UNIVERSITÀ DEGLI STUDI DI BARI
"ALDO MORO"

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Informatica

Caso di Studio del Corso in Modelli e Metodi Per La Sicurezza delle Applicazioni

ANOMALY DETECTOR FOR PEDESTRIAN AREAS

Prof. Donato Impedovo

Giuseppe Tanzi

ANNO ACCADEMICO 2020/2021

Sommario

Introduzione

Il rilevamento delle anomalie è un processo che si occupa del problema di individuare elementi o dati che non seguono gli schemi definiti dalla restante parte di essi che costituiscono quindi ciò che il sistema considera dati "normali". Il compito è distinguere gli elementi buoni da quelli anomali. Questo può essere definito come un problema di classificazione binaria e come tale risolto con tecniche di apprendimento supervisionato.

Tuttavia, le classi possono essere molto sbilanciate. In un processo di produzione industriale, in cui ogni giorno vengono prodotti milioni di pezzi, l'1% della produzione potrebbe essere difettoso. Un approccio di apprendimento supervisionato soffrirebbe chiaramente di questo squilibrio. Gli auto-encoder, tuttavia, sono perfetti per questa situazione, perché possono essere addestrati su parti normali e non richiedono dati annotati. Una volta addestrato, possiamo dargli in input un'immagine e confrontare l'output dell'*autoencoder* con l'input. Maggiore è la differenza, più è probabile che l'input contenga un'anomalia.

Il sistema che propongo usufruisce di un autoencoder per identificare anomalie nelle zone pedonali, come ad esempio bikers, skater, furgoni, ecc… Un autoencoder, altro non è, che una rete neurale la quale restituisce in output ciò che ha ricevuto in input. Quindi, si può ben intuire, che se non dovesse riconoscere l'input avuto in ingresso, non saprebbe ricostruirlo e ciò può essere identificato come anomalo. L'obiettivo è quello di evidenziare anomalie presenti in flussi di videosorveglianza in una zona pedonale, quali bikers, skaters, camminate su manto erboso, pedoni che portano un carrello, furgoni e persone su sedie a rotelle.

L'algoritmo, che si propone di seguito, si basa sull'architettura sviluppata dagli autori *Yong Shean Chong et al.* nell'articolo *Abnormal Event Detection in Video* [1] nel quale si descrive un'architettura spaziotemporale con l'obiettivo di individuare anomalie nei flussi video di sorveglianza delle zone pedonali. L'architettura che propongono *Yong Shean Chong et al.* è composta da due principali componenti: una per la rappresentazione delle feature spaziali e l'altra per apprendere l'evoluzione temporale delle feature spaziali. Questa architettura è stata utilizzata nel sistema con l'intento di poter risolvere il problema delle classi sbilanciate tra anomalo e non-anomalo e individuare al meglio le anomalie con l'aiuto delle Convolutional LSTM, le quali analizzano 10 frame alla volta, al fine di imparare pattern regolari nei video e prevedere frame successivi.

Requisiti di Sistema

Strumenti utilizzati

Per lo sviluppo del sistema è stato utilizzato un notebook HP:

- CPU: i5 11th generation
- GPU: Nvidia GeForce MX350
- 12GB RAM

Il sistema è stato sviluppato interamente in linguaggio Python 3.6. Per facilitare l'interazione con le feature di Python, si è deciso di interagire con un ambiente di sviluppo open-source per Python, Pycharm sviluppato da IntelliJ.

Per eseguire l'allenamento della rete neurale su GPU, è richiesta l'installazione di *CUDA 10.1* e *cudnn for CUDA 10.1 – v. 7.6.5.32*.

Librerie utilizzate

- Mxnet 1.6.0
- Mxnet-cu101 1.5.0
- Matplotlib 3.3.4
- Pillow 8.2.0
- Scipy 1.5.4
- Numpy 1.16.6

Progettazione

Gli auto-encoder sono costituiti da due parti: un codificatore che codifica i dati di input utilizzando una rappresentazione ridotta e un decodificatore che tenta di ricostruire i dati di input originali dalla rappresentazione ridotta. La rete è soggetta a vincoli che obbligano l'auto-encoder ad apprendere una rappresentazione compressa del training set. Lo fa in modo non supervisionato ed è quindi più adatto per problemi legati al rilevamento di anomalie.

