پروژه تشخیص خواب الودگی رانندگان

درس پردازش تصویر – دکتر علیرضا آکوشیده - دانشکده فنی دانشگاه گیلان – پائیز ۱۴۰۳

مریم مشکین – مرتضی ملائی چافی

- **بخش ۱. مقدمه**

تصادفات رانندگی به عنوان یکی از بزرگ‌ترین معضلات ایمنی در سطح جهانی، سالانه جان میلیون‌ها نفر را می‌گیرد. طبق آمار سازمان بهداشت جهانی، خواب آلودگی و خستگی رانندگان یکی از عوامل کلیدی در بروز این حوادث هستند. رانندگانی که به‌دلیل کم‌خوابی یا خستگی ناشی از رانندگی طولانی‌مدت دچار خواب آلودگی می‌شوند، نه تنها جان خود را در خطر می‌اندازند، بلکه ایمنی دیگر رانندگان و عابران را نیز تهدید می‌کنند. به همین دلیل، تشخیص خواب آلودگی به‌عنوان یک چالش مهم در حوزه ایمنی جاده‌ها مطرح است و نیاز به توسعه سیستم‌های هوشمند و کارآمد برای شناسایی به‌موقع این وضعیت احساس می‌شود.

در این پروژه، ما به طراحی و پیاده‌سازی یک مدل مناسب برای تشخیص خواب آلودگی رانندگان پرداخته‌ایم. این مدل با استفاده از تکنیک‌های پردازش تصویر و تجزیه و تحلیل وضعیت چشم رانندگان، به شناسایی علائم خواب آلودگی می‌پردازد. ما با انتخاب یکی از دیتاست‌های معتبر و مقایسه عملکرد مدل خود با دیگر الگوریتم‌های موجود، موفق به دستیابی به دقت بالاتری در تشخیص خواب آلودگی شده‌ایم. این نتایج نشان‌دهنده کارایی و قابلیت اعتماد بالای مدل ما در شناسایی خواب آلودگی است و می‌تواند به‌عنوان ابزاری مؤثر در بهبود ایمنی رانندگی و کاهش حوادث ناشی از خواب آلودگی مورد استفاده قرار گیرد. در ادامه، جزئیات فنی پروژه و نتایج به‌دست‌آمده را بررسی خواهیم کرد تا اهمیت و کارایی این سیستم را به‌طور کامل تبیین کنیم.

دیتاست DDD (Driver Drowsiness Detection) یکی از مجموعه‌های داده معتبر و گسترده در زمینه تشخیص خواب آلودگی رانندگان است که به‌طور خاص برای پژوهش‌های مرتبط با ایمنی جاده و تحلیل رفتار رانندگان طراحی شده است. این دیتاست شامل تصاویر و ویدئوهای متنوعی از رانندگان در شرایط مختلف خواب آلودگی و بیداری است و به محققان این امکان را می‌دهد که الگوریتم‌های خود را برای شناسایی و تشخیص خواب آلودگی بهینه‌سازی کنند. در پروژه ما، از دیتاست DDD به‌عنوان منبع اصلی داده استفاده شده است، زیرا تنوع و حجم بالای داده‌ها، همراه با ویژگی‌های کلیدی مانند وضعیت چشم، حرکات صورت و دیگر نشانه‌های رفتاری، امکان توسعه و ارزیابی دقیق مدل‌های یادگیری

ماشین را فراهم می‌کند. انتخاب این دیتاست به ما کمک کرده است تا دقت و کارایی مدل خود را در تشخیص خواب آلودگی رانندگان بهبود بخشیم و نتایج قابل اعتمادی ارائه دهیم.

در این پروژه، از مدل YOLOv8 (You Only Look Once version 8) به‌عنوان الگوریتم اصلی برای تشخیص خواب آلودگی رانندگان استفاده شده است .YOLOv8 یکی از پیشرفته‌ترین و سریع‌ترین مدل‌های تشخیص شیء در حوزه بینایی ماشین است که به‌طور خاص برای شناسایی و طبقه‌بندی اشیاء در زمان واقعی طراحی شده است. این مدل به دلیل ساختار خاص خود، قادر است به‌صورت همزمان چندین شیء را در یک تصویر شناسایی کند و به‌طور چشمگیری سرعت و دقت تشخیص را افزایش دهد. انتخاب YOLOv8 برای پروژه ما به این دلیل بوده است که این مدل نه تنها دقت بالایی در شناسایی ویژگی‌های کلیدی مانند وضعیت چشم و حرکات صورت دارد، بلکه زمان پردازش بسیار کمتری نسبت به سایر الگوریتم‌های مشابه ارائه می‌دهد. این ویژگی‌ها به ما این امکان را می‌دهد که در شرایط واقعی و در زمان واقعی، خواب آلودگی رانندگان را به‌طور مؤثر و کارآمد تشخیص دهیم، که در نهایت به بهبود ایمنی جاده‌ها کمک می‌کند.

در این پروژه، ابتدا مدل‌های تشخیص خواب آلودگی را بدون اعمال هیچ‌گونه روش پردازش تصویری آزمایش کردیم تا عملکرد پایه آن‌ها را ارزیابی کنیم. این مرحله به ما کمک کرد تا دقت و کارایی مدل‌ها را در شرایط اولیه و بدون پیش‌پردازش داده‌ها بررسی کنیم. سپس، در مرحله بعدی، متدهای مختلف پردازش تصویر، از جمله افزایش کنتراست، حذف نویز و بهبود وضوح تصاویر، به فرآیند تشخیص اضافه شد. این کار به ما این امکان را داد که تأثیر مثبت پیش‌پردازش بر دقت و عملکرد مدل‌ها را مشاهده کنیم. نتایج نشان داد که با استفاده از تکنیک‌های پردازش تصویر، دقت تشخیص خواب آلودگی به‌طور قابل توجهی افزایش یافته و مدل‌ها توانسته‌اند ویژگی‌های کلیدی را بهتر شناسایی کنند، که در نهایت به بهبود عملکرد کلی سیستم کمک کرد.

در مدلYOLO ، سه پارامتر کلیدی به نام‌های image size، batch size و epoch وجود دارد که هر یک تأثیر مهمی بر فرآیند آموزش و عملکرد مدل دارند. Image size به ابعاد تصاویر ورودی اشاره دارد که به مدل داده می‌شود؛ انتخاب اندازه مناسب می‌تواند تأثیر قابل توجهی بر دقت و سرعت پردازش مدل داشته باشد، زیرا اندازه‌های بزرگ‌تر معمولاً اطلاعات بیشتری را در بر می‌گیرند اما زمان پردازش را نیز افزایش می‌دهند. Batch size به تعداد تصاویری که در هر مرحله از آموزش به مدل ارائه می‌شود، اشاره دارد؛ مقادیر بزرگ‌تر می‌توانند به بهبود ثبات در به‌روزرسانی وزن‌ها کمک کنند، اما نیاز به حافظه بیشتری دارند. در نهایت، epoch به تعداد دفعاتی که کل مجموعه داده برای آموزش مدل استفاده می‌شود، اشاره دارد؛ افزایش تعداد epochs می‌تواند به مدل کمک کند تا بهتر یاد بگیرد، اما اگر بیش از حد باشد، ممکن است منجر به overfitting شود، جایی که مدل به داده‌های آموزشی بیش از حد وابسته می‌شود و در داده‌های جدید عملکرد ضعیفی دارد. تنظیم بهینه این پارامترها برای دستیابی به بهترین عملکرد در تشخیص خواب آلودگی رانندگان بسیار حیاتی است.

