03.2

Prototipo degli algoritmi e dei modelli di “nowcasting” (primo ciclo di sviluppo)

Code	O3.2
Date	16/7/2020
Type	Confidential
Participants	UNIMORE
Authors	Simone Calderara (UNIMORE), Francesco Del Buono, (UNIMORE), Francesco Guerra (UNIMORE), Fabio Lanzi (UNIMORE), Matteo Paganelli (UNIMORE), Maurizio Vincini (UNIMORE)
Corresponding Authors	Francesco Guerra

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

Indice

Abstract

Questo documento si compone principalmente di due sezioni: nella prima vengono illustrate le implementazioni effettuate nel primo ciclo di sviluppo degli algoritmi di Advanced Analytics e nella seconda le implementazioni dei metodi di Nowcasting.

Sezione A. Implementazione degli algoritmi di advanced analytics

In questa sezione vengono illustrati gli algoritmi di advanced analytics implementati nel primo ciclo di sviluppo del progetto. La descrizione include 3 sezioni: la descrizione del repository, i modelli di anomaly detection e analisi di time-series valutati nei dataset del progetto e il lavoro futuro.

1. Repository

Il repository del progetto si trova all'indirizzo https://github.com/FrancescoDelBuono/SBDIO Per il funzionamento è richiesto il compilatore Python 3.* nella distribuzione Anaconda. L'installazione del pacchetto richiede l'esecuzione delle seguenti operazioni:

```
$ cd source
$ virtualenv -p python3 venv
$ source venv/bin/activate
$ pip install -r requirements.txt
```

Il repository è strutturato nel seguente modo:

- *models* contiene gli algoritmi implementati e testati in ambito anomaly detection e forecasting
- *transforms* contiene i trasformatori ed estrattori di feature che possono essere utilizzati su serie temporali
- *evaluation* contiene le funzioni, il codice e i parametri utilizzati per ottenere le valutazioni finali
- *demo* contiene una serie di esempi su come utilizzare i modelli sia in fase di training e sia in fase di testing

2. Algoritmi di analisi dei dati

La cartella "*models*" contiene gli algoritmi implementati e testati nei dataset del progetto. Sono state valutate implementazioni di algoritmi di anomaly detection e di forecasting, anche se in questa prima fase del progetto ci si è soffermati maggiormente sulla anomaly detection. Inoltre, la cartella "*transforms*" contiene una serie di algoritmi in grado di estrarre feature significative e trasformare le serie temporali in ingresso consentendo un'esecuzione più efficace dei modelli in alcuni scenari.

2.1 Algoritmi di anomaly/novelty detection

L'obiettivo di questa tipologia di algoritmi è quello di individuare se ci sono delle anomalie di comportamento da parte dell'applicazione che si sta analizzando. Nel nostro contesto industriale l'obiettivo è quello di identificare condizioni di setup e sbalzi nei valori osservati, in modo tale da poter avvertire l'operatore di un eventuale guasto e/o risultato sbagliato nella produzione rispetto a una condizione normale della macchina.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

Più in generale, molte applicazioni richiedono la capacità di decidere se una nuova osservazione appartiene alla stessa distribuzione che si sta osservando ("*inlier*"), oppure no ("*outlier*"), le tecniche in grado di farlo prendono il nome di *Anomaly Detection* e si dividono in due categorie:

- *Outlier Detection*: metodo unsupervised in cui si identificano gli outlier come deviazioni nei dati, andando a identificare le regioni più concentrate e segnalando eventuali scostamenti.
- *Novelty Detection*: metodo semi-supervised in cui si cerca di imparare un comportamento/storia nei dati con l'obiettivo di identificare nelle nuove osservazioni un eventuale scostamento dal comportamento normale imparato.

Nella nostra situazione ci siamo focalizzati sul secondo caso, *novelty detection*, proponendo una serie di algoritmi che possono adatti anche in un contesto di *outlier detection*.

Gli algoritmi riportati nella libreria sono:

- *IsolationForest*: si tratta dell'implementazione del modello di Isolation Forest implementato dalla libreria *scikit-learn* così come introdotto in Liu, Fei Tony, Ting, Kai Ming and Zhou, Zhi-Hua. "*Isolation Forest*" Data Mining, 2008, ICDM'08, e "*Isolation-based anomaly detection*" ACM Transactions on Knowledge Discovery from Data (TKDD);

- *LOF* (*Local Outlier Factor*): implementazione fornita da *scikit-learn* dell'algoritmo Breunig, M. M., Kriegel, H. P., Ng, R. T., & J. Sander "*LOF: identifying density-based local outliers*" in ACM SIGMOD record;

- *OneClassSVM*: implementazione fornita da *scikit-learn* dell'algoritmo One Class SVM;

- *PCA*: è l'implementazione fornita dal *LogPAI Team* di algoritmi di anomaly detection basati su PCA. Il riferimento è Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael I. Jordan, "*Large-Scale System Problems Detection by Mining Console Logs*", SOSP 2009;

