עבודת גמר 5 יח"ל

התמחות DL&ML

תחזית מחיר הביטקוין ע"פ תמונת מצב השוק

שם התלמיד: איתמר זם **שם בית הספר:** מקיף י"א ראשונים

תעודת זהות: 331651976 **תאריך הבחינה:** 27.6.2025

מנחה: דינה קראוס **שם החלופה:** למידת מכונה בהנדסת תוכנה

תוכן עניינים

מבוא

בינה מלאכותית היא תחום מרכזי במדעי המחשב שמטרתו לדמות דפוסי חשיבה ופתרון בעיות כמו בני אדם. בתוך התחום הזה יש שני נושאים עיקריים שהם למידת מכונה(Machine Learning) ולמידה עמוקה(Deep Learning), נושאים שבהם משתמשים באלגוריתמים מורכבים כדי לאפשר למחשב ללמוד מנתונים ולבצע חיזויים לפי מה שלמד.

הפרוייקט שלי עוסק בחיזוי התנודות הבאות במחיר הביטקוין בהתבסס על תמונות של גרפי השוק שאני יוצר בעצמי בעזרת data מAPI של חברות מסחר כמו Binanace. חיזוי מחירים של מטבעות קריפטו היא משימה מורכבת כי יש הרבה גורמים שמשפיעים על המחיר, כמו אינפלציה, ריבית, מצב הכלכלה, רגשות המשקיעים ועוד...

בפרוייקט הזה החלטתי להתרכז רק בתנודתיות הטבעית של המטבע מהסיבה ששאר הגורמים תלויים בדברים כמו רגשות ופוליטיקה שאותם יהיה מסובך מאוד לחזות.

הפרוייקט מיועד בעיקר למשקיעים וסוחרים שמתחילים בדרכם ורוצים לקבל הכוונה בחיזוי תנועות השוק וזאת בכדי לשפר את יכולות המסחר שלהם ובכדי להגן עליהם מלבצע טעויות ככל הניתן.

המערכת תנסה לנבא אם מחיר הביטקוין יעלה או ירד ביום הקרוב לפי ניתוח של תמונות הגרף של השוק.

כחלק מהפיתוח הבנתי שעליי להשתמש במודלים מסובכים יותר מ cnn פשוט ולכן אני משתמש במודל מסוג RNN שנקרא LSTM בנוסף למודל CNN כחלק מארכיטקטורת המודל שלי.

בחרתי בנושא הזה מהסיבה שאני בעצמי מתעניין ומתעסק בעולם הקריפטו וגם אני מתקשה לחזות את תנודתיות שוק הביטקוין. מהסיבה הזאת רציתי לבנות כלי שיעזור לי ולמשקיעים אחרים לעשות זאת. כשחיפשתי מידע ראיתי שיש מעט פרויקטים שחוזים את תנועות הביטקוין דרך תמונות ורובם משתמשים בנתונים גולמיים ולא בתמונות. כלומר הפרוייקט שלי ייחודי גם בגלל העובדה שאין הרבה מתחרים שניתן למצוא באינטרנט וכל המתחרים האלו פועלים על פי מידע גולמי והמודל שלי עובד עם תמונות מה שהופך אותו ליותר אינטראקטיבי.

בתהליך העבודה יצרתי אלפי תמונות של גרפים, חקרתי מודלים שונים של למידה עמוקה, התגברתי על מספר מכשולים וניסיתי להבין איך לחבר בין התמונות השונות כדי לקבל תחזית מדויקת ככל הניתן.

לטובת פרוייקט הגמר התרכזתי בלימוד המודל בעניין התנודתיות הטבעית של השוק אך בהמשך אני מתכנן להוסיף מודלים שיחזו גם גורמים נוספים המשפיעים על השוק על מנת לקבל תמונת מצב מדויקת יותר ולשפר את דיוק המודל.

לסיכום אני מקווה ללמוד יותר על עולם הבינה המלאכותית דרך הפרוייקט ושהוא יהיה כלי שימושי למשקיעים בשוק הביטקוין.

מבנה \ ארכיטקטורת הפרוייקט

שלב איסוף הכנה וניתוח הנתונים

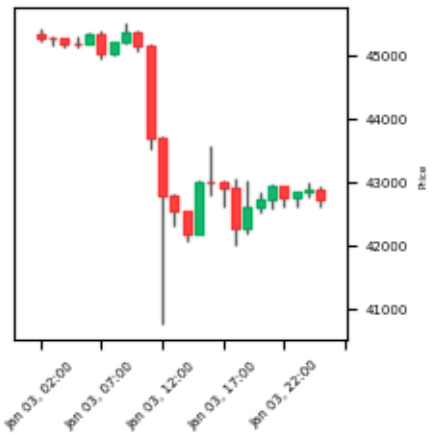
איסוף הנתונים הייתה בעיה עבורי בהתחלה. מכיוון שהפרויקט משתמש בתמונה כדי לבצע את החיזוי בתנודתיות הבאה הייתי צריך קודם כל להבין איך אני מחלץ מספר גדול כל כך של תמונות(20-30 אלף) מגרף הביטקוין.

כדי לעשות זאת עמדו בפני שני אופציות עיקריות:

1. לצלם את הגרפים בעצמי מה שאינו אפקטיבי בזמן ועלול לקחת זמן רב.
2. לבנות בוט שמצלם את הגרפים בעצמו למען הפרוייקט מה שהיה עלול לסכן אותי בחסימה מהפלטפורמה.

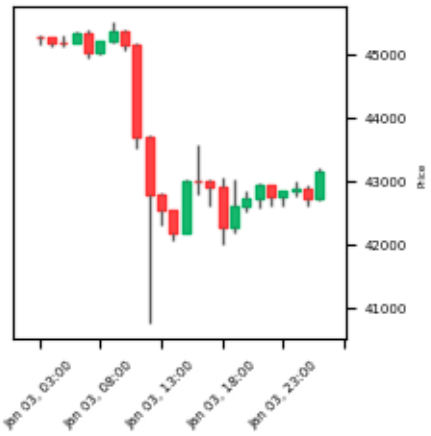
לאחר מחשבה ארוכה הגעתי לרעיון שבו אני יוצר את תמונות הגרפים בעצמי דרך ספריות ו-API של פלטפורמות מסחר אשר מכילות data על כל המטבעות והמניות שנסחרים בהם לפי זמנים ולשם כך מצאתי פלטפורמה אשר מציעה את אותו ה data שאני זקוק לו בחינם. אותן הספריות וה-API נתנו לי לגשת ל data מתחילת 2017 ועד מרץ 2025. ובכדי ליצור את הגרפים לקחתי את ה- data שאני זקוק לו בכדי ליצור גרפי נרות ונעזרתי ב-matplotlib.pyplot בכדי ליצור גרפי נרות שיראו כמו בפלטפורמת מסחר.

הגרפים שלי הם גרפים באורכים של יום שמתחלקים לנרות באורך של שעה. ונראים כך:



אורך הגרפים הוחלט כך שאוכל לתפוס כמה שיותר data בשביל דיוק המודל. ובכדי לייעל את זה עוד יותר דאגתי לכך שהקפיצות בין גרף לגרף(כלומר בין שורת data לשורה אחרת במודל) יהיה מרחק של שעה.

לדוגמא, זהו הגרף שבא אחרי הגרף הקודם שהצגתי:



ניתן לראות ששעת ההתחלה היא שעה אחת אחרי וכך גם כל שאר השעות על השנתות בציר הזמן.

חשוב לציין שגודל התמונה הוא 224 X 224 וזאת בכדי לאפשר למודל לרוץ מהר יותר ובכל זאת לשמור על איכות הגרף.

בתחילת הדרך עבדתי עם 3 אופציות לקיטלוג הגרפים ["up", "down", "static"] שאותן הצמדתי לכל גרף לפי האם המחיר שבו התנודה שבאה אחרי הגרף (שנבחרה להיות באורך 24 שעות ומתחילה בשעה שבאה ישר אחרי סיום הגרף מטעמי התאמה לאורך הdataset ומניעת בזבוז data) התחילה, גדול או קטן מהמחיר שבו היא סיימה (כלומר 24 שעות לאחר מכן) ובדקתי בנוסף לכך האם השינוי בין שני הנקודות קטן (תחילת התנודה שאחרי הגרף וסופה(כלומר 24 שעות לאחר מכן)) מ-500$ או גדול ממנו וזאת בכדי לבדוק את המצב הסטטי על פי התחום שהגדרתי לו שהוא 500$ של שינוי לכל צד.

בהמשך הבנתי שבגלל שהגרף מתאים את עצמו כך שיוכלו לראות טוב את התנועה כלומר גם תנועה שהיא סטטית(יורדת או עולה במקצת) תראה כמו ירידה או עלייה רגילה מבחינת המודל, תיווצר שגיאה בחיזויי המודל ולכן הפסקתי בחיזוי המצבים הסטטים. מה שגם בסך הכל הגיוני כי אפשר להרוויח מכל תנועה לא משנה כמה קטנה היא תהיה.

כלומר הclasses השונים הם ["up", "down"] וכל גרף עבר את התנאי הבא בכדי לקבוע לאיזה class לשייך את הגרף:

```
classification = "up" if price_change > 0 else "down"
```

כאשר Price change הוגדר כמחיר בתום 24 השעות (שבאות לאחר הגרף שיצרנו) פחות המחיר בתחילת 24 השעות(שבאות לאחר הגרף שיצרנו) כלומר השינוי במחיר במשך 24 השעות שבאות לאחר הגרף.

הקוד של בנייה ואיסוף הנתונים נראה כך:

```
def initialize_client(api_key=None, api_secret=None):
    return Client(api_key, api_secret)
```

הפונקציה Initialize_client יוצרת ומחזירה אובייקט client לפי מפתח וסוד API שהם זוג קודים שמנפיקה מערכת עבור כל משתמש על מנת שיבצע אליה(למערכת) חיבור ישיר, פרטי ומאובטח שדרכו ניתן להשתמש בפעולות שקיימות במערכת.

בפונקציה זו נשתמש בשביל לקרוא לAPI של Binance (שהיא פלטפורמת מסחר) בהמשך.

```
# Fetch hourly Bitcoin data from Binance
def fetch_hourly_data(client, symbol, interval, start_date, end_date):
    all_data = []
    while True:
        klines = client.get_historical_klines(symbol, interval, start_date, end_date, limit=1000)
        if not klines:
            break
        all_data.extend(klines)
        last_timestamp = klines[-1][0]
        start_date = datetime.utcfromtimestamp(last_timestamp / 1000) + timedelta(milliseconds=1)
        start_date = start_date.strftime('%Y-%m-%d %H:%M:%S')
        if len(all_data) >= 60000:  # Limit data
            break
    columns = ['timestamp', 'open', 'high', 'low', 'close', 'volume']
    data = pd.DataFrame(all_data, columns=columns + [None] * 6)  # Extra columns ignored
    data = data.iloc[:, :6]  # Keep relevant columns
    data['timestamp'] = pd.to_datetime(data['timestamp'], unit='ms')
    data = data[['timestamp', 'open', 'high', 'low', 'close', 'volume']]
    data[['open', 'high', 'low', 'close', 'volume']] = data[['open', 'high', 'low', 'close', 'volume']].astype(float)
    return data
```

הפונקציה fetch_hourly_data מושכת מה־ APIשל Binance נתונים עבור מטבע מסוים(שנקבע אותו להיות הביטקוין בזימון של הפונקציות) בטווח תאריכים נתון, ומחזירה אותם כטבלת DataFrame מסודרת של pandas .

```
# Ensure folders exist
def ensure_folders(base_folder, categories):
    for category in categories:
        input_folder = os.path.join(base_folder, category, "input")
        output_folder = os.path.join(base_folder, category, "output")
        os.makedirs(input_folder, exist_ok=True)
        os.makedirs(output_folder, exist_ok=True)
```

הפונקציה ensure_folders יוצרת תיקיות input (בהן השתמשתי לאורך הפרויקט) ותיקיותoutput (בהן שמורות תמונות של גרף הביטקוין ב־24 השעות שלאחר כל גרף input למרות שבפועל

השתמשתי בנתונים עצמם ולא בתמונות כדי לקבוע אם הייתה עלייה או ירידה במחיר ב-24 השעות שלאחר כל גרף Input) עבור כל קטגוריה (label) שברשימת הקטגוריות שנעביר לה, בתוך תיקיית בסיס שנבחר.
אם התיקיות כבר קיימות, היא לא מוחקת או משנה אותן.

```
# Balance categories by trimming excess files
def balance_categories(base_folder, categories):
    file_counts = {
        category: len(os.listdir(os.path.join(base_folder, category, "input")))
        for category in categories
    }
    min_count = min(file_counts.values())

    for category in categories:
        for subfolder in ["input", "output"]:
            folder_path = os.path.join(base_folder, category, subfolder)
            files = sorted(os.listdir(folder_path))
            for file_to_remove in files[min_count:]:
                os.remove(os.path.join(folder_path, file_to_remove))
```

בכדי להימנע ממצב של overfitting ו- imbalancing הייתי צריך לדאוג שכמות ה data שיש לי המקוטלגות כירידות שווה לכמות ה data שיש לי המקוטלגות כעליות, ואת זה עשיתי דרך פונקציה שקראתי לה balance_categories שתפקידה להוריד שורות של data מהclass שבו יש יותר data כך שיהיה מספר שווה בין ה classes .

```python
# Split data into train, validation, and test sets
def split_data(base_folder, categories, train_ratio=0.8, val_ratio=0.1):
    test_ratio = 1 - train_ratio - val_ratio

    for category in categories:
        input_folder = os.path.join(base_folder, category, "input")
        output_folder = os.path.join(base_folder, category, "output")

        files = sorted(os.listdir(input_folder))
        random.shuffle(files)
        total_files = len(files)

        train_split = int(total_files * train_ratio)
        val_split = int(total_files * (train_ratio + val_ratio))

        train_input_folder = os.path.join(base_folder, "train", category, "input")
        train_output_folder = os.path.join(base_folder, "train", category, "output")
        val_input_folder = os.path.join(base_folder, "validation", category, "input")
        val_output_folder = os.path.join(base_folder, "validation", category, "output")
        test_input_folder = os.path.join(base_folder, "test", category, "input")
        test_output_folder = os.path.join(base_folder, "test", category, "output")

        os.makedirs(train_input_folder, exist_ok=True)
        os.makedirs(train_output_folder, exist_ok=True)
        os.makedirs(val_input_folder, exist_ok=True)
        os.makedirs(val_output_folder, exist_ok=True)
        os.makedirs(test_input_folder, exist_ok=True)
        os.makedirs(test_output_folder, exist_ok=True)

        for i, file in enumerate(files):
            src_input = os.path.join(input_folder, file)
            src_output = os.path.join(output_folder, file.replace("input", "output"))

            if i < train_split:
                dest_input = os.path.join(train_input_folder, file)
                dest_output = os.path.join(train_output_folder, file.replace("input", "output"))
            elif i < val_split:
                dest_input = os.path.join(val_input_folder, file)
                dest_output = os.path.join(val_output_folder, file.replace("input", "output"))
            else:
                dest_input = os.path.join(test_input_folder, file)
                dest_output = os.path.join(test_output_folder, file.replace("input", "output"))

            shutil.move(src_input, dest_input)
            shutil.move(src_output, dest_output)
```

בכדי לשמור על סדר וארגון ה dataset החלטתי לפצל את ה dataset כבר בתוך תיקיית הפרוייקט לתיקיות של ["train", "test", "validation"] ועשיתי זאת בעזרת פונקציה שקראתי לה split_data אשר יוצרת את התיקיות החדשות במידת הצורך ומחלקת את ה dataset אליהן לפי יחס חלוקה נתון.

```python
def generate_and_categorize(data, base_folder, window_size=24, batch_size=500, resolution=(224, 224)):
    ensure_folders(base_folder, ["up", "down"])

    num_images = len(data) - (2 * window_size) + 1
    total_batches = (num_images + batch_size - 1) // batch_size

    for batch_idx in range(total_batches):
        start_idx = batch_idx * batch_size
        end_idx = min(start_idx + batch_size, num_images)

        print(f"Processing batch {batch_idx + 1}/{total_batches} ({start_idx}-{end_idx})...")

        for i in range(start_idx, end_idx):
            input_subset = data.iloc[i:i + window_size]
            input_subset.set_index('timestamp', inplace=True)

            output_subset = data.iloc[i + window_size:i + 2 * window_size]
            output_subset.set_index('timestamp', inplace=True)

            open_price = output_subset.iloc[0]['open']
            close_price = output_subset.iloc[-1]['close']
            price_change = close_price - open_price

            classification = "up" if price_change > 0 else "down"

            input_folder = os.path.join(base_folder, classification, "input")
            output_folder = os.path.join(base_folder, classification, "output")

            input_file_name = os.path.join(input_folder, f"input_graph_{i + 1:05d}.png")
            fig, ax = plt.subplots(figsize=(resolution[0] / 100, resolution[1] / 100))
            mpf.plot(input_subset, type='candle', ax=ax, style="yahoo", volume=False)
            ax.set_ylabel("Price", fontsize=4)
            ax.tick_params(axis='both', labelsize=5)
            plt.savefig(input_file_name, dpi=100, bbox_inches='tight')
            plt.close(fig)

            output_file_name = os.path.join(output_folder, f"output_graph_{i + 1:05d}.png")
            fig, ax = plt.subplots(figsize=(resolution[0] / 100, resolution[1] / 100))
            mpf.plot(output_subset, type='candle', ax=ax, style="yahoo", volume=False)
            ax.set_ylabel("Price", fontsize=4)
            ax.tick_params(axis='both', labelsize=5)
            plt.savefig(output_file_name, dpi=100, bbox_inches='tight')
            plt.close(fig)

        print(f"Batch {batch_idx + 1}/{total_batches} completed.")

    print(f"Generated and categorized {num_images} candlestick chart pairs.")
```

הפונקציה generate_and_categorize יוצרת זוגות של גרפים מסוג candlestick charts מפרקי זמן עוקבים מנתוני הביטקוין ומסווגת כל זוג לפי כיוון השינוי במחיר של פרק הזמן הבא (24 שעות הבאות) העוקב לגרף ה-input המדובר ושומרת את גרפי ה Input וה-output כקבצי תמונה בעלי אותו המספר בתיקיות מתאימות.

שוב אציין שבתיקיית ה- output לא השתמשתי בסוף.

```python
def main_data_creation():
    # API keys (use None if not needed)
    api_key = None
    api_secret = None

    # Initialize Binance client
    client = initialize_client(api_key, api_secret)

    # Define parameters for data fetching
    symbol = "BTCUSDT"
    interval = Client.KLINE_INTERVAL_1HOUR
    start_date = "2017-01-03"
    end_date = "2025-03-02"

    print("Fetching data from Binance...")
    data = fetch_hourly_data(client, symbol, interval, start_date, end_date)

    # Define base folder for dataset storage
    base_folder = "candlestick_dataset"

    print("Generating and categorizing candlestick images...")
    generate_and_categorize(data, base_folder)

    print("Balancing categories...")
    balance_categories(base_folder, ["up", "down"])

    print("Splitting data into train, validation, and test sets...")
    split_data(base_folder, ["up", "down"])

    print("Process completed successfully!")
```

זו היא פונקציית הזימון שבה השתמשתי כדי לקרוא לפונקציות וליצור את הגרפים והתיקיות.