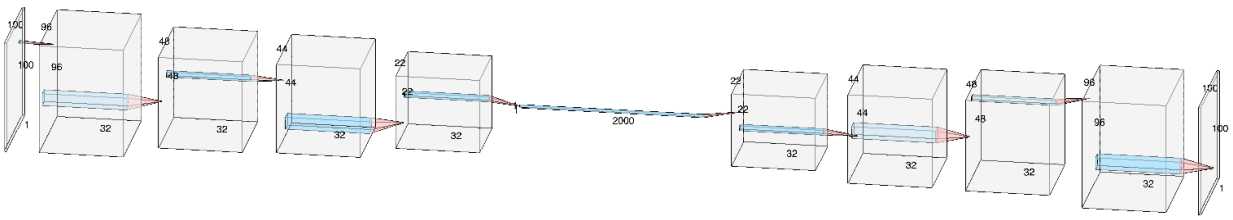

Figura 1 – Autoencoder

Un problema del *Convolutional AutoEncoder Standard* è che non tiene conto dell'aspetto temporale della sequenza di immagini. Pertanto, l'identificazione di alcune anomalie come una persona che si muove più velocemente della media non può essere facilmente rilevata.

Per questo motivo si è scelto di utilizzare un'architettura chiamata *Convolutional spatiotemporal autoencoder,* proposta nell'articolo [1], nella quale vengono impilati 10 frame alla volta e tramite convoluzioni successive vengono estratte le feature principali. L'architettura della rete è visibile in *Figura 2*.

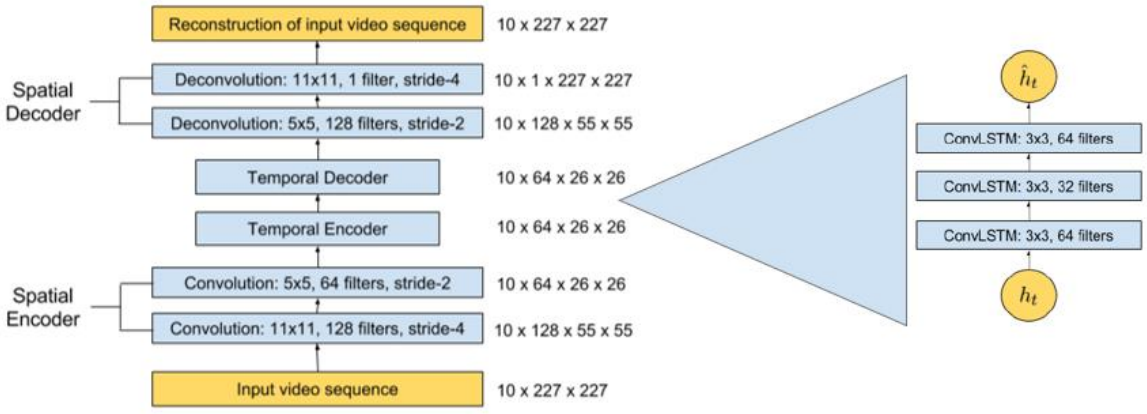

Figura 2 - Architettura della Rete: i numeri a destra denotano l'output di ogni livello.

Vengono innanzitutto estratte le feature principali tramite convoluzioni successive. Dopodiché, viene utilizzato un *Covolutional LSTM* per poter lavorare su sequenze di frame, nel nostro caso 10, e prevedere quindi frame successivi, imparando pattern regolari nei video. Infine, il decoder ricostruisce l'immagine avuta in input.

Il modello è addestrato con video costituiti solo da scene normali, con l'obiettivo di ridurre al minimo l'errore di ricostruzione tra il video in input e il video in output ricostruito dal modello appreso. Dopo che il modello è stato adeguatamente addestrato, il video normale dovrebbe avere un errore di ricostruzione basso, mentre il video costituito da scene anomale dovrebbe avere un elevato errore di ricostruzione. Sogliando sull'errore prodotto da ciascun test dei video in ingresso, il nostro sistema sarà in grado di rilevare quando si verifica un evento anomalo.

Implementazione

Per l'implementazione del sistema si è deciso di utilizzare la sottocartella *UCSDped1* per il train e il test dell'autoencoder.

