Identifying Deepfake Faces with ResNet50-Keras using Amazon EC2 DL1 Instances powered by Gaudi Accelerators from Habana Labs

Saket Pradhan
Department of Information Technology
University of Mumbai
Mumbai, India
saketspradhan@gmail.com

Raj Shah
Department of Information Technology
University of Mumbai
Mumbai, India
rajshah.rutu@gmail.com

Ranveer Shah
Department of Information Technology
University of Mumbai
Mumbai, India
ranveer2001s@gmail.com

Anuj Goenka
Department of Computer Engineering
University of Mumbai
Mumbai, India
anujgoenka06@gmail.com

***Abstract*—With the emergence of deepfake technology, it has become exponentially more difficult to identify real content from the artificially generated on social media. To counter the ill-effects of online deepfake content, we propose an application to predict if a given facial image is real or generated virtually via deepfake technology. We train a custom ResNet50-Keras model on the new AWS EC2 DL1 Instances powered by Gaudi accelerators developed by Habana Labs (an Intel company) and integrate the instance with Amazon's S3 bucket. With the model saved as an h5 file, we make a RestAPI using FastAPI. The API takes an input image, converts it into a JSON request, and passes it to the backend. Faces extracted from these images are passed through various pre-processing methods and finally to the model that classifies them to be either a deepfake or not and generates a face mesh accordingly. The model combines the processed faces into a single image sent back as a response to the client that changes the stateHooks to display the desired result. Further, we have summarized the results obtained when detecting such manipulated images.**

Keywords—habana, gaudi, aws ec2 dl1, resnet50, faceforensics, tensorflow, keras, deepfake, mediapipe, facial landmark, react, fastapi, restful api

I. Introduction

With the advent of deepfake technology, false images and videos have been spreading through social media and the internet. Deepfake is a technique using which videos and images can be manipulated using various deep learning techniques. Using deepfake techniques, one can create fake news, spread propaganda, defame people or engage in other malicious activities. When counterfeit recordings turn into a web sensation, individuals tend to accept them as they are without questioning their integrity since they appear normal to the human eye and continue exporting them to other people. This process makes a vicious cycle of content propagation online. As a proposed solution to this issue, our application takes facial images sourced from various social media sites, the user's device, or from the web in general, checking if it is a manipulated image or not. The mechanisms for creating deepfakes include building deep learning models such as Autoencoders and Generative Adversarial Networks (GANs), which have been researched extensively and applied widely in the computer vision domain. These models are used to study the facial textures, expressions, and movements of people and synthesize false facial images using the collected information. Our proposed application not only detects the morphing of a facial image but also draws a mesh around the parts where the model feels the subject has been manipulated or photoshopped.

II. Software and Hardware Used

A. AWS EC2 DL1 Instances and Habana-Gaudi Accelerators

Deep learning is becoming more mainstream as of late, as many consumers have been realizing its tangible implications on their businesses—from the deployment of these deep learning models in the cloud at scale to building entire enterprises based on them. The task of maintaining a high standard, accuracy and, quality of such models is a rather difficult endeavor, while engineers are required to re-train and fine-tune the models frequently; thus, requiring a considerable amount of high performance and cost-efficient computing resources, thereby resulting in an increase in the infrastructure costs, which may be prohibitive for a lot of consumers.

Amazon Web Services (AWS) had recently released the 'Amazon Elastic Compute Cloud' (Amazon EC2) DL1 instances on October 26th, 2021; a new type of instance expressly designed for training deep learning applications. The new Amazon EC2 'DL1 Instances' powered by 'Gaudi' accelerators from Habana Labs (an Intel company) are capable of delivering extremely low cost-to-train deep learning models for object detection and classification, image recognition, natural language processing, recommendation and personalization engines, fraud detection, business forecasting, intelligent document processing, etc. use cases while providing upwards of 40% better price-performance for the training of deep learning models and applications as compared against current-generation GPU-based EC2 instances on AWS. These are now available without any upfront commitments on-demand in a 'pay-as-you-go' usage model.

2022 IEEE Region 10 Symposium (TENSYMP) | 978-1-6654-6658-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/TENSYMP54529.2022.9864501

With the new ‘EC2 DL1 Instances’ being planned to be introduced by AWS for general availability, AWS will further be able to reduce the time and cost for training deep learning datasets while also decreasing the overall cost of operations for consumers. The ‘EC2-DL1 Instances’ use the ‘Habana-Gaudi’ accelerators, the sole purpose of which is to accelerate the training for deep learning models by providing a much greater computing efficiency at a lesser cost than general-purpose GPUs available earlier.