- **بخش ۲ . مدل ها بدون انجام عمل پیش پردازش**

در این قسمت ، به توضیح مدل پیاده سازی شده بدون انجام پیش پردازش بر روی دادگان می پردازیم . کلیه کد های توصیف شده در این بخش در ریپازیتوری پروژه موجود و قابل دسترس هستند. (در فولدر models_without_augmentations می توانید به این نوت بوک ها دسترسی داشته باشید.)

```
!pip install ultralytics
from google.colab import files
from ultralytics import YOLO
import zipfile
import zipfile
import os
import random
import shutil
```

این کد برای آماده سازی محیط YOLO طراحی شده است. ابتدا کتابخانه UltraLytics نصب و ماژول YOLO وارد می شود. سپس ابزارهایی برای آپلود فایل ها در کولب ، مدیریت فایل های ZIP ، دسترسی به سیستم فایل ، تولید مقادیر تصادفی و انتقال یا حذف فایل ها جهت اماده سازی داده ها برای آموزش یا ارزیابی مدل استفاده می شوند.

```
files.upload()

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d ismailnasri20/driver-drowsiness-dataset-ddd

dataset_zip = 'driver-drowsiness-dataset-ddd.zip'
with zipfile.ZipFile(dataset_zip, 'r') as zip_ref:
    zip_ref.extractall('./driver_drowsiness_dataset')
```

این بخش برای دانلود و استخراج دیتاست از Kaggle استفاده می شود . ابتدا فایل Kaggle.json برای احراز هویت در Kaggle آپلود و تنظیم می شود. سپس دیتاست خواب الودگی راننده دانلود و فایل ZIP مربوطه باز شده و محتوای آن در پوشه ای به نام driver_drowsiness_dataset استخراج می شود.

```
[ ] base_dir = './driver_drowsiness_dataset/Driver Drowsiness Dataset (DDD)'

    train_dir = './data_split/train'
    val_dir = './data_split/val'
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)

    labels = {'Drowsy': 0, 'Non Drowsy': 1}
```

در این بخش ، مسیرهای مربوط به دیتاست و پوشه های تقسیم بندی داده ها را تنظیم می کند. متغیر base_dir مسیر اصلی دیتاست خواب آلودگی راننده را مشخص می کند. سپس دو پوشه train و val برای داده های آموزشی و اعتبارسنجی در مسیر data_split ایجاد می شوند. همچنین دیکشنری labels برچسب های داده ها را تعریف می کند : Drowsy (خواب آلود) با مقدار ۰ و Non Drowsy (غیر خواب آلود) با مقدار ۱

```python
def balance_classes(base_dir, train_dir, val_dir, labels):
    """
    Equalize the number of samples from each class and split them into train and validation sets.

    Parameters:
        base_dir (str): Path to the base dataset directory containing labeled data.
        train_dir (str): Path to the directory where training data will be stored.
        val_dir (str): Path to the directory where validation data will be stored.
        labels (dict): A dictionary where the key is the label name and the value is the label id.
    """
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)
```

```python
    min_class_size = float('inf')
    for label_name in labels.keys():
        label_dir = os.path.join(base_dir, label_name)
        all_files = os.listdir(label_dir)
        min_class_size = min(min_class_size, len(all_files))
```

```python
    for label_name, label_value in labels.items():
        label_dir = os.path.join(base_dir, label_name)
        all_files = os.listdir(label_dir)
        subset_files = random.sample(all_files, min_class_size)
        train_size = int(0.8 * len(subset_files))
        train_files = subset_files[:train_size]
        val_files = subset_files[train_size:]
        train_label_dir = os.path.join(train_dir, label_name.lower())
        val_label_dir = os.path.join(val_dir, label_name.lower())
        os.makedirs(train_label_dir, exist_ok=True)
        os.makedirs(val_label_dir, exist_ok=True)
```

```python
            for file_name in train_files:
                shutil.copy(os.path.join(label_dir, file_name), os.path.join(train_label_dir, file_name))

                label_file = os.path.splitext(file_name)[0] + '.txt'
                with open(os.path.join(train_label_dir, label_file), 'w') as f:
                    f.write(f"{label_value} 0.5 0.5 1.0 1.0\n")

            for file_name in val_files:
                shutil.copy(os.path.join(label_dir, file_name), os.path.join(val_label_dir, file_name))

                label_file = os.path.splitext(file_name)[0] + '.txt'
                with open(os.path.join(val_label_dir, label_file), 'w') as f:
                    f.write(f"{label_value} 0.5 0.5 1.0 1.0\n")

        print("Data balanced and split into train and validation sets.")

    balance_classes(base_dir, train_dir, val_dir, labels)
```

این کد برای آماده‌سازی ساختار فایل‌ها و برچسب‌گذاری دیتاست خواب‌آلودگی راننده طراحی شده است. ابتدا مسیر دیتاست اصلی با متغیر base_dir تعریف می‌شود. سپس دو پوشه train و val برای تقسیم داده‌ها به مجموعه‌های آموزشی و اعتبارسنجی ایجاد می‌شوند. اگر این پوشه‌ها از قبل وجود نداشته باشند، با استفاده از os.makedirs ساخته می‌شوند. دیکشنری labels برچسب‌های دودویی برای طبقه‌بندی داده‌ها را مشخص می‌کند: Drowsy با مقدار ۰ نشان‌دهنده حالت خواب‌آلودگی و Non Drowsy با مقدار ۱ نشان‌دهنده حالت هوشیاری است. این ساختار برای آماده‌سازی داده‌ها جهت آموزش مدل استفاده می‌شود.

```
[ ]  train_data_path = './data_split/data.yaml'
     with open(train_data_path, 'w') as f:
         f.write(
             f"""
             path: {os.path.abspath('./data_split')}
             train: {os.path.abspath(train_dir)}
             val: {os.path.abspath(val_dir)}
             nc: 2
             names: ['drowsy', 'non_drowsy']
             weights: [3.0, 3.0]
             """
         )

     model = YOLO('yolov8n.pt')
     model.train(data=train_data_path, epochs=1, imgsz=128, batch=32)
```

در این بخش، ما ابتدا یک فایل YAML به نام data.yaml ایجاد می‌کنیم که شامل مسیرهای مربوط به داده‌های آموزشی و اعتبارسنجی، تعداد کلاس‌ها و نام‌های آن‌ها است. با استفاده از os.path.abspath() ، مسیرهای مطلق دایرکتوری‌ها را به دست می‌آوریم و درون فایل می‌نویسیم .سپس، با استفاده از مدلYOLO ، ما مدل را با بارگذاری وزن‌های پیش‌آموزش دیده (yolov8n.pt) فراخوانی کرده و آن را برای یک دوره آموزشی (epochs=1) با اندازه تصویر ۱۲۸ و اندازه بچ ۳۲ آموزش می‌دهیم. این مراحل به ما کمک می‌کند تا مدل شناسایی خواب آلودگی رانندگان را آموزش دهیم.

```
[ ]  metrics = model.val()
```

در این خط کد، ما از مدل YOLO که قبلاً آموزش داده‌ایم، برای ارزیابی عملکرد آن استفاده می‌کنیم. با فراخوانی تابع model.val()، ما متریک‌های مربوط به اعتبارسنجی مدل را به دست می‌آوریم. این متریک‌ها معمولاً شامل معیارهایی مانند دقت(Precision) ، یادآوری(Recall) ، میانگین دقت (mAP) و سایر معیارهای مرتبط با عملکرد مدل در شناسایی و دسته‌بندی داده‌ها هستند. این ارزیابی به ما کمک می‌کند تا کیفیت آموزش مدل را بررسی کنیم و ببینیم که آیا مدل به خوبی بر روی داده‌های اعتبارسنجی عمل می‌کند یا خیر.

```
precision = metrics.results_dict['metrics/precision(B)']
recall = metrics.results_dict['metrics/recall(B)']

f1_score = 2 * (precision * recall) / (precision + recall + 1e-6)
print("F1-Score:", f1_score)
print("Precision:", precision)
print("Recall:", recall)
```

در این بخش از کد، ما به استخراج مقادیر دقت (Precision) و یادآوری (Recall) از نتایج ارزیابی مدل می‌پردازیم. ابتدا، با استفاده از metrics.results_dict['metrics/precision(B)'] و metrics.results_dict['metrics/recall(B)']، مقادیر دقت و یادآوری را از دیکشنری نتایج به دست می‌آوریم. سپس، برای محاسبه نمرهF1 ، از فرمول زیر استفاده می‌کنیم:

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall + 1e-6}$$

در این فرمول، مقدار بسیار کوچک 1e−6 به منظور جلوگیری از تقسیم بر صفر در صورتی که دقت و یادآوری هر دو صفر باشند، اضافه شده است. در نهایت، ما مقادیرF1-Score ، دقت و یادآوری را چاپ می‌کنیم تا عملکرد مدل در شناسایی خواب آلودگی رانندگان را بررسی کنیم .این مقادیر به ما کمک می‌کنند تا ارزیابی دقیقی از کیفیت مدل داشته باشیم.