Per il train del sistema sono stati utilizzati i seguenti iperparametri:

- *Epoche*: 2
- *Batch Size*: 8
- *Loss*: L2 Loss
- *Optimizer*: Adam
    - *Learning rate*: 0.0001
    - *Weight decay*: 0.00001
    - *Epsilon*: 0.000001

Per lo sviluppo del sistema sono state utilizzate le seguenti librerie:

- *Mxnet*, framework open source, utilizzato per definire e addestrare la rete neurale;
- *Matplotlib*, libreria utilizzata per poter colorare le anomalie identificate nei frame e visualizzarle;
- *Pillow*, libreria utilizzata per poter caricare i frame e ridimensionarli;
- *Scipy* e *Numpy*, librerie utilizzate per lavorare con i frame e identificare le anomalie.

La rete è stata implementata come in *Figura 3*:

```
class ConvLSTMAE(gluon.nn.HybridBlock):
    def __init__(self, **kwargs):
        super(ConvLSTMAE, self).__init__(**kwargs)
        with self.name_scope():
            self.encoder = gluon.nn.HybridSequential()
            self.encoder.add(gluon.nn.Conv2D(128, kernel_size=11, strides=4, activation='relu'))
            self.encoder.add(gluon.nn.Conv2D(64, kernel_size=5, strides=2, activation='relu'))

            self.temporal_encoder = mx.gluon.rnn.HybridSequentialRNNCell()
            self.temporal_encoder.add(gluon.contrib.rnn.Conv2DLSTMCell((64, 26, 26), 64, 3, 3, i2h_pad=1))
            self.temporal_encoder.add(gluon.contrib.rnn.Conv2DLSTMCell((64, 26, 26), 32, 3, 3, i2h_pad=1))
            self.temporal_encoder.add(gluon.contrib.rnn.Conv2DLSTMCell((32, 26, 26), 64, 3, 3, i2h_pad=1))

            self.decoder = gluon.nn.HybridSequential()
            self.decoder.add(gluon.nn.Conv2DTranspose(channels=128, kernel_size=5, strides=2, activation='relu'))
            self.decoder.add(gluon.nn.Conv2DTranspose(channels=10, kernel_size=11, strides=4, activation='sigmoid'))

    def hybrid_forward(self, F, x, states=None, **kwargs):
        x = self.encoder(x)
        x, states = self.temporal_encoder(x, states)
        x = self.decoder(x)

        return x, states
```

Figura 3 - Autoencoder con LSTM

Prima di allenare la rete, si sono prelevate le immagini dalla cartella di *train* in formato 158x238 pixels e sono state ridimensionate a 227x227 pixels e normalizzate, dividendo ogni pixel per 255, facendo sì che abbiano dei valori compresi tra 0 e 1 al fine di ridurre il calcolo computazionale durante le operazioni di convoluzione delle immagini, come possiamo notare in *Figura 4:*

```
# create dataloader (batch_size, 10, 227, 227)
def create_dataset(path, batch_size, shuffle, augment=False):
    files = sorted(glob.glob(path))
    if augment:
        files = files + files[2:] + files[4:] + files[6:] + files[8:]
    data = np.zeros((int(len(files) / 10), 10, 227, 227))
    i, idx = 0, 0
    for filename in files:
        im = Image.open(filename)
        im = im.resize((227, 227))
        data[idx, i, :, :] = np.array(im, dtype=np.float32) / 255.0
        i = i + 1
        if i > 9:
            i = 0
            idx = idx + 1
    dataset = gluon.data.ArrayDataset(mx.nd.array(data, dtype=np.float32))
    dataloader = gluon.data.DataLoader(dataset, batch_size=batch_size, last_batch="rollover", shuffle=shuffle)

    return dataloader
```

Figura 4 - Creazione del dataloader

Per evitare una situazione di *underfitting*, si è deciso di aumentare la dimensione del dataset aggiungendo duplicati di frame di immagini. Inoltre, come suggerisce sono stati impilati 10 frame alla volta, da dare successivamente in input alla rete.