The ‘EC2-DL1 Instances’ currently include up to 8 Habana-Gaudi accelerators, 768 GB of system memory, 256 GB of high-bandwidth memory (HBM) for each accelerator, the 2nd generation of Amazon’s custom ‘Intel Xeon’—Scalable (Cascade Lake) Processors, upwards of 400 Gbps of networking throughput, and local NVMe storage of up to 4 TB. The ‘EC2-DL1 Instance’ is an eight-card EC2 Habana-Gaudi instance capable of processing 12,000 images every second whilst training the Keras ResNet50 model from the TensorFlow Model Garden. Every single Habana-Gaudi processor integrates about 32GB of HBM2 memory (Generation 2 of High-Bandwidth Memory). For inter-processor connectivity inside the server, these processors also feature RoCE on-chip integration. The 'AWS Elastic Fabric Adapter' (EFA) allows to scale across multiple servers, thereby allowing consumers to expand the use of several Habana-Gaudi systems seamlessly for more productive, scalable, and efficient distributed training of deep learning models.

The new ‘EC2-DL1 Instances’ come along with the Habana SynapseAI® SDK, which is integrated with deep learning frameworks of PyTorch and TensorFlow. This can help consumers easily migrate code onto the new ‘EC2-DL1 Instances’, with minimal changes, if required. The SynapseAI software suite has the Tensor Processor Core (TPC) kernel library, the SynapseAI Profiler, graph compiler, drivers, firmware, runtime, and the developer tools (for example the TPC SDK used for the development of custom kernels) from HabanaAI.

The ‘EC2 DL1 Instances’ can be launched with the ‘AWS Deep Learning AMIs’ or with the ‘Amazon Elastic Kubernetes Service (Amazon EKS)’ or the ‘Amazon Elastic Container Service (Amazon ECS)’ for more containerized applications. Alternatively, the ‘EC2-DL1 Instances’ can also be accessed through ‘Amazon SageMaker’, thus using edge computing to make the process of training and deploying even faster in the cloud. The ‘EC2-DL1 Instances’ greatly advantage due to the ‘AWS Nitro System’; building blocks that discharge the conventional virtualization functions to the relevant hardware and software to give better security, availability, and performance. As of now, the ‘EC2-DL1 Instances’ are operable in the US East (North Virginia) and US West (Oregon) AWS Regions.

B. ResNet50 Keras Model

Whilst the quantity of stacked layers might enhance the model's features, a deeper network can reveal the frequently occurring degradation issue [1]. Simply, as the number of layers in a neural network grows, the accuracy may stagnate, become saturated and gradually decline. As a result, the model's performance degrades on both training and test data. Overfitting is not the cause of this degeneration [2, 3]. It could be caused by the network's initialization, optimization algorithm, or, more crucially, the issue of vanishing (because the gradient is carried back to older levels, repeated multiplication may result in a very small gradient) or exploding gradients. This problem exists in VGG architecture, hence the reason we chose to go with ResNet50 instead.

ResNet50-Keras solves this challenge by creating a deep residual learning framework, which uses skip connections to execute identity mappings. They expressly allowed the layers to fit a residual mapping and denoted it as H(x), and they explicitly allowed the nonlinear layers to fit another mapping F(x):=H(x)x, resulting in H(x):=F(x)+x, as shown in Figure 1. There are two ways in which skip connections help:

- They avoid the issue of vanishing gradient by enabling gradient to flow along this other shortcut channel.
- They enable the model to learn an identity function that guarantees that the deeper layer performs at least as well as the upper layer, if not better.

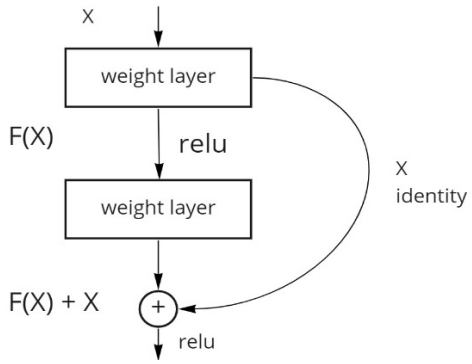

Fig. 1. Layers fit a residual mapping denoted as H(x), while explicitly allowing the non-linear layers to fit another mapping F(x):=H(x)x, resulting in H(x):=F(x)+x.

As a result, ResNet50-Keras boosts the performance of deep neural networks by adding more layers while lowering the error rate. To put it another way, skip connections combine the outputs of previous layers with the outputs of stacked layers.

