Prognozowanie krótkoterminowych opadów atmosferycznych z wykorzystaniem modeli rekurencyjnych

Raport

Wojciech Makurat
Mikołaj Fiedorczuk

14 sierpnia 2025

Spis treści

1

Wstęp

Zagrożenie powodziowe spowodowane przez niezwykle obfite opady jest najważniejszym zagrożeniem naturalnym podchodzącym pod rangę katastrofy naturalnej, które może spotkać w naszym klimacie Europy środkowo-wschodniej. Nagłe zmiany pogody obserwowane jako szybki ruch chmur burzowych są szczególnie niebezpieczne nawet w dużo niższym nasileniu dla nawet miliona osób rocznie uprawiających sporty wodne. Zdarzenia pogodowe stanowią dla nich zagrożenie w dużo niższym nasileniu, co przekłada się na dużo częstsze występowanie takich sytuacji. Dlatego też powstał specjalnie dedykowany dla tego celu system wczesnego ostrzegania, który powstał po 2007 roku. Wykorzystuje on obserwacje w wielu pasmach częstotliwości, które mierzą różne parametry atmosfery. Dzięki tym danym klasyczne metody numeryczne tworzą prawdopodobne prognozy średnio i długodystansowe. Moc obliczeniowa wymagana do tworzenia prognoz na podstawie lokalnych pomiarów jest tak duża, że jest sobie w stanie na to pozwolić tylko kilka centrów dysponujących centrami obliczeniowymi. Szczególnie obiecujące w tworzeniu krótko-terminowych lokalnych prognoz o większej dokładności niż obecne metody pokazują algorytmy predykcyjne bazujące na architekturze ConvLSTM, które dzięki swojemu działaniu potrafią uwzględnić długo- i krótkoterminowe cechy czasowe, jak i cechy przestrzenne dzięki zastosowaniu sieci splotowych. Dlatego też zdecydowaliśmy się na zgłębienie tego tematu w niniejszym projekcie.

2

Przykładowe rozwiązania prognozowania czasowego

Na datasecieSEVIRtestowane było wiele rozwiązań. Przykładowe wyniki wyglądają następująco:

Pomimo, że model Earthformer uzyskał najlepsze rezultaty, zdecydowano wykorzystać model ConvLSTM ze względu na prostotę oraz powszechnie dostępną dokumentację.

Model	#Param. (M)	GFLOPS	Metrics							
			CSI-M ↑	CSI-219 ↑	CSI-181 ↑	CSI-160 ↑	CSI-133 ↑	CSI-74 ↑	CSI-16 ↑	MSE (10^{-3}) ↓
Persistence	-	-	0.2613	0.0526	0.0969	0.1278	0.2155	0.4705	0.6047	11.5338
UNet [43]	16.6	33	0.3593	0.0577	0.1580	0.2157	0.3274	0.6531	0.7441	4.1119
ConvLSTM [36]	14.0	527	0.4185	0.1288	0.2482	0.2928	0.4052	0.6793	0.7569	3.7532
PredRNN [45]	46.6	328	0.4080	0.1312	0.2324	0.2767	0.3858	0.6713	0.7507	3.9014
PhyDNet [13]	13.7	701	0.3940	0.1288	0.2309	0.2708	0.3720	0.6556	0.7059	4.8165
E3D-LSTM [44]	35.6	523	0.4038	0.1239	0.2270	0.2675	0.3825	0.6645	0.7573	4.1702
Rainformer [3]	184.0	170	0.3661	0.0831	0.1670	0.2167	0.3438	0.6585	0.7277	4.0272
Earthformer w/o global	13.1	257	0.4356	0.1572	0.2716	0.3138	0.4214	0.6859	**0.7637**	3.7002
Earthformer	15.1	257	**0.4419**	**0.1791**	**0.2848**	**0.3232**	**0.4271**	**0.6860**	0.7513	**3.6957**

Rysunek 2.1: Przykładowe wyniki dla metod powszechnie stosowanych w problematyce [3]

3

Realizacja projektu

3.1 Baseline

3.1.1 Opis zbioru danych (Dataset)

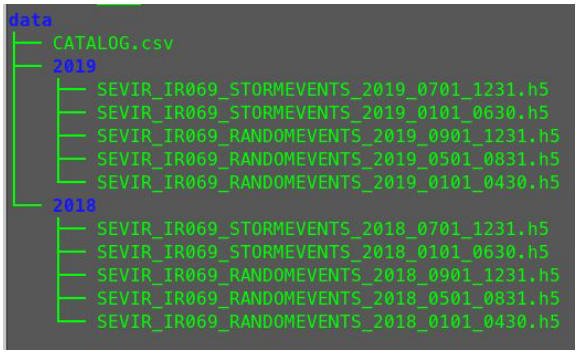
W projekcie wykorzystano zbiór danych **SEVIR** [4], który zawiera sekwencje obrazów związanych z ruchem atmosfery w pięciu różnych kanałach z czego my przyjrzeliśmy się dokładniej trzem najbardziej pasującym do problemu krótkoterminowego prognozowania ruchów mas pary wodnej.

- Vil (verticly integrated liquid) - jest to miara radarowa, mierzonej ilości wody w słupie atmosfery wyrażana w kg/m^2 o rozdzielczości w tym datasecie - 384x384.
- ir107 - infrared 10,7um dłuższe fale które lepiej penetrują atmosferę. Można było czasami na nich zobaczyć ląd który przypominał „smugi" na obiektywie. Bez nakładania na obraz lądu w kolejnym kanale była by to informacja bez wytłumaczenia dla modelu/
- Ir069 - infrared 6,9um któtsze fale które również śledzą wysycenie pary wodnej w atmosferze. Śledzą tylko ruchy mas pary wodnej.