לAPI של Binance התחברתי ללא מפתח וסוד.

הפונקציה מתחברת ל API של Binance ולוקחת ממנה את הנתונים שהיא זקוקה להם, יוצרת ומסווגת גרפי candlestick מאזנת את מספר הדוגמאות בכל קטגוריה ולבסוף מחלקת ושומרת את ה dataset בתיקיות לפי הקטגוריות(classes) המתאימים.

כך מתקבל מערך נתונים מוכן לשימוש במודלים של למידת מכונה לניתוח מגמות בשוק הביטקוין.

אך לאחר שיצרתי את ה-dataset הבנתי שעליי להשתמש רק בתיקיית ה- input ולכן פיצלתי את תיקיית ה dataset לשתי תיקיות שונות:

candlestick_dataset_input-

candlestick_dataset_output-

שמתוכן השתמשתי רק בתיקיית ה-Input לטובת הפרויקט וזאת עשיתי בעזרת קריאה לפונקציה הבאה בנפרד.

```python
def devide_dataset():

    # Original dataset path
    original_dataset = 'candlestick_dataset'
    input_dataset = 'candlestick_dataset_input'
    output_dataset = 'candlestick_dataset_output'

    # Dataset splits
    splits = ['train', 'validation', 'test']
    classes = ['up', 'down']

    # Create new directory structures
    for dataset in [input_dataset, output_dataset]:
        for split in splits:
            for class_name in classes:
                os.makedirs(os.path.join(dataset, split, class_name), exist_ok=True)

    # Move files into the new structure
    for split in splits:
        for class_name in classes:
            input_folder = os.path.join(original_dataset, split, class_name, 'input')
            output_folder = os.path.join(original_dataset, split, class_name, 'output')

            input_target = os.path.join(input_dataset, split, class_name)
            output_target = os.path.join(output_dataset, split, class_name)

            # Move input files
            if os.path.exists(input_folder):
                for file in os.listdir(input_folder):
                    shutil.move(os.path.join(input_folder, file), os.path.join(input_target, file))

            # Move output files
            if os.path.exists(output_folder):
                for file in os.listdir(output_folder):
                    shutil.move(os.path.join(output_folder, file), os.path.join(output_target, file))

    print('Dataset successfully split into input and output datasets.')
```

כך נראה מבנה ה dataset שלי:

candleStick_dataset_input

- **train**
  - **up**
    - img_001.png
    - img_002.png
    - ...
  - **down**
    - img_001.png
    - img_002.png
    - ...
- **validation**
  - **up**
    - img_001.png
    - img_002.png
    - ...
  - **down**
    - img_001.png
    - img_002.png
    - ...

- **test**
  - **up**
    - img_001.png
    - img_002.png
    - ...
  - **down**
    - img_001.png
    - img_002.png
    - ...

לאחר מכן ביצעתי נורמליזציה על הנתונים בזמן העלאת מערך הנתונים שלי לקוד המודל בעזרת הקריאה הזו. וחזרתי עליה גם לתיקיות המבחן והוולידציה.

נורמליזציה הוא תהליך שמשנה את טווחי הפיקסלים בתמונה מטווח של 0-255 לטווח שבין 0 ל-1 וזאת בכדי לשפר את הביצועים, למנוע בעיות חישוביות ולשמור על היציבות במודלים של למידת מכונה ולמידה עמוקה.

```
# Data loading
train_raw = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    image_size=(224, 224),
    batch_size=batch_size,
    seed=seed
    ).map(lambda x, y: (x/255, y)).map(preprocess).map(augment).unbatch()  # UNBATCH here
```

שלב בנייה ואימון המודל

המודלים הראשוניים שניסיתי

לצורך בניית המודל הייתי צריך קודם לכן לבחור ולנסות סוגי מודלים שונים בפרוייקט כמו: RNN, CNN, ViT.

לטובת ניסוי ותהייה התחלתי בלנסות לבנות מודל מסוג ViT .

הסבר קצר על מודל ViT

מודל ה- ViT הוא מודל למידת מכונה שמביא את שיטת הטרנספורמרים, שפותחה במקור לעיבוד טקסט לעולם של ניתוח תמונות. בניגו למודלי CNN שמחפשים תבניות מקומיות בתמונה, ViT מחלק את התמונות שנכנסות אליו לחלקים קטנים שנקראים patches , ממיר כל חתיכה לווקטור בתהליך שנקרא embedding ומעבד את כל הרצף הזה בעזרת מנגנון Self-Attention המצוי לרוב בטרנספורמרים ומאפשר למודל להבין קשרים בין כל חלקי התמונה.

לכל חתיכה מוסיפים גם מידע על מיקומה המקורי בתמונה (positional embedding) וזאת בכדי שהמודל ידע לשמור על המידע המקורי.

בסוף יש טוקן(יחידת מידע בסיסית שממנה מורכב הקלט של המודל, לדוגמה מילה במשפט) מיוחד שנקרא class token שמרכז את המידע כולו והוא עובר דרך שכבה המוציאה את הפלט של המודל.

כלומר מודל ViT מחזיק ביתרון של למידת קשרים "רחוקים" ומורכבים בתמונה, אבל זאת רק כאשר יש מאגר נתונים גדול.

```
[ ]  # Variables
     image_size= 224
     batch_size= 8  # Reduced for better memory management
     seed = 42
     channels = 3
     epochs =10
     optimizer= tf.keras.optimizers.Adam(learning_rate=0.0001)
```

```
#Load ViT mode
vit_url = "https://tfhub.dev/sayakpaul/vit_b16_fe/1"
vit_model = hub.KerasLayer(vit_url, trainable=False)


#model structure
inputs= layers.Input(shape=(image_size, image_size, channels))

x=layers.Rescaling(1./127.5, offset=-1)(inputs)

x =vit_model(x)  # ViT outputs a feature vector

x=layers.GlobalAveragePooling1D()(x)

x=layers.Dense(128, activation="relu")(x)
x=layers.Dropout(0.5)(x)
outputs= layers.Dense(1, activation="sigmoid")(x)  # Binary classification

model =models.Model(inputs=inputs, outputs=outputs)
```

במודל שלמעלה טענתי מודל ViT בעזרת kersas hub וקישור למודל ViT .

לאחר מכן יצרתי שכבת Input שאליה נכנסים הנתונים למודל.

לאחר מכן בניתי שכבת נורמליזציה שהופכת את ערכי הפיקסלים מ-0 עד 255 למינוס 1 עד 1 כדי להתאים לדרישות המודל.

לאחר מכן העברתי את הנתונים דרך מודל ה-ViT

ומיד אחרי זה השתמשתי בסוגי שכבות נוירונים מוכרים כמו Dense, Dropout ו-GlobalAveragePooling בכדי לפשט את הנתונים למנוע overfitting ולהוציא פלט בינארי.

אך לפני שבכלל הספקתי להצליח לבנות מודל מתפקד ראשוני(עקב בעיה של pre- processing) הבנתי שסוג המודל הזה אינו מתאים בגלל שאין לי מספיק data ומודל ViT דורש כמות data של מיליונים על גבי מיליונים. לאחר שהבנתי את זה התחלתי לחפש ולחקור מודלים שיתאימו למודל שלי.

לאחר החקירה הבנתי שעליי להשתמש במודל CNN+RNN בשביל שהמודל שלי יוכל ללמוד לפי קטעי זמן ולקשר בין פיסות מידע קודמות לעכשוויות.

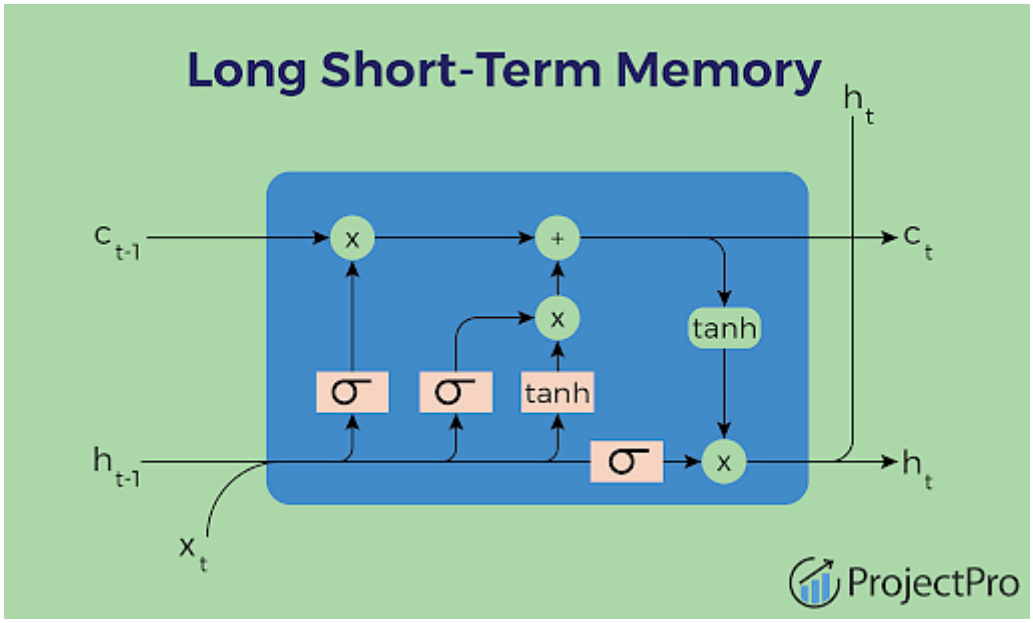
חקרתי קצת ולמדתי מה זה מודל RNN ואיך הוא משתלב במודל של CNN ונתקלתי בסוג מודל שנקרא LSTM שהוא מודל מסוג RNN ומה שמייחד את המודל הזה זה היכולת שלו ללמוד מסגמנטים של data בבת אחת ובכך יכול לתת למודל שלי את היכולת לקשר בין שורות של data וללמוד את הקשר פרק זמן לפרק זמן הקודם לו, דבר שעוזר בצורה משמעותית למודל ללמוד את תנודתיותו הטבעית של שוק הביטקוין.

איך עובד מודל RNN

מודל RNN (Recurrent Neural Network) הוא סוג מיוחד של רשת נוירונים שמותאם לעבודה עם רצפים וסדרות זמן. מה שמייחד את המודל זה שיש לו "זיכרון", כלומר, הוא שומר מידע מהשלבים הקודמים ברצף ומשתמש בו כדי להבין טוב יותר את החלק הנוכחי.

בכל שלב המודל מקבל את הקלט העכשווי וגם את המידע מהשלב הקודם ובכך יכול לזהות דפוסים לאורך זמן ולהבין את הקשרים בין חלקים שונים ברצף.

איך עובד מודל LSTM



מודל LSTM (Long Short Term Memory) הוא סוג ספציפי של RNN שמיועד להתמודדויות עם מידע שמסודר ברצף, ולזכור מידע לאורך זמן רב יותר מרשתות רגילות.

המודל בנוי מיחידות LSTM וכל אחד מהן כוללת 3 שערים: שער קלט, שער שכחה ושער פלט. כאשר כל שער הוא מנגנון שמחליט באמצעות חישוב מתמטי הפולט ערך אשר מחליט איזה מידע להכניס, איזה מידע לשכוח ואיזה מידע להעביר הלאה בהתבסס על הערך המתקבל מהחישוב המתמטי והקלט הנוכחי .

כל מידע שעובר בתוך המודל בצורת רצף (sequence) עובר דרך שלושת השערים:

- input gate- : **שער הקלט** בודק כמה מהמידע החדש כדאי להכניס לזיכרון של התא.

Input Gate

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

החישוב הראשון משתמש בפונקציית sigmoid בכדי להחזיר ערך בין 0 ל-1 על כפל של מטריצה המכילה את משקולות שער הקלט בוקטור שמורכב מהקלט הנוכחי ומהזיכרון הקיים מהשלב(החלק הקודם ב-sequence , הסבר על sequence בהמשך) הקודם בתוספת הbias .

הערך שמתקבל הוא הערך שמייצג כמה המידע החדש חשוב או רלוונטי עבור הזיכרון של התא בשלב הזה. ככל שהערך קרוב יותר ל-1 כך המודל פותח את עצמו יותר למידע חדש שיתפוס אצלו מקום בזיכרון וכשהוא קרוב ל-0 בדיוק ההפך.

החישוב השני בשער מחשב את המידע החדש שהמודל מציע להכניס לזיכרון. כמו בחישוב הראשון גם כאן לוקח המודל את הקלט הנוכחי ואת הזיכרון הקיים מהשלב הקודם ומחבר אותם לוקטור שאותו הוא מכפיל במטריצת משקולות שמותאמת לערך החדש המוצע לזיכרון. לתוצאה זו הוא מוסיף bias ואז מעביר את התוצאה דרך פונקציה tanhשמחזירה ערך בין 1 למינוס 1.

ערך חיובי מחזק תכונה מסוימת בזיכרון, ערך שלילי מפחית או מבטל אותה, וערך קרוב לאפס לא מבצע שינוי משמעותי.

- Forget gate: **שער השכחה** מחליט כמה מידע מהשלב הקודם יישמר או יימחק מהזכרון.

Forget Gate

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

החישוב של שער השכחה זהה לחישוב של שער הקלט רק שמטריצת המשקולות וה bias מתאימים לשער השכחה באופן ספציפי. והוא מחזיר ערך בין 0 ל-1. כאשר אם הערך קרוב ל-1 המידע יישמר בזיכרון וכשקרוב ל-0 הפוך מכך.

- Output gate : שער הקלט מחליט איזה מידע יעבור ויצא בתור הקלט של השלב הנוכחי

Output Gate

$$h_t = o_t \odot \tanh(c_t)$$

לאחר שהחלקים מהזיכרון עוברים סינון בעזרת משוואה דומה למשוואות הקודמות רק עם מטריצת משקלים ו-bias שמתאימים לשער הקלט באופן ספציפי, כאשר עבור כל חלק בזיכרון מקבל ערך בין 0 ל-1. קרוב ל-1 אומר שהמידע יעבור אחרת המידע לא יעבור ויוסתר מהפלט.

אותם החלקים שעברו את הסינון יעברו דרך פונקציית tanh וכך יצאו כפלט.

דרך מנגנון זה הקיים בכל יחידת LSTM המודל יכול לסנן את המידע החשוב ולשמור אותו לאורך זמן ובנוסף לא לייחס חשיבות לחלקים הפחות חשובים. בזכות כך מאפשר למודל להתמודד עם בעיות טיפוסיות למודלי RNN די בקלות.

<u>הדגמה ויזואלית של איך עובר המידע דרך המודל:</u>

<u>Input sequence</u>

3 gates decision for each part of the sequence

|

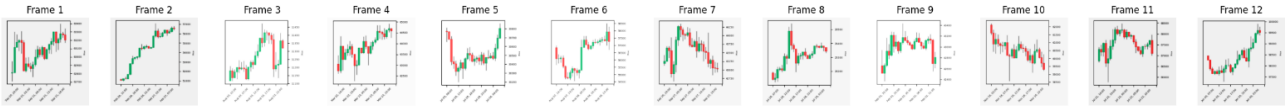
|

Learning through time

sequences הם אוסף של פיסות מידע (שלבים לפי ההגדרה שהשתמשתי בה קודם) שמחוברות אחת לשנייה באורך של פיסות מידע קבוע מראש, ונכנסות למודל בתור קבוצה ומנותחים ביחד.

הדגמה ויזואלית:

מה שניתן לראות בהדגמה שלמטה, הוא בעצם sequence אחד שלם באורך של 12- כלומר מורכב מ-12 פיסות מידע שנלקחו ממאגר ה-train .



כדי לשלב את היכולת של מודל ה- LSTM עם יכולת מציאת פיצ'רים ודפוסים בתמונות, אני משתמש במודל שמשלב CNN ו- RNN(LSTM) . ובעצם מה שאני עושה בכך זה יוצר מודל שלומד לא רק על פיצ'רים בתמונה אלא גם לומד לפי קשר לתמונות קודמות ורצפים(sequences).

מודל זה מחלץ תכונות ופיצ'רים מהתמונות דרך שימוש ב-CNN כך שכל תמונה הופכת לייצוג מספרי(וקטור) של המאפיינים החשובים שבה. לאחר מכן המודל בונה את הרצף מחדש לפי סדר כרונולוגי ומשם המידע נכנס לשכבות הLSTM שמנתח את הקשרים והדפוסים בין הוקטורים שקיבל לאורך כל ה-sequence ולבסוף עובר דרך שכבות אשר מכוונות לפלט הרצוי.

זהו מודל ה-LSTM הראשוני שהשתמשתי בו:

```
# Setup
image_size_x = 224
image_size_y = 224
batch_size = 8
sequence_length = 8
seed = 42
channels = 3
epochs = 10
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9, nesterov= True)
```

```
#CNN & LSTM model
model = models.Sequential([
    layers.Input(shape=(sequence_length, image_size_x, image_size_y, channels)),

    # CNN feature extractor for each frame
    layers.TimeDistributed(layers.Conv2D(32, (3,3), activation="relu")),
    layers.TimeDistributed(layers.Conv2D(32, (3,3), activation="relu")),
    layers.TimeDistributed(layers.BatchNormalization()),
    layers.TimeDistributed(layers.MaxPooling2D((2,2))),
    layers.TimeDistributed(layers.Dropout(0.2)),

    layers.TimeDistributed(layers.Conv2D(64, (3,3), activation="relu")),
    layers.TimeDistributed(layers.Conv2D(64, (3,3), activation="relu")),
    layers.TimeDistributed(layers.BatchNormalization()),
    layers.TimeDistributed(layers.MaxPooling2D((2,2))),
    layers.TimeDistributed(layers.Dropout(0.2)),

    layers.TimeDistributed(layers.Conv2D(128, (3,3), activation="relu")),
    layers.TimeDistributed(layers.Conv2D(128, (3,3), activation="relu")),
    layers.TimeDistributed(layers.BatchNormalization()),
    layers.TimeDistributed(layers.MaxPooling2D((2,2))),
    layers.TimeDistributed(layers.Dropout(0.2)),


    layers.TimeDistributed(layers.GlobalAveragePooling2D()),

    # LSTM for temporal dependencies
    layers.LSTM(128, return_sequences=True),
    layers.LSTM(64, return_sequences=False),

    # Fully connected
    layers.Dense(64, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(1, activation="sigmoid")  # Binary classification
    ])
```

במהלך פיתוח במודל חוויתי בעיות של זיכרון RAM בפרויקט והתקשיתי מאוד להריץ את הקוד ולכן ניסיתי להתפשר בגרסה אחרת של מודל RNN ששמו GRU. מודל זה פועל בעיקרון דומה לLSTM אך מהיר יותר, פשוט יותר ותופס פחות זיכרון.