The ResNet50-Keras model is divided into five stages, each with its own convolution and identity block. There are three convolution layers in each convolution block, and three convolution layers in each identity block. The architecture of ResNet50 contains the following layers as shown in Figure 2.

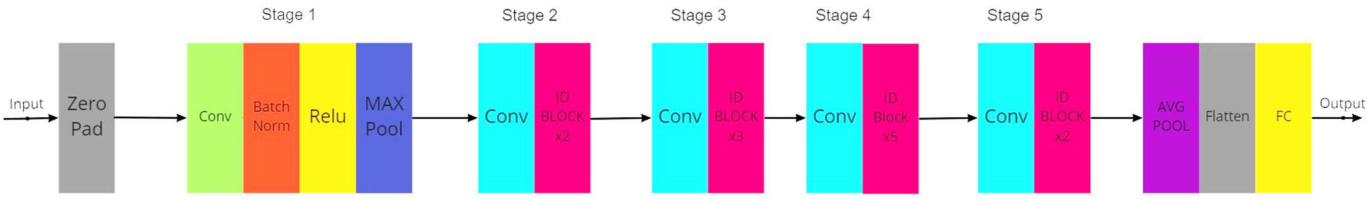

Fig. 2. ResNet50-Keras model divided into 5 stages; each with its own convolution and identity block, 3 convolution layers in each convolution block, and three convolution layers in each identity block.

- We get one layer from a convolution with a kernel size of 7 * 7 and 64 distinct kernels, all with a stride of size 2.
- Following that, we have max pooling with a stride size of 2.
- There is a 1 * 1,64 kernel in the next convolution, followed by a 3 * 3,64 kernel, and finally a 1 * 1,256 kernel. These three layers are repeated three times in total, giving us nine layers in this phase.
- This phase was repeated four times, giving us 12 layers. Next, we see a kernel of 1 * 1,128 followed by a kernel of 3 * 3,128 and finally a kernel of 1 * 1,512.
- Then there's a 1 * 1,256 kernel, followed by 3 * 3,256 and 1 * 1,1024 kernels, which are repeated six times for a total of 18 layers.
- Then a 1 * 1,512 kernel was added, followed by two more 3 * 3,512 and 1 * 1,2048 kernels, for a total of nine layers.
- After that, we run an average pool and finish with a fully linked layer with 1000 nodes, followed by a SoftMax function, giving us one layer.

As a result, we get a Deep Convolutional network with 1 + 9 + 12 + 18 + 9 + 1 = 50 layers.

The model uses the LARS (Layerwise Adaptive Rate Scaling) optimizer which is used to split the model across GPUs and processors and compute the loss on a fragment of the training data for each copy. This could have been done using the Stochastic Gradient Descent (SGD) optimizer as well, but it was observed that the number of iterations per epoch reduces as the batch size grows [4]. We can adjust by altering the learning rate to coalesce in the same amount of dataset iterations. However, as the rate of learning increases, the training becomes more uncertain [5]. To prevent this divergence, the LARS optimizers is used.

LARS is a technique for large-batch optimization, with a couple of key differences between LARS and other similar adaptive algorithms like Adam or RMSProp: firstly, LARS has a separate learning rate for every individual layer and not for every weight, equation (1), and secondly, the magnitude of the update is controlled with respect to the weight norm for better control of training speed, equation (2).

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)(g_t + \lambda x_t) \quad (1)$$

$$x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(||x_t^{(i)}||)}{||m_t^{(i)}||} m_t^{(i)} \quad (2)$$

The proportion between the norm of the layers' weights and the norm of gradients' update was used to assess training stability with large learning rate. If this percentage is too high, the training may become unstable, according to our findings. If the ratio is too tiny, the weights do not change quickly enough. This proportionality can be explained by equation (3).

$$\lambda^l = \eta \times \frac{\|w^l\|}{\|\nabla L(w^l)\|} \quad (3)$$

The learning rate (LR) λ is the global learning rate η multiplied by the ratio of the layer weights' norm to the layer gradients' norm. We may just add weight decay to the denominator if we use it. When we input this into SGD optimizer, the denominator normalizes the gradients to have unit norm, preventing divergence.

