Low-Level Wind Shear Alert Prediction System Using Machine Learning

Ejha Larasati Siadari

June 2023

1. INTRODUCTION

The utilization of machine learning techniques in the Low-Level Wind Shear Alert Prediction System represents a novel approach to enhance the forecasting and identification of low-level wind shear occurrences. Low-level wind shear, characterized by rapid changes in wind speed and direction over short distances, poses significant challenges and risks to aviation operations, especially during critical phases such as takeoff and landing. In this specific research, we tried to predict the occurrence of low-level windshear in Soekarno Hatta International Airport using one of the machine learning techniques of Artificial Neural Network, which is The Temporal Convolutional Network (TCN). We aimed to predict the occurrence of wind shear in the next one hour within 15 minutes windows by utilizing 10 minutes data points.

Soekarno-Hatta airport is located in Tangerang, Indonesia and consists of two runway zones. Each runway zone is surrounded by 12 LLWAS (Low-Level Wind Shear Alert System) anemometers, as depicted in Figure 1. The LLWAS Wind shear warning data is generated by analyzing the divergence of wind speed and direction data using an algorithm developed by Wilson in 1991. The analysis involves three sensors selected from the 12 existing sensors, which represent the zone surrounding them. The LLWAS algorithm compares the wind speed and direction data from these three sensors to determine the level of divergence. If the divergence value exceeds a predefined threshold, the LLWAS system considers the corresponding area as a wind shear zone. Since Soekarno-Hatta airport has two runways, the LLWAS system divides the runway area into four parts, with each end of the runway having two divided sections. This division allows for more specific identification and monitoring of wind shear conditions in different areas of the airport.

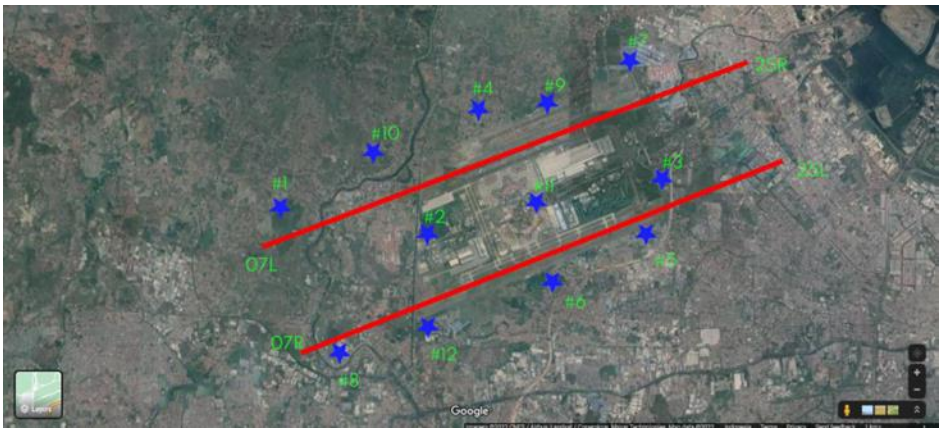

Fig. 1. LLWAS anemometer placement at Soekarno-Hatta airport (Ryan, et.all, 2021).

2. DATA

Our raw data consists of two datasets, which are the LLWAS warning data and the Anemometer Sensors data. LLWAS data is every 10 seconds from 8 parts of the runway over a period of 40 days, from April 1st to May 10th, 2018. Anemometer sensors data contained measurements of wind speed and direction at a height of 10 meters above ground level from 12 sensors collected over a period of 40 days, from April 1st to May 10th, 2018. In the preprocessing part, there is data cleaning, u and v conversion, and interpolation. During the data cleaning process, approximately 30% of the data was removed. This involved discarding all sequences that contained missing values. To convert the wind direction data, it was transformed into x-y components of wind. In cases where there were only 10 seconds of missing data in our 10 minutes sequences, a linear interpolation method was applied to fill in the gaps.

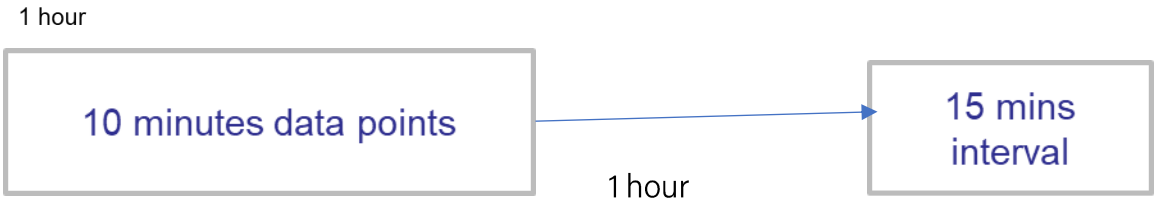

Fig. Input and output (x,y) in our model.

Our dataset consists of predictors (u and v components of wind from 12 sensors) at 10-minute intervals, which serve as the input (x). Within this dataset, there is a one-hour gap, and within each 15-minute interval, we examine whether a wind shear warning is present or not. The target variable (y) is represented by binary values: 0 indicates no wind shear warning, while 1 indicates the presence of a wind shear

warning. In total, our dataset comprises 352,601 instances. A small portion, specifically 0.014% of the dataset, is labeled as wind shear warning, while the remaining instances are labeled as no wind shear warning.

The dataset initially included all instances labeled as wind shear warnings, along with an equal number of randomly selected instances labeled as no wind shear warnings. Prior to performing under-sampling, there were 345,601 instances labeled as no wind shear warnings, while the instances labeled as wind shear warnings corresponded to "rwy_25 ra" with a total of 416 positive values. After applying under-sampling techniques, the total dataset used for each scheme consisted of 332,195 instances. This dataset was split into an 80% training set and a 20% test set, with the training set containing 265,756 instances and the test set containing 66,439 instances.

3. METHODS

   a. Temporal Convolutional Network (TCN)

      The TCN is a specialized neural network model designed for predicting low-level wind shear occurrences by effectively processing sequential wind data. It was introduced by Lea et al. in 2016 and has gained popularity due to its ability to capture long-term dependencies in sequential data while remaining computationally efficient. In the context of low-level wind shear prediction, TCN's unique architecture becomes highly relevant. It utilizes one-dimensional dilated convolutions, which allow the model to analyze the sequential nature of wind data effectively. By adjusting the dilation factor in deeper layers, TCN can capture both local and global patterns in the data, which is crucial for detecting wind shear events.

      The significance of TCN in low-level wind shear prediction lies in its capability to capture complex temporal dependencies across different time scales. Traditional models, like recurrent neural networks (RNNs), often struggle to model these long-range dependencies accurately. However, TCN overcomes this limitation by using dilated convolutions. Moreover, TCN's parallel processing capabilities contribute to its efficiency, enabling faster training and inference times. This is particularly valuable in low-level wind shear prediction, as real-time alerts are essential for ensuring aviation safety during critical flight phases.

b. Model Evaluation

We used three metrices to evaluate the model. The first metric is recall, also known as true positive rate or sensitivity, measures the proportion of actual positive cases (occurrence of wind shear) that are correctly identified by the model (TP / (TP + FN)). Then, precision which quantifies the accuracy of positive predictions made by the model (TP / (TP + FP)) and lastly accuracy which measures the overall correctness of the model's predictions (both true positives and true negatives) over the total number of predictions ((TP + TN) / (TP + FP + TN + FN)).

In the context of wind shear prediction, the occurrence of wind shear events is often rare compared to non-wind shear conditions. This creates an imbalanced dataset where the majority of instances belong to the non-wind shear class. In such cases, a prediction model that always predicts the majority class (non-wind shear) will achieve high accuracy but fail to detect the critical wind shear events.

c. Tradeoff between FP and FN

False negatives occur when the wind shear prediction system fails to identify its presence. Undetected wind shear increases the risk of accidents and compromises flight safety. The trade-off between FP and FN is about finding the right balance. The system needs to minimize both false positives and false negatives to ensure accurate and reliable wind shear predictions. However, reducing one type of error often comes at the expense of increasing the other.

In the case of the Low-Level Wind Shear Alert Prediction System, it is crucial to minimize false negatives (FN). This is because the system's primary objective is to provide timely warnings for wind shear events to enhance aviation safety. Missing a wind shear event (FN) could lead to potential hazards and accidents. Therefore, the focus is on reducing false negatives, even if it means accepting a slightly higher number of false positives.