- **بخش ۳. نتایج حاصل از اجرای مدل بر روی دیتاست ها**

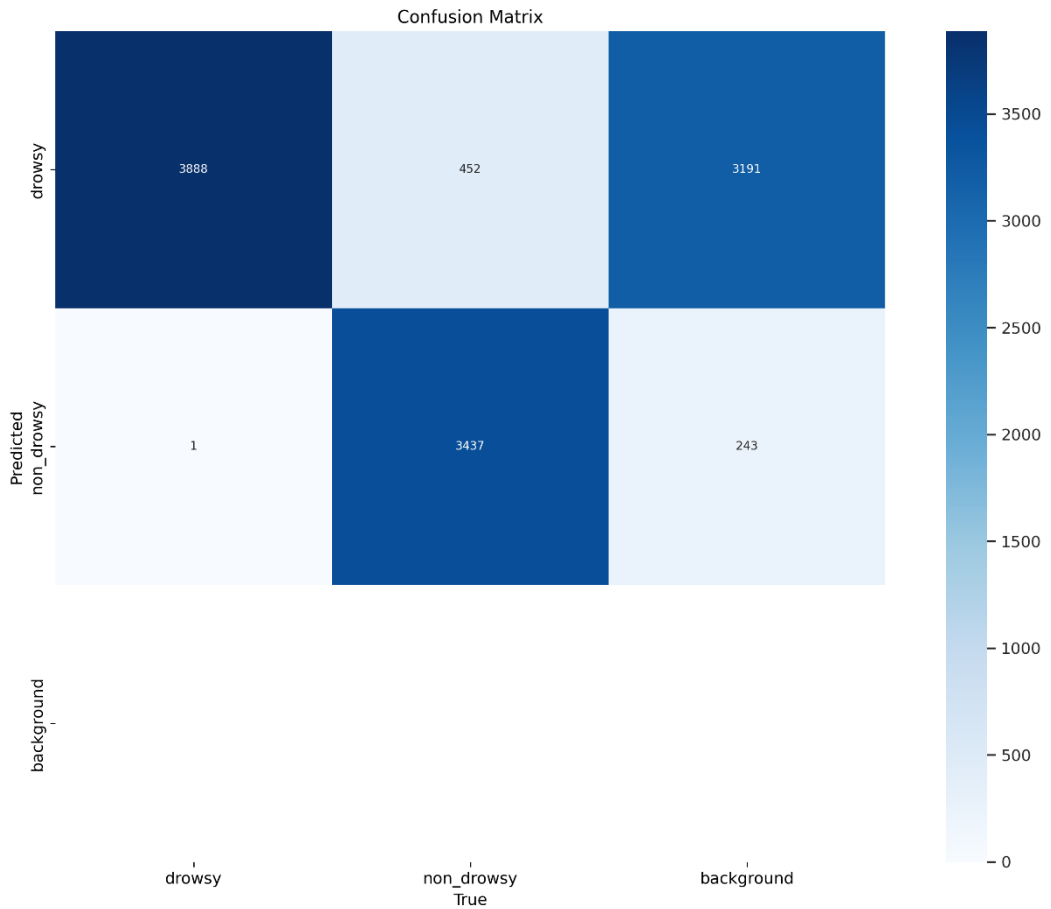
در جدول زیر ، مقادیر f1-socre ، precision و recall حاصل از اجرای مدل ها بر روی دیتاست موجود نمایش داده شده است. کلیه این مقادیر ، از بخش تست و ارزیابی انتهای مدل ها برداشته شده است که قابل مشاهده می باشد.

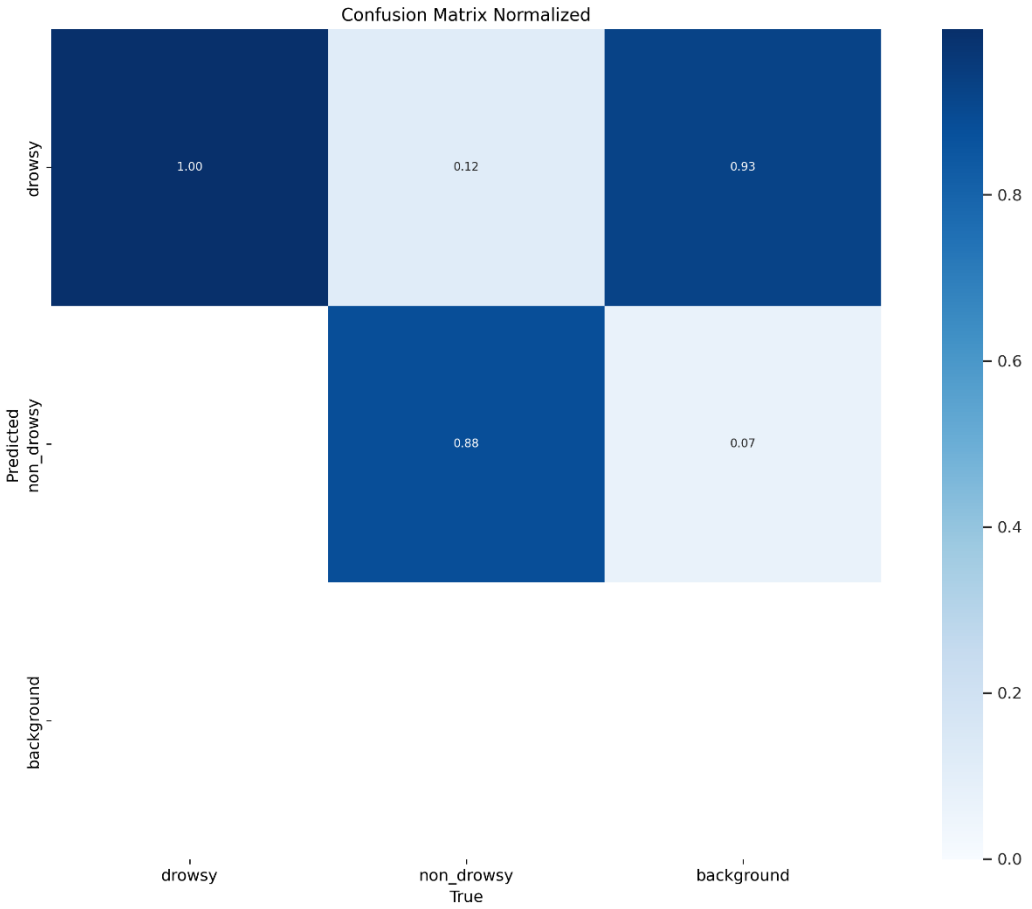
	f1-score	precision	recall
imgsz_128_batch_24	0.956	0.946	0.967
imgsz_128_batch_32	0.932	0.926	0.938
imgsz_224_batch_24	0.980	0.984	0.976
imgsz_224_batch_32	0.877	0.839	0.917

در ادامه برخی نمودارها و جداول حاصل نمایش داده شده است.

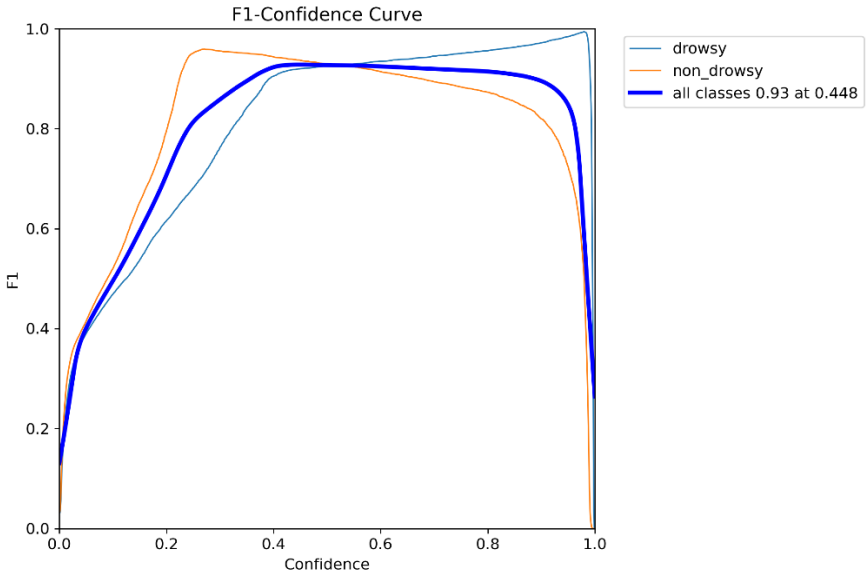
برای مدل با image size = 128 و batch size = 24 :