C. Dataset Used

We the FaceForensics++ dataset [6] to train the model, which is over an order of magnitude larger than comparable, publicly available, forgery datasets. We perform a thorough analysis of data-driven forgery detectors. There are two types of facial manipulation methods currently in use: facial expression manipulation and facial identity manipulation. Face2Face is one of the most well-known ways for manipulating facial expressions. It allows for the real-time transfer of facial emotions from one individual to another using only common hardware. The second type of facial forgery is identity manipulation. Instead of changing features, these methods swap out a person's face for that of another. We show in this paper that we can detect such modifications automatically and reliably, outperforming human observers by a large margin. We take advantage of recent breakthroughs in deep learning, particularly convolutional neural networks' ability to learn incredibly powerful visual features. We solve the detection challenge by using supervised learning to train a neural network. To do this, we create a large-scale dataset of alterations using the traditional computer graphics-based methods Face2Face and FaceSwap, as well as the learning-based techniques DeepFakes and NeuralTextures.

All four approaches require the input of source and target actor video pairs. Each method's result is a video made up of created images. We also build ground truth overlays that indicate whether a pixel has been manipulated or not, which can be utilized to train forgery localization techniques, in addition to the manipulation output.

'FaceSwap' is a graphical fidelity technique for transferring the facial region from one video to another. The face region is extracted using sparsely identified facial markers. The method uses 'blendshapes' to fit a 3-dimensional template model using these landmarks. Using the textures from the input image, this model is back projected to the target object, reducing the disparity between the projected shape and the localized landmarks. Finally, the image is mixed with the rendered model, and colour correction is applied. Until one video ends, we repeat these processes for all combinations of source and target frames. The implementation is computationally light and can be run on the CPU effectively. 'Deepfake' has become a catch-all term for deep learning-based face replacement. A face from a given video or image file is used to substitute a face from a target sequence. The method is based on the training of two autoencoders [7] using a shared encoder to rebuild training images of the sources and target faces, respectively. Cropping and aligning the photos is done with the help of a face detector. The trained encoder and decoder of the source face are applied to the target face to create a false image. Using Poisson image editing, the autoencoder output is merged with the picture.

Face2Face [8] is a facial reconstruction system that transfers a source video's expressions to a target clip while keeping the target person's identity. The initial solution used two video feed streams with keyframe selection done manually. These frames are utilized to create a dense face reconstruction that can be used to re-synthesize the face under various lighting and expressions. We use the Face2Face technique to fully automate the creation of reenactment alterations for our video database. We use the first frames to build a transient face identity (i.e., a computer model) and monitor the expressions over the subsequent frames in a preprocessing pass for each film. 'NeuralTextures' learns a neural structure of the target individual, including a rendering network, from the original video data. A photometric reconstruction loss is combined with an adversarial loss to train this. A comparative study of the aforementioned techniques is given in Table 1.

TABLE I. COMPARITIVE STUDY OF COMMONLY USED TECHNIQUES

Algorithm	**Techniques Used**[a]			
	Deep Fake	***Face 2 Face***	***Neural Textures***	***Pristine***
Predict Fake	0.973	0.847	0.820	0.894
Resnet50	0.527	0.504	0.387	0.786
Inception	0.655	0.350	0.333	0.768
Efficient Net	0.709	0.445	0.387	0.674

[a.] Data sourced from [9]

III. IMPLEMENTATION

A. Google MediaPipe

Google's MediaPipe Face Mesh is a key that calculates 468 3D face landmarks in real time and is even capable on mobile phones. It utilizes machine learning to infer a 3D facial-surface, needing only one camera input without needing any dedicated depth-sensors. Using a light-weight model architecture alongside GPU-acceleration all through the entire pipeline, this solution provides critical performances in real time. Further, the arrangement is packaged with the 'Face Transform' module which is able to overcome any possible issues arising between 'facial landmark estimation' and helpful 'Augmented Reality 'use cases. It sets a metric 3D space and utilizes the facial landmark screen positions to evaluate a facial transformation inside that space. This facial transformation data comprises common 3D-primitives, which comprise of a facial posture transformation matrix as well as a triangular-shaped face mesh. 'Procrustes Analysis', a light-weight statistical analysis tool is used to drive a powerful logic system. This analysis computes on a CPU with minimum speed and memory on top of the ML model surmising. This ML pipeline comprises two real-time deep neural network models (DNNs) working simultaneously:

- A detector working on the entire picture and works on the facial features.
- A 3D-facial-landmark model working on these locations predicting the approximate 3D surface using regression.

Having the facial features accurately cropped diminishes the requirement for data augmentations such as affine modifications that include scaling, rotating, and translation changes. Rather, it redirects the network to commit a large portion of its capacity to accurately predicting the coordinates. Additionally, the cropped images can be developed based on the facial landmarks that are recognized in the previous frame in this pipeline. When this landmark model is longer able to identify the presence of any facial features, the face detector module is called to find and re-centre the face inside the frame.

