני נבדוק את הקורלציות בין הפיצ'רים, והאם דומות לקורלציות בין הפיצ'רים בנתונים האמיתיים. על מנת לפרשן יותר בקלות, יצרנו מטריצת קורלציות המראה את ההפרשים בקורלציות בתוך הנתונים המזויפים למול בתוך הנתונים האמיתיים.

בפשטות יחסית, תמונה בה הצבע הכחול שולט יותר מציגה לנו למעשה שההבדלים בקורלציות בין הפיצ'רים שלו יותר קטנים, ז"א שהוא יחסית מייצג יותר את היחסים בין הפיצ'רים. ערכים אדומים מוחלטים - כנראה ערכים שהמודל לא הצליח לייצר בכלל. 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 שכשנסה להשתמש בנתונים אלו לאימון מודל, הוא יוכל להבחין שאינם קוהרנטיים עם עצמם, ושהקורלציות לא נשמרות.

מהאיור נראה שמודל ה-cGAN יצר קורלציות פחות משכנעות (יותר "מוכתמות" משמע פחות מקורלציות ביחס לדאטה האמיתי). כמו כן אנחנו רואים חולשה יותר משמעותית ברבע העליון השמאלי של המטריצה. כנראה שהנתונים שם יותר קשים לג'נרט אמין, והמודל מתקשה לעשות זאת. קיימת הבעיה של הערכים הקטגוריאליים, בהם נראה ששני המודלים מתקשים. 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 (הסימונים האדומים המובהקים).

ביצוע הערכה לדאטה הסינטי באמצעות **detection & efficiency**:

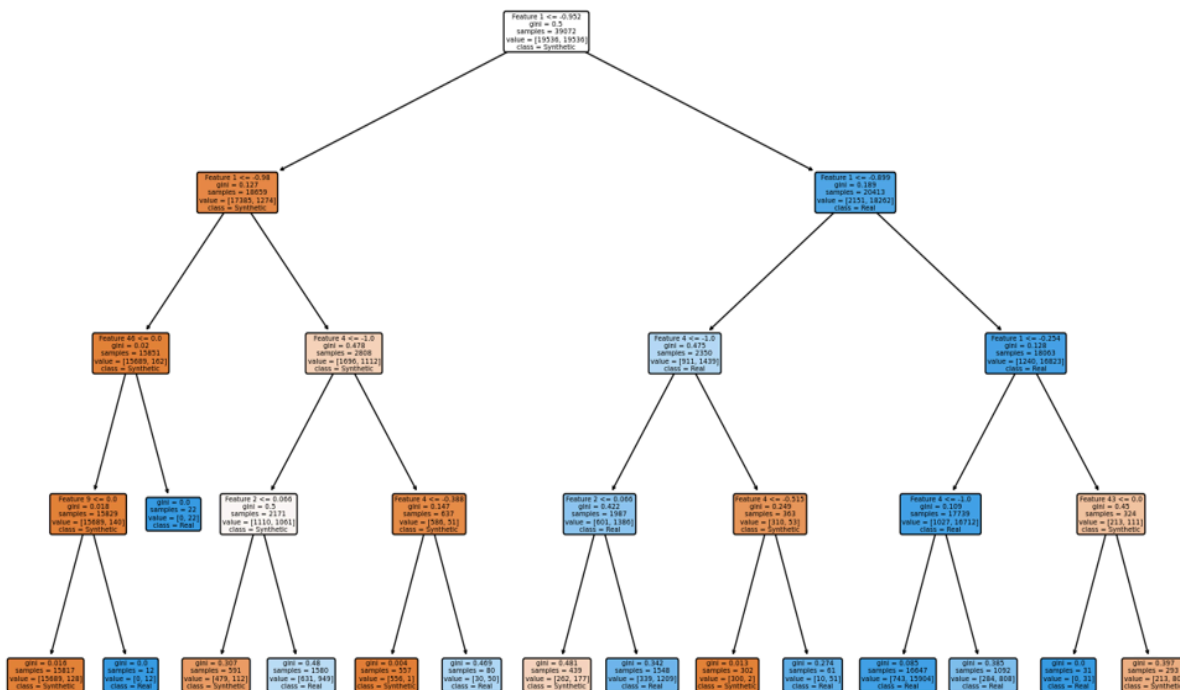
התוצאות בחלק זה מתייחסות למודלי GAN ו-cGAN הרגילים. הניסויים בוצעו על חלוקות train-test באמצעות 3 ערכי SEED שונים (42, 76, 2005) והתוצאות המופיעות פה הן ממוצע של ציוני הביצוע שהתקבלו ע"י 2 המודלים. בשלב הראשון, בהתאם לבקשה, ניקח את ה-train_set וgenerated set באותו הגודל (מושפעים מערך ה-SEED) ובאמצעות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