<u>איך עובד מודל GRU</u>

לעומת מודל LSTM , מודל GRUמורכב רק מ-2 שערים:

-שער עדכון שמהווה שילוב של שער הקלט והשכחה ושער זה מחליט אם לשמור מידע קודם או לעדכן אותו ומחליט גם על הכמות לפי הקרבה ל-0 או ל-1

-שער איפוס שקובע כמה מידע מה- Sequence הקודם יישכח וכמה יישאר לשלב הבא

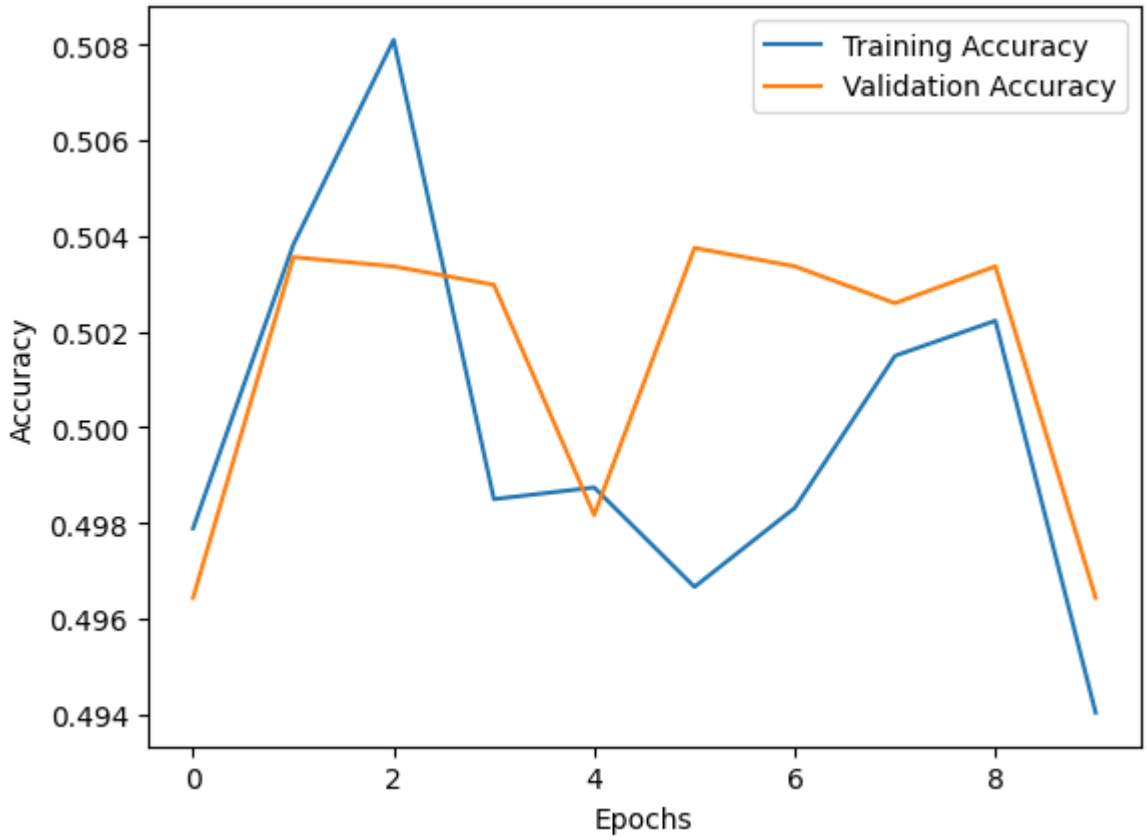
מודל ה-GRU שלי היה נראה כך:

```
#Hyperparameters
image_size_x= 224
image_size_y= 224
batch_size= 16
sequence_length =8
seed= 42
channels= 3
epochs= 50
optimiser=tf.keras.optimizers.Adam(learning_rate=0.0001)
```

```
#CNN & RNN(GRU) model
model = models.Sequential([
    layers.Input(shape=(sequence_length, image_size_x, image_size_y, channels)),

    # CNN feature extractor for each frame
    layers.TimeDistributed(layers.Conv2D(32, (3,3), activation="relu")),
    layers.TimeDistributed(layers.Conv2D(32, (3,3), activation="relu")),
    layers.TimeDistributed(layers.BatchNormalization()),
    layers.TimeDistributed(layers.MaxPooling2D((2,2))),
    layers.TimeDistributed(layers.Dropout(0.2)),
    layers.TimeDistributed(layers.Conv2D(64, (3,3), activation="relu")),
    layers.TimeDistributed(layers.Conv2D(64, (3,3), activation="relu")),
    layers.TimeDistributed(layers.BatchNormalization()),
    layers.TimeDistributed(layers.MaxPooling2D((2,2))),
    layers.TimeDistributed(layers.Dropout(0.2))
    layers.TimeDistributed(layers.Conv2D(128, (3,3), activation="relu")),
    layers.TimeDistributed(layers.Conv2D(128, (3,3), activation="relu")),
    layers.TimeDistributed(layers.BatchNormalization()),
    layers.TimeDistributed(layers.MaxPooling2D((2,2))),
    layers.TimeDistributed(layers.Dropout(0.2)),
    layers.TimeDistributed(layers.GlobalAveragePooling2D()),

    #GRU
    layers.GRU(128, return_sequences=True),
    layers.GRU(64, return_sequences=False),

    #CNN
    layers.Dense(128, activation= "relu"),
    layers.Dropout(0.3),
    layers.Dense(2, activation= "sigmoid")
    ])
```

והתוצאות שלו היו:

accuracy: 0.4921



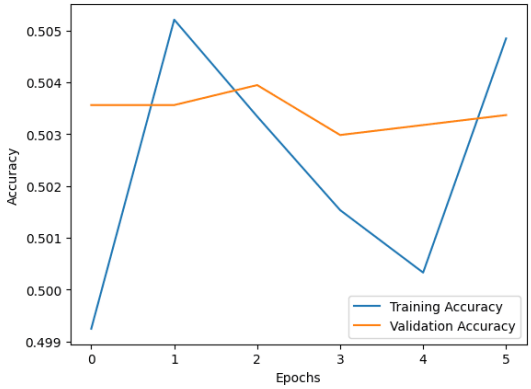
דרך הגרף ניתן לשים לב שנוצר underfitting . underfitting זה מצב שבו המודל לא מצליח ללמוד כמו שצריך מהנתונים מה שמוביל לביצועים נחותים בנתוני ה- train ובנתוני ה-validation .

ניתן לשים לב לתופעה דרך התכנסות הגרפים של דיוק ה-train ודיוק ה- validation והעובדה שלאורך ה- epochs הדיוק שלי שני העקומות לא גדל.

מודל ה- GRU ביצע פחות טוב ממודל ה-LSTM והצלחתי לפתור את בעיית הזיכרון ב-RAM שבגללה החלפתי למודל GRU ולכן חזרתי למודל -LSTM .

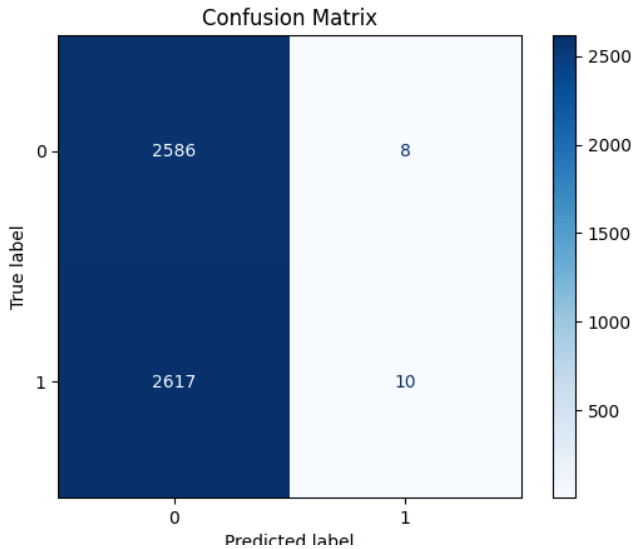
לאחר הרצה חוזרת של מודל ה-LSTM ביצועי המודל עדיין לא היו טובים.

ה accuracy היה: 0.5076



דרך הגרף ניתן לשים לב שוב ל- underfitting שמתבטאת בכך שהדיוק של שני עקומות לא משתפרים לאורך הזמן ונשארים מאוד סטטיים.

```
Classification Report:
              precision    recall  f1-score   support

           0     0.4970    0.9969    0.6633      2594
           1     0.5556    0.0038    0.0076      2627

    accuracy                         0.4972      5221
   macro avg     0.5263    0.5004    0.3354      5221
weighted avg     0.5265    0.4972    0.3334      5221
```



דרך ה- confusion matrix גיליתי שהמודל שלי פשוט מנחש את ה-class הספציפי down שסימולו הבינארי בגרף הוא 0. דרך הבנה זו הסקתי שיש לי imbalance ב- dataset האימון.

בכדי להבין את הבעיה לעומק יצרתי קוד שבודק כמה פיסות מידע יש לי תחת כל קטגוריה בתיקיית ה- train של הפרוייקט שלי.

```
#helped me check whether the data was even for both classes
down_files = os.listdir(train_dir_label_down)
up_files = os.listdir(train_dir_label_up)

down_count = len(down_files)
up_count = len(up_files)

print(f"Number of files in 'down' directory: {down_count}")
print(f"Number of files in 'up' directory: {up_count}")

if down_count > up_count:
    print("The 'down' directory has more files.")
elif up_count > down_count:
    print("The 'up' directory has more files.")
else:
    print("The 'down' and 'up' directories have the same number of files.")

label_list = list(train_labels.as_numpy_iterator())
print(Counter(label_list))
```

הקוד אישר לי את מה שחששתי וגיליתי שלמרות הפונקציה שקיימת בחלק של איסוף הנתונים אשר אמורה לוודא שכמות הנתונים תחת כל קטגוריה שווה, נוצר הבדק קטן של כ-10 פיסות מידע בין קטגוריית ה- down לקטגוריית ה- up .

הדבר הראשון שניסיתי לעשות הוא להשתמש בפעולה שנקראת class weight trick . הפעולה מריצה פונקציה שנקראת "Compute_class_weight " ובעצם מייצר משתנה מסוג dictionary שאותו מכניסים לשלב אימון המודל("fit") ובכך אתה מאפשר לו להוסיף משקל גבוה יותר למחלקות המיעוט ומשקל נמוך יותר לדוגמאות מהמחלקות הגדולות. כלומר המודל נענש יותר על טעות בחיזוי של מחלקת המיעוט ובכך המודל משקיע יותר "מאמץ" כדי לנחש אותם נכן בפעמים הבאות.

```
label_list = []

for _, label in train_raw:
    label_list.append(int(label.numpy()))

    label_array = np.array(label_list)

    # class weight decision
    class_weights = compute_class_weight(
        class_weight='balanced',
        classes=np.unique(label_array),
        y=label_array
        )

    class_weight_dict = dict(enumerate(class_weights))
    print("Class Weights:", class_weight_dict)
```

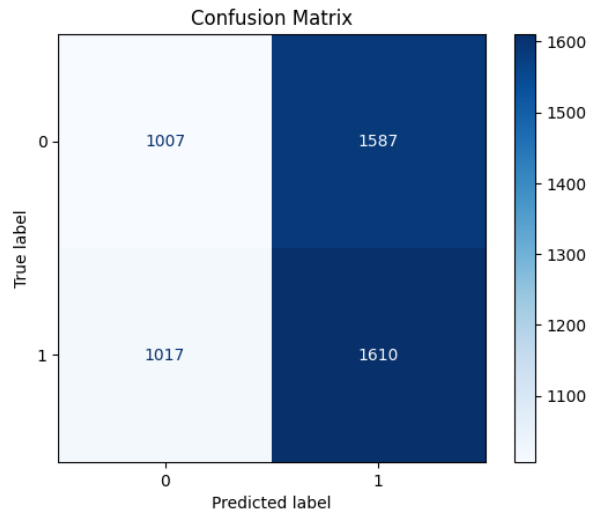
```
label_list = []

for _, label in train_raw:
    label_list.append(int(label.numpy()))

label_array = np.array(label_list)

# class weight decision
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(label_array),
    y=label_array
)

class_weight_dict = dict(enumerate(class_weights))
print("Class Weights:", class_weight_dict)
```

השיטה שיפרה במקצת את בעיית ה -imbalancing .

ולכן הפתרון הבא שניסיתי הוא undersampling כלומר הורדה במספר ה data במחלקה שבה נמצא יותר data . עשיתי זו בצורה פשוטה פשוט דרך העברת מספר קבצים (שבחרתי לפי ההבדל בין כמות ה- data בין שני המחלקות) לתיקייה אחרת כך שלא ייכנסו ללימוד המודל.

לאחר שביצעתי את הדבר הבעיה של ה- imbalancing נפתרה וה- confusion matrix הראה תוצאות הולמות:

```
Classification Report:
              precision    recall  f1-score   support

           0     0.4975    0.3882    0.4361      2594
           1     0.5036    0.6129    0.5529      2627

    accuracy                         0.5012      5221
   macro avg     0.5006    0.5005    0.4945      5221
weighted avg     0.5006    0.5012    0.4949      5221
```



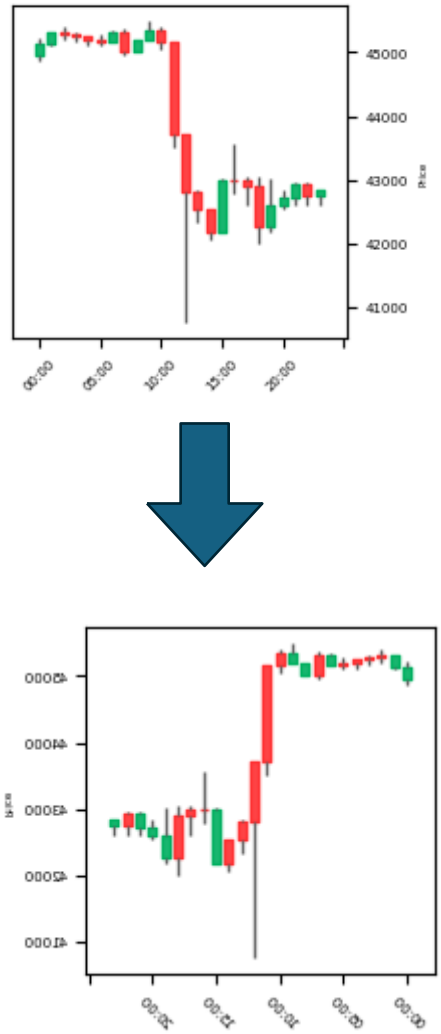
אך דיוק המודל ירד במקצת.

ובשלב הזה נשאר לי לנסות לשפר את דיוק המודל (דיוק המודל בכמעט כל המודלים שבניתי היה נמוך ובאזור ה50%) ומדדים אחרים כמו F1 score וה- Loss ולשם כך ניסיתי מספר דברים. הדבר הראשון שניסיתי היה להגדיל את ה train data דרך אוגמנטציה.

```
datagen = ImageDataGenerator(
    rotation_range= 25,
    channel_shift_range=10,
    width_shift_range =0.05,
    height_shift_range= 0.05,
    shear_range= 0.05,
    zoom_range= 0.05,
    horizontal_flip= False,
    brightness_range =[0.7, 1.1]
)


aug_generator = datagen.flow_from_directory(
    train_dir,
    target_size=(image_size_x, image_size_y),
    batch_size=1,
    class_mode='categorical',
    #save_to_dir=augmented_train_dir,
    save_prefix='aug',
    save_format='png'
)
```

אוגמנטציה היא פעולה שמעוותת (לפי תת פעולות שאתה מגדיר שנקראות transformers) תמונות קיימות ובכך יוצרות תמונות חדשות. בשביל האוגמנטציה בחרתי בתת פעולות שלא יפגעו בdataset , לדוגמא אם הייתי משתמש ב- horizontal_flip כיוון הגרף היה משתנה ובכך המודל היה לומד מ data שגוי.

הדגמה של פעולת horizontal_flip:



לאחר שביצעתי אוגמנטציה(הגדלתי את גודל הdataset) והרצתי את המודל מחדש גיליתי שאין שינוי של ממש בתוצאות המודל והבנתי שככל הנראה הבעיה לא נובעת מגודל ה- dataset שאספתי.

ובכדי לבדוק שהאוגמנטציה לא יוצר imblanace נוסף בין ה- classes ביצירת תמונות נוספות יצרתי פונקציה שבודקת את זה.

```python
print(f"Checking image counts in augmented_train_dir:")

augmented_down_dir = os.path.join(augmented_train_dir, 'down')
augmented_up_dir = os.path.join(augmented_train_dir, 'up')

#Ensure directories already exist in the folder
if not os.path.exists(augmented_down_dir):
    print(f"Warning: Directory '{augmented_down_dir}' not found.")
    augmented_down_count = 0
else:
    augmented_down_files = os.listdir(augmented_down_dir)
    augmented_down_count = len(augmented_down_files)

if not os.path.exists(augmented_up_dir):
    print(f"Warning: Directory '{augmented_up_dir}' not found.")
    augmented_up_count = 0
else:
    augmented_up_files = os.listdir(augmented_up_dir)
    augmented_up_count = len(augmented_up_files)

print(f"  Number of images in '{os.path.basename(augmented_down_dir)}' (augmented): {augmented_down_count}")
print(f"  Number of images in '{os.path.basename(augmented_up_dir)}' (augmented): {augmented_up_count}")

#cehcks if even
if augmented_down_count > 0 and augmented_down_count == augmented_up_count:
    print("  The augmented 'down' and 'up' directories have an even number of images.")
elif augmented_down_count > augmented_up_count:
     print("  The augmented 'down' directory has more images.")
elif augmented_up_count > augmented_down_count:
     print("  The augmented 'up' directory has more images.")
else:
     print("  Counts for augmented directories are not available or are zero.")
```

כל מה שנותר לי לעשות בשלב הזה הוא ניסוי ותהייה ב Hyper parameters ובארכיטקטורת המודל שלי.

הרצתי כל מיני ניסיונות והגעתי למודל סופי שעובד בצורה הכי טובה מכל שאר המודלים (שבאו לאחר תיקון ה- imbalacing של ה-data) מבחינת תוצאות מדדי ה- accuracy וה- loss .

המודל משלב pretrained model בשם Mobilenetv2

מטרת השימוש במודל מאומן מראש (Pretrained model) היא לסייע למודל שלי לזהות תכונות (features)בתמונות בקלות, תוך ניצול היכולות שפיתח במהלך האימון המקדים שלו על כמות עצומה של תמונות.
הודות ללימוד המוקדם הזה, המודל יודע לאתר ולהבין מאפיינים ויזואליים ברמה גבוהה, בצורה מדויקת וטובה בהרבה ממודל שמאומן רק על מערך נתונים קטן.

מה שמייחד את Mobilenetv2 ממודלים מאומנים מראש אחרים הוא שהוא משיג תוצאות טובות במיוחד עם פחות משאבים (כמו זיכרון וחישוב.