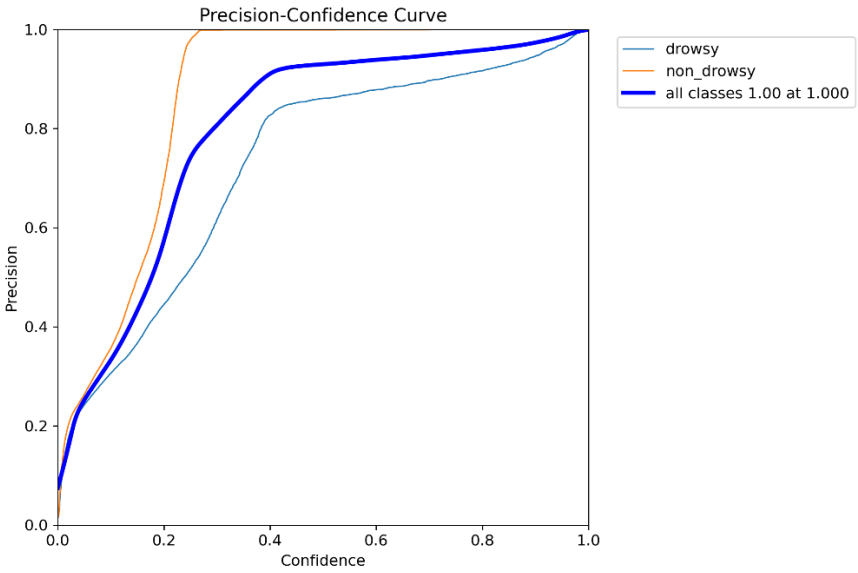
- جدول درهم آمیختگی به همراه نسخه نرمال سازی شده آن:

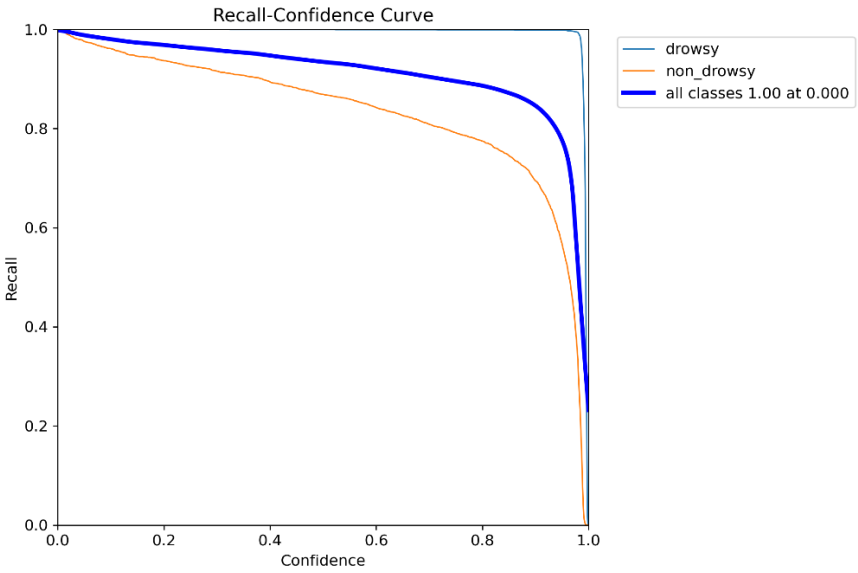




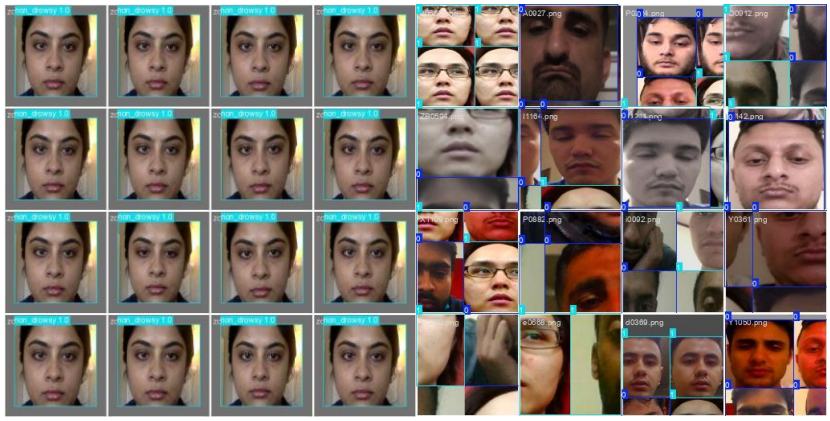
- Curve های مرتبط با f1-score و precision و recall:







- نمونه ای از دسته های train و validation:



برای بقیه مدل ها نیز ، جداول درهم آمیختگی ، کِرو های مربوطه ، نمونه ای از دسته های train و validation به اضافه سایر دیتاهای مدل در ریپازیتوری پروژه قابل دسترس میباشند.

- **بخش ۴ . مدل ها همراه با انجام پیش پردازش بر روی دادگان:**

در این قســـمت ، کدهای مدلهای بدون پیش پردازش را ریفکتور کرده و توابع مربوط به انجام پیش پردازش بر روی تصـــاویر دیتاست را اضافه نموده ایم. همچون بخش ۲ ، در این قسمت نیز توصیف کلیه توابع مورد استفاده به تفصیل توضیح داده شده است.

```python
import os
import shutil

# do not run this cell on every run.

def reset_runtime_directory(runtime_path):
    """
    Deletes all files and folders in the specified runtime directory
    and recreates the directory.

    Parameters:
        runtime_path (str): Path to the runtime directory.
    """
    # Check if the path exists
    if os.path.exists(runtime_path):
        # Delete all files and folders in the directory
        shutil.rmtree(runtime_path)
        print(f"Deleted all files and folders in: {runtime_path}")
    else:
        print(f"Path does not exist: {runtime_path}")

    # Recreate the runtime directory
    os.makedirs(runtime_path, exist_ok=True)
    print(f"Recreated the runtime directory: {runtime_path}")

runtime_path = "/content"
reset_runtime_directory(runtime_path)
```

در کد بالا، ما یک تابع به نام reset_runtime_directory تعریف کرده‌ایم که وظیفه‌اش پاک‌سازی و بازسازی یک دایرکتوری مشخص است. این تابع ابتدا بررسی می‌کند که آیا دایرکتوری مورد نظر (که در پارامتر runtime_path مشخص شده) وجود دارد یا خیر؛ اگر وجود داشته باشد، تمام فایل‌ها و پوشه‌های آن را با استفاده از shutil.rmtree حذف می‌کند. سپس دایرکتوری را دوباره با os.makedirs ایجاد می‌کند و با گزینه exist_ok=True از بروز خطا در صورت وجود قبلی دایرکتوری جلوگیری می‌کند. در نهایت، پیام‌هایی به کنسول چاپ می‌شود تا کاربر از وضعیت عملیات مطلع شود. این تابع در انتها با مسیر /content فراخوانی می‌شود تا دایرکتوری مربوطه را ریست کند.

```python
import os
import zipfile
from google.colab import files

def download_and_extract_kaggle_dataset():
    files.upload()

    !mkdir -p ~/.kaggle
    !cp kaggle.json ~/.kaggle/
    !chmod 600 ~/.kaggle/kaggle.json

    !kaggle datasets download -d ismailnasri20/driver-drowsiness-dataset-ddd

    dataset_zip = 'driver-drowsiness-dataset-ddd.zip'
    with zipfile.ZipFile(dataset_zip, 'r') as zip_ref:
        zip_ref.extractall('./driver_drowsiness_dataset')

# Example usage
download_and_extract_kaggle_dataset()
```

در کد بالا، ما تابعی به نام download_and_extract_kaggle_dataset تعریف کرده‌ایم که برای دانلود و استخراج یک دیتاست از کگل طراحی شده است. ابتدا ما از کاربر می‌خواهیم که فایل kaggle.json را بارگذاری کند تا اطلاعات احراز هویت برای دسترسی به API کگل را فراهم کنیم. سپس یک دایرکتوری به نام kaggle. در دایرکتوری خانگی ایجاد کرده و فایل احراز هویت را به آنجا منتقل و مجوزهای لازم را تنظیم می‌کنیم. پس از آن، دیتاست مورد نظر را با استفاده از دستور کگل دانلود کرده و در نهایت، فایل ZIP آن را استخراج کرده و محتویاتش را در دایرکتوری ./driver_drowsiness_dataset قرار می‌دهیم. با فراخوانی این تابع، تمامی مراحل دانلود و استخراج به طور خودکار انجام می‌شود.

```python
import os

def setup_directories_and_labels(base_dir, train_dir, val_dir):
    """
    Sets up directories for training and validation datasets and initializes labels.

    Parameters:
        base_dir (str): Base directory containing the dataset.
        train_dir (str): Directory to store training data.
        val_dir (str): Directory to store validation data.

    Returns:
        dict: A dictionary containing labels.
    """
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)

    labels = {'Drowsy': 0, 'Non Drowsy': 1}
    return labels

base_dir = './driver_drowsiness_dataset/Driver Drowsiness Dataset (DDD)'
train_dir = './data_split/train'
val_dir = './data_split/val'

labels = setup_directories_and_labels(base_dir, train_dir, val_dir)
print("Labels:", labels)
```

