Machine Learning-Based Orbit Estimation from the Seeing Effect

G. Dofri Vidarsson
Computer Science MSc. student
EPFL
Lausanne, Switzerland
gunnar.vidarsson@epfl.ch

***Abstract*—This is a report for a semester project that is a collaboration between CVLab, eSpace and LASTRO. The aim of this project was to design and evaluate machine learning models to estimate the angular velocity of uncataloged space objects in Earth orbit. Images of streaks left by these objects on astronomical observation, along with environmental metadata from the time of observation, were used to train various neural network architectures. Three types of models were considered: convolutional neural networks (CNNs), transformers, and a hybrid approach combining the strengths of both. Synthetic datasets were used to validate the models and training pipelines. When applied to real observational data, the models struggled to achieve meaningful results, likely due to the limited size of the dataset and the increased complexity of the problem compared to the synthetic datasets. These findings highlight the challenges associated with this task and underscore the need for larger datasets, while also demonstrating the potential of the models in controlled environments.**

I. Introduction

The space near Earth is becoming increasingly crowded, with small satellites and debris posing significant risks to space missions and satellite deployments. Cataloging the orbits of these objects is crucial for ensuring the safety of operational spacecraft. These objects appear as streaks in astronomical images taken from the ground, often crossing the entire image sensor during the typical exposure time of 30 - 300 seconds [1].

Ground-based observations are influenced by atmospheric turbulence, which distorts the light from these objects, a phenomenon known as the seeing effect. This distortion adds complexity to the streaks captured in astronomical images but might encode valuable information about the object's motion.

The central hypothesis of this project is that a neural network can infer the angular velocity of these objects by training on images of their streaks, combined with relevant weather data recorded at the time of observation. This approach seeks to leverage the information embedded within the distortions caused by the seeing effect to enable angular velocity estimation.

II. Methodology

A. Real Data and Data Processing

The dataset we used consists of images from the Omega-CAM imager of the VLT Survey Telescope at the Paranal Observatory [2]. The data is from a month long period, January 2022. It contained 213 individual observations containing streaks that had been identified and correlated with objects from a known database [3]. Each observation comprises image data from the 32-sensor array of the OmegaCAM imager, along with metadata from the telescope's instruments. Since objects frequently cross multiple sensors, their streaks appear on multiple images. For this project, each appearance of an object on a different sensor is treated as a separate streak. From here on out, when we refer to a "streak" we refer to the instance from a single object on a single sensor. In total, there were around 1000 of these streaks in the dataset.

The data is labeled with the start and end pixel coordinates of individual streaks on each sensor image along with the angular velocity of the object that caused the streak.

1) Image Data: Each sensor produces an image of the size of 4200 × 2144 pixels, with each pixel represented as a 16-bit unsigned integer. These are very large images for neural networks, which typically work with input sizes like 128 × 128 or 256 × 256 pixels. Given that the streak only exists in a narrow 10 to 30 pixel-wide neighborhood around a line across the image, most of the image data is not relevant to the task on hand.

To address this, we cropped the images around the streaks. For every streak, we rotated the image so that the streak was horizontal. The images were then cropped to a height of 50 pixels, a size chosen to be large enough to include all streaks while allowing leeway for data augmentation through vertical shifts. After preprocessing, we obtained a dataset of 1012 streak images, each with varying widths and a fixed height of 50 pixels. The models were designed to work with images with a height of 32 pixels so these images were cut further during training, possibly after a vertical shift. Figure 1 shows an example of a streak.

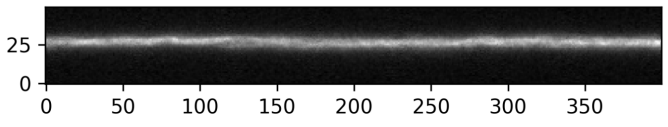

Fig. 1. Subsection of a cut streak

Even after cropping, the resulting images still contained mostly dark background pixels, while the streak pixels were

significantly brighter. This led to a highly skewed pixel intensity distribution, as shown in Fig. 2. The figure illustrates the pixel intensity distribution of the images before and after applying the normalization strategies described below:

1) **Log Scaling**: Pixel values were log-transformed to reduce their dynamic range to help with training stability.
2) **Standard Normalization**: The images were normalized to have zero mean and unit standard deviation. Care was taken to compute the mean and standard deviation using only the training set, and these values were applied to normalize the validation and test sets to prevent data leakage.