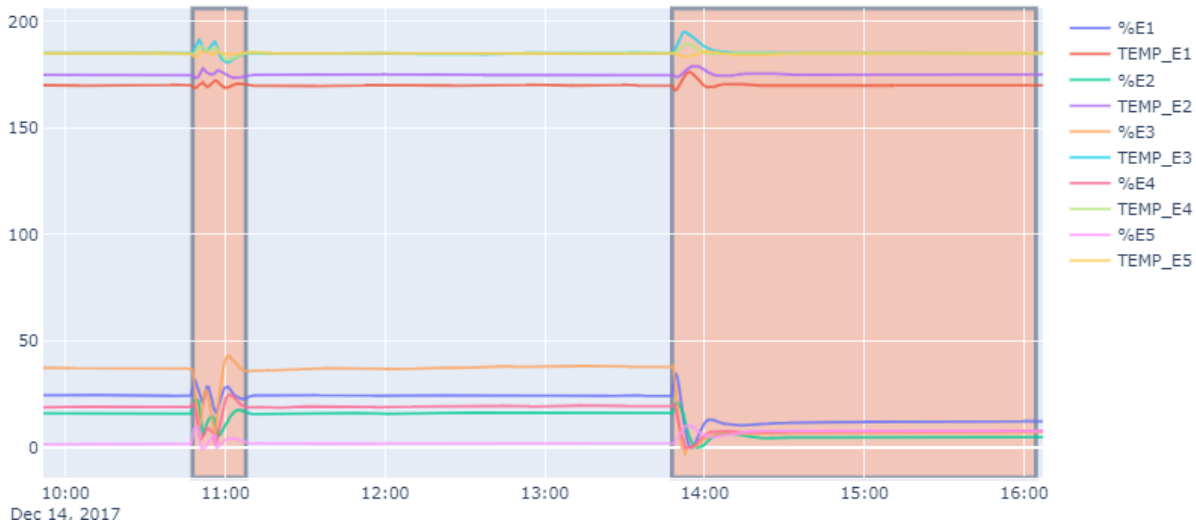
- *One Threshold*: tecnica in grado di identificare variazioni significative da una condizione di equilibrio della serie temporale che si sta analizzando;

- *Setup Clustering:* tecnica di estrazione di pattern dal comportamento normale del macchinario per l'individuazione di anomalie sulla base dello scostamento. Lo scostamento viene misurato con diversi tipi di distanze che possono essere selezionate dall'utente.

Gli algoritmi possono essere testati eseguendo il codice all'interno della cartella *demo* in cui viene mostrato un caso d'uso di ogni singolo modello di *anomaly detection,* in cui l'output viene presentato in forma di triplette dove ogni record rappresenta una condizione anomala con timestamp di inizio e di fine.

```
$ python -m demo.clustering -h
usage: clustering.py [-h] --test TEST --train TRAIN [--sep SEP]
                     [--features FEATURES [FEATURES ...]] [--save]

Run "Setup Clustering Algorithm"

optional arguments:
  -h, --help            show this help message and exit
  --test TEST           file to analyze and detect abnormal condition
  --train TRAIN         file used to learn the normal state
  --sep SEP             table separator to analyze input dataset
  --features FEATURES [FEATURES ...]
                        feature list where the algorithm is applied
  --save                to save the final algorithm result
```

```
$ python -m demo.clustering --train data/ts_normal1.CSV --test data/ts_anomaly_setup3.CSV --save

Read input data
Get params
Select features
Model initialization
Create training set
Training...
Create testing set
Testing...
Results:
+----+---------------+---------------------+---------------------+
|    | feature       | start               | end                 |
|----+---------------+---------------------+---------------------|
|  0 | all features  | 2017-12-14 10:47:48 | 2017-12-14 11:07:47 |
|  1 | all features  | 2017-12-14 13:47:48 | 2017-12-14 16:04:27 |
+----+---------------+---------------------+---------------------+

...
```



2.2 Algoritmi di forecasting

L'obiettivo di questa tipologia di algoritmi è quello di individuare le condizioni operative della macchina, prevedendo l'andamento futuro e se il processo è soggetto a usura. Gli algoritmi di forecasting si basano sia sull'analisi di serie temporali, sia sull'applicazione di algoritmi di predizione. Gli algoritmi che si sono (preliminarmente) sperimentati e che sono riportati nella libreria sono:

- *ARIMA*, *VAR*: tecniche di analisi di dati basate su time series
- *LinearRegression*, *GradientBoostingRegressor*, *RandomForestRegressor*: regressore lineare e altri regressori combinati con tecniche di ensemble.

