Master Computer Science

3D Visualization of Convolutional Neural Networks in real-time and time

Name: Bogdan-Ionel COVRIG
Student ID: 2346540
Date:

Specialization: Computer Science and Business Studies

1st supervisor: Dr Michael Lew
2nd supervisor: Dr Erwin M. Bakker

Master's Thesis in Computer Science
Leiden Institute of Advanced Computer Science

Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Convolutional neural networks represent the core of a variety of computer vision tasks. Visualizations of neural networks are usually created in 2D environments with static images as inputs. Usually, these visualizations can only be used on toy-sized networks and they are not able to display the networks' scale and complexity. Hence, no meaningful experimentation can be done. Having as starting point the capabilities of the 2D visualization, the aim of the current thesis is to get to the next level and propose a new tool for visualizing neural networks for all types of pre-trained models that are created using TensorFlow framework in a 3D environment. The tool we developed enables the usage of webcam or videos as an input and allows the user to see the dynamic response of the network in real-time or in time. Developed using modern technologies, CNN 3D Visualizer, is a desktop-based application which runs locally without the need of being connected to Internet, thus being very accessible in any conditions.

Key words: real-time, time, 3D visualization, deep learning

Contents

1. INTRODUCTION 5
2. LITERATURE REVIEW 7
   2.1. 2D Visualization 7
   2.2. 3D Visualization 9
3. CHALLENGES AND GOALS 11
   3.1. Challenges 11
   3.2. Goals 12
4. VISUALIZATION INTERFACE OF CNN 3D VISUALIZER 13
   4.1. Models 13
   4.2. Customizable 13
   4.3. Visualization scene 15
   4.4. Desktop based implementation 15
   4.5. Workflow 16
5. IMPLEMENTATION ISSUES 17
   5.1. Data serialization/deserialization 17
      5.1.1. Serialization 17
      5.1.2. Deserialization 19
   5.2. Failed tries 20
      5.2.1. TensorSpace improvement 20
      5.2.2. Blender 21
6. USAGE SCENARIOS 23
   6.1. Visualization of features 23
   6.2. Model comparison 24
   6.3. Model errors 25
   6.4. Adversarial attacks 25
   6.5. Data quality of synthetic data generation with GAN 27
   6.6. Different inputs response 27
7. EVALUATION 29
   7.1. Feature visualization 29
   7.2. Custom models 30
   7.3. Multiple models 31

7.4. Desktop based implementation 32
8. CONTRIBUTION 33
9. FUTURE WORK 34
10. CONCLUSIONS 35
REFERENCES 36
ANNEX 39

1. INTRODUCTION

Digital videos and images are present in everyone's life. We are consuming and producing billions of images and millions of videos every year. For humans it is easy to assess visual information. We can identify the objects and actions or events just looking at an image or at a video. The quality of an image can be easily perceived by us. If, for example, an image is blurred we will notice it right away. Identifying something we have not seen before or associating it with some video that we saw before is also easy, same as grouping images that are meaningful to us. However, for computers such tasks are by no means straightforward or easy. Given the huge number of images and video content that is created, together with the heterogeneity of the applications (from infotainment to defense applications, and from creating personalized video streams or summaries to supporting video recommendation) that require some form of visual information assessment, it makes it impossible for us to be able to make decisions based on the all content in the world.

The need of making computers understand the visual information in the same manner as we do and to be able to make related decisions becomes extremely important. Thus, on the one hand, humans developed image and video analysis and understanding techniques such as: video temporal decomposition in shots and scenes, video event detection and recounting [1], video concept detection [2], image/video quality and aesthetics assessment [3], or image/video organization tools and applications [4] [5]. On the other hand, different machine learning techniques were also created: learning with uncertainty, multi-label learning [6], subclass methods for dimensionality reduction and learning, deep learning for image/video understanding or cascades and other classifier combinations.

Furthermore, machine learning has seen an extraordinary twist in recent times due to the rise of the Artificial Neural Networks (ANN). The most impressive ANN architecture is the Convolutional Neural Network (CNN). CNN's are like ANN, more exactly they are composed of neurons that optimize themselves through learning. Each neuron will receive an input and perform different types of operations. Starting from input to the final output of the class score, the whole network will express a single perceptive score function named weight [7].

CNNs are composed of finite sets of processing layers that can learn various features of input such as images. As we can see in the picture below, CNN are composed of multiple blocks named layers in architecture. Usually, a traditional CNN is made of multiple blocks of convolution and pooling layers followed by one or multiple fully connected layers and an output layer. The first layers learn and extract high level features such as finding dots, lines, curves etc. The later layers learn and extract low level features such as recognizing objects and shapes.

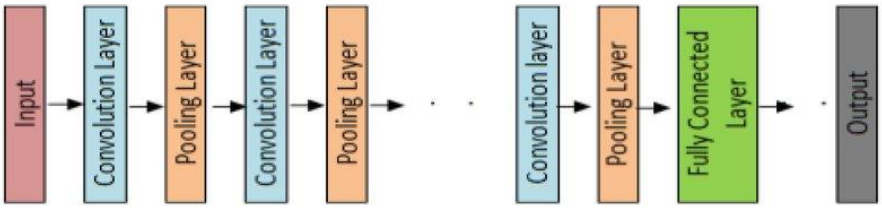


Figure 1: Conceptual model of CNN [8]

Given CNN's high popularity, the interest in visualization increased at a high pace. More specifically, researchers focused primarily on developing 2D visualization tools, while 3D visualization has been less pursued.

Our thesis focuses on presenting a new tool for 3D visualizations in time and real time. The first chapter provides a better overview of what CNN is and what it is used for, specifying what breakthroughs it enables and what kind of visualization techniques and tools were created. In chapter 2 we go deeper and explore different tools and techniques created for 2D and 3D visualizations. Based on chapter 2 findings we define the goals and challenges of the research in chapter 3. In chapter 4 we present our tool and its characteristics. Failed tries in achieving the creation of the tool are presented in the next chapter with the purpose of providing an overview of the path for creation of this tool. Usage scenarios are presented in chapter 6 and evaluation of the tool in chapter 7. In the last chapters we present our contribution (chapter 8), future possible work (chapter 9) and conclusions (chapter 10).

2. LITERATURE REVIEW

In the last years, Convolutional Neural Networks have enabled many breakthroughs in a variety of computer vision tasks from image classification [9] [10]to image captioning [11]or object detection [12]. Even these models enable superior performance their components are hard to interpret [13]. Based on growing interest in visualization field many tools or libraries that use different strategies were created. A significant part of the work is focused on 2D visualization with focus on how the structure of the neural network looks like. Tools or libraries worth mentioning are: TensorBoard which provides the visualization and tooling needed for machine learning experimentation[1] and Keras which is an open-source library that provides a Python interface for artificial neural networks[2]. When it comes to 3d visualization, based on our research, currently there are only two approaches available: Tensorspace.js[3] and a tool created by Adam W. Harley [14].

In the following part we are going to present some of the above-mentioned visualization techniques and tools, the most relevant ones for our purpose, which also represented the starting point for our research and led us to the implementation of a new approach in this field.

2.1. 2D Visualization

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs and Hod Lipson in their research named "Understanding Neural Networks Through Deep Visualization" [15] are advancing two tools. The goal of the first one is to visualize the activations produced on each layer of a trained convolutional neuronal network as it processes images or video. Their research led them to multiple conclusions.

[1] *Tensorboard*: https://www.tensorflow.org/tensorboard?fbclid=IwAR10ysZ45i7Bd7Xmf5R8_56dQNUrS4xOjk23TBJ536BcMCEbkTruIz-hOo

[2] *Keras* : https://keras.io/getting_started/

[3] *TensorSpace* : https://tensorspace.org/

Firstly, the conclusion they reached to emphasizes that representations on some layers are local and therefore instead of having a representation distributed on all layers, the researchers identified, for example, that detectors for text, flowers, fruit and faces are displayed on specific conv layers. Moreover, they discovered that by using Google images or images from Flickr the classifications are often correctly displayed, while when the input is represented by a webcam the predictions are frequently incorrect because no items from the training set are shown in the image.

Secondly, the researchers are introducing a new regularization method in order to obtain better interpretable images for large convolutional neural networks. They took existing regularization methods such as: 2 decay, Gaussian blur, clipping pixels with small norm and clipping pixels with small contribution and combined them in order to obtain better visualizations. They identified four hyperparameter combinations which produce different styles of recognizable images after viewing images produced by 300 randomly hyperparameters combinations. Moreover, they suggested that based on their research, discriminative parameters contain significant generative structure from the training dataset. By using the results of this research as a starting point, future researchers could transfer discriminatively trained parameters to generative models.

Having as a starting point the tool implemented by Yosinski and his team [15], in 2016 a new tool called Quiver was created by Jake Bian[4] where he visualizes the feature maps of each layer for a model of your choice. He managed to extract the feature maps for a specific layer and to show the structure of the network. In comparison with the implementation created by Yosinski we can say that Bian managed to add a clearer structure view of the network. However, in his approach the only available option is to use static images with no support for video or webcam capturing.

The same year another research conducted this time by Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh and Dhruv Batra [16]is proposing a new technique for producing visual explanations for decisions from a large class of CNN based models making them more transparent and explainable. They managed to obtain high resolution and class discriminative Guided Grad-CAM visualizations that can discriminate accurately

[4] Bian, J. (2016). *Quiver :* https://github.com/keplr-io/quiver

between classes, better expose the trustworthiness of a classifier and help identify biases in datasets.

Later in 2020, Zijie J. Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, and Duen Horng (Polo) Chau [17] have designed a tool addressed to non-experts who want to learn and examine convolutional neural networks through interactive 2d visualization. The main advantage of the tool being its user-friendly web-based and open source character.