מודל סופי

ארכיטקטורת המודל הסופי שהגעתי אליו נראה כך:

```
#CNN & LSTM model with pretrained model
model = models.Sequential([
    layers.Input(shape=(sequence_length, image_size_x, image_size_y, channels)),

    layers.Lambda(
        apply_mobilenet_per_frame,
        output_shape=(sequence_length, mobilenet_feature_dim)
    ),

    layers.LSTM(
        256,
        return_sequences=True,
        activation="relu"
    ),
    layers.Bidirectional(layers.LSTM(128, return_sequences=True,activation="relu")),
    layers.Bidirectional(layers.LSTM(64)),

    layers.Dense(128, activation='relu'),
    layers.Dropout(0.1),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

תיאור המודל:

המודל הסופי של הפרוייקט שלי הוא מודל שמשלב מודל LSTM עם מודל CNN שמשלב למידה של פיצ'רים ודפוסים בתמונות עם למידה של דפוסי קטעי זמן ונעזר במודל מאומן מראש בשם Mobilenetv2 שעוזר למודל להוציא פיצ'רים מתמונות.

הסבר על שכבות המודל

- שכבת הקלט(Input Layer): שכבה זו אחראית על קביעת הצורה של המידע שייכנס למודל.

- שכבת מודל מאומן מראש(Mobilnetv2 Layer): שכבה זו מעבירה כל תמונה ברצף(sequence) דרך המודל המאומן מראש כדי שייחלץ מהמידע "תכונות חשובות" כדי שייכנסו למודל ה LSTM בתור וקטורים של התכונות החשובות ביותר בשביל המודל ללמוד. מודל זה הוא מסוג CNN.

- שכבת LSTM(LSTM Layer): שכבה זו היא שכבה של מודל LSTM ומה שהיא עושה זה לקרוא את ה-sequence של התמונות, תמונה אחרי תמונה זוכרת מידע רלוונטי מהעבר ופולטת רצף חדש (וזאת בגלל התכונה return_sequence=True) שמבין את הקשר הזמני בין התמונות ב-sequence.

- שכבה דו כיוונית(Bidirectional Layer): השכבה הזו מעבדת את הרצף בצורה דומה לשכבת LSTMרגילה רק שהיא מעבדת את התמונות בשני הכיוונים כדי לקבל הבנה

עמוקה יותר על התלות בזמן. שכבה זו מחזירה sequence (וזאת בגלל התכונה return_sequence=True) שמבין את התלות הדו כיוונית בזמן.

- שכבה צפופה (Dense Layer):שכבות אלו מחברות את כלל הנוירונים לנוירונים שנמצאים בשכבה הבאה ומכינה אותו עד שמקבלים את הפלט הסופי של המודל.

- שכבת השמטה(Dropout Layer): שכבה זו היא שכבת השמטה אשר משמיטה נוירונים על פי ערך שקבוע לה מראש ובכך עוזרת למנוע Overfitting.

- שכבת נורמליזציה באצווה(Batch Normalization Layer): שכבה זו אחראית לשמור על יציבות תהליך האימון ובנוסף לגרום לו להיות מהיר יותר.

היפר פרמטרים

ראשוניים:

-epochs: 10

-batch_size : 8

-optimizer : SGD

-learning_rate : 0.001

-sequence_length: 8

סופיים:

-epochs: 60

-batch_size : 32

-optimizer : Adam

-learning_rate : 0.000055

-sequence_length: 12

```
# Setup
image_size_x = 224
image_size_y = 224
batch_size = 32
sequence_length = 12
seed = 42
channels = 3
epochs = 60
optimizer= tf.keras.optimizers.Adam(learning_rate=0.0000055)
```

את ההיפר פרמטרים האלה בחרתי דרך ניסוי ותהייה.

sequence_length הוא המשתנה בו השתמשתי בכדי להגדיר את אורך ה-sequences הקבוע שהמודל יעבוד איתו. (ניתן לראות הסבר על sequence_length)

הסבר על פונקציית ה-Loss

פונקציית Loss היא מדד שמטרתו למדוד עד כמה תחזיות של המודל היו רחוקות מהתוצאות האמיתיות של הנתונים, לפי המדד הזה המודל מנסה לשפר את הדיוק שלו וזאת דרך צמצום ערך פונקציית ההפסד.

הפונקצייה שאני משתמש בה היא אנטרופיה צולבת בינארית(Binary Crossentropy).

```
loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
```

פונקציה זו משמשת בבעיות סיווגים בינאריים כלומר כאשר יש רק שני מחלקות שאותן ניתן לחזות.

ובעצם מה שהיא עושה זה לחשב את ההבדל בין ה- label האמיתי לבין מה שהמודל חוזה.

פונקציה זו מתאימה במיוחד למצבים שבהם הפלט של המודל הוא הסתברות (כלומר יש שכבת סיגמויד בסוף הרשת), כפי שמוגדר אצלי עם from_logits=False

הסבר על פונקציית ה-Optimization

תפקיד פונקציית האופטימיזציה הוא לשנות את משקלי המודל במהלך שלב האימון על מנת למזער את ערך פונקציית הloss . פונקציית האופטימיזציה משתמשת בגרדיאנטים של פונקציית ההפסד לפי המשקלים ומהצעת צעדים קטנים(learning rate) במשקלים של הפרמטרים בכדי לשפר את ביצועי המודל.

פונקציית האופטימיזציה שאני בחרתי היא Adam (Adaptive Moment Estimation).

הדברים שמייחדים את הפונקציה הזו הם:

- היכולת שלה להתאים באופן דינמי ופרטי לכל פרמטר את ה learning rate שלו על פי היסטוריית הגרדיאנטים של אותו הפרמטר.
- בנוסף Adam הוא מאין שילוב של אלגוריתם Momentum ושל אלגוריתם RMSprop בכך שהוא עושה שימוש בממוצע נע של הגרדיאנטים להאצת הירידה ובנוסף גם בממוצע נע של ריבועי הגרדיאנטים להתאמת שיעור הלמידה בכל פעם.
- Adam הוא מאוד חסכוני בזיכרון ולכן יותר מהיר מאופטימייזרים אחרים.

בעיקר הפונקציה הזו מאוד טובה עם מודלים מורכבים כמו מודלי LSTM ומודלים מאומנים מראש ולכן בחרתי בה עבור המודל שלי.

תרשים UML של המחלקות המשמשות את המודל

תוצאות המודל

```
Epoch 1/60
    667/Unknown 141s 185ms/step - accuracy: 0.5048 - loss: 0.6936/usr/local/lib/python3.11/dist-packages/keras/src/trainers/
  self._interrupted_warning()
667/667 ━━━━━━━━━━━━━━━━━━━━ 167s 224ms/step - accuracy: 0.5048 - loss: 0.6936 - val_accuracy: 0.4960 - val_loss: 0.6934
Epoch 2/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 135s 196ms/step - accuracy: 0.4954 - loss: 0.6935 - val_accuracy: 0.5001 - val_loss: 0.6934
Epoch 3/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 135s 196ms/step - accuracy: 0.5037 - loss: 0.6934 - val_accuracy: 0.5009 - val_loss: 0.6932
Epoch 4/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 135s 196ms/step - accuracy: 0.5067 - loss: 0.6934 - val_accuracy: 0.4993 - val_loss: 0.6931
Epoch 5/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 135s 197ms/step - accuracy: 0.4921 - loss: 0.6934 - val_accuracy: 0.5080 - val_loss: 0.6931
Epoch 6/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 135s 197ms/step - accuracy: 0.5120 - loss: 0.6931 - val_accuracy: 0.4982 - val_loss: 0.6933
Epoch 7/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 135s 197ms/step - accuracy: 0.4954 - loss: 0.6933 - val_accuracy: 0.4949 - val_loss: 0.6932
Epoch 8/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 136s 197ms/step - accuracy: 0.4970 - loss: 0.6934 - val_accuracy: 0.4999 - val_loss: 0.6932
Epoch 9/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 136s 198ms/step - accuracy: 0.4974 - loss: 0.6934 - val_accuracy: 0.4991 - val_loss: 0.6932
Epoch 10/60
667/667 ━━━━━━━━━━━━━━━━━━━━ 136s 198ms/step - accuracy: 0.5031 - loss: 0.6933 - val_accuracy: 0.4999 - val_loss: 0.6931
```

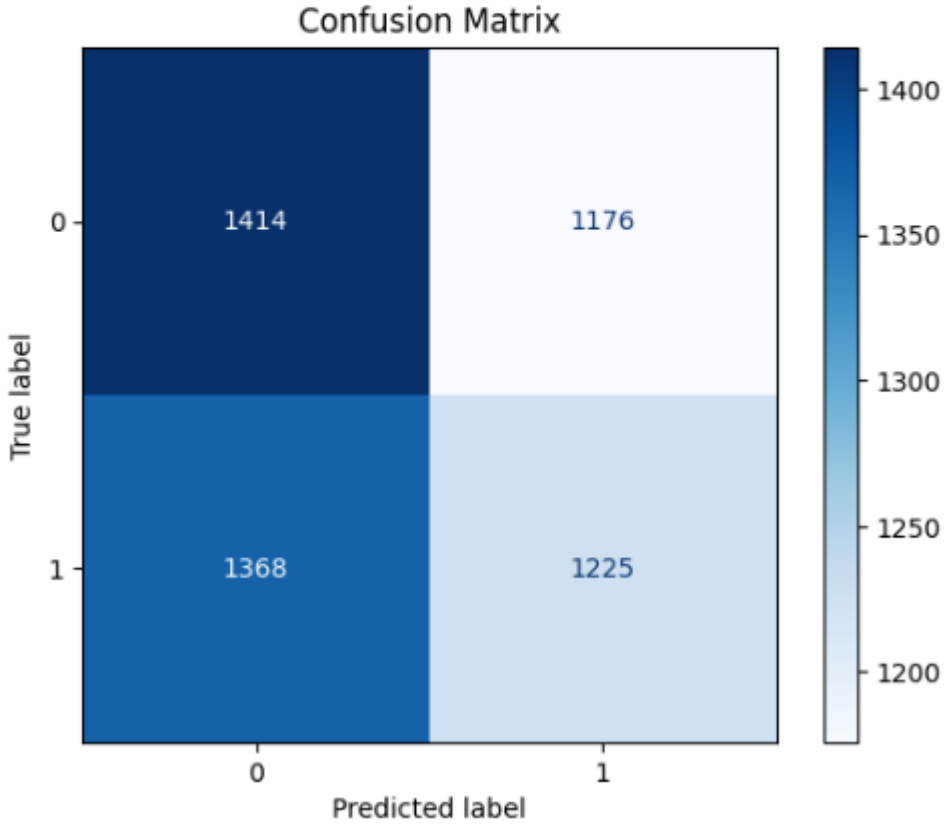
```
162/162 ━━━━━━━━━━━━━━━━━━━━ 24s 135ms/step - accuracy: 0.5094 - loss: 0.6931
Test acuracy: 0.5054987668991089
```

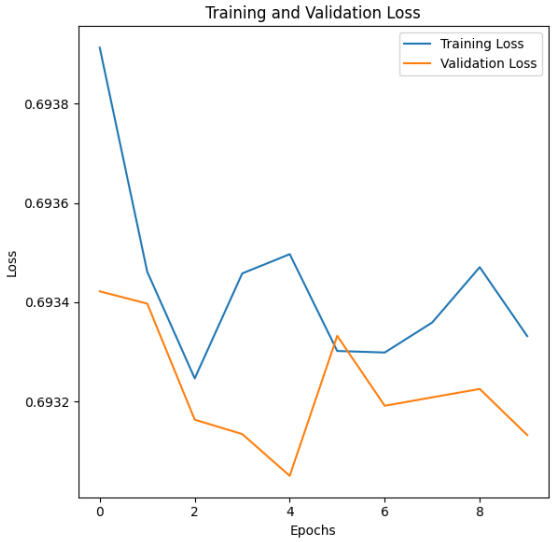
Training Accuracy: 0.5031

Test Accuracy : 0.5054

Loss: 0.6933

```
Classification Report:
              precision    recall  f1-score   support

           0     0.5083    0.5459    0.5264      2590
           1     0.5102    0.4724    0.4906      2593

    accuracy                         0.5092      5183
   macro avg     0.5092    0.5092    0.5085      5183
weighted avg     0.5092    0.5092    0.5085      5183
```



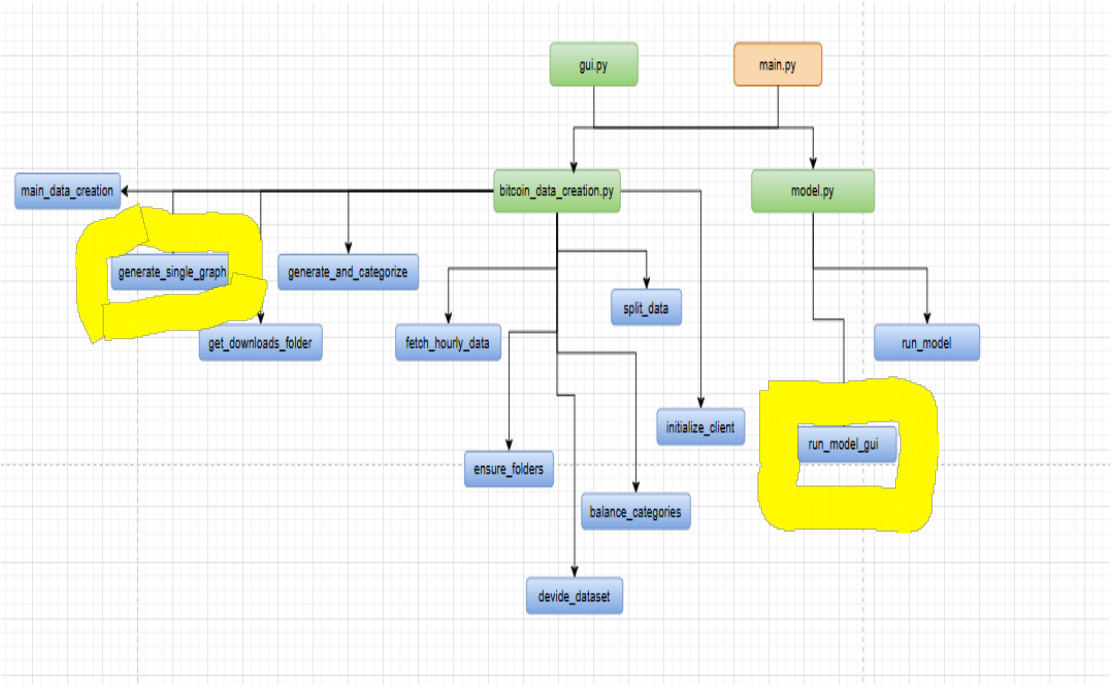


מסקנה

דרך תוצאות המודל אפשר לשים לב שהמודל לא מצליח ללמוד לעומק את הדפוסים ושהוא באמת נמצא במצב של underfitting . בהתחשב בעובדה שניסיתי מספר סוגי מודלים שונים ופתרתי בעיות כמו ה-imbalance , אני לא מצליח לחשוב על עוד דרך שבה אוכל לגרום למודל לעבוד בצורה טובה יותר, ואני יכול להגיד שככל הנראה עקב ההשפעות החיצוניות שיש על שוק הביטקוין שציינתי במבוא, המודל לא מצליח ללמוד מנתונים מסודרים ולזהות דפוס קבוע שאותו הוא יכול לחזות ולכן המודל חוזה באופן שהוא כמעט אקראי(אחוז דיוק קרוב ל50 אך גדול ממנו).

לאחר מחקר שעשיתי גיליתי שכמעט כל המודלים הסובבים סביב תחזית מחיר הביטקוין נמצאים במצב דומה לשלי(דיוק של 0.57/ דיוק של 0.55 ועוד...) מהסיבה שהמשימה של תחזית תנודתיות הביטקוין היא משימה מורכבת שלטובתה נדרשים ידע וכלים מעמיקים (שבשלב זה עדיין אין לי), אך בכל זאת לקחתי על עצמי את האתגר ולמרות שהמודל שלי לא חוזה את תנודתיות השוק עם דיוק גבוה אני מאוד מרוצה מתהליך שבו למדתי המון ומהדרך שעברתי בבניית הפרוייקט.

שלב היישום



לטובת יישום הפרוייקט לממשק משתמש אני משתמש בספרייה הנקראת tkinter.

ספרייה זו היא ספרייה סטנדרטית לבניית GUI (Graphic User Interface) בשפת פייתון.

הספרייה מאוד פשוטה וקלה לשימוש ופיתוח ובנוסף היא לא דורשת התקנה ולרוב כבר קיימת על רוב המחשבים .

בנוסף לכך הספרייה מציעה מגוון של widgets שונים שניתן לבנות דרכה מה שמאפשר בנייה של ממשק משתמש פשוט אך איכותי ואטרקטיבי.

את המודל אני מעלה לקוד הGUI שלי דרך קובץ השמור באותה התייקיה המכיל את המודל המאומן, ובעזרת הקוד הבא. ויש גם אופציה להעלאת מודל מצד המשתמש.

```
try:
    model = load_model(
        'C:/Users/itama/Bitcoin Prediction/gui/model.keras',
        custom_objects={'apply_mobilenet_per_frame': apply_mobilenet_per_frame}
    )
    print("Model loaded successfully.")
except Exception as e:
    print(f"Model Loading Error: Failed to load model: {e}")
    model = None
```

המודל עצמו נטען ביחד עם הפונקציה הבאה בכדי להתאים לדרישות המודל המאומן מראש(Mobilenetv2).

```
def apply_mobilenet_per_frame(x):
    batch_size = tf.shape(x)[0]
    steps = tf.shape(x)[1]
    x_new = tf.reshape(x, (-1, image_size_x, image_size_y, channels))
    features = mobilenet_layer(x_new)
    features = tf.reshape(features, (batch_size, steps, mobilenet_feature_dim))
    return features
```

קוד הקולט את ה-data המיועד לחיזוי ומתאים אותו לחיזוי

את ה data שעליו יבוצע החיזוי המודל קולט דרך הפונקציה הבאה:

```python
def upload_predict_image():
    global img_tk # keeping reference to avoid garbage collection
    if model is None:
        prediction_lbl.config(text="Model Not Loaded: Cannot predict.")
        return
    file_path = filedialog.askopenfilename(title="Select Image", filetypes=[("Image Files", "*.jpg;*.png;*.jpeg")])

    if not file_path:
        return

    try:
        show_loading()
        img= Image.open(file_path).convert( 'RGB')
        img= img.resize((image_size_x, image_size_y))
        img_tk= ImageTk.PhotoImage(img)
        image_lbl.config(image= img_tk)
        img_array= np.array(img) / 255.0
        img_array= np.expand_dims(img_array , axis= 0)
        seq_len= 12
        img_seq= np.tile(img_array, (1, seq_len, 1, 1, 1))
        pred = model.predict(img_seq)
        confidence = float(pred[0][0])
        if confidence > 0.5:
            predicted='Up'
        else:
            predicted= 'Down'

        output_text= f"prediction: {predicted} (Confidence: {confidence:.2f})"
        prediction_lbl.config(text=output_text)
        pulse_lbl(prediction_lbl)
        history_listbox.insert(0, output_text)
    except Exception as e:
        prediction_lbl.config(text=f"Prediction Error: {e}")
    finally:
        hide_loading()
```

הפונקציה הזאת מקבלת את התמונה דרך הקובץ אותו בחר המשתמש.

הסינון הראשוני הוא הסינון בסוג הקבצים שהמודל מקבל בשורה הבאה:

```python
file_path = filedialog.askopenfilename(title="Select Image", filetypes=[("Image Files", "*.jpg;*.png;*.jpeg")])
```

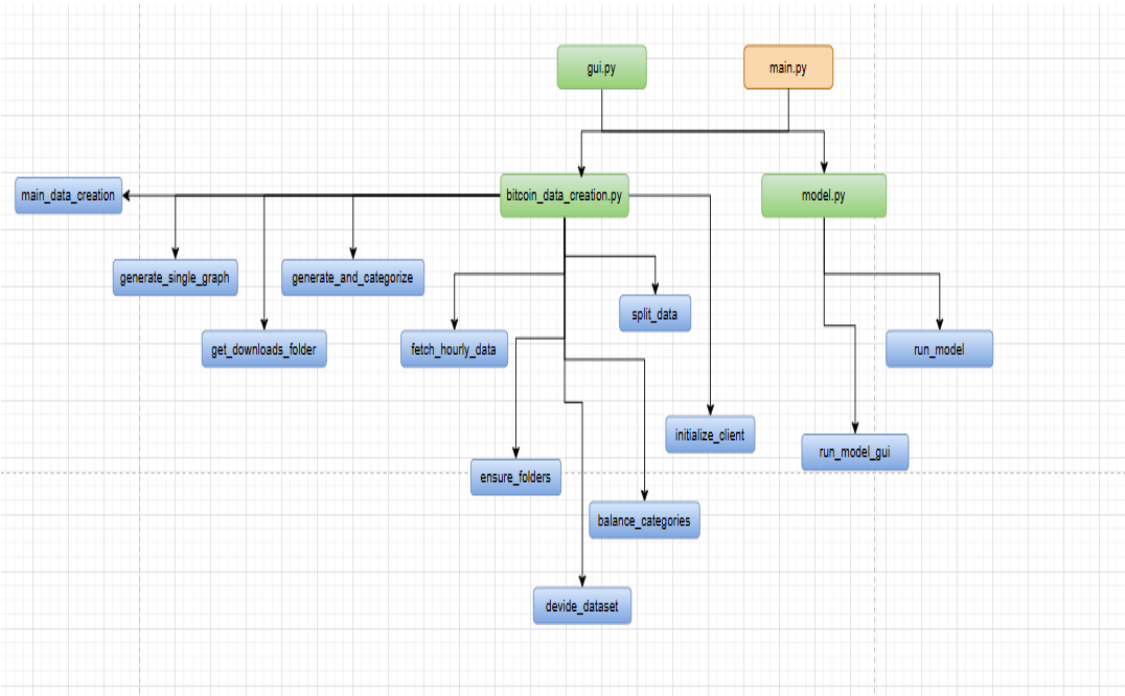
לאחר מכן התמונה נטענת וממורת ל-RGB, עושים על התמונה פעולה של resize המתאימה את הגודל שלה לגודל המתאים למודל(224X224),לאחר מכן מעבירים את התמונה למערך מסוג numpy ומנרמלים את הערכים לטווח של בין 0 ל-1.

בשלב הבא מוסיפים לתמונה מימד batch ומעבירים אותה למודל בתור sequence של 12 פעמים את התמונה.

```python
img_seq= np.tile(img_array, (1, seq_len, 1, 1, 1))
```

ולאחר מכן מבצעים את החיזוי.

מדריך למפתח



:Bitcoin_data_creation.py

הקובץ הזה הוא קובץ קוד (מסוג py.) שמטרתו ליצור את ה-data ולהכין אותו לטובת שלב אימון, בדיקה וולידציית המודל . הקובץ נבנה בצורה הבאה:

```python
import os
import random
import shutil
import pandas as pd
import matplotlib.pyplot as plt
import mplfinance as mpf
from datetime import datetime, timedelta
from binance.client import Client
```

ייבוא הספריות לטובת בניית הקוד.

```python
def initialize_client(api_key=None, api_secret=None):
    return Client(api_key, api_secret)
```

הפונקציה Initialize_client יוצרת ומחזירה אובייקט client לפי מפתח וסוד API שהם זוג קודים שמנפיקה מערכת עבור כל משתמש על מנת שיבצע אליה(למערכת) חיבור ישיר, פרטי ומאובטח שדרכו ניתן להשתמש בפעולות שקיימות במערכת.

בפונקציה זו נשתמש בשביל לקרוא לAPI של Binance (שהיא פלטפורמת מסחר) בהמשך.

```python
# Fetch hourly Bitcoin data from Binance
def fetch_hourly_data(client, symbol, interval, start_date, end_date):
    all_data = []
    while True:
        klines = client.get_historical_klines(symbol, interval, start_date, end_date, limit=1000)
        if not klines:
            break
        all_data.extend(klines)
        last_timestamp = klines[-1][0]
        start_date = datetime.utcfromtimestamp(last_timestamp / 1000) + timedelta(milliseconds=1)
        start_date = start_date.strftime('%Y-%m-%d %H:%M:%S')
        if len(all_data) >= 60000:  # Limit data
            break
    columns = ['timestamp', 'open', 'high', 'low', 'close', 'volume']
    data = pd.DataFrame(all_data, columns=columns + [None] * 6)  # Extra columns ignored
    data = data.iloc[:, :6]  # Keep relevant columns
    data['timestamp'] = pd.to_datetime(data['timestamp'], unit='ms')
    data = data[['timestamp', 'open', 'high', 'low', 'close', 'volume']]
    data[['open', 'high', 'low', 'close', 'volume']] = data[['open', 'high', 'low', 'close', 'volume']].astype(float)
    return data
```

הפונקציה fetch_hourly_data משכת מה־ APIשל Binance נתונים עבור מטבע מסוים(שנקבע אותו להיות הביטקוין בזימון של הפונקציות) בטווח תאריכים נתון, ומחזירה אותם כטבלת DataFrame מסודרת של pandas .

```python
# Ensure folders exist
def ensure_folders(base_folder, categories):
    for category in categories:
        input_folder = os.path.join(base_folder, category, "input")
        output_folder = os.path.join(base_folder, category, "output")
        os.makedirs(input_folder, exist_ok=True)
        os.makedirs(output_folder, exist_ok=True)
```

הפונקציה ensure_folders יוצרת תיקיות input (בהן השתמשתי לאורך הפרויקט) ותיקיותoutput (בהן שמורות תמונות של גרף הביטקוין ב־24 השעות שלאחר כל גרף input למרות שבפועל השתמשתי בנתונים עצמם ולא בתמונות כדי לקבוע אם הייתה עלייה או ירידה במחיר ב-24 השעות שלאחר כל גרף Input) עבור כל קטגוריה (label) שברשימת הקטגוריות שנעביר לה, בתוך תיקיית בסיס שנבחר.
אם התיקיות כבר קיימות, היא לא מוחקת או משנה אותן.

```
# Balance categories by trimming excess files
def balance_categories(base_folder, categories):
    file_counts = {
        category: len(os.listdir(os.path.join(base_folder, category, "input")))
        for category in categories
    }
    min_count = min(file_counts.values())

    for category in categories:
        for subfolder in ["input", "output"]:
            folder_path = os.path.join(base_folder, category, subfolder)
            files = sorted(os.listdir(folder_path))
            for file_to_remove in files[min_count:]:
                os.remove(os.path.join(folder_path, file_to_remove))
```

בכדי להימנע ממצב של overfitting ו- imbalancing הייתי צריך לדאוג שכמות ה data שיש לי המקוטלגות כירידות שווה לכמות ה data שיש לי המקוטלגות כעליות, ואת זה עשיתי דרך פונקציה שקראתי לה balance_categories שתפקידה להוריד שורות של data מהclass שבו יש יותר data כך שיהיה מספר שווה בין ה classes .

```python
# Split data into train, validation, and test sets
def split_data(base_folder, categories, train_ratio=0.8, val_ratio=0.1):
    test_ratio = 1 - train_ratio - val_ratio

    for category in categories:
        input_folder = os.path.join(base_folder, category, "input")
        output_folder = os.path.join(base_folder, category, "output")

        files = sorted(os.listdir(input_folder))
        random.shuffle(files)
        total_files = len(files)

        train_split = int(total_files * train_ratio)
        val_split = int(total_files * (train_ratio + val_ratio))

        train_input_folder = os.path.join(base_folder, "train", category, "input")
        train_output_folder = os.path.join(base_folder, "train", category, "output")
        val_input_folder = os.path.join(base_folder, "validation", category, "input")
        val_output_folder = os.path.join(base_folder, "validation", category, "output")
        test_input_folder = os.path.join(base_folder, "test", category, "input")
        test_output_folder = os.path.join(base_folder, "test", category, "output")

        os.makedirs(train_input_folder, exist_ok=True)
        os.makedirs(train_output_folder, exist_ok=True)
        os.makedirs(val_input_folder, exist_ok=True)
        os.makedirs(val_output_folder, exist_ok=True)
        os.makedirs(test_input_folder, exist_ok=True)
        os.makedirs(test_output_folder, exist_ok=True)

        for i, file in enumerate(files):
            src_input = os.path.join(input_folder, file)
            src_output = os.path.join(output_folder, file.replace("input", "output"))

            if i < train_split:
                dest_input = os.path.join(train_input_folder, file)
                dest_output = os.path.join(train_output_folder, file.replace("input", "output"))
            elif i < val_split:
                dest_input = os.path.join(val_input_folder, file)
                dest_output = os.path.join(val_output_folder, file.replace("input", "output"))
            else:
                dest_input = os.path.join(test_input_folder, file)
                dest_output = os.path.join(test_output_folder, file.replace("input", "output"))

            shutil.move(src_input, dest_input)
            shutil.move(src_output, dest_output)
```

בכדי לשמור על סדר וארגון ה dataset החלטתי לפצל את ה dataset כבר בתוך תיקיית הפרוייקט לתיקיות של ["train", "test", "validation"] ועשיתי זאת בעזרת פונקציה שקראתי לה split_data אשר יוצרת את התיקיות החדשות במידת הצורך ומחלקת את ה dataset אליהן לפי יחס חלוקה נתון.

```python
def generate_and_categorize(data, base_folder, window_size=24, batch_size=500, resolution=(224, 224)):
    ensure_folders(base_folder, ["up", "down"])

    num_images = len(data) - (2 * window_size) + 1
    total_batches = (num_images + batch_size - 1) // batch_size

    for batch_idx in range(total_batches):
        start_idx = batch_idx * batch_size
        end_idx = min(start_idx + batch_size, num_images)

        print(f"Processing batch {batch_idx + 1}/{total_batches} ({start_idx}-{end_idx})...")

        for i in range(start_idx, end_idx):
            input_subset = data.iloc[i:i + window_size]
            input_subset.set_index('timestamp', inplace=True)

            output_subset = data.iloc[i + window_size:i + 2 * window_size]
            output_subset.set_index('timestamp', inplace=True)

            open_price = output_subset.iloc[0]['open']
            close_price = output_subset.iloc[-1]['close']
            price_change = close_price - open_price

            classification = "up" if price_change > 0 else "down"

            input_folder = os.path.join(base_folder, classification, "input")
            output_folder = os.path.join(base_folder, classification, "output")

            input_file_name = os.path.join(input_folder, f"input_graph_{i + 1:05d}.png")
            fig, ax = plt.subplots(figsize=(resolution[0] / 100, resolution[1] / 100))
            mpf.plot(input_subset, type='candle', ax=ax, style="yahoo", volume=False)
            ax.set_ylabel("Price", fontsize=4)
            ax.tick_params(axis='both', labelsize=5)
            plt.savefig(input_file_name, dpi=100, bbox_inches='tight')
            plt.close(fig)

            output_file_name = os.path.join(output_folder, f"output_graph_{i + 1:05d}.png")
            fig, ax = plt.subplots(figsize=(resolution[0] / 100, resolution[1] / 100))
            mpf.plot(output_subset, type='candle', ax=ax, style="yahoo", volume=False)
            ax.set_ylabel("Price", fontsize=4)
            ax.tick_params(axis='both', labelsize=5)
            plt.savefig(output_file_name, dpi=100, bbox_inches='tight')
            plt.close(fig)

        print(f"Batch {batch_idx + 1}/{total_batches} completed.")

    print(f"Generated and categorized {num_images} candlestick chart pairs.")
```

הפונקציה generate_and_categorize יוצרת זוגות של גרפים מסוג candlestick charts מפרקי זמן עוקבים מנתוני הביטקוין ומסווגת כל זוג לפי כיוון השינוי במחיר של פרק הזמן הבא (24 שעות הבאות) העוקב לגרף ה-input המדובר ושומרת את גרפי ה Input וה-output כקבצי תמונה בעלי אותו המספר בתיקיות מתאימות.

שוב אציין שבתיקיית ה- output לא השתמשתי בסוף.

```python
def main_data_creation():
    # API keys (use None if not needed)
    api_key = None
    api_secret = None

    # Initialize Binance client
    client = initialize_client(api_key, api_secret)

    # Define parameters for data fetching
    symbol = "BTCUSDT"
    interval = Client.KLINE_INTERVAL_1HOUR
    start_date = "2017-01-03"
    end_date = "2025-03-02"

    print("Fetching data from Binance...")
    data = fetch_hourly_data(client, symbol, interval, start_date, end_date)

    # Define base folder for dataset storage
    base_folder = "candlestick_dataset"

    print("Generating and categorizing candlestick images...")
    generate_and_categorize(data, base_folder)

    print("Balancing categories...")
    balance_categories(base_folder, ["up", "down"])

    print("Splitting data into train, validation, and test sets...")
    split_data(base_folder, ["up", "down"])

    print("Process completed successfully!")
```

זו היא פונקציית הזימון שבה השתמשתי כדי לקרוא לפונקציות וליצור את הגרפים והתיקיות.

לAPI של Binance התחברתי ללא מפתח וסוד.

הפונקציה מתחברת ל API של Binance ולוקחת ממנה את הנתונים שהיא זקוקה להם, יוצרת ומסווגת גרפי candlestick מאזנת את מספר הדוגמאות בכל קטגוריה ולבסוף מחלקת ושומרת את ה dataset בתיקיות לפי הקטגוריות(classes) המתאימים.

כך מתקבל מערך נתונים מוכן לשימוש במודלים של למידת מכונה לניתוח מגמות בשוק הביטקוין.

אך לאחר שיצרתי את ה-dataset הבנתי שעליי להשתמש רק בתיקיית ה- input ולכן פיצלתי את תיקיית ה dataset לשתי תיקיות שונות:

candlestick_dataset_input-

candlestick_dataset_output-

שמתוכן השתמשתי רק בתיקיית ה-Input לטובת הפרויקט וזאת עשיתי בעזרת קריאה לפונקציה הבאה בנפרד.

```python
def devide_dataset():

    # Original dataset path
    original_dataset = 'candlestick_dataset'
    input_dataset = 'candlestick_dataset_input'
    output_dataset = 'candlestick_dataset_output'

    # Dataset splits
    splits = ['train', 'validation', 'test']
    classes = ['up', 'down']

    # Create new directory structures
    for dataset in [input_dataset, output_dataset]:
        for split in splits:
            for class_name in classes:
                os.makedirs(os.path.join(dataset, split, class_name), exist_ok=True)

    # Move files into the new structure
    for split in splits:
        for class_name in classes:
            input_folder = os.path.join(original_dataset, split, class_name, 'input')
            output_folder = os.path.join(original_dataset, split, class_name, 'output')

            input_target = os.path.join(input_dataset, split, class_name)
            output_target = os.path.join(output_dataset, split, class_name)

            # Move input files
            if os.path.exists(input_folder):
                for file in os.listdir(input_folder):
                    shutil.move(os.path.join(input_folder, file), os.path.join(input_target, file))

            # Move output files
            if os.path.exists(output_folder):
                for file in os.listdir(output_folder):
                    shutil.move(os.path.join(output_folder, file), os.path.join(output_target, file))

    print('Dataset successfully split into input and output datasets.')
```

```python
def get_downloads_folder():
    return os.path.join(os.path.expanduser("~"), "Downloads")
```

```python
def generate_single_graph(client, symbol, interval, start_date, resolution=(224, 224)):

    # Convert start_date to datetime and compute end_date (24 hours later)
    start_dt = datetime.strptime(start_date, "%Y-%m-%d %H:%M:%S")
    end_dt = start_dt + timedelta(days=1)
    end_date = end_dt.strftime("%Y-%m-%d %H:%M:%S")

    # Fetch data
    data = fetch_hourly_data(client, symbol, interval, start_date, end_date)

    if data.empty:
        print("No data available for the selected timeframe.")
        return None

    # Get the user's Downloads folder dynamically
    downloads_folder = get_downloads_folder()

    # Ensure the Downloads folder exists (it always should, but just in case)
    os.makedirs(downloads_folder, exist_ok=True)

    # File name based on start date
    filename = f"btc_graph_{start_date.replace(':', '-')}.png"
    filepath = os.path.join(downloads_folder, filename)

    # Plot the candlestick chart with matching style
    fig, ax = plt.subplots(figsize=(resolution[0] / 100, resolution[1] / 100))
    mpf.plot(data.set_index('timestamp'), type='candle', ax=ax, style='yahoo', volume=False)

    # Apply the same label styling
    ax.set_ylabel("Price", fontsize=4)
    ax.tick_params(axis='both', labelsize=5)

    # Save the chart
    plt.savefig(filepath, dpi=100, bbox_inches='tight')
    plt.close(fig)

    print(f"Graph saved successfully at {filepath}")

    return filepath
```

שני הפונקציות האלו משמשות את ה gui ליצירת גרף יחיד לפי בקשת המשתמש ולשמירת הגרף על המחשב של המשתמש.

:gui_main.py

קובץ זה הוא קובץ קוד(מסוג py.) שבו השתמשתי בכדי ליצור את ה-GUI(Graphic User Interface) שבו כל מי שמעוניין להשתמש במודל יוכל להשתמש במידת הצורך.

הממשק מאפשר למשתמש להעלות תמונה של גרף הביטקוין שממנה המודל יחזה מה יקרה בטווח הזמן העוקב לתמונה(עלייה או ירידה במחיר).

הקוד נבנה בצורה הבאה:

```
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
import numpy as np
from PIL import Image
from PIL import ImageTk
from tensorflow.keras.models import load_model
import tensorflow_hub as hub
import os
import random
import tensorflow as tf
from tensorflow.keras.saving import register_keras_serializable
```

ייבוא הספריות הנחוצות לבניית הממשק.

```
image_size_x = 224
image_size_y = 224
channels = 3
mobilenet_feature_dim = 1280
fun_facts = [
    "Bitcoin was created in 2009!",
    "The smallest unit of Bitcoin is called a Satoshi.",
    "There will only ever be 21 million Bitcoins.",
    "Bitcoin transactions are recorded on a public ledger called the blockchain.",
    "The creator of Bitcoin is known as Satoshi Nakamoto.",
    "The first ever Bitcoin transaction was made for two Pizzas:)",
    "Bitcoin peak as of right now is 114,000$"
]

mobilenet_layer = hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4", trainable=False)
```

הגדרת משתנים גלובליים בהם אני אשתמש בהמשך.

```python
@register_keras_serializable()
def apply_mobilenet_per_frame(x):
    batch_size = tf.shape(x)[0]
    steps = tf.shape(x)[1]
    x_new = tf.reshape(x, (-1, image_size_x, image_size_y, channels))
    features = mobilenet_layer(x_new)
    features = tf.reshape(features, (batch_size, steps, mobilenet_feature_dim))
    return features
```

פונקציה זו היא פונקציה שצריך לטעון אותה ביחד עם טעינת המודל בהתאם לדרישות של המודל המאומן מראש שבו אני משתמש במודל שלי Mobilenetv2 .

```python
try:
    model = load_model(
        'C:/Users/itama/Bitcoin Prediction/gui/model.keras',
        custom_objects={'apply_mobilenet_per_frame': apply_mobilenet_per_frame}
    )
    print("Model loaded successfully.")
except Exception as e:
    print(f"Model Loading Error: Failed to load model: {e}")
    model = None
```

טעינת הקוד נעשית בחלק הקוד הזה אשר מאפשר אופציה בה קרתה בעיה בטעינת המודל ואז מופיעה הודעה אשר מזהירה ואומרת שאין מודל טעון בשביל להשתמש בו.

```python
def upload_m():
    global model
    m_path=filedialog.askopenfilename(title="Select model file", filetypes=[("keras models", "*.h5 *.keras *.tf"), ("All Files","*.*")])
    if not m_path:
        return
    try:
        new_model= load_model(m_path,custom_objects={'apply_mobilenet_per_frame': apply_mobilenet_per_frame,'KerasLayer': hub.KerasLayer})
        model=new_model
        model_status.config(text=f"Model Loaded: {os.path.basename(m_path)}")
        messagebox.showinfo("Success","Model Loaded")
    except Exception as e:
        messagebox.showerror("Error ",f"Failed to load the model:{str(e)}")
        model_status.config(text="model failed to load")
```

הפונקציה upload_m() מאפשרת למשתמש להעלות מודל משל עצמו. הכפתור המשתמש בפעולה זו מוגדר בהמשך.

```python
root = tk.Tk()
root.title("BTC Prediction App")
root.geometry("500x300")
root.configure(bg="#161616")

s = ttk.Style()
s.theme_use("clam")
s.configure("TFrame", background="#161616")
s.configure("TLabel", background="#161616", foreground="#FFD700", font=("Segoe UI", 12))
s.configure("TButton", background="#232323", foreground="#FFD700", font=("Segoe UI", 12, "bold"))
```

בחלק הזה של הקוד אני יוצר את חלון הממשק בעזרת השורה הראשונה ולאח מכן מגדיר את הממדים ואת העיצוב שיהיה באפליקציה.\

```python
def pulse_lbl(lbl, count=0):
    colors = ["#FFD700", "#FFFACD", "#FFD700"]
    lbl.config(foreground=colors[count % len(colors)])
    if count < 6:
        lbl.after(100, pulse_lbl, lbl, count + 1)
```

ברגע שקוראים לה הפונקציה הזו יוצרת הבהוב(שינוי מהיר בצבע) שמתרחש על התחזית של המודל מיד לאחר שהתחזית מופיעה(אחרי שמעלים את התמונה אותה רוצים לחזות).

```python
def show_fun_fact(event=None):
    fact= random.choice(fun_facts)
    messagebox.showinfo("Fun Fact", fact)
```

ברגע שקוראים לה הפונקציה הזו מציגה באמצעות התראה על המסך עובדה מעניינת רנדומלית מהמערך שהגדרתי בתחילת הקוד על הביטקוין.

```python
def end_app():
    if messagebox.askokcancel("Quit", "Do you really wish to quit?"):
        root.destroy()
```

ברגע שקוראים לה הפונקציה הזו מעלה התראה אשר שואלת האם המשתמש רוצה לצאת מהאפליקציה ואם הוא לוחץ על האוציה כן האפלקציה נסגרת

```python
def upload_predict_image():
    global img_tk # keeping reference to avoid garbage collection
    if model is None:
        prediction_lbl.config(text="Model Not Loaded: Cannot predict.")
        return
    file_path = filedialog.askopenfilename(title="Select Image", filetypes=[("Image Files", "*.jpg;*.png;*.jpeg")])

    if not file_path:
        return

    try:
        show_loading()
        img= Image.open(file_path).convert( 'RGB')
        img= img.resize((image_size_x, image_size_y))
        img_tk= ImageTk.PhotoImage(img)
        image_lbl.config(image= img_tk)
        img_array= np.array(img) / 255.0
        img_array= np.expand_dims(img_array , axis= 0)
        seq_len= 12
        img_seq= np.tile(img_array, (1, seq_len, 1, 1, 1))
        pred = model.predict(img_seq)
        confidence = float(pred[0][0])
        if confidence > 0.5:
            predicted='Up'
        else:
            predicted= 'Down'

        output_text= f"prediction: {predicted} (Confidence: {confidence:.2f})"
        prediction_lbl.config(text=output_text)
        pulse_lbl(prediction_lbl)
        history_listbox.insert(0, output_text)
    except Exception as e:
        prediction_lbl.config(text=f"Prediction Error: {e}")
    finally:
        hide_loading()
```

ברגע שקוראים לפונקציה הזו היא פותחת חלון לסייר הקבצים שממנו ניתן לבחור את התמונה ממנה ירצה המשתמש לבצע חיזוי ולאחר הבחירה של הקובץ היא מכינה אותו כדי שיתאים לעיבוד המודל. לאחר מכן חוזה ממנו את התנודתיות בפרק הזמן העוקב של הביטקוין(כלומר האם הוא יעלה או ירד לאחר הגרף) ולבסוף מציגה את התחזית ואת הביטחון של המודל בתחזית שלו.

```
main_frame= ttk.Frame(root, padding="20")
main_frame.pack(fill=tk.BOTH, expand=True)
```

קטע הקוד הזה מגדיר מאין מסגרת שבה נמצאים האלמנטים(הכפתורים והכותרות שאני אצור בהמשך) בשביל לשמור על ויזואליות נעימה.

```
try:
    logo_img= Image.open("bitcoin_logo.png").resize((60, 60))
    logo_tk=ImageTk.PhotoImage(logo_img)
    logo_lbl= ttk.Label(main_frame, image=logo_tk, background="#161616")
    logo_lbl.image= logo_tk
    logo_lbl.pack(pady= 5)
except Exception as e:
    print(f"Could not load logo: {e}")
    logo_lbl= ttk.Label(main_frame, text="BTC", background="#161616", foreground="#FFD700", font=("Segoe UI", 28, "bold"))
    logo_lbl.pack(pady=5)
```

קטע הקוד הזה אחראי על הצגת הלוגו של הביטקוין על המסך ובמקרה שלא מצליח להציג טקסט " BTC"

```
info_lbl= ttk.Label(main_frame, text= "Click to predict from uploaded image")
info_lbl.pack(pady=20)

upload_button= ttk.Button(main_frame, text="Upload Image" , command= upload_predict_image)
upload_button.pack(pady= 10)
```

קוד זה מגדיר את הכפתור של העלאה וחיזוי מתמונה ומשתמש בפקודה upload_predict_image כדי לעשות זאת. מעל הכפתור יש טקסט שבו רשום "לחץ פה בשביל להעלות תמונה ולחזות ממנה"

```
def on_enter(e):
    upload_button.config(style= "Hover.TButton")
def on_leave(e):
    upload_button.config(style= "TButton")
def on_press(e):
    upload_button.config(style= "Pressed.TButton")
def on_release(e):
    upload_button.config(style= "Hover.TButton")

s.configure("Hover.TButton", background="#FFD700", foreground= "#232323")
s.configure("Pressed.TButton", background= "#FFA500" , foreground="#232323")
```

בקוד זה כתבתי מספר פונקציות אשר מגדירות פונקציונליות של כפתור כאשר נמצאים בשטח שלו עם העכבר, כאשר עוזבים את השטח הזה , כאשר לוחצים וכאשר משחררים. ולמטה הגדרתי עיצוב למצבים אשר הפעולות שלמעלה מתייחסות אליהם אשר לא הוגדר להם עיצוב לפני.

```
history_lbl= ttk.Label(main_frame, text="Prediction History:")
history_lbl.pack(pady=(10,0))

history_listbox= tk.Listbox(main_frame, width= 40, height= 6 )
history_listbox.pack(pady=(0,10))

model_button=ttk.Button(main_frame,text="Upload custom model", command=upload_m)
model_button.pack(pady=10)


history_scroll= ttk.Scrollbar( main_frame, orient= "vertical", command= history_listbox.yview)
history_listbox.config( yscrollcommand= history_scroll.set)
history_scroll.pack(side= "right", fill= "y")
```

חלק הקוד הזה יוצר את תיבת היסטוריית החיזויים יוצר לה כותרת ומוסיף את האופציה לגלול בתיבה זו ובנוסף יוצר את כפתור העלאת המודל של המשתמש ומשתמש בפונקציה upload__m() למטרה זו.

```
upload_button.bind("<Enter>" , on_enter)
upload_button.bind("<Leave>" , on_leave)
upload_button.bind("<ButtonPress-1>" , on_press)
upload_button.bind("<ButtonRelease-1>", on_release)

end_button= ttk.Button(main_frame, text="End" , command= end_app)
end_button.pack(pady= 10)
s.configure("End.TButton", background="#c0392b" , foreground= "#fff" , font= ("Segoe UI", 12, "bold"))
end_button.config(style="End.TButton")

model_status= ttk.Label(main_frame, text= "Model Status: Loading.....")
model_status.pack( pady= 5)

prediction_lbl= ttk.Label(main_frame , text ="" , font =("Arial", 12))
prediction_lbl.pack(pady= 10)

image_lbl= ttk.Label(main_frame)
image_lbl.pack(pady= 10)

root.bind('<Control-e>', show_fun_fact)
root.focus_set()
```

בקטע זה הגדרתי לכפתור ההעלאה והחיזוי את האנימציות שיוצרות הפעולות ,on_enter() on_leave(), on_press() ...

יצרתי את כפתור הסיום המשתמש בפעולה end_app() בכדי לסגור את האפליקציה ויצרתי לו עיצוב.

הגדרתי תיבת טקסט שמציג טקסט על סטטוס המודל(נטען(ברירת מחדל), מוכן, לא נמצא...).

הגדרתי את תיבת הטקסט לתשובת החיזוי ותיבה להצגת התמונה שעליה נעשה החיזוי.

ולבסוף יצרתי קישור למקש ctrl+E שמציג עובדה מעניית כל פעם שלוחצים עליו.

```
prog = ttk.Progressbar(main_frame , mode= 'indeterminate')
```

```
def show_loading():
    prog.pack(pady= 10)
    prog.start()
def hide_loading():
    prog.stop()
    prog.pack_forget()
```

הקוד הבא מגדיר קו התקדמות לטובת המודל בזמן החיזוי ובעזרת הפונקציות המופיעות פה והקריאות המתאימות להן בפונקציית העלאה והחיזוי מופיעות בזמן החיזוי ונעלמות אחריו.

```
def train_model():
    try:
        accuracy,loss=run_model_gui()
        messagebox.showinfo("Test Accuracy:", accuracy)
        messagebox.showinfo("Test Loss:", loss)
    except Exception as e:
        messagebox.showinfo("error has accured")
```

הפונקציה הבאה נקראת train_model() והיא בעצם קוראת לפונקציה במחלקת המודל המיועדת במיוחד לאימון עבור ה-gui ובכך מאפשרת למשתמש לאמן את המודל מה-gui. לאחר סיום האימון הפונקציה מקפיצה בהודעה את ערכי ה Test accuracy וה-Test loss.

```
train_button= ttk.Button(main_frame,text="Train the model",command=run_model_gui)
train_button.pack(pady=10)
```

בקטע קוד זה אני מגדיר כפתור שברגע הלחיצה קורא לפונקציית train_model() ובכך מאמן את המודל ומעלה הודעות שמכילות את ערכי ה- test accuracy וה-test loss.

```
def graph_parameters():
    symbol= simpledialog.askstring("Symbol", "Enter your market symbol(BTCUSD)", parent= root )
    if not symbol:
        return
    time_frame = simpledialog.askstring("Time Frame", "Enter you wished time frame(1h)", parent= root)
    if not time_frame:
        return
    start_date=simpledialog.askstring("Start Full Date","Enter strat date (YYYY-MM-DD HH:MM:SS" )
    if not start_date:
        return

    try:
        show_loading()
        filepath= generate_single_graph(client,symbol,time_frame,start_date)
        hide_loading()

        if filepath:
            messagebox.showinfo("Success", f"Graph saved to:\n{filepath}")
        else:
            messagebox.showwarning("No Data", "No data available for the selected timeframe.")
    except Exception as e:
        hide_loading()
        messagebox.showerror("Error", f"Failed to generate graph:\n{str(e)}")
```

הפונקציה graph_parameters() קוראת לפונקציה ממחלקת bitcoin_data_creation ושמה generate_single_graph() אשר יוצרת גרפי candlestick לפי הערכים שמכניסים לה. ובכך בעצם הפונקציה שלנו קוראת למשתמש לבחור בשוק, טווח זמנים והקפיצות והפונקציה graph_parameters() יוצרת את הגרף ושומרת אותו בתיקיית ההורדות של המשתמש.

```
graph_button= ttk.Button(main_frame, text= "Generate Candlestick Graph", command= graph_parameters)
graph_button.pack(pady=10)
```

ופה בעצם בניתי את הכפתור שכאשר לוחצים עליו הוא קורא לפונקציה graph_parameters()
ובכך יוצר למשתמש את הגרף הרצוי ושומר אותו בזיכרון המחשב שלו.

```
if model is not None:
    model_status.config(text= "Model Loaded with Success!")
else:
    model_status.config(text= "Model Failed to Load:)")
```

חלק הקוד הזה מוודא האם מודל הצליח להטען ומשנה את המשתנה mode_status בהתאם.

```
root.mainloop()
```

לסיום, שורה זו היא השורה שדאגת שהכל ימשיך לרוץ עד שאומרים לממשק אחרת.

Model.py :

קובץ זה הוא קובץ קוד (מסוג py.) שבו אני משתמש עבור הרצת פונקציות שונות במחלקות המודל.

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import tensorflow_hub as hub #is responsible for importing the pretrained models
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator# data augmentation addition- augmentation
from tensorflow.keras.callbacks import EarlyStopping# early stopping addition- augmentation
from collections import Counter
import collections
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import os
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import EfficientNetB3
from sklearn.utils.class_weight import compute_class_weight# help me decide on the wieghts of the calsses when
```

בקטע הקוד זה ייבאתי את המחלקות הנחוצות לטובת קוד אימון המודל.

```
def run_model():
```

הקוד שמתחת לפונקציה הזו נראה כך

```
# Setup
image_size_x = 224
image_size_y = 224
batch_size = 32
sequence_length = 12
seed = 42
channels = 3
epochs = 60
optimizer= tf.keras.optimizers.Adam(learning_rate=0.0000055)
```

אלו המשתנים וההיפר פרמטרים שהגדרתי בתחילת פונקציית בניית המודל

```
os.makedirs(augmented_train_dir, exist_ok=True)
os.makedirs(os.path.join(augmented_train_dir, 'down'), exist_ok=True)
os.makedirs(os.path.join(augmented_train_dir, 'up'), exist_ok=True)


datagen = ImageDataGenerator(
    rotation_range= 25,
    channel_shift_range=10,
    width_shift_range =0.05,
    height_shift_range= 0.05,
    shear_range= 0.05,
    zoom_range= 0.05,
    horizontal_flip= False,
    brightness_range =[0.7, 1.1]
    )


aug_generator = datagen.flow_from_directory(
    train_dir,
    target_size=(image_size_x, image_size_y),
    batch_size=1,
    class_mode='categorical',
    #save_to_dir=augmented_train_dir,
    save_prefix='aug',
    save_format='png'
    )
```

קטע הקוד הזה יוצר תחילה תיקיות לאחסון ה data ובמקרה שהן כבר קיימות הוא לא משנה אותן או יוצר נוספות וממשיך לשלב הגדרת פעולת האוגמנטציה.

ראשית הגדרתי את הטרנספורמציות שיקרו בתהליך האוגמנטציה דרך פונקציית ImageDataGenerator()

לאחר מכן יצרתי את הגנרטור שבו הגדרתי את המיקום ממנו ילקחו התמונות לביצוע האוגמנטציה את מימדי התמונות את סוג ה labels ואת התת שם של הקבצים ואת סוג הקובץ שיווצר.

```python
class_indices = aug_generator.class_indices
# Reverse the dictionary to map index to class name
idx_to_class = {v: k for k, v in class_indices.items()}

num_augmented_images_to_generate = 5000
print(f"Generating and saving {num_augmented_images_to_generate} augmented images...")

i = 0
original_image_i = 0
num_original_images = len(aug_generator.filepaths)

for batch_data in aug_generator:
    images_batch= batch_data[0]
    labels_batch= batch_data[1]

    img= images_batch[0]
    label_one_hot = labels_batch[0]

    #checks class index and class name
    class_i= np.argmax(label_one_hot)
    class_name= idx_to_class[class_i]


    original_filepath = aug_generator.filepaths[original_image_i % num_original_images]
    original_filename = os.path.basename(original_filepath).split('.')[0] # Get base name without extension

    #makes the file path to the right label
    save_path =os.path.join(augmented_train_dir, class_name)

    #Saves the images and ensures vcorrect formatting
    image_to_save= tf.keras.preprocessing.image.array_to_img(img)

    #Gives the files names
    augmented_name= f'aug_{original_filename}_{i}.{aug_generator.save_format}'
    full_path= os.path.join(save_path, augmented_name)

    image_to_save.save(full_path, format=aug_generator.save_format)

    i += 1
    original_image_i += 1

    if i >= num_augmented_images_to_generate:
        break

    print("Augmentation complete.")
```

קטע הקוד הבא בעצם מתחיל בלשמור את הlabel של כל התמונות שעליהן הוא עושה אוגמנטציה כדי שיוכל לתת לתמונות החדשות את ה label המתאים להן(לחלק אותן לתיקיות המתאימות).

לאחר מכן הוא מגדיר את כמות התמונות שארצה ליצור.

ולאורך מספר התמונות בלולאה נוצרת תמונה ונשמרת במחלקה המתאימה לה.

את שני קטעי הקוד האחרונים(האחראים על יצירת התמונות דרך אוגמנטציה) הרצתי פעם אחת ולאחר מכן השארתי כהערה בקוד המודל.

```python
def preprocess(image, label):
    image = tf.image.convert_image_dtype(image, tf.float16)  # Convert to float16
    return image, label
```

פונקציית ה-preprocess אחראית על ביצוע עיבוד מקדים בכך שהוא מעביר את התמונה לסוג float16 (ומשאיר את ה labels כמו שהם) ובכך חוסך בזיכרון

```python
def augment(image, label):
    image= tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image , 0.9, 1.1)
    image= tf.image.random_saturation(image, 0.9 , 1.1)
    return image, label
```

הפונקציה הזו מבצעת אוגמנטציה על נתונים קיימים מבלי להגדיל את מאגר הנתונים אך היא מגדילה את הגיוון של הנתונים.

```python
# Data loading
train_raw = tf.keras.preprocessing.image_dataset_from_directory(
    "candlestick_dataset_input/train",
    image_size=(224, 224),
    batch_size=batch_size,
    seed=seed
    ).map(lambda x, y: (x/255, y)).map(preprocess).map(augment).unbatch()

train_raw_augmented= tf.keras.preprocessing.image_dataset_from_directory(
    "candlestick_dataset_input/train_aug",
    image_size=(224, 224),
    batch_size=batch_size,
    seed=seed
    ).map(lambda x, y: (x/255, y)).map(preprocess).unbatch()

val_raw = tf.keras.preprocessing.image_dataset_from_directory(
    "candlestick_dataset_input/validation",
    image_size=(224, 224),
    batch_size=batch_size,
    seed=seed
    ).map(lambda x, y: (x/255, y)).map(preprocess).unbatch()

test_raw = tf.keras.preprocessing.image_dataset_from_directory(
    "candlestick_dataset_input/test",
    image_size=(224, 224),
    batch_size=batch_size,
    seed=seed
    ).map(lambda x, y: (x/255, y)).map(preprocess).unbatch()

#combining and shuffling the train dirs for feeding the model and avoiding bias
train_raw_combined= train_raw.concatenate(train_raw_augmented)
train_raw = train_raw_combined.shuffle(buffer_size=1000, seed=seed)
```

קטע הקוד הבא הוא העלאת התיקיות לתוך משתנים המתאימים להם בתור ה-train test validation ובנוסף גם את תיקיית הtrain_aug העלתי ולאחר מכן חיברתי אותה למשתנה המחזיק את הdata הרגיל של מחלקת הtrain וערבבתי את התיקייה כדי שלא יווצר הבדל בין הepochs בהרצה. שילוב שני התיקיות(train, train_aug) שמור במשתנה בשם train_raw.

על כל תיקייה שהעלתי אני ביצעתי נורמליזציה, הפעלתי את פעולת הpreprocess ופירקתי את הבאצ'ים כדי שכל פיסת מידע תהיה יחידה עצמאית.

```python
def split_images_labels(dataset):
    images = dataset.map(lambda x, y: x)
    labels = dataset.map(lambda x, y: y)
    return images, labels


train_images, train_labels= split_images_labels(train_raw)
val_images, val_labels= split_images_labels(val_raw)
test_images, test_labels = split_images_labels(test_raw)
```

הפונ' split_images_labels() מיועדת לפרק את ה-datasets המחזיקים את כל אחד מחלקי ה-data של המודל(train, test, validation) לשני datasets שאחד מכיל את כל התמונות ואחד את כל הlabels של התמונות בהתאמה. ומחזירה את שני החלקים האלה.

```
class_weight_dict = {0: 1.0, 1: 1.0}
```

המשתנה הזה הוא משתנה שהסברתי עליו בשלב בנייה ואימון המודל ולאחר שתיקנתי את בעיית ה imbalancing לא היה בו עוד צורך וכששני הערכים של שני ה labels הם שווים ל1 אז למשתנה אין השפעה על המודל, ולכן השארתי אותו כך.

```
def create_sequences_tf(images, labels, sequence_length=8):
    image_sequences = images.window(size=sequence_length, shift=1, drop_remainder=True)
    image_sequences = image_sequences.flat_map(lambda window: window.batch(sequence_length))

    label_sequences = labels.skip(sequence_length - 1)  # So labels match the last frame of each sequence

    dataset = tf.data.Dataset.zip((image_sequences, label_sequences))
    return dataset


train_seq = create_sequences_tf(train_images, train_labels, sequence_length)
val_seq = create_sequences_tf(val_images, val_labels, sequence_length)
test_seq = create_sequences_tf(test_images, test_labels, sequence_length)
```

בקטע הקוד הזה מוגדרת פונקציה שהופכת את הdataset המחולק לתמונות וlabels לsequence של תמונות והlabels שלהם. הוא מקבל את האורך המוגדר מראש ממשתנה שקוראים לו sequence_length שאותו הגדרתי בתחילת הקוד.

מבנה הdataset צריך להיות בצורת sequence בכדי שיתאים למודל מסוג LSTM.

```
train_seq = train_seq.batch(batch_size).prefetch(tf.data.AUTOTUNE)
val_seq = val_seq.batch(batch_size).prefetch(tf.data.AUTOTUNE)
test_seq = test_seq.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

בקטע הקוד הזה עבור כל אחד מחלקי הdataset של המודל שכבר מומרים לsequences הקוד הופך אותם לבאצ'ים כשכל אחד בגודל batch_size המוגדר מראש ובכך מכניס למודל בבת אחת מספר של sequences ובכך חוסך בזמן מחשוב. וכל זה דרך הקריאה לפעולה batch(batch_size)

הפונקציה הנוספת הנקראת על הdatasets שנקראת prefetch(tf.data.Autotune) גורמת לכך שבזמן שהמודל מעבד את את הבאץ' הנוכחי הוא גם מבצע טעינה מוקדמת לבאץ' הבא. ובכך המודל חוסך בזמן ריצה.

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

שורה זו מגדירה את מנגנון הearly stopping שהוא מנגנון המשמש את המודל בזמן האימון ובעזרתו לאחר אורך epochs שקובע המשתמש אם ביצועי הvalidation של המודל לא משתפרים הוא מפסיק את תהליך האימון מוקדם. ובדרך זו עוזר למנוע overfitting.

```python
mobilenet_url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
mobilenet_layer = hub.KerasLayer(mobilenet_url, trainable=False)

mobilenet_feature_dim = 1280

def apply_mobilenet_per_frame(x):
    # x: [batch, time, h, w, c]
    batch_size = tf.shape(x)[0]
    time_steps = tf.shape(x)[1]
    x_reshaped = tf.reshape(x, (-1, image_size_x, image_size_y, channels))
    features = mobilenet_layer(x_reshaped)  # shape: [batch*time, feature_dim]
    features = tf.reshape(features, (batch_size, time_steps, mobilenet_feature_dim))
    return features
```

קטע הקוד הבא מגדיר ומייבא את סוג המודל המאומן מראש אבוא אני משתמש בקוד ואת ממדי ה features שהוא מחזיר, ולאחר מכן יוצר פונקציה שדרושה כדי להתאים את הנתונים לדרישות המודל המאומן מראש.

```python
def display_image_sequence_batch(image_sequence_batch, labels_batch, batch_index=0):
    """
    Displays one image sequence from a batch.
    """
    if batch_index >= image_sequence_batch.shape[0]:
        print(f"Batch index {batch_index} out of bounds for batch size {image_sequence_batch.shape[0]}")
        return

    sequence_to_display = image_sequence_batch[batch_index]
    label_to_display = labels_batch[batch_index]

    print(f"Displaying sequence from batch index {batch_index}, Label: {label_to_display.numpy()}")

    num_frames = sequence_to_display.shape[0]
    fig, axes = plt.subplots(1, num_frames, figsize=(num_frames * 2, 2)) # Adjust figure size as needed

    for i in range(num_frames):
        image_to_plot = tf.image.convert_image_dtype(sequence_to_display[i], tf.float32)
        axes[i].imshow(image_to_plot.numpy())
        axes[i].set_title(f"Frame {i+1}")
        axes[i].axis('off')

        plt.tight_layout()
        plt.show()


for image_seq_batch, label_batch in train_seq.take(1):
    # Call the display function with the first batch
    display_image_sequence_batch(image_seq_batch, label_batch, batch_index=0)
```

קטע הקוד זה מיועד בכדי להראות דוגמא ויזואלית של sequence לפני האימון, הדבר עוזר בהבנה מצידי בעיקר.

```
#CNN & LSTM model with pretrained model
model = models.Sequential([
    layers.Input(shape=(sequence_length, image_size_x, image_size_y, channels)),

    layers.Lambda(
        apply_mobilenet_per_frame,
        output_shape=(sequence_length, mobilenet_feature_dim)
        ),

    layers.LSTM(
        256,
        return_sequences=True,
        activation="relu"
        ),
    layers.Bidirectional(layers.LSTM(128, return_sequences=True,activation="relu")),
    layers.Bidirectional(layers.LSTM(64)),

    layers.Dense(128, activation='relu'),
    layers.Dropout(0.1),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
    ])


# Compile model
model.compile(
    optimizer=optimizer,
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),#
    metrics=['accuracy'],
    run_eagerly=False # eager execution is required for some reson when I ran the LSTM model option
    )


print(model.summary())

# Train model
history = model.fit(
    train_seq,
    validation_data=val_seq,
    epochs=epochs,
    callbacks=early_stopping,
    class_weight=class_weight_dict
    )
```

חלק זה הוא קטע הקוד שבו מוגדרת ארכיטקטורת המודל והקומפילציה בנוסף מתבצעת קריאה להדפסת מבנה המודל בצורת טבלה וקריאה לאימון המודל על הtrain_seq, ביצוע ולידציה בזמן האימון על ה val_seq, הצבת מספר הepochs וערך הclass weights שקבענו קודם לכן ובנוסף לכך קריאה למנגנון הearly stopping.

```python
# Evaluate
test_loss, test_accuracy= model.evaluate(test_seq)
print("Test acuracy:", test_accuracy)


y_true=[]
y_pred=[]
for x_batch , y_batch in test_seq:
    preds=model.predict( x_batch)
    preds_binary =( preds > 0.5).astype(int)
    y_true.extend(y_batch.numpy().astype(int))
    y_pred.extend(preds_binary.flatten())

    #classification Report
    print("Classification Report:")
    print(classification_report(y_true, y_pred, digits=4))

    #confusion matrix
    m= confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=m, display_labels=[0, 1])
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.show()


#Plot results
plt.figure(figsize=(12, 6))
```

קטע הקוד הבא מריץ את המודל המאומן על הtest_seq dataset ובכך מוודא איך הוא מבצע בזמן אמת. ומוציא מהרצה זו את הaccuracy והloss .

על פי החיזויים שעשה המודל על הtest_seq dataset נוצר דוח קלסיפיקציה ומטריצת בלבול שהם דרכים ויזואליים וטקסטואליים שמציגים מידע על ביצועי המודל ובכך עוזרים להבין יותר על איך המודל עובד, עוזרים בזיהוי בעיות ומספקים מדדים נוספים להערכת המודל.

```python
#Accuracy Plot
plt.subplot(1, 2, 1) #1st plot
plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()

#Loss plot
plt.subplot(1, 2, 2) #2nd plot
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()

plt.tight_layout() #preventing overstepping
plt.show()
```

בקטע זה אני יוצר גרפים ויזואלים של הaccuracy (לאורך האימון) של שני הdatasets הרצו במהלך האימון שהם ה train_seq וה val_seq ובנוסף גרף של ערכי הloss (לאורך האימון) על אותם הdatasets.

```
model.save("c:/users/itama/bitcoin prediction/gui/model.keras")
```

בשורה זו אני שומר את המודל המאומן למיקום בזיכרון של המחשב שלי שממנו אוכל להטעין אותו ל gui.

```
def run_model_gui():
```

במחלקה זו קיימת פונקציה נוספת שמיועדת לכפתור אימון המודל שקיים בgui. ההבדל היחיד בין הפונקציות הוא שלאחר שהמודל מאומן, מבצעים את הרצת המודל על הtest_seq ומקבלים את ערכי הaccuracy והloss שלו פשוט מחזירים את הערכים האלו ולא מייצרים תצוגות ויזואליות או טקסטואליות וגם לא שומרים את המודל.

:Main.py

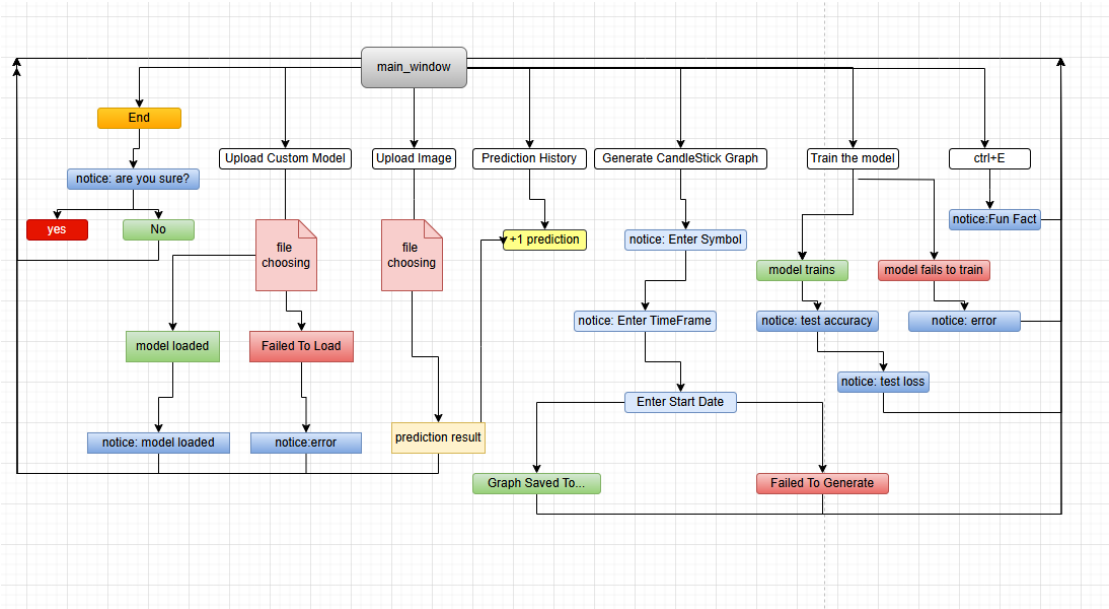
```
from bitcoin_data_creation import (
    main_data_creation,
    devide_dataset,
    generate_single_graph
)
from model import(
    run_model,
    )
```

פה ייבאתי את הפונקציות הנחוצות לי מהמחלקות הנחוצות לי לטובת הרצת קוד ה main שדרכו אני מריץ את הפונקציות המרכזיות במחלקות שבפרוייקט.