This pipeline is carried out as a MediaPipe graph that utilizes a facial landmark subgraph from the facial landmark module and commences rendering the utilization of a committed face renderer subgraph. The facial milestone subgraph internally utilizes a facial detection subgraph from the facial recognition module. For 3D-facial-landmarks, transfer learning is used to train a network with multiple objectives: The network is able to predict 3D-landmark coordinates simultaneously on the artificially rendered data and 2D semantic-contours on the labelled real data. The subsequent network provides a sensible 3D-landmark prediction on artificially generated as well as real data. A video frame is cropped and is passed to the 3D-landmark network. The model outputs the following:

- positions of the 3D-points and their facial positions.

- the percentage probability of the presence of a face and its alignment with the background.

B. FastAPI backend

For implementing our deepfake detection model as a Web application we use FastAPI as the backend to create RestAPI for the model and React JS as the frontend. FastAPI is a modern framework that allows us to build API seamlessly without much effort and time. In contradiction to Flask, FastAPI is built over Asynchronous Server Gateway Interface (ASGI) instead of Web Server Gateway Interface (WSGI), which makes it faster with lower latency. Due to the ASGI, FastAPI supports concurrency/asynchronous code by declaring the endpoints with *async def* syntax. Figure 3 shows the FastAPI connections.

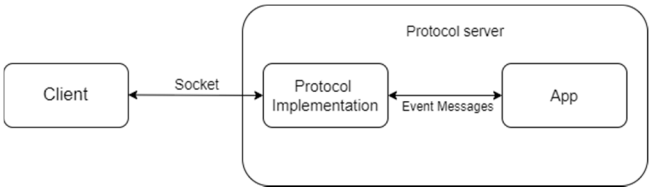

Fig. 3. FastAPI connection between the web client and the model.

The ASGI server we use is that of *Uvicorn*. *Uvicorn* fills the gap of Python that lacked a minimal low-level server/application interface for async frameworks. FastAPI has a built-in data validation system that can detect invalid datatype during the run and returns the reason for improper input in JSON format. Fast API uses 'Pydantic' for data validation, something that flask lacks. Upon deploying with FastAPI Framework, it generates documentation and creates an interactive GUI (Swagger UI) that enables developers to test the API endpoints more conveniently.

FastAPI is chosen considering these three main concerns:

- Speed of Operation
- Developer Experience
- Open Standards

To use our TensorFlow model with FastAPI we save it as a .h5 file. For accepting images into the API from the frontend we create a post request "/predict" and use the "UploadFile" method from FastAPI. The uploaded image is converted into a NumPy array which is then converted into a cv2 object and furthermore passed as a NumPy float32 object.

To make the cv2 object of the image ready to be processed by the model we grayscale it. This grayscale image is passed to the Haar-Cascade front face detection which in turn detects individual faces within the image. Initially, an axis is formed based on the image, its transition, rotation with a defined focal point and principal point to detect faces in three-dimensional space. An abounding box is created with its coordinates saved around these individual faces and they are segregated from the main image as individual objects. Cv2 objects of these individual faces are normalised and resized to a size of 244 x 244. With images normalised and resized, they are temporarily stored parsed into our Deep-fake detecting deep learning model.

The deep learning model classifies whether the parsed image is a deepfake or not. If the face is detected to be a deepfake based on the probabilistic output it is first colourised to RGB and passed to the landmark detection method. Facial landmark detection helps in detecting and tracking key points from a human face. These facial landmarks will be used to create mesh on the deepfake/morphed images.

To detect landmarks and draw the mesh, we use Google's MediaPipe and matplotlib. The facial landmarks detected in the faces are plotted as circles and these points are connected to form a face mesh. We further optimize the mesh by modifying the thickness, opacity and colour of the connecting lines and the landmark points. This process is repeated till face mesh is drawn on each and every individual face detected as a deep fake. The deep-fake images are appended into a NumPy array. For an uploaded image that contains multiple faces, we need to reconstruct the image with the deepfake faces having mesh over them. For this, we loop through the NumPy array storing the deepfake faces and overlay them on their original counterpart in the main image using the corresponding bounding box coordinates stored earlier. Through this, we overlay the deepfake face with a mesh drawn over the same face without the mesh drawn over it. This results in the formation of a single image similar to the original uploaded image with the deepfake faces detected.

With the image reconstructