به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین امتیازی

نام و نام خانوادگی	مریم دادخواه – جواد سراج
شماره دانشجویی	۸۱۰۱۰۱۱۵۱ - ۸۱۰۱۰۱۱۸۶
تاریخ ارسال گزارش	۱۴۰۲.۰۳.۱۵

فهرست

شکل‌ها

جدول‌ها

پاسخ ۱. عنوان پرسش اول به فارسی

۱-۱. توضیحات مدل‌ها

LSTM و GRU دو ساختار شبکه‌های عصبی هستند که برای تحلیل‌های سری زمانی و متن بسیار کاربرد دارند. هر دو شبکه مذکور برای حل مشکل vanishing gradient که در RNN ها رخ می‌دهد ارائه شده اند. تفاوت اصلی دو شبکه GRU و LSTM نحوه کنترل جریان اطلاعات و حافظه مدل است که درادامه به توضیح آن میپردازیم.

LSRM سه گیت مجزا به نام های input gate، forget gate، output gate دارد. همچنین دو state به نام cell state و hidden state دارد. در GRU دو گیت input و forget ادغام شدند و به نام update gate شناخته می‌شوند و دو state نیز باهم ادغام می‌شوند و این باعث کاهش هزینه محاسباتی GRU نسبت به LSTM می‌شود. همچنین در عمل GRU برای داده‌های کمتر و منابع پردازشی کمتر مناسب و LSTM برای داده‌های بیشتر و منابع محاسباتی بیشتر مناسب می‌باشد.

شبکه طراحی شده در مقاله، از دو بخش (مسیر) تشکیل شده است. این دو مسیر به صورت موازی جریان اطلاعات را عبور می‌دهد. مسیر اول مسیر LSTM است که از دو بلاک LSTM تشکیل شده است و در ادامه آن یک لایه Dence قرار میگیرد. مسیر دوم مسیر GRU است که در آن یک بلوک GRU قرارگرفته و خروجی آن به یک لایه Dence داده می‌شود. در نهایت خروجی هر دو مسیر با هم ادغام می‌شود و پیشبینی قیمت صورت می‌گیرد.

۱-۲. دیتاست و پیش‌پردازش داده

مجموعه داده‌ها را از سایت investing دانلود می‌کنیم. دو دیتاست Litecoin و Monero برای تخمین مورد بررسی قرار می‌گیرند. Monero دارای ۱۸۵۱ دیتاپوینت و Litecooin دارای ۱۲۷۹ دیتاپوینت است. دیتاست به صورت زیر است:

	Date	Price	Open	High	Low	Vol.
0	Jan 30, 2015	0.303	0.311	0.350	0.303	1330
1	Jan 31, 2015	0.290	0.303	0.303	0.290	480
2	Feb 01, 2015	0.290	0.290	0.290	0.290	-1
3	Feb 02, 2015	0.303	0.290	0.303	0.287	350
4	Feb 03, 2015	0.331	0.303	0.360	0.302	1470
...	...	...	...	...	...	...
1846	Feb 19, 2020	77.843	86.158	86.914	76.718	304860
1847	Feb 20, 2020	76.355	77.843	78.903	75.102	384420
1848	Feb 21, 2020	80.313	76.355	82.285	75.886	290150
1849	Feb 22, 2020	79.027	80.308	82.896	78.169	277070
1850	Feb 23, 2020	85.329	79.024	85.541	78.818	271000

1851 rows × 6 columns

شکل ۱. دیتاست **Monero**

ما فقط از ستون Price دیتاست برای پیشبینی قیمت استفاده می‌کنیم اما میتوان ستون های دیگر دیتاست را به عنوان ویژگی‌های جدید برای آموزش شبکه به کار برد.

پیش‌پردازش داده‌ها بخش مهم و تاثیرگذار در فرآیند آموزش است. برای پردازش داده‌ها طبق مقاله ابتدا داده‌های قیمت را نرمالایز می‌کنیم. نرمال سازی (MinMax) داده‌ها را بین ۰ و ۱ قرار میدهد. در تصویر زیر روند نرمالسازی ستون price نمایش داده‌شده است.

$$x_{normalized} = \frac{x_{original} - x_{min}}{x_{max} - x_{min}}$$

```python
class Normalizer():
    def __init__(self, df):
        self.df = df
        self.scaler = MinMaxScaler()
        self.normalized_df = None

    def normalize(self):
        self.scaler.fit(self.df)
        normalized_data = self.scaler.transform(self.df)
        self.normalized_df = pd.DataFrame(normalized_data, columns=["Price"])

        return self.normalized_df


litecoin_df_price = litecoin_df[["Price"]]
monero_df_price = monero_df[["Price"]]

litecoin_normalizer = Normalizer(litecoin_df[["Price"]])
monero_normalizer = Normalizer(monero_df[["Price"]])

litecoin_df_normal = litecoin_normalizer.normalize()
monero_df_normal = monero_normalizer.normalize()
litecoin_df_normal
```

شکل ۲. نرمالسازی داده‌ها پیش از آموزش مدل با روش **MinMaxscaler**

برای ساخت داده مناسب برای شبکه از روش Sliding Window استفاده می‌کنیم که در مقاله نیز برای ساخت داده‌ها از این روش استفاده شده. مطابق مقاله Window size = 30 در نظر گرفته شده است. یعنی هر سمپل ورودی به شبکه شامل اطلاعات قیمتی ۳۰ روز قبل است. در مقاله اشاره شده که سه سناریو وجود دارد. پیشبینی ۱ روز یا ۳ روز یا ۷ روز که به آن prediction_window_size می‌گویند.

لیبل‌ها از طریق prediction_window_size مشخص میشوند. برای مثال اگر prediction_window_size = 3 باشد انگاه لیبل ما قیمت سه روز آخر خواهد بود.

```python
def generate_data(df, col_name, window_size=30):
    row_data  = np.array(df[col_name])
    x = []
    y =  []
    for i in range(len(row_data)-window_size):
        x.append(row_data[i:i+window_size])
        y.append(row_data[i+window_size])
    return np.array(x)[...,np.newaxis], np.array(y)[...,np.newaxis]
```

```python
x, y = generate_data(monero_df_normal, 'Price')
x.shape,y.shape
```

```
((1821, 30, 1), (1821, 1))
```

شکل ۳. آماده سازی داده‌های ورودی به شبکه

سه سناریو برای هر دیتاست وجود دارد.

- ابتدا با استفاده از همه داده‌ها به جز روز آخر در دیتاست، قیمت روز آخر را پیش بینی میکنیم.
- با استفاده از همه داده‌ها به جز سه روز آخر، قیمت سه روز آخر را پیش‌بینی می‌کنیم.
- با استفاده از همه داده‌ها به جز ۷ روز آخر قیمت ۷ روز آخر را پیش‌بینی می‌کنیم.

۱-۳. آموزش مدل‌ها

طراحی مدل‌های مقاله

به سه سناریو تخمین قیمت در بالا اشاره شده. در این بخش به جزییات طراحی و آموزش شبکه می‌پردازیم.

در مقاله به دو شبکه اشاره شده است.

- شبکه اول شبکه LSTM است. ما برای شبکه LSTM هیدن سایز را ۳۰ در نظر گرفتیم و خروجی LSTM به یک لایه Fully connected با ۳۰ نورون داده می‌شود.
- شبکه پیشنهادی هابیرید مقاله را در ادامه به طور کامل مورد بررسی قرار می‌دهیم.

در ادامه شبکه مذکور را با دو مسیر LSTM و GRU در تورچ طراحی کردیم که در مسیر LSTM ابتدا یک بلاک LSTM با hidden size = 30 وجود دارد. سپس مقاله به dropout اشاره کرده و ما نیز این نکته را در کد در نظر گرفتیم. در ادامه یک بلاک LSTM ۵۰ نورون قرار میگیرد و خروجی این بلاک به یک لایه fully connect داده می‌شود. ابعاد لایه‌های fully connected در مقاله ذکر نشده و ما سعی میکنیم به صورتی این ابعاد را در نظر بگیریم که مدل بهتر عمل کند.

```
def lstm_path(self, x):
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dims[0]).requires_grad_()
    c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dims[0]).requires_grad_()
    out, _ = self.lstm1(x, (h0.detach(), c0.detach()))
    out, _ = self.lstm2(out)
    out  = self.fc1(out[:,-1,:])
    return out
```

در مسیر GRU یک بلاک GRU با hidden size = 30 قرار داده می‌شود و مطابق مقاله dropput این لایه مانند مسیر LSTM فعال می‌شود و خروجی آن به یک لایه fully connected داده می‌شود.

```
def gru_path(self, x):
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dims[0]).requires_grad_()
    out, _ = self.gru(x, (h0.detach()))
    out  = self.fc2(out[:,-1,:])
    return out
```

در نهایت خروجی دو مسیر با هم ادغام می‌شود و از لایه fully connected نهایی عبور داده می‌شود و مسیر forward تکمیل می‌شود. با توجه به predicrion_window_size تعداد نورون‌های خروجی شبکه نیز تعیین می‌شود.

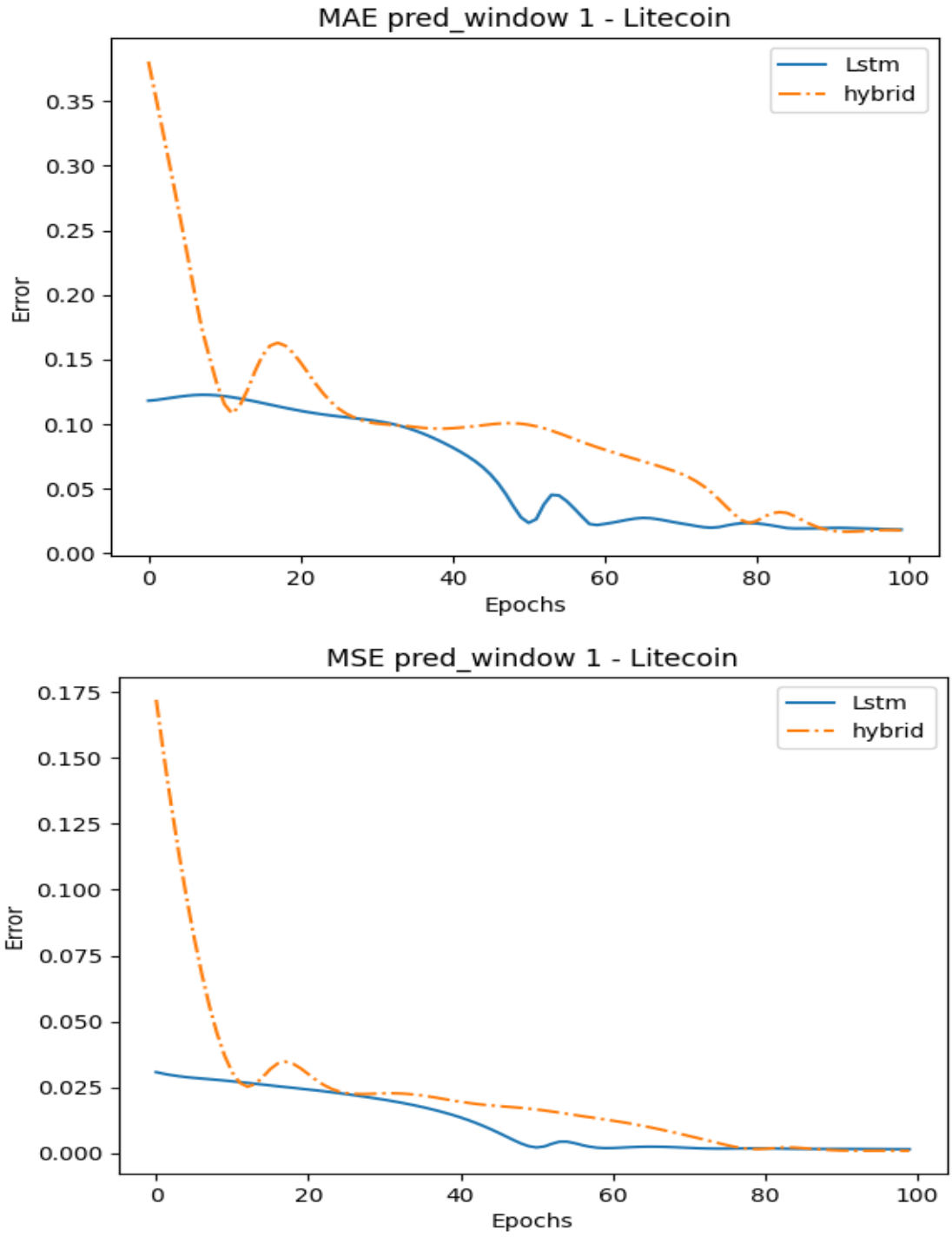
```
def forward(self,x):
    out1 = self.lstm_path(x)
    out2 = self.gru_path(x)
    out = torch.cat((out1, out2), dim=1)
    out = self.fc3(out)
    return out
```

برای آموزش شبکه از تابع هزینه Mean Square Error و بهینه‌ساز Adam با نرخ یادگیری ۰.۰۰۱ استفاده شده است. تعداد ایپاک‌ها نیز مطابق مقاله ۱۰۰ در نظر گرفته شده است.

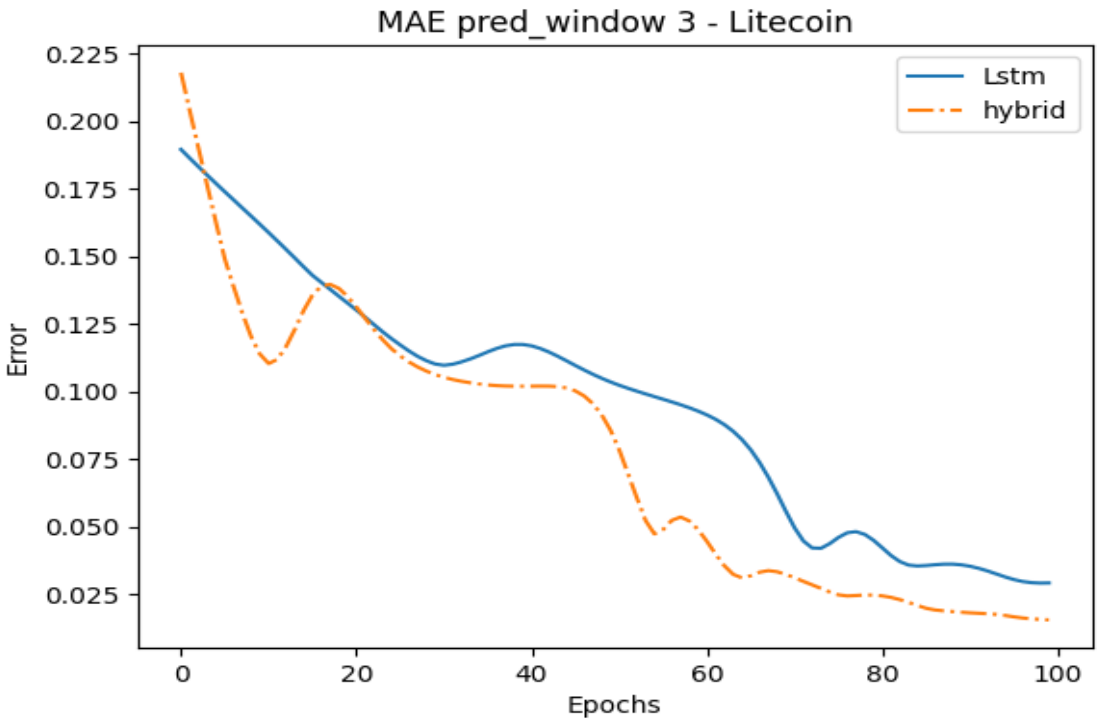
نمودارهای کاهش MSE و MAE در روند آموزش

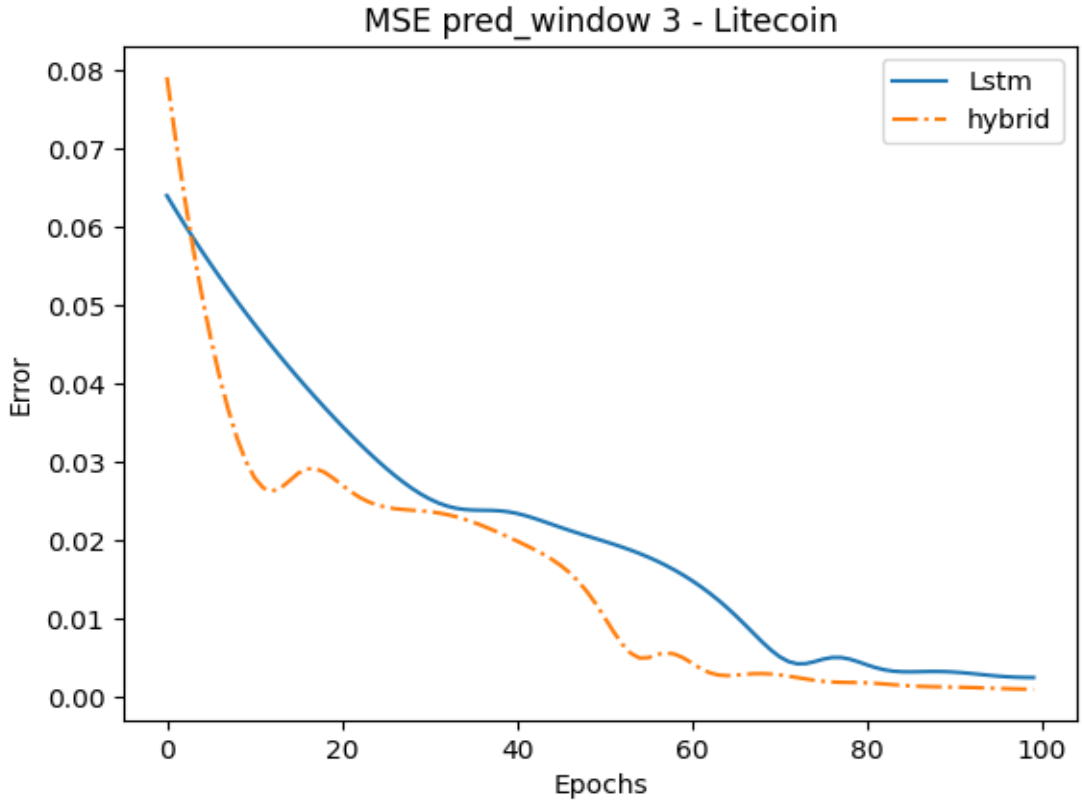
در ادامه نمودارهای MSE و MAE در طول روند آموزش برای سناریوهای مختلف و دو شبکه LSTM و Hybrid برای دو دیتاست Litecoin و Monero آورده شده است. نمودار هر دو شبکه LSTM و Hybrid در یک گراف آورده شده است تا دو شبکه قابل مقایسه باشند.

