ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

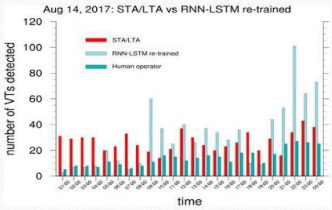
Харківський національний університет радіоелектроніки

Моделі глибокого навчання для прогнозування часових рядів сейсмологічних даних

Виконала:
маг. гр. СПм-22-5 Тимошенко Д. О.
Науковий керівник:
доцент каф. ЕОМ Іващенко Г. С.

Актуальність проблеми

	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	sensor_7	sensor_8	sensor_9	sensor_10
0	614.0	653.0	35.0	478.0	-314.0	0.0	217.0	121.0	243.0	632.0
1	561.0	718.0	26.0	491.0	-269.0	-60.0	153.0	-55.0	131.0	363.0
2	496.0	846.0	18.0	423.0	-166.0	-160.0	30.0	40.0	40.0	81.0
3	429.0	1000.0	14.0	415.0	-33.0	-223.0	-87.0	-131.0	-60.0	-203.0
4	368.0	1098.0	19.0	419.0	108.0	-95.0	-219.0	-55.0	-152.0	-456.0
5	324.0	1073.0	26.0	368.0	256.0	-41.0	-345.0	232.0	-236.0	-655.0
6	289.0	949.0	46.0	317.0	355.0	-313.0	-381.0	286.0	-280.0	-813.0
7	259.0	770.0	15.0	283.0	369.0	-255.0	-280.0	602.0	-307.0	-932.0
8	240.0	472.0	9.0	207.0	327.0	-75.0	-71.0	458.0	-328.0	-1018.0
9	239.0	22.0	-21.0	43.0	265.0	-45.0	106.0	537.0	-363.0	-1066.0

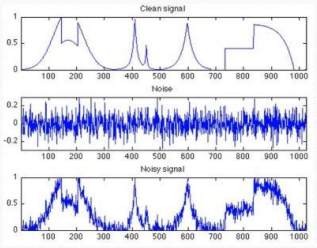


- На сьогоднішній день прогнозування часу до виверження вулканів є актуальним завданням, від точності результатів якого залежить велика кількість людських життів. Через недостатню точність поширених методів прогнозування, актуальним є використання нових підходів, що використовують джерела інформації, які зазвичай надають дані у вигляді часових рядів.

2

Методи прогнозування часових рядів

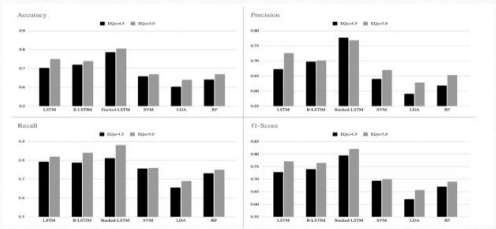
- експоненціальне згладжування, ARIMA та моделі простору станів;
- регресійне прогнозування: лінійна та нелінійна регресія, дерева рішень, випадкові ліси;
- штучні нейронні мережі;
- ансамблеві методи, методи вилучення ознак і наскрізні методи.

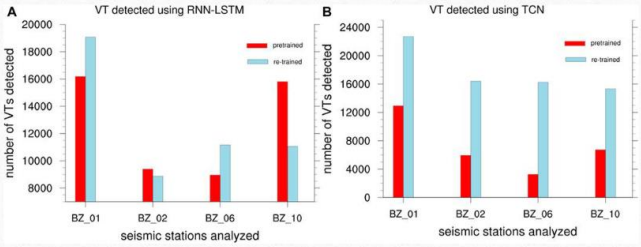


3

Сучасні дослідження

- моніторинг тектонічних землетрусів вулканів на основі архітектур CNN;
- методи глибокого навчання на основі LSTM для прогнозування землетрусів з використанням іоносферних даних;
- дослідження прогнозування виверження вулкану за допомогою муографії з використанням згорткової нейронної мережі.