در کد بالا، ما تابعی به نام setup_directories_and_labels تعریف کرده‌ایم که برای راه‌اندازی دایرکتوری‌های مربوط به دیتاست‌های آموزشی و اعتبارسنجی و همچنین ایجاد برچسب‌ها طراحی شده است. ابتدا، ما دایرکتوری‌های مربوط به داده‌های آموزشی و اعتبارسنجی را با استفاده از os.makedirs ایجاد می‌کنیم و در صورت وجود، از ایجاد دوباره آن‌ها جلوگیری می‌کنیم . سپس یک دیکشنری برای برچسب‌ها تعریف می‌کنیم که شامل برچسب‌های "Drowsy" (خواب‌آلود) و "Non Drowsy" (غیر خواب‌آلود) است و مقادیر عددی ۰ و ۱ را به آن‌ها اختصاص می‌دهیم. در نهایت، با فراخوانی این تابع و مشخص کردن مسیرهای مربوطه، برچسب‌ها را دریافت کرده و آن‌ها را چاپ می‌کنیم.

```python
def _copy_files_with_labels(source_dir, file_list, target_dir, label_value):
    """
    Copies files and creates corresponding label files.

    Parameters:
        source_dir (str): Source directory of the files.
        file_list (list): List of file names to copy.
        target_dir (str): Target directory for the copied files.
        label_value (int): The label value to write in the label file.
    """
    for file_name in file_list:
        # Copy the image file
        shutil.copy(os.path.join(source_dir, file_name), os.path.join(target_dir, file_name))

        # Create a corresponding label file
        label_file = os.path.splitext(file_name)[0] + '.txt'
        label_file_path = os.path.join(target_dir, label_file)
        with open(label_file_path, 'w') as f:
            f.write(f"{label_value} 0.5 0.5 1.0 1.0\n")
```

در کد بالا، ما تابعی به نام copy_files_with_labels_ تعریف کرده‌ایم که وظیفه‌اش کپی کردن فایل‌ها و ایجاد فایل‌های برچسب مربوطه است. این تابع ابتدا یک دایرکتوری مبدا(source_dir) ، لیستی از نام فایل‌ها(file_list) ، دایرکتوری مقصد (target_dir) و مقدار برچسب (label_value) را به عنوان ورودی دریافت می‌کند. سپس برای هر نام فایل در لیست، فایل تصویر را از دایرکتوری مبدا به دایرکتوری مقصد کپی می‌کند. علاوه بر این، یک فایل متنی برچسب با همان نام فایل تصویر (با پسوند txt.) ایجاد می‌کند و در آن مقدار برچسب را به همراه مقادیر ثابتی برای مختصات نوشتاری ذخیره می‌کند. این کار به ما کمک می‌کند تا فایل‌های تصویر و برچسب‌های مربوط به آن‌ها را به طور منظم و هماهنگ در دایرکتوری مقصد ذخیره کنیم.

```python
import os
import random
import shutil

def balance_classes(base_dir, train_dir, val_dir, labels):
    """
    Equalizes the number of samples from each class and splits them into train and validation sets.

    Parameters:
        base_dir (str): Path to the base dataset directory containing labeled data.
        train_dir (str): Path to the directory where training data will be stored.
        val_dir (str): Path to the directory where validation data will be stored.
        labels (dict): A dictionary where the key is the label name and the value is the label id.
    """
    # Ensure train and validation directories exist
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)

    # Determine the minimum class size
    min_class_size = min(
        len(os.listdir(os.path.join(base_dir, label_name)))
        for label_name in labels.keys()
    )

    # Process each class
    for label_name, label_value in labels.items():
        label_dir = os.path.join(base_dir, label_name)
        all_files = os.listdir(label_dir)

        # Balance the class by sampling a subset
        subset_files = random.sample(all_files, min_class_size)
```

```
        # Split into train and validation sets
        train_size = int(0.8 * len(subset_files))
        train_files = subset_files[:train_size]
        val_files = subset_files[train_size:]

        # Create subdirectories for the label
        train_label_dir = os.path.join(train_dir, label_name.lower())
        val_label_dir = os.path.join(val_dir, label_name.lower())
        os.makedirs(train_label_dir, exist_ok=True)
        os.makedirs(val_label_dir, exist_ok=True)

        # Copy files and create label files for training data
        _copy_files_with_labels(label_dir, train_files, train_label_dir, label_value)

        # Copy files and create label files for validation data
        _copy_files_with_labels(label_dir, val_files, val_label_dir, label_value)

    print("Data balanced and split into train and validation sets.")

balance_classes(base_dir, train_dir, val_dir, labels)
```

در کد بالا، ما تابعی به نام balance_classes تعریف کرده‌ایم که هدف آن برابر کردن تعداد نمونه‌ها از هر کلاس و تقسیم آن‌ها به مجموعه‌های آموزشی و اعتبارسنجی است. ابتدا، ما اطمینان حاصل می‌کنیم که دایرکتوری‌های مربوط به داده‌های آموزشی و اعتبارسنجی وجود دارند. سپس، با بررسی اندازه کوچک‌ترین کلاس، تعداد نمونه‌ها را برای هر کلاس تعیین می‌کنیم. برای هر کلاس، ما فایل‌های موجود را لیست کرده و یک زیرمجموعه تصادفی به اندازه کلاس کوچک‌تر انتخاب می‌کنیم. این زیرمجموعه به دو بخش آموزشی و اعتبارسنجی تقسیم می‌شود، به طوری که 80% به داده‌های آموزشی و 20% به داده‌های اعتبارسنجی اختصاص می‌یابد. سپس، دایرکتوری‌های مربوط به هر کلاس در مجموعه‌های آموزشی و اعتبارسنجی ایجاد شده و با استفاده از تابع _copy_files_with_labels، فایل‌ها و فایل‌های برچسب مربوطه به این دایرکتوری‌ها کپی می‌شوند. در پایان، یک پیام مبنی بر اینکه داده‌ها متعادل و تقسیم شده‌اند، چاپ می‌شود.

```
import cv2

def _process_and_save_image(file_path, augmentation):
    """
    Reads an image, applies augmentations, and saves the processed image.

    Parameters:
        file_path (str): Path to the image file.
        augmentation (albumentations.Compose): Augmentation pipeline to apply.
    """
    # Read the image
    image = cv2.imread(file_path)
    if image is None:
        print(f"Failed to read image: {file_path}")
        return

    # Apply augmentations
    augmented = augmentation(image=image)
    augmented_image = augmented["image"]

    # Save the processed image
    cv2.imwrite(file_path, augmented_image)
    print(f"Processed and saved: {file_path}")
```

در کد بالا، ما تابعی به نام _process_and_save_image تعریف کرده‌ایم که وظیفه‌اش خواندن یک تصویر، اعمال تغییرات (Augmentation) بر روی آن و ذخیره تصویر پردازش شده است. ابتدا، ما تصویر را از مسیر مشخص شده با استفاده از OpenCV می‌خوانیم و در صورتی که خواندن تصویر موفقیت‌آمیز نباشد، یک پیام خطا چاپ می‌کنیم. سپس، تغییرات مشخص شده در پارامتر augmentation را بر روی تصویر اعمال می‌کنیم و تصویر تغییر یافته را ذخیره می‌کنیم. در نهایت، با موفقیت تصویر پردازش شده را ذخیره کرده و پیامی مبنی بر اینکه تصویر پردازش و ذخیره شده است، چاپ می‌کنیم.

```python
import os
import cv2
from albumentations import HorizontalFlip, RandomBrightnessContrast, Rotate, GaussianBlur, Compose

def apply_light_augmentation_in_folder(folder_path):
    """
    Applies light augmentation, including Gaussian Blur, to all .png images in the given folder and subfolders.

    Parameters:
        folder_path (str): Path to the folder containing the images.
    """
    # Define light augmentations
    augmentation = Compose([
        HorizontalFlip(p=0.2),  # Small chance for horizontal flip
        Rotate(limit=5, p=0.2),  # Slight rotation (max ±5 degrees)
        GaussianBlur(blur_limit=(3, 5), p=0.06),  # Gaussian blur with kernel size between 3x3 and 5x5
        RandomBrightnessContrast(brightness_limit=0.1, contrast_limit=0.1, p=0.5),  # Small brightness/contrast change
    ])

    # Process images in the folder and subfolders
    for root, _, files in os.walk(folder_path):
        for file in files:
            if file.lower().endswith('.png'):  # Check file extension (case insensitive)
                file_path = os.path.join(root, file)
                _process_and_save_image(file_path, augmentation)

train_drowsy_path = "/content/data_split/train/drowsy"
train_non_drowsy_path = "/content/data_split/train/non drowsy"

apply_light_augmentation_in_folder(train_drowsy_path)
apply_light_augmentation_in_folder(train_non_drowsy_path)
```