Zdecydowaliśmy się na kanał Ir069, który najlepiej wydawał się odpowiadać założeniom badawczym. Po pobraniu tego kanału za pomocą `awscli` okazało się, że jest to łącznie ok. 45 GB danych, zapisanych w dziesięciu hierarchicznych plikach HDF5 (z rozszerzeniem `.h5`), z opisem indeksów w pliku `.csv`.

Zbiór danych obejmuje:

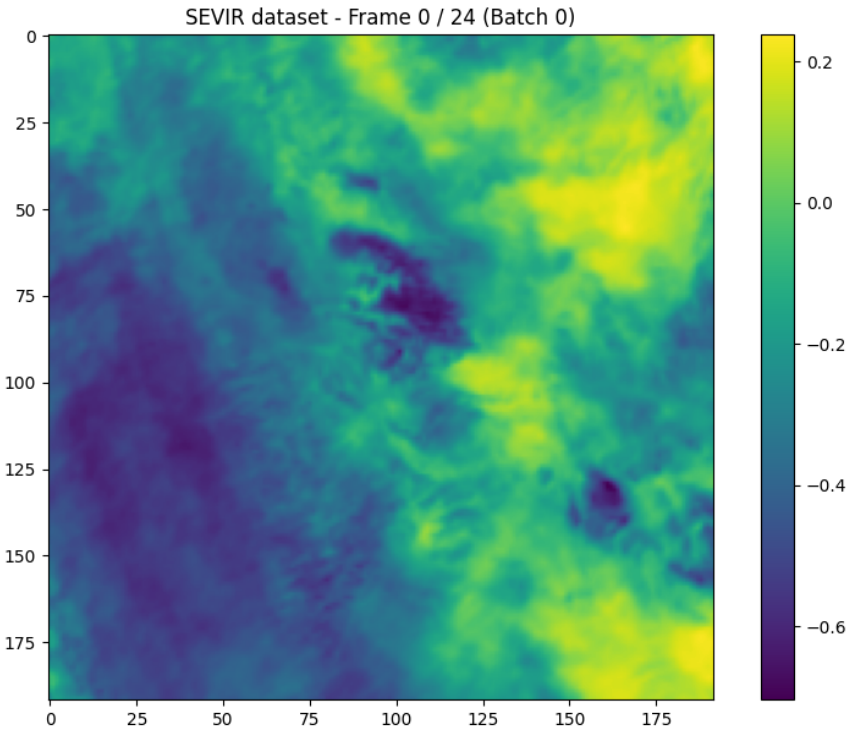
- Obserwacje burz (tzw. *storm events*),



Rysunek 3.1: Struktura datasetu

- Obserwacje ogólne (*mixed events*).

Kanał IR 6,9 μm (*ir069*), tj. zobrazowaniach w podczerwieni, prezentujących ruchy mas powietrza w atmosferze. Każda sekwencja ma rozmiar 192×192 piksele i liczy 49 klatek czasowych, rejestrowanych w odstępach co 5 minut (łącznie 4 godziny obserwacji). Aby zbadać dataset, napisaliśmy kilka funkcji do wizualizacji.



Rysunek 3.2: Enter Caption

Dzięki możliwości spojrzenia na sekwencje pogodowe mogliśmy porównać dostępne rodzaje danych, w tym obserwacje burzowe i obserwacje mieszane, aby lepiej zrozumieć potencjalne warianty treningu i zastosowania modelu.

3.1.2 Eksploracyjna analiza danych

Pierwszym etapem prac była eksploracyjna analiza danych, obejmująca:

W celu ułatwienia tej analizy przygotowano skrypty umożliwiające szybkie wczytywanie, przeglądanie i przetwarzanie wybranych próbek z plików HDF5 wraz z wykorzystaniem pliku indeksowego `.csv`.

3.1.3 Custom dataset

Aby ułatwić pracę z dużą liczbą sekwencji i różnego rodzaju transformacjami, zaimplementowano własną klasę `Dataset` w oparciu o bibliotekę PyTorch. Klasa ta spełnia wymagane metody:

- `__init__(...)` – inicjalizacja i wczytanie informacji o dostępnych sekwencjach,
- `__len__(...)` – określenie liczby sekwencji,
- `__getitem__(...)` – zwracanie pojedynczej sekwencji na żądanie.

Ważnym aspektem jest **leniwe ładowanie (lazy loading)**: w momencie wywołania `__getitem__` pobierana jest tylko ta sekwencja, która jest w danym momencie potrzebna do treningu lub walidacji. Zapewnia to optymalne wykorzystanie pamięci. W samej metodzie `__getitem__` zaimplementowano też różne parametry kontroli (np. co którą klatkę wczytywać, jaki jest docelowy rozmiar klatek czy długość sekwencji).

```
try:
    # otwarcie sampla z pliku za pomocą indeksu lokalnego(właściwego dla danego pliku)
    with h5py.File(file_path, mode: 'r') as f:
        sample = f['ir069'][local_index]
        sample = torch.tensor(sample, dtype=torch.float32)
        # zamienia z 192x192x49 na 49x192x192
        permuted_sample = sample.permute(2, 0, 1)
        # przy kroku 2 zamienia na 25x192x192
        permuted_sample_step = self._get_sample_with_step(permuted_sample, self.step)
        # check czy oczekiwana długość jest mniejsza niż wzięcie co ntej klatki, a torch robi ceil przy samplowaniu
        if self.sequence_length <= math.ceil(TIME_STEPS/self.step):
            permuted_sample_step_len = permuted_sample_step[:self.sequence_length]
        # zmiana rozmiaru na na np. 25 x height x width
        if self.width != 192 or self.height != 192:
            resize = transforms.Resize( size: (self.height, self.width), antialias=True)
            permuted_sample_step_resized = resize(permuted_sample_step_len)
        else:
            permuted_sample_step_resized = permuted_sample_step
        # normalizacja z zakresu 0-255 na 0-1
        permuted_sample_step_resized_normalized = ((permuted_sample_step_resized - DATA_MIN)*2 / (DATA_MAX - DATA_MIN))-1
        permuted_sample_step_resized_normalized_channel = permuted_sample_step_resized_normalized.unsqueeze(1)

        return permuted_sample_step_resized_normalized_channel
```

Rysunek 3.3: Pipeline do przetwarzania danych podczas lazy loadingu

Dataloadery i datamodule

W celu uporządkowania procesu wczytywania i podziału danych na zbiory treningowe, walidacyjne i testowe, wykorzystano `DataModule` z biblioteki *PyTorch Lightning*.