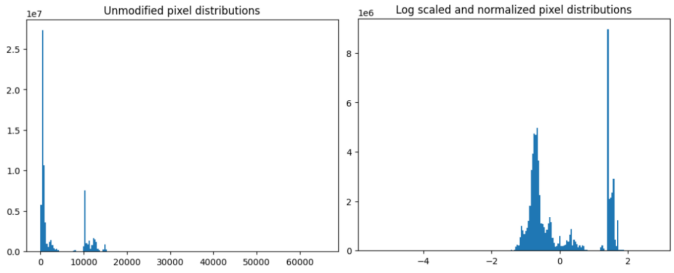

Fig. 2. Pixel intensity distributions of cropped streak images. The left image shows the distribution before normalization, and the right image shows the distribution after applying log scaling and standard normalization.

When training on the two synthetic datasets (described later in this section), both normalization methods were applied. For the real data, training was conducted using all possible combinations of the normalization methods: log scaling, standard normalization, both methods combined, and no normalization. The choice of normalization method did not significantly influence the results on the real dataset. The final results presented in the Results chapter were obtained using both log scaling and standard normalization, consistent with the approach used for the synthetic datasets.

2) Environmental Data: The FITS files included various environmental data points from the time of the observation. Seven were identified as potentially useful for understanding the atmospheric turbulence causing the seeing effect and were used for training. They are listed below by their headers in the FITS files:

- **ESO TEL AMBI IRSKY TEMP**: Temperature of the IR sky
- **HIERARCH ESO TEL AMBI TEMP**: Observatory ambient temperature
- **HIERARCH ESO TEL AMBI WINDSP**: Observatory ambient wind speed
- **HIERARCH ESO TEL AMBI PRES START**: Observatory ambient air pressure
- **HIERARCH ESO TEL AMBI FWHM END**: Observatory seeing queried from AS
- **HIERARCH ESO TEL AMBI RHUM**: Observatory ambient relative humidity
- **HIERARCH ESO TEL AMBI TAU0**: Average coherence time

Each data point was normalized to have zero mean and unit standard deviation for improved training stability, while ensuring no data leakage during this scaling process.

B. Targets

Along with the start and end image coordinates of each streak, the data provided for this project included the angular velocity of the identified object. That is the target for our regression models. They were normalized in the same fashion as the environmental data.

C. Synthetic Data

Two synthetic image datasets were created for validating that the models, data processing and training pipelines were without bugs and effective on a simpler task than the real data. The images and targets of these datasets were normalized and scaled in the same way as for the Real Streaks Dataset.

1) Sine Dataset: We wrote a method in python that generates images of horizontal sine waves. The image dimensions were chosen to 32 x 600 pixels. The frequency of each sine wave was randomized and used as the target for the regression task. 10000 such images were used for training This task is very simple and any half decent model should be able to learn it. This was only made to validate that there were no major bugs in the code. Figure 3 shows an example of a sine wave image.

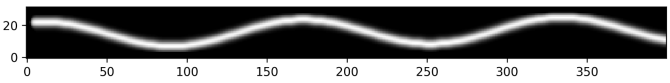

Fig. 3. Subsection of a sine image

2) Synthetic Streaks Dataset: Because the real dataset was relatively small and diverse, a synthetic dataset was created to bridge the gap between the trivial Sine Dataset and the more complex real streaks. This dataset represents the simplest version of a real streak: A uniform-brightness object on a noise-free black background. The pertubations on these streaks are modeled by a Moffat distribution which should make them resemble the seeing effect. This dataset was generated using a code provided by LASTRO and consisted of 5000 images of synthetic streaks, each with a size of 50 × 1000 pixels.

The only parameter that differed in the generation of each streak was the diffuse coefficient of the Moffat distribution. The diffuse coefficient indirectly correlates with the angular velocity of the object: slower objects result in a greater diffusion of light, as they create more spread-out streaks due to their longer exposure on the image sensor. The task for this dataset was to train models to predict the diffuse coefficient from the image of the streak, providing a simpler task for the models to learn compared to the real streaks dataset. Figure 4 shows an example of a synthetic streak.

D. Data Augmentation

To counteract the small size of our datasets and help prevent overfitting, some data augmentation methods were made available for training:

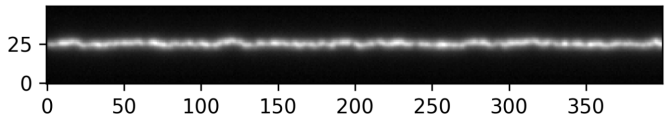

Fig. 4. Subsection of a synthetic streak

- **Vertical shifts**: A slight vertical translation was applied to simulate variations in streak positioning.
- **Flipping**: Images were flipped along one or both axes, as there is nothing inherently special about the orientation of the streaks.
- **Rotational shifts**: Small random rotations were applied to account for potential deviations in streak alignment.

These augmentations were designed to preserve the fundamental structure of the streaks while introducing minor variability, helping the models to avoid overfitting to specific patterns in the training data. However, in the end, only flipping was used for the training of the results presented in this report, as shifts and rotations appeared to confuse the models.

While the models (both CNNs and Transformers) were designed to handle varying input lengths—the CNNs through global average pooling and the Transformers through their inherent ability to process variable-length sequences—training was performed using fixed-length 600-pixel subsections of the streaks. This decision was motivated by the data augmentation potential offered by random subsection sampling.

During training, a random 600-pixel-wide subsection was extracted from each image in every iteration and passed to the model. Streaks shorter than 600 pixels were excluded from training. The width of 600 pixels was chosen as it is long enough for the model to capture meaningful features from the streak while being short enough to discard only a small portion of the data, approximately 4%.

E. Models

For this image regression task, a few models were considered. One is the classic model choice for image regression, the CNN (convolutional neural network). Another was a modified ViT (Vision transformer), a more recent development in the image processing world that has outperformed CNNs, especially given large training datasets. Third model was a hybrid of the two.

1) CNN: The implemented CNN is comprised of three convolutional layers with ReLU activations. The first layer applies 32 filters of size 5×21 with padding of 2×10, followed by a 2×2 max-pooling layer. The second layer utilizes 64 filters of the same dimensions and pooling. The third layer employs 128 filters of size 3×11 with padding of 1×5, also followed by pooling. A global average pooling layer reduces the spatial dimensions before a fully connected layer outputs the predicted angular velocity. The rectangular kernels were chosen to effectively capture the elongated features of streak images and the global average pooling was used to significantly reduce the model size to be trainable with the small datasets we are using.

2) Transformer: The idea of using a transformer came from the sequential nature of our data. The images are long and thin. Each vertical strip of pixel is essentially a snapshot of time. If there are long range relationships across the image that are useful for the regression task, a CNN which is inherently designed to capture local spatial features through convolutional filters, may struggle to fully leverage those relationships. Transformers on the other hand are particularly well-suited for this scenario due to their self-attention mechanism, which allows them to model dependencies between distant parts of the input effectively. Inspired by the ViT (Vision Transformer) [4] architecture we use a simple fully connected embedding layer to take the a vertical strip of pixels, 1 pixel wide, and embed it into a higher dimensional space. For our purposes, the strip was of length 32 (height of the image) and the embedding dimension was 128. These embedded vectors serve as tokens and form a sequence that is passed into a standard transformer encoder. The positional relationships are preserved by adding sinusoidal positional encodings to the tokens in the same way as in the original Transformer Paper [5]. After the sequence has been encoded by the transformer encoder, mean pooling is done across the encoded tokens and the result from that pooling is passed on to a regression head consisting of two fully connected layers with a ReLU activation function in between to output the predicted value.

3) Mixed Model: The third approach explored was a hybrid model that combines the strengths of both CNNs and Transformers. The motivation behind this design was to leverage the ability of CNNs to efficiently extract local spatial features and the capacity of Transformers to capture long-range dependencies across the input data. This hybrid architecture could allow a model to obtain a more comprehensive understanding of the image data, addressing both local patterns and global relationships effectively.

The model begins with a set of convolutional layers to process the input images. Depending on the configuration, one to three convolutional layers are applied, each followed by a ReLU activation and a 1×2 max-pooling layer. Rectangular kernels were used to capture the elongated features of the data, with a size of 5×11 and padding of 2×5. These layers progressively reduce the vertical dimension of the image while preserving the horizontal structure. By pooling only in the vertical direction, each horizontal point in the resulting feature maps retains information about the local features around that corresponding horizontal position in the original image. This transforms the image into a series of feature maps where each horizontal slice encodes the local patterns of the image at that specific point.

The feature maps are reshaped into vertical slices, which are then passed through a fully connected embedding layer to produce a sequence of tokens. These tokens are processed using the same transformer architecture described in the previous section, with sinusoidal positional encodings, multi-head self-attention, and feed-forward layers. As before, mean pooling is applied to the transformer output to produce a single vector representation, which is then passed to a regression head to

predict the target value.

By integrating the CNN and Transformer, this hybrid model benefits from the local feature extraction capabilities of CNNs and the global dependency modeling strengths of Transformers.

F. Integration of Environmental Data

When working with the dataset of real streaks, we needed to incorporate the 7 aforementioned environmental data points that accompany each streak. Two methods were explored to integrate this information into the models.

The first method, referred to as **External Integration**, was used with both the Transformer and CNN models. In this approach, the environmental data points were passed through a separate feed-forward network consisting of two fully connected layers with ReLU activations. The result from that network was then concatenated with the output from the Transformer or the CNN. The combined representation was subsequently passed through a final two-layer, fully connected regression head to predict the target value.

The second method, referred to as **Embedded Integration**, was applied to the Transformer and Mixed models. Here, each environmental data point was treated as an independent input token. A separate embedding layer was applied to each environmental data point, transforming it into a token of the same dimensionality as the image tokens. These numeric tokens were concatenated with the sequence of image tokens produced by the token embedding layer or CNN in the mixed model. The sinusoidal position encodings were then added to the sequence that was subsequently processed by the Transformer encoder. The output from the Transformer encoder was mean-pooled and passed to the regression head for final predictions.

This second approach tightly integrates the environmental data into the Transformer's sequence processing, enabling the model to learn relationships between numeric features and image-derived features directly within the attention mechanism.

G. Model Sizes

Care was taken to ensure that the models did not differ too much in size, although some variations were inevitable. Detailed listings of the trainable parameter counts for the models, both with and without numeric integration, can be found in Tables I and II.

TABLE I
TRAINABLE PARAMETER COUNTS FOR MODELS WITHOUT NUMERIC INTEGRATION

Model	Trainable Parameter Count
CNN	489089
Transformer	607361
Mixed 1L	615889
Mixed 2L	627991
Mixed 3L	657655

TABLE II
TRAINABLE PARAMETER COUNTS FOR MODELS WITH NUMERIC INTEGRATION

Model	Trainable Parameter Count
CNN - External Integration	500001
Transformer - External Integration	649601
Transformer - Embedded Integration	724737
Mixed 3L - Embedded Integration	775031

H. Training

Initially, a simple training loop was created, splitting the dataset into three subsets: training, validation, and testing. This approach provided a straightforward way to train and evaluate the models. However, after observing instability in training results and recognizing the small size of our datasets, we adopted k-fold cross-validation to ensure a more robust evaluation. In our case, we used $k = 6$.

In k-fold cross-validation, the dataset is split into k equally sized subsets, referred to as folds. Training is then conducted k individual times. Each time, one fold is used as the validation set, while the remaining $k-1$ folds are combined to form the training set. In each training run, a different fold serves as the validation set. To ensure consistency across runs, we initialized each fold's training with the same model weights, eliminating any variability that could arise from random weight initialization.

For small datasets like ours, a single train-test split can introduce variability, where the results may be overly optimistic or pessimistic depending on how the data is partitioned. By averaging the results across all k folds, k-fold cross-validation reduces the impact of such variability, providing a more reliable evaluation of the model's performance.