نتایج آموزش روی دیتاست Litecoin

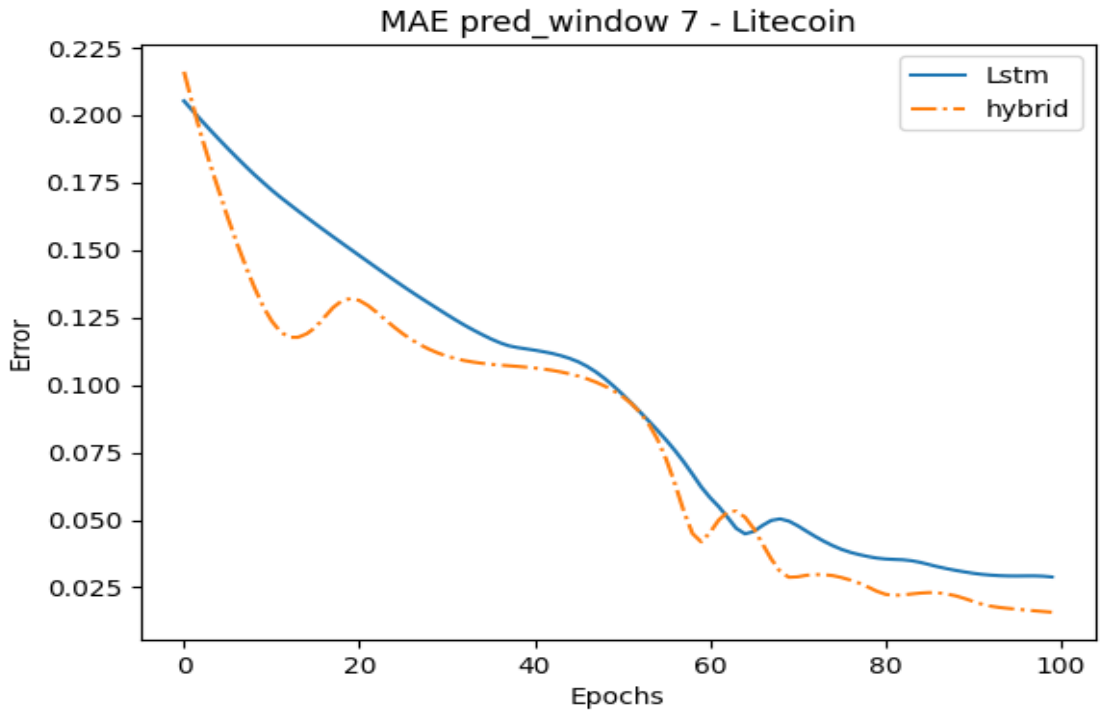


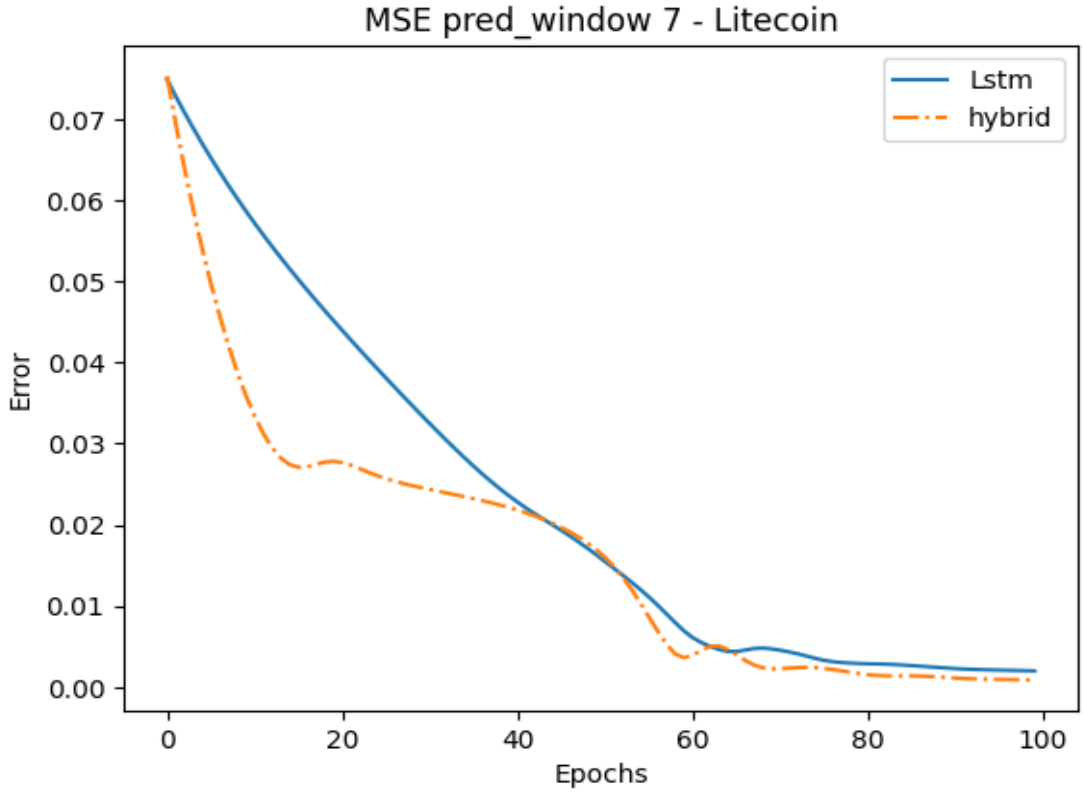
شکل ۴. نمودار **MSE** و **MAE** برای دیتاست **Litecoin**





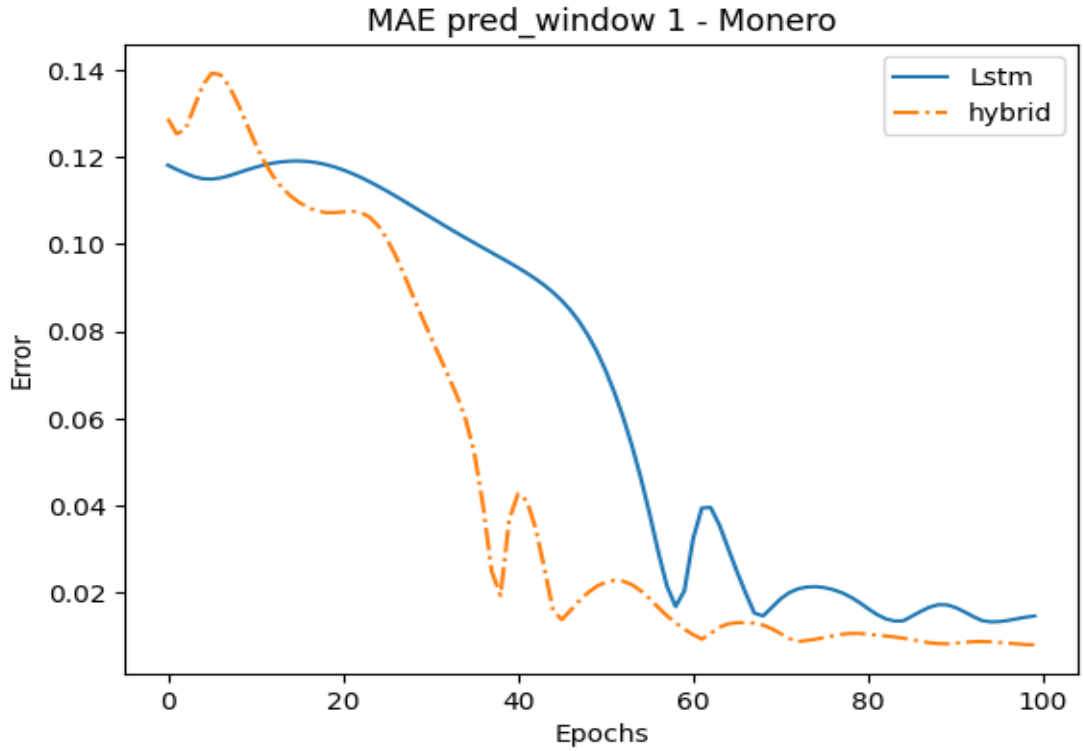
شکل ۵. نمودار **MSE** و **MAE** برای دیتاست **Litecoin**

