Path-Planning and Collision Avoidance: Neural Network Approach

Mahdis Rabbani
University of California, Davis
mrabbani@ucdavis.edu

***Abstract*—This paper presents a study on path planning and collision avoidance using a neural network approach. Inspired by two other papers with similar concepts, the author investigates different CNN architectures, including custom CNN structures and U-Net with different backbones, and evaluates their performance.**

***Index Terms*—Path-planning, Collision Avoidance, Neural Network**

I. Introduction

Path planning is the problem of finding a collision-free path from a start point to a target. It is considered one of the most fundamental problems in robotic research, allocating many resources and energy. Various methods have been developed to address the path planning problem. These methods can be classified into main categories: (1) Grid-based, (2) Sample-based, (3) Artificial Potential Field, (4) Learning-based, and (5) Model-based approaches.

In Grid-based methods, the environment is defined in a graph form with the locations of the environment as nodes and the feasible movements from one node to another as the edges [1]. The basic concept behind this search method is to start from the current position, searching the neighbor nodes and expanding the searching set until the goal is reached. Originally, grid-based algorithms were not optimal controllers. However, combining the point-to-point shortest path problem with these path planners resulted in optimal path planners such as the A* search algorithm [2].

Sampling-based path planners, on the other hand, randomly sample the points of the space and perform the path planning for them, avoiding the discretization requirement of the state space [3]. Rapidly-exploring Random Tree (RRT) and probabilistic roadmap are the most famous sampling-based search methods [4].

Artificial Potential Field (APF) methods are another approach to path planning. These methods work by defining a potential field in the environment that attracts the robot towards the goal and repels it from obstacles. The path is then found by following the gradient of the potential field. APF methods are simple and efficient, but they can sometimes get stuck in local minima or fail to find a path in complex environments [5].

Learning-based methods for path planning use machine learning techniques to learn a policy or a model of the environment that can be used to find a path. These methods can be divided into model-based and model-free approaches. Model-based methods learn a model of the environment and use it to plan a path, while model-free methods learn a policy directly from experience [6].

Finally, Model-based approaches incorporate the model and dynamic of the system itself, the environment, and the objective to find the optimal path in the environment [7].

By combining these approaches, researchers are leveraging their advantages and enhancing the performance of the path planners in the environment. The Grid-based and Sampling-based approaches are heuristic approaches, meaning they use a heuristic function as a part of their cost. This results in performance degradation from one environment to the other. To generalize the path planning and overcome this issue, [8] proposes a novel method to speed up the RRT* search process by limiting its state space using a trainable algorithm. The aforementioned paper feeds the map of the environment, including the start and target positions into an encoder-decoder structure to get a probability map of the traversable paths the robot can move through. Then, the RRT* is implemented in this promising region, searching the available and more probable paths rather than the entire environment, leading to a faster process.

The same concept is implemented in [9], where authors integrate a trainable network in an A* Grid-based search and leverage the info from the environment to speed up the A* search. Conventional A* algorithm gets stuck in the dead-ends due to the lack of prior info from the environment. They provide an end-to-end method for path planning using neural networks.

In this project, the main focus is to implement [9]'s method, enhancing its structure with some concepts from [8] to observe how it affects performance. Also, the attempts to employ the same concept for collision avoidance in dynamic environments will be discussed.

The remainder of this report is organized as follows: We start with preliminaries at II. A brief description of the methodology in [9] is provided in III, the problem statement in this course project is explained in IV, followed by V, explaining the methodology and approaches implemented in this project. Finally, the results and discussion are provided in VI and VII, respectively.