2.3 Trasformatori

La cartella "*transformers*" contiene tecniche per modificare il dataset di partenza in modo da prepararlo per le analisi successive. Le tecniche utilizzate sono:

- *Moving Average*, *EWMA*: tecniche utilizzate nell'analisi di time series che servono a normalizzare i dati per evitare fluttuazioni transitorie nei dati. La prima è una media mobile, la seconda è una media pesata in cui i valori più recenti assumono una importanza maggiore.
- *Normalizer*, *RobustScaler*, *StandardScaler*, *PCA*. Il normalizer e gli scaler sono operatori che portano i valori assunti da un attributo in un intervallo ristretto normalmente tra 0 e 1. Questo evita che un diverso range di valori abbia effetto sulla analisi dei dati. La tecnica del PCA serve a definire gli attributi del dataset che portano un maggiore peso informativo.

3. Valutazione anomaly detection

Attualmente si è valutati il processo di anomaly detection utilizzando un dataset fornito e labellato contente serie temporali multi-variate con 25 feature. Le label andavano a indicare solamente la presenza e l'assenza delle anomalie in un chunk di dati, quindi non fornivano informazioni riguardo alla locazione temporale.

La valutazione è stata fatta sia un contesto di serie temporali uni-variate, quindi andando a considerare le feature singolarmente, e sia nel contesto multi-variato.

Valutazione anomaly detection su serie temporali uni-variate:

Model	Accuracy	Precision	Recall	F-score
PCA	90.53	99.76	84.10	91.26
SetupClustering	72.06	83.10	65.90	73.51

OneClassSVM	58.71	59.11	96.70	73.37
Isolation Forest	66.24	68.52	78.80	73.30
LOF	58.58	59.06	96.50	73.27

Valutazione anomaly detection su serie temporali multi-variate:

Model	**Accuracy**	**Precision**	**Recall**	**F-score**
PCA	100.0	100.0	100.0	100.0
SetupClustering	91.17	100-0	85.00	91.89
OneClassSVM	61.76	60.60	100.0	75.47
Isolation Forest	83.82	91.42	80.00	85.33
LOF	60.29	59.70	100.0	74.77

4. Interfaccia grafica

Un'interfaccia grafica per l'attivazione della pipeline è stata realizzata in collaborazione con il team di UNIBO.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

Sezione B. Package Python per Algoritmi e Modelli di NowCasting

Con il termine “nowcasting” facciamo riferimento all’insieme di tecniche finalizzate alla predizione delle condizioni meteorologiche all’istante di tempo attuale o comunque nell’immediato futuro (generalmente entro un massimo 5/10 minuti) circoscritte a una particolare zona d'interesse. Questo concetto si affianca spesso a quello più noto di “forecasting”, che riguarda tuttavia previsioni di accuratezza inferiore, ma relative a una finestra temporale più ampia, arrivando anche a stime di una settimana in avanti.

Sebbene a un primo sguardo nowcasting e forecasting possano apparire molto simili tra loro, le finalità di questi strumenti presentano delle sostanziali differenze. Nel caso del nowcasting, infatti, più che fare previsioni su ciò che accadrà a livello meteorologico, il fine è quello di raccogliere precise statistiche relative a uno o più fenomeni d'interesse in una zona circoscritta.

Il nowcasting è quindi un importante strumento statistico può fungere da supporto nelle analisi sul clima finalizzate a descrivere andamento, intensità e variabilità di fenomeni meteorologici, osservati a diversa scala temporale. Tuttavia, automatizzare il processo di nowcasting risulta essere un’operazione piuttosto complessa che, se affrontata tramite metodologia standard, richiede l’acquisto di strumentazione sofisticata e dai costi piuttosto elevati; per questo motivo, il nowcasting è spesso prerogativa delle stazioni meteo più attrezzate. L’idea è quindi quella di ridurre il più possibile il numero e la complessità dei sensori necessari al raccoglimento dei dati di nowcasting per rendere questa pratica maggiormente accessibile.

Il problema del nowcasting può essere efficacemente affrontato affidandosi a tecniche di Computer Vision e Deep Learning, limitando quindi la richiesta di sensori a semplici telecamere RGB. Questo tipo di approccio risulta particolarmente conveniente in termini di semplicità d’uso e di risorse impiegate, ma presenta una serie di problematiche di progettazione e implementazione che non sono facilmente affrontabili dai non esperti del settore. Per questo motivo nasce l’idea di PyNowCast, un package Python che permette di gestire algoritmi e modelli di nowcasting basati su DeepLearning in modo semplice e veloce, occupandosi in modo trasparente di tutti gli aspetti più complessi e macchinosi che caratterizzano questo tipo di tecnologia. Tramite PyCast, quindi, il nowcasting tramite Deep Learning sarà alla portata di tutti.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

1. Quick Start

Si propone di seguito una guida rapida dall'utilizzo di PyNowCast con i principali passaggi da seguire per arrivare alla classificazione di una data immagine di input.