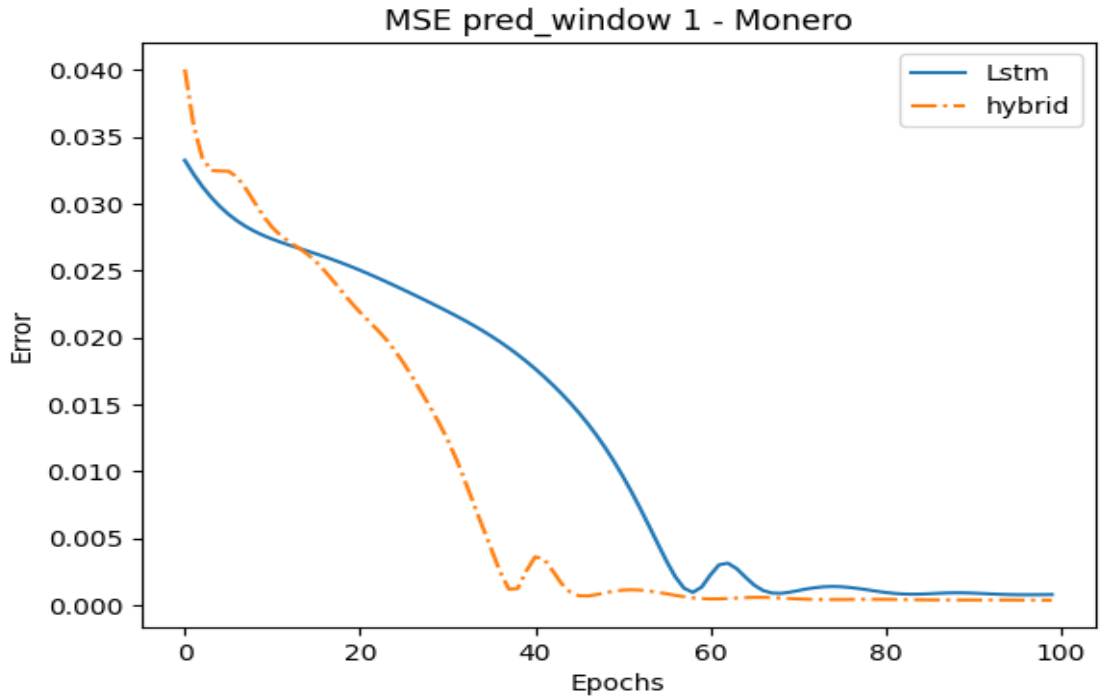




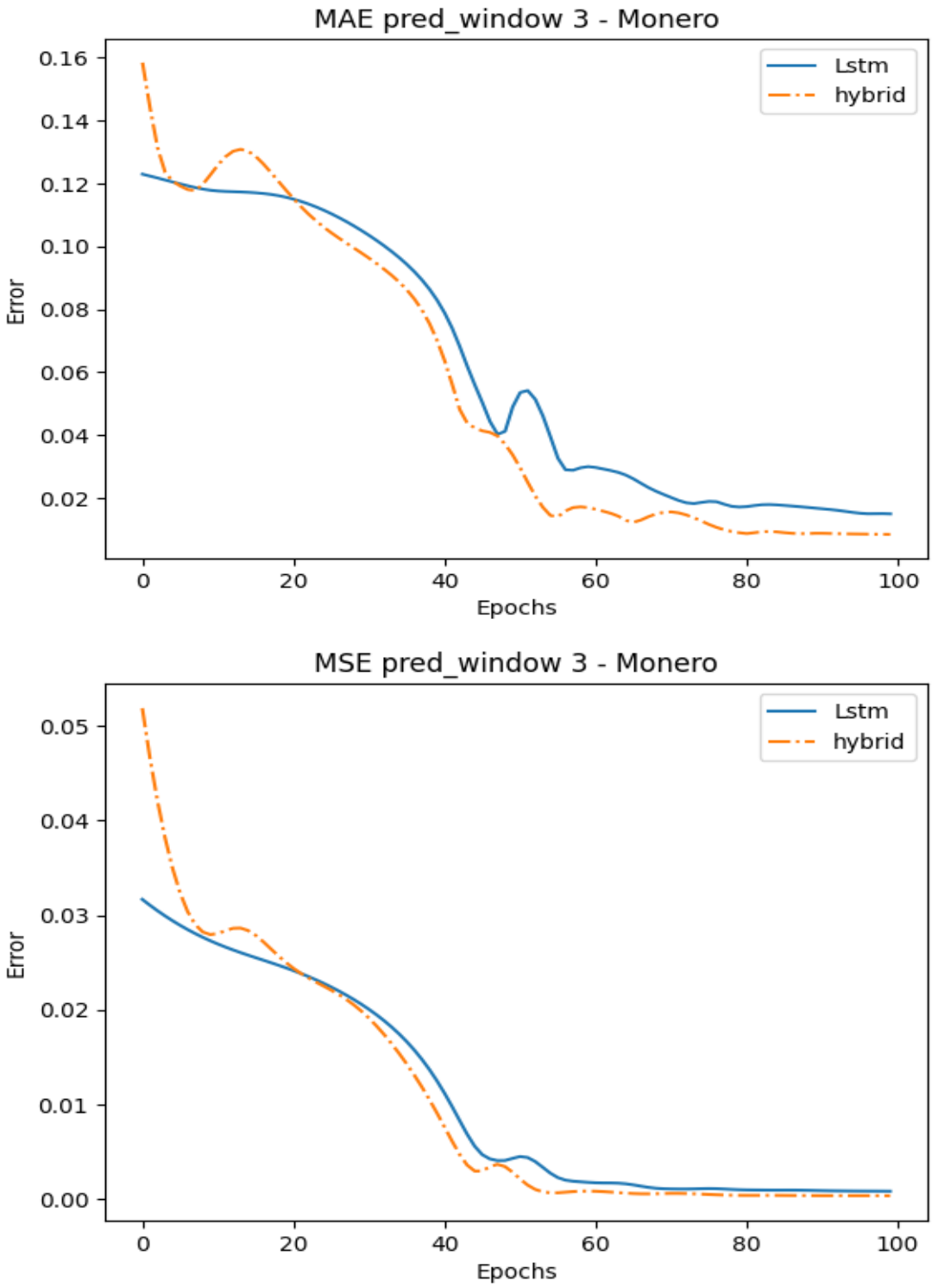
شکل ۶. نمودار **MSE** و **MAE** برای دیتاست **Litecoin**