In *Figura 5* si ha l'identificazione delle anomalie: per poter identificare le anomalie nei frame, si riceve l'immagine in output dall'encoder e si calcola la differenza tra l'immagine inviata in input e l'immagine ricevuta in output, creando una mappa di pixel *H* con un kernel di convoluzione 4x4. Se un valore di pixel in H è maggiore di una data soglia, verrà contrassegnato come anomalia; dunque saranno contrassegnati anomali anche i pixel ad esso vicini.

```
for image in dataloader:
    # perform inference
    image = image.as_in_context(ctx)
    states = model.temporal_encoder.begin_state(batch_size=1, ctx=ctx)
    reconstructed, states = model(image, states)
    # compute difference between reconstructed image and input image
    reconstructed = reconstructed.asnumpy()
    image = image.asnumpy()
    diff = np.abs(reconstructed - image)

    # we need to compute the regularity score per pixel
    image = np.sum(image, axis=1, keepdims=True)
    reconstructed = np.sum(reconstructed, axis=1, keepdims=True)
    diff_max = np.max(diff, axis=1)
    diff_min = np.min(diff, axis=1)
    regularity = diff_max - diff_min
    # perform convolution on regularity matrix
    H = signal.convolve2d(regularity[0, :, :], np.ones((4, 4)), mode='same')

    # if neighboring pixels are anomalous, then mark the current pixel
    x, y = np.where(H > threshold)
```

Figura 5 - Identificazione delle anomalie

Sperimentazione e analisi dei risultati

Dataset

Per lo sviluppo del sistema, è stato utilizzato il seguente dataset UCSD Anomaly Detection Dataset.

Il dataset di rilevamento delle anomalie UCSD è stato acquisito con una telecamera fissa montata ad una data elevazione, affacciata sui passaggi pedonali. La densità di folla nelle passerelle era variabile, da sparsa a molto affollata. Nell'impostazione normale, il video contiene solo pedoni.

Gli eventi anomali sono dovuti a:

- la circolazione dei soggetti non pedonali nei camminamenti
- modelli anomali di movimento dei pedoni

Una volta scaricato il dataset si troveranno due cartelle, *Peds1* e *Peds2:*

- *Peds1*: clip di gruppi di persone che si avvicinano e si allontanano dalla fotocamera, in una zona pedonale. Contiene 34 video di train e 35 video di test.
- *Peds2*: scene con movimento pedonale parallelo al piano della telecamera. Contiene 16 video di train e 12 video di test.

Ognuna di queste cartelle conterrà una cartella di *Train* e una di *Test.* All'interno delle cartelle di *Train* e di *Test* troveremo nuove cartelle, ognuna rappresentante un singolo video del dataset, contenenti 200 frame ciascuna rappresentanti i singoli video. Nella cartelle di *Train* sono contenuti solo clip di video normali. Ogni clip video nelle cartelle di *Test,* invece, contiene almeno un'anomalia.

Figura 6 - Esempio di frame normale

In *Figura 6* possiamo vedere un esempio di frame di un video normale, *Train001*, contenuto all'interno della cartella *Train* di *Peds1*

Figura 7 - Esempio di frame anomalo

In *Figura 7*, invece, possiamo vedere une esempio di frame di un video anomalo. In particolare possiamo notare il passaggio di un furgone all'interno della zona pedonale, la quale rappresenta un'anomalia. Il frame è preso dalla cartella *Test023* all'interno della cartella *Test* di *Peds1*.

Per lo sviluppo del sistema sono state utilizzate le clip video di *Peds1*. Le anomalie presenti nelle clip di video anomali, quindi nelle clip di *Test* sono:

- Bikers – 13 video;
- Skaters – 10 video;
- Pedoni che passeggiano sul manto erboso che circonda la zona pedonale – 2 video;
- Furgoni – 5 video;
- Persone su sedia a rotelle – 2 video;

Sono anche presenti video contenenti più di un'anomalia:

- Bikers + furgone – 1 video;
- Bikers + pedoni che passeggiano sul manto erboso – 1 video.

All'interno delle cartelle di *Test* sono presenti 10 cartelle di *ground truth* di 10 video diversi, in particolare:

- Test003
- Test004
- Test014
- Test017
- Test018
- Test020
- Test021
- Test022
- Test023
- Test024

Ognuna di queste cartelle ha il nominativo terminante con *_gt*; ad esempio *Test003_gt*. All'interno sono contenuti 200 frame rappresentanti le maschere dell'anomalia del test a cui si riferiscono.