For all data splitting into training, validation, and test sets, whether using k-fold cross-validation or not, care was taken to ensure that all streaks from the same observation in the Real Streaks Dataset were assigned to the same set. This precaution was necessary because the environmental metadata is identical for all streaks within an observation, and any object appearing across multiple sensors in the same observation would generate multiple streaks. Allowing such streaks to be distributed across training, validation, or testing sets would result in a significant data leak, as the model could potentially learn from the shared numeric features or visual similarities rather than generalizing to unseen data.

The dataset splits used were 80% training, 10% validation, and 10% testing. This applies to the training on all three datasets. In cases where k-fold cross-validation was employed, the testing set was still 10%, and the rest was used for the k-fold cross-validation.

To optimize the models, the Adam optimizer was used with its default parameters and a fixed learning rate of 1×10^{-4}. The loss function used for all training was Mean Squared Error (MSE) loss, as it is well-suited for regression tasks. Training was conducted with a batch size of 8. During training, the validation loss at each epoch was monitored to assess the

model's performance. Early stopping was implemented, and training was halted if the validation loss did not improve over 5 consecutive epochs. This approach helped prevent overfitting and avoided unnecessary training of models that were not improving.

III. Results

This chapter presents the results of the experiments conducted to evaluate the performance of the different model architectures on various datasets. We first present the results of the training on the Sine Dataset, as it was a highly controlled and straightforward dataset designed to test the correctness of the models and training pipeline. Next, we discuss the results from training on the Synthetic Streak Dataset, which is a less trivial dataset to learn while still being simpler than the real data. Finally, we look at the model's performance on the Real Streaks Dataset.

Note that for each dataset, the targets were normalized during preprocessing to have a mean of zero and a standard deviation of one. Consequently, the reported Mean Squared Error (MSE) values reflect errors in the normalized space.

A. Results on the Sine Dataset

Three models were trained on the Sine Dataset: the CNN model, the Transformer model, and the Mixed model with three convolutional layers. The purpose of training on this dataset was to verify that there were no obvious bugs or other issues with the models or training pipeline. Any correct model should be able to learn this task. Unlike the other datasets, this task was trained using a simple train, validation, and test split, rather than k-fold cross-validation, given the simplicity of the task and the controlled nature of the data.

The results, measured in terms of Mean Squared Error (MSE), are summarized in Table III.

TABLE III
MSE Results on the Sine Dataset

Model	MSE
CNN	0.0051
Transformer	0.0178
Mixed	0.0081

All three models were able to learn the patterns in this dataset effectively, achieving low MSE values. The CNN model performed the best on this task, likely due to its ability to capture local patterns in images. The mixed model also performed well, although it is not obvious how much work the transformer part of that model is doing. It is plausible that this task is simple enough for the majority of the computation to be handled by the convolutional layers, with the transformer encoder primarily acting as a pass-through. The Transformer-only model performed slightly worse than the other two, suggesting that for a task dominated by localized features, the self-attention mechanism may not leverage its strengths as effectively as convolutional approaches.

B. Results on the Synthetic Streaks Dataset

Having validated the models and training pipelines on the Sine Dataset, additional experiments were conducted using the Synthetic Streaks Dataset. This dataset was used to further evaluate the models' ability to learn streak-like patterns under controlled conditions. As before, the CNN and Transformer models were trained, along with the three-layer Mixed model. In this set of experiments, the Mixed model was also trained in two-layer and one-layer variants to explore the impact of reducing the depth of the convolutional component.

This dataset is generated to model the streaks in its absolute simplest form. If the models struggle to learn this, they will likely struggle with the real streaks. After performing two runs of 6-fold cross-validation with these models, it was observed that training was quite unstable, with different random initializations leading to varying levels of performance. To address this variability, each 6-fold run was conducted 4 times for each model.

The results, measured in terms of Mean Squared Error (MSE), are summarized in Table IV. These values represent the average validation set performance across the 6-fold cross-validation runs. Unfortunately, the test set results were not preserved, meaning these validation results may present a slightly optimistic view of model performance.

TABLE IV
MSE Results on the Synthetic Streaks Dataset

Model	Run 1	Run 2	Run 3	Run 4	Mean
Transformer	0.195	0.256	0.205	0.194	0.212
CNN	0.078	0.114	0.109	0.111	0.103
Mixed-1L	0.166	0.111	0.214	0.176	0.167
Mixed-2L	0.179	0.212	0.154	0.097	0.161
Mixed-3L	0.067	0.088	0.168	0.086	0.102

The results reveal significant variability between the runs, but we do see some trends. The Transformer-only model performed the worst overall, with an average MSE of 0.212. This indicates that while strong local reasoning is more important than long-range dependencies for this task.

The CNN model consistently outperformed the Transformer, achieving an average MSE of 0.103. Its ability to capture local spatial patterns likely contributed to its strong performance. The Mixed models, which combine convolutional layers with a Transformer encoder, demonstrated varying degrees of success depending on the number of convolutional layers. The three-layer Mixed model performed best among the Mixed variants, with an average MSE of 0.102, nearly matching the CNN's performance. This suggests that the added transformer component did not significantly enhance performance on this dataset but did not hinder it either.

Interestingly, reducing the depth of the convolutional component in the Mixed models (e.g., to one or two layers) generally resulted in worse performance, underscoring the importance of the convolutional layers in extracting meaningful features from the data.

In summary, while the CNN model showed the most consistent performance on this dataset, the Mixed-3L model also performed well. However, the variability across runs highlights the need for careful evaluation and possibly further regularization to stabilize training.

C. Results on the Real Streaks Dataset

Four models were trained on the Real Streaks Dataset, using the different methods of incorporating numeric values described in the Methodology chapter. The models included the CNN with External Integration, the Transformer with External Integration and Embedded Integration and the Mixed model with Embedded Integration, each designed to handle numeric data in distinct ways, either by concatenating processed numeric embeddings at the end or by embedding and prepending numeric tokens to the image sequence.

Unlike the experiments conducted on the Synthetic Streaks Dataset, only a single 6-fold cross-validation run was performed for each model on the Real Streaks Dataset. This decision was made because the results from the initial runs were clearly nonsense, with the models failing to learn any meaningful patterns. The Mean Squared Error (MSE), as shown in Table V, was indistinguishable from random predictions, indicating that additional runs would not provide further insights.

TABLE V
MSE RESULTS ON THE REAL STREAKS DATASET

Model	MSE
CNN - External Integration	1.346
Transformer - External Integration	1.661
Mixed - Embedded Integration	1.380
Transformer - Embedded Integration	1.405

These results highlight significant challenges with the Real Streaks Dataset, including the limited size of the dataset and the increased noise in the images compared to the synthetic streaks. To fully understand the limitations here, a much larger dataset of real streaks would be necessary.

IV. DISCUSSION

The results demonstrate both the potential and limitations of using machine learning models to estimate angular velocities from streak images.

The models performed well on synthetic datasets, with CNNs and Mixed models showing strong capabilities. This success validated the architectures and training pipelines, confirming their functionality. However, these tasks were significantly simpler than the real dataset, and success on synthetic data does not guarantee the feasibility of solving the real-world problem.

Future work with synthetic data should focus on increasing complexity by varying parameters to create more realistic and diverse streaks. Success on these advanced synthetic datasets could lead to exploring transfer learning to improve performance on real data.

The most critical step forward is acquiring a larger dataset of real streaks. A dataset one or two orders of magnitude larger than the current one could provide the necessary diversity and volume to enable meaningful learning and significantly improve model performance.

REFERENCES

[1] S. Hellmich, E. Rachith, B. Yu Irureta-Goyena Chang, and J.-P. Kneib, "Harvesting large astronomical data archives for space debris observations," in *2nd NEO and Debris Detection Conference*, Jan. 2023, p. 75.

[2] E. Deul, K. Kuijken, and E. Valentijn, "OmegaCAM: the 16k × 16k Survey Camera for the VLT Survey Telescope," Proc. SPIE, vol. 4844, 2002. doi:10.1117/12.456694.

[3] E. Rachith, S. Hellmich, and B. Y. Irureta-Goyena, "A Novel Machine Learning Based Algorithm for Efficient Streak Detection," in *2nd International Orbital Debris Conference*, vol. 2852, LPI Contributions, Dec. 2023, p. 6132.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," arXiv preprint arXiv:2010.11929, 2021.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," arXiv preprint arXiv:1706.03762, 2023.