By optimizing the system's parameters, such as adjusting the prediction threshold or fine-tuning the machine learning algorithms, efforts can be made to strike a balance between FP and FN, aiming to minimize the occurrence of both types of errors and improve the overall accuracy and reliability of wind shear predictions.

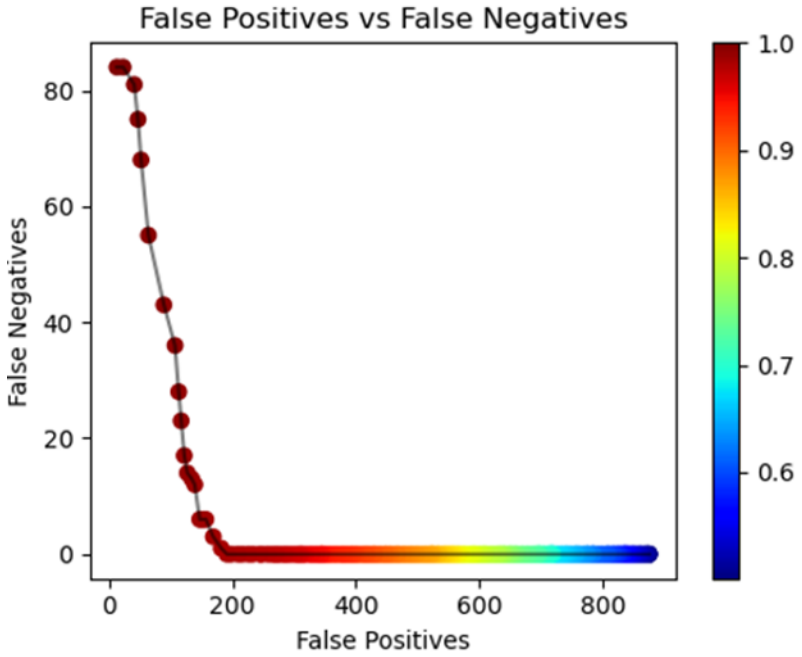

Fig. 3. Trade off between FP and FN

d. Best thresholds
The best threshold in this case is considered the threshold value that leads to zero false negatives (FN). It is the threshold at which the system accurately identifies all instances of wind shear, without missing any. By identifying this optimal threshold, the system can be fine-tuned to minimize the risk of missing critical wind shear events, ensuring a high level of accuracy and safety in wind shear predictions.

4. RESULTS
We tried to conduct 4 different versions of predictors as an input to feeding our model and we evaluated those 4 versions using the metrices and as well as looking for the trade off between FP and FN. We also compared the TCN model with baseline model. In the context of the Low-Level Wind Shear Alert Prediction System, the baseline model refers to a simple, standard or default model that serves as a benchmark for performance comparison. In our case, the baseline number of false positives is always 0, since we are predicting all negatives.

a. Version1- using u and v components of wind, by dropping the missing values. In Version 1, the system discards any sequences or data instances that have missing values. This means that if any data points within a sequence or timeframe

contain missing values, the entire sequence is excluded from the analysis. By doing so, we are only using 70% of our dataset.

```
In [23]: # baseline

         # TN | FP
         # ---|---
         # FN | TP
         baseline = np.zeros(len(target))
         confusion_matrix(target, baseline)

Out[23]: array([[20418,     0],
                [   20,     0]], dtype=int64)
```

Figure 4. Baseline accuracy, baseline FP, and baseline FN

The values of our baseline accuracy, baseline FP, and baseline FN are 99.90214306683629, 0, 20. This means that the baseline accuracy of the model is 99.90%. The baseline number of false positives is 0, since we are predicting all negatives. The baseline number of false negatives.

```
In [22]: # best threshold with n false negatives
         n = 0
         best_threshold = thresholds[np.where(fn == n)[0]][-1]
         print(best_threshold)

         # TN | FP
         # ---|---
         # FN | TP
         confusion_matrix(target, probs > best_threshold)

         0.548

Out[22]: array([[20395,    23],
                [    0,    20]], dtype=int64)
```

Figure 5. Best threshold and accuracy

Our model is able to reduce FN to zero. But as a result of the trade off, we got 23 FP. In this case, the model achieved a validation accuracy of 99.87% when using the best threshold of 0.548. We also consider other evaluation metrices as shown in Table 1.