4

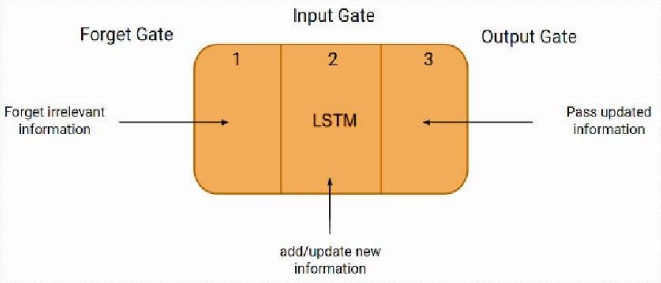
Постановка задачі

- Метою роботи є дослідження методів прогнозування сейсмічної активності вулканів за допомогою моделей глибокого навчання.
- На основі аналізу існуючих досліджень, для вирішення задачі були обрані моделі штучних нейронних мереж – LSTM та CNN.
- Досліджується залежність метрики MSE та швидкості навчання мережі від наступних гіперпараметрів: розмір прихованих станів, значення dropout та розмір згорткового ядра.

5

Довга короткострокова пам’ять

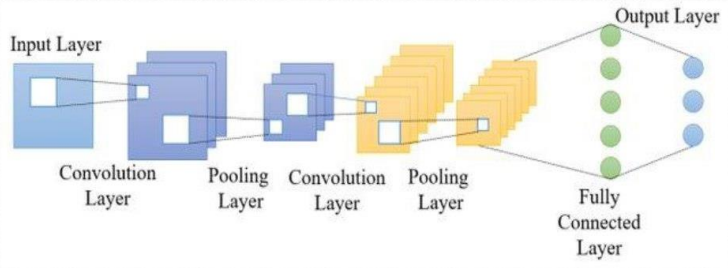
Довга короткострокова пам'ять (LSTM) – це тип архітектури рекурентної нейронної мережі, призначеної для фіксації довготривалих залежностей у послідовних даних.



6

Згорткова нейронна мережа

Згорткова нейронна мережа CNN – це глибока ШНМ, що використовується для роботи з часовими рядами. CNN складається з згорткових шарів, шару пулінгу та повнозв'язного шару.

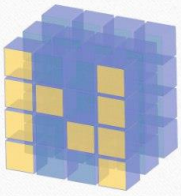


7

Використані технології

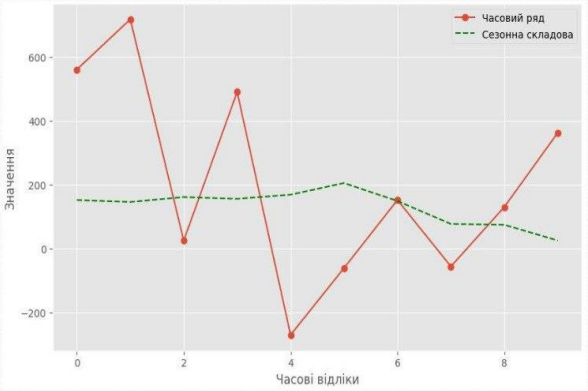
- обчислювальна платформа Jupyter Notebook;
- мова програмування Python;
- фреймворк машинного навчання PyTorch;
- бібліотеки обробки та аналізу даних: Pandas та Numpy;
- бібліотеки візуалізації: Seaborn та Matplotlib.