2.2. 3D Visualization

Adam W. Harley thanks to his research called "An Interactive Node-Link Visualization of Convolutional Neural Networks'' [14], he creates a neural network trained on handwritten digit recognition with the intent of showing the actual behavior of the network given user-provided input. The network was trained on an augmented version of MNIST. This approach it is based on a small network and it is limited to just handwriting recognition. Moreover, it does not allow you to see the fully connected layers and the convolutional layers in the same time, you can either visualize one or the other.

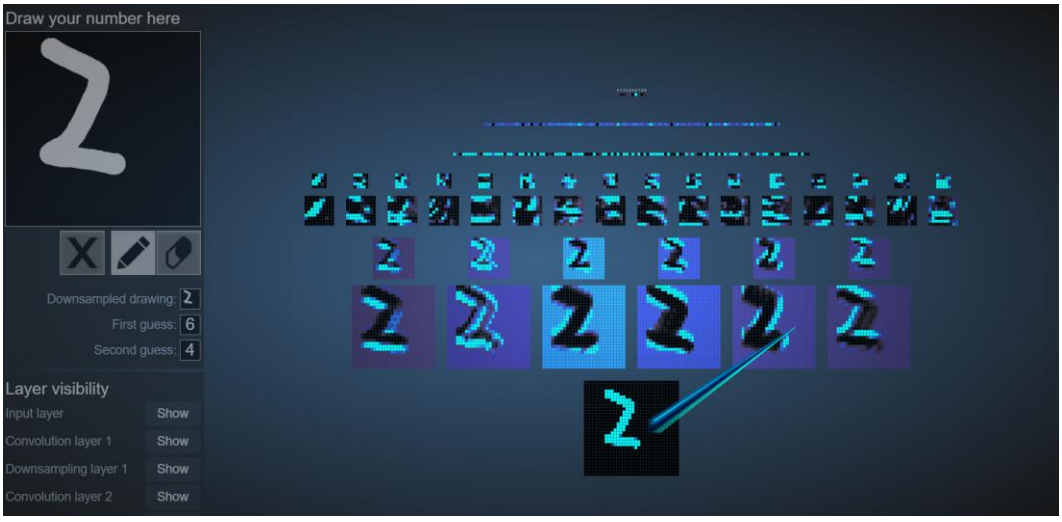


Figure 2: Screenshot Capture of the tool [14]

TensorSpace proposes a more advanced technology. It is a neural network 3D visualization framework built by TensorFlow.js, Three.js and Tween.js. TensorSpace provides Layer APIs to build deep learning layers, load pre-trained models, and generate a 3D visualization in the browser. By applying TensorSpace API, it is more intuitive to visualize and understand any pre-trained models built by TensorFlow, Keras, TensorFlow.js, etc. Their approach offers a better understanding of the internal structures of models by presenting not only the basic structure but also the processes of internal feature abstractions, intermediate data manipulations and final inference[5]. The tool is the only one that offers 3D visualization and let you add your custom models to visualize them. Also, it offers a range of pre-built models like LeNet, AlexNet or InceptionV3.

Besides the positive aspects, we have also identified some downsides. First of all, the necessity of having a good internet connection for big models. In this case a good example being represented by a prebuild model such as InceptionV3. Web browser limitations, in the case of Inception V3, led for the application to freeze when trying to open multiple layers at once. Another disadvantage is the fact that for the last layers it is almost impossible to distinguish the activated neurons.

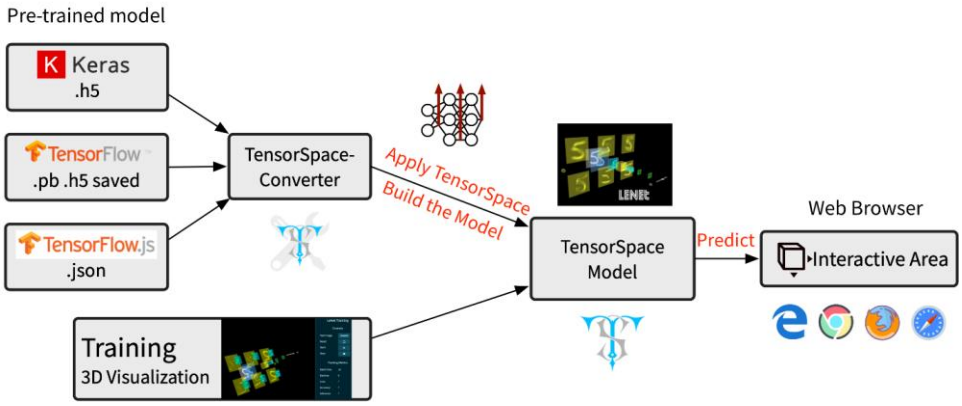


Figure 3: TensorSpace Workflow[6]

[5] *TensorSpace*. (n.d.): https://github.com/tensorspace-team/tensorspace
[6] https://tensorspace.org/html/docs/startIntro.html

3. CHALLENGES AND GOALS

3.1. Challenges

By analyzing the previous research done in the field of 3D and 2D visualization we have identified the following:

- TensorSpace 3d visualization is limited to static image input;
- Yosinki's implementation does not offer support for modern framework as TensorFlow;
- Adam W Harley's implementation is limited to either visualizing convolutional layers or fully connected layers for only one model.

Having as a starting point the above-mentioned findings we have identified several challenges:

C1. Video input stream/ Real-time input stream feature visualization. Static input feature visualization is not a difficult task anymore. There are number of tools that are able to visualize static images and offer multiple information like the ones presented in the literature review chapter [18] [15] [17]. However, there are no available tools offering a real-time input or a video input stream as processing data in time/ real-time represents an extensive task hard to tackle.

C2. Adding your custom model. Creating a tool that not only offers support to different models but also allows users to bring their own pretrained models is an even more challenging task. The difficulty comes from the fact that CNNs have different architectures and layer organization such as MobileNetV2 [19], VGG16, DenseNet [20] and many others.

C3. Challenge in deploying interactive 3D Visualization tool. Most of CNNs are written using well known frameworks such as TensorFlow [21] or PyTorch [22]. However, they are lacking in terms of features visualization when it comes to 3D visualization. Even if tools like TensorSpace managed to use modern libraries having a web-based implementation, they are limited when it comes to visualizing big models.

The above listed challenges cover most of the limitations and/or lacks faced by the tools introduced in the literature review chapter.

3.2. Goals

Based on the challenges and missing features presented in the previous sub chapter, we defined the following goals for our tool called CNN 3D Visualizer. It is an interactive tool that can be useful to both beginners and experienced researchers.

G1. Feature visualization of video stream. One of our goals is to develop a system that offers the possibility to have: a webcam real-time capturing, a recording option which records predictions over a specific period of time and also a video uploading option. All these are especially helpful as they allow users to move different items around a camera, occlude and combine them, and perform other manipulations to actively learn how different features in the network respond (Sect 4.2-4.3).

G2. Custom model option. In accordance with the second challenge (C2) more specifically to the option of adding multiple models in an easy manner, we aim to have this capability implemented in our tool (Sect 4.1). The result we expect to achieve is to engage users to use their own models and to improve future models' creation (Sect 4.2).

G3. Big models' representation. Another goal we want to accomplish is to implement the possibility of visualizing both small and big models, no matter the size, in a 3D environment.

G4. Desktop-based implementation. We also aim to develop a tool that is accessible to all users without the need of having a strong internet connection. We want to achieve that by having a desktop-based implementation. That will facilitate users to access the tool in any situation not being mandatory to be connected to Internet (Sect 4.4).

We assess to offer these capabilities also for back-propagation. However, offering this functionality it will necessitate designs that are hard to unify with visualization of forward propagation [23]. Therefore, we have decided to focus only on visualizing predictions of pre-trained models. We consider that offering support to back propagation can represent a topic for future work (Sect future work).

4. VISUALIZATION INTERFACE OF CNN 3D VISUALIZER

CNN 3D visualizer, visualizes the forward propagation, transforming input images in class predictions of a pretrained network. Users can on one hand to choose from a wide range of trained preexisting models without being necessary to have an understanding of model and on the other hand they have the possibility to use their own models (Sect 4.1). Our tool also allows the customization of the visualization (Sect 4.2) based on multiple options and the interaction with the model (Sect 4.3).

4.1. Models

In order to offer a wide range of models that can be visualized we used a large majority of the available TensorFlow Keras models, namely: MobileNetV2, VGG16, VGG19, Xception, DenseNet121, DenseNet169, DenseNet201, InceptionV3, ResNet50, ResNet101, ResNet152, ResNet50V2, ResNet101V2, ResNet152V2, InceptionResNetV2, NASNetLarge, NASNetMobile. By using TensorFlow, we can visualize all the most important, used and popular models that are available. However, we allow users to add their own CNN models making the application more user friendly. In order to add their models, they have to convert the model into a functional model. Conversion of the model can be done easily if the user makes usage of the convertor example that we offer as a template. After conversion the model is saved as h5 using TensorFlow Keras.

4.2. Customizable

The settings screen offers a wide range of options that allow the user to choose multiple configurations simultaneously. By doing this we provide the user with the possibility of using the application for creating multiple scenarios.

Change of input source. Users can choose between (1) prerecorded video as input or (2) webcam capturing. Using video from local storage as an input enables users to use the same set of video stream data for multiple networks that can help identify differences or similarities between networks. Webcam capturing offers the possibility to visualize in real-time or with a buffering period the network classification. In both cases CNN 3d visualizer resizes the images while preserving aspect ratio.

Recording. This option was created for two reasons. The first reason refers to the fact that the user would like to have the option to record himself or different objects and afterwards to be able to analyze the results. The second one is related to some of the big networks that can achieve only a low frame rate in time visualization. In order to still offer a good framerate, the user can record for a period of time of up to 360 seconds and to see the results with a decent framerate.

Device management. Given the fact that a user can have different computers with different specifications we also added a limitation in performance. Therefore, the user can choose the number of threads that will start making calls to the server. This will have as a consequence the reduction of the frame-rate, but it will enable the program to be used on less powerful computers.