Ułatwia on skalowalne przetwarzanie danych i standaryzuje wywołania związane z:

- `prepare_data()` – jednorazowe przygotowanie lub pobranie danych,

- `setup(stage)` – inicjalizacja odpowiednich `Datasetów` dla danego etapu (`train`, `val`, `test`),
- `train_dataloader()`, `val_dataloader()`, `test_dataloader()` – tworzenie loaderów z odpowiednią konfiguracją.

Dzięki `DataModule` wystarczy utworzyć jeden obiekt, który zarządza całym zbiorem danych, jego podziałem oraz transformacjami.

```
''' pytorch lightning datamodule '''
# przykład użycia
dm = ConvLSTMSevirDataModule(
    step=2,
    width=192,
    height=192,
    batch_size=4,
    num_workers=1,
    sequence_length=9,
    train_files_percent=0.7,
    val_files_percent=0.15,
    test_files_percent=0.15,
    files_dir=file_path_h5_dir)
```

Rysunek 3.4: Struktura obiektu DataModule

W naszym data module zawarliśmy kilka możliwych transformacji jako parametry, które potem testowaliśmy w celu wyłonienia najlepszych parametrów do konkretnych architektur.

1. **step** - parametr odpowiedzialny za branie co ntej klatki z sekwencji(domyślnie 1)
2. **width, height** - rozmiary pojedyńczej klatki
3. **batch size** - ilość sekwencji w jednym batchu
4. **num workers** - ilość wątków zaanagarzowanych w pobieranie danych z dysku
5. **sequence length** - długość sekwencji (domyślnie 49)

6. **train, val, test percents** - podział na zbiory testowy, walidacyjny i treningowy jako procent

Sprzętowe limitacje uczenia

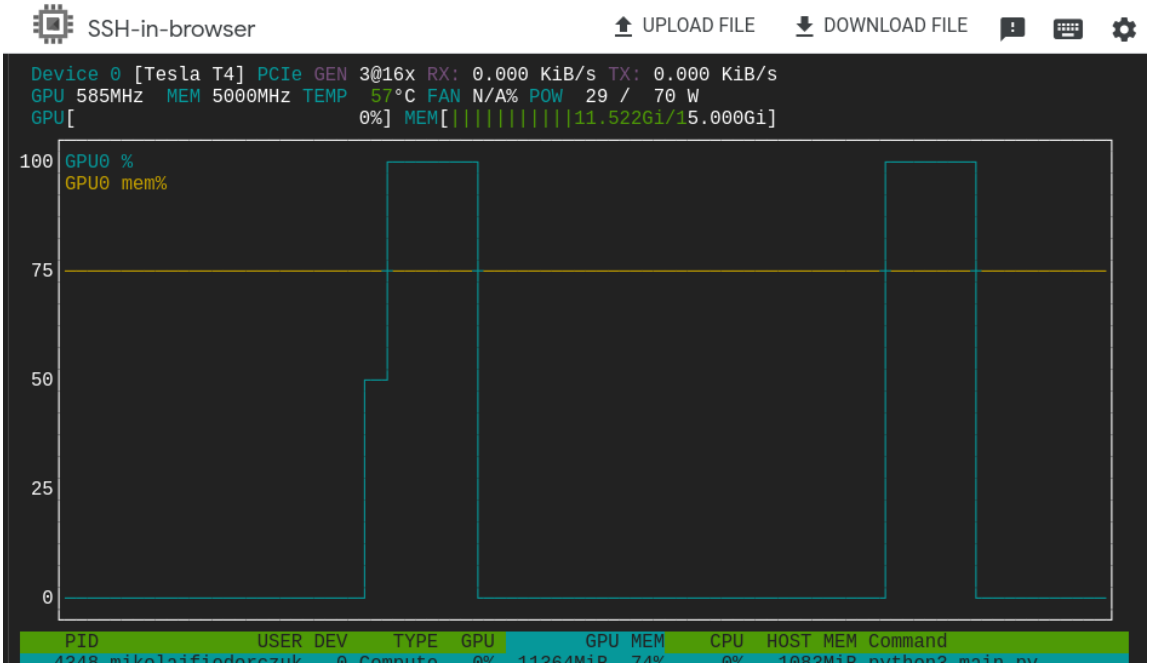
Przy dużym datasecie i stosunkowo skomplikowanej architekturze komórki rekurencyjnej dużą część problemów, jakie musieliśmy pokonać, wynikały z przyczyn niewystarczającej mocy obliczeniowej do trenowania modelu.

Jednym z pomysłów było zaimplementowanie leniwego ładowania w połączeniu ze strumieniowaniem plików bezpośrednio z pamięci chmurowej (*S3*, np. w usłudze AWS). Korzystając z biblioteki *boto* i serwisu *DagsHub*. Mechanizm ten byłby przydatny w szybkim rozpoczynaniu trenowania na nowych platformach, również na Google Colab czy Kaggle Notebook, ale wiązałby się z dużymi problemami przy trenowaniu lokalnym.

Bez mechanizmów jednoczesnego zapisywania albo cache'owania danych, przy kolejnej epoce dataset musiałyby być pobierane od nowa, co znacząco limitowałoby prędkość uczenia.

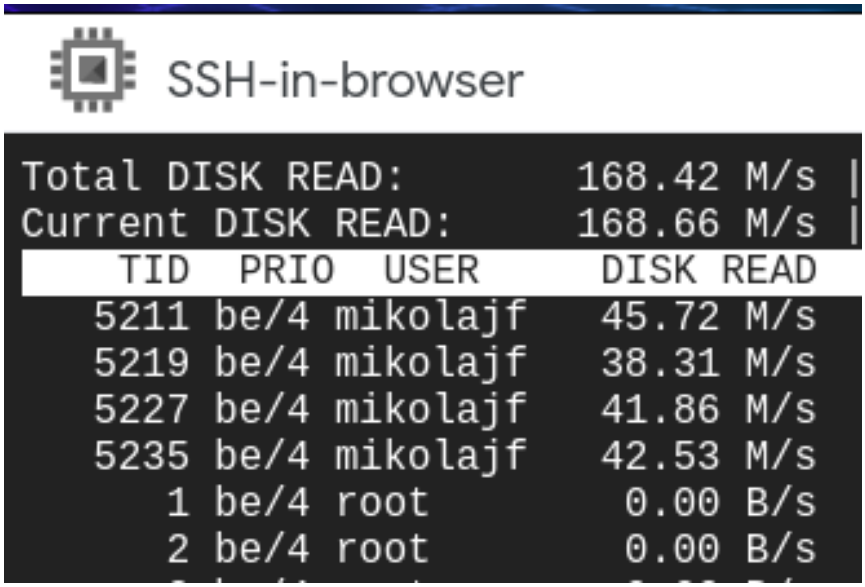
Najlepszym ustawieniem, do jakiego doszliśmy, okazało się tworzenie i testowanie kodu w zakresie jego poprawności na własnych komputerach oraz pobieranie repozytorium przez git na stacji roboczej i przeprowadzanie tam eksperymentów w serii.

Próbowaliśmy wykorzystać zasoby chmurowe Google Cloud, tworząc maszynę wirtualną z kartą graficzną Tesla T4 z 16 GB vRAM. Proces uczenia okazał się jednak skrajnie nieefektywny przez problemy z wąskim gardłem sprzętowym, co zostało pokazane narzędziem `nvtop`.



Rysunek 3.5: Limitowanie wykorzystania karty graficznej przez inny komponent

problemy związane z maszyną wirtualną okazały się spowodowane wąskim gardłem na poziomie dysku, które limitowało przepływ danych do tylko 160 MB/s. Było to spowodowane ograniczeniem na GCP, które uniemożliwiało wykorzystanie dysku SSD na naszym poziomie umowy licencyjnej, co powodowało, że pomimo możliwości karty graficznej jej wydajność była ograniczona prawie do 15% jej zwykłej mocy, co pokazuje ten zrzut z narzędzia `iotop`, mierzącego prędkość poboru i zapisu danych z pamięci trwałej.



Rysunek 3.6: Wąskie gardło w postaci wolnego pobierania danych z dysku

Dzięki przerzuceniu trenowania na stację roboczą wyposażoną w SSD wykorzystujące dysk NVMe M.2 SSDs z szyną PCIe 5.0, zapewniającą maksymalne 15 GB/s, i przy dobrym dobraniu wielkości batcha treningowego, udało się nam odwrócić praktycznie trend wykorzystania karty graficznej, w której to jedynie przestawała wykonywać operacje podczas oczekiwania na skończenie walidacji pod koniec epoki treningowej.

Rysunek 3.7: Optymalne wykorzystanie GPU

3.1.4 Implemenacja modelu

Rodzina modeli RNN dla obrazów

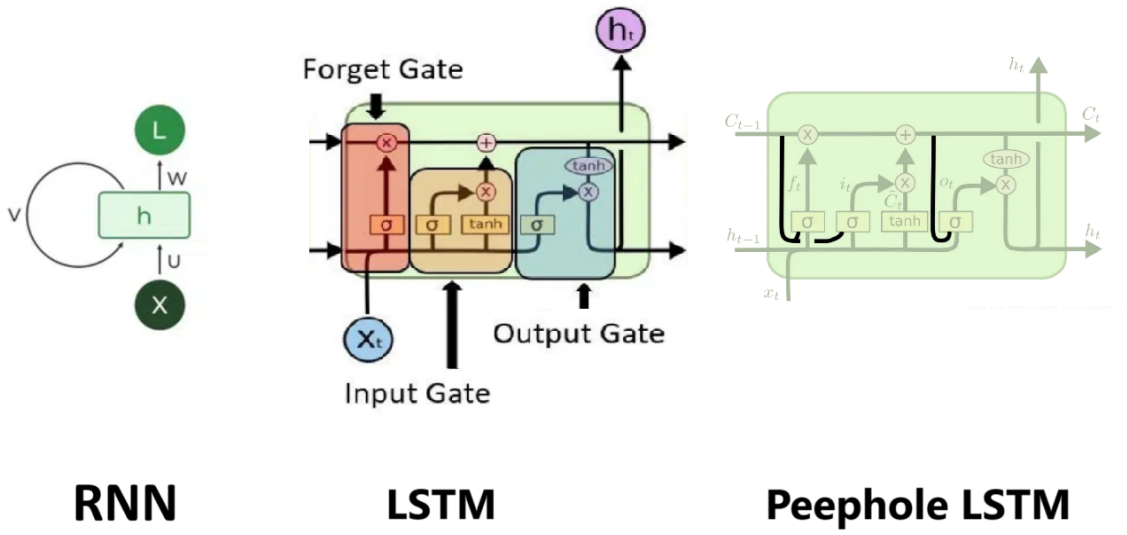
W projekcie rozważano modele z rodziny **RNN (Recurrent Neural Networks)**, które przetwarzają sekwencje klatek w czasie. Podstawowym wariantem był **ConvLSTM**, gdzie zwykłe liniowe operacje w LSTM zostały zastąpione warstwami splotowymi (ang. *Convolutional Layers*). LSTM jest wariantem RNN przeznaczonym do lepszego uchwycenia długoterminowych zależności czasowych.

Sprawdzono także prostsze i bardziej zaawansowane odmiany:

- **ConvRNN** – prostsza sieć rekurencyjna ze splotami,
- **Peephole ConvLSTM** – rozszerzenie ConvLSTM o tzw. *peepholes*, co umożliwia dodatkowy wgląd wewnętrzny w stan komórki.

Struktura głównego modelu (*RainPredictor*)

W celu ujednolicenia eksperymentów zaimplementowano klasę *RainPredictor*, która przyjmuje w konstruktorze komórkę (*cell*) wybranego modelu RNN (np. *ConvLSTMCell*, *ConvRNNCell*, *PeepholeConvLSTMCell*).



Rysunek 3.8: Testowane architektury

- Komórka modelu RNN przetwarza sekwencje klatek jedna po drugiej, wykorzystując stan ukryty powstały przy przetwarzaniu poprzedniej klatki.
- Wyjście komórki może mieć wiele kanałów, dlatego dodatkowo zastosowano warstwę mapującą (warstwę splotową), aby dostosować liczbę kanałów wyjściowych do wartości docelowych (w naszym przypadku jeden kanał).
- Architekturę konfiguruje się parametrami takimi jak liczba kanałów w stanie ukrytym, głębokość warstwy splotowej (liczba jej poziomów) czy funkcje aktywacji w warstwach.

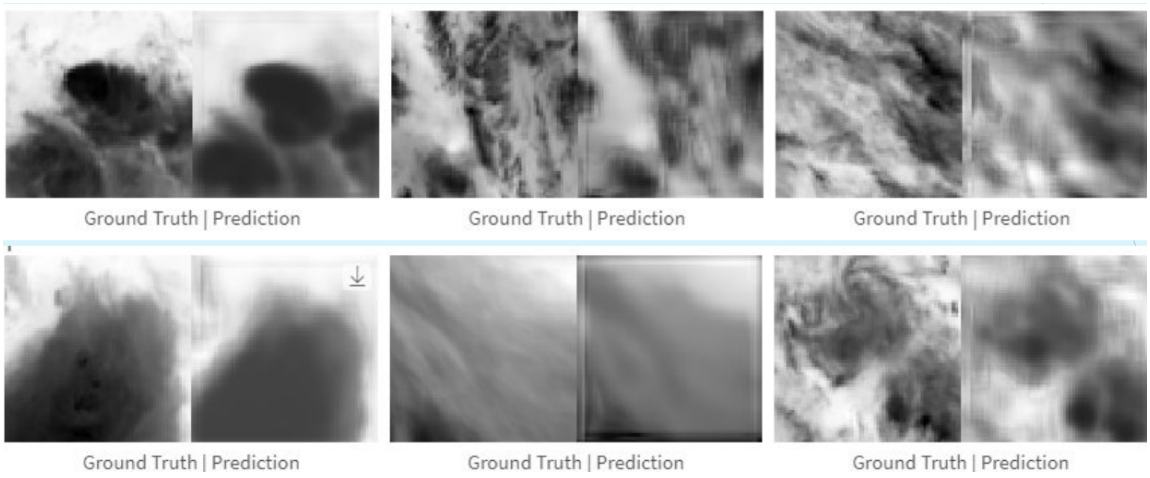
Przeprowadzone badania i wyniki

Do przeprowadzenia badań wykorzystano środowiska Hydra - pozwalające na zarządzanie konfiguracjami eksperymentów oraz Weights & Biases do monitorowania i analizowania wyników.

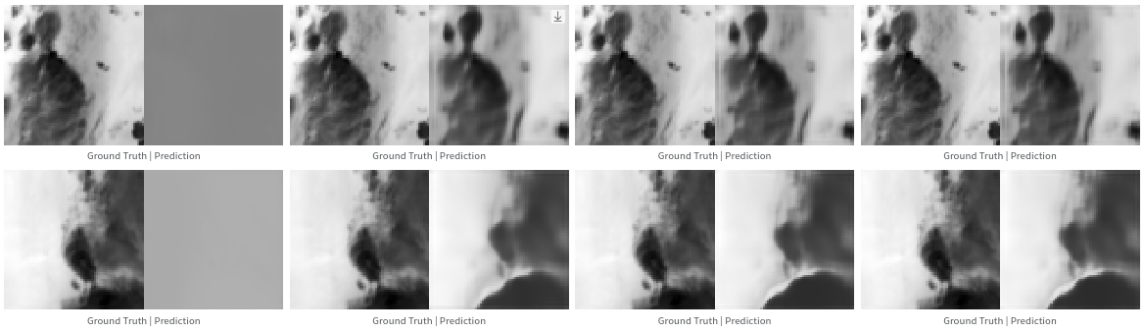
W ramach badań przeprowadzono kilkadzieścia testów, gdzie głównie sprawdzane były różne kombinacje typu komórki RNN (niestety w ramach pierwszego etapu nie udało się zbadać skuteczności Peephole ConvLSTM), rozmiar kernela, głębokość sieci czy liczba kanałów warstwy ukrytej. Poniżej przedstawiono jedynie przykładowe wyniki.

RNN Type	Kernel Size	Depth	Channels	MSE
ConvRNN	5	3	10	0.049
ConvRNN	7	1	8	0.045
ConvRNN	5	2	8	0.173
ConvRNN	5	1	12	**0.018**
ConvLSTM	3	2	12	0.034
ConvLSTM	7	1	8	0.026
ConvLSTM	7	2	8	0.046

Tabela 3.1: Przykładowe uzyskane wyniki (najlepszy wynik wyróżniony)



Rysunek 3.9: Przykładowe obrazy generowane przez model zestawione z poprawnymi



Rysunek 3.10: Pokazanie zmian w predykcji wraz z kolejnymi epokami

Jak widzimy, najwieksze różnice można zauważyć w pierwszych etapach uczenia, co jest standardowym spostrzeżeniem.

3.2 Usprawnienia i badania

3.2.1 Hydra

Aby uprościć znacząco kod, zmieniliśmy strukturę pliku konfiguracyjnego `config.yaml` zawierającego konfigurację modelu, która jest inicjalizowana jako obiekt za pomocą hydra.utils.instantiate(). Główną zaletą tej zmiany było to, że nie musieliśmy już „wypakowywać"obiektu model z pytorch-lightning, tylko można było wykorzystać jego inicjalizację bezpośrednio z pliku konfiguracyjnego. Wykorzystaliśmy jego strukturę w hierarchiczny sposób tak, aby wszystkie jego elementy były podległe pod jeden obiekt główny. Dzięki temu uzyskaliśmy kod z jednym obiektem konfiguracyjnym.

```
  checkpoint:
-   monitor: "validation_loss"
-   dirpath: "./model/"
-   filename: "model-{epoch:02d}-{validation_loss:.2f}"
-   save_top_k: 3
-   mode: "min"
+     monitor: "validation_loss"
+     dirpath: "./model/"
+     filename: "model-{epoch:02d}-{validation_loss:.2f}"
+     save_top_k: 3
+     mode: "min"
```

Rysunek 3.11: Przykład zmiany w pliku konfiguracyjnym

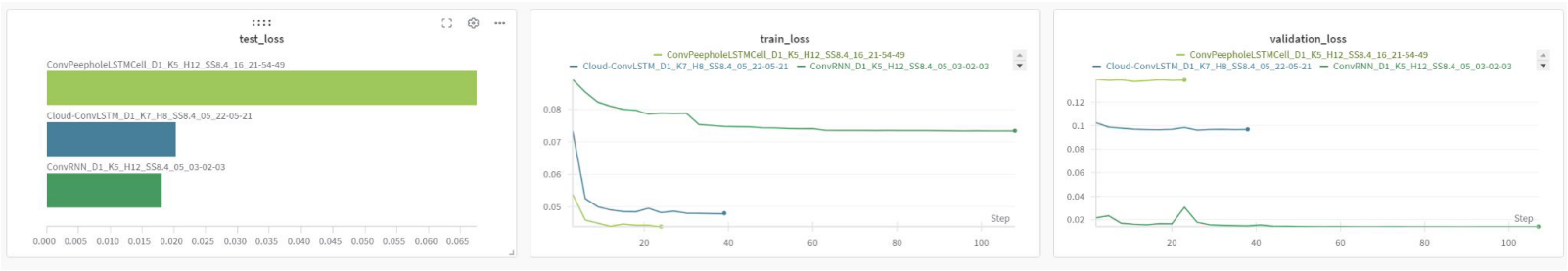
3.2.2 Peephole ConvLSTM

W ramach poprzedniego etapu projektu nie udało się zaimplementować modelu z Peephole ConvLSTM. W ramach tego projektu model ten został poprawnie zaimplementowany, a także przeprowadzono na nim szereg testów.

3.2.3 Próby z Batch Normalization

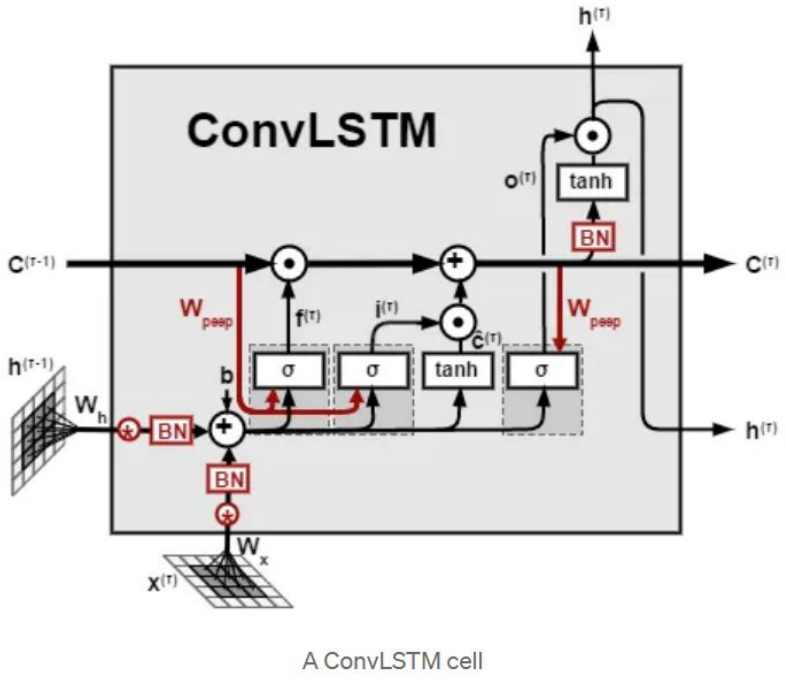
Podczas nauki modelu zauważono, że najbardziej złożony model (głębokość warstw, rozmiar kernela) uzyskuje znacznie lepsze wyniki dla zbioru treningowego, jednak w przypadku walidacji i testu znacznie odstaje od najlepszych z ConvLSTM i Peephole ConvLSTM. Pośredni efekt wystąpił w przypadku ConvRNN. Sugeruje to

wystąpienie overfittingu, na który prostszy model jest bardziej odporny.



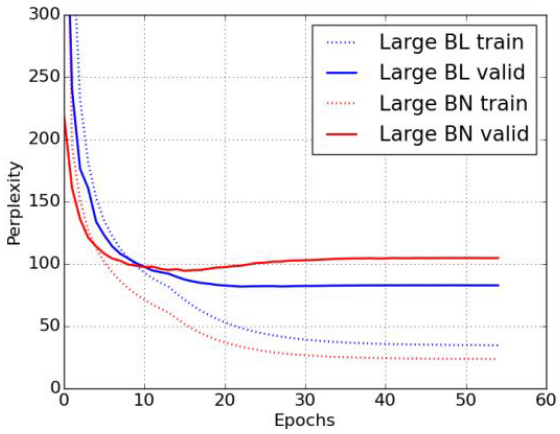
Rysunek 3.12: Przebieg trenowania dla przykładowego modelu Peephole ConvLSTM w zestawieniu z najlepszymi modelami RNN i LSTM

W celu walki z overfittingiem dodano warstwę batch normalization po splocie wewnątrz komórki ConvLSTM, sugerując się modelem z artykułu („An Introduction to ConvLSTM") [1]. Chociaż dla walidacji wyniki wydawały się poprawić, wynik testu nie wykazał skuteczności podejścia (brązowy wykres).

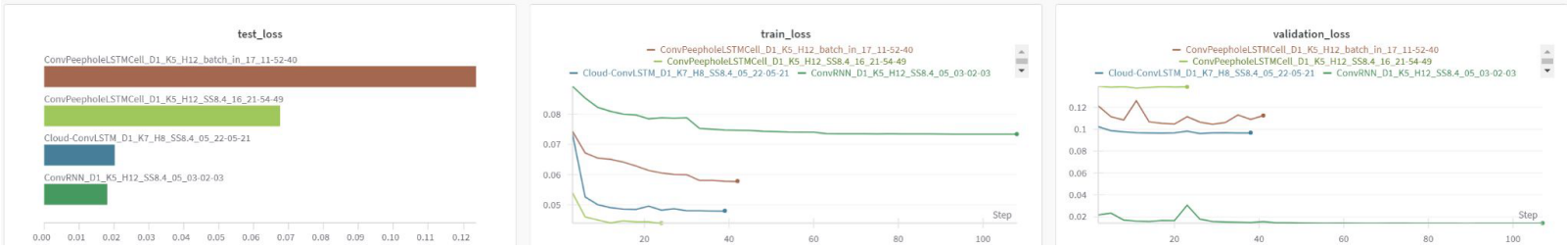


Rysunek 3.13: Architektura Peephole ConvLSTM z zawartym Batch Normalization

Artykuł [2] wykazuje, że dla modeli rekurencyjnych Batch Normalization nie poprawia wyników. Efekt ten był spotęgowany początkowo niewielkim rozmiarem batcha (15).

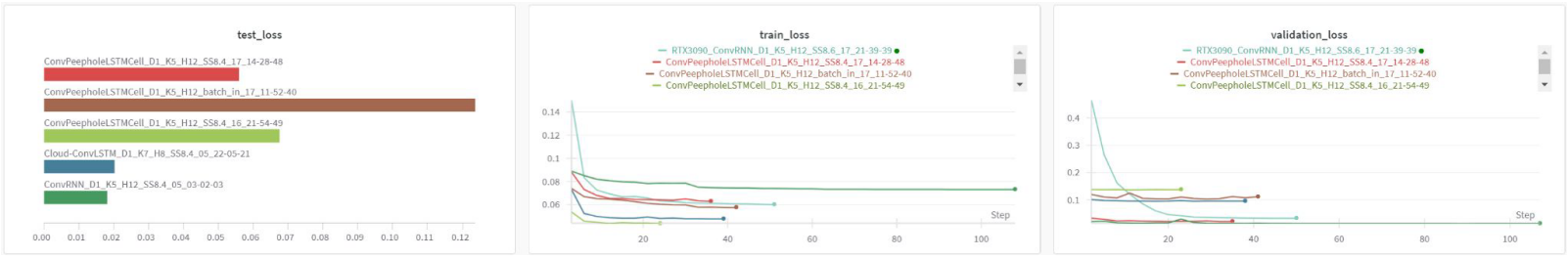


Rysunek 3.14: Zobrazowanie skuteczności BN w RNN



Rysunek 3.15: Wyniki po wprowadzeniu Batch Normalization wewnątrz komórki RNN

Następnie umieszczono normalizację batcha na zewnątrz komórki RNN, przed wejściem do warstwy mapującej. Jednocześnie zwiększono rozmiar batcha do 30. Przebieg jest zaznaczony kolorem czerwonym.



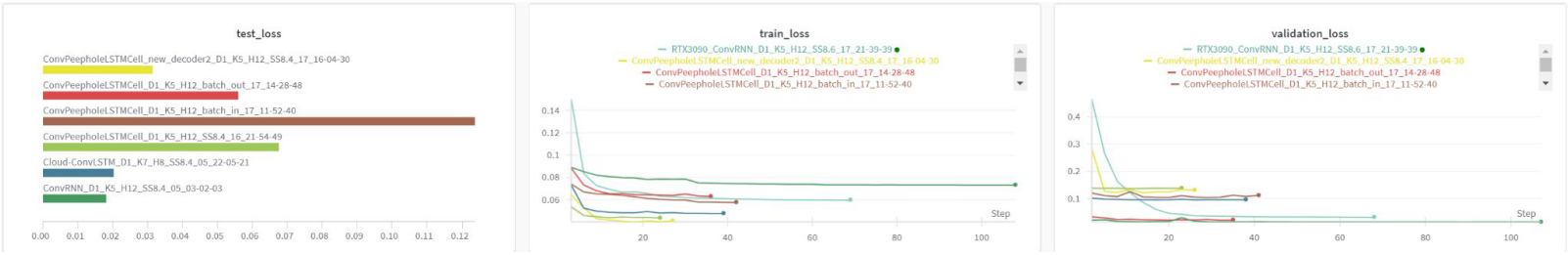
Rysunek 3.16: Wyniki po wprowadzeniu Batch Normalization na wyjściu komórki RNN

3.2.4 Wprowadzenie warstwy dekodującej dla modeli LSTM

Model RNN różni się od LSTM kilkoma cechami. Specyficzną różnicą jest założenie, że RNN posiada warstwę dekodującą i output jest zwracany po wykonaniu dekodowania. Dla LSTM literatura wskazuje, że warstwa ukryta stanowi wyjście i dopiero następnie należy je rzutować na dane wyjściowe algorytmu.