Version1	
Metrices	Values
Recall	1.0
Precision	0.465
Accuracy	0.9989

Table 1. Model evaluation from version 1

b. Version2- using u and v components of wind, linearly interpolated missing data.
Version 2 in this case incorporates the u and v components of wind data while handling missing values through linear interpolation. This approach aims to address the missing values in the data by estimating their values based on neighboring data points.

Linear interpolation involves filling in missing values by estimating them as points along a straight line between adjacent known data points. By utilizing this technique, the system can retain more data instances (90% of the data) and make predictions based on a more complete dataset.

```
In [24]: # baseline

         # TN | FP
         # ---|---
         # FN | TP
         baseline = np.zeros(len(target))
         confusion_matrix(target, baseline)

Out[24]: array([[66355,     0],
                [   84,     0]], dtype=int64)
```

Figure 6. Baseline accuracy, baseline FP, and baseline FN

The values of our baseline accuracy, baseline FP, and baseline FN are 99.87356823552432, 0, 84. This means that the baseline accuracy of the model is 99.87% with zero FP and 84 FN.

```
In [22]: # best threshold with n false negatives
         n = 0
         best_threshold = thresholds[np.where(fn == n)[0]][-1]
         print(best_threshold)

         # TN | FP
         # ---|---
         # FN | TP
         confusion_matrix(target, probs > best_threshold)

         0.9410000000000001

Out[22]: array([[65581,   774],
                [    0,    84]], dtype=int64)
```

Figure 7. Best threshold and accuracy

Our model is able to reduce FN to zero. But as a result of the trade off, we got 773 FP. In this case, the model achieved a validation accuracy of 99.84% when using the best threshold of 0.941. We also consider other evaluation metrices as shown in Table 2.

Version2	
Metrices	Values
Recall	1.0
Precision	0.097
Accuracy	0.9984

Table 1. Model evaluation from version 2

c. Version2- using u and v components of wind, linearly interpolated missing data.

In this version, the system calculates the variance and mean of the u and v components for each sequence or timeframe. These statistical measures provide insights into the dispersion and average values of the wind data, offering a representation of the overall wind characteristics.

Furthermore, if there are missing values within a sequence or timeframe, the system employs linear interpolation to estimate those missing values. By linearly interpolating the missing data points, the system ensures that a more complete dataset is used for predicting low-level wind shear events.

```
In [24]: # baseline

         # TN | FP
         # ---|---
         # FN | TP
         baseline = np.zeros(len(target))
         confusion_matrix(target, baseline)

Out[24]: array([[66355,     0],
                [   84,     0]], dtype=int64)
```

Figure 8. Baseline accuracy, baseline FP, and baseline FN

The values of our baseline accuracy, baseline FP, and baseline FN are 99.87356823552432, 0, 84. This means that the baseline accuracy of the model is 99.87% with zero FP and 84 FN.

```
In [24]: # best threshold with n false negatives
         n = 12
         best_threshold = thresholds[np.where(fn == n)[0]][-1]
         print(best_threshold)

         # TN | FP
         # ---|---
         # FN | TP
         confusion_matrix(target, probs > best_threshold)

         0.987

Out[24]: array([[66216,   139],
                [   12,    72]], dtype=int64)
```

Figure 9. Best threshold and accuracy

Our model is able to reduce FN to zero. But as a result of the trade off, we got 139 FP. In this case, the model achieved a validation accuracy of 99.77% when using the best threshold of 0.987. We also consider other evaluation metrices as shown in Table 3.

Version3	
Metrices	Values
Recall	0.857
Precision	0.341
Accuracy	0.9972

Table 1. Model evaluation from version 3

d. Version4- using variance and mean of u and v components of wind, linearly interpolated missing data, and 200 seconds interval data.