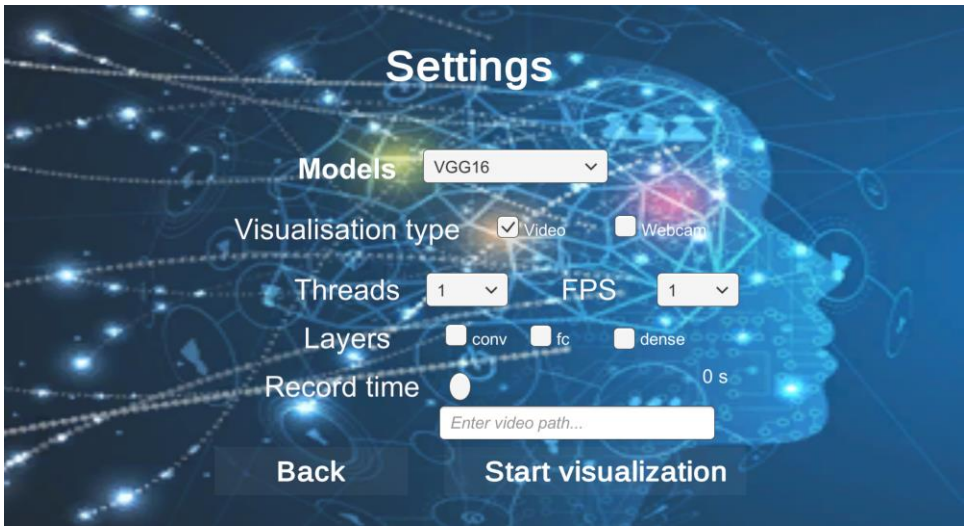


Figure 4: Settings panel with options available for use

4.3. Visualization scene

User benefits from an interactable space after the visualization started. He has the possibility to move in the scene to visualize the layers from different distances and he can also receive the information about which specific feature belongs to which layer, as in the below image.

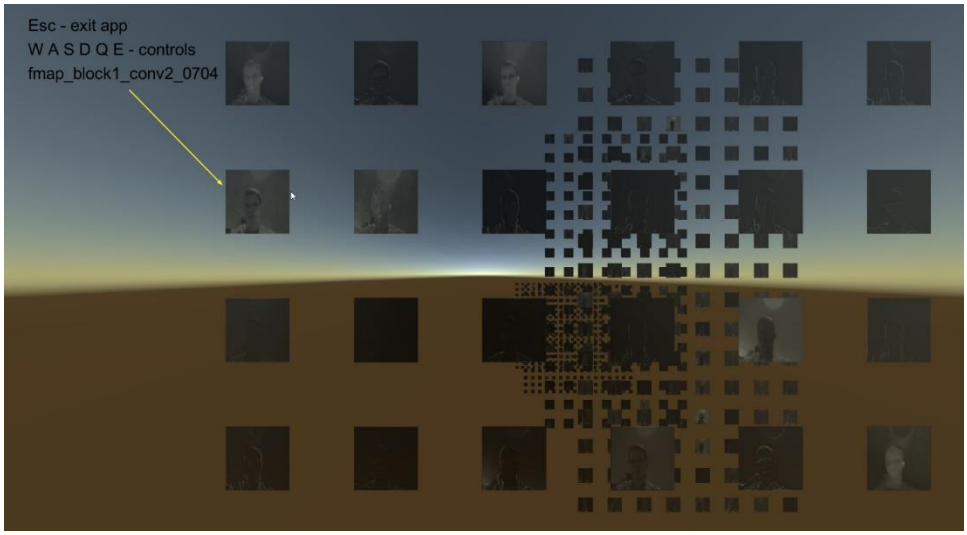


Figure 5: VGG16 scene visualization

4.4. Desktop based implementation

CNN 3d visualizer is a desktop-based visualization tool that enables users to understand the way a network thinks by using a dynamic input. A new user is able to use the tool locally on Windows machine, no internet connection being necessary after the models are run for the first time.

Interface visualization. CNN 3d visualizer loads pre-trained TensorFlow models and computes forward propagation results in real-time and time.

4.5. Workflow

In order to offer a better overview of CNN 3d visualizer we present below the workflow of the application.

Once the user opens the application, there are two options available, namely quitting the application or going forward to the model selection screen. In the model selection screen, there are three important triggers. Thus, one can choose between using a video or a webcam as an input. When choosing a video, a customizable text box, where you need to specify the path to the video is being displayed. In the case of the webcam it will automatically capture images using computer camera once you start visualization. The second trigger refers to the option of adding a custom model or using an already pre-trained model. If you choose adding a custom model, then a path to the h5 file and a text box where you need to specify the input size will appear on the screen. The last trigger refers to enabling the recording period. In case it is greater than 0, it will make the predictions for the selected period without showing them. The results will be displayed only after the time passed. Once the user sets up the desired configuration, and starts the visualization, in the visualization scene he/she can move around the scene using the keys WASD. By doing this he/she can get information about the features' names by clicking on them.

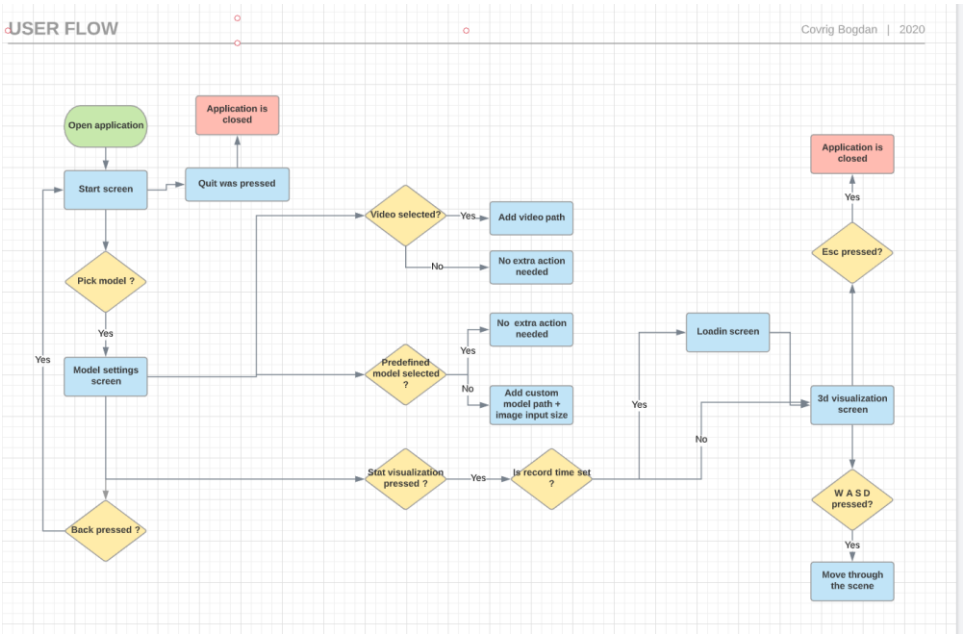


Figure 6: Workflow

5. IMPLEMENTATION ISSUES

No tool development goes smooth. There are a range of different issues and challenges that come up and block us, make our task more difficult, force us to find new solutions, rethink our strategy in order to achieve our goal. Programming CNN 3D Visualizer by using a client-server implementation was a challenging journey which brought us several blockers and disruptors which made our job more difficult and more time consuming. In the following section we will present the main issue we have faced. We believe that insisting on it can be beneficial also for future researchers.

5.1. Data serialization/deserialization

One of the main problems we had to surmount was the application's speed. In this regard, one of the major drawbacks in speed was represented by data serialization and deserialization. When using VGG models, for example, we are working with millions and millions of parameters. Serializing and deserializing those parameters was at the beginning a very time-consuming task. Therefore, we started investigating and searching for the most performant tool available in terms of serialization and deserialization. The purpose was to find the best and fastest option to serialize the message in python and deserialize it in unity using c#.

5.1.1. Serialization

After consulting several papers and research among which we can mention *Evaluating serialization for a publish-subscribe based middleware for MPSoCs* [24] or *Smart grid serialization comparison: Comparison of serialization for distributed control in the context of the Internet of Things* [25], we found a good serializer which was answering our needs, namely

MessagePack. Even if the author of this tool stated that in certain cases there are other serializers that outperform this one[7], our analysis demonstrated that this tool was the best solution for our research.

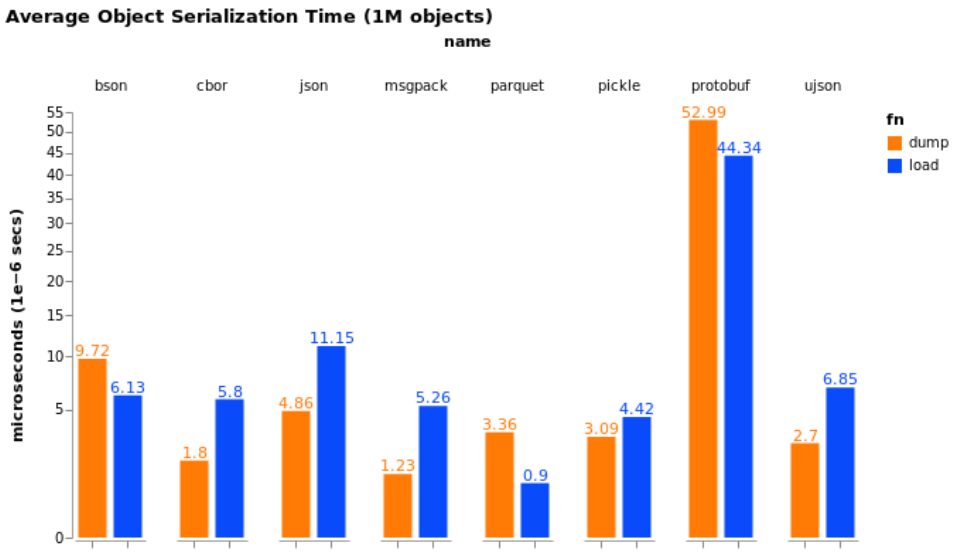


Figure 7: Python serialization[8]

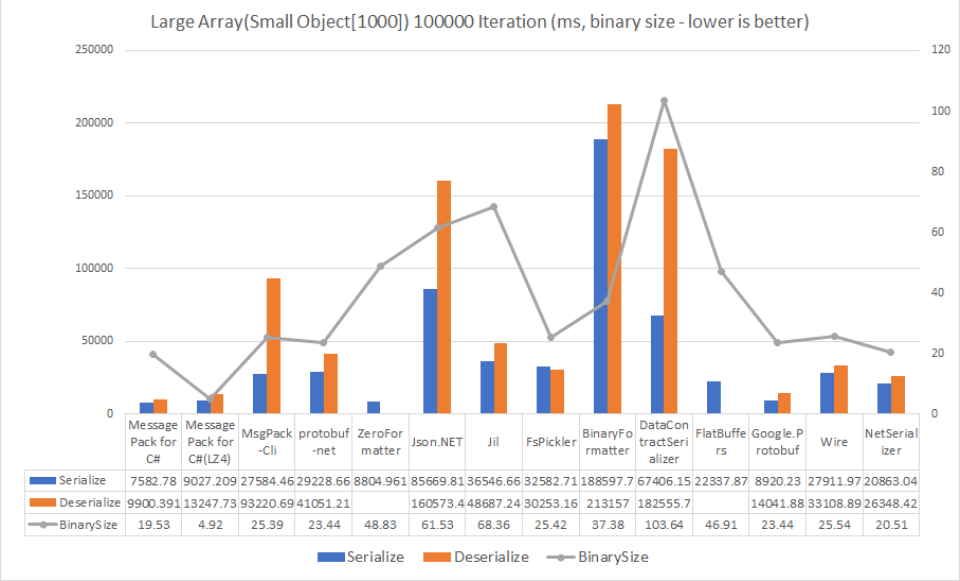
The decision of choosing this tool for serialization is based on a several key features. First of all, MessagePack offers support for streaming deserializers which is useful for network communication. Secondly, this tool supports "bytes" which represents a missing feature in JSON. Moreover, unlike Protocol Buffers, for example, MessagePack does not necessarily require defining a template for the way the message should look like.

This tool is also recommended when one needs speed and a flexible schema[9], as in our case. As highlighted by certain researchers [25] MessagePack has the smallest size when it comes to send data, this being an important feature for our implementation when we use the record feature. Record feature saves data in a queue for a specific period of time, therefore in order not to have memory issue, data size that is sent needs to be as small as possible. Moreover, the deserialization process that we will focus on in the next sub-section made MessagePack the best choice for our research in terms of speed and flexibility.

[7] https://gist.github.com/frsyuki/2908191
[8] https://medium.com/@shmulikamar/python-serialization-benchmarks-8e5bb700530b
[9] https://medium.com/@shmulikamar/python-serialization-benchmarks-8e5bb700530b

5.1.2. Deserialization

If in the case of serialization choosing MessagePack implied a more complex decision making-process, the analysis and comparison of several options before concluding that this was the best option, in the case of deserialization the decision was easier. After making some analysis MessagePack was the only winner when using C#.



	Message Pack for C#	Message Pack for C#(LZ4)	MsgPack -Cli	protobuf -net	ZeroFor matter	Json.NET	Jil	FsPickler	BinaryFo rmatter	DataCon tractSeri alizer	FlatBuffe rs	Google.P rotobuf	Wire	NetSerial izer
Serialize	7582.78	9027.209	27584.46	29228.66	8804.961	85669.81	36546.66	32582.71	188597.7	67406.15	22337.87	8920.23	27911.97	20863.04
Deserialize	9900.391	13247.73	93220.69	41051.21		160573.4	48687.24	30253.16	213157	182555.7		14041.88	33108.89	26348.42
BinarySize	19.53	4.92	25.39	23.44	48.83	61.53	68.36	25.42	37.38	103.64	46.91	23.44	25.54	20.51

Figure 8: MessagePack for C# comparison with other serializers[10]

Based on the elements presented above and by choosing MessagePack we managed to solve the issue of serialization and deserialization speed and to make an important step towards our goal of real-time visualization.

[10] https://github.com/neuecc/MessagePack-CSharp#performance

5.2. Failed tries

Research is often one step forward, two steps back process and we believe that also failures and drawbacks are part of the success. By talking openly about our failed tries make science more open and transparent and can help future researchers to avoid making the same mistakes and save valuable time when implementing their own ideas. In the next section we will present some of our failures.

5.2.1. TensorSpace improvement

By analyzing the existing tools, the one closest to our goal was TensorSpace. Its representation was partially responding to our expected representation. You can see below an example of a model representation in TensorSpace. The first image shows the way the first layers are represented, while the second one presents the way the structure is displayed.

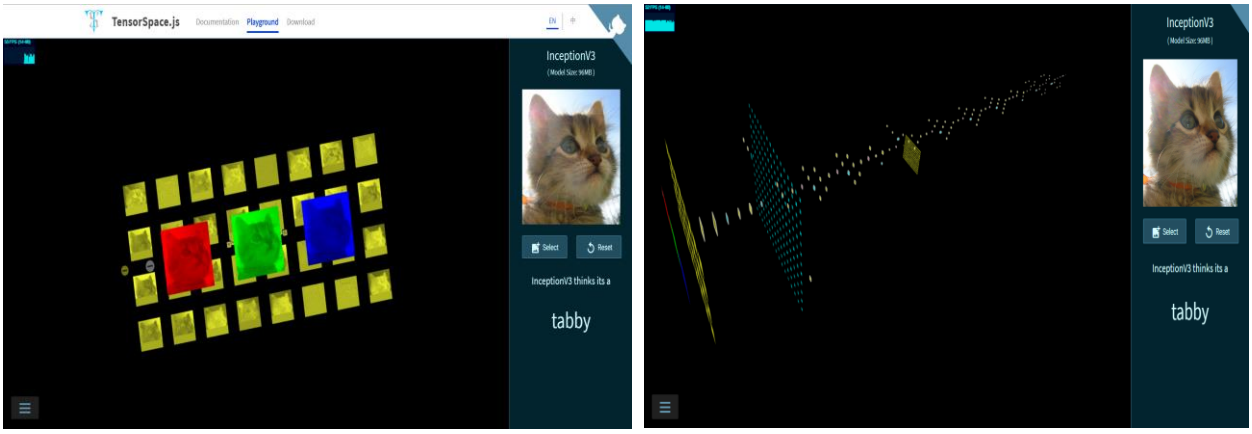


Figure 9: InceptionV3 TensorSpace representation

Even if the TensorSpace implementation was interactive, it was still lacking the ability of having a real time prediction. Moreover, there was no webcam capturing available. These aspects lead us to the conclusion that this tool has to be thoroughly analyzed in order to find the available

options for improving it and obtaining the real-time visualization we were targeting from the beginning. As a consequence, we started analyzing the code and modifying it based on our purpose.

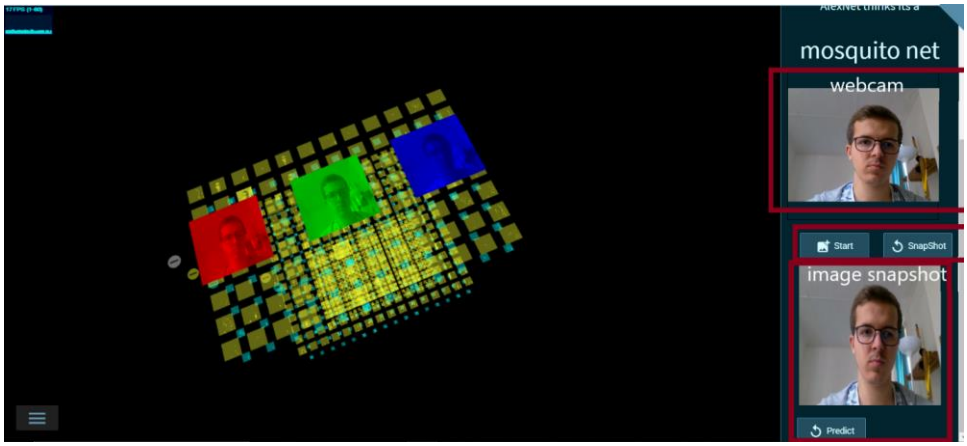


Figure 10: TensorSpace codebase modified

By modifying the TensorSpace implementation we succeeded to obtain a webcam capturing but without a real time prediction. We reached the point of taking a snapshot of the image and predict based on that. We had to stop at this point with this approach because we discovered that by trying to visualize all layers at the same time in models like InceptionV3 or by trying to visualize big models, the browser lead to screen freeze due to the web browser limitations. Another downside was the fact that in the last layers it was hard to distinguish the activated neurons.

5.2.2. Blender

Having the previous approach in mind, we considered the option of developing a desktop-based application by using a native implementation. Therefore, we have investigating different solutions that could have helped us get closer to our goal and decided to use Blender. Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline —modeling,

rigging, animation, simulation, rendering, compositing and motion tracking, video editing and 2D animation pipeline[11].

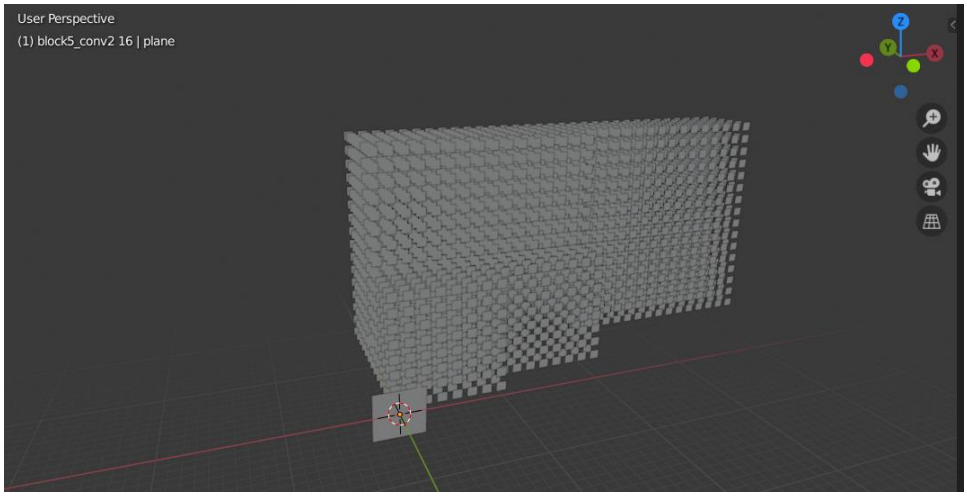


Figure 11: Blender model visualization

With the help of Blender, we tried to generate the model on the fly and have everything built based on the network that you add in the python script. The idea was to not be limited to a couple of models only. In other words, everyone could have just added the model which would have been generated accordingly. No additional converter being needed in between would gave extra flexibility.

After writing the code and testing it we discovered that the generation of the model tended to be slow and no real time approach was doable due to the lack of speed. After an in-depth analysis and trying to do the webcam capturing with a decent frame rate, we discovered that the webcam capturing is writing the image at a framerate that is extremely low and the viewport updates sampling method does not get enough samples to update the image fast enough. Besides that, Blender uses expensive computationally methods to create high quality imagery and rendering is not seen as an important factor. We can say that Blender is used to author content like 3d models and animations and not for interactive real time applications. [26]

[11] https://www.blender.org/

6. USAGE SCENARIOS

In pursuance of providing a perspective on what this new tool can offer, we consider it is important to understand the benefits of using it. Hence, in the following section we will stop upon the most important usage scenarios.

6.1. Visualization of features

Ionel is a new researcher and he starts using CNN for his project. He has a limited understanding of deep learning. So far, he knows that each layer is learning different things such as edges, patterns or parts of the objects. However, he is willing understand how networks are really predicting and recognizing these features. For doing this, the fastest and most efficient learning process [27] is the one through visualization. Thus, by using CNN 3d visualizer he should immediately manage to see how the predictions are made and he should not see these networks as black-boxes anymore. Understanding what was learned by these models might give him ideas of how to improve models in the future. We can take as an example the deconvolutional technique for visualizing features learned by the hidden units of DNNs which lead to state of the art performance on the ImageNet benchmark in 2013 [18].

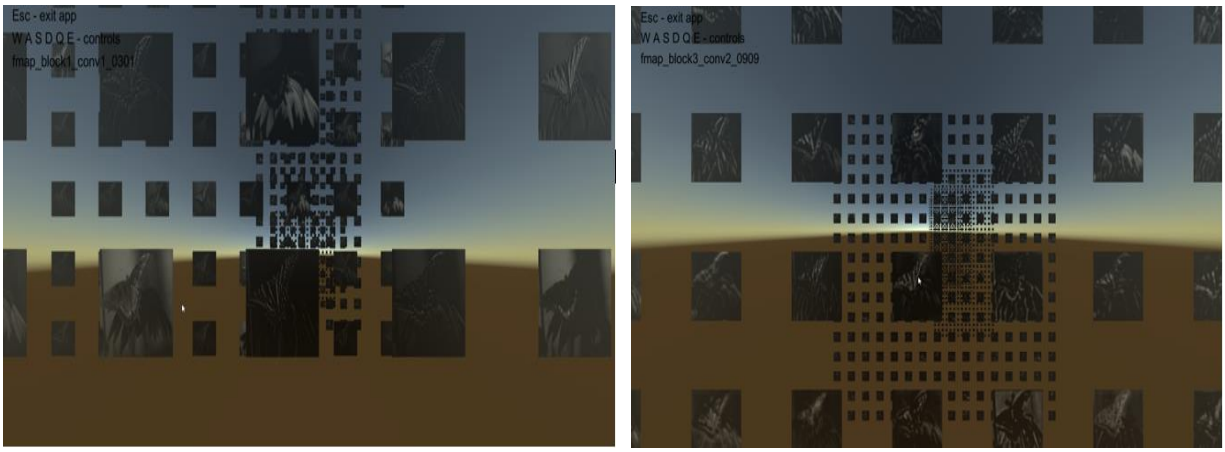


Figure 12: Vgg16 network with a static image as input

6.2. Model comparison

A professor at a top university in Europe is using different CNN models in her research. She is trying to identify which models are better for a specific video that contains multiple predefined images. She succeeds in getting top predictions from each model at different points in time of the video, but she wants to understand how these predictions are made. Using CNN 3d visualizer she could immediately put her video stream in each of the models and identify how features are activated. Thus, she can see that in case of MobileNet v2 less features are recognized than in case of VGG 16. She should be able to understand now the difference between the way the models are predicting using the same inputs and she could explain it in an easier manner in her research.

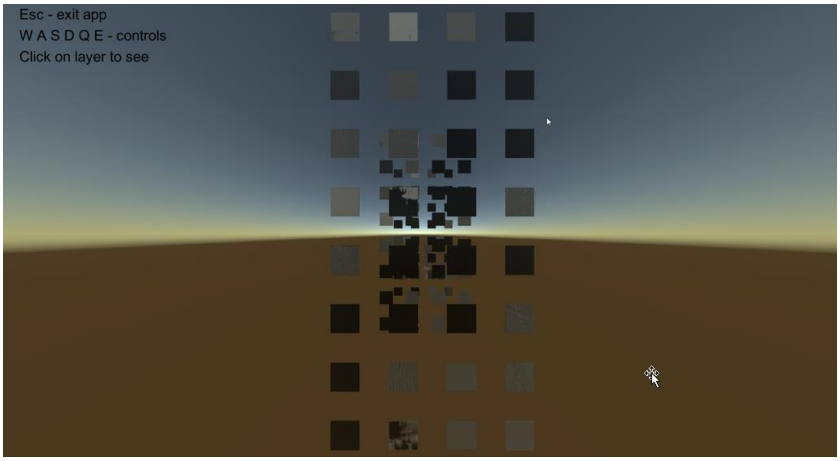


Figure 13: MobileNet v2 first layer activations

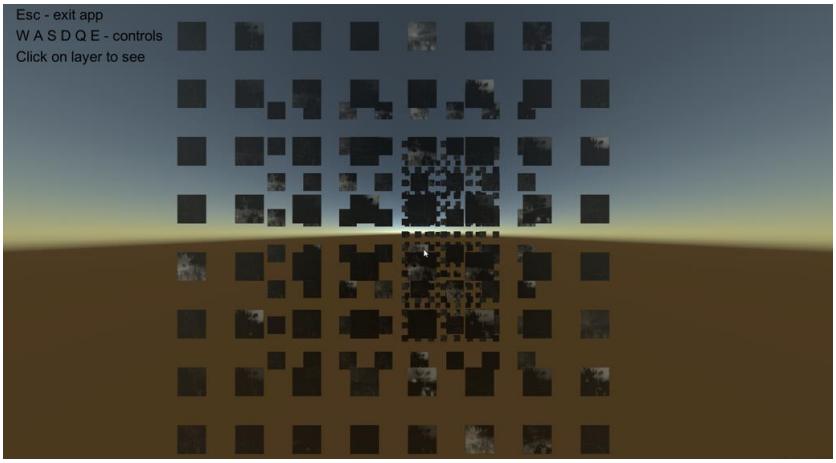


Figure 14: VGG16 first layer activations

6.3. Model errors

Javier is working on the creation of a new model. After finishing his model creation, he discovered a large error rate when classifying the image. The model was not able to make the difference between a dog and a cat for example. In order to fix the error, he run CNN 3D visualizer with his model and webcam. He might spot out which layers are identifying incorrectly the features and the layers which are not identifying features at all. He should have now a clearer path on how to fix the problem.

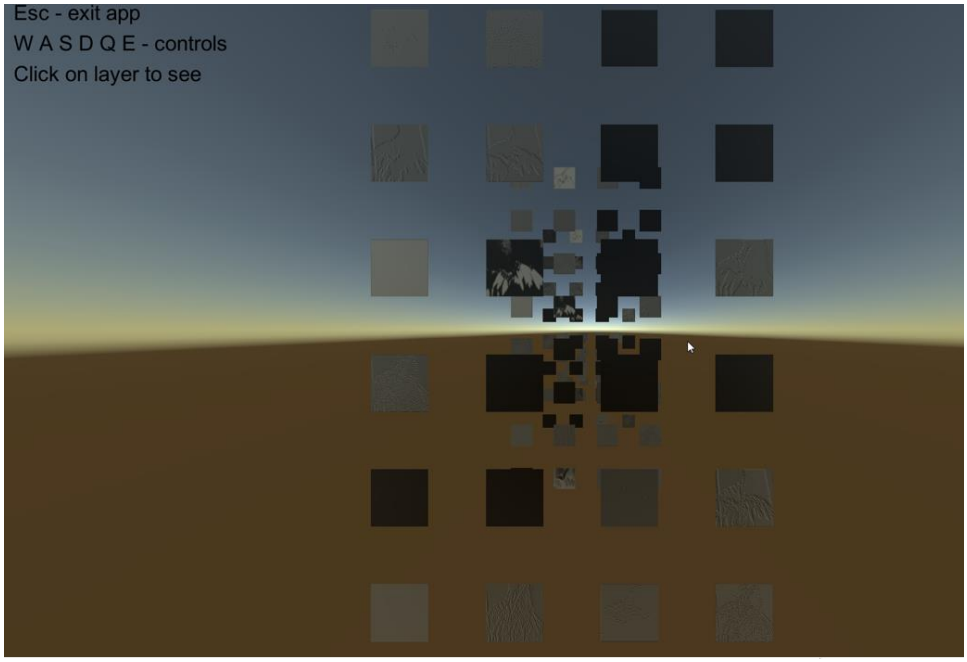


Figure 15: Feeding the network with an image is not activating a big part of neurons

6.4. Adversarial attacks

A colleague of Javier is working at identifying how different models react against adversarial attack [28]. In order to do that he uses CNN 3d visualizer. He should now able to see what features are identified for these images. By seeing that it facilitates the provision of robustness by potential adversarial perturbations. Thus, he can trust more and make confidence in models when applied to sensitive problems.

We can see below an example inspired from paper Adversarial patch [29] using VGG16. While in the first case top prediction is an orange when the prediction is done using the same image but with a sticker added in the image the top prediction is hair slide (physical attack).

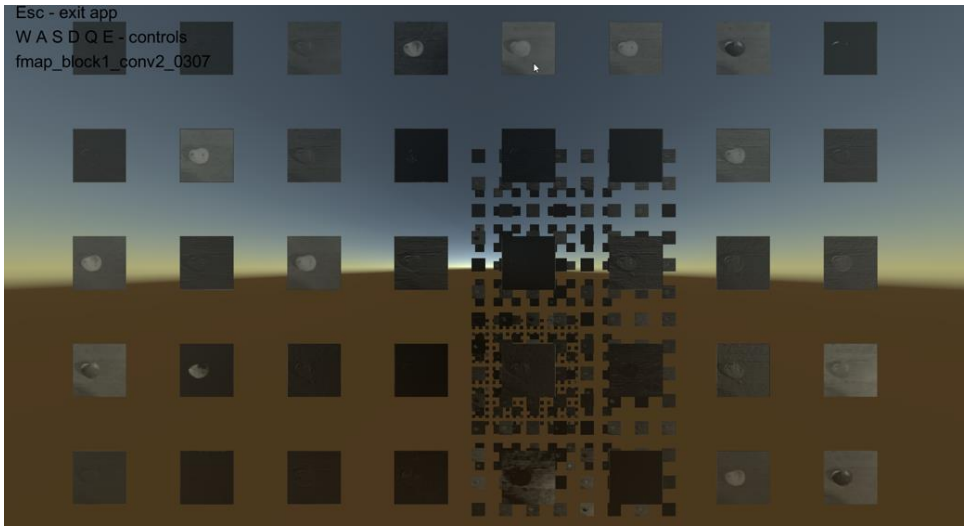


Figure 16: Second layer feature extraction from original image

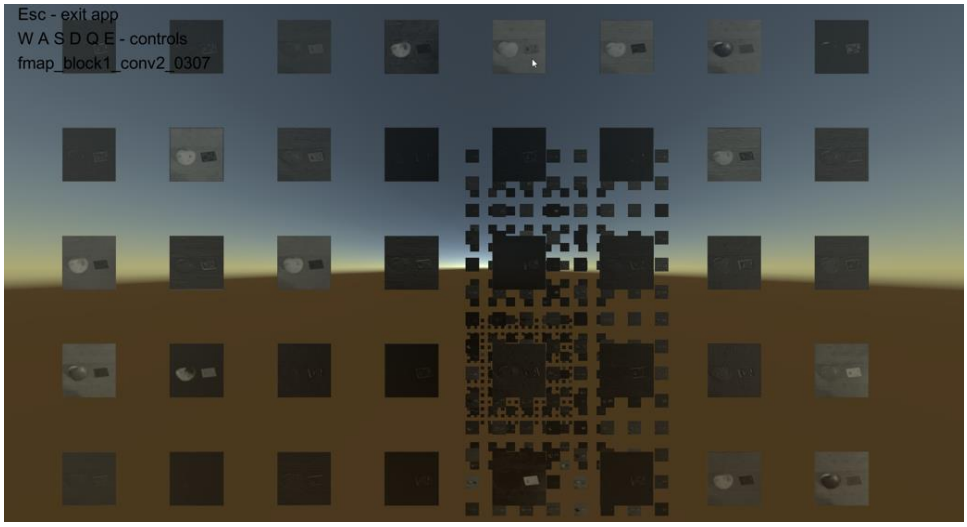


Figure 17: Second layer extraction feature with physical attack

6.5. Data quality of synthetic data generation with GAN

Creating data sets in areas where there is a lack of large labeled training data sets, as in the case of medical system, [30] is an important task. In order to improve CNN classification results by enlarging the training data, Markus is using generative adversarial networks. In order to assess the realism of synthesized images he uses the same technique as paper Colorful Image Colorization [31]. He feds his generated images to a model that was trained on real photos. CNN 3d visualizer might enable him not only to check if the classifier performs are well indicating that the sample is accurate enough but also to check why that is happening by having a visual validation. He should be more confident that the samples generated are truly good and can be used. Below we can see an example of samples that are not good.

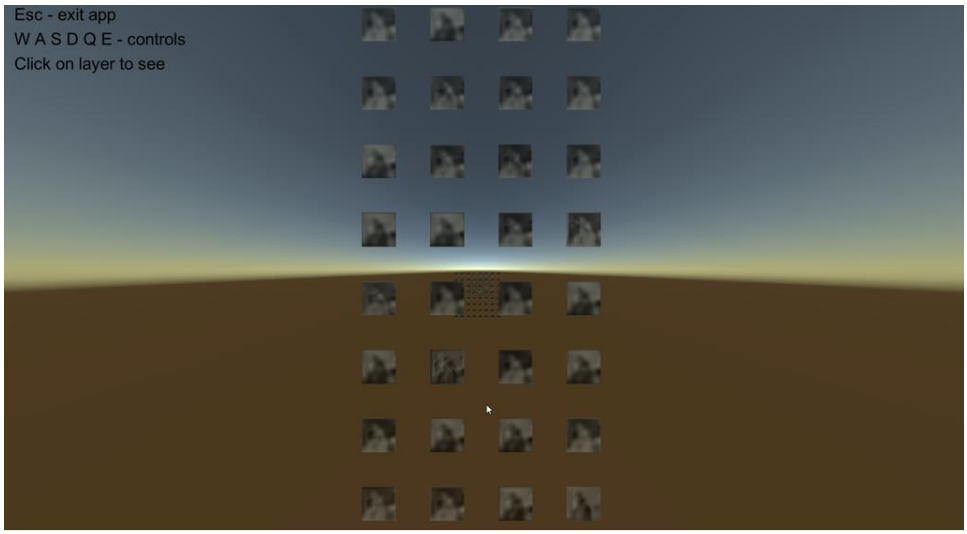


Figure 18: Custom model cannot detect features properly.

6.6. Different inputs response

Based on certain readings such as: *Adversarial Attacks to Scale-Free Networks: Testing the Robustness of Physical Criteria* [32] , *Interpreting adversarial examples by activation*

promotion and suppression [33], *ATMPA: Attacking Machine Learning-based Malware* [34], *COPYCAT: practical adversarial attacks on visualization-based malware detection* [35], we believe that our visualization technique might be beneficial to spot out how the network is responding to different dynamic inputs, therefore the user can test if the model can be fooled by some modified images. The user can test the model in a controlled environment using a predefined set of images saved as a video. This is one of the reasons why we decided to also develop the possibility of adding input from a video.

7. EVALUATION

In the following chapter we will focus on evaluating the results we obtained and the extent they are answering the goals we set at the beginning. In order to make this assessment we will make the analysis based on the usage scenarios be introduced in the previous section.

7.1. Feature visualization

As we presented in the chapter *Challenges and goals* of the current paper one of the goals we wanted to achieve was to create a 3D feature visualization with video as an input or webcam capturing. As you can see in the image below this goal was achieved.

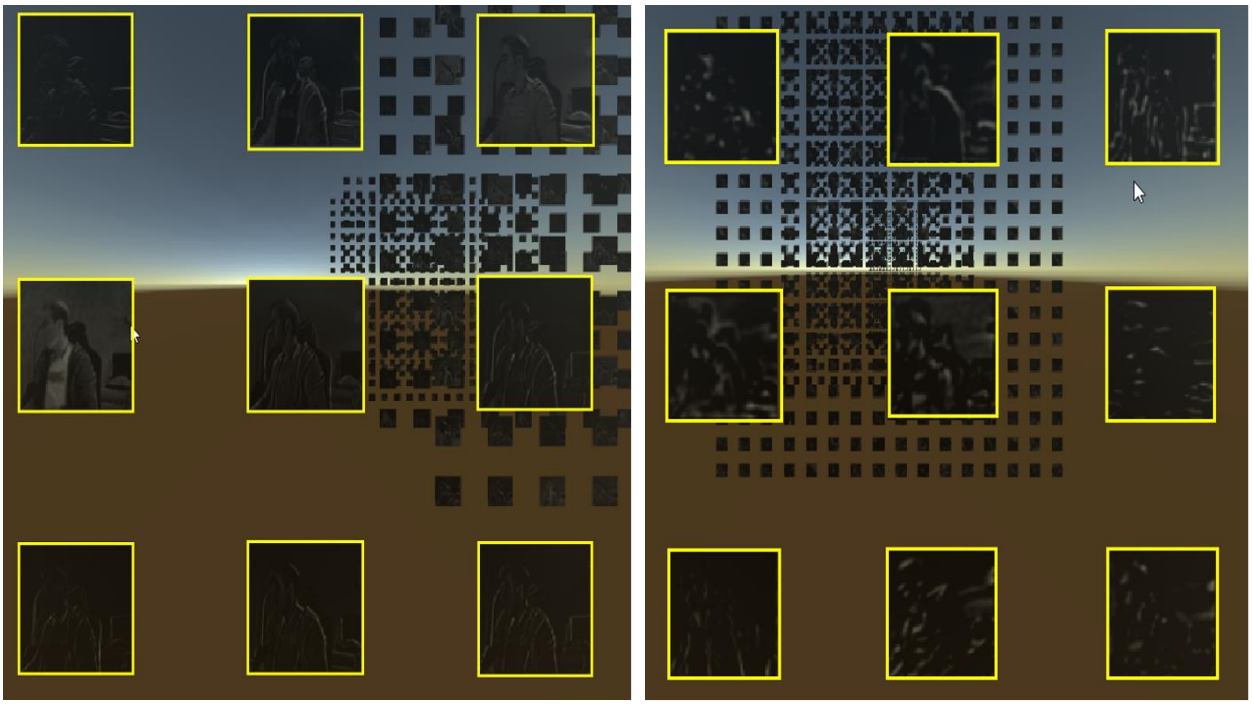


Figure 19: (1) Features in the second conv layer of VGG16; (2) Feature extraction from a layer located in the middle of the networks

In the images above, we used webcam capturing. It is visible that we can see differences between the features of different layers and also the output after applying a filter to a layer. We believe our work will enable a better understanding of the way filters are applied in different scenarios during classifications. Even if we focused on a single way of extracting features, there are also other visualization types the tool can be upgraded to as mentioned in the paper *How convolutional neural network see the world-A survey of convolutional neural network visualization methods* [36].

7.2. Custom models

Based on the undergone research, we have noticed that a key feature was missing from most of the tools presented in papers like: "An Interactive Node-Link Visualization of Convolutional Neural Networks" [14], "Understanding neural networks through deep visualization" [15] or "CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization" [17]. More specifically, we are talking about the difficulty to add custom models or even in some other cases not having this option at all. With our tool we offer the capability to specify where your model is stored in your computer and from there it is automatically loaded into CNN 3D Visualizer. All you have to do is to specify the path (1) to the model and also to specify the input size of the input image (2).

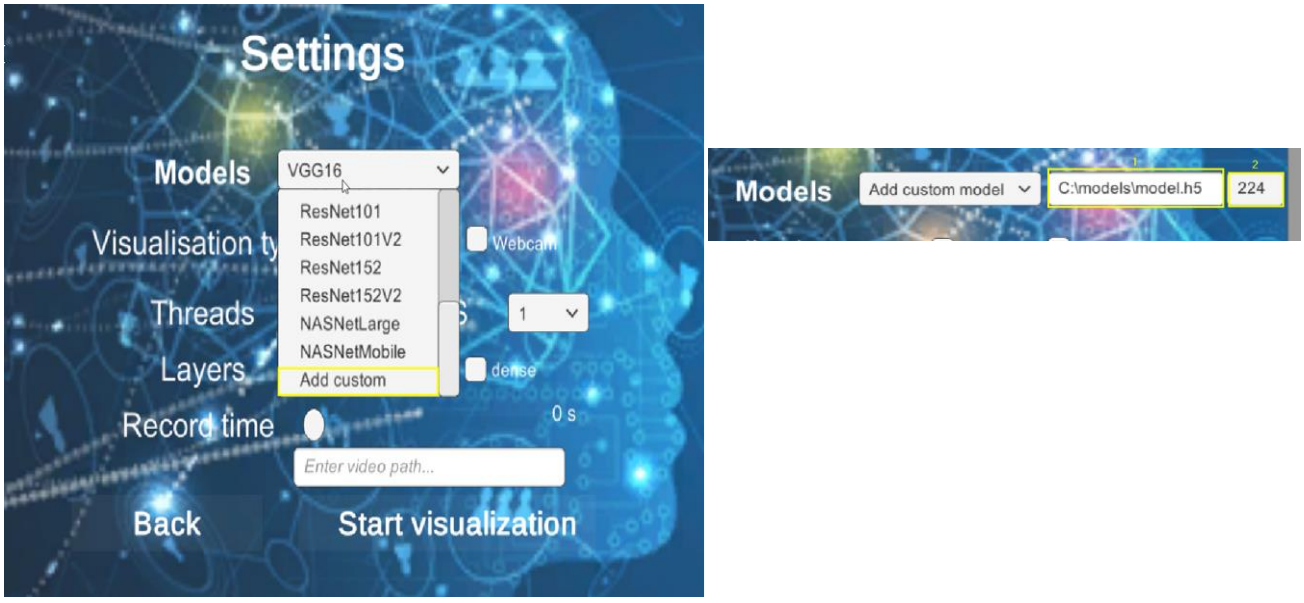


Figure 20: Custom modeling

When testing, by adding in our tool a model created with TensorFlow framework, we discovered that for some models the input layer was not accessible. We further deep dive into the documentation and we found out that when using InputLayer with Keras Sequential model, it can be skipped by moving the input shape parameter to the first layer after the InputLayer[12] which in our current implementation was causing issues. In order to be able to offer the possibility of adding custom models no matter if they are functional or sequential, we added a small sample on how to do the conversion that is attached in (Annex). As we mention in the chapter dedicated to future work, this can be further improved by adding custom models that are created with other frameworks.

7.3. Multiple models

Another important goal we wanted to achieve was to have on one hand a video input stream and webcam capturing, but at the same time also to have multiple models available. This feature is either not offered by other tools as presented in "An Interactive Node-Link Visualization of Convolutional Neural Networks" [14] or it is limited because of the web browser capabilities as in the case of TensorSpace. In this regard, we succeeded in implementing a visualization for the majority of the well-known and used networks.

Models	(Average) Peak performance update realtime approach
VGG16 - 138,357,544 parameters	1 update at 1.17 seconds
VGG19 - 143,667,240 parameters	1 update at 1.20 seconds
Xception – 22,910,480 parameters	1 update at 8.82 seconds
DenseNet121 - 8,062,504 parameters	1 update at 8.02 seconds
DenseNet169 – 14,307,880 parameters	1 update at 10.01 seconds
DenseNet201 - 20,242,984 parameters	1 update at 15.89 seconds
MobileNetV2 - 3,538,984 parameters	1 update at 0.08 seconds
InceptionV3 – 23,851,784 parameters	1 update at 0.99 seconds
InceptionResNetV2 - 55,873,736 parameters	1 update at 2.86 seconds
ResNet50 – 25,636,712 parameters	1 update at 5.60 seconds
ResNet50V2 - 25,613,800 parameters	1 update at 5.05 seconds
ResNet101 – 44,707,176 parameteres	1 update at 9.02 seconds
ResNet101V2 – 44,675,560 parameenters	1 update at 8.67 seconds
ResNet152 – 60,419,944 parameters	1 update at 13.93 seconds
ResNet152V2 – 60,380,648 parameters	1 update at 13.43 seconds
NASNetLarge – 88,949,818 parameters	1 update at 16.25 seconds
NASNetMobile – 5,326,716	1 update at 1.22 seconds

Table 1: Peak performance for each model with number of parameters

[12] https://www.tensorflow.org/api_docs/python/tf/keras/layers/InputLayer

As we can see in the table above, we have these models available that can be started without any prior experience required in model creation. However, some models are performing the frame update at a lower speed than others when real-time webcam capturing is enabled. As presented above this issue is one of the reasons we introduced the recording feature which acts like a buffering period where frames are predicted and stored in a queue for a period of time (maximum 360 seconds) and afterwards shown at a specified framerate. However, we can conclude that we managed to reach the goal of offering a wide range of models available. Real-time visualization for some models was achieved only as an in-time visualization mostly because of the issues that we described in the chapter *Implementation issues*.

7.4. Desktop based implementation

With the client server architecture run locally on the computer, we manage to offer a more consistent user experience. Moreover, it is a and more responsive application than the web counterparts. Also, another plus consists in the fact that for the pretrained models once you run the application for the first time, the models are brought to local machine and after that you will not need internet connection for accessing them. Thanks to this feature we eliminate the loading time for models every time we start the application. Our application allows you to move through the scene while predictions are made and also get the name of the feature that you clicked on. All these are running smoother than in the example of big models' manipulation we have in TensorSpace[13].

[13] https://tensorspace.org/

8. CONTRIBUTION

Having as a starting point the limitations already introduced at the beginning of the current paper in the chapter dedicated to challenges and goal, the aim of our thesis was to implement a system which allows a continuous capturing of webcam, extracts feature maps of the predictions done based on the input image from the live webcam feed and visualize them in a 3d environment in time or real-time. We consider that the usage of a static image allows a detailed investigation of an input, while the usage of a video input highlights the CNN response to dynamic input.

In our thesis, we propose a system which has the following new aspects:

- desktop application for feature visualization in 3D space
- video input from webcam or a video from a local folder
- real-time visualization for small models with 20 fps and 0.3 – 0.5 fps for big models
- record time up to 360 seconds that afterwards can be watched with 1/3/5/7 fps
- capability to use multiple models from tf.keras.applications: MobileNetV2, VGG16, VGG19, Xception, DenseNet121, DenseNet169, DenseNet201, InceptionV3, ResNet50, ResNet101, ResNet152, ResNet50V2, ResNet101V2, ResNet152V2, InceptionResNetV2, NASNetLarge, NASNetMobile.
- the possibility to add your functional models created and saved as h5 using TensorFlow Keras (we also offer an example of convertor from sequential to functional)
- customization of choosing the layers to be visualized: fully connected, convolutional, predictions or all of them at the same time.

We believe that such a tool which enables the understanding of convolutional neuronal networks will be first of all beneficial for newcomers in this field who would like to take advantage of software packages like TensorFlow [37]. Secondly, experts can benefit from this tool by using their own models or transfer learning in order to understand if the filters applied are providing the expected results.

9. FUTURE WORK

3D Visualization in real-time with improved FPS. As presented in the previous chapters our implementation has some limitations when it comes to the size of the models. Some of them like MobileNetV2 can achieve 60fps without recording anything in advance, while VGG16 can only reach a maximum of 1fps without recording. Although we tried to use the best visualization tool, more exactly Unity combined with the best language used for model manipulation, python, the tool has limitations when it comes to sending data from one side to another. This is happening because we have a client-server architecture where we are tied to serializing and deserializing data. We believe that in the future a native implementation created in Unity will be possible and it will eliminate the limitations linked to serialization/ deserialization data.

Offer a 3D visualization of back propagation. CNN 3D Visualizer focuses on having feature visualization in time/ real-time for a pre-trained model and helps on understanding how CNN model transforms the input image into a class prediction. However, we consider that a step forward in this direction will be represented by the visualization of training process. In the future, having the possibility to add back-propagation visualization can be an extraordinary improvement for the CNN visualization that may lead researchers to new heights regarding the way a network training is done.

Extend model compatibility. At this point, our tool offers the possibility to visualize only TensorFlow models. As future improvement, the capabilities of the platform can be extended in order to offer support for other frameworks as well, such as PyTorch.

Model structure. Another aspect that can be improved is the tool's capability of generating models and at the same time respecting their structure. This enhancement could facilitate a better understanding of the way the network layers are positioned.

10. CONCLUSIONS

This thesis presented a new 3D visualization tool for aiding interpreting pre-trained neural networks. The network input used can come from a video or from webcam in real-time or in time. The visualization done with CNN 3D Visualizer displays all the convolutional, dense, fully connected and prediction layers based on layer naming. Having clear goals set from the beginning, we managed to achieve most of the results we strived for.

Our tool addresses to a wide range of users. It will help both experienced researchers and learners as it can be used for a wide range of models, no prior knowledge in model creation being required. Moreover, given the ability of the tool to visualize features based on real-time/time input, it will facilitate the users the possibility to learn about the network's strengths and weaknesses through dynamic interaction.

Furthermore, a key functionality the tool comes with is the option of visualizing besides the pre-existing models we have already integrated in the application, also the users' own pre-trained models created using TensorFlow. This component widens the tool's applicability and makes it more versatile.

Moreover, our tool is a desktop-based application designed to increase the user's experience by eliminating the need of an Internet connection which most of the time slows down the process.

Whereas previous visualizations have only described non-interactive or toy-sized neural networks, the current work demonstrates that practical vision or pre-trained convolutional networks can be effectively depicted in a 3D visualization with dynamic input.

We hope our work will make the understanding of CNN easier and it will also inspire future researchers to bring it to the next level, expend its capabilities, create better visualization options and widen up its applicability.

REFERENCES

[1] R. Kovvuri, R. Nevatia and C. G. Snoek, "Segment-based models for event detection and recounting," *23rd International Conference on Pattern Recognition (ICPR),* pp. 3868-3873, 2016/12/4.

[2] N. B. Lonkar and D. B. Hanchate, "Automatic Visual Concept Detection in Videos," *OPEN ACCESS INTERNATIONAL JOURNAL OF SCIENCE & ENGINEERING* , 2017/07.

[3] Y. Kao, R. He and K. Huang, "Deep aesthetic quality assessment with semantic information," *IEEE Transactions on Image Processing,* vol. 26, pp. 1482-1495, 2017/1/11.

[4] B. H. Kang, "A Review on Image & Video Processing," *International Journal of Multimedia and Ubiquitous Engineering,* 2007/05.

[5] M. Gelautz, E. Stavrakis and M. Bleyer, "Stereo-Based Image and Video Analysis for Multimedia Applications," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences (XXth ISPRS Congress),* 2004/01.

[6] M.-L. Zhang and Z.-H. Zhou, "A Review On Multi-Label Learning Algorithms," *IEEE Transactions on Knowledge and Data Engineering,* pp. 1819-1837, 2014/08.

[7] K. T. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *arXiv preprint arXiv:1511.08458,* 2015/11/26.

[8] F. Sultana, A. Sufian and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network," *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN),* pp. 122-129, 2018.

[9] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional.," *Proceedings of the 25th International Conference on Neural Information Processing Systems,* vol. I, pp. 1097-1105, 2012.

[10] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 770-778, 2016.

[11] H. Fang, S. Gupta, F. Iandola, R. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. Platt, L. Zitnick and G. Zweig, "From Cations to Visual Concepts and Back," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1473-1482, 2015/6/1.

[12] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 580-587, 2014.

[13] Z. C. Lipton, "The Mythos of Model Interpretability," *Communications of the ACM (CACM),* 2016/6.

[14] A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks," *International Symposium on Visual Computing,* pp. 867-877, 2015/12/14.

[15] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs and H. Lipson, "Understanding neural networks through deep visualization," *ICML Deep Learning Workshop,* 2015/6/22.

[16] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," *International Journal of Computer Vision,* pp. 1-24, 2019/10/11.

[17] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng and D. H. Chau, "CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization," *arXiv preprint arXiv:2004.15004,* 2020/4/30.

[18] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *European conference on computer vision,* pp. 818-833, 2014/9/6.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 4510-4520, 2018.

[20] G. Huang, Z. Liu, G. Pleiss, L. V. D. Maaten and K. Weinberger, "Densely connected convolutional networks," *EEE Conference on Pattern Recognition and Computer Vision (CVPR),* 2017.

[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker and P. W. M. W. Y. Y. X. Z. V. Vasudevan, "Tensorflow: A system for large-scale machine learning," *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265-283, 2016.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, pp. 8026-8037, 2019.

[23] M. Kahng, N. Thorat, D. H. Chau, F. B. Viégas and M. Wattenberg, "GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 310-320, 2019/1.

[24] J. C. Hamerski, A. R. Domingues, F. G. Moraes and A. Amory, "Evaluating serialization for a publish-subscribe based middleware for MPSoCs," *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 773-776, 2018/12/9.

[25] B. Petersen, H. Bindner, S. You and B. Poulsen, "Smart grid serialization comparison: Comparision of serialization for distributed control in the context of the Internet of Things," *2017 Computing Conference*, pp. 1339-1346, 2017/7/18.

[26] S. Blackman, Beginning 3D Game Development with Unity 4: All-in-one, Multi-platform Game Development, 27 August, 2013.

[27] K. Shatri and K. Buza, "The Use of Visualization in Teaching and Learning Process for Developing Critical Thinking of Students," *European Journal of Social Science Education and Research*, vol. 4, no. 1, pp. 71-74, 2017/1/21.

[28] K. Xu, S. Liu, G. Zhang, M. Sun, P. Zhao, Q. Fan, C. Gan and X. Lin, "Interpreting adversarial examples by activation promotion and suppression," *arXiv preprint arXiv:1904.02057*, 2019/4/3.

[29] T. B. Brown, D. Mané, A. Roy, M. Abadi and J. Gilmer, "Adversarial patch," *arXiv preprint arXiv:1712.09665*, 2017/12/27.

[30] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger and H. Greenspan, "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification," *Neurocomputing*, vol. 321, pp. 321-331, 2018/12/10.

[31] R. Zhang, P. Isola and A. Efros, "Colorful Image Colorization," *European Conference on Computer Vision (ECCV)*, 2016.

[32] Q. Xuan, Y. Shan, J. Wang, Z. Ruan and G. Chen, "Adversarial Attacks to Scale-Free Networks: Testing the Robustness of," *arXiv:2002.01249v1*, 4 Feb 2020.

[33] K. Xu, S. Liu, G. Zhang, M. Sun, P. Zhao, Q. Fan, C. Gan and X. Lin, "Interpreting adversarial examples by activation promotion and suppression," *arXiv preprint arXiv:1904.02057*, 2019/4/3.

[34] X. Liu, J. Zhang, Y. Lin and H. Li, "ATMPA: Attacking Machine Learning-based Malware," *arXiv:1808.01546v3 [cs.CR]*, 30 Dec 2019.

[35] A. Khormali, A. Abusnaina, S. Chen, D. Nyang and A. Mohaisen, "COPYCAT: practical adversarial attacks on visualization-based malware detection," *arXiv preprint arXiv:1909.09735*, 2019/9/20.

[36] Z. Qin, F. Yu, C. Liu and X. Chen, "How convolutional neural network see the world-A survey of convolutional neural network visualization methods," *Journal of Mathematical Foundations of Computing*, vol. 1, p. 149–180, 2018/5.

[37] P. Goldsborough, "A tour of tensorflow," *arXiv preprint arXiv:1610.01178*, 2016/10/1.

[38] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017/4/17.

[39] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, pp. 2672-2680, 2014.

[40] A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, pp. 5455-5516, 2020/12.

[41] K. R. Mopuri, U. Garg and R. V. Babu, "CNN fixations: an unraveling approach to visualize the discriminative image regions," *IEEE Transactions on Image Processing*, pp. 2116-2125, 2018/11/16.

ANNEX

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']


model = models.Sequential()
model.add(layers.InputLayer(input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), padding='same'))
model.add(layers.Conv2D(32, (3, 3), padding='same'))
model.add(layers.Activation('relu', name="activation" ))
model.add(layers.Conv2D(32, (3, 3)))
model.add(layers.Activation('relu', name="activation2"))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.25, name="drop"))

model.add(layers.Conv2D(64, (3, 3), padding='same'))
model.add(layers.Activation('relu', name="activation3"))
model.add(layers.Conv2D(64, (3, 3)))
model.add(layers.Activation('relu', name="activation7"))
model.add(layers.Conv2D(64, (3, 3)))
model.add(layers.Activation('relu', name="activation8"))
model.add(layers.Conv2D(64, (3, 3)))
model.add(layers.Activation('relu', name="activation4"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), name="pooling"))
model.add(layers.Dropout(0.25, name="drop2"))

model.add(layers.Flatten())
model.add(layers.Dense(512))
model.add(layers.Activation('relu', name="activation5"))
model.add(layers.Dropout(0.5 , name="drop3"))
model.add(layers.Dense(10))
model.add(layers.Activation('softmax', name="activation6"))


input_layer = layers.Input(batch_shape=model.layers[0].input_shape)
prev_layer = input_layer
for layer in model.layers:
   prev_layer = layer(prev_layer)

funcmodel = models.Model([input_layer], [prev_layer])

funcmodel.compile(optimizer='adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy'])
```

```
history = funcmodel.fit(train_images, train_labels, epochs=1,
                validation_data=(test_images, test_labels))

test_loss, test_acc = funcmodel.evaluate(test_images,  test_labels, verbose=2)

funcmodel.save('model.h5')
```