עבור מודל ה-GAN ביצועי ה-detection בממוצע היו 0.997.

עבור מודל ה-cGAN ביצועי ה-detection בממוצע היו: 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. "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 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.

מודל ה-GAN קיבל ציון efficiency ממוצע נמוך יותר - 0.55 - פער שאינו מפתיע מבחינה תיאורטית, שכן אנחנו מצפים שמודל המג'ונרט דגימות לפי לייבל יצליח לשכנע יותר טוב מודל ML.

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

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. מגד, G בארכיטקטורה זו מנסה לדאוג שהדגימות המזויפות כן ייבנו היטב, ומהווה קונטרה, כאשר דאגנו שיקנס על דגימות בהן הקורלציה בין הפיצ'רים לא טובה. מכיוון שארכיטקטורה זו מעבירה את הדאטה המקורי דרך encoder למרחב לטנטי קטן יותר, אנחנו מקווים שזה יקל על ההתמודדות עם הערכים הקטגוריאליים. הארכיטקטורה מגדירה פרמטר k שאמור לאזן באופן אוטומטי את השיפור של G למול D, מה שאנחנו מקווים שיעזור לנו להגיע לביצועים יותר טובים של שני חלקי המודל, שיאמנו זה את זה באופן הדדי. בשלב הראשון, בנינו מודל AE אותו מודל ה-BEGAN מקבל כקלט ויעשה לו fine-tune במהלך האימון לצרכיו.

הגדרת מודל AutoEncoder:

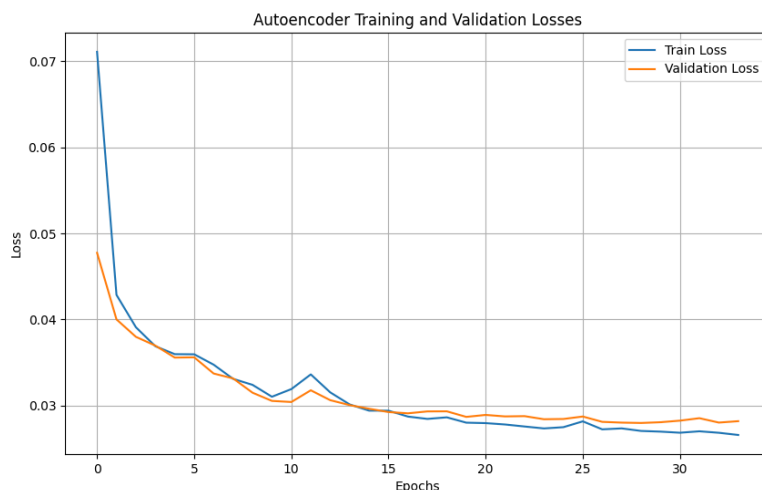
בתחילה, היה למודל קשה להתכנס והוא היה ב-underfit משמעותי, אז התאמנו את ה-learning rate שלו באמצעות scheduler לצמצום כאשר הירידה ב-val_loss יורדת (plateau), תוך שהתחלנו עם LR גדול פי 10 משל ה-GAN. הגדרנו לו גם עצירה מוקדמת, ללא warm up. ברגע שהעברנו את המודל לפונקציית loss מסוג nn.SmoothL1Loss ראינו שיפור 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 בבעיה. 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 אופן 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:



```
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,
```