8

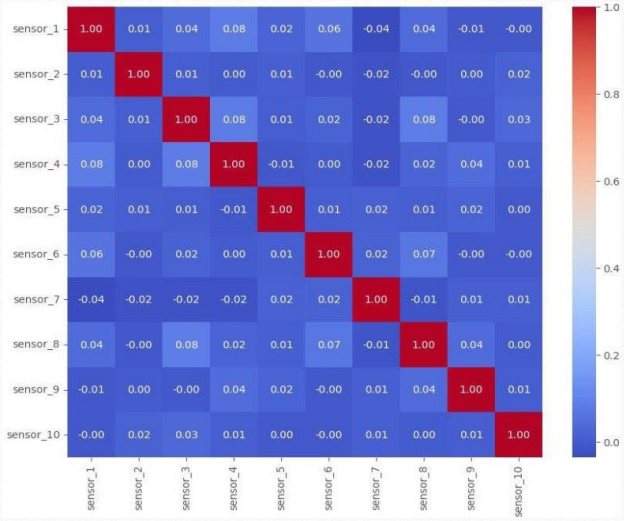
Аналіз часового ряду




```
Results of Dickey-Fuller Test:
Test Statistic: -2.5240316238004157
p-value: 0.10973611757469942
```

9

Теплова карта кореляційної матриці даних



10

Програмне рішення

```
df_train, df_valid = train_test_split(df_train, test_size=0.2, random_state=42)
```

```
df_train[:60000]
```

	segment_id	time_to_eruption
1345	609963923	29684540
3130	770119099	40625627
4062	437665247	44259173
52	1202935850	4208163
402	784608403	24330676
...	...	...
4426	873340274	15695097
466	1099864364	26858020
3092	571683257	34317444
3772	1344839103	40144143
860	64719838	28453756

3544 rows × 2 columns

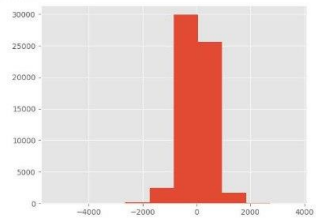
```
class INGVDataset(torch.utils.data.Dataset):
    def __init__(self, df):
        self.segment = df.segment_id.values
        self.time = df.time_to_eruption.values
    def __len__(self):
        return self.segment.shape[0]
    def __getitem__(self, index):
        df = pd.read_csv(f"input/predict-volcanic-eruptions-ingv-oe/train/{self.segment[index]}.csv")

        df = df.fillna(0)
        df = df[:60000].values.reshape(6000, 100)
        label = self.time[index]
        return df , label
```

	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	sensor_7	sensor_8	sensor_9	sensor_10
count	60001.000000	60001.000000	60001.000000	60001.000000	60001.000000	60001.000000	60001.000000	60001.000000	59533.000000	60001.000000
mean	-2.586141	-3.353361	-0.018533	-2.012000	1.721038	2.547641	-0.632766	-0.176447	1.026372	-116.737804
std	290.689446	519.035022	231.045818	264.541268	256.266425	314.754419	415.310810	339.036140	295.248576	1042.540568
min	-1234.000000	-5323.000000	-1279.000000	-1573.000000	-1837.000000	-1437.000000	-3553.000000	-3787.000000	-1933.000000	-5174.000000
25%	-165.000000	-318.000000	-148.000000	-170.000000	-151.000000	-211.000000	-239.000000	-216.000000	-182.000000	-723.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	162.000000	311.000000	148.000000	162.000000	156.000000	212.000000	238.000000	218.000000	187.000000	565.000000
max	1442.000000	3630.000000	1229.000000	1678.000000	2382.000000	1585.000000	4132.000000	3020.000000	2053.000000	4925.000000

```
df_first.sensor_2.hist()
```