In this version, the system calculates the variance and mean of the u and v components of wind for each data sequence or timeframe. These statistical measures provide insights into the variability and average values of the wind, capturing important characteristics of wind behavior that may be indicative of wind shear events. Moreover, this version introduces a shorter interval of 200 seconds to feed into the model.

```
In [24]: # baseline

         # TN | FP
         # ---|---
         # FN | TP
         baseline = np.zeros(len(target))
         confusion_matrix(target, baseline)

Out[24]: array([[66355,     0],
                [   84,     0]], dtype=int64)
```

Figure 10. Baseline accuracy, baseline FP, and baseline FN

The values of our baseline accuracy, baseline FP, and baseline FN are 99.87356823552432, 0, 84. This means that the baseline accuracy of the model is 99.87% with zero FP and 84 FN.

```
In [24]: # best threshold with n false negatives
         n = 0
         best_threshold = thresholds[np.where(fn == n)[0]][-1]
         print(best_threshold)

         # TN | FP
         # ---|---
         # FN | TP
         confusion_matrix(target, probs > best_threshold)

         0.898

Out[24]: array([[65854,   501],
                [    0,    84]], dtype=int64)
```

Figure 9. Best threshold and accuracy

Our model is able to reduce FN to zero. But as a result of the trade off, we got 501 FP. In this case, the model achieved a validation accuracy of 99.24% when using the best threshold of 0.898. We also consider other evaluation metrices as shown in Table 3.

5. CONCLUSIONS

Data cleaning, incorporating variance as additional targets, and utilizing longer sequences instead of shorter sequences are three strategies that can enhance the performance of the model in the context of the Low-Level Wind Shear Alert Prediction System. Data cleaning involves preprocessing the data to remove missing values. By performing thorough data cleaning, the model is trained on a cleaner and more reliable dataset, which can improve its ability to capture meaningful patterns and make accurate predictions. By considering variance as an additional target, the model can learn to predict not only the presence or absence of wind shear events but also the variability of the wind components. Incorporating variance as a target adds another dimension to the prediction task and provides valuable information about the intensity or magnitude of wind shear. Utilizing longer sequences instead of shorter sequences allows the model to capture more temporal dependencies and patterns in the wind data. Longer sequences provide a broader context and enable the model to understand the evolving behavior of the wind components over time. This can lead to improved prediction accuracy and the ability to detect wind shear events earlier or with greater precision.

Metrices	Version1	Version2	Version3	Version4
Recall	1.0	1.0	1.0	1.0
Precision	0.465	0.097	0.305	0.143
Accuracy	0.9989	0.9984	0.9972	0.9924