نتایج آموزش روی دیتاست Monero

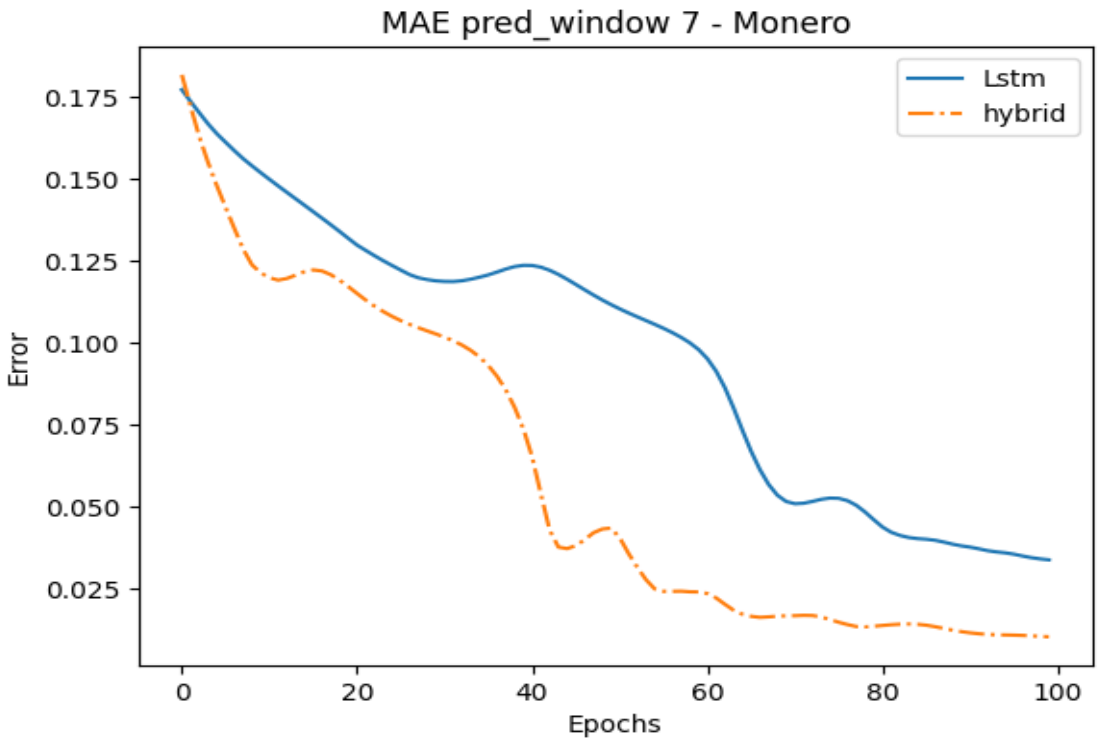


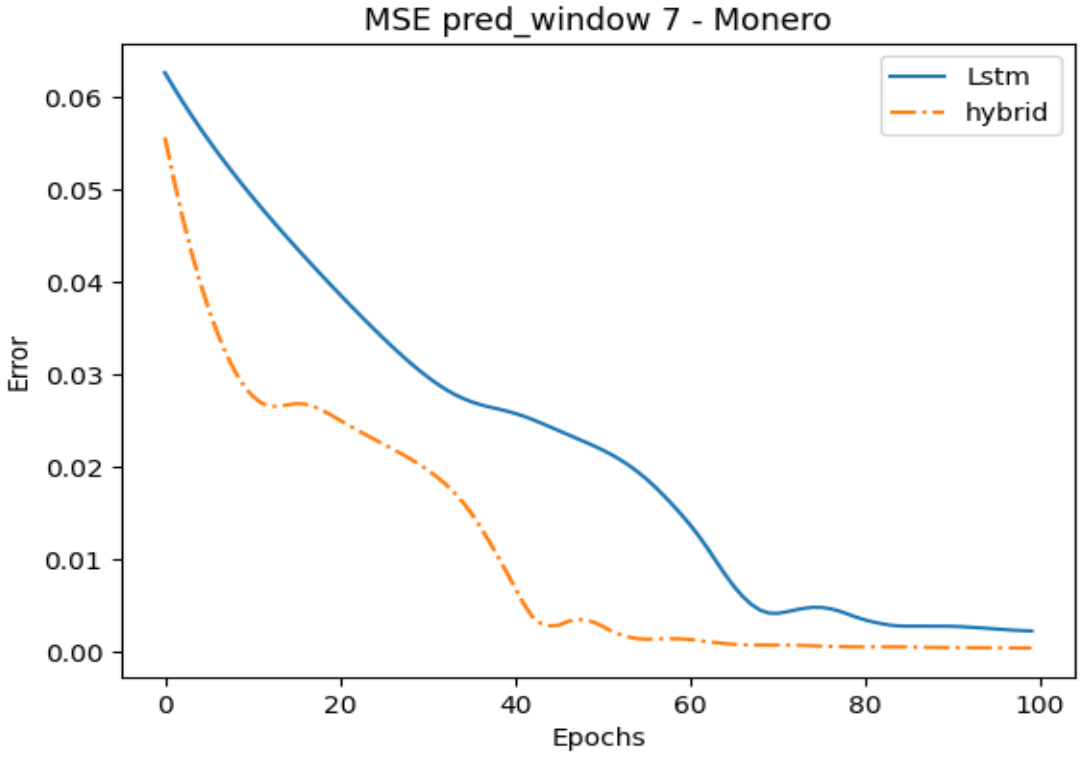


شکل ۷. نمودار **MSE** و **MAE** برای دیتاست **Monero**



شکل ۸. نمودار **MSE** و **MAE** برای دیتاست **Monero**





شکل ۹. نمودار **MSE** و **MAE** برای دیتاست **Monero**

۱-۴. نتایج و بحث و بررسی

در ادامه مطابق پس از آموزش مدل‌ها خطاهای خواسته شده را برای همه سناریوهای مقاله به دست می‌آوریم و در انتهای این بخش به تحلیل نتایج میپردازیم.

جدول ۱. نتایج مدل برای پنجره پیشبینی ۱

		MSE	RMSE	MAE	MAPE
LSTM	Litecoin	18.41	4.29	8.59	0.5
	Monero	13.19	3.63	7.02	2.12
Hybrid	Litecoin	14.25	3.78	8.10	3.44
	Monero	8.76	2.96	3.86	0.22

جدول ۲.نتایج مدل برای پنجره پیشبینی ۳

		MSE	RMSE	MAE	MAPE
LSTM	Litecoin	23.44	4.84	13.92	4,65
	Monero	13.26	3.64	7.08	1.14
Hybrid	Litecoin	14.37	3.79	7.31	1.26
	Monero	8.29	2.88	4.04	0.78

جدول ۳. نتایج مدل برای پنجره پیشبینی ۷

		MSE	RMSE	MAE	MAPE
LSTM	Litecoin	20.94	4.58	13.51	5.42
	Monero	22.38	4.73	15.91	7.86
Hybrid	Litecoin	13.69	3.70	7.35	2.52
	Monero	9.5	3.08	4.81	1.15

تحلیل نتایج

با توجه به روند آموزش مدل‌ها و نمودارهای کاهش MSE و MAE نتیجه می‌گیریم که مدل‌ها به خوبی آموزش داده شده اند. برای اینکه با مشکل overfitting مواجه نشویم باید دیتاهای validation نیز برای مدل قرار میدادیم تا روند آموزش به درستی کنترل شود. همان طور که در نمودارهای MSE و MAE دیده می‌شود، مدل Hybrid سریع‌تر از LSTM همگرا شد و به مقادیر MSE کمتری رسید این نشان می‌دهد که مدل طراحی شده در مقاله (ترکیبی LSRM و GRU) بهتر از مدل LSTM عمل می‌کند. از طرف دیگر اگر به جداول ۱ و ۲ و۳ دقت شود می‌بینیم که هم روی داده‌های Litecoin و هم روی داده‌های Monero هر ۴ پارامتر خطای مذکور روی مدل Hybrid کمتر از LSTM است و این نشان می‌دهد مدل Hybrid خطای کمتر و دقت بیشتری در تخمین و پیشبینی قیمت دارد. نکته دیگری که از جداول ۱ تا ۳ برداشت می‌شود این است که هر چه پنجره پیشبینی (Predict_window_size) افزایش یابد خطای مدل افزایش می‌یابد و این موضوع کاملا منطقی هست چرا که تخمین قیمت ۷ روز سخت‌تر از تخمین قیمت یک روز است و به همین دلیل مدل خطای بیشتری در تخمین قیمت ۷ روز نسبت به ۳ روز و ۱ روز دارد.

پاسخ ۲ - تشخیص خشونت در فیلم

۲-۱. دریافت و پیش پردازش دادگان

همانطور که در صورت سوال نیز گفته شده است ، دادگان مربوط به این قسمت شامل هزار فیلم است که پانصد عدد آن از دسته ی خشونت و سایرین بدون خشونت هستند.

```
def label_video_names(in_dir):

    # list containing video names
    names = []
    # list containin video labels [1, 0] if it has violence and [0, 1] if not
    labels = []


    for current_dir, dir_names,file_names in os.walk(in_dir):

        for file_name in file_names:

            if file_name[0:2] == 'fi':
                labels.append(0)
                names.append(file_name)
            elif file_name[0:2] == 'no':
                labels.append(1)
                names.append(file_name)

    return names, labels
```

شکل ۱۰. تابع استخراج نام و برچسب ویدئو ها

.

در تابع بالا نام تمامی ویدئو های موجود در دیتاست به همراه برچسب هر کدام در دو آرایه جداگانه ذخیره می شود. پس از استخراج نام و برچسب ها ، داده ها را با نسبت 0.2/0.8 به داده های test و train تقسیم می کنیم.

```
train_df = pd.DataFrame(data ={'video_name':X_train,
                               'tag':y_train},
                               columns = ('video_name' , 'tag'))

test_df = pd.DataFrame(data ={'video_name':X_test,
                              'tag':y_test},
                              columns = ('video_name' , 'tag'))
```

```
print(f"Total videos for training: {len(train_df)}")
print(f"Total videos for testing: {len(test_df)}")
```

```
Total videos for training: 800
Total videos for testing: 200
```

شکل ۱۱. کد ایجاد دیتافریم از داده های استخراجی

برای راحتی استفاده از داده های استخراجی آنها را طبق کد بالا در دو دیتافریم برای Test و train ذخیره می کنیم. همانطور که دیده می شود 800 داده برای train و200 عدد برای test داریم. در ادامه گزارش چندین تابع در مرحله پیش پردازش برای آماده سازی فریم های ورودی به شبکه از آنها استفاده کردیم را توضیح خواهیم داد.

```python
def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y : start_y + min_dim, start_x : start_x + min_dim]


def load_video(path, max_frames=20, resize=(IMG_SIZE, IMG_SIZE)):
    cap = cv2.VideoCapture(path)
    frames = []
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            # frame = crop_center_square(frame)
            frame = cv2.resize(frame, resize)
            frame = frame[:, :, [2, 1, 0]]
            frames.append(frame)

            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    return np.array(frames)
```

شکل ۱۲. دوتابع مربوط به مرحله پیش پردازش داده ها

تابع crop_center_square که در تصویر نمایش داده شده است بر روی frame های ویدئو ها اعمال می شود و وظیفه آن جداسازی یک مربع از وسط هر فریم با توجه به ابعاد آن است. تابع بعدی یعنی load_video ویدئو را از مسیر داده شده می خواند و به اندازه max_frames که در این سوال برابر 20 قرار داده شده است ، فریم های ویدئو را از ابتدا خوانده آنها را resize کرده (۲۵۶) و در آرایه ای ذخیره می کند و سپس آن را بر می گرداند.

```
def get_frames_diff(frames):
  images = []
  for i in range(10):
    index = ((i+1)*2)
    temp = frames[(index-2):index]
    frame_diff = cv2.absdiff(temp[0],temp[1])
    images.append(frame_diff)

  resul = np.array(images)
  return resul
```

شکل ۱۳. تابع **get_frame_diff**

get_frames_diff که در بالا نمایش داده شده است ، هر دو فریم مجاور را خوانده و سپس difference یا تفاوت دو فریم را در یک تصویر جدا ذخیره می کند. هدف این کار این است که همانطور که در مقاله نیز ذکر شده، اینکار بهتر می تواند اکشن ها را نشان دهد و چون جداسازی ما بر اساس اکشن است این کار کمک زیادی به مدل می کند.