در کد بالا، ما تابعی به نام apply_light_augmentation_in_folder تعریف کرده‌ایم که تغییرات سبک (light augmentation) را بر روی تمام تصاویر با فرمت PNG در یک دایرکتوری و زیرشاخه‌های آن اعمال می‌کند. ابتدا، یک مجموعه از تغییرات شامل چرخش جزئی، بلور گوسی، تغییرات روشنایی و کنتراست و معکوس افقی تعریف می‌کنیم. سپس با استفاده از تابع os.walk ، به طور بازگشتی به تمام فایل‌ها در دایرکتوری مشخص شده دسترسی پیدا می‌کنیم و برای هر فایل با پسوندPNG ، تابع process_and_save_image_ را فراخوانی می‌کنیم تا تغییرات را بر روی آن اعمال کرده و تصویر پردازش شده را ذخیره کنیم. در نهایت، این تابع را برای دو دایرکتوری جداگانه که شامل تصاویر "خواب‌آلود" و "غیر خواب‌آلود" هستند، فراخوانی می‌کنیم تا تغییرات به طور همزمان بر روی هر دو مجموعه داده اعمال شود.

```python
import os
from ultralytics import YOLO

def create_yaml_train_and_evaluate(train_dir, val_dir, output_yaml_path, model_path, epochs=1, img_size=128, batch_size=24):
    """
    Creates a YAML configuration file, trains a YOLO model, and evaluates its performance.

    Parameters:
        train_dir (str): Path to the training data directory.
        val_dir (str): Path to the validation data directory.
        output_yaml_path (str): Path to save the YAML configuration file.
        model_path (str): Path to the pre-trained YOLO model file.
        epochs (int): Number of training epochs. Default is 1.
        img_size (int): Image size for training. Default is 128.
        batch_size (int): Batch size for training. Default is 24.
    """
    # Create YAML configuration file
    yaml_content = f"""
path: {os.path.abspath('./data_split')}
train: {os.path.abspath(train_dir)}
val: {os.path.abspath(val_dir)}
nc: 2
names: ['drowsy', 'non_drowsy']
weights: [3.0, 3.0]
"""
    with open(output_yaml_path, 'w') as f:
        f.write(yaml_content)
    print(f"YAML configuration file created at: {output_yaml_path}")
```

```python
    # Train the YOLO model
    model = YOLO(model_path)
    model.train(data=output_yaml_path, epochs=epochs, imgsz=img_size, batch=batch_size)
    print("Model training completed.")

    # Evaluate the model
    metrics = model.val()
    precision = metrics.results_dict['metrics/precision(B)']
    recall = metrics.results_dict['metrics/recall(B)']
    f1_score = 2 * (precision * recall) / (precision + recall + 1e-6)

    # Print evaluation metrics
    print("Evaluation Results:")
    print("F1-Score:", f1_score)
    print("Precision:", precision)
    print("Recall:", recall)

    return f1_score, precision, recall

train_dir = './data_split/train'
val_dir = './data_split/val'
train_data_path = './data_split/data.yaml'
pretrained_model_path = 'yolov8n.pt'

f1_score, precision, recall = create_yaml_train_and_evaluate(
    train_dir, val_dir, train_data_path, pretrained_model_path,
    epochs=1, img_size=128, batch_size=24
)
```

در کد بالا، ما تابعی به نام create_yaml_train_and_evaluate تعریف کرده‌ایم که وظیفه‌اش ایجاد یک فایل پیکربندی YAML، آموزش یک مدل YOLO و ارزیابی عملکرد آن است. ابتدا، محتوای فایل YAML را با مسیرهای دایرکتوری‌های آموزشی و اعتبارسنجی، تعداد کلاس‌ها و نام‌های آن‌ها تنظیم می‌کنیم و سپس این محتوا را در مسیر مشخص شده ذخیره می‌کنیم. پس از آن، مدل YOLOرا با استفاده از مسیر مدل پیش‌آموزش دیده بارگذاری کرده و آن را با داده‌های مشخص شده در فایل YAMLآموزش می‌دهیم. پس از اتمام آموزش، مدل را ارزیابی کرده و معیارهای دقت، یادآوری و امتیاز F1 را محاسبه می‌کنیم. در نهایت، نتایج ارزیابی شامل امتیازF1 ، دقت و یادآوری را چاپ کرده و این مقادیر را به عنوان خروجی تابع بازمی‌گردانیم. ما این تابع را برای دایرکتوری‌های آموزشی و اعتبارسنجی مشخص شده و با استفاده از مدل پیش‌آموزش دیده YOLO فراخوانی می‌کنیم.

- **بخش ۵. نتایج حاصل از اجرای مدل بر روی دیتاست ها همراه با پیش پردازش**

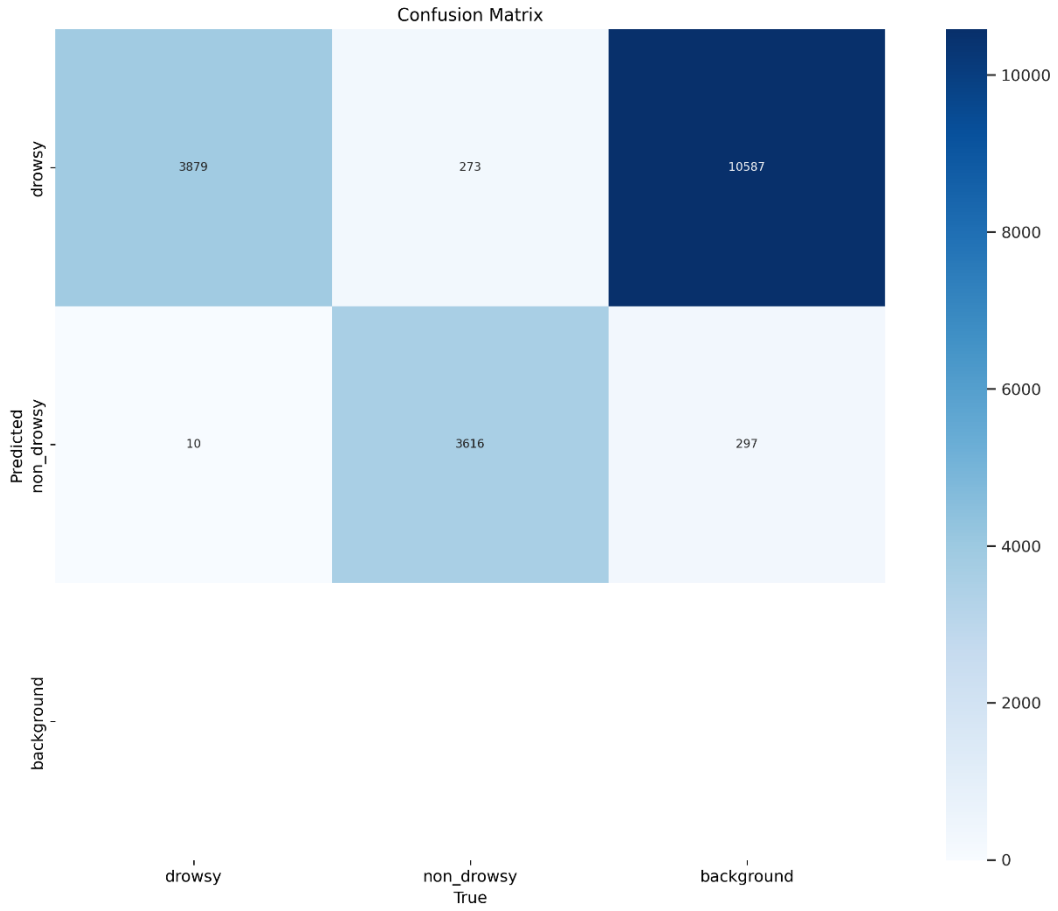
در جدول زیر ، مقادیر f1-socre ، precision و recall حاصل از اجرای مدل ها بر روی دیتاست موجود نمایش داده شده است. کلیه این مقادیر ، از بخش تست و ارزیابی انتهای مدل ها برداشته شده است که قابل مشاهده می باشد.