Per lo sviluppo del sistema, non sono state utilizzate le cartelle contenenti il *ground truth* poiché non disponibili per tutti i video di *Test*, ma le anomalie sono state verificate manualmente, analizzando ogni singolo frame.

Analisi dei risultati

L'allenamento della rete neurale ha raggiunto una loss di 0.0035, un risultato abbastanza soddisfacente.

Per tutti i test video, la soglia è stata impostata ad un valore di 7, a seguito di valutazioni empiriche: con soglie dal valore più alto o più basso si sono raggiunti risultati meno soddisfacenti.

Per il calcolo delle metriche, sono stati analizzati singoli frame di ogni video manualmente, precisando se ognuno di questi fosse un *true postive*, un *false positive*, un *true negative* o un *false negative*. In particolare:

- *True Positive* se nel frame è presente un'anomalia e il sistema la identifica;
- *False Positive* se nel frame non è presente un'anomalia, ma il sistema identifica anomalie;
- *True Negative* se nel frame non è presente un'anomalie e il sistema non identifica nessuna anomalia;
- *False Negative* se nel frame è presente un'anomalia, ma il sistema non identifica anomalie o identifica altre anomalie, le quali non rappresentano un'anomalia.

Per singola anomalia il sistema ha ottenuto le seguenti performane:

Anomalia	Accuracy	Precision	Recall	F1-Score	FAR	FRR
Bikers	0.70	0.71	0.83	0.76	0.48	0.17
Skaters	0.63	0.64	0.78	0.71	0.57	0.22
Camminata sul manto erboso	0.63	0.61	0.84	0.71	0.61	0.16
Pedone con carrello	0.70	0.76	0.87	0.81	0.80	0.13
Furgone	0.79	0.78	0.95	0.86	0.50	0.05
Persona su sedia a rotelle	0.18	0.57	0.12	0.20	0.50	0.88

Tabella 1 - Metriche anomalie

Come si evince dalla *Tabella 1*, il sistema ha ottenuto le migliori performance nelle situazioni di anomalie causate da un *furgone* ottenendo un'accuratezza dello 0.79, mentre non è riuscito a ben evidenziare l'anomalia causata da una persona che cammina su una sedia a rotelle, ottenendo solo un'accuratezza dello 0.18.

Le performance totali del sistema sui vari test – video sono state le seguenti:

METRICA	VALORE
Accuracy	0.65
Precision	0.68
Recall	0.77
F1-Score	0.72
False Acceptance Rate	0.53
False Rejection Rate	0.23

Tabella 2 – Performance del sistema

Come si evince dalla *Tabella 2*, il sistema ha ottenuto un medio valore di *FAR* e un basso valore di *FRR*. Questo è un buon risultato poiché nei sistemi di *anomaly detection* è più importante ottenere un basso *FRR* rispetto ad un basso *FAR*: è meno grave segnalare anomalie, anche se non presenti, piuttosto che evitare di segnalare anomalie presenti.

Test	Anomalia	Accuracy	Precision	Recall	F1-Score	FAR	FRR
Test001	Bikers	0.70	0.78	0.64	0.70	0.22	0.36
Test002	Bikers	0.90	0.87	1.00	0.93	0.29	0.00
Test003	Bikers	0.40	0.50	0.33	0.40	0.50	0.67
Test004	Skaters	0.60	0.71	0.71	0.71	0.67	0.29
Test005	Bikers	0.90	0.88	1.00	0.94	0.40	0.00
Test006	Bikers	1.00	1.00	1.00	1.00	/	0.00
Test007	Skaters	0.90	0.90	1.00	0.95	1.00	0.00
Test008	Skaters	0.65	0.57	0.89	0.70	0.55	0.11
Test009	Camminata sul manto erboso	0.25	0.00	0.00	/	0.69	1.00
Test010	Skaters	0.60	0.67	0.77	0.71	0.71	0.23