1. Clonare il repository PyNowCast e installare i requisiti:
   - **`git clone https://github.com/FabioLanzi/PyNowCast.git`** **(se si è collaboratori del repository)**
   - scaricare il progetto utilizzando il seguente link: `https://drive.google.com/drive/folders/1Rvfqj6M56gmk5IgeueCiqAs_krTy2nzn?usp=sharing` (se non si è collaboratori del repository)
   - **`cd PyNowCast; pip install -r re quirements.txt`**
2. Organizzare il dataset come descritto alla Sezione 2; supponiamo ad esempio di porre tale dataset nella directory `/nas/dataset/nowcast_ds`
3. Allenare il feature extractor come descritto nella Sezione 3 tramite l'apposito script `train_extractor.py`
   - esempio: `python train_extractor.py --exp_name='fx_example' --ds_root_path='/nas/dataset/nowcast_ds'`
4. Allenare il classificatore come descritto nella Sezione 4 tramite l'apposito script train_classifier.py
   - esempio: `python train_classifier.py --exp_name='nc_example' --ds_root_path='/nas/dataset/nowcast_ds' --pync_file_path='/nas/pync/example.pync'`
5. Utilizzare il classificatore precedentemente allenato su un'immagine a piacere utilizzando l'apposito comando `classify` dello script `pync.py`
   - esempio: `python pync.py --pync_file_path=/nas/pync/example.pync`

Nelle sezioni che seguono, saranno illustrati nel dettaglio tutti i passaggi qui accennati, accompagnati da opportune motivazioni relative alle scelte effettuate.

2. Dataset

Come prima cosa è necessario organizzare i dati sui quali si vuole allenare il modello di nowcasting secondo il semplice schema mostrato in Figura 1.

Si avrà pertanto una directory principale, indicata con `<dataset_name>`, con un nome a piacere, che dovrà necessariamente contenere due sotto-directory denominate rispettivamente `train` e `test`, che a loro volta devono contenere le varie sotto-directory contenenti le immagini suddivise per classi.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

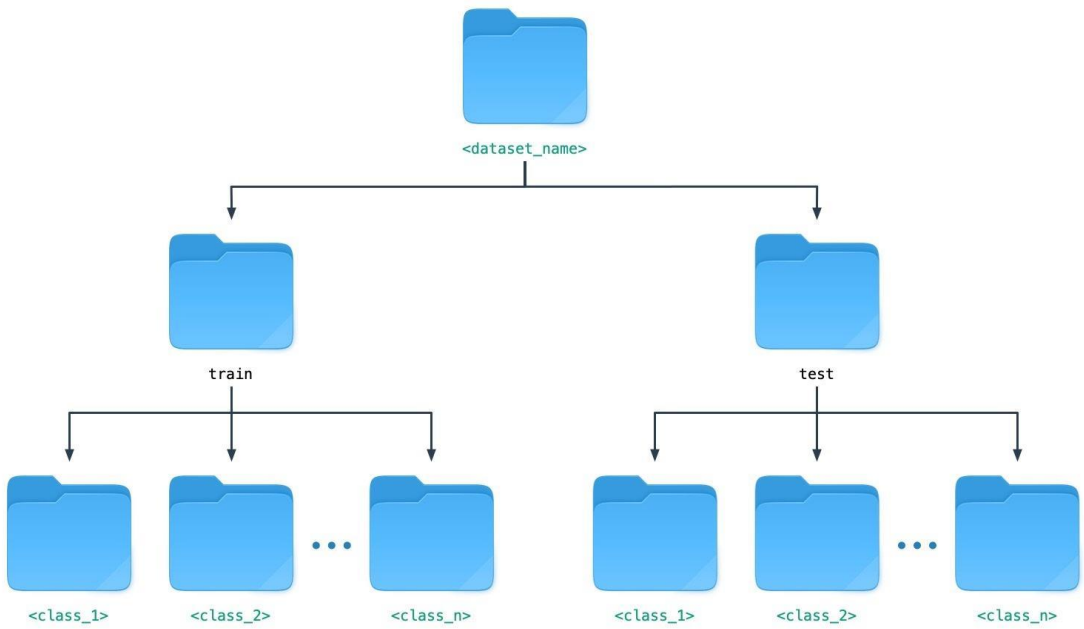


Figura 1. Struttura della directory contenente il dataset. I nomi indicati in verde in figura possono essere scelti a piacere.

NOTA: È fondamentale che tutte le immagini utilizzate per l'allenamento e la verifica del modello di nowcasting abbiano la stessa dimensione. Il package PyNowCast è stato infatti pensato per problemi di nowcasting a camera fissa, quindi si presuppone che tutte le immagini che compongono il training set e il test set provengano dalla stessa camera e presentino di conseguenza le medesime dimensioni.

2.1. Directory di Train e Test

La directory `train`, come il nome suggerisce, è quella preposta a ospitare le immagini di training, ovvero quelle sulle quali sarà allenato il modello di nowcasting; allo stesso modo, la directory `test` conterrà le immagini utilizzate per valutare le prestazioni del suddetto modello. Si noti che, per ottenere una valutazione corretta delle prestazioni del modello, gli insiemi composti dalle immagini di training e dalle immagini di test dovrebbero essere completamente disgiunti, quindi privi di immagini comuni.

Le sotto-directory `train` e `test` devono contenere a loro volta *n* sotto-directory, dove *n* (con *n* maggiore o uguale a 2) è il numero di classi scelte per lo specifico problema di nowcasting che si vuole approcciate con il presente framework. La directory di ogni classe deve contenere esclusivamente le

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

immagini relative a quella specifica classe, affiancate al più da un file `sensors.json` contenente le informazioni relative a eventuali dati aggiuntivi provenienti da appositi sensori (esempio: sensori di umidità, temperatura, pressione, ...).

Per allenare correttamente il modello di nowcasting, si consiglia di avere un training set bilanciato, quindi con un quantitativo di immagini simile per ogni classe; si consiglia inoltre di avere un numero di immagini maggiore di 1000 per ogni classe.

Per velocizzare il processo di inizializzazione del dataset, è possibile creare all'interno delle directory `train` e `test` un file di cache in formato JSON che prenderà il nome di `cache.json`. La creazione di tale file avviene in automatico quando si chiama il costruttore della classe `NowCastDS` con il parametro `create_cache=True` (*NOTA*: il valore di default è `False`).

```
training_set = NowCastDS(ds_root_path='/your/ds/root/path',
mode='train', create_cache=True)
```

```
test_set = NowCastDS(ds_root_path='/your/ds/root/path',
mode='test', create_cache=True)
```

2.2. Struttura dei File `sensor.json`

Considerando una classe con *K* immagini ed *m* valori provenienti dai sensori associati a ciascuna di esse, la struttura del relativo file opzionale `sensors.json` sarà la seguente:

```
{
  "img1_name": [x1_1, x1_2, ..., x1_m],
  "img2_name": [x2_1, x2_2, ..., x2_m],
  ...
  "imgK_name": [xK_1, xK_2, ..., xK_m]
}
```

Si tratta dunque di un file JSON in cui le chiavi sono i percorsi relativi alla directory contenente il file `sensors.json` stesso e i valori sono liste contenente gli *m* dati letti dai sensori per quella specifica immagine.

Si noti che, se si sceglie di inserire il file `sensors.json` all'interno di una directory relativa a una classe, anche tutte le altre classi devono contenerlo. Qualora per alcune immagini non fossero

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

disponibili uno o più valori relativi a un sensore, sarà sufficiente inserire al loro posto il valore `null`. Se ad esempio per l'immagine `img1_name` non si disponesse del valore 2, all'interno del JSON si avrà:

- `"img1_name": [x1_1, null, ..., x1_m]`

2.3. Verifica della Correttezza della Struttura del Dataset

È possibile verificare la correttezza della struttura del proprio dataset utilizzando lo script `check_dataset_structure.py` tramite il seguente comando, in cui si indica con `<dataset_path>` il percorso assoluto alla directory principale del dataset:

- `python chech_dataset_structure.py <dataset_path>`

Lo script verificherà la presenza di errori strutturali e li comunicherà all'utente con un apposito messaggio autoesplicativo. Saranno inoltre forniti avvertimenti su eventuali aspetti che non si ritengono ottimali per iniziare la procedura di traning; ad esempio potrebbe essere segnalata la presenza di un numero di immagini ritenuto insufficiente per una o più classi.

Gli errori saranno evidenziati con un pallino rosso e la dicitura "ERROR" e andranno necessariamente risolti prima di intraprendere la procedura di allenamento del modello di nowcasting.

Gli avvertimenti saranno evidenziati con un pallino giallo e la dicitura "WARNING"; in questo casto non sarà necessario (sebbene caldamente consigliato) risolvere la problematica indicata prima di procedere con l'allenamento del modello.

2.4. Esempio

In Figura 2. si propone un esempio di struttura della directory `train` nel caso di un problema di nowcasting a due classi, in cui, partendo dall'immagine RGB e dai dati di un sensore di temperatura si vuole verificare la presenza o l'assenza di nebbia nell'immagine in ingresso. Le sotto-directory `fog` e `no_fog` contengono rispettivamente immagini con nebbia e immagini in cui la nebbia è assente. Le immagini mostrate in figura sono state acquisite presso l'Osservatorio di Modena.

Un esempio completo che mostra la struttura di un dataset valido, seppur contenente un numero esiguo di immagini, è contenuto all'interno di questo stesso repository:

- `PyNowCast/dataset/example_ds`

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

NOTA: il dataset `example_ds` ha solo uno scopo esemplificativo e non può essere utilizzato per allenare un modello di nowcasting a causa del numero ridotto di immagini presenti.