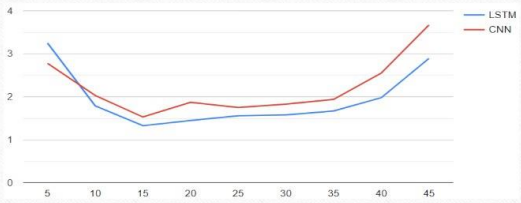
```
<Axes: >
```



11

Вплив параметру epochs на результати прогнозування

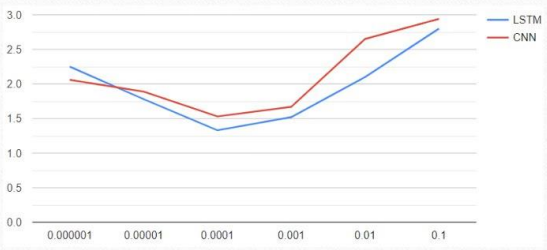
№	Параметри			MSE	
	epochs	batch	lr	LSTM	CNN
1	5	5	0.0001	3.25	2.78
2	10	5	0.0001	1.79	2.03
3	15	5	0.0001	1.33	1.53
4	20	5	0.0001	1.45	1.87
5	25	5	0.0001	1.56	1.75
6	30	5	0.0001	1.58	1.83
7	35	5	0.0001	1.67	1.94
8	40	5	0.0001	1.98	2.55
9	45	5	0.0001	2.89	3.67



12

Вплив параметру lr на результати прогнозування

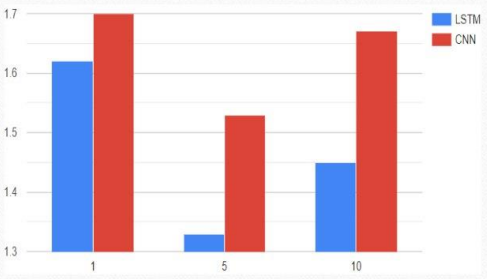
№	Параметри			MSE	
	epochs	batch	lr	LSTM	CNN
1	15	5	0.000001	2.25	2.06
2	15	5	0.00001	1.78	1.89
3	15	5	0.0001	1.33	1.53
4	15	5	0.001	1.52	1.67
5	15	5	0.01	2.10	2.65
6	15	5	0.1	2.80	2.94



13

Вплив параметру batch на результати прогнозування

№	Параметри			MSE	
	epochs	batch	lr	LSTM	CNN
1	15	1	0.0001	1.62	1.70
2	15	5	0.0001	1.33	1.53
3	15	10	0.0001	1.45	1.67



14

Вплив гіперпараметрів на результати прогнозування LSTM

№	hidden_size	dropout	MSE	Час навчання
1	100	0.2	1.14	9
1	200	0.2	1.13	8
1	300	0.2	1.25	15
1	400	0.2	1.26	15
2	100	0.3	1.12	8
2	200	0.3	1.12	9
2	300	0.3	1.15	14
2	400	0.3	1.18	11
3	100	0.4	1.36	14
3	200	0.4	1.51	14
3	300	0.4	1.23	18
3	400	0.4	1.56	19

15

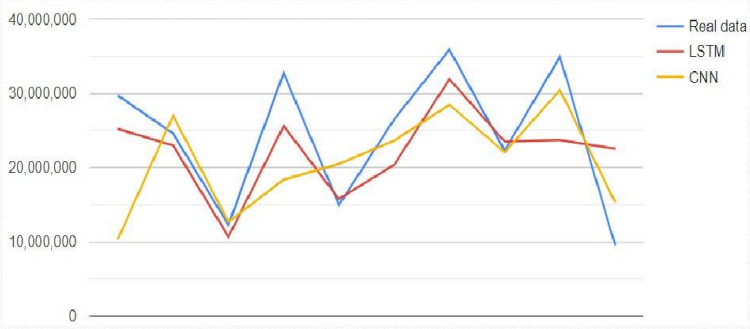
Вплив гіперпараметрів на результати прогнозування CNN

№	kernel_size	dropout	MSE	Час навчання
1	2	0.2	1.73	11
1	3	0.2	1.68	12
1	4	0.2	1.65	12
1	5	0.2	1.67	14
2	2	0.3	1.50	11
2	3	0.3	1.62	11
2	4	0.3	1.58	13
2	5	0.3	1.60	11
3	2	0.4	1.82	14
3	3	0.4	1.93	14
3	4	0.4	1.90	18
3	5	0.4	1.92	19

16

Порівняння реальних даних та прогнозованих

Середня абсолютна похибка (MAE) для LSTM при обчислені дорівнює 1502387.5 та середня абсолютна похибка у відсотках (MAPE) дорівнює 6.9815%.
MAE для CNN дорівнює 2182185.4, а MAPE дорівнює 8.919%.



17

Висновок

У роботі досліджена і проаналізована проблема прогнозування сейсмічної активності вулканів з використанням моделей глибокого навчання. Дослідження включає аналіз даних про сейсмічну активність вулканів, створення та навчання моделей глибокого навчання на цих даних, а також аналіз отриманих результатів.

Результати вказують на перевагу моделі LSTM. В порівнянні з CNN, LSTM забезпечує кращу точність у відтворенні реальних даних, проте обидва підходи демонструють здатність відображати загальні тенденції в даних.

В процесі роботи опублікована стаття на тему «Моделі глибокого навчання для прогнозування часових рядів» у журналі «Системи управління навігації та зв'язку», №1(75).

18

ДОДАТОК Б

Додаткові матеріали дослідження

Таблиця 1 – Порівняльна характеристика deep learning та традиційного машинного навчання

Характеристика	Deep Learning	Машинне навчання
Архітектура моделей	Глибокі нейронні мережі з безліччю шарів, здатних витягувати складні ознаки.	Поверхневі моделі з невеликою кількістю шарів, найчастіше лінійні або деревоподібні.
Використання даних	Потребує великих обсягів даних для ефективного навчання.	Може працювати з меншими обсягами даних, часто більш ефективно в завданнях з обмеженими даними.
Робота з ознаками	Автоматичне вилучення ознак у процесі навчання.	Вимагає ручної інженерії ознак для визначення та вибору найкращих характеристик.
Навчання	Ієрархічне, з уточненням уявлень на різних рівнях.	Зазвичай більш плоске, не має ієрархії в поданні даних.
Обробка даних	Ефективно обробляє неструктуровані дані.	Більш орієнтований на структуровані дані.
Продуктивність	Вимагає потужних обчислювальних ресурсів, особливо GPU.	Може працювати на менш потужних пристроях, що робить його доступним.
Інтерпретованість	Моделі можуть бути складними і важкозрозумілими.	Простіші моделі, легше інтерпретувати і пояснити.

Таблиця 2 – Обчислення метрик для LSTM

№	Реальне значення	Спрогнозоване значення	AE	APE
1	29684540	25175888	4508652	15.19%
2	24569825	22971505	1598320	6.505%
3	12262005	12643378	381373	3.11%
4	32739612	32578563	161049	0.492%
5	14965999	15765654	799655	5.34%
6	26469720	23357436	3112284	11.76%
7	35887276	34906658	980618	2.73%
8	22264886	23482570	1217684	5.46%
9	34952168	33704850	1247318	3.57%
10	9504818	10521740	1016922	10.7%

Таблиця 3 – Обчислення метрик для CNN

№	Реальне значення	Спрогнозоване значення	AE	APE
1	29684540	28194180	1490360	5.01%
2	24569825	26971505	2401680	9.77%
3	12262005	12251564	10441	0.0852%
4	32739612	31367327	1372285	4.19%
5	14965999	20474824	5508825	36.81%
6	26469720	23618695	2851025	10.78%
7	35887276	28456959	7430317	20.7%
8	22264886	22054297	210589	0.94%
9	34952168	34433471	518697	1.48%
10	9504818	9532453	27635	0.29%

ДОДАТОК В

Вихідний код розроблених програмних засобів

Лістинг 1 – Загальний лістинг дослідження

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pytorch_lightning as pl
from sklearn.model_selection import train_test_split
import seaborn as sns
import torch
import torch.nn as nn
from sklearn import preprocessing
from scipy.stats import norm, skew #for some statistics
from scipy import stats
import math
%matplotlib inline
plt.style.use('ggplot')
df_train = pd.read_csv("input/predict-volcanic-eruptions-ingv-
oe/train.csv")
df_train.head(10)
df_train.describe()
Out[3]:
df_train.isnull().sum()
df_train.time_to_eruption.hist()
Out[5]:
df_train.time_to_eruption.max()
Out[6]:
sns.distplot(df_train.time_to_eruption , fit=norm);
# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(df_train.time_to_eruption)
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f}
)'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('Time distribution')
#Get also the QQ-plot
fig = plt.figure()
res = stats.probplot(df_train.time_to_eruption, plot=plt)
plt.show()
df_first = pd.read_csv(f"../input/predict-volcanic-eruptions-
ingv-oe/train/{df_train.segment_id[2]}.csv")
df_first.head()
df_first.describe()
```

```
Out[9]:
df_first.sensor_2.hist()
Out[10]:
%matplotlib inline
# calculate the correlation matrix
corr = df_first.corr()
# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
df_train, df_valid = train_test_split(df_train, test_size=0.2,
random_state=42)
In [14]:
df_train[:60000]
Out[14]:
class INDVDataset(torch.utils.data.Dataset):
    def __init__(self, df):
        self.segment = df.segment_id.values
        self.time = df.time_to_eruption.values
    def __len__(self):
        return self.segment.shape[0]
    def __getitem__(self, index):
        df = pd.read_csv(f"../input/predict-volcanic-eruptions-
ingv-oe/train/{self.segment[index]}.csv")

        df = df.fillna(0)
        df = df[:60000].values.reshape(6000, 100)
        label = self.time[index]
        return df , label
BATCH_SIZE  = 5
NUM_WORKERS = 2
LR =1e-4
EPOCHS = 5
DEVICE= 'cuda:0'  if torch.cuda.is_available() else 'cpu'
from torch.nn import functional as F
class INGVNet(pl.LightningModule):
    def __init__(self):
        super().__init__()
        self.lstm1 = nn.LSTM(100 , 200 , bidirectional=False,
batch_first=True)
        self.linear1 = nn.Linear(400, 400)
        self.linear_aux_out = nn.Linear(400, 1)
        self.critrion = nn.MSELoss(reduction='mean')
        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        lstm1, _ = self.lstm1(x)
        avg_pool = torch.mean(lstm1, 1)
        max_pool, _ = torch.max(lstm1, 1)

        h_conc = torch.cat((max_pool, avg_pool), 1)
        h_conc_linear1  = self.linear1(h_conc)
        hidden = 1000*h_conc + h_conc_linear1
```

```
        hidden = self.dropout(hidden)
        hidden = nn.LeakyReLU()(hidden)
        aux_result = 10*self.linear_aux_out(hidden)
        return aux_result
    def training_step(self, batch, batch_idx):
        # training_step defined the train loop. It is
independent of forward
        x, y = [i.float().to(DEVICE) for i in batch]
        x_pred = self(x)
        loss = torch.sqrt(self.critrion(x_pred, y.reshape(-1,
1)))
        with torch.no_grad():
            logs = {
                'loss': loss,

            }
        return {'loss': loss, 'log': logs, "progress_bar":
{"MAE": nn.L1Loss()(x_pred, y.reshape(-1, 1)) }}
    @torch.no_grad()
    def validation_step(self, batch, batch_idx):
        x, y = [i.float().to(DEVICE) for i in batch]
        x_pred = self(x)
        loss = self.critrion(x_pred, y.reshape(-1,1))
        logs = {
                'val_loss': loss
            }
        return logs
    def train_dataloader(self):
        train_dataset = INDVDataset(df_train)
        train_loader =
torch.utils.data.DataLoader(train_dataset,
batch_size=BATCH_SIZE,

pin_memory=True, num_workers = NUM_WORKERS, shuffle=True)
        return train_loader
    def val_dataloader(self):
        valid_dataset = INDVDataset(df_valid)
        valid_dataloader =
torch.utils.data.DataLoader(valid_dataset,
batch_size=BATCH_SIZE,
pin_memory=True, num_workers = NUM_WORKERS, shuffle=False)
        return valid_dataloader
    def configure_optimizers(self):
        self.optimizer = torch.optim.Adam(self.parameters(),
lr=LR, betas= (0.9,0.999), weight_decay= 5e-7, amsgrad=True) #,
betas= (0.9,0.999), weight_decay= 5e-7, amsgrad=True
        self.scheduler =
torch.optim.lr_scheduler.StepLR(self.optimizer, step_size=3,
gamma=0.6)
        return [self.optimizer], [self.scheduler]
In [18]:
net = INGVNet()
In [19]:
```

```
linkcode
torch.backends.cudnn.benchmark =  True
trainer = pl.Trainer(max_epochs=EPOCHS ,gradient_clip_val=0,
limit_val_batches=0.2, gpus=1)
trainer.fit(net)
class INGVNet(pl.LightningModule):
    def init(self):
        super().init()
        self.conv1 = nn.Conv1d(in_channels=6000,
out_channels=200, kernel_size=3, padding=1)
        self.conv2 = nn.Conv1d(in_channels=200,
out_channels=200, kernel_size=3, padding=1)
        self.conv3 = nn.Conv1d(in_channels=200,
out_channels=200, kernel_size=3, padding=1)
        self.pool = nn.MaxPool1d(kernel_size=2)
        self.fc1 = nn.Linear(200*25, 400) # Adjust input size
based on your data
        self.fc2 = nn.Linear(400, 1)
        self.criterion = nn.MSELoss(reduction='mean')
        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = self.pool(x)
        x = torch.relu(self.conv3(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1) # Flatten the output of conv
layers
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
    def training_step(self, batch, batch_idx):
        x, y = [i.float().to(DEVICE) for i in batch]
        x_pred = self(x)
        mse_loss = self.criterion(x_pred, y.reshape(-1, 1))
        rmse_loss = torch.sqrt(mse_loss)
        mae_loss = nn.L1Loss()(x_pred, y.reshape(-1, 1))
        self.log('train_rmse', rmse_loss, prog_bar=True,
logger=True)
        self.log('train_mae', mae_loss, prog_bar=True,
logger=True)
        return {'loss': rmse_loss}
    @torch.no_grad()
    def validation_step(self, batch, batch_idx):
        x, y = [i.float().to(DEVICE) for i in batch]
        x_pred = self(x)
        mse_loss = self.criterion(x_pred, y.reshape(-1, 1))
        rmse_loss = torch.sqrt(mse_loss)
        mae_loss = nn.L1Loss()(x_pred, y.reshape(-1, 1))
        self.log('val_rmse', rmse_loss, prog_bar=True,
logger=True)
```

```
        self.log('val_mae', mae_loss, prog_bar=True,
logger=True)
        return {'val_loss': rmse_loss}
    def train_dataloader(self):
        train_dataset = INDVDataset(df_train)
        train_loader =
torch.utils.data.DataLoader(train_dataset,
batch_size=BATCH_SIZE,
                                                pin_memory=Tr
ue, num_workers = NUM_WORKERS, shuffle=True)
        return train_loader
    def val_dataloader(self):
        valid_dataset = INDVDataset(df_valid)
        valid_dataloader =
torch.utils.data.DataLoader(valid_dataset,
batch_size=BATCH_SIZE,
                                                pin_memory=Tr
ue, num_workers = NUM_WORKERS, shuffle=False)
        return valid_dataloader
    def configure_optimizers(self):
        self.optimizer = torch.optim.Adam(self.parameters(),
lr=LR, betas=(0.9,0.999), weight_decay=5e-7, amsgrad=True)
        self.scheduler =
torch.optim.lr_scheduler.StepLR(self.optimizer, step_size=3,
gamma=0.6)
        return [self.optimizer], [self.scheduler]
from tqdm import tqdm
test_dataset = INDVTest(df_test)
test_loader = torch.utils.data.DataLoader(test_dataset,
batch_size=BATCH_SIZE,
pin_memory=True, num_workers = NUM_WORKERS, shuffle=True)
arr_time = []
net.cuda()
for batch in tqdm(test_loader):
    arr_time= [*arr_time, *
(net(batch.float().to(DEVICE)).squeeze().detach().cpu().numpy())
]
df_test.time_to_eruption = arr_time
```