Test011	Camminata sul manto erboso	0.75	0.69	0.90	0.78	0.40	0.10
Test012	Skaters	0.35	0.31	0.50	0.38	0.75	0.50
Test013	Pedone con carrello	0.70	0.76	0.87	0.81	0.80	0.13
Test014	Bikers e Furgone	0.85	1.00	0.85	0.92	/	0.15
Test015	Bikers	0.65	0.44	0.67	0.53	0.36	0.33
Test016	Bikers	0.50	0.44	1.00	0.62	0.83	0.00
Test017	Skaters	0.55	0.38	0.83	0.53	0.57	0.17
Test018	Furgone	0.70	0.57	1.00	0.73	0.50	0.00
Test019	Furgone	0.90	0.87	1.00	0.73	0.29	0.00
Test020	Persona su sedia a rotelle	0.30	1.00	0.22	0.36	0.00	0.78
Test021	Skaters	0.60	0.71	0.45	0.56	0.22	0.55
Test022	Persona su sedia a rotelle	0.05	0.00	0.00	/	0.75	1.00
Test023	Furgone	1.00	1.00	1.00	1.00	0.00	0.00
Test024	Skaters	0.75	0.75	0.82	0.78	0.33	0.18
Test025	Bikers	0.35	0.14	0.13	0.13	0.50	0.88
Test026	Furgone	0.75	0.72	1.00	0.84	0.71	0.00
Test027	Bikers	0.80	0.78	0.78	0.78	0.18	0.22
Test028	Bikers	0.75	0.67	0.75	0.71	0.25	0.25
Test029	Bikers	0.45	0.00	/	/	0.55	/
Test030	Bikers e Camminata	0.90	0.90	1.00	0.95	1.00	0.00

	sul manto erboso						
Test031	Bikers	0.55	0.56	0.91	0.69	0.89	0.09
Test032	Bikers	0.80	0.84	0.94	0.89	1.00	0.06
Test033	Skaters	0.65	0.73	0.67	0.70	0.38	0.33
Test034	Skaters	0.65	0.63	1.00	0.77	0.88	0.00
Test035	Furgone	0.55	0.53	0.90	0.67	0.80	0.10

Tabella 3 - Metriche sui singoli test

Nella *Tabella 3*, si può evincere come il sistema abbia ottenuto le migliori performance in situazioni anomale in cui erano presenti *Bikers* o *Furgoni*, mentre ha ottenuto performance peggiori in situazioni anomale in cui erano presenti *pedoni che camminavano sul manto erboso* o *persone su sedia a rotelle*.

Analizzando i singoli frame, si può ben intuire che il sistema non riesce a rilevare in maniera ottimale le anomalie le quali non superano la velocità normale del passo d'uomo; quindi, in situazioni dove sono presenti biciclette o furgoni, i quali camminano a velocità sostenuta, il sistema li evidenzia con risultati soddisfacenti, mentre nei casi in cui sono presenti pedoni che camminano su manto erboso o su sedia a rotelle, il sistema non riesce a rilevarli come anomalie poiché non viaggiano ad una velocità superiore a quella del passo d'uomo. Il sistema, inoltre, non riesce ad evidenziare, in maniera soddisfacente, le anomalie presenti nei frame, nel caso in cui nella zona pedonale ci sia molta affluenza pedonale, come nel caso del *Test003* nel quale il sistema ha raggiunto solo un'accuratezza del 0.40, nonostante l'anomalia presente fosse un *bikers.*

Conclusioni

In conclusione, il sistema sviluppato sfrutta le potenzialità degli algoritmi non supervisionati e, in particolare dell'autoencoder, per poter rilevare le anomalie presenti in una zona pedonale. Questo ha permesso di utilizzare un dataset non etichettato e, quindi, di non andare incontro allo sbilanciamento delle classi. Inoltre, grazie alla tecnica citata nell'articolo [1] si sono riusciti a raggiungere valori di accuratezza pari a 0.65 e di F1-Score pari a 0.72, nonché un valore di FRR pari a 0.23, dato molto importante nei sistemi di anomaly detection.

Sviluppi Futuri

In futuro, il sistema sviluppato potrà essere utilizzato in larga scala da più comuni e metropoli per l'identificazione di anomalie in zone pedonali, ma anche in diversi contesti d'uso: un esempio consiste nel riallenare il software per l'identificazione di anomalie nelle strade urbane e extraurbane. Per far ciò sarà necessario utilizzare un dataset diverso da quello utilizzato fino ad ora.