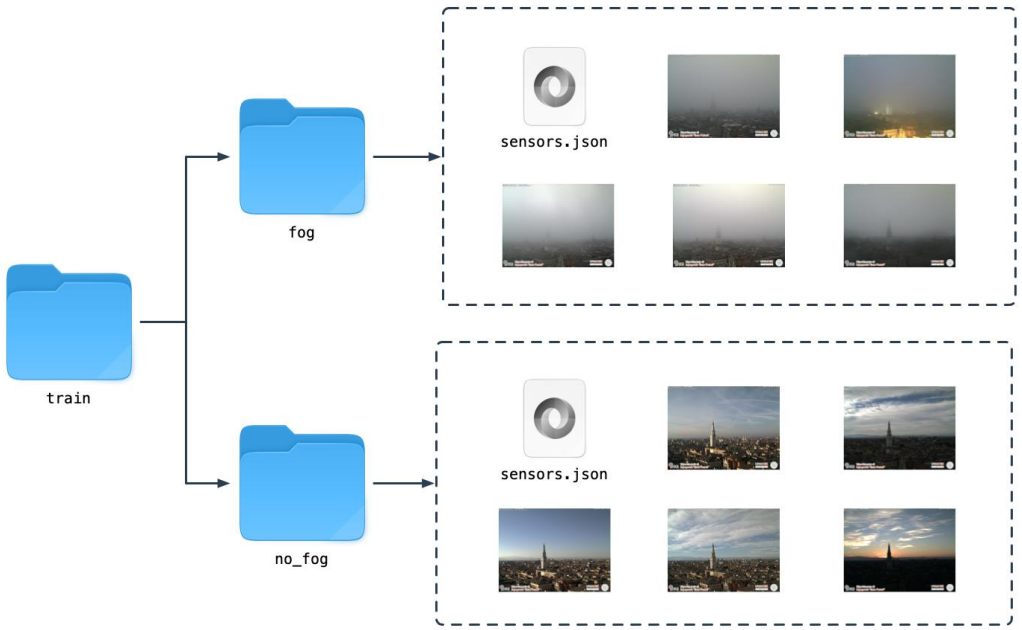


Figura 2. Esempio di struttura della directory `train` nel caso di un problema di nowcasting a due classi (classe `fog` e classe `no_fog`).

3. Feature Extractor

Il feature extractor è un componente essenziale della maggior parte dei modelli di classificazione basati su reti neurali; il suo compito è, come suggerisce il nome stesso, quello di estrarre una serie di caratteristiche che "riassumano" i tratti salienti dell'oggetto passato in ingresso, che nel nostro caso è un'immagine RGB proveniente da una camera fissa.

Nell'ambito della Computer Vision, esistono una serie di feature extractor ([1], [2], [3], [4]) standard che vengono utilizzati per un vasto numero di task, in quanto risultano molto flessibili e in grado di fornire feature di alto livello che possono venire incontro alle esigenze di problemi anche molto diversi tra loro.

Nel nostro caso specifico, tuttavia, risulta più opportuno affidarsi a un feature extractor maggiormente mirato e incentrato sulle nostre esigenze particolari. Il problema che *PyNowCast* va ad affrontare infatti è molto vincolato e i vincoli che lo contraddistinguono, se ben sfruttati, possono semplificarlo

enormemente. In questo senso, per il nostro package abbiamo deciso di orientarci verso un feature extractor personalizzato basato su un autoencoder, che sia in grado di essere allenato efficacemente in modo non supervisionato.

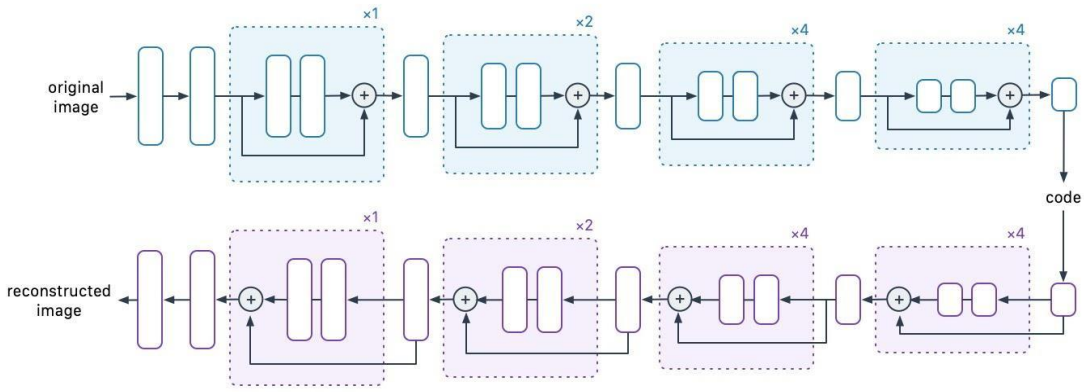


Figura 3. Schema a blocchi dell'autoencoder utilizzato durante la fase di allenamento del feature extractor; il feature extractor vero e proprio è rappresentato dalla sola parte di encoding (ramo "blu" in figura).

3.1. Autoencoder

Per adottare questa soluzione ci siamo basati su una semplice osservazione: poiché ci troviamo a trattare immagini provenienti da una camera fissa che riprende una certa porzione di paesaggio, gli elementi che variano tra un'immagine e l'altra sono essenzialmente la condizione meteorologica e le condizioni di illuminazione. Fortunatamente ciò che varia sono esattamente le feature che servono a un classificatore che si occupa di nowcasting.

In tal senso, l'uso di un autoencoder composto da un encoder e un decoder speculari risulta particolarmente appropriato. Allenando un autoencoder di questo tipo a ricostruire semplicemente le immagini di input all'uscita dell'encoder si avrà un "codice" che rappresenta per l'appunto un riassunto delle immagini di input. Trovando la giusta dimensione di tale codice, l'encoder sarà forzato a rimuovere tutte le informazioni ridondati, che nel nostro caso sono appunto le caratteristiche che non variano tra un'immagine e l'altra; al contempo dovrà preservare gli elementi mutevoli delle medesime (meteo e condizioni di illuminazione).

L'autoencoder utilizzato è mostrato in Figura 3.

3.2. Procedura di Allenamento

Per avviare la procedura di allenamento del feature extractor è possibile utilizzare lo script `train_extractor.py` con le seguenti opzioni:

- `--exp_name`: etichetta assegnata alla corrente procedura di allenamento del feature extractor; per differenziare le etichette assegnate al feature extractor da quelle assegnate al modello utilizzato per la classificazione, si suggerisce di utilizzare il prefisso `fx_`.
- `--ds_root_path`: percorso della directory principale contenete il proprio dataset.
- `--device`: tramite questa opzione è possibile selezionare il device su cui sarà effettuata la procedura di allenamento del feature extractor; i possibili valori sono i seguenti: `'cuda'` o `'cuda:<numero specifica gpu>'` per un allenamento su GPU (consigliato) o `'cpu'` per un allenamento su CPU (sconsigliato). Se non specificato, il valore di default è `'cuda'`

Si propone di seguito un esempio di chiamata:

- `python train_extractor.py --exp_name='fx_try1' --ds_root_path='/nas/dataset/nowcast_ds'`

Per gli utenti più esperti, è possibile modificare il file `conf.py` per personalizzare i parametri del training; salvo casi molto particolari, tuttavia, si suggerisce di utilizzare i parametri di default. Per completezza, si riporta la porzione del file di configurazione relativa all'allenamento del feature extractor:

```
# feature extractor settings
FX_LR = 0.0001        # learning rate used to trane the feature extractor
FX_N_WORKERS = 4      # worker(s) number of the dataloader
FX_BATCH_SIZE = 8     # batch size used to trane the feature extractor
FX_MAX_EPOCHS = 256   # maximum training duration (# epochs)
FX_PATIENCE   =  16   #  stop  training  if  no  improvement  is  seen  for
a                                  'FX_PATIENCE' number of epochs
```

4. Classificatore

Il cuore di PyNowCast è il modello utilizzato per la classificazione che si compone di due elementi fondamentali: il feature extractor di cui si è discusso nella Sezione 3 e una rete completamente connessa a 3 livelli che svolge il ruolo di classificatore vero e proprio. La struttura semplificata dell'intero modello è mostrata in Figura 4 (presupponendo un problema di classificazione a 3 classi).

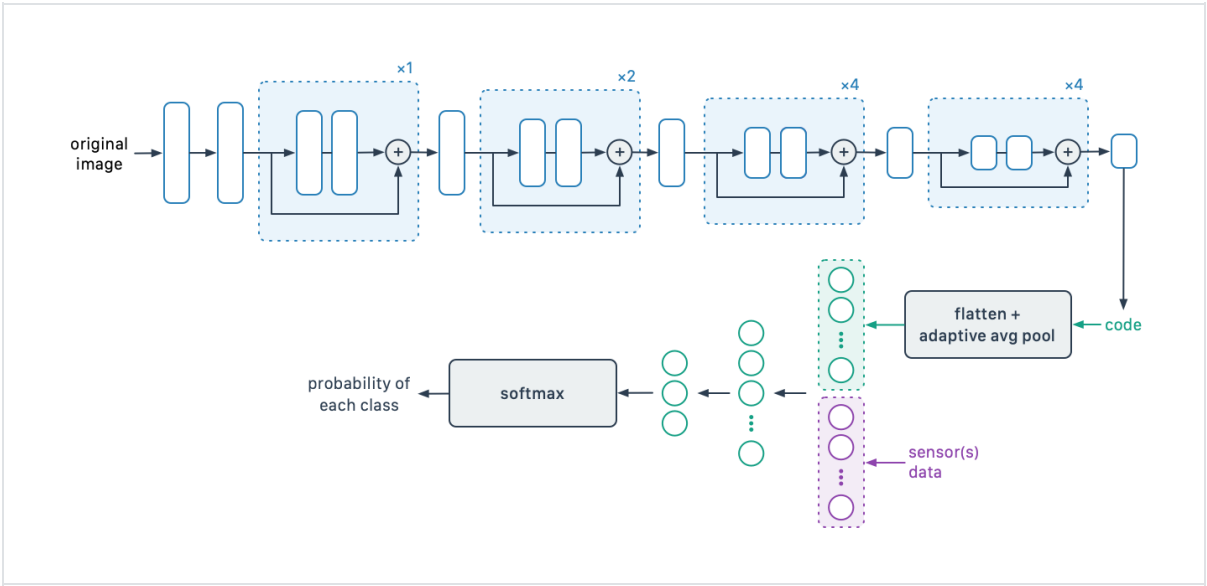


Figura 4. Struttura semplificata del modello di classificazione (feature extractor + rete completamente connessa a 3 livelli). Nel caso in figura, si considera un problema di classificazione a 3 classi.

Data un'immagine di input il feature extractor ne estrae una rappresentazione compatta (indicata con il termine "code" in Figura 4). Tale rappresentazione viene opportunamente ridimensionata e "srotolata" ottenendo un array di valori che rappresenta l'ingresso delle rate completamente connessa preposta alla classificazione. Se disponibili, anche i dati dei sensori relativi all'immagine di input sono passati in ingresso alla rete completamente connessa, affiancandosi quindi alle feature visuali. L'output finale del modello è rappresentato da valori compresi tra 0 e 1 che rappresentano le probabilità associate a ciascuna delle classi del problema specifico che si sta affrontando.

4.1. Procedura di Allenamento

Per avviare la procedura di allenamento del feature extractor è possibile utilizzare lo script `train_classifier.py` con le seguenti opzioni:

- `--exp_name`: etichetta assegnata alla corrente procedura di allenamento del classificatore; per differenziare le etichette assegnate al classificatore da quelle assegnate al feature extractor, si suggerisce di utilizzare il prefisso `nc_`.
- `--ds_root_path`: percorso della directory principale contenete il proprio dataset.
- `--pync_file_path`: percorso del file `.pync` contenente i risultati della procedura di allenamento del classificatore; se non specificato, tale file sarà salvato nell'attuale working directory e il suo nome sarà quello specificato con l'opzione `--exp_name` (più l'estensione `.pync`).
- `--device`: tramite questa opzione è possibile selezionare il device su cui sarà effettuata la procedura di allenamento del classificatore; i possibili valori sono i seguenti: `'cuda'` o `'cuda:<numero specifica gpu>'` per un allenamento su GPU (consigliato) o `'cpu'` per un allenamento su CPU (sconsigliato). Se non specificato, il valore di default è `'cuda'`

Si propone di seguito un esempio di chiamata:

- `python train_classifier.py --exp_name='nc_try1' --ds_root_path='/nas/dataset/nowcast_ds'`

Per gli utenti più esperti, è possibile modificare il file `conf.py` per personalizzare i parametri del training; salvo casi molto particolari, tuttavia, si suggerisce di utilizzare i parametri di default. Per completezza, si riporta la porzione del file di configurazione relativa all'allenamento del feature extractor:

```
# nowcasting classifier settings
NC_LR = 0.0001  # learning rate used to trane the nowcasting classifier
NC_N_WORKERS = 4  # worker(s) number of the dataloader
NC_BATCH_SIZE = 8  # batch size used to trane the nowcasting classifier
NC_MAX_EPOCHS = 256  # maximum training duration (# epochs)
NC_PATIENCE = 16 # stop training if no improvement is seen for a 'NC_PATIENCE' number of epochs
```

4.2. File .pync

I risultati della procedura di allenamento vengono salvati in un apposito file con estensione `.pync` il cui path può essere impostato dall'utente tramite l'opzione `--pync_file_path`; tale file contiene i pesi del classificatore e alcune informazioni utili in fase di evaluation del modello, come il nome delle classi considerate e la dimensione dell'array dei valori dei sensori.

Dato un file `.pync` è possibile consultare le informazioni in esso contenute utilizzando il comando `show-info` contenuto nello script `pync.py` nel modo che seghe:

- `python pync.py show-info --pync_file_path=/your/pync/file.pync`

Si mostra di seguito un esempio di output del comando di cui sopra:

```
▶ Showing info of '/your/pync/file.pync'
├── model weights (size): 10536 bytes
├── sensor data len: 3
└── classes:
  ├──[0] sun
  ├──[1] rain
  ├──[2] fog
  └──[3] snow
```

5. Classificazione di un'Immagine

Una volta ottenuto un file `.pync` in seguito all'allenamento di un classificatore, applicarlo ad un'immagine di test risulta molto semplice: è infatti sufficiente utilizzare il comando `classify` contenuto in `pync.py` specificando il path dell'immagine da classificare e il path del file `.pync` che si vuole utilizzare. Si propone di seguito un esempio di chiamata del comando.

- `python pync.py classify --img_path=/your/input/image.jpg --pync_file_path=your/pync/file.pync`

In output si ottengono quindi le probabilità associate a ciascuna delle classi contemplate, con un'apposita indicazione della classe a probabilità più alta. Si veda l'esempio seguente.

```
▶ Classifying image '/your/input/image.jpg'
├──[0]──[sun]:  0.10 %
├──[1]──[rain]: 84.11 % ◀●
├──[2]──[fog]: 14.17 %
```

```
└──[3]──[snow]:  1.62 %
```