```
def main():
    run_model()
if __name__=="__main__":
    main()
```

בקטע קוד זה אני משתמש בכדי להריץ בכל פעם פונקציה שונה. פונקציית הmain מזומנת כפונקצייה המרכזית שמזדמנת(שמתבצעת) בהרצת הקוד.

מדריך למשתמש



מדריך זה מסביר איך ניתן להשתמש בממשק המשתמש שכתבתי עבור הפרוייקט.

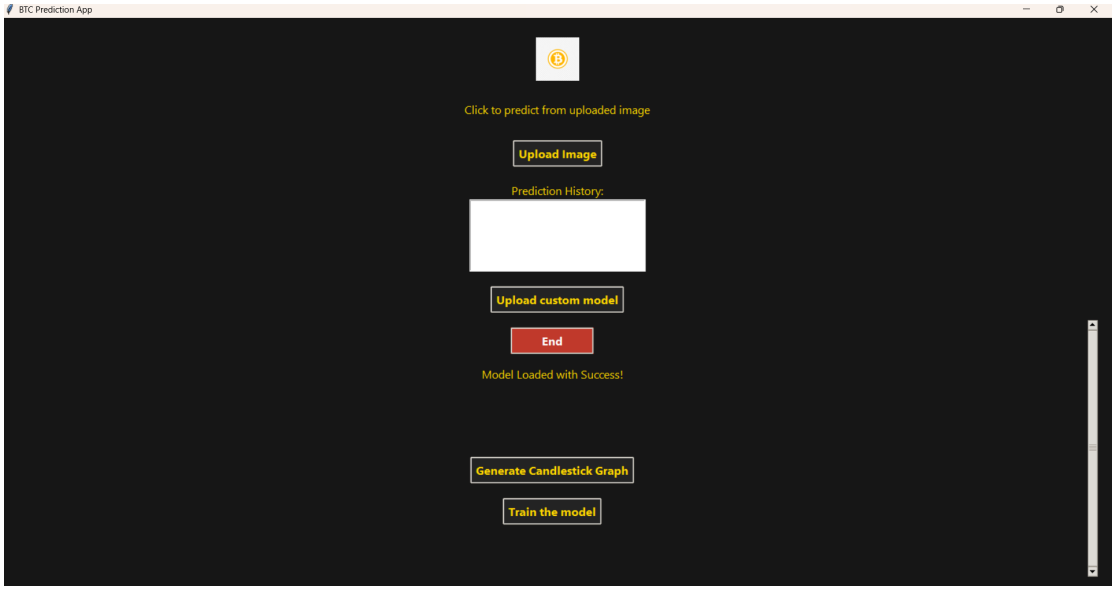
ראשית כדי לפתוח את ממשק המשתמש צריך להריץ את קוד ה gui מהמחשב.

כדי לוודא שלא ייווצרו בעיות בפתיחת ממשק המשתמש כדאי ראשית לוודא שקיים Python על המחשב ובמקרה שלא ניתן להוריד את השפה דרך הקישור הבא:
https://www.python.org/downloads/

בנוסף בכדי להריץ את הקו בצורה נכונה צריך לוודא שהספריות הבאות קיימות על המחשב שלך ולהתקין אותן במידת הצורך בעזרת הקריאות המוצמדות אליהן:

TensorFlow	pip install tensorflow
random	pip install random
tkinter	pip install tkinter
numpy	pip install numpy
PIL	pip install PIL
os	pip install os
Threading	pip install threading

מסך הפתיחה



בחלון הפתיחה של הממשק נמצאים חמשיה כפתורים ושדה שבו נמצאת היסטוריית התחזיות של הממשק.

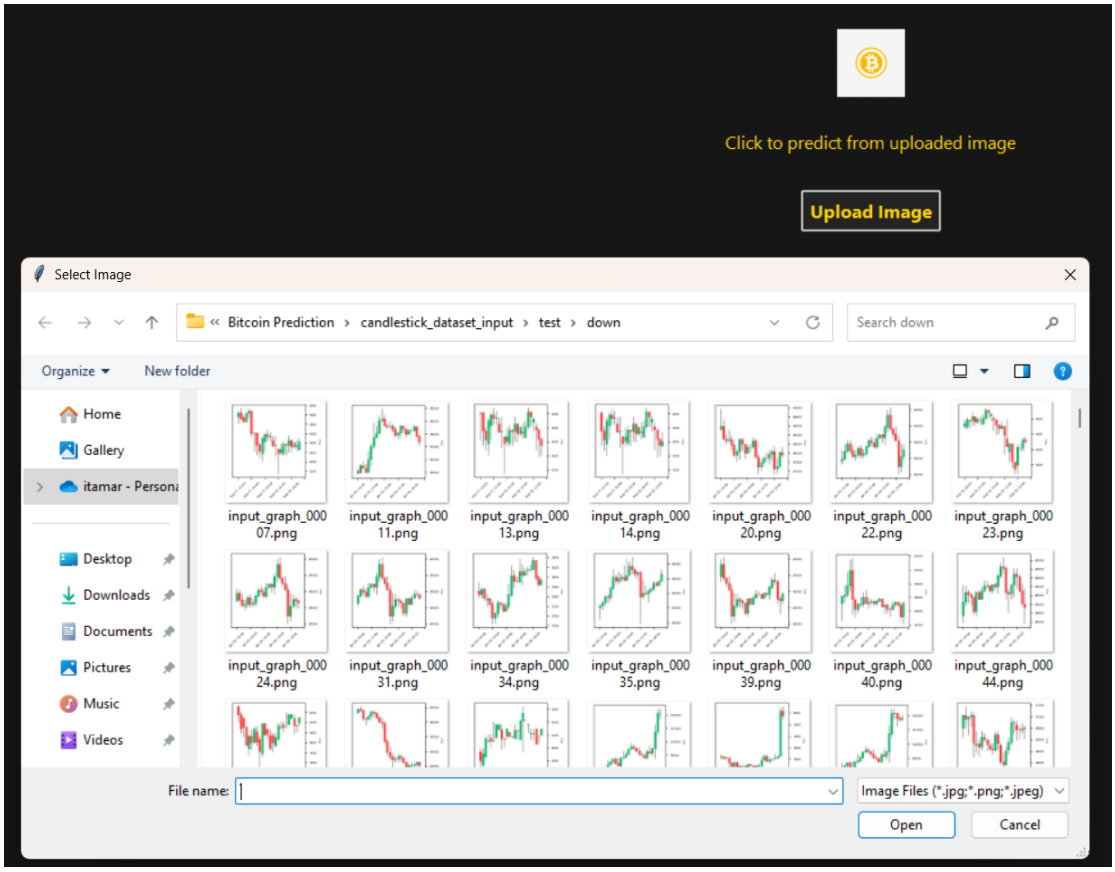
-הכפתור הראשון הוא העלאת תמונה.

-הכפתור השני משמש להעלאת מודל אישי שאיתו יעבוד הממשק.

-הכפתור השלישי הוא כפתור סיום ה-process והוא בעצם סוגר את האפליקציה.

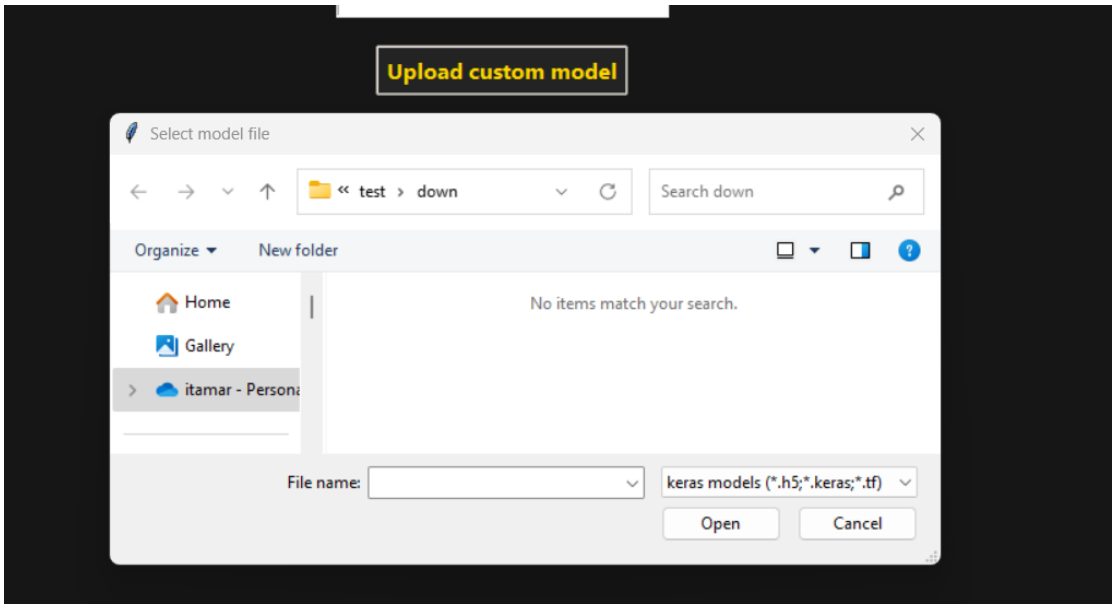
-הכפתור הרביעי יוצר תמונה של גרף מכל שוק שהוא על פי טווח זמנים וזמן התחלה שאתה קובע מראש.

-הכפתור החמישי מפעיל את הרצת המודל.

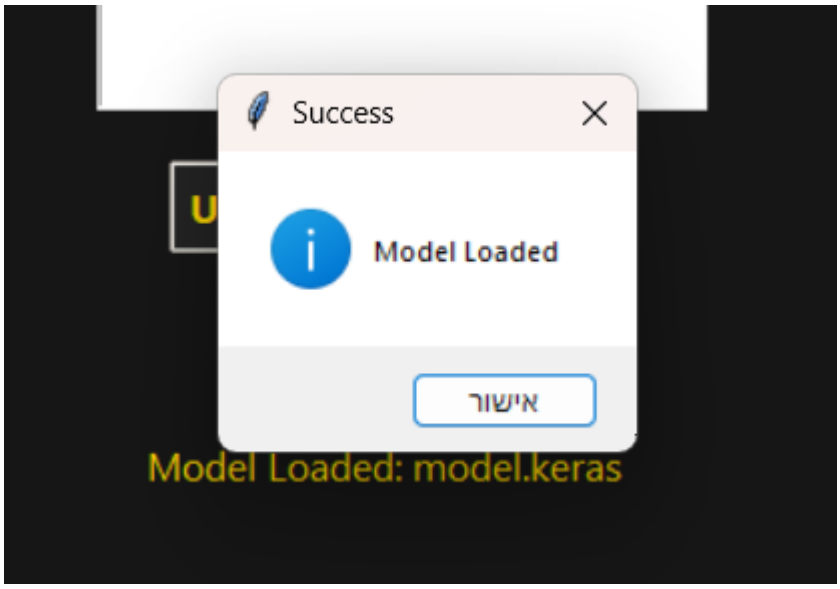


כאשר לוחצים על כפתור הupload image נפתחת חלונית לסייר הקבצים שממנו ניתן לבחור את תמונת הגרף אותה תרצה להעלות.

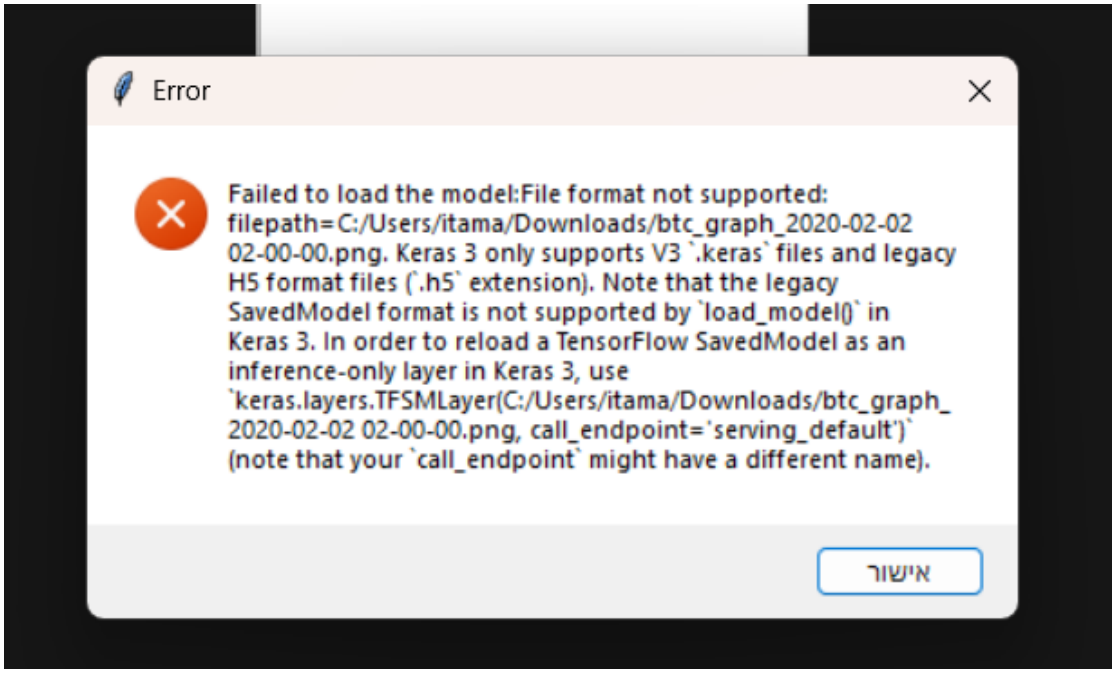
לאחר שמתבצעת העלאה יהיה ניתן לראות את התמונה שהועלתה, את התחזית(עם ביטחון המודל) ובנוסף היסטורית החיזויים מתעדכנת.



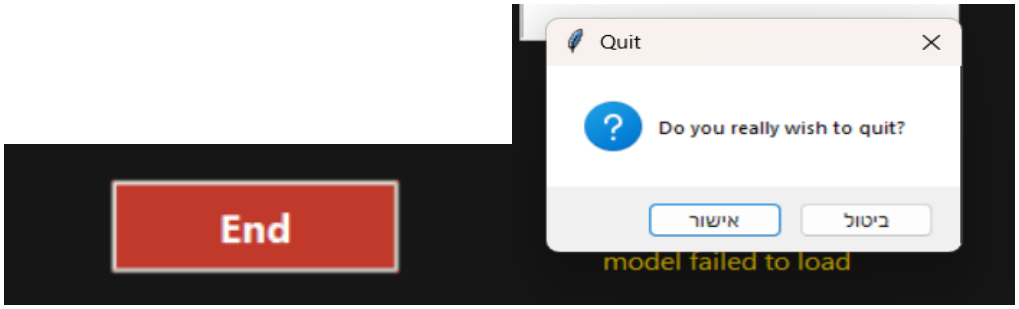
כאשר לוחצים על כפתור העלאת מודל אישי נפתח חלונית של סייר הקבצים שאיתה אפשר לחפש את הקובץ אותו תרצה להעלות לממשק.



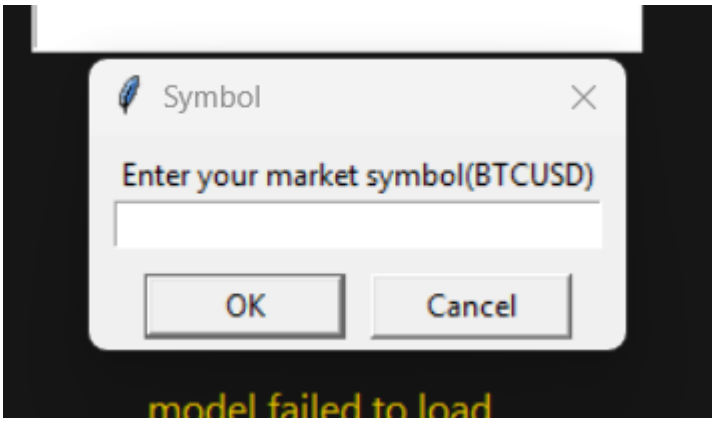
לאחר העלאה מתקבלת ההודעה הזו במקרה שהמודל עלה כמו שצריך.

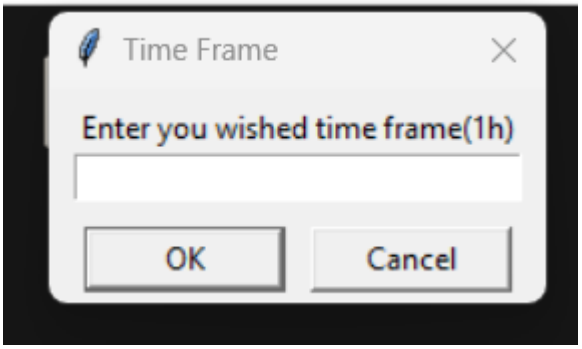


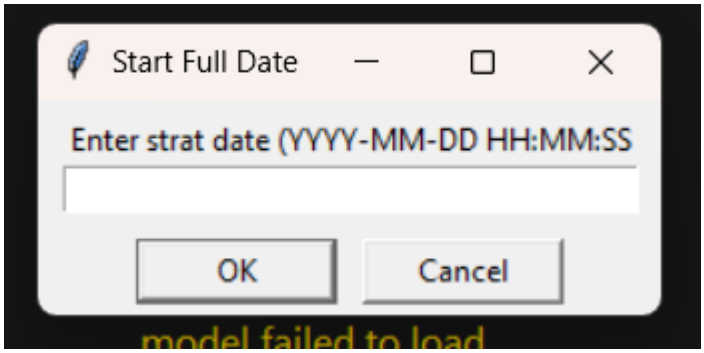
במקרה שהמודל לא עולה כמו שצריך מתקבלת הודעת שגיאה.



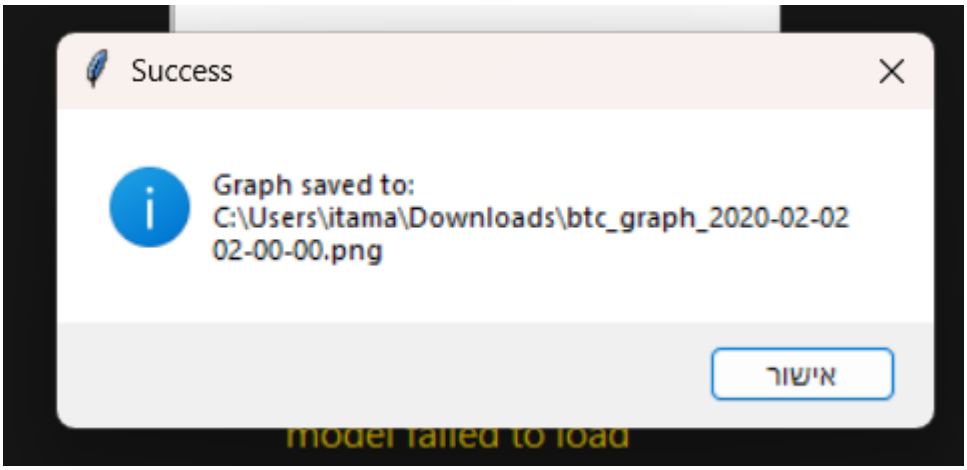
כאשר לוחצים על כפתור ה End מופיעה מודעה השואלת האם אתה בטוח שתרצה לעזוב. בלחיצה על כפתור האישור הממשק נסגר ובמידה שלוחצים על כפתור הביטול חוזרים למסך הרגיל.



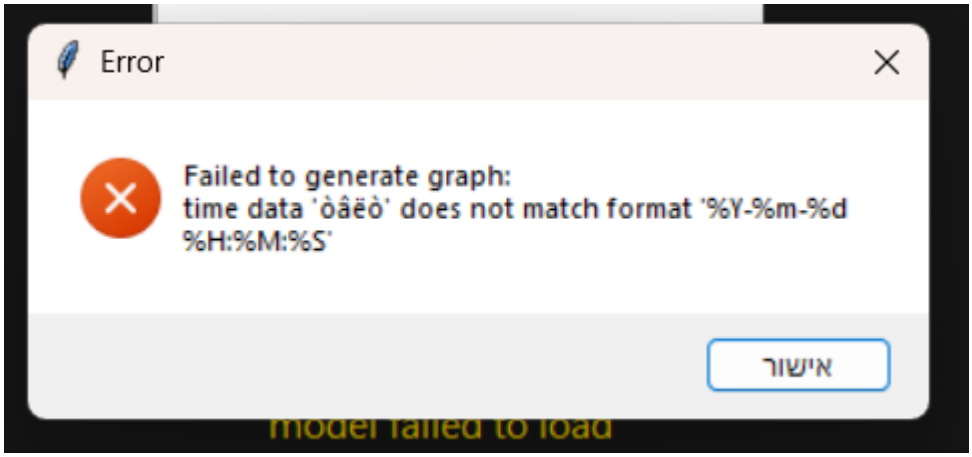




שלושת החלוניות האלו הן החלוניות שדרכן המשתמש מגדיר את הנתונים של הגרף שהוא רוצה ליצור לאחר לחיצה על כפתור ייצור הגרף. כאשר אחד מהשדות מוחזר ריק או נלחץ על כפתור ה cancel הפעולה נעצרת, ולא נשמר שום גרף בזיכרון המחשב של המשתמש.



במקרה והנתונים שהכניס המשתמש תקינים והגרף נשמר בהצלחה מתקבלת ההודעה הזו אשר מצביעה על מיקום הקובץ ומאשרת שהגרף נשמר.



במקרה אחר מופיעה הודעת שגיאה. הודעה זו עלולה לנבוע מבעיות בשמירת הקובץ ומבעיות בנתונים שהכניס המשתמש.

כאשר לוחצים על כפתור אימון המודל, המודל רץ על המחשב ובסיום האימון ושמירת התוצאות הרצויות, מופיעות הודעות על המסך המראות את דיוק המודל שנמדד על נתוני הtest ובנוסף גם את ה- loss הנמדד על נתוני הtest.

במקרה וחלה בעיה במהלך ההרצה מתקבלת הודעת שגיאה.

סיכום אישי - רפלקציה

העבודה על הפרויקט הייתה מאוד מלמדת ומאתגרת עבורי. בחרתי בפרוייקט מאוד מיוחד שהציב לי אתגר ודחף אותי ללמוד ולנסות. למדתי המון דברים דרך הפרויקט, רכשתי ניסיון נוסף בנושא שמרתק אותי ויותר מכך ביצעתי פרויקט שלם שבו פיתחתי מערכת ממשק משתמש וגם כתבתי ספר מה שמספק את הערך המוסף הגדול ביותר.

נושא ה-DL הוא נושא שלמדתי גם לפני הלמידה בהתמחות ולכן באתי עם ידע מקדים אך למרות זאת הפרוייקט הזה לימד אותי הרבה יותר ממה שהצלחתי ללמוד לבדי וקידם את היכולות שלי בנושא בצורה משמעותית.

בתחילת הלמידה למדתי איך להשתמש בספריות וAPI של אפליקציות מסחר כמו Binance לטובת הפרויקט וליצור גרפי נרות אמיתיים בעצמי. ייצור הגרפים היה שלב משמעותי בשבילי בעשייה של הפרויקט ודרש הרבה מחשבה ויצירתיות.

לאחר מכן למדתי קצת על סוגי מודלים שונים ויצא לי להתנסות בהם וללמוד איך עובדת הארכיטקטורה שלהם. המודלים הראשונים נכשלו אבל לאט לאט דרך שינוי המודלים שינוי הפרמטרים, שינוי מבני הנתונים ועוד הצלחתי להגיע למודל הסופי שבו אני מתגאה.

במהלך בניית המודל נתקלתי במספר קשיים טכניים (כמו underfitting, model guessing, unbalanced categories, poor memory allocation ועוד...) שלקחו לי זמן רב ודרשו הרבה עבודה בכדי לפתור אותם, ולמרות שהיה קשה והיה נראה לפעמים כאילו הבעיות שצצו הן בעיות שאי אפשר לפתור אותן, המשכתי לנסות עד שלבסוף הצלחתי להגיע למודל הסופי שלי.

אני מאוד גאה בדרך שעברתי ובפרוייקט שיצרתי ולמרות שאחוזי הדיוק שלו לא הגבוהים ביותר השגתי את המטרה שלי – יצרתי data בניתי ממשק, חקרתי מודלים שונים, נכשלתי, ניסיתי שוב, שאלתי, אימנתי מודל משל עצמי וכתבתי ספר על הפרוייקט. בקיצור פעלתי ממש כמו חוקר Data Scientist אמיתי, כלומר בניתי פרוייקט שלם שכולל כתיבת ספר וממשק משתמש על נושא מסובך ולא ויתרתי גם כשהיה קשה.

הפרוייקט הגביר בי את האהדה לנושא וגרם לי להבין שזה הנושא שבו ארצה לעסוק בעתיד הקרוב והרחוק.

ביבליוגרפיה:

UML Design:

/https://www.drawio.com

code running software:

/https://colab.research.google.com

Online trading platform used for creating the data:

-Binance: https://www.binance.com/en