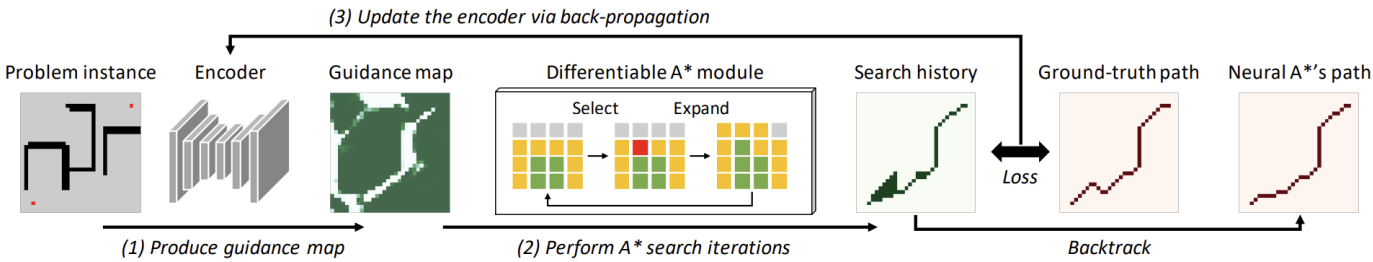

Fig. 1. The structure of the end-to-end Neural A* proposed in [9].

II. Preliminaries

A. A* Algorithm

A* is a widely used Grid-based pathfinding algorithm that divides the environment into grids, where each grid represents a node in a graph. The algorithm uses a heuristic function to guide the search toward the goal, making it more efficient than other pathfinding algorithms. The algorithm starts by exploring the nodes from the start or current node and chooses the most promising neighbor node to explore. It then explores the neighbor's neighbor nodes, and this process continues until the goal is obtained. To find the most promising neighbor and optimal path, A* requires a cost definition. In fact, it keeps a priority queue of the nodes that need to be explored, with the lowest-cost node in the front of the queue, having the highest priority. The cost includes two parts: the actual cost ($g(v)$), which is the actual incremental cost of the shortest path from the start node to the current node (v), and the cost-to-go ($h(v)$), which is the cost the agent should pay to get to the target node from the current node. A heuristic function, such as Euclidean or Chebyshev distance usually defines the cost-to-go. The total cost is given as follows:

$$c(v) = g(v) + h(v) \tag{1}$$

B. U-Net

The U-Net architecture is based on an encoder-decoder cascade structure. The encoder part of the architecture compresses the input image into a lower-dimensional representation, while the decoder part of the architecture decodes this information back to the original image dimension. This results in an overall U-shape for the architecture, which is where the name U-Net comes from.

The U-Net architecture consists of a contracting path and an expanding path. The contracting path is similar to the encoder part of an encoder-decoder structure and is responsible for capturing the context of the input image. It consists of several convolutional layers with increasing numbers of filters, followed by max pooling layers to reduce the spatial dimensions of the feature maps.

The expanding path is similar to the decoder part of an encoder-decoder structure and is responsible for enabling precise localization of the segmented structures. It consists of several convolutional layers with decreasing numbers of filters, followed by upsampling layers to increase the spatial dimensions of the feature maps.

One of the key features of U-Net is the use of skip connections between the contracting and expanding paths. These skip connections allow the network to combine the high-level context captured by the contracting path with the precise localization information captured by the expanding path. This helps to improve the accuracy and efficiency of the segmentation process [10].

III. Neural A*: A Brief Description

The novel method proposed by [9] is to get the information from the environment and pass it to the A* search to speed up the process as well as avoid getting stuck in dead-ends. The way the paper implements this idea is to modify the cost term of the A* based on the environment. They generate guidance, containing information on the environment and occupied spaces, using a neural network and pass it to the A* module to modify its cost. The output of the A* module contains the output optimal path and the search history of the A* algorithm. By search history, the paper means the neighbors the algorithm explored to get to the target. The goal of the paper is to make the algorithm produce a search history that is as similar to the ground truth path as possible. The input map, search history, and ground truth are binary tensors, containing 1 if a pixel is occupied or contains a path, and 0 if the pixel is traversable or should not include the path.

To get the histories similar to the ground truth, the paper defines an L1 Loss, $|a_n - y_n|$ to penalize the system either for false positives, where the pixel in the generated history should be 0 but is predicted 1, or false negatives, where the pixel should contain 1 but it contains 0. Based on this loss function and through back-propagation, the encoder learns to generate an appropriate guidance map based on the environment.

It is noteworthy that since the A* is a Grid-based search algorithm, it is not differentiable and thus, cannot be used during back-propagation. The paper modifies the A* algorithm to become differentiable and calls the new algorithm differentiable A*. The proposed methodology is shown in Fig. 1. As shown, the procedure contains two parts: (1) the encoder generates the guidance map, containing the geometrical cues and (2) the differentiable A* module implements the search method and produces the path. The paper uses a U-Net convolutional neural network (CNN) with the encoder-decoder

structure with the backbone of VGG-16 for the encoder part. The main focus of this project is to change the structure of the encoder and monitor the performance of the path planner.

IV. Problem Statement

In this project, different CNN architectures are being developed and a comparison of their performances is being demonstrated with the main purpose of investigating the efficiency of the encoder. Then, a dynamic obstacle is introduced to the problem and the structure is modified to compromise the dynamic environment and avoid collision.

V. Methodology

A. Encoder Architectures

In this section, different architectures tested in the same algorithm are described. One option was to flatten the input map and feed it to a Multi-Layer Perceptron (MPL) network. This was not a good approach since every neuron in MLP is connected to all other neurons, disabling the local feature extraction. CNNs on the other hand, are an appropriate choice when dealing with images as each neuron is connected to limited neighbor neurons, enabling them to recognize local patterns. CNNs are capable of recognizing a pattern regardless of whether it is rotated, translated, or scaled. CNNs take advantage of downsampling the input to reduce the size of the feature map. Being that said, this project sticks to the CNN architectures as the original paper.

Although we are using CNN architecture, many choices are available since there is a wide range of CNN structures for different applications. Based on the nature of the problem in this project, where the input map is a binary image with occupied pixels being 1, and the ground truth is also a binary image with the pixels containing a feasible path being 1, we can assume it as a pixel-wise binary classification. In this project, The first architecture implemented in the encoder in this project is a conventional CNN. The main purpose of this structure is to learn how to implement a CNN structure and train the network using PyTorch and PyTorch Lightning.

Trying different CNN algorithms, we realized that we have a hard constraint for this problem. The dimension of the output should be the same as that of the input. The input map is 32×32, so should be that of the output. However, as we get deeper into the convolutional layers, the dimension is typically decreased. The way the paper originally tackles this issue is to just increase the number of channels without changing the dimension by using specified kernel size, stride, and padding. After some literature review, we realized that an alternative to this limiting approach is to use an encoder-decoder structure, similar to what [8] did, where the detailed feature map is extracted during the encoder part and the size is increased in the decoder part. The second structure used in this project is to build an encoder-decoder structure from scratch.

Digging deeper into this project, one can infer that the problem is the binary image segmentation problem. The main purpose of image segmentation algorithms is to find the patterns and segments of the image based on the edges, texture of the objects, etc. In this problem, we are willing to find the parts of the image occupied by obstacles. Thus, image segmentation structures are expected to work well for this problem. One of the most efficient image segmentation structures is U-Net. Basically, U-Net has an encoder-decoder structure. But it takes the advantage of some skipping connections. The second CNN structure implemented in this project is of the U-Net type. The paper, originally, implements VGG-16 as the backbone of U-Net, but in this project, we will try to change its backbone structure to observe its impact on the performance.

TABLE I
First Custom CNN Structure

Layer	Input Shape	Output Shape	Kernel Shape
Conv2d	2, 32, 32	32, 32, 32	5, 5
BatchNorm2d:	32, 32, 32	32, 32, 32	
ReLU:	32, 32, 32	32, 32, 32	5, 5
Conv2d	32, 32, 32	64, 32, 32	5, 5
BatchNorm2d:	64, 32, 32	64, 32, 32	
ReLU:	64, 32, 32	64, 32, 32	5, 5
Conv2d	64, 32, 32	128, 32, 32	5, 5
BatchNorm2d:	128, 32, 32	128, 32, 32	
ReLU:	128, 32, 32	128, 32, 32	5, 5
Conv2d	128, 32, 32	256, 32, 32	5, 5
BatchNorm2d:	256, 32, 32	256, 32, 32	
ReLU:	256, 32, 32	256, 32, 32	5, 5
Conv2d	256, 32, 32	1, 32, 32	5, 5
BatchNorm2d:	1, 32, 32	1, 32, 32	

*1) **Custom CNN Structure**:* The constraint in this part is to keep the dimension of the input the same in all convolutional layers. Equation (2) shows the relation between the kernel size, padding, and stride size for the case $n_in = n_out = n$. Using this relation, we build the CNN structure with the layers given in Table I.

$$n \times (s-1) = 2 \times p - k + s \tag{2}$$

In this structure, we used 5×5 kernels and we kept the dimensions of all layers the same. Compared to the original CNN provided by the paper, we increased the kernel size, expecting to enable the algorithm to have a better understanding of the corners and dead-ends since in this case, we have increased the algorithm's field of view. We are using the L1 loss function and the solver is RMSprop in this case.

*2) **Custom Encoder-Decoder Structure**:* We tried to build an encoder-decoder architecture from scratch. The first custom encoder-decoder architecture we created is given in Table II. Since the results from CNN demonstrated some failure in dead-end detection, we increased the size of the kernel in the first convolutional layer to enable the encoder to capture local features of a wider area. Later, an average pooling layer is used to sum up all the info gathered in the early layers. Finally, the image is decoded to the input size in the last two layers. The loss function in this step is L1 and we use RMSprop optimizer.

TABLE II
FIRST CUSTOM ENCODER-DECODER STRUCTURE

Layer	Input Shape	Output Shape	Kernel Shape
Conv2d:	2, 32, 3	32, 28, 2	5, 5
BatchNorm2d:	32, 28, 2	32, 28, 2	
ReLU:	32, 28, 2	32, 28, 2	
Conv2d:	32, 28, 28	128, 26, 2	3, 3
ReLU:	128, 26, 26	128, 26, 26	
AvgPool2d:	128, 26, 26	128, 13, 13	
Conv2d:	128, 13, 13	256, 11, 11	3, 3
ReLU:	256, 11, 11	256, 11, 11	
ConvTranspose2d:	256, 11, 11	64, 16, 16	6, 6
ConvTranspose2d:	64, 16, 16	1, 32, 32	2, 2

Inspired by [8], we tried changing the loss function. We tried torch's weighted binary cross entropy loss function. We expected to get better results since the problem is a binary image segmentation and we are accounting for the unbalanced dataset in this case by setting the weights. The dataset is unbalanced since the majority of the map would be traversable, represented as 0. We also changed the solver to Adam for this case. However, the results got worse in this case. Looking at the training loss history, we realized that in this case the solver is tucked at local minima.

*3) **U-Net with ResNet50 Backbone**:* Paper originally used VGG-16 as the backbone of U-Net. VGG-16 is a popular deep-learning structure for image classification. It has fewer parameters, which makes it faster compared to other deep networks. However, based on the results from the original U-Net, we realized the network sometimes fails to detect the corners and dead-ends. Thus, we wondered how using a deeper image classification network can affect the results. We chose ResNet50, a 50-layer residual network introduced by Microsoft in 2015. Having more layers allows ResNet50 to capture more complex patterns in the data. Also, ResNet50 uses residual connections, allowing the model to learn more efficiently by bypassing the need to learn the identity function [11].

B. Dynamic Obstacle

Originally, the paper did not consider the dynamic obstacles in the map. In this project, we are interested in modifying the algorithm to compromise for dynamic obstacles. In the literature, most papers introduce Long-Short Term Memory (LSTM) to predict the obstacles' paths. However, in this paper, we will assume the obstacles' paths are available, predicted by a low-level planner for the sake of simplicity. This assumption eliminates the necessity of LSTM for obstacle path prediction and reduces the problem in path planning for a series of static maps.

initially, we wanted to add the time dimension as the channels. The idea was to have N channels for each dynamic obstacle, where N is the number of time steps, and have the dynamic obstacle's occupancy map at each channel. So, the dimensions of the channel would have been the spatial map and the channels would have represented the time series. We wanted to concatenate this with the input map, containing the occupancy map of the static environment, and then feed that into a U-Net structure which employs time distributed convolutional layer.

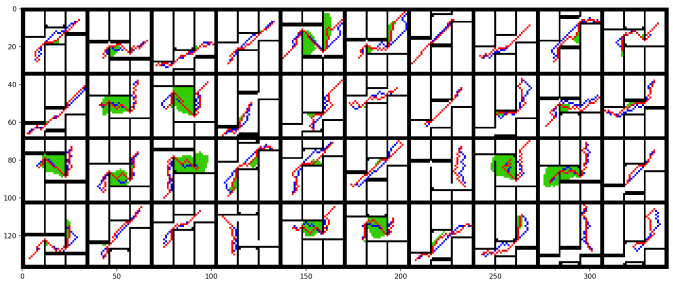

Fig. 2. Test results from paper's original CNN structure. The green area is the search history, the red path is the final optimal path generated by the neural A*, and the blue path shows the ground truth.

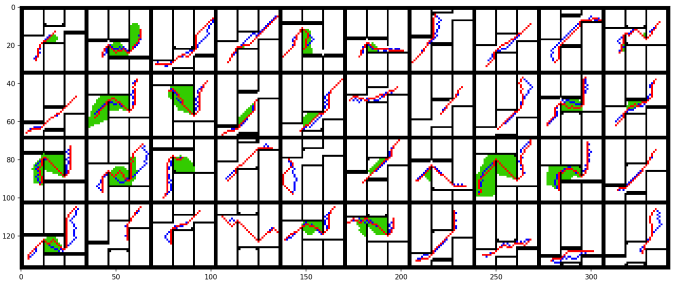

Fig. 3. Test results from paper's original U-Net with VGG-16 backbone. The green area is the search history, the red path is the final optimal path generated by the neural A*, and the blue path shows the ground truth.

VI. RESULTS

Table III shows the performances of different structures. Three measures of performance, P-opt, P-exp, and h-mean, are adopted from [9], and the execution time is added in this project. The Path optimality ratio (P-opt) measures the percentage of shortest path predictions for each environmental map. The Reduction ratio of node explorations (P-exp) measures the number of search steps reduced by a model compared to the standard A* search and the Harmonic mean (h-mean) of P-opt and P-exp shows how much their trade-off was improved by a model. The execution time shows the duration of path planning for 100 samples in the test set.

A. Paper's Original Results

Fig. 2. shows the results obtained from inputting the test set to the original trained CNN. Also, the results obtained from inputting the test data to the paper's U-Net with VGG-16 backbone are displayed in Fig. 3. We will compare these results with the results achieved from the custom structures.

B. Encoder Architectures

*1) **Custom CNN Structure**:* Fig. 4. depicts the results obtained from the first custom CNN structure.

TABLE III
COMPARISON BETWEEN DIFFERENT ARCHITECTURES

	P-opt	P-exp	h-mean	Execution Time (s)
Paper's CNN	0.84	0.42	0.5	0.410
Paper's U-Net VGG-16	0.93	0.43	0.5	0.537
Custom U-Net (ResNet50)	0.85	0.43	0.47	0.767
Custom Encoder-Decoder	0.96	0.0087	0.01	0.655
Custom CNN	0.91	0.48	0.54	0.359

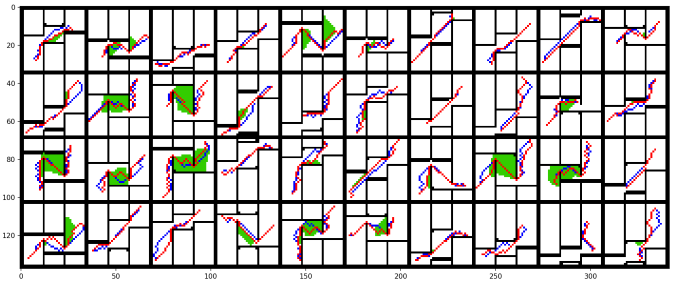

Fig. 4. Test results from the first custom CNN structure. The green area is the search history, the red path is the final optimal path generated by the neural A*, and the blue path shows the ground truth.

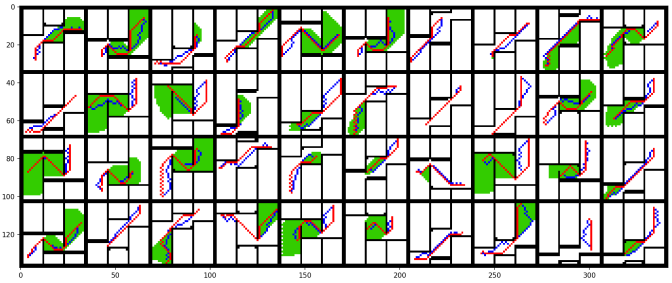

Fig. 5. Test results from first custom encoder-decoder structure. The green area is the search history, the red path is the final optimal path generated by the neural A*, and the blue path shows the ground truth.

*2) **Custom Encoder-Decoder Structure**:* The test result obtained from this architecture is shown in Fig. 5. As illustrated, the A* history (green area) is increased in many instances. Most of them indicate that the network failed to detect the dead-ends.

*3) **U-Net with ResNet50 Backbone**:* Fig. 6 shows the test results on the U-Net with ResNet50 backbone. As shown, in the majority of cases, the history is the same as the optimal path, meaning the A* search directly finds the optimal path. This indicates that the performance of this architecture is performing as expected. However, still, in some instances, the planner is stuck in the dead-ends and corners.

Comparing the test results obtained from this architecture and the paper's original U-Net architecture with VGG-16 backbone, shown in Fig. 3, one can see in many instances, the search history is improved with ResNet50 backbone.

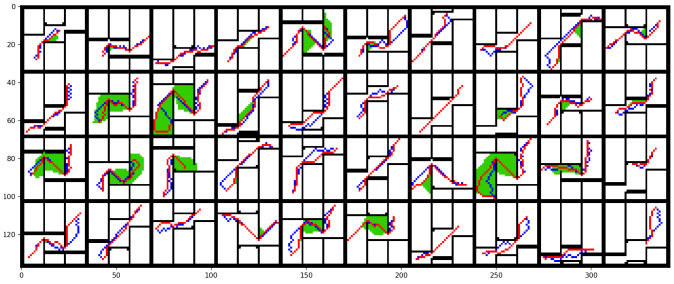

Fig. 6. Test results from U-Net with ResNet50 backbone. The green area is the search history, the red path is the final optimal path generated by the neural A*, and the blue path shows the ground truth.

VII. DISCUSSION

In this paper, we defined different architectures for the encoder part of the algorithm developed in [9], comparing their performances and investigating the impact of different parts of the CNN on the planned path. However, one of the most important parts of training a network is hyperparameter tuning. Without hyperparameter tuning, comparison between different networks is senseless, i.e. a structure might be a good match for a specific application, but the learning rate is such that the solver gets stuck in local minima. The paper claims that they did the hyperparameter tuning, but in this project, we only changed the hyperparameters manually a few times. Thus, it is more likely that our structures do not show their best possible performance. Moreover, We intended to adopt this algorithm for collision avoidance in dynamic environments as well, But unfortunately, we could not make it work.

REFERENCES

[1] J. Champagne Gareau, Beaudry, and V. Makarenkov, "Fast and optimal branch-and-bound planner for the grid-based coverage path planning problem based on an admissible heuristic function," *Frontiers in Robotics and AI*, vol. 9, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2022.1076897

[2] D. Šišlák, P. Volf, and M. Pechoucek, "Accelerated a* trajectory planning: Grid-based path planning comparison," in *Proceedings of the 19th International Conference on Automated Planning & Scheduling (ICAPS)*. Citeseer, 2009, pp. 74–81.

[3] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.

[4] Y. Lin and S. Saripalli, "Sampling-based path planning for uav collision avoidance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 11, pp. 3179–3192, 2017.

[5] M. G. Park, J. H. Jeon, and M. C. Lee, "Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing," in *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*, vol. 3. IEEE, 2001, pp. 1530–1535.

[6] J. Wang *et al.*, "Neural rrt*: Learning-based optimal path planning," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, pp. 1748–1758, 2020.

[7] A. Wolek and C. A. Woolsey, *Model-Based Path Planning*. Cham: Springer International Publishing, 2017, pp. 183–206.

[8] J. Wang *et al.*, "Robot path planning via neural-network-driven prediction," *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 3, pp. 451–460, 2022.

[9] R. Yonetani *et al.*, "Path Planning using Neural A* Search," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 12 029–12 039. [Online]. Available: http://proceedings.mlr.press/v139/yonetani21a.html

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[11] S. Mascarenhas and M. Agarwal, "A comparison between vgg16, vgg19 and resnet50 architecture frameworks for image classification," in *2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, vol. 1, 2021, pp. 96–99.