Appendice A: Guida all'utilizzo

Struttura Progetto

Il progetto è strutturato nel seguente modo:

```
|-- output
|    |-- Test001
|    |-- Test002
|    |      .
|    |      .
|    |      .
|    |-- Test035
|-- parameters
|    |-- autoencoder_ucsd_convLSTMAE.params
|--UCSD_Anomaly_Dataset
|-- .gitignore
|-- convLSTMAE.py
|-- Documentazione.pdf
|-- LICENSE
|-- main.py
|-- README.md
|-- utils.py
```

Nel seguito si dettagliano i ruoli dei diversi componenti:

- *output*: cartella in cui sono presenti i frame output dei 35 test con le anomalie evidenziate in rosso;
- *parameters*: cartella in cui è presente il file *.params* salvato dopo aver completato il train della rete neurale;
- *UCSD_Anomaly_Dataset*: dataset utilizzato per lo sviluppo del sistema
- *main.py*: file sorgente utilizzato come main del progetto;
- *convLSTMAE.py*: file sorgente utilizzato per definire la rete neurale e il corrispettivo train;
- *utils.py*: file sorgente utilizzato per definire la creazione del dataloader e del plot dei frame c on le anomalie evidenziate;
- *.gitignore*: file che specifica tutti i file che devono essere esclusi dal sistema di controllo ver sione;
- *Documentazione.pdf*: documentazione del caso di studio.

Primi passi

Per eseguire il codice è necessario scaricare il dataset al seguente link: UCSD Anomaly Detection Dataset, estrarlo e inserirlo nel *path* di progetto. Eliminare, dopodiché, la cartella di *test017* all'interno di *UCSDped1/Test* poiché corrotta.

Prima di eseguire il codice all'interno di *main.py* verificare di aver configurato un ambiente Python 3.6 e installato correttamente le seguenti librerie:

- Mxnet 1.6.0
- Mxnet-cu101 1.5.0
- Matplotlib 3.3.4

- Pillow 8.2.0
- Scipy 1.5.4
- Numpy 1.15.6

Per eseguire il codice su GPU installare i seguenti programmi:

- Cuda 10.1
- cudnn for Cuda 10.1 – v. 7.6.5.32

Se non si dispone di una GPU, modificare la riga 13 del file *main.py*, scrivendo *ctx = mx.cpu()* per poter effettuare il train con sola CPU.

Dopo aver configurato correttamente l'ambiente Python, è possibile avviare il codice presente in *main.py*. La riga *21* è commentata per non dover riallenare la rete neurale, poiché i parametri della rete sono presenti nella cartella *parameters*.

Quando sarà terminata l'esecuzione del codice *main.py* il sistema avrà creato una cartella *output* nella directory del progetto, in cui saranno presente 35 cartelle di Test, ognuna contenente 200 frame di singoli video con le relative anomalie evidenziate.

Impostazioni Avanzate

Rete Neurale

Per poter modificare parametri della rete, come numero di kernel o dimensione dell'input/output recarsi alla classe *ConvLSTMAE* del file *convLSTMAE.py*.

Allenamento

I parametri di allenamento della rete possono essere cambiati alle righe 11 e 12 del file *main.py*, per cambiare, quindi, numero di epoche e batch size. Per poter cambiare ulteriori parametri, come ad esempio l'ottimizzatore durante l'allenamento, sarà necessario recarsi alla riga 44 del file *convLSTMAE.py*. Per riallenare la rete dopo aver modificato i parametri, è necessario decommentare la riga 21 del file *main.py* e avviare il codice.

Per poter allenare la rete su un dataset diverso, importare il dataset da utilizzare nella directory di progetto e inserire il path di train e test del dataset alle righe 8 e 9 del file *main.py*.

Grafici di Output

All'interno del file *utilis.py* sono presenti righe commentate, le quali permettono di visualizzare ulteriori grafici di output:

- l'immagine di input data alla rete;
- l'immagine di output ricostruita dalla rete;
- l'immagine differenza tra quella di input e quella di output;
- le anomalie evidenziate nell'immagine di output.

Per poter visualizzare questi grafici è necessario decommentare le righe 69, 71, 73, 75, 76, 77, 78, 79, 80 e commentare la riga 70.

Bibliografia

[1] Y. H. T. Yong Shean Chong, «Abnormal Event Detection in Videos,» 2017.