W pierwszym etapie projektu rzutowanie odbywało się jedynie z wykorzystaniem warstwy splotowej o kernelu 1x1, zmieniającej liczbę kanałów wyjściowych na 1 oraz aktywacji ReLu.

Może to sugerować, że w poprzednim etapie RNN miało przewagę w postaci bardziej rozbudowanej warstwy dekodującej. Przeniesiono ją więc na zewnątrz RNN i wykorzystano jako ogólny dekoder także dla LSTM. Przebieg zaznaczony kolorem żółtym.



Rysunek 3.17: Wyniki po wykorzystaniu warstwy dekodującej dla Peephole ConvLSTM

3.2.5 Dalsze badania

W ramach dalszych badań wypróbowano wiele różnych kombinacji. Jednocześnie Dalsze uczenie przeprowadzono na urządzeniu umożliwiającym znaczne zwiększenie rozmiaru batchy do 4096 co w teorii powinno poprawić stabilność warstwy normalizacji batch.

RNN Type	**Kernel Size**	**Depth**	**Channels**	**MSE**
ConvRNN	7	1	8	**0.016**
ConvLSTM	7	1	8	0.02
PeepHoleConvLSTM	7	1	12	0.025
ConvRNN	5	3	8	0.026
PeepHoleConvLSTM	5	1	15	0.031
PeepHoleConvLSTM	9	1	18	0.033
ConvRNN	5	1	12	0.034
ConvLSTM	7	2	8	0.046

Tabela 3.2: Przykładowe uzyskane wyniki

Wyniki wskazują, że udało się poprawić wyniki zarówno dla ConvLSTM i ConvRNN.

3.2.6 Najlepszy wynik

Najlepszy wynik uzyskano dla konfiguracji:

- model komórki RNN: ConvRNN
- liczba kanałów warstwy ukrytej: 8
- głębokość warstwy splotowej w komórce RNN: 1
- rozmiar kernela: 7

4

Problem przeprowadzonych badań

Niestety podczas badania zauważono, że wyniki są bardzo niestabilne, co widać na poniższym obrazie, gdzie dwukrotnie wyuczono model na identycznych parametrach, a wyniki okazały się znacznie różne. Sugeruje to, że wybrana najlepsza konfiguracja nie musi być najbardziej sprzyjająca. Wymaga to dalszych badań.

Rysunek 4.1: Zestawienie wyników dwóch przykładowych modeli uczonych z wykorzystaniem tej samej konfiguracji

5

Podsumowanie

Wyniki testów pokazały, że rodzina rekurencyjnych sieci neuronowych może być skutecznie wykorzystywana do krótkoterminowej predykcji opadów - uzyskane MSE okazało się mieć ten sam rząd wielkości (około czterokrotnie wyższe), co standardowe metody wykorzystywane w tym celu. Wynik ten jest zgodny z początkowymi założeniami zważywszy na prostotę zaprojektowanego algorytmu oraz ograniczoną ilość danych - wykorzystano tylko jeden z kanałów dostępnych w datasecie.

Choć w przypadku wielu zastosowań sieć ConvLSTM jest skuteczniejsza od ConvRNN, ze względu na krótką sekwencję wejściową, składającą się z siedmiu obrazów, prostsza sieć ConvRNN okazała się skuteczniejsza (także od najbardziej złożonego wariantu - Peephole ConvLSTM).

Zmiany wprowadzone w drugim etapie pozwoliły poprawić wynik zarówno dla modelu ConvLSTM iaj i ConvRNN.

Niestety, jak wykazano, że wyniki uczenia sieci nie są stabilne, co ogranicza wiarygodność badań.

Bibliografia

[1] Neuronio, *An Introduction to ConvLSTM*, Medium, 2018. Dostępne online: `https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7` (dostęp: 18 stycznia 2025).

[2] Laurent, César, Pereyra, Gabriel, Brakel, Philémon, Zhang, Ying, Bengio, Yoshua, *Batch normalized recurrent neural networks*, w: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2016, s. 2657–2661.

[3] Neuronio, *Numeryczne modele pogody AI w służbie meteorologicznej IMGW-PIB.*, Centrum Modelowania Meteorologicznego. Dostępne online: `https://cmm.imgw.pl/?page_id=39712` (dostęp: 18 stycznia 2025).

[4] SEVIR Dataset, Papers with Code. Dostępne online: `https://paperswithcode.com/dataset/sevir` (dostęp: 18 stycznia 2025).

[5] Shi, Xingjian i in., *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*, w: *arXiv preprint arXiv:1506.04214*, 2015. Dostępne online: `https://arxiv.org/pdf/1506.04214`.

[6] Repozytorium ConvLSTM, GitHub. Dostępne online: `https://github.com/ndrplz/ConvLSTM_pytorch` (dostęp: 18 stycznia 2025).

[7] Tianchi Rainfall Forecasting Competition, Alibaba Tianchi. Dostępne online: `https://tianchi.aliyun.com/competition/entrance/231596` (dostęp: 18 stycznia 2025).

[8] Gao, Zhihan, Shi, Xingjian, Wang, Hao, Zhu, Yi, Wang, Yuyang Bernie, Li, Mu, Yeung, Dit-Yan, *Earthformer: Exploring space-time transformers for earth system forecasting*, w: *Advances in Neural Information Processing Systems*, vol. 35, s. 25390–25403, 2022.