ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Методи прогнозування розвитку пандемій на основі штучних нейронних мереж з урахуванням зовнішніх факторів

Виконав:
магістрант групи СПм-20-1 Мотькін М. А.

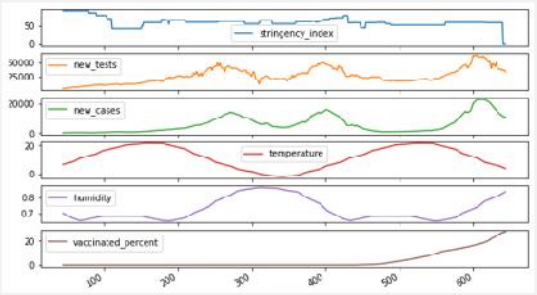
Науковий керівник:
доцент каф. ЕОМ Івашенко Г. С.

Харківський національний університет радіоелектроніки
Кафедра ЕОМ, тел. 702-13-54

Актуальність проблеми

	stringency_index	new_tests	new_cases	date	temperature	humidity	vaccinated_percent
44	88.89	3174.00	394.00	2020-04-16	6.46	0.70	0.00
45	88.89	3368.00	351.00	2020-04-17	6.50	0.70	0.00
46	88.89	3557.00	371.00	2020-04-18	6.73	0.70	0.00
47	88.89	3650.00	382.00	2020-04-19	6.86	0.69	0.00
48	88.89	3724.00	373.00	2020-04-20	7.00	0.69	0.00
...	...	...	...	...	...	...	...
639	57.41	37420.00	12078.00	2021-12-02	4.70	0.83	26.52
640	57.41	37547.00	11793.00	2021-12-03	4.48	0.83	26.94
641	0.00	36412.00	11539.00	2021-12-04	4.26	0.83	27.11
642	0.00	37250.00	11411.00	2021-12-05	4.04	0.83	27.21
643	0.00	36738.00	11204.00	2021-12-06	3.81	0.84	27.47

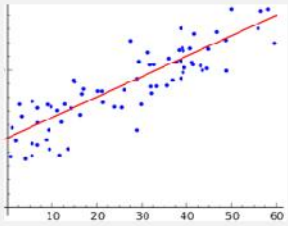
600 rows × 7 columns

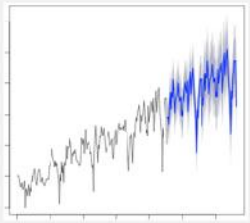
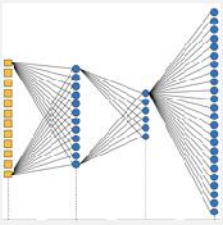


На сьогоднішній день прогнозування розвитку пандемій є вкрай актуальним завданням, від результатів вирішення якого можуть залежати людські життя та економічний стан країни.

Інформація про розвиток пандемій може бути представлена у вигляді багатопараметричних часових рядів, тому для вирішення подібного завдання доцільним є використання методів аналізу та прогнозування часових рядів.

Існуючі рішення



- методи згладжування: ковзне середнє та експоненційне згладжування
- регресійне прогнозування: лінійна та нелінійна регресія;
- авторегресійні моделі: ARMA, ARIMA, SARIMA;
- штучні нейронні мережі.

3

Постановка задачі

У роботі досліджуються методи прогнозування розвитку пандемій з допомогою штучних нейронних мереж з урахуванням зовнішніх чинників.

Для аналізу медичних зображень використовуються штучні нейронні мережі з наступними архітектурами:

- CNN та TCN;
- RNN та LSTM;

Досліджується залежність метрики ефективності MeanSquaredError та швидкості навчання мереж від наступних гіперпараметрів:

- розмір навчальних блоків;
- значення dropout;
- Units (для LSTM та RNN) та filters (для CNN та TCN);
- розмір згортки та страйд (для CNN та TCN).

Для покращення результатів прогнозування використовується попередня обробка, зокрема:

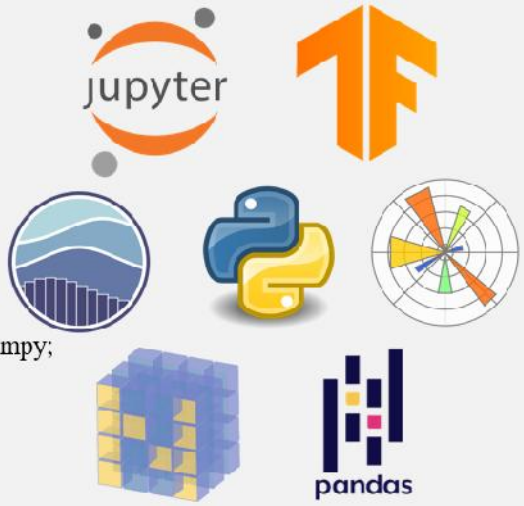
- усунення пропущених значень;
- згладжування викидів та сезонних коливань;
- нормалізація.

4

Використані технології

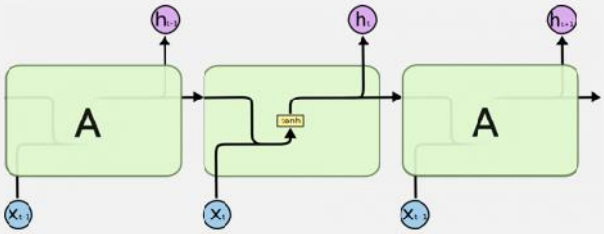
У ході дослідження використані наступні технології:

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



5

Рекурентні нейронні мережі (RNN)

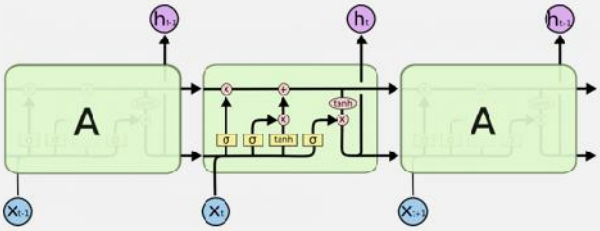


Рекурентні нейронні мережі – тим нейронних мереж, який заснований на застосуванні результатів попередніх обчислень. Тобто це тип нейронних мереж, які мають «пам'ять» і можуть її застосовувати для нових обчислень.

Рекурентні нейронні мережі можуть бути представлені у вигляді послідовності осередків, кожен з яких передає дані не тільки на вихід пару, а й на вхід наступного осередку.

6

Нейронні мережі з довгою короткостроковою пам'яттю (LSTM)

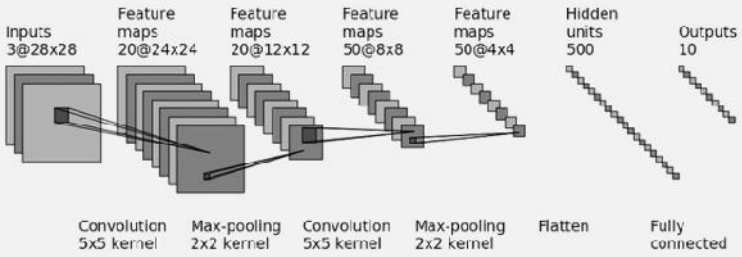


Архітектура LSTM є модифікацією RNN мережі, в якій структура осередків пам'яті ускладнена, щоб збільшити запам'ятовуючу здатність мережі та уповільнити забування вже вивченої інформації.

Сьогодні LSTM (Long short-term memory) є однією з найкращих архітектур для прогнозування часових рядів.

7

Згорткові нейронні мережі (CNN)



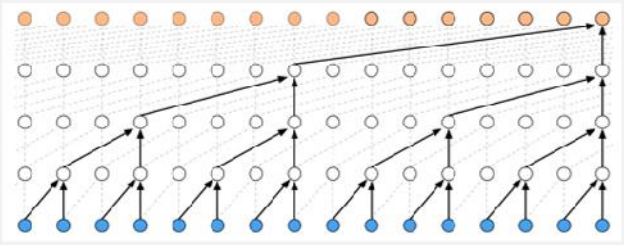
Згорткова нейронна мережа – спеціальна архітектура штучних нейронних мереж, націлена на ефективне розпізнавання образів.

В основі CNN лежить операція згортки, яка обчислює значення на виході шару, на основі групи значень попереднього шару.

Даний тип нейронних мереж найкраще показує себе при роботі з зображеннями.

8

Часові згорткові мережі (TCN)



Часові згорткові нейронні мережі – це модифікація згорткової нейронної мережі, що призначена для роботи з часовими рядами.

У основі TCN лежить операція причинно-наслідкової згортки, тобто вихідне значення шару залежить тільки від значень, які розташовані раніше у вхідній послідовності.

Зазвичай ця модель нейронних мереж демонструє результати, що перевершують результати мереж LSTM.

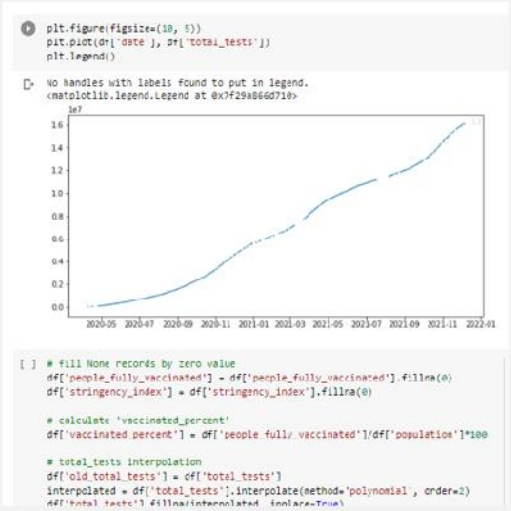
9

Програмне рішення для виконання дослідження

Програмне рішення представлене у вигляді файлу з розширенням ipynb, який використовується обчислювальним середовищем Jupyter Notebook.

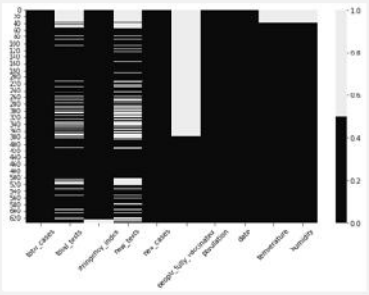
При відкритті файлу в середовищі Jupyter Notebook, програмне рішення виглядає як послідовність обчислювальних осередків з кодом, кожен з яких може бути обчислений незалежно від інших.

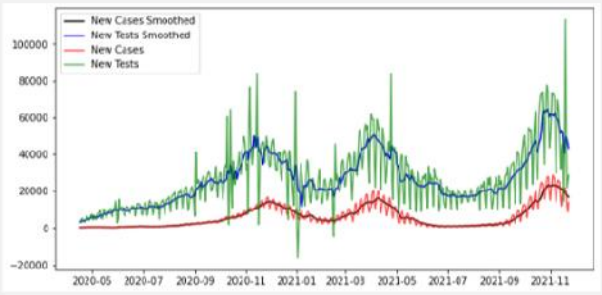
Між осередками розташовані проміжні результати у вигляді графіків та звітів.



10

Попередня обробка даних





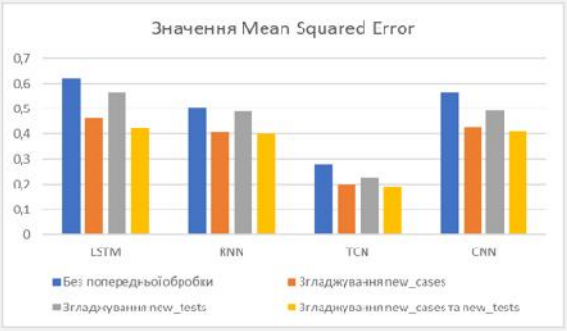
Попередня обробка складалася з наступних етапів:

- аналіз та усунення пропусків даних;
- згладжування тижневих коливань та викидів;
- нормалізація даних.

11

Результати досліджень. Залежність значення MSE від попередньої обробки

Варіант обробки	Значення Mean Squared Error			
	LSTM	RNN	TCN	CNN
Без попередньої обробки	0,6205	0,5052	0,2805	0,5623
Згладжування new_cases	0,4623	0,4052	0,2001	0,4281
Згладжування new_tests	0,5648	0,4896	0,2249	0,4913
Згладжування new_cases та new_tests	0,4227	0,4006	0,1922	0,4089



12

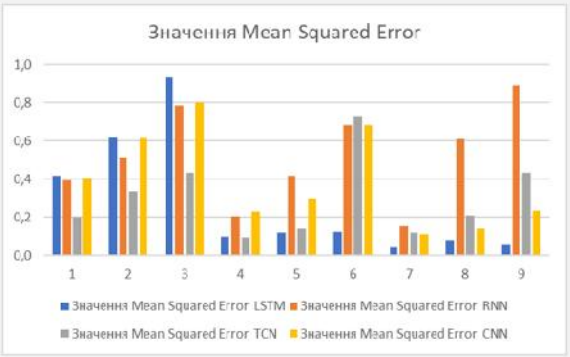
Результати досліджень. Залежність значення MSE від параметрів прогнозування та налаштувань навчання

№	Параметри			Значення Mean Squared Error			
	batch	history	prediction horizon	LSTM	RNN	TCN	CNN
1	10	60	20	0,4154	0,3908	0,2011	0,4002
2	5	60	20	0,6203	0,5103	0,3341	0,6134
3	1	60	20	0,9313	0,7842	0,4345	0,8011
4	10	30	10	0,0983	0,2057	0,0921	0,2291
5	5	30	10	0,1192	0,4179	0,1390	0,2985
6	1	30	10	0,1241	0,6813	0,7313	0,6954
7	10	15	5	0,0423	0,1512	0,1174	0,1103
8	5	15	5	0,0753	0,6073	0,2068	0,1385
9	1	15	5	0,0531	0,8922	0,4315	0,2315

13

Результати досліджень. Залежність значення MSE від параметрів прогнозування та налаштувань навчання

№	Параметри		
	batch	history	prediction horizon
1	10	60	20
2	5	60	20
3	1	60	20
4	10	30	10
5	5	30	10
6	1	30	10
7	10	15	5
8	5	15	5
9	1	15	5



14

Результати досліджень. Використання мереж LSTM та RNN

Мережа	Units	Dropout	Activation	MSE	Час навчання
LSTM	16	0,4	tanh	0,405	18
LSTM	32	0,4	tanh	0,4397	18
LSTM	64	0,8	tanh	0,4037	23
LSTM	128	0,4	tanh	0,5023	35
LSTM	64, 32	0,4; 0,4	tanh	0,5131	36
LSTM	8	0,4	tanh	0,3176	18
LSTM	4	0,4	tanh	0,4113	18
LSTM	2	0,4	tanh	0,6467	18
RNN	64	0,4	tanh	0,2293	8
RNN	64	0,4	relu	0,1453	9
RNN	16	0,4	tanh	0,4354	8
RNN	4	0,4	tanh	0,5323	8
RNN	128	0,4	tanh	0,1103	12
RNN	128	0,8	tanh	0,5831	12

15

Результати досліджень. Використання мереж TCN та CNN

Мережа	KernelSize	Stride	Dilations	MSE	Час навчання
CNN	2	1	-	0,4054	8
CNN	2	1	-	0,5635	6
CNN	2	1	-	0,7952	3
CNN	2	1	-	0,3702	12
CNN	5	1	-	0,9928	4
CNN	2	2	-	0,4623	4
CNN	2	3	-	0,4871	4
TCN	3	1	1,2,4,8,16,32	0,2056	26
TCN	3	1	1,2,4,8,16,32	0,5641	24
TCN	3	1	1,2,4,8,16,32	0,1783	67
TCN	5	1	1,2,4,8,16,32	0,1825	33
TCN	2	1	1,2,4,8,16,32	0,2923	24
TCN	3	1	1,2,4,8	0,3564	16
TCN	3	1	1,4,16,64	0,109	30

15

Висновки

У роботі досліджені і проаналізовані методи прогнозування розвитку пандемії за допомогою штучних нейронних мереж з урахуванням зовнішніх факторів. В якості вихідних даних використовувалися дані про захворюваність COVID-19, зібрані Всесвітньою Організацією Охорони Здоров'я (ВООЗ). Створено та проаналізовано 60 моделей штучних нейронних мереж на основі 4 архітектур з різними гіперпараметрами.

Було поставлене та виконане завдання автоматизації прогнозування кількості хворих з використанням засобів штучного інтелекту, а саме, мереж:

- LSTM та RNN;
- CNN та TCN.

Часткові результати дослідження були наведені у рамках Дев'ятої міжнародної науково-технічної конференції "Проблеми інформатизації".

Основні напрямки розвитку рішення:

- дослідження можливості додаткової оптимізації параметрів використаних мереж;
- дослідження переваг і недоліків застосування глибокого навчання для прогнозування часових рядів.

17

ДОДАТОК Б

Вихідний код розроблених програмних засобів

Лістинг 1 – Загальний лістинг дослідження

```
import os
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
from statsmodels.tsa.api import SimpleExpSmoothing

# Configure base plot params
matplotlib.rcParams['figure.figsize'] = (10, 5)
matplotlib.rcParams['axes.grid'] = False

# Download latest dataset
full_dataset =
pd.read_csv('https://covid.ourworldindata.org/data/owid-covid-
data.csv')

# Filter data by location
df = full_dataset.loc[full_dataset['location'] == 'Ukraine']

# Convert date from string to DateTime
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')

# Add base temperature & humidity info
wdf = pd.DataFrame([[1,-2.7,0.86],  [2,-0.5,0.83],
                    [3,4.3,0.74],   [4,9.7,0.64],
                    [5,15.2,0.68],  [6,19.9,0.68],
                    [7,21.6,0.68],  [8,21.2,0.64],
                    [9,16,0.68],    [10,9.2,0.76],
                    [11,3.8,0.84],  [12,0.3,0.87]],
                   columns=['month','temperature','humidity'])

# merge with main dataset
df = pd.merge(df.assign(grouper=df['date'].dt.month),
wdf.assign(grouper=wdf['month']), how='left', on='grouper')
df = df.drop(['grouper', 'month'], axis=1)

# smooth temperature & humidity info
WINDOW_SIZE = 40
df['temperature'] =
```

```
df['temperature'].rolling(WINDOW_SIZE).mean()
df['humidity'] = df['humidity'].rolling(WINDOW_SIZE).mean()

# select target columns
df = df[['total_cases', 'total_tests', 'stringency_index',
         'new_tests', 'new_cases',
         'people_fully_vaccinated', 'population',
         'date', 'temperature','humidity']]

# NEXT CELL
for col in df.columns:
    null_records = df[col].isnull()
    pct_missing = np.mean(null_records)
    rounded_percent = round(pct_missing*100)
    print('{} - {}%'.format(col, rounded_percent))

# NEXT CELL
matplotlib.rcParams['figure.figsize'] = (10, 6)

# show heatmap for empty values
sns.heatmap(df[df.columns].isnull(),
cmap=sns.color_palette(['#000099', '#ffff00']))
plt.xticks(rotation=45)
plt.show()

# NEXT CELL
plt.figure(figsize=(10, 5))
plt.plot(df['date'], df['total_tests'])
plt.legend()

# NEXT CELL
# fill None records by zero value
df['people_fully_vaccinated'] =
df['people_fully_vaccinated'].fillna(0)
df['stringency_index'] = df['stringency_index'].fillna(0)

# calculate 'vaccinated_percent'
df['vaccinated_percent'] =
df['people_fully_vaccinated']/df['population']*100

# total_tests interpolation
df['old_total_tests'] = df['total_tests']
interpolated =
df['total_tests'].interpolate(method='polynomial', order=2)
df['total_tests'].fillna(interpolated, inplace=True)

# show plot for 'total_tests' && 'old_total_tests'
plt.figure(figsize=(10, 5))
plt.plot(df['date'], df['total_tests'], color='red')
plt.plot(df['date'], df['old_total_tests'])
plt.legend()

# drop useless column
```

```
df = df.drop(['old_total_tests'], axis=1)

# NEXT CELL
# recalculate 'new_tests'
df['new_tests'] = df['total_tests'].diff()
interpolated = df['new_tests'].interpolate(method='linear')
df['new_tests'].fillna(interpolated, inplace=True)

# smooth 'new_cases' & 'new_tests'
df['new_cases_smoothed'] = df['new_cases'].rolling(7).mean()
df['new_tests_smoothed'] = df['new_tests'].rolling(7).mean()

# show plot for 'new_cases_smoothed' & 'new_tests_smoothed'
plt.figure(figsize=(10, 5))
plt.plot(df['date'], df['new_cases_smoothed'], label='New Cases
Smoothed', color='black')
plt.plot(df['date'], df['new_tests_smoothed'], label='New Tests
Smoothed', color='blue')

# show plot for 'new_cases' & 'new_tests'
plt.plot(df['date'], df['new_cases'], label='New Cases',
color='red', alpha=0.7)
plt.plot(df['date'], df['new_tests'], label='New Tests',
color='green', alpha=0.7)
plt.legend()

# NEXT CELL
# move column values
df['new_cases'] = df['new_cases_smoothed'].round(0)
df['new_tests'] = df['new_tests_smoothed'].round(0)
df = df.drop(['new_cases_smoothed', 'new_tests_smoothed',
'total_cases', 'total_tests','people_fully_vaccinated',
'population' ], axis=1)

# drop rows with None values
df = df.dropna()

# plot option for float values
pd.set_option('display.float_format', lambda x: '%.2f' % x)

# show data statistics
df.describe()

# NEXT CELL
# show data correlation info
df.corr()

# NEXT CELL
# show correlation heatmap
sns.heatmap(df.corr(), annot = True, vmin=-1, vmax=1, center= 0,
cmap= 'coolwarm', linewidths=3, linecolor='black')

#NEXT CELL
```

```
import tensorflow as tf

from keras.callbacks import EarlyStopping, Callback
from keras.models import Sequential
from keras.layers import LSTM, Dense, Flatten, ConvLSTM2D,
Dropout, Bidirectional
from keras.layers import RNN, Input, Conv1D, MaxPooling1D,
SimpleRNN
from tensorflow.keras.optimizers import SGD

# try import external lib for TCN network
try:
  from tcn import TCN
except:
  !pip install keras-tcn --no-dependencies
  from tcn import TCN

# NEXT CELL
def split_frame(frame, history_size, horizon_size,
result_labels):
  input = []
  output = []

  for i in range(history_size, len(frame)-horizon_size+1):
    temp = frame[i-history_size:i]
    data = temp
    input.append(data)

    temp = frame[i:i+horizon_size]
    data = temp[result_labels]
    output.append(data.to_numpy())

  return np.array(input), np.array(output)

# convert date to index
df.index = df['date']

# drop date column
df = df.drop(['date'], axis=1)

# split dataframe to train & validation frames
TRAIN_SPLIT = int(len(df)*0.6)
train_frame = df[TRAIN_SPLIT:]
validation_frame = df[:TRAIN_SPLIT]

# normalize columns
df['stringency_index'] = df['stringency_index']/100
df['vaccinated_percent'] = df['vaccinated_percent']/100
df['temperature'] = (df['temperature']+36)/72
nc_mean, nc_std = train_frame['new_cases'].mean(),
train_frame['new_cases'].std()
df['new_cases'] = (df['new_cases']-nc_mean)/nc_std
nt_mean, nt_std = train_frame['new_tests'].mean(),
```

```
train_frame['new_tests'].std()
df['new_tests'] = (df['new_tests']-nt_mean)/nt_std

# NEXT CELL
history_size = 60
prediction_horizon = 20
target_labels = ['new_cases']
features_size = len(df.columns)

# NEXT CELL
class NNData(object):
  def __init__(self, train_frame, validation_frame):
    self.train_frame = train_frame
    self.validation_frame = validation_frame
  def prepare_slices(self, history_size, prediction_horizon,
target_labels):
    x_train_data, y_train_data = split_frame(
        train_frame, history_size,
        prediction_horizon, target_labels
    )
    x_val_data, y_val_data = split_frame(
        validation_frame, history_size,
        prediction_horizon, target_labels
    )
    self.train_tuple = (x_train_data, y_train_data)
    self.validation_tuple = (x_val_data, y_val_data)

# NEXT CELL
class NNTestBench(object):
  name = None
  def __init__(self):
    self.fitlog = None
    self.model = None

  def configure_model(self, params):
    pass

  def set_optimizer(self, optimizer, loss='mse'):
    self.model.compile(optimizer=optimizer, loss=loss)

  def fit(self, epochs, batch_size, nndata):
    self.fitlog = self.model.fit(
        nndata.train_tuple[0],
        nndata.train_tuple[1],
        epochs=epochs,
        batch_size=batch_size,
        validation_data=nndata.validation_tuple)

class LSTMBench(NNTestBench):
  name="LSTM"

  def configure_model(self, params, custom_layers = None):
    self.model = Sequential()
```

```
    # for nonstandard models
    if custom_layers is not None:
      for layer in custom_layers:
        self.model.add(layer)
        return

    self.model.add(LSTM(
        units=params['units'],
        input_shape=params['shape'],
        activation=params['activation']))
    self.model.add(Dropout(rate=params['dropout']))
    self.model.add(Dense(params['prediction_horizon']))

class RNNBench(NNTestBench):
  name="RNN"

  def configure_model(self, params, custom_layers = None):
    self.model = Sequential()

    # for nonstandard models
    if custom_layers is not None:
      for layer in custom_layers:
        self.model.add(layer)
        return

    self.model.add(SimpleRNN(units=params['units'],
                             input_shape=params['shape'],
                             activation=params['activation']))
    self.model.add(Dropout(rate=params['dropout']))
    self.model.add(Dense(params['prediction_horizon']))

class TCNBench(NNTestBench):
  name="TCN"

  def configure_model(self, params, custom_layers = None):
    self.model = Sequential()

    # for nonstandard models
    if custom_layers is not None:
      for layer in custom_layers:
        self.model.add(layer)
        return

    self.model.add(TCN(
        nb_filters=params['filters'],
        kernel_size=params['kernel_size'],
        dilations=params['dilations'],
        dropout_rate=params['dropout'],
        input_shape=params['shape'],
        activation=params['activation']))
    self.model.add(Dense(params['prediction_horizon']))
```

```
class CNNBench(NNTestBench):
  name="CNN"

  def configure_model(self, params, custom_layers = None):
    self.model = Sequential()

    # for nonstandard models
    if custom_layers is not None:
      for layer in custom_layers:
        self.model.add(layer)
        return

    self.model.add(Conv1D(filters=params['filters'],
                          kernel_size=params['kernel_size'],
                          strides=params['stride'],
                          activation=params['activation'],
                          input_shape=params['shape']))
    self.model.add(MaxPooling1D(pool_size=params['pool_size']))
    self.model.add(Flatten())
    self.model.add(Dense(params['prediction_horizon']))

# NEXT CELL
# init data container
nndata = NNData(train_frame, validation_frame)
nndata.prepare_slices(history_size, prediction_horizon,
target_labels)

# load SGD optimizer
optimizer = SGD(learning_rate=0.1, clipnorm=1.0, momentum=0.9)

# NEXT CELL
lstmbench = LSTMBench()
lstmbench.configure_model({
    'units': 16,
    'shape': nndata.train_tuple[0].shape[-2:],
    'activation': 'tanh',
    'dropout': 0.4,
    'prediction_horizon': prediction_horizon
})
lstmbench.set_optimizer(optimizer)
lstmbench.fit(epochs=25, batch_size=10, nndata=nndata)

# NEXT CELL
rnnbench = RNNBench()
rnnbench.configure_model({
    'units': 128,
    'shape': nndata.train_tuple[0].shape[-2:],
    'activation': 'tanh',
    'dropout': 0.4,
    'prediction_horizon': prediction_horizon
})
rnnbench.set_optimizer(optimizer)
rnnbench.fit(epochs=25, batch_size=10, nndata=nndata)
```

```
# NEXT CELL
tcnbench = TCNBench()
tcnbench.configure_model({
    'filters': 64,
    'kernel_size': 3,
    'dilations': (1,4,16,64),
    'shape': nndata.train_tuple[0].shape[-2:],
    'activation': 'relu',
    'dropout': 0.3,
    'prediction_horizon': prediction_horizon
})
tcnbench.set_optimizer(optimizer)
tcnbench.fit(epochs=25, batch_size=10, nndata=nndata)

# NEXT CELL
cnnbench = CNNBench()
cnnbench.configure_model({
    'filters': 256,
    'kernel_size': 2,
    'stride': 1,
    'shape': nndata.train_tuple[0].shape[-2:],
    'activation': 'relu',
    'dropout': 0.3,
    'prediction_horizon': prediction_horizon,
    'pool_size': 5
})
cnnbench.set_optimizer(optimizer)
cnnbench.fit(epochs=25, batch_size=10, nndata=nndata)

# NEXT CELL
dates = df[-(history_size+prediction_horizon):].index
real_data = df[-
(history_size+prediction_horizon):]['new_cases']*nc_std +
nc_mean
history_data = df[-(history_size+prediction_horizon):-
prediction_horizon]
dates_for_predicted_values = df[-
prediction_horizon:].index.to_numpy()

plt.figure(figsize=(10, 5))
plt.plot(dates, real_data, label='Real Data')

reshaped_history_data =
history_data.to_numpy().reshape((1,history_size,features_size))
for bench in [lstmbench, rnnbench, tcnbench, cnnbench]:
  predicted = bench.model.predict(reshaped_history_data)*nc_std
+ nc_mean
  plt.plot(dates_for_predicted_values, predicted[0],
label=f'{bench.name} Prediction')
plt.legend()

# NEXT CELL
```

```
frame_slice = validation_frame[:-40]

dates = frame_slice[-(history_size+prediction_horizon):].index
real_data = frame_slice[-
(history_size+prediction_horizon):]['new_cases']*nc_std +
nc_mean
history_data = frame_slice[-(history_size+prediction_horizon):-
prediction_horizon]
dates_for_predicted_values = frame_slice[-
prediction_horizon:].index.to_numpy()

plt.figure(figsize=(10, 5))
plt.plot(dates, real_data, label='Real Data')

reshaped_history_data =
history_data.to_numpy().reshape((1,history_size,features_size))
for bench in [lstmbench, rnnbench, tcnbench, cnnbench]:
  predicted = bench.model.predict(reshaped_history_data)*nc_std
+ nc_mean
  plt.plot(dates_for_predicted_values, predicted[0],
label=f'{bench.name} Prediction')
plt.legend()

# NOTEBOOK END
```

ДОДАТОК В

Результати тестування

Таблиця 1 – Результати тестування впливу гіперпараметрів на нейронні мережі LSTM та RNN

Мережа	Filters/Units	Dropout	Activation	MSE	Час навчання
LSTM	16	0,4	tanh	0,405	18
LSTM	32	0,4	tanh	0,4397	18
LSTM	64	0,8	tanh	0,4037	23
LSTM	128	0,4	tanh	0,5023	35
LSTM	64, 32	0,4; 0,4	tanh	0,5131	36
LSTM	8	0,4	tanh	0,3176	18
LSTM	4	0,4	tanh	0,4113	18
LSTM	2	0,4	tanh	0,6467	18
RNN	64	0,4	tanh	0,2293	8
RNN	64	0,4	relu	0,1453	9
RNN	16	0,4	tanh	0,4354	8
RNN	4	0,4	tanh	0,5323	8
RNN	128	0,4	tanh	0,1103	12
RNN	128	0,8	tanh	0,5831	12

Таблиця 2 – Результати тестування впливу гіперпараметрів на нейронні мережі TCN та CNN

Мережа	Filters/Units	KernelSize	Stride	Dilations	MSE	Час навчання
CNN	64	2	1	-	0,4054	8
CNN	32	2	1	-	0,5635	6
CNN	16	2	1	-	0,7952	3
CNN	128	2	1	-	0,3702	12
CNN	64	5	1	-	0,9928	4
CNN	64	2	2	-	0,4623	4

Продовження таблиці 2 – Результати тестування впливу гіперпараметрів на нейронні мережі TCN та CNN

CNN	64	2	3	-	0,4871	4
TCN	64	3	1	1,2,4,8,16,32	0,2056	26
TCN	32	3	1	1,2,4,8,16,32	0,5641	24
TCN	128	3	1	1,2,4,8,16,32	0,1783	67
TCN	64	5	1	1,2,4,8,16,32	0,1825	33
TCN	64	2	1	1,2,4,8,16,32	0,2923	24
TCN	64	3	1	1,2,4,8	0,3564	16
TCN	64	3	1	1,4,16,64	0,109	30

Таблиця 3 – Вплив глобальних гіперпараметрів на нейронні мережі

Мережа	Batch	History	Horizon	MSE	Час навчання
LSTM	10	60	20	0,4154	15
RNN	10	60	20	0,3908	8
TCN	10	60	20	0,2011	24
CNN	10	60	20	0,4002	3
LSTM	5	60	20	0,6203	30
RNN	5	60	20	0,5103	15
TCN	5	60	20	0,3341	30
CNN	5	60	20	0,6134	5
LSTM	1	60	20	0,9313	75
RNN	1	60	20	0,7842	54
TCN	1	60	20	0,4345	53
CNN	1	60	20	0,8011	14
LSTM	10	30	10	0,0983	12
RNN	10	30	10	0,2057	7
TCN	10	30	10	0,0921	22

Продовження таблиці 3 – Вплив глобальних гіперпараметрів на нейронні мережі

CNN	10	30	10	0,2291	4
LSTM	5	30	10	0,1192	21
RNN	5	30	10	0,4179	21
TCN	5	30	10	0,139	44
CNN	5	30	10	0,2985	7
LSTM	1	30	10	0,1241	83
RNN	1	30	10	0,6813	41
TCN	1	30	10	0,7313	51
CNN	1	30	10	0,6954	20
LSTM	10	15	5	0,0423	12
RNN	10	15	5	0,1512	6
TCN	10	15	5	0,1174	19
CNN	10	15	5	0,1103	3
LSTM	5	15	5	0,0753	22
RNN	5	15	5	0,6073	11
TCN	5	15	5	0,2068	23
CNN	5	15	5	0,1385	10
LSTM	1	15	5	0,0531	42
RNN	1	15	5	0,8922	33
TCN	1	15	5	0,4315	54
CNN	1	15	5	0,2315	20