Table 1. Model evaluation from version 1, 2, 3, and 4

REFERENCES

1. Boille, A., Mahfouf, J., 2013. Wind shear over the Nice Cote d'Azur airport: Case studies. Nat. Hazards Earth Syst. Sci. 13, 2223–2238.
2. Chan, P.W., Hon, K.K., 2016. Performance of super high resolution numerical weather prediction model in forecasting terrain-disrupted airflow at the Hong Kong International Airport: case studies. Meteorology. Appl. 23, 101–114.
3. Liu, N., Kwong, K., Chan, P., 2012. Chaotic oscillatory-based neural network for wind shear and turbulence forecast with lidar data. IEEE Trans. Syst. Man Cybern. Part C, 42, 1412–1423.
4. Meng, L., Xu, J., Xiong, X., Ma, Y., Zhao, Y. 2018. A novel ramp method based on improved smoothing algorithm and second recognition for windshear detection using lidar. *Curr. Opt. Photonics*, *2*, 7–14.
5. Nechaj, P., Gaál, L., Bartok, J., Vorobyeva, O., Gera, M., Kelemen, M., Polishchuk, V., 2019. Monitoring of low-level wind shear by ground-based 3D lidar for increased flight safety, protection of human lives and health. Int. J. Environ. Res. Public Health 16, 4584.
6. Sadique, F., Sengupta, S., 2021. Modeling and Analyzing Attacker Behavior in IoT Botnet using Temporal Convolution Network (TCN) 2021, 1–22.
7. Shun, C.M., Chan, P.W., 2008. Applications of an infrared Doppler lidar in detection of wind shear. J. Atmos. Ocean. Technol. 25, 637–655.
8. Tai, Y., Yang, J., Liu, X., 2017. Image super-resolution via deep recursive residual network. Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017 2017-Janua, 2790–2798.

```
import torch
import torch.nn as nn
from torch.nn.utils import weight_norm


class TemporalBlock(nn.Module):
    """
    A class representing a temporal block used in Temporal Convolutional Networks (TCNs).

    Args:
    - n_inputs (int): the number of input channels
    - n_outputs (int): the number of output channels
    - kernel_size (int): the size of the convolutional kernel
    - stride (int): the stride used in convolution
    - dilation (int): the dilation used in convolution
    - padding (int): the padding used in convolution
    - dropout (float): the dropout probability

    Attributes:
    - conv1 (torch.nn.Conv2d): the first convolutional layer
    - pad (torch.nn.ZeroPad2d): the zero-padding layer
    - relu (torch.nn.ReLU): the activation function
    - dropout (torch.nn.Dropout): the dropout layer
    - conv2 (torch.nn.Conv2d): the second convolutional layer
    - net (torch.nn.Sequential): the sequence of layers composing the temporal block
```

```
    - downsample (torch.nn.Conv1d): the optional downsample layer
    - relu (torch.nn.ReLU): the activation function
    """

    def __init__(self, n_inputs, n_outputs, kernel_size, stride, dilation, padding,
dropout=0.2):
        super(TemporalBlock, self).__init__()
        self.conv1 = weight_norm(nn.Conv2d(n_inputs, n_outputs, (1, kernel_size),
                                           stride=stride, padding=0, dilation=dilation))
        self.pad = torch.nn.ZeroPad2d((padding, 0, 0, 0))
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout)
        self.conv2 = weight_norm(nn.Conv2d(n_outputs, n_outputs, (1, kernel_size),
                                           stride=stride, padding=0, dilation=dilation))
        self.net = nn.Sequential(self.pad, self.conv1, self.relu, self.dropout,
                                 self.pad, self.conv2, self.relu, self.dropout)
        self.downsample = nn.Conv1d(
            n_inputs, n_outputs, 1) if n_inputs != n_outputs else None
        self.relu = nn.ReLU()
        self.init_weights()

    def init_weights(self):
        """
        Initializes the weights of the convolutional layers using normal distribution with
mean 0 and standard
        deviation 0.01.
        """
```

```
        self.conv1.weight.data.normal_(0, 0.01)
        self.conv2.weight.data.normal_(0, 0.01)
        if self.downsample is not None:
            self.downsample.weight.data.normal_(0, 0.01)

    def forward(self, x):
        """
        Performs a forward pass through the temporal block.

        Args:
        - x (torch.Tensor): the input tensor of shape (batch_size, input_channels, sequence_length)

        Returns:
        - out (torch.Tensor): the output tensor of shape (batch_size, output_channels, sequence_length)
        """
        out = self.net(x.unsqueeze(2)).squeeze(2)
        res = x if self.downsample is None else self.downsample(x)
        return self.relu(out + res)


class TemporalConvNet(nn.Module):
    """
    The TemporalConvNet class is a PyTorch module that implements a temporal convolutional
    network for sequence forecasting tasks.
```

Args:

- num_inputs (int): The number of input features.

- num_channels (List[int]): A list of integers representing the number of channels in each convolutional layer.

- kernel_size (int, optional): The kernel size of the convolutional layers. Default is 2.

- dropout (float, optional): The dropout rate to apply after each convolutional layer. Default is 0.2.

Attributes:

- network (nn.Sequential): A PyTorch sequential module containing the TemporalBlock layers.

"""

```
def __init__(self, num_inputs, num_channels, input_length=60, output_length=1, kernel_size=2, dropout=0.2):
    super(TemporalConvNet, self).__init__()
    layers = []
    num_levels = len(num_channels)
    for i in range(num_levels):
        dilation_size = 2 ** i
        in_channels = num_inputs if i == 0 else num_channels[i-1]
        out_channels = num_channels[i]
        layers += [TemporalBlock(in_channels, out_channels, kernel_size, stride=1, dilation=dilation_size,
                                 padding=(kernel_size-1) * dilation_size, dropout=dropout)]

    self.network = nn.Sequential(*layers)
    self.fc = nn.Linear(input_length*num_channels[-1]+48, output_length)
```

```
def forward(self, x):
    mu = x.mean(2)
    sig2 = x.var(2)
    x = self.network(x)
    x = torch.cat([x.reshape(x.shape[0], -1), mu, sig2], axis=1)
    return self.fc(x).flatten()
```