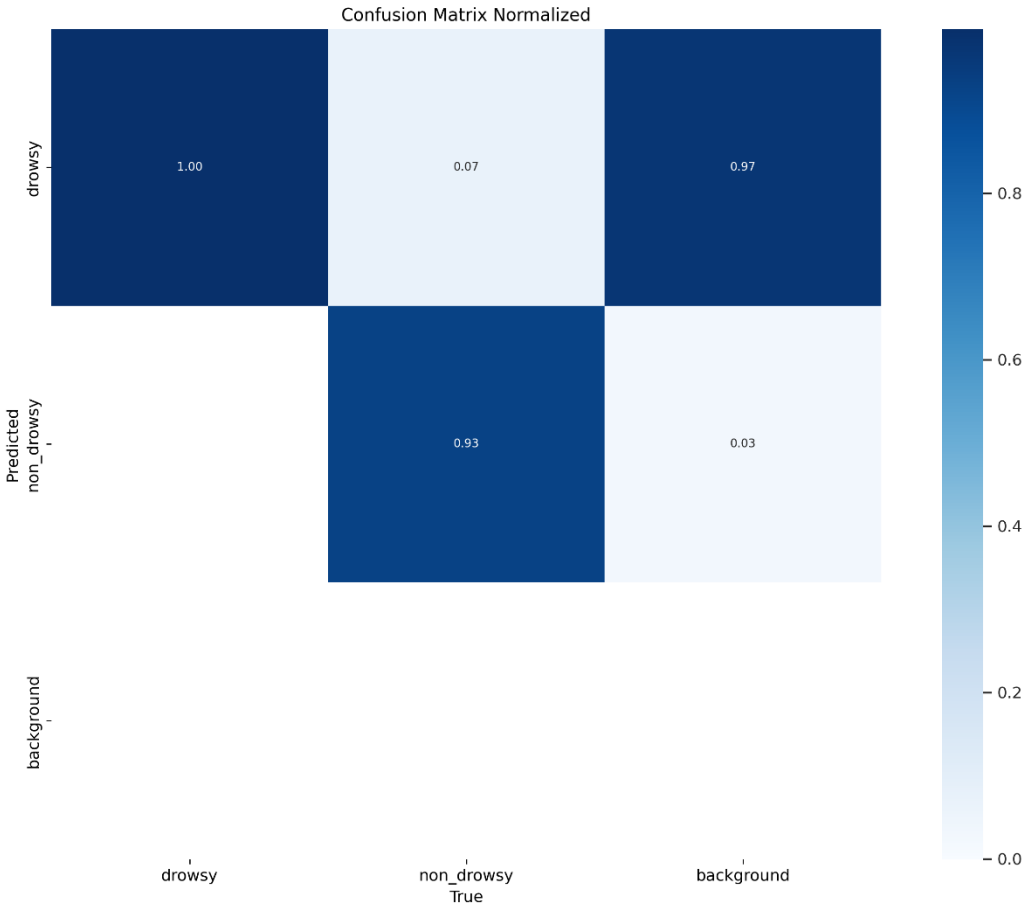
	f1-score	precision	recall
imgsz_128_batch_24	0.985	0.988	0.981
imgsz_128_batch_32	0.969	0.971	0.966
imgsz_224_batch_24	0.991	0.993	0.990
imgsz_224_batch_32	0.934	0.949	0.919

در ادامه برخی نمودارها و جداول حاصل نمایش داده شده است.

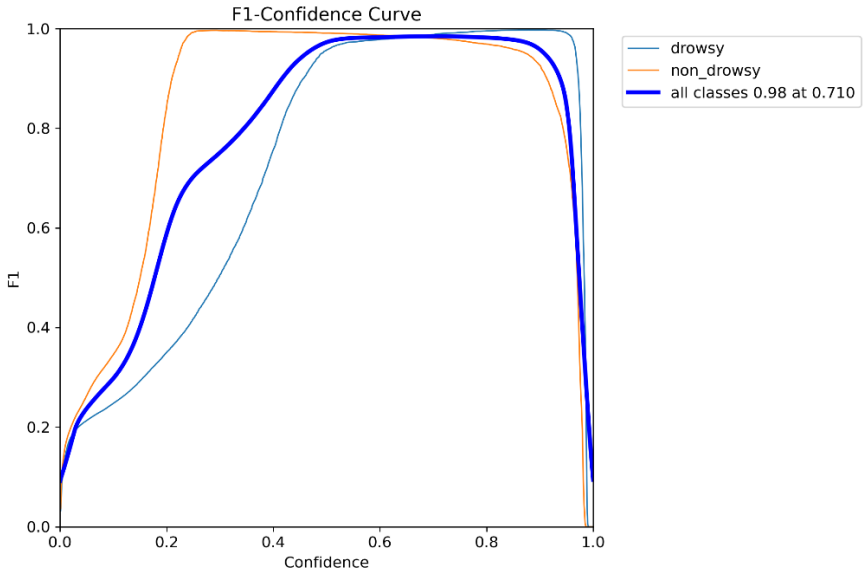
برای مدل با image size = 128 و batch size = 24 :

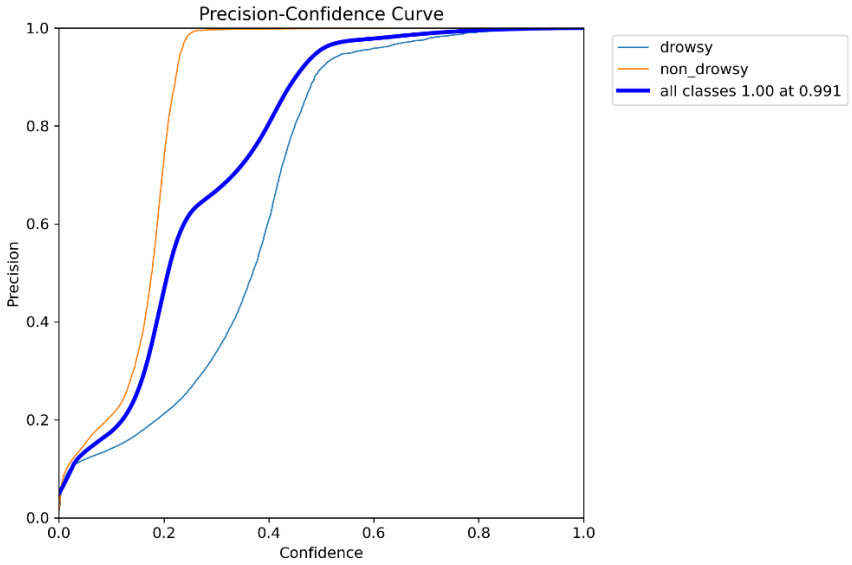
- جدول درهم آمیختگی به همراه نسخه نرمال سازی شده آن:

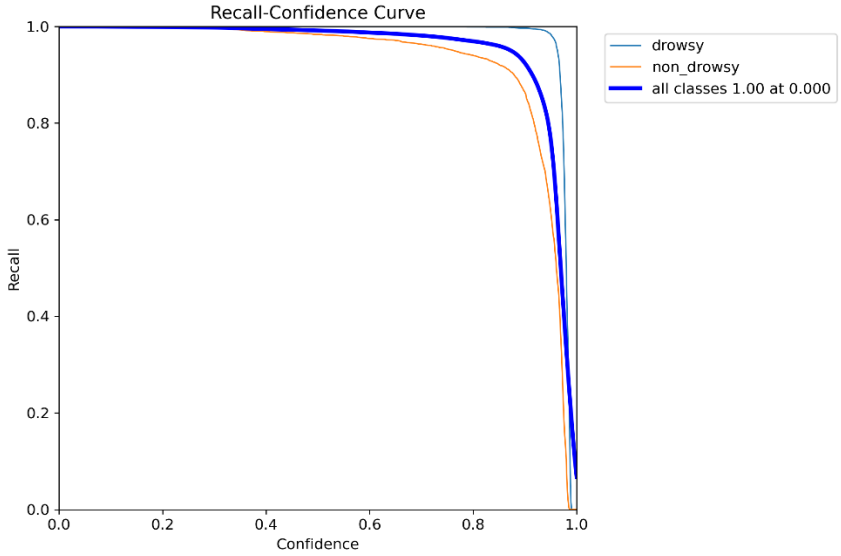




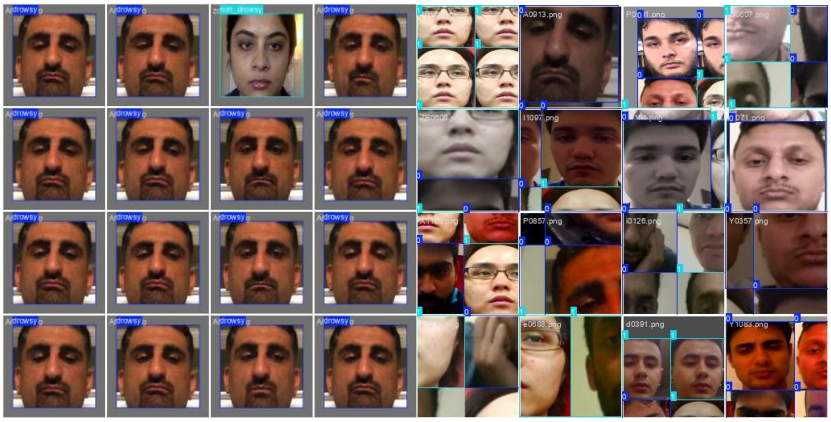
- Curve های مرتبط با f1-score و precision و recall:







- نمونه ای از دسته های train و validation:



برای بقیه مدل ها نیز ، جداول درهم آمیختگی ، کِرو های مربوطه ، نمونه ای از دسته های train و validation به اضافه سایر دیتاهای مدل در ریپازیتوری پروژه قابل دسترس میباشند.