در شکل صفحه بعد دوتا دیگر از توابعی که برای مرحله preprocessing داده ها از آنها استفاده کردیم نمایش داده شده است . تابع اول یعنی frame_loader وظیفه scale و نرمال کردن فریم های ورودی به شبکه را دارد . و تابع دوم که در مقاله نیز به آن اشاره شده است وظیفه دارد حاشیه های سیاه اطراف فریم را crop کند.

```python
def frame_loader(frames,figure_shape,to_norm = True):
    output_frames = []
    for frame in frames:
        image = frame

        # Scale
        figure = (image / 255.).astype(np.float32)
        # Normalize
        mean = [0.485, 0.456, 0.406]
        std = [0.229, 0.224, 0.225]
        figure = (figure - mean) / std
        output_frames.append(figure)
    return output_frames
```

```python
def crop_img__remove_Dark(img,x_crop,y_crop,x,y,figure_size):
    x_start = x_crop
    x_end = x-x_crop
    y_start = y_crop
    y_end = y-y_crop
    return cv2.resize(img[y_start:y_end,x_start:x_end,:]
                      ,(figure_size,figure_size))
```

شکل ۱۴. دو تابع مربوط به مرحله پیش پردازش داده ها

تابع بعدی یعنی Crop_img از دیگر توابعی است که برای مرحله پیش پردازش از آن استفاده شده است و در مقاله نیز به آن اشاره شده بود. این تابع با توجه و مقدار corner که می تواند یکی از مقادیر "Center","Left_up","Left_down","Right_up","Right_down" باشد ، فریم ورودی را crop می کند. توجه نماییئ که ضرورت استفاده از این توابع تولید داده های بیشتر و با variation بشتر است تا مدل نسبت به تصاویری که در دیتاست وجود ندارند اصطلاحا robust شود.

```python
def crop_img(img,figure_shape,percentage=0.8,corner="Left_up"):
    if(corner == None):
        corner = random.choice(corner_keys)

    if corner not in corner_keys:
        raise ValueError(
            'Invalid corner method {} specified. Supported '
            'corners are {}'.format(
                corner,
                ", ".join(corner_keys)))

    resize = int(figure_shape*percentage)
    if(corner =="Left_up"):
        x_start = 0
        x_end = resize
        y_start = 0
        y_end = resize
    if (corner == "Right_down"):
        x_start = figure_shape-resize
        x_end = figure_shape
        y_start = figure_shape-resize
        y_end = figure_shape
    if(corner =="Right_up"):
        x_start = 0
        x_end = resize
        y_start = figure_shape-resize
        y_end = figure_shape
    if (corner == "Left_down"):
        x_start = figure_shape-resize
        x_end = figure_shape
        y_start = 0
        y_end = resize
    if (corner == "Center"):
        half = int(figure_shape*(1-percentage))
        x_start = half
        x_end = figure_shape-half
        y_start = half
        y_end = figure_shape-half

    img = cv2.resize(img[y_start:y_end,x_start:x_end, :],
                     (figure_shape, figure_shape)).astype(np.float32)
    return img
```

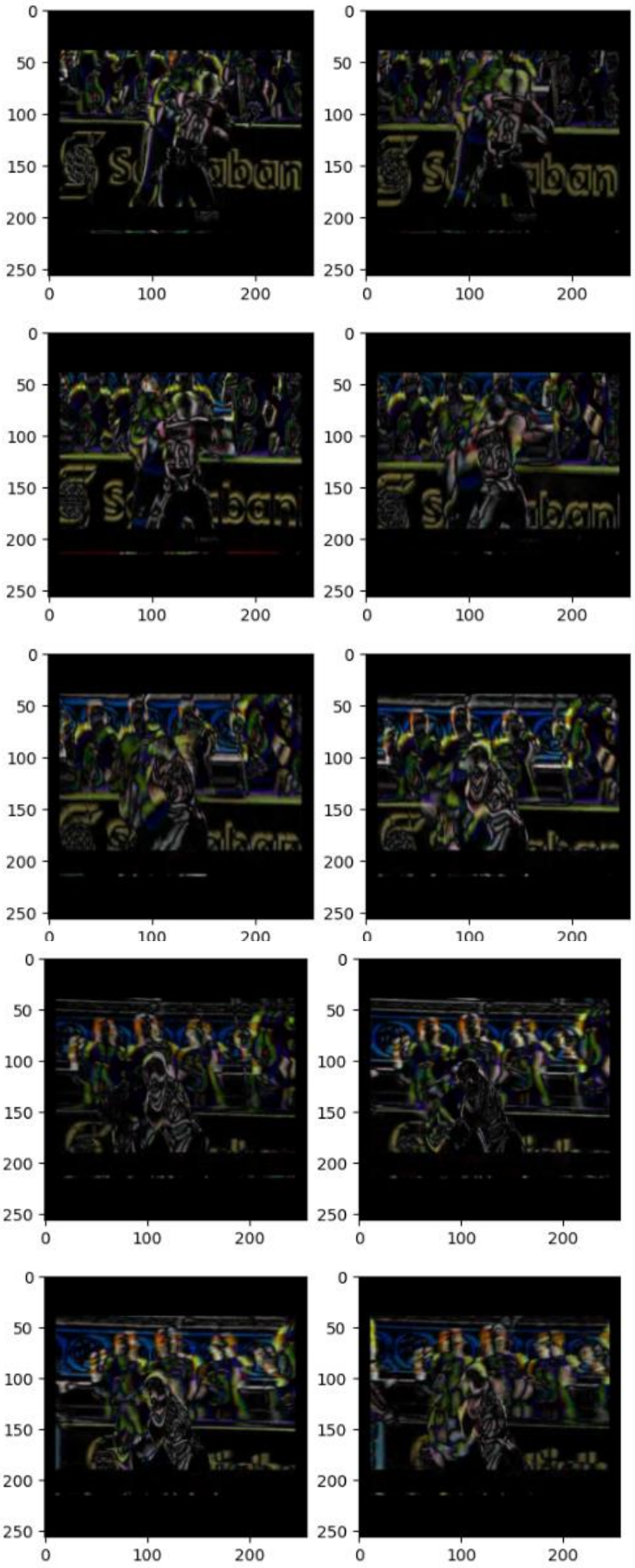
شکل ۱۵تابع **crop_img**

تابع get_sequences که در شکل بعد آمده است یکی از توابه مهم این قسمت است که در واقع از تمامی توابعی که برای مرحله پیش پردازش تعریف شده اند استفاده می کند و آنها را به نیاز بر روی فریم ها پیاده سازی می کند. علاوه بر آن درصورت لزوم image_augmentation نیز انجام می دهد که هدفش همان تولید دیتای جدید و متفاوت و robust تر کردن شبکه است که پیش تر توضیح دادیم .

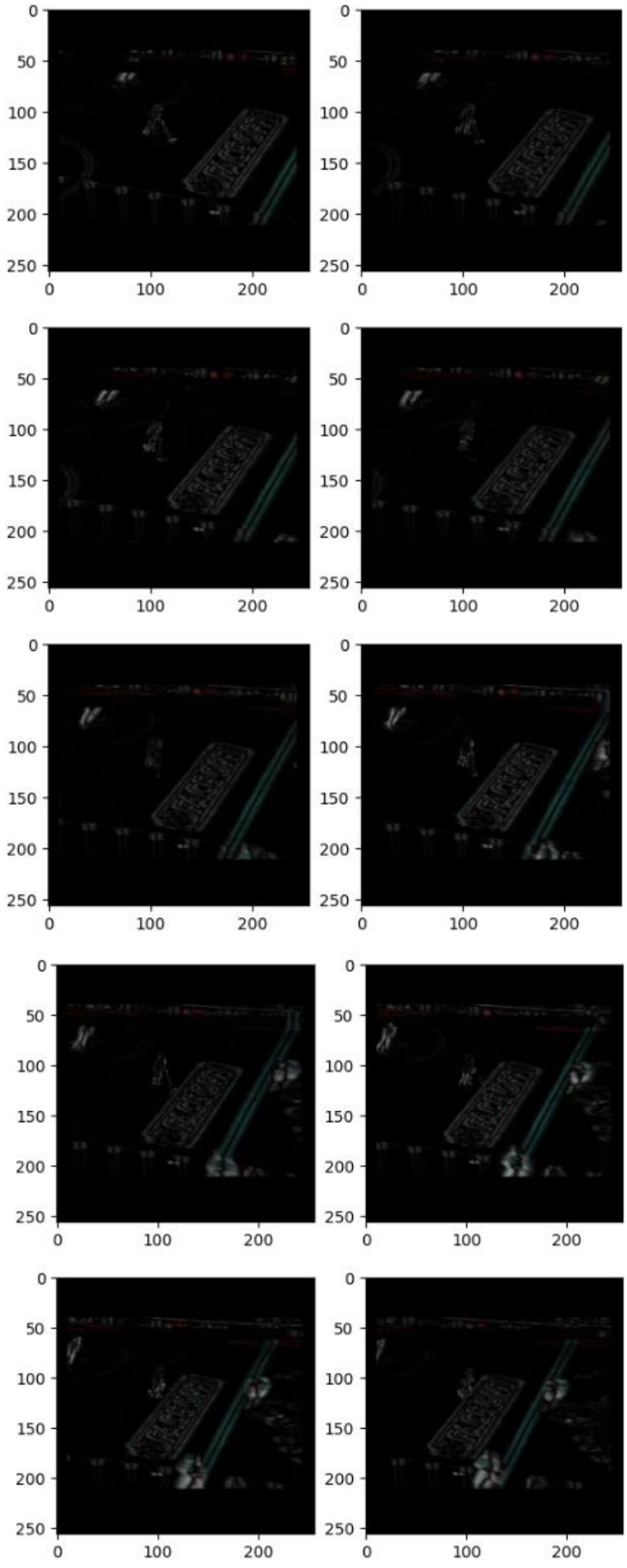
```
def get_sequences(data_paths,labels,figure_shape,seq_length,classes=1, use_augmentation = False,use_crop=True,crop_x_y=None):
    X, y = [], []
    seq_len = 0
    for data_path, label in zip(data_paths,labels):
        frames = data_path
        x = frame_loader(frames, figure_shape)
        if(crop_x_y):
            x = [crop_img__remove_Dark(x_,crop_x_y[0],
                                       crop_x_y[1],x_.shape[0],
                                       x_.shape[1],figure_shape) for x_ in x]
        if use_augmentation:
            rand = np.random.random()
            corner=""
            if rand > 0.5:
                if(use_crop):
                    corner=random.choice(corner_keys)
                    x = [crop_img(x_,figure_shape,0.7,corner) for x_ in x]
                x = [frame.transpose(1, 0, 2) for frame in x]
                if(Debug_Print_AUG):
                    to_write = [list(a) for a in zip(frames, x)]
                    [cv2.imwrite(x_[0] + "_" + corner, x_[1] * 255) for x_ in to_write]

        x = [x[i] - x[i+1] for i in range(len(x)-1)]
        X.append(x)
        y.append(label)
    X = pad_sequences(X, maxlen=seq_length, padding='pre', truncating='pre')
    if classes > 1:
        x_ = to_categorical(x_,classes)
    return np.array(X), np.array(y)
```

شکل ۱۶. تابع **get_sequences**

همچنین دو تابع data_generator و get_generators نیز در کد همراه با گزارش وجود دارد که به علت طولانی بودن تصویر آن را نیاوردیم اما در واقع چون ما از model.fit_generator هنگام فیت کردن مدل استفاده کردیم ، این دو تابع برای ایجاد این generator ها نوشته شده اند. در ادامه همانطور که در صورت سوال خواسته شده است ، ۱۰ فریم نهایی هر یک از دو کلاس fight و no_fight را نمایش داده ایم .



شکل ۱۷. فریم های ورودی به شبکه مربوط به کلاس **fight**



شکل ۱۸. فریم های ورودی به شبکه مربوط به کلاس **no_fight**

فریم های نمایش داده شده در دو تصویر قبل در واقع difference دو فریم مجاور است که پیش تر درباره لزوم آن توضیح دادیم . به جز اینکار سایر مراحل preprocessing و image_augmentation بر روی فریم ها در هنگام ترین کردن مدل و توسط generator ها صورت می گیرد.

۲-۲. پیاده سازی مدل و آموزش

تابع build_feature_extractor وظیفه دارد تا یک مدل ResNet50 از پیش آموزش دیده با imagenet را پیاده سازی کند که فریم را به عنوان ورودی دریافت کرده مراحل preprocessing را انجام می دهد و نهایتا همانطور که در تصویر زیر نمایش داده شده است یک ماتریس با ابعاد 2048*8*8 به عنوان خروجی می دهد که در واقع همان ماتریس ویژگی های استخراجی از هر فریم است.ResNet50 یک مدل کانولوشنی است که همانطور که می دانیم به طور گسترده در زمینه image processing مورد استفاده قرار می گیرند. وظیفه مدل های کانولوشنی استخراجی ویژگی های مهم از تصاویر است. همچنین در این قسمت از روش transfer learning استفاده کردیم یعنی مدلی که پیش تر بر روی داده های imagenet آموزش دیده و وزن های آن تعیین شده را برای نتیجه بهتر و سریعتر برای تسک مشابه استفاده کرده .

```
def build_feature_extractor():
    feature_extractor = keras.applications.ResNet50(
        weights="imagenet",
        include_top=False,
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.resnet50.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)

    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")


feature_extractor = build_feature_extractor()
```

شکل ۱۹. تابع ایجاد **feature_extractor**

```
Model: "feature_extractor"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 256, 256, 3)]     0

 tf.__operators__.getitem_1   (None, 256, 256, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_1 (TFOpLambd  (None, 256, 256, 3)      0
 a)

 resnet50 (Functional)       (None, 8, 8, 2048)        23587712

=================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
_________________________________________________________________
```

شکل ۲۰. خلاصه ساختمان مدل **feature_extractor**

ساختمان اصلی مدل در شکل صفحه بعد نمایش داده شده است. لایه ورودی مدل دنباله ای از ۱۰ فریم با سایز 256 را دریافت کرده وارد مدل feature_extractor که پیش تر توضیح دادیم می کند . سپس یک لایه conv دیگه قرار می دهیم تا dimension را کمی کاهش داده و ویژگی استخراجی فریم ها را یکی کند. خروجی آن 256*8*8 خواهد بود و سپس برای مرحله بعد آن ها را به یک بردار 16384 تایی تبدیل کی می کنیم. سپس دو لایه dense به ترتیب با تعداد256 و 10 نورون و با relu activation function قرار می دهیم . از آنجایی که مدل یک binary_classifier است و قرار است داده ها را به یکی از دو کلاس fight و یا no_fight طبقه بندی کند ، برای لایه خروجی یک dense با یک نورون با تابع فعالساز sigmoid و تابع loss ، binary_crossentropy قرار می دهیم. نکته ای که باید به آن توجه شود این است که ما قرار نیست وزن های resnet50 را fine-tune کنیم و میخواهیم از همان وزن های imagenet استفاده کنیم، لذا لایه های feature_extractor را freeze می کنیم. تعداد پارامتر های trainableو non_trainable این مدل نیز در تصویر زیر آمده است . پارامتر ها non_trainable متعلق به مدل feature_extractor هستند که همانطور که مشاهده می کنید تعدادشان بسیار زیاد است (حدود 38M) و اگر قرار بود از اول این پارامتر ها آموزش داده شوند به زمان و منابع بسیار زیادی نیاز بود. همچنین به هنگام آموزش مدل از callback_function ها نیز مثل EarlyStopping و ReduceLROnPlateau استفاده کردیم که در کد وجود دارند.

```
# Model structure

seq_len=10
size=256
dropout = 0.0

input_layer = Input(shape=(seq_len, size, size, 3))
ResNet50_base_model = ResNet50(weights='imagenet', include_top=False)
cnn = TimeDistributed(ResNet50_base_model)(input_layer)

lstm= ConvLSTM2D(256, (3, 3), padding='same', return_sequences=False)(cnn)

#lstm = MaxPooling2D(pool_size=(2, 2))(lstm)
flat = Flatten()(lstm)

flat = BatchNormalization()(flat)
flat = Dropout(dropout)(flat)
linear = Dense(1000)(flat)

relu = Activation('relu')(linear)
linear = Dense(256)(relu)
linear = Dropout(dropout)(linear)
relu = Activation('relu')(linear)
linear = Dense(10)(relu)
linear = Dropout(dropout)(linear)
relu = Activation('relu')(linear)

activation = 'sigmoid'
loss_func = 'binary_crossentropy'
classes=1
dropout = 0.0

predictions = Dense(classes,  activation=activation)(relu)

model = Model(inputs=input_layer, outputs=predictions)

for layer in ResNet50_base_model.layers:
    layer.trainable = False

model.compile(optimizer='RMSprop', loss='binary_crossentropy', metrics=['accuracy'])
```

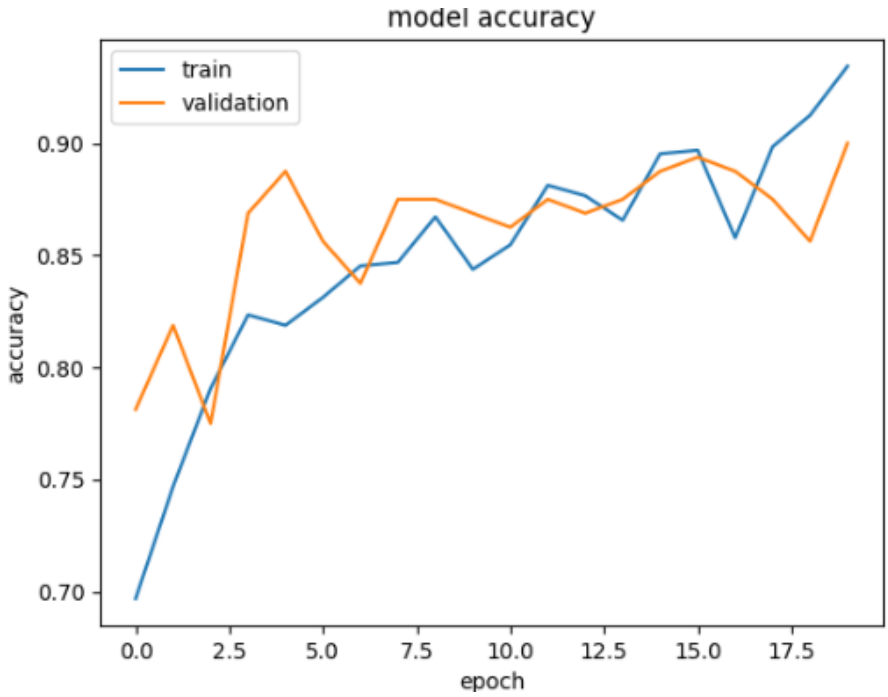
شکل ۲۱. ساختمان مدل اصلی

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 10, 256, 256, 3)  0
                             ]

 time_distributed (TimeDistr  (None, 10, 8, 8, 2048)   23587712
 ibuted)

 conv_lstm2d (ConvLSTM2D)    (None, 8, 8, 256)         21234688

 flatten (Flatten)           (None, 16384)             0

 batch_normalization (BatchN  (None, 16384)            65536
 ormalization)

 dropout (Dropout)           (None, 16384)             0

 dense (Dense)               (None, 1000)              16385000

 activation (Activation)     (None, 1000)              0

 dense_1 (Dense)             (None, 256)               256256

 dropout_1 (Dropout)         (None, 256)               0

 activation_1 (Activation)   (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

 dropout_2 (Dropout)         (None, 10)                0

 activation_2 (Activation)   (None, 10)                0

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 61,531,773
Trainable params: 37,911,293
Non-trainable params: 23,620,480
_________________________________________________________________
```

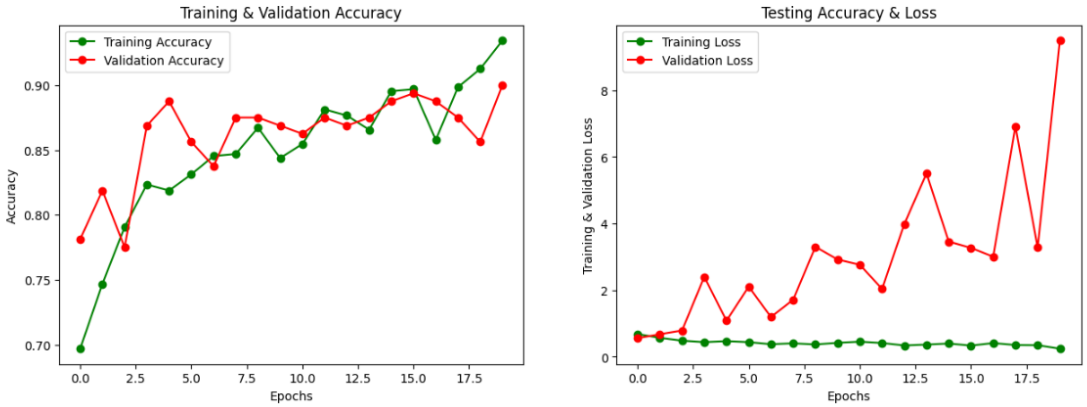
شکل ۲۲. خلاصه ساختمان و پارامتر های مدل اصلی

۲-۳. نتایج

ابتدا در نمودار های زیر نتایج حاصل از اجرای مدل در مرحله training را بررسی می کنیم. در نمودار اول Accuracy مدل بر روی داده های train وvalidation در طی ۲۰ ایپاک نمایش داده شده است. همانطور که مشخص است روند Accuracy مدل بر روی داده های train و validation صعودی بوده است و مدل توانسته به دقت خوبی دست یابد. همچنین اخلاف دقت مدل برای داده های train و validation زیاد نیست که این نشانه خوبی است از اینکه مدل overfit نشده است . البته با نگاه به نمودار loss برای این دو دیتاست ، اختلاف زیادی را می بینیم که این احتمال را میدهد که مدل تا حدی overfit شده باشد.

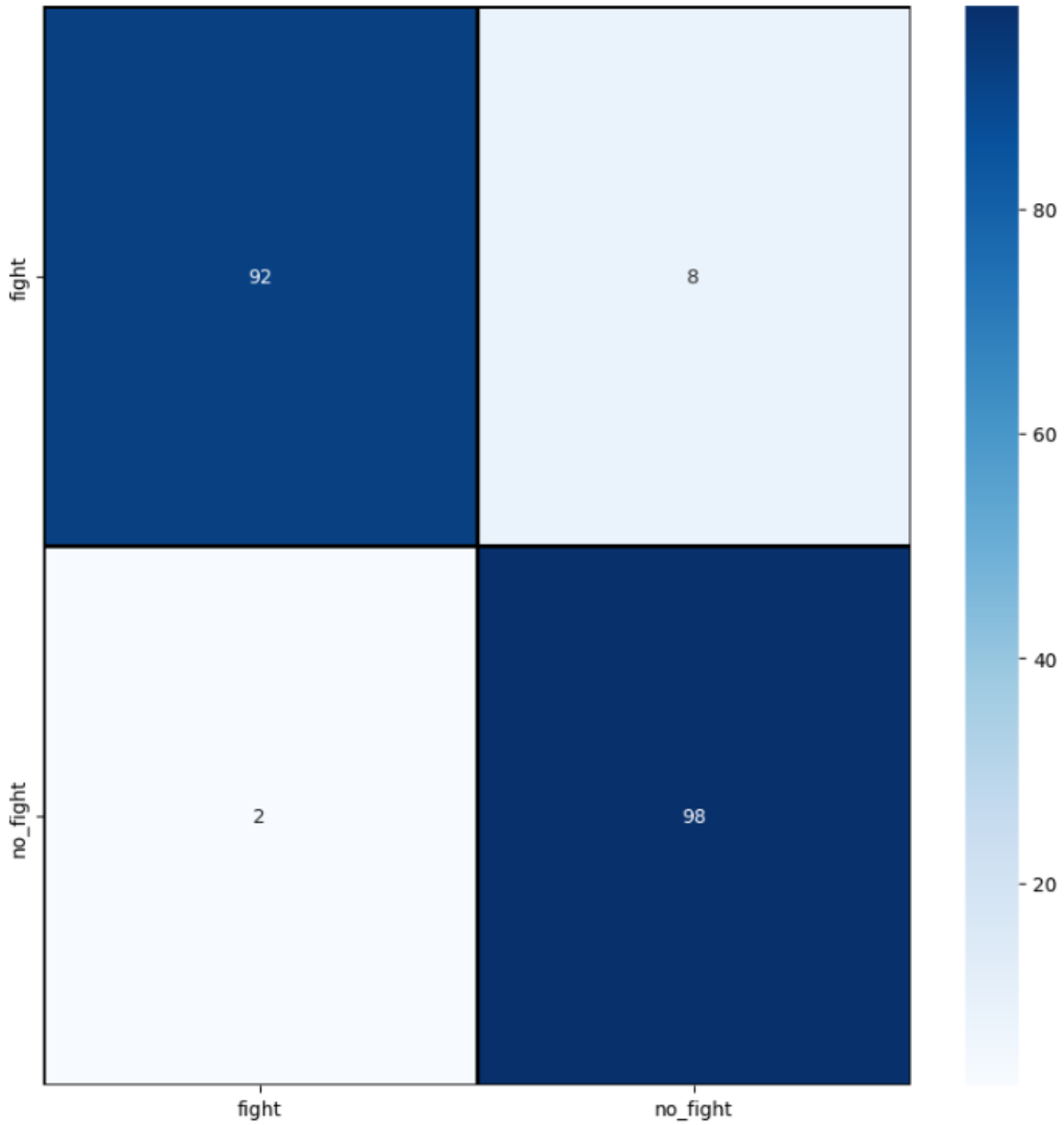


شکل ۲۴. دقت مدل در آموزش برای داده های **train, validation**



شکل ۲۳. نمودار دقت و **loss** مدل بر روی داده های **train** و **validation**

در ادامه به بررسی نتایج مدل بر روی داده های test می پردازیم و از چندین روش و متریک برای این ارزیابی استفاده می کنیم. اولین روش استفاده از ماتریس آشفتگی است که در شکل زیر نمایش داده شده است. از آنجایی که قطر اصلی این ماتریس تعداد بسیار بیشتری دارد می توان نتیجه گرفت که مدل در پیش بینی کلاس ویدئو ها عملکرد بسیار خوبی داشته است .



شکل ۲۵. ماتریس آشفتگی

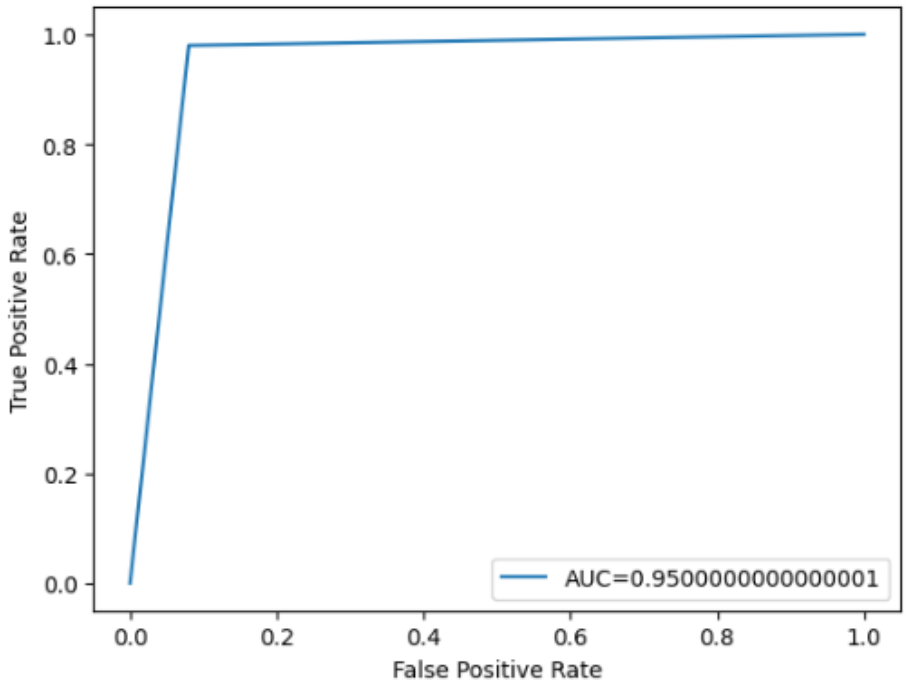
سپس در تصویر زیر مقادیر precision,recall,f1-score برای دو کلاس نمایش داده شده است که با توجه به آنچه در ماتریس آشفتگی دیدیم قابل پیش بینی بود و همانطور که مجددا می بینیم مدل عملکرد بسیار خوبی توانسته ارائه دهد به خصوص در پیش بینی کلاس فیلم ها مربوط به کلاس no_fight.

	precision	recall	f1-score	support
0	0.98	0.92	0.95	100
1	0.92	0.98	0.95	100
accuracy			0.95	200
macro avg	0.95	0.95	0.95	200
weighted avg	0.95	0.95	0.95	200

شکل ۲۶. نتایج مدل بر روی داده های **test**

در آخر نیز

ROC_Curve را برای مدل طراحی کردیم که همانطور که مشخص شده توانسته AUC = 0.95 کسب کند که نتیجه درخشانی استو همانطوز که میدانیم هرچقدر مساحت زیر اسن نمودار به ۱ نزدیک تر باشد نشان از عملکرد بهتر مدل در خصوص انجام classification است.



شکل ۲۷. نمودار **ROC**