– Independent Work Report Spring 2022 –

Convolutional Transformers for Inertial Navigation

Raymond Liu
Adviser: Olga Russakovsky

Abstract

The task of inertial navigation involves estimating a body's trajectory using IMU sensor measurements. It is inherently a difficult task, as small amounts of noise in these measurements can accumulate into large errors in position estimation over time. Applications of neural network architectures to this task have resulted in highly accurate results. The current state-of-the-art model is a 1-dimensional version of ResNet-18, a convolutional neural network (CNN) architecture [8]. Recently, transformers have shown improved performance over convolutional architectures on a variety of tasks, such as machine translation and image classification [15, 5]. This paper introduces data augmentation methods and several transformer-based architectures for inertial navigation that achieve more accurate results over existing convolutional architectures. Specifically, we first show that a baseline pure-transformer model outperforms a baseline ResNet-18 model. Then, we introduce transformer models that incorporate convolutional layers to extract additional information, further improving performance.

1. Motivation and Goal

An Inertial Measurement Unit (IMU) combines an accelerometer and a gyroscope to measure the movement of a body. An accelerometer measures a body's linear acceleration in 3 dimensions, and a gyroscope measures a body's angular velocity in 3 dimensions, totalling 6 degrees of freedom. Measurements from IMU sensors are commonly used in inertial navigation systems to estimate the position and orientation of a moving body.

In theory, one can calculate the exact position of a body relative to its initial starting point by integrating the measured accelerations from an IMU to get velocities, then integrating again to get positions. However, in practice, after double integration, small amounts of noise in these

measurements result in errors in position estimation that grow quadratically with time [13]. As a result, IMUs are typically used to supplement other navigation systems, such as GPS or visual odometry.

However, there is much motivation for the use of IMUs as a standalone method of predicting an object's position. IMUs are inexpensive and ubiquitous, and can be found in every smartphone today. IMUs are extremely energy-efficient, and can run 24 hours a day with negligible battery usage. IMUs are robust and capable of functioning anywhere, including indoors (unlike GPS), as well as inside of a bag or a pocket (unlike visual odometry). Lastly, IMUs are used in a wide variety of applications, such as navigation for robots [11] and pedestrians [7], as well as augmented reality systems such as Google ARCore and Apple ARKit [4]. Therefore, developing a system that can accurately estimate a body's position using only IMU data would be extremely valuable for a number of important applications.

Several different methods have been developed to deal with this problem. Recently, a 1-dimensional variant of the ResNet-18 CNN architecture has achieved state-of-the-art performance on position estimation using only IMU sensor measurements [17].

The goal of this work is to introduce transformer-based models that outperform the existing best models. In this paper, we first implement and evaluate a baseline transformer model and find that it performs significantly better than a baseline ResNet model. Additionally, we implement and evaluate three improved transformer models that incorporate convolutional layers to extract additional information, and we find that two of them outperform the baseline transformer model.

2. Related Work

Several different methods have been previously proposed to handle the task of inertial navigation.

One method proposed in 2015 is pedestrian dead reckoning (PDR), which involves detecting footsteps, estimating the step length, and updating the location at every step [14]. Figure 1 illustrates the PDR architecture. This method works well in cases with simple motions. For example, PDR achieves extremely accurate performance on the RIDI and OxIOD datasets; due to the limited way

these datasets were collected, they contain less complex and less diverse movement cases - see Section 6.1 for more details. PDR fails in more complex use cases, as well as in scenarios where steps can't be detected, e.g. when the device is placed on a shopping cart.

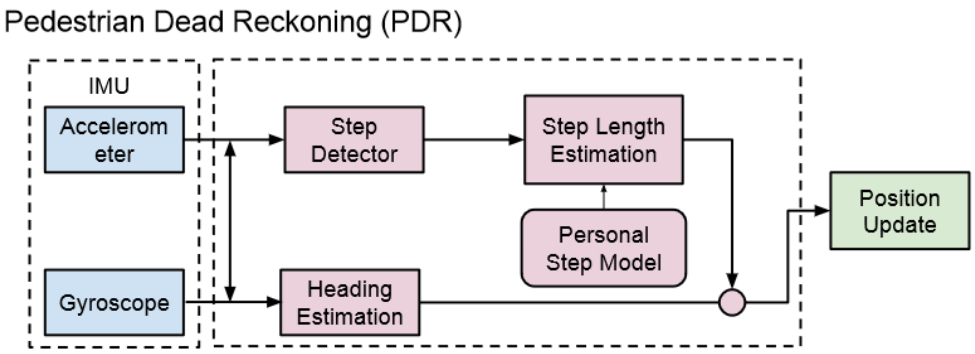

Figure 1: The PDR architecture (image from Chen et al. [3]).

The first machine learning based approach towards inertial navigation was robust IMU double integration (RIDI), proposed in late 2017 [18]. RIDI assumes that a phone is either in a leg pocket, in a bag, hand-held, or on body. Therefore, RIDI uses an SVM to classify the phone placement into one of these four types, then applies an SVR based on this classification to regress a 2D velocity vector, and finally integrates regressed velocities to obtain positions. Figure 2 illustrates this architecture. RIDI is more robust to complex motion cases, such as walking backwards, where PDR would predict a body to be moving in the complete opposite direction of where it is actually moving [18]. However, it is not entirely robust to all use cases, as it only covers four possible placement types.

IONet, a neural network based approach proposed in early 2018, uses a 2-layer bidirectional LSTM to regress the velocity and orientation of a body [3]. IONet achieves significantly more accurate position estimations than PDR on its accompanying dataset; however, since it integrates orientation angle differences to calculate the device's orientation, small errors accumulate over time, leading to inaccurate orientation estimations and thus inaccurate position estimations [3, 17].

Recently, the RoNIN project introduced a 1-dimensional variant of the ResNet-18 architecture; RoNIN ResNet consistently outperforms the previously mentioned methods for inertial navigation [17]. RoNIN ResNet is the baseline model that we will compare against other models that we

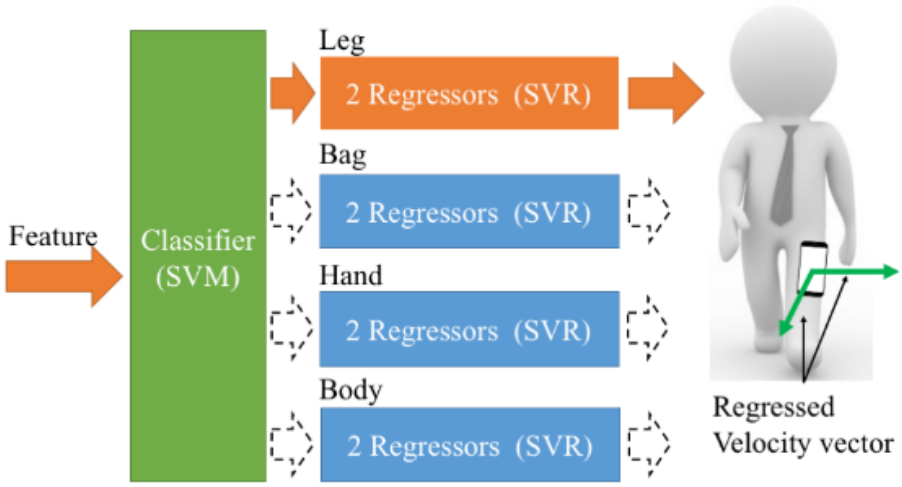

Figure 2: The RIDI architecture (image from Yan et al. [18]).

propose in this work. It achieves top performance on the RoNIN dataset, which we use for this work. Details on the RoNIN dataset are covered in Section 6.1.

3. Approach

Introduced in 2017, the transformer architecture is highly parallelizable and achieves better performance on machine translation tasks than previous state-of-the-art recurrent and convolutional architectures [15]. Transformers are similarly effective in many other tasks such as image classification [5] and object detection [2].

Following the successful use of transformers in a variety of tasks, we develop transformer architectures for the task of inertial navigation. We first adapt a vision transformer (ViT) [5] to use as a baseline pure-transformer model. The original use of ViT was for image classification. We reformulate the task of image classification - in this case, regressing two class scores from an image - into a task of regressing a 2D velocity vector from a sequence of IMU sensor measurements.

Transformer performance can be improved with the addition of convolutions [2, 6, 16]. Particularly for the task of inertial navigation, introducing convolutions could be beneficial for processing sequences of IMU data, as the ability for convolutions to exploit local dependencies complements the ability for attention to model global, long-distance dependencies [6, 16].

We therefore take three broad approaches to integrating convolutions into our baseline transformer model:

1. Extract local information from the input using convolution layers before passing it to the transformer
2. Apply multiple convolutions sequentially before the transformer
3. Apply multiple convolutions in parallel to the transformer and concatenate the results

4. Model Architectures

The best-performing architectures are described in this section. Variants of these architectures are discussed in Section 7 - Evaluation.

4.1. RoNIN ResNet

ResNet is a convolutional neural network architecture that includes skip connections to address the degradation problem, where accuracy begins to decrease as networks become too deep [8]. This allowed ResNet to achieve state-of-the-art results on image classification tasks.

RoNIN ResNet is a modified 1-dimensional version of ResNet-18. The input at each frame i has dimension 6×200; IMU data at each frame consists of 3 accelerometer measurements and 3 gyroscope measurements, and data is taken from frame $i - 200$ to i. The model regresses a 2D vector that represents the velocity at frame i.

Below we describe the architecture of RoNIN ResNet in detail. Note that the inputs to each convolutional layer are padded according to the kernel size to maintain the original input lengths, or to result in the input length being exactly halved when using a stride of 2. Additionally, batch normalization is applied to the output of each convolutional layer.

The input block of RoNIN ResNet consists of, in order:

1. 1D convolution, kernel size = 7, stride = 2
2. ReLU activation
3. 1D max pool, kernel size = 3, stride = 2

The convolution in this input block increases the number of channels from 6 to 64, while the use of strides decreases the size of each channel from 200 to 50.

Next, there are 8 building blocks, each of which utilizes skip connections. These building blocks gradually increase the number of channels from 64 to 512, and decrease the size of the channels from 50 to 7. The general architecture of this block is illustrated in Figure 3. Each block consists of, in order:

1. 1D convolution, kernel size = 3, stride = 1 or stride = 2
2. ReLU activation
3. 1D convolution, kernel size = 3, stride = 1
4. Skip connection from the original input to the current tensor [1]
5. ReLU activation

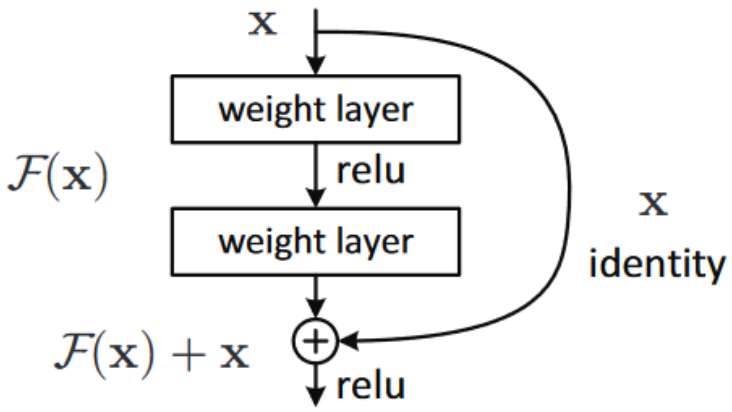

Figure 3: ResNet building blocks (image from He et al. [8]).

Finally, the fully connected output block consists of an 896×512 linear layer with ReLU activation, a 512×512 linear layer with ReLU activation, and finally a 512×2 linear layer to output a 2D vector.

4.2. Baseline Transformer

The baseline transformer model is based on the vision transformer [5], adapted to regress 2D velocity vectors rather than class scores.

[1] For blocks where the dimension of the input is changed via increasing the number of channels and using stride = 2, the identity of the input x is scaled accordingly to match the new dimension.

The 6×200 inputs are split into 6 vectors, each of size 1×200. A learnable classification token is concatenated to the input, then a learnable positional embedding is added to the result. Then the 7 vectors are used as input tokens to a transformer encoder. The output is passed through an MLP layer to output a 2D vector for the velocity. This architecture is illustrated in Figure 4.

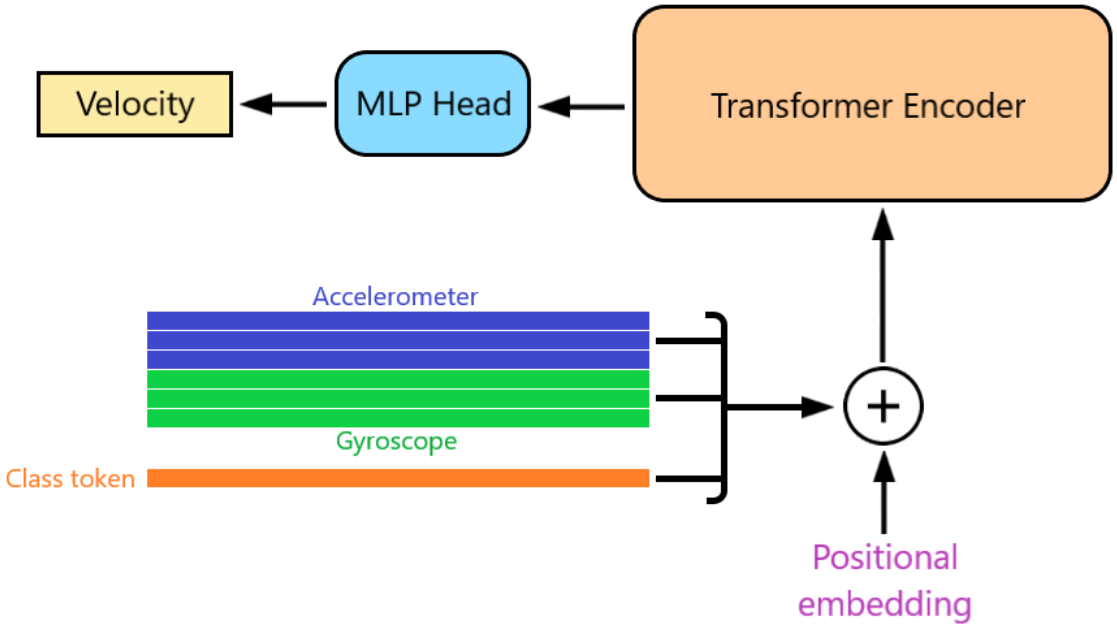

Figure 4: Baseline transformer architecture. The input is split into 6 input tokens, each of length 200, along with a learnable class token. A learnable positional embedding is added, and the result is passed through 6 multi-head self-attention units that comprise the transformer encoder. Finally, the result is passed through an MLP layer to obtain a 2D output vector.

4.2.1. Transformer encoder

The transformer encoder acts the same as the original transformer proposed by Vaswani et al [15]. Our transformer encoder consists of 6 encoder units. Each encoder unit consists of a multi-head self-attention unit (described in section 4.2.2) followed by a feedforward unit. A skip connection [8] and layer normalization [1] is used on the output of the self-attention unit, as well as the feedforward unit. The encoder unit is illustrated in Figure 5.

4.2.2. Multi-Head Self-Attention Unit

The multi-head self-attention unit calculates the self-attention of the input $X \in \mathbf{R}^{n \times d}$ (in this case, $n = 7$ and $d = 200$), simultaneously across h heads. We use $h = 16$ heads for both the baseline

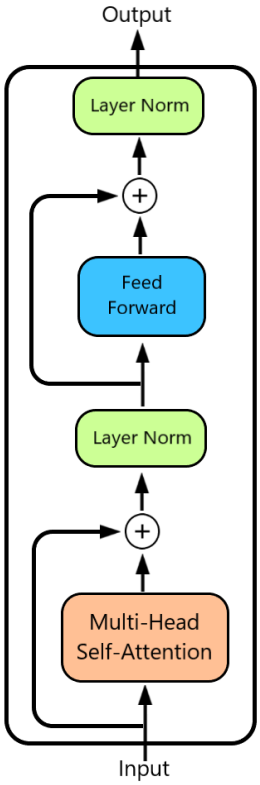

Figure 5: A transformer encoder unit. The full encoder consists of 6 consecutive encoder units.

and all transformer variants. The dimensionality of each head is $d_h = 64$ for the baseline and all variants.

For each head i, X is projected using a linear layer $W_i^q, W_i^k, W_i^v \in \mathbf{R}^{d \times d_h}$ without bias to create q_i, k_i, and v_i, i.e. the queries, keys, and values, each of which are $\mathbf{R}^{n \times d_h}$:

$$q_i = XW_i^q, \quad k_i = XW_i^k, \quad v_i = XW_i^v \tag{1}$$

Self-attention for each head i has dimension $\mathbf{R}^{n \times d_h}$ and is calculated as follows [15]:

$$S_i = \text{softmax}\left(\frac{q_i k_i^\top}{\sqrt{d_h}}\right) v_i \tag{2}$$

Each S_i is finally passed through an MLP layer $W_i^m \in \mathbf{R}^{d_h \times d}$. The final output has dimension $n \times d$.

4.2.3. Feedforward Unit

The feedforward unit consists of a linear layer from d to d_{hidden}, GELU activation [9], and a

linear layer from d_{hidden} to d, where $d = 200$ and $d_{\text{hidden}} = 1024$. The input and output of this unit both have dimension $n \times d$.

4.3. Convolutional Transformers

The first convolutional transformer architecture applies a single convolutional layer with a kernel size of 3 and a stride of 1 to the accelerometer data, as well as a single convolutional layer with a kernel size of 3 and a stride of 1 to the gyroscope data, then uses the outputs of those layers as inputs to the transformer alongside the original inputs. The architecture is shown in Figure 6.

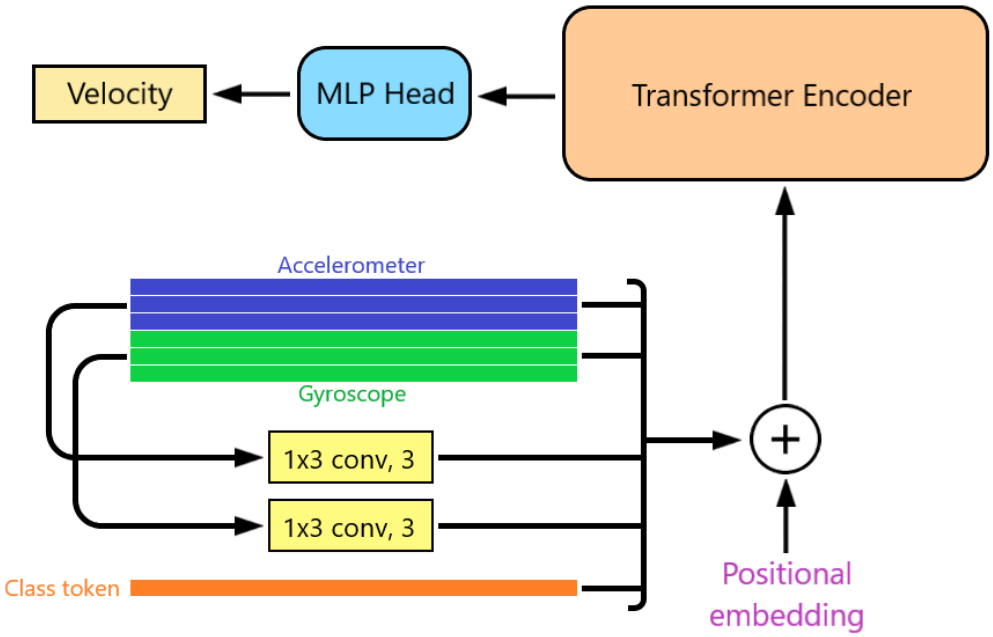

Figure 6: First convolutional transformer architecture. The only change from the baseline is to the input block.

The second convolutional transformer applies multiple convolutional layers to the input before passing it to the transformer. The architecture of these convolutional layers is based on the beginning of RoNIN ResNet - these layers increase the number of channels from 6 to 256, and decrease the size of each channel from 200 to 7. The output has dimensions 7×256, meaning the transformer takes in 8 token vectors, each of dimension 256.

This architecture is illustrated in Figure 7. Batch normalization and ReLU activation is applied to

the output of every convolutional layer, and a skip connection is used over every two convolutional layers after the maxpool layer.

The third convolutional transformer applies the same convolutional layers in parallel to the transformer, then concatenates the results from the convolutions and the transformer afterwards.

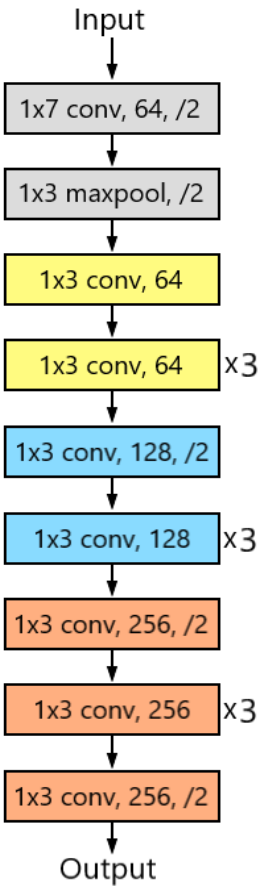

Figure 7: CNN architecture for the second and third convolutional transformers. The text on each layer indicates [kernel size], [layer type], [number of channels] for conv layers, and optionally, "/2" means using a stride of 2.

5. Additional Modifications

We take two additional steps to improve model performance. First, we apply Gaussian smoothing with $\sigma = 1.0$ on each of the 6 IMU channels. This involves convolving a filter (Figure 8) over the data.

The filter contains values drawn from the 1D Gaussian distribution, where x represents the

0.004	0.054	0.242	0.399	0.242	0.054	0.004

Figure 8: The filter applied to each channel of IMU data when $\sigma = 1.0$.

distance from the center of the filter:

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{3}$$

Figure 9 shows a visualization of the effect of this smoothing when applied to 100 frames (equivalent to half of a second) of data from one axis of gyroscope data.

Figure 9: Visualizing the effect of Gaussian smoothing with $\sigma = 1.0$ on one gyroscope channel.

For the second modification, we parameterize the skip connections. Rather than simply adding x (the original output) with $F(x)$ (the output after several layers), we instead multiply x by a learnable parameter α, and multiply $F(x)$ by $2 - \alpha$.

We evaluate these two methods and variations of them on RoNIN ResNet (Section 7 - Evaluation)

and use them as defaults in all transformer models. We test the effects of these methods on the best-performing transformer model in an ablation study in Section 7.4.

6. Implementation

6.1. Dataset

There are three inertial navigation datasets available: the RoNIN dataset [17], the RIDI dataset [18], and the OxIOD dataset [3].

The RIDI and OxIOD datasets both used one device to capture both IMU data and ground truth motion; since ground truth motion was captured using a visual odometry system, the device needed to have have line-of-sight to the system at all times, which prohibited natural movement, particularly for scenarios where the device was placed in a pocket or a bag. Additionally, the RIDI and OxIOD datasets each used a few specific device placements, such as in a leg pocket, in a bag, hand-held, on a body, or on a trolley.

On the other hand, the RoNIN dataset was collected using two devices - one for IMU data, and the other for ground truth motion. This allowed the IMU device to move without any constraints [17]. There were also no specific device placements to follow.

Because the RoNIN inertial navigation dataset contains the most extensive, diverse set of naturally complex motions, we use the RoNIN dataset for training and evaluation.

The publicly available [2] RoNIN dataset contains 152 data sequences, each lasting between 4 and 10 minutes, totaling roughly 19.8 hours of IMU data. 100 human subjects were used to collect this data; these subjects are split into two groups, with the first group containing 85 subjects and the second group containing 15 subjects. Data sequences collected from subjects in the first group are split into training, validation, and testing subsets, which contain 70, 15, and 35 data sequences, respectively. The second group of 15 subjects contains 32 sequences in total; this group is used to evaluate how well a model generalizes to data collected from unseen subjects.

[2] Due to security concerns, half of the RoNIN dataset is withheld. This still leaves more data than previous datasets - RIDI contains 2.5 hours of data, and OxIOD contains 14.7 hours of data.

6.2. Training

All models were implemented using PyTorch [12] and trained using an NVIDIA Tesla P100 with 16GB GPU memory.

The RoNIN dataset contains IMU sensor data recorded at 200Hz. During training, one training/validation sample is extracted every 10 frames. Each sample consists of the past 200 frames of IMU data. This results in a total of 924695 training/validation samples. Loss is calculated as the MSE loss between the predicted velocity and the ground truth velocity.

For all models, we use the ADAM optimizer [10] with a batch size of 128 and an initial learning rate of 0.0001. During training, if the validation loss does not decrease after 10 epochs, the learning rate is multiplied by 0.1.

For RoNIN ResNet, dropout is applied to the linear layers in the final fully-connected module with keep probability 0.5. RoNIN ResNet converges after 100 epochs, taking 5.2 hours.

For all transformer models, dropout is applied to all linear layers with keep probability 0.8. The baseline transformer model converges after 80 epochs, taking 3.5 hours.

7. Evaluation

7.1. Metrics

Relative positions are reconstructed from predicted velocities as follows:

$$s(t) = s(t-1) + v(t-1)dt \tag{4}$$

Where s is the position, v is the velocity, and $dt = 0.005$, as IMU data is sampled at 200Hz, and during testing, we regress the velocity at every frame.

Using these reconstructed positions, we will be evaluating different models using two standard metrics used in [17]:

- Absolute Trajectory Error (ATE): the root mean squared error between the ground truth trajectory and the estimated trajectory.

- Relative Trajectory Error (RTE): the average root mean squared error over fixed time intervals of 1 minute.

Our goal is to minimize the absolute trajectory error and relative trajectory error on the unseen subject testing set. Specifically, we want to show that the baseline transformer model achieves lower ATE and RTE than the baseline ResNet model, and that the convolutional transformers achieve lower ATE and RTE than the baseline transformer.

While the RoNIN paper contains evaluations for baseline models, we have trained our own baseline models using the publicly available data, because half of the dataset is withheld due to security concerns, as mentioned in Section 6.1 (Dataset).

7.2. Results

Results are shown in Table 1. ATE and RTE are reported in meters.

The baseline transformer achieves significantly lower ATE and RTE than baseline ResNet. The first convolutional transformer, which applies a convolution to the input and uses the result alongside the original input, further reduces ATE and RTE compared to the baseline transformer. This aligns with what we believed would happen, as we are simply extracting additional information on relationships between local values, which should be a direct improvement on the baseline transformer.

The second convolutional transformer, which applies multiple convolutional layers sequentially before the transformer, reduces ATE and RTE even further. This turns out to be the best-performing model of all models we trained.

The third convolutional transformer, which applies the same convolutional layers in parallel to the transformer, performs worse than the baseline transformer. We believe that this is because, unlike the previous improvements, in this case the convolutions and the transformer don't actually supplement one another and instead are working separately towards the same goal.

In addition to the main architectures, this table contains the results of several variations of the main architectures, which we explain here.

Model	ATE	RTE
Baseline ResNet	5.61	4.49
Smoothing with $\sigma = 0.5$	5.23	4.38
Smoothing with $\sigma = 1$	5.21	4.39
Parameterized Skip Connections	5.29	4.43
Baseline Transformer	5.12	4.40
Dropout 0.1	5.38	4.49
20-frame patches	5.64	4.59
25-frame patches	5.08	4.60
α and $1-\alpha$ for parameterized skips	5.46	4.36
Conv Stacked with Input	5.03	4.31
1x7 kernel size	6.48	4.81
1x25 kernel size	5.87	4.73
CNN Sequential to Transformer	4.99	4.24
Conv-Maxpool	5.47	4.88
Conv-Maxpool-3×Conv	5.19	4.68
CNN Parallel to Transformer	5.26	4.49
Conv-Maxpool-5×Conv	5.78	4.76
Conv-Maxpool-5×Conv (128 channels wide)	5.59	4.86
Conv-Maxpool-5×Conv (24 output channels)	5.51	4.67

Table 1: Model performance on data from unseen test subjects.

For baseline ResNet, we tested applying 1D Gaussian smoothing with $\sigma = 1.0$ and with $\sigma = 0.5$. We also separately tested using parameterized skip connections. Both of these methods (explained in Section 5) reduce ATE and RTE.

For the baseline transformer, we use a dropout rate of 0.2, but additionally tested using a dropout rate of 0.1. We also tried splitting the input into sequences of 20 or 25 frames to use as tokens, rather than using each IMU channel as a token. For 20-frame patches, we split the 6×200 input every 20 frames, resulting in 10 patches, each of size 6×20, which are then flattened to 1×120. Similarly, for 25-frame patches, we split the input every 25 frames, resulting in 8 patches, each of size 150. Lastly, rather than using α and $2-\alpha$ as weights for the parameterized skip connections, we tried using α and $1-\alpha$ - this change significantly degraded performance. We believe this is

caused by the transformer encoder using 12 consecutive skip connections (two skips for each of the six consecutive encoder layers) without normalization, which leads to the output being significantly smaller than before.

For the first convolutional transformer, we tested using different kernel sizes for the convolution, namely a 1×7 kernel and a 1×25 kernel, rather than the original 1×3 kernel. The wider kernels perform worse, although this could be a result of using too few convolutional channels - we used three channels for both of the two convolutional layers that were applied to the accelerometer and gyroscope data. However, using too many channels significantly increases transformer size and training time.

We also tried using simpler convolutional architectures for the second convolutional transformer. The first variation consists of a convolutional layer followed by a max pooling layer, with outputs of size 16×50. The second variation consists of a conv layer, a maxpool layer, and three more conv layers, each containing 64 channels - the number of channels is downsized to 16 at the end, and the output has size 16×50. The second variation performs better than the first variation, since it simply adds more layers. Similar to the more complex version, these variations apply ReLU activation after each convolutional layer and utilize skip connections. The original sequential convolutional transformer performs better than both of these variations, which is to be expected, given its increased depth and width.

For the third convolutional transformer, we similarly tested different convolutional architectures. The first variation consists of a conv layer, a maxpool layer, and five more conv layers, each containing 64 channels - at the end, the number of channels is downsized to 16, making the output have size 16×50. The second variation is the same as the first, but with the number of channels expanded from 64 to 128 in later layers. The third variation expands the output size to 24×50, rather than 16×50. The second and third variations, being wider than the first variation, both perform better with respect to ATE, although the second variation has increased RTE. However, the original parallel convolutional transformer performs better than all of these variations, which we expected, given its increased depth and width.

7.3. Visualizations

Figure 10 shows selected visualizations (labeled "Plot 1" through "Plot 4") of four ground truth trajectories from the testing set, along with the predicted trajectories from ResNet, the baseline transformer, and the sequential convolutional transformer (our best-performing model).

Plot 1 and Plot 2 demonstrate cases where the baseline transformer clearly performs worse than ResNet, but with the addition of convolutions, the convolutional transformer is able to perform on par with ResNet.

Plot 3 and Plot 4 demonstrate cases where the convoluutional transformer performs better than both ResNet and the baseline transformer, showing how it can go beyond what either a CNN or a transformer-only model can do. The trajectory in Plot 4 appears to be particularly difficult for all models to perform well on, but the convolutional transformer's predicted trajectory still makes significant improvements over the other models.

7.4. Ablation Study

All of our transformer models utilize parameterized skip connections and apply Gaussian smoothing with $\sigma = 1.0$ to each channel of the IMU data. We perform an ablation study to analyze how the absence of these methods affects the performance of our best-performing transformer model, which uses multiple convolution layers sequentially before the transformer. The results of our ablation study are shown in Table 2.

Smoothing the data and parameterizing the skip connections both result in lower ATE and RTE. Smoothing the data appears to have a slightly larger impact than parameterizing the skip connections, as it decreases RTE by a fair amount. ATE is significantly lowered when both methods are combined.

These findings are consistent with the results from applying these methods to RoNIN ResNet (shown in Table 1) - in particular, smoothing has a larger impact on ResNet than parameterized skip connections.

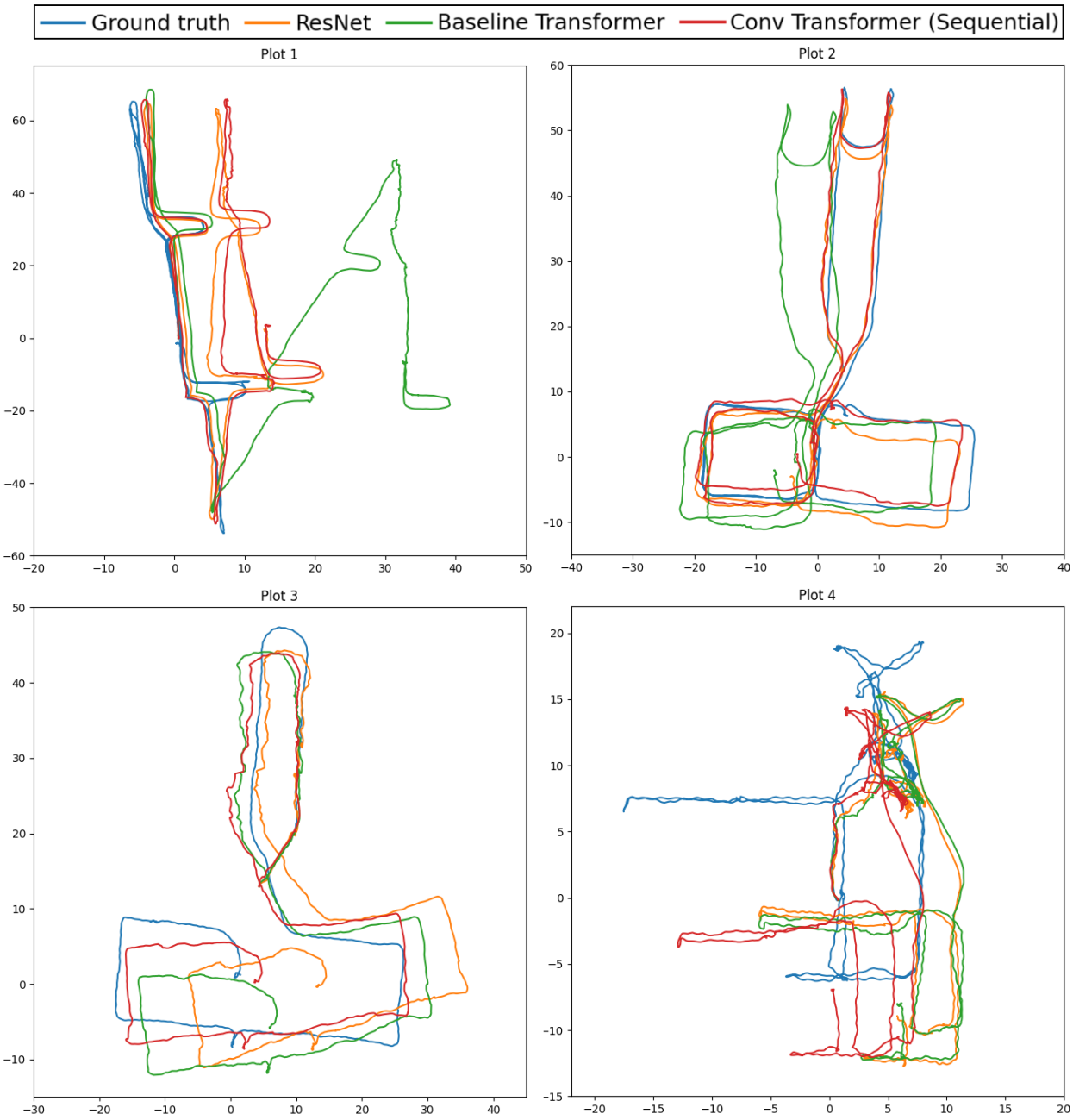

Figure 10: Selected visualizations of ground truth trajectories alongside predicted trajectories from three different models.

8. Conclusions

In this paper, we explore the application of transformers and convolutional transformers to inertial navigation and attain state-of-the-art performance on a diverse set of complex motion tasks.

We find that transformers work well with IMU data and perform better than previous approaches

Model	ATE	RTE
CNN Sequential to Transformer	4.99	4.24
Only Parameterized Skips	5.12	4.26
Only Smoothing	5.13	4.20
Neither	5.17	4.28

Table 2: Ablation study on best-performing transformer model.

for the task of inertial navigation, including CNNs such as ResNet, and by extension, recurrent networks such as LSTM. We additionally find that smoothing the data, parameterizing the skip connections, and incorporating convolutions into transformers to extract additional information can further improve performance.

The findings presented here open up a new pathway of research into applications of deep learning and transformers for inertial navigation.

8.1. Limitations and Future Work

As mentioned earlier, roughly half of the original RoNIN dataset is withheld for security reasons, which limits the effectiveness of our models.

Additionally, there are many other possible methods and variations of transformers besides the ones we implemented. There are undoubtedly variations that are even more effective than the ones we have tried, which could be a topic for future work.

Lastly, magnetometers, which measure the orientation of a device in three dimensions, are becoming increasingly common in IMUs. Currently, magnetometer readings are often distorted indoors and thus too noisy to use. A possible path for future work is to develop a method that can leverage noisy magnetometer data to further improve results.

9. Acknowledgements

I would like to thank my adviser, Prof. Olga Russakovsky, for her continued support over the course of this project. I would also like to thank the authors of the RoNIN project for providing code for their ResNet implementation, as well as their IMU dataset.

References

[1] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016. Available: https://arxiv.org/abs/1607.06450

[2] N. Carion *et al.*, "End-to-end object detection with transformers," 2020. Available: https://arxiv.org/abs/2005.12872

[3] C. Chen *et al.*, "Ionet: Learning to cure the curse of drift in inertial odometry," *CoRR*, vol. abs/1802.02209, 2018. Available: http://arxiv.org/abs/1802.02209

[4] S. Cortes *et al.*, "Advio: An authentic dataset for visual-inertial odometry," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[5] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," 2020. Available: https://arxiv.org/abs/2010.11929

[6] A. Gulati *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," 2020. Available: https://arxiv.org/abs/2005.08100

[7] R. Harle, "A survey of indoor inertial positioning systems for pedestrians," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.

[8] K. He *et al.*, "Deep residual learning for image recognition," 2015. Available: https://arxiv.org/abs/1512.03385

[9] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2016. Available: https://arxiv.org/abs/1606.08415

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. Available: https://arxiv.org/abs/1412.6980

[11] S. Leutenegger *et al.*, "Keyframe-based visual-inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, 02 2014.

[12] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," 2019. Available: https://arxiv.org/abs/1912.01703

[13] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, ser. Springer Handbook of Robotics. Springer Berlin Heidelberg, 2008, p. 484. Available: https://books.google.com/books?id=Xpgi5gSuBxsC

[14] Q. Tian *et al.*, "An enhanced pedestrian dead reckoning approach for pedestrian tracking using smartphones," in *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2015, pp. 1–6.

[15] A. Vaswani *et al.*, "Attention is all you need," 2017. Available: https://arxiv.org/abs/1706.03762

[16] Z. Wu *et al.*, "Lite transformer with long-short range attention," 2020. Available: https://arxiv.org/abs/2004.11886

[17] H. Yan, S. Herath, and Y. Furukawa, "Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, and new methods," *CoRR*, vol. abs/1905.12853, 2019. Available: http://arxiv.org/abs/1905.12853

[18] H. Yan, Q. Shan, and Y. Furukawa, "RIDI: robust IMU double integration," *CoRR*, vol. abs/1712.09004, 2017. Available: http://arxiv.org/abs/1712.09004