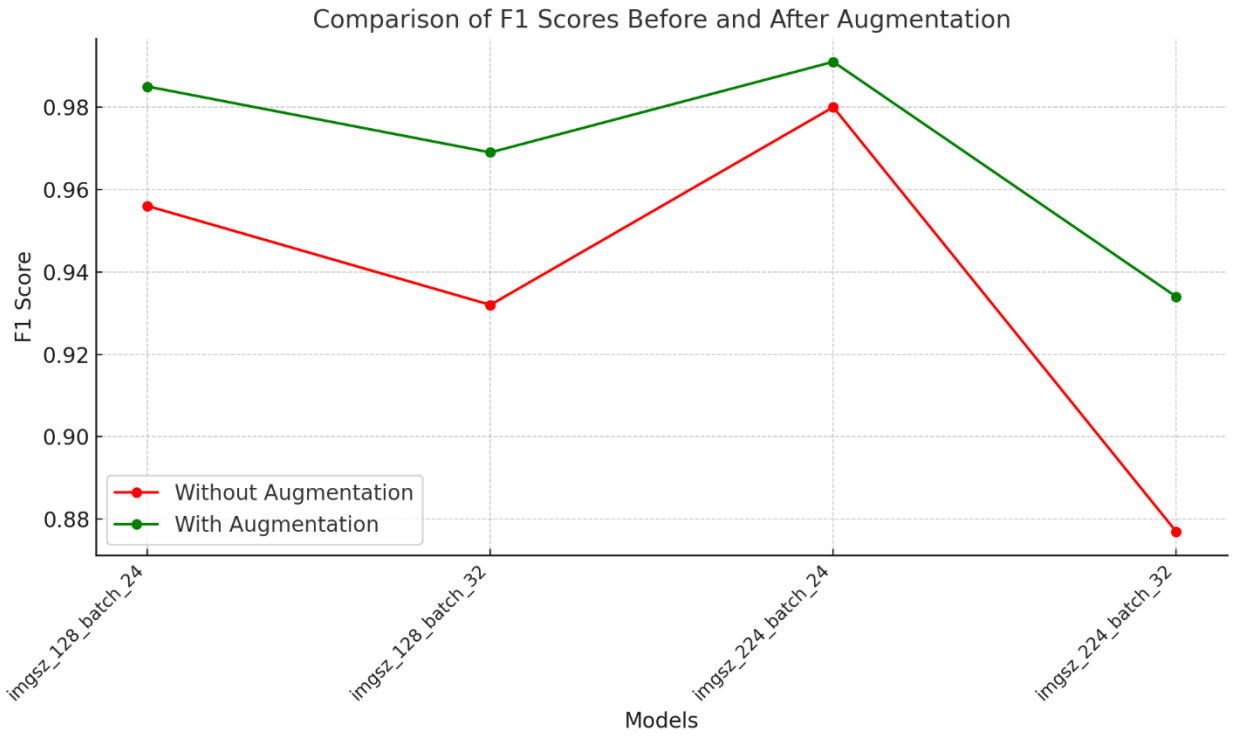
- **بخش ۶ . مقایسه نتایج و میزان پیشرفت:**

پس از اعمال آگمنتیشن بر روی داده ها شاهد افزایش مقادیر حاصل از مدل ها هستیم. در جدول زیر میزان پیشرفت را بررسی می کنیم :

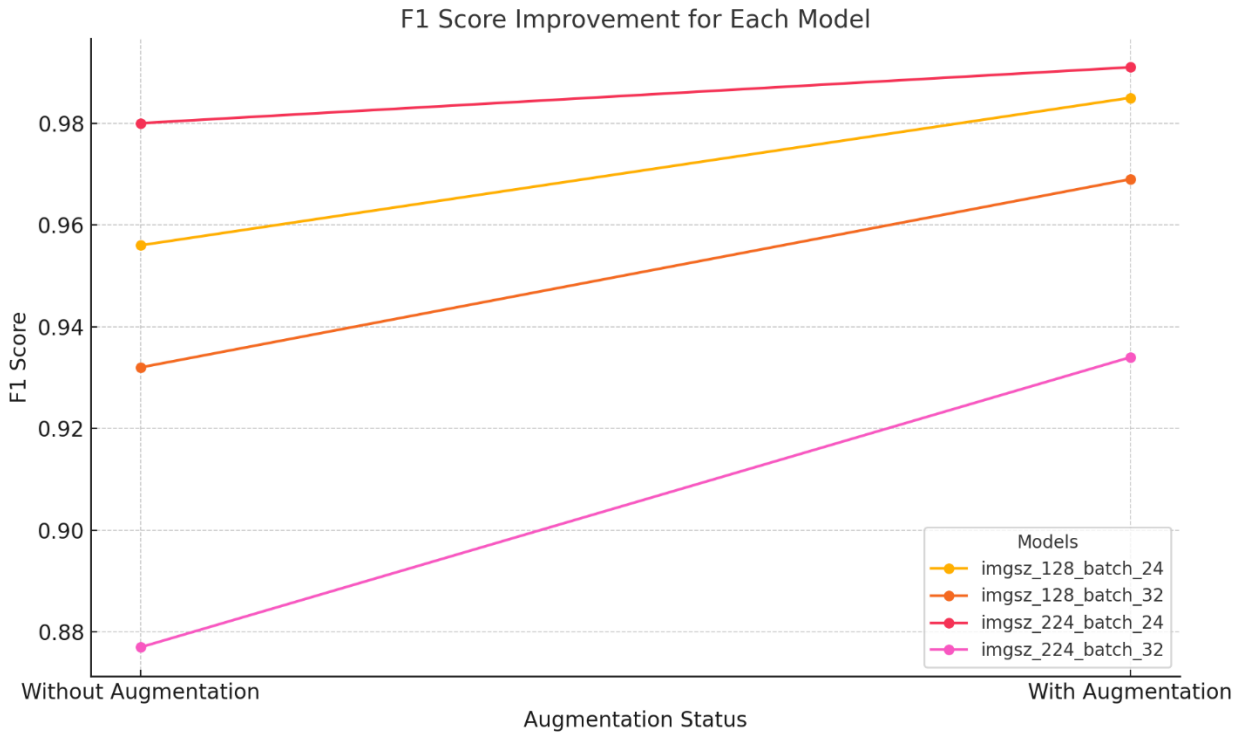
	F1_score without_augmentation	F1_score with_augmentation	Amount of increase
imgsz_128_batch_24	0.956	0.985	0.029
imgsz_128_batch_32	0.932	0.969	0.037
imgsz_224_batch_24	0.980	0.991	0.011
imgsz_224_batch_32	0.877	0.934	0.057

همانطور که مشاهده می شود ، مقدار f1-socre در مدل اول ۲.۹ درصد ، در مدل دوم ۳.۷ درصد ، در مدل سوم ۱.۱ درصد و در مدل چهارم ۵.۷ درصد افزایش یافته است.

در نمودار زیر ، روند پیشرفت نمایش داده شده است :



همچنین در نمودار زیر ، پیشرفت مجزای هر مدل نمایش داده شده است :



در جدول زیر ، میزان تغییرات precision نمایش داده شده است :

	Precision without_augmentation	Precision with_augmentation	Amount of increase
imgsz_128_batch_24	0.946	0.988	0.042
imgsz_128_batch_32	0.926	0.971	0.045
imgsz_224_batch_24	0.984	0.993	0.009
imgsz_224_batch_32	0.839	0.949	0.011

در جدول زیر نیز ، میزان تغییرات recall نمایش داده شده است :

	recall without_augmentation	recall with_augmentation	Amount of increase
imgsz_128_batch_24	0.967	0.981	0.014
imgsz_128_batch_32	0.938	0.966	0.028
imgsz_224_batch_24	0.976	0.990	0.014
imgsz_224_batch_32	0.917	0.919	0.002