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, -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, -1.0000, -1.0000, -1.0000, -1.0000])

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 במידת LATENT_DIM, ועובר הסבה למימד הרגיל ע"י ה-Discriminator. בנוסף, נדרשנו כעת להגדיר מספר פרמטרים חדשים (המתווספים לפרמטרים הקיימים במודל הקודם):

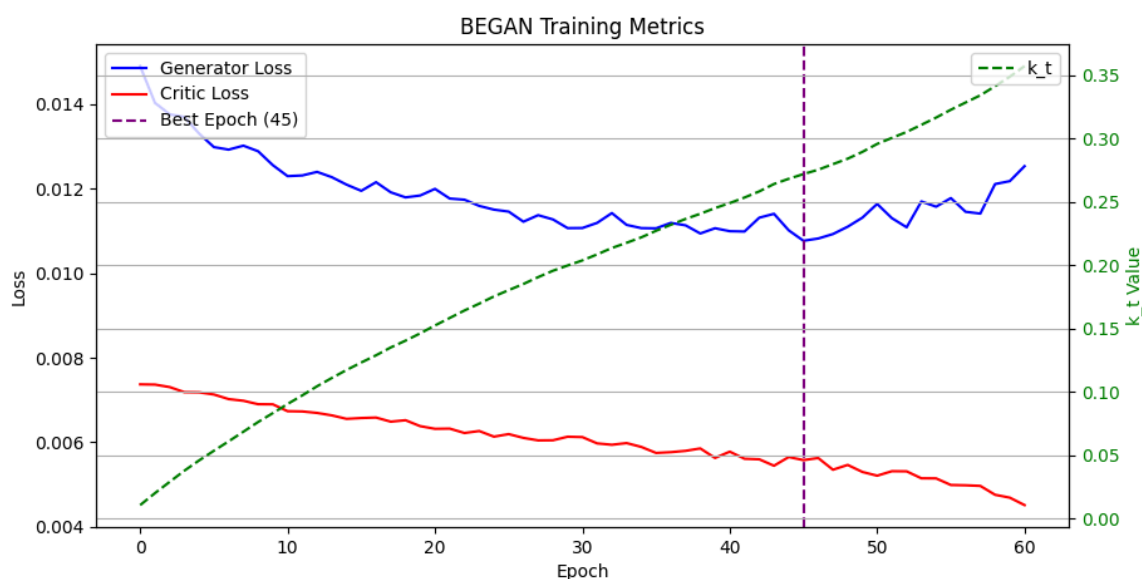
```
LAMBDA_K = 1e-2      # Learning rate for `k_t` balance term, for BEGAN.
GAMMA = 0.75         # BEGAN balance factor
```

הפרמטר Γ 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 $\text{reconstruction loss}$, 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 להיות קצב למידה של העלייה / ירידה בפרמטר K , שאמור להצביע על הפרשים ביחסי הכוחות שלהם ולאזן בהתאם. במילים אחרות - כמה אגרסיבי צריך להיות האזינו ביניהם.

מכיוון שה-Critic הוא AE מאומן, הגדרנו כ-25 איטרציות חימום בהן G יתאמן עם קצב למידה גדול בבערך 5-1e 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, על מנת שהפרשים משמעותיים יודגשו, בדומה לאופן האימון של ה-AE. 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'ר'ים 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 פונקצינאליות זו במודל.

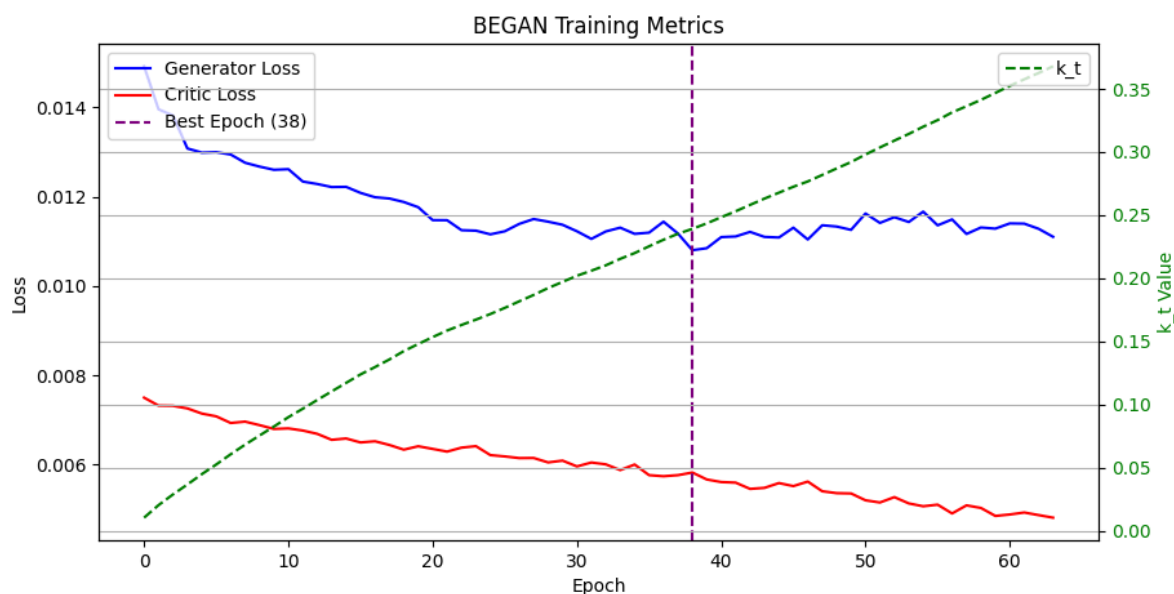
בתהליך האימון, היות ומודל ה-AE כבר מאומן, אנחנו מצפים לראות loss ללא שינוי משמעותי עבור C, כאשר G, הלומד לייצר דגימות דומות למציאות ולכן המקבלות פענוח משמעותי, אמור לשפר בהדרגה את ה-loss שלו. אחרי עלייה הדרגתית, אנחנו מצפים לראות תנודתיות מסוימת (בסקאלה של 0.2-0.7) גם בערך ה-k שתצביע על כך שהארכיטקטורה עובדת על איזון של יחסי הכוחות בין האירטציות. 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 ביחס לביצועי G.

אימון ה-cBEGAN:

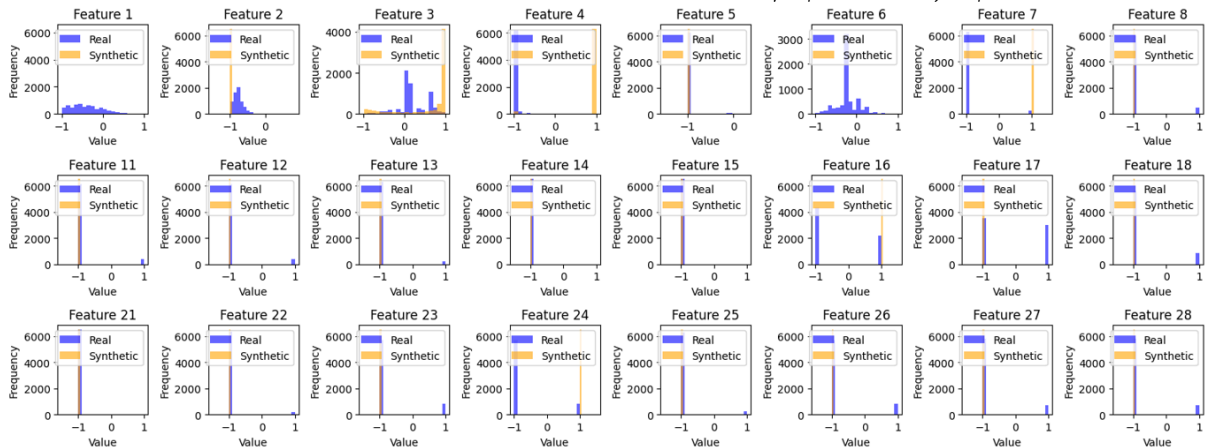


המודל מפגין התנהגות דיי דומה לזו של ה-BEGAN, למרות שנראה שאפילו היה פחות יציב בשיפור של G. בניסיון זה ניסינו לאפשר עצירה מוקדמת אחרי יותר אפוקים, אך לא נראה שזה עוזר לנו.

המימוש של שני המודלים נמצא במודול `began.py`.

ביצועי המודלים:

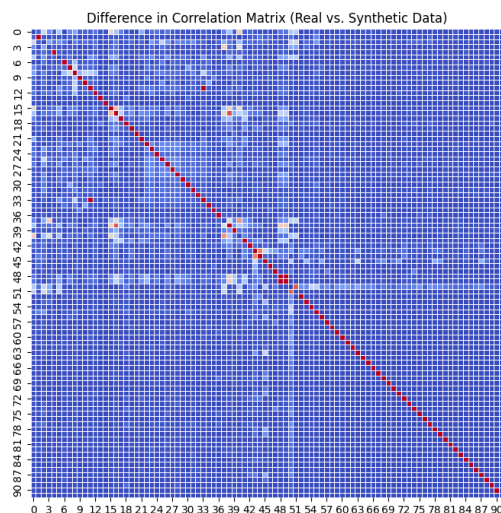
בדומה לסט המודלים הקודם, רצינו לבדוק איך מתפלגים הפיצ'רים במידע המג'ונרט היוצא ממודל זה:



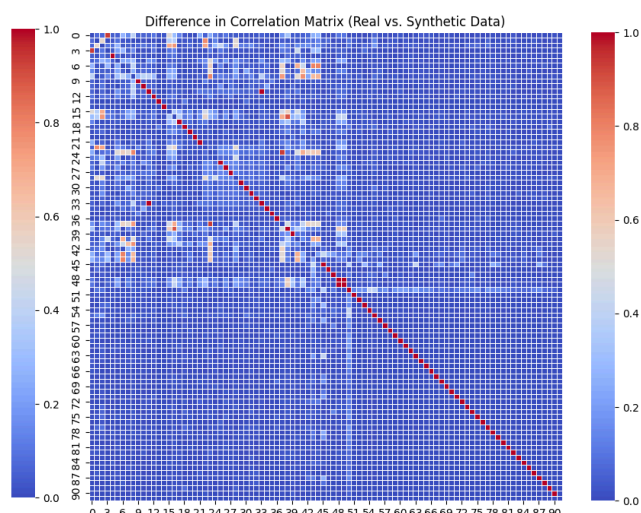
ההתפלגות של מודל ה-BEGAN נראית לא טוב, עם שוני דרסטי בהתפלגות בין הפיצ'רים האמיתיים לבין המג'ונרטים, שגם נראה שממוקדים על ערכי קצה באופן יחסי. ההתפלגות של מודל ה-cBEGAN נראית די דומה, עם פערים מאוד ברורים בייצוג של פיצ'רים שונים, בעיקר הנומריים.

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

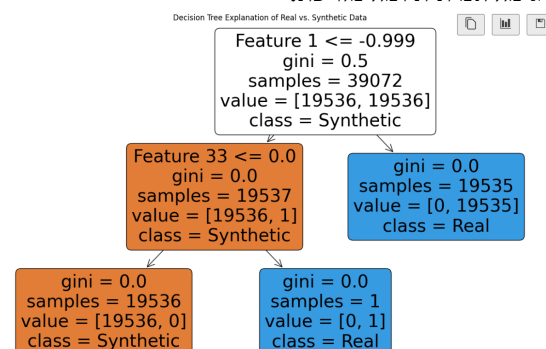
BEGAN:



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, ובבדיקת העץ שלהם ראינו שהפיצ'רים הנומריים מסגירים לגמרי מה דגימה אמיתית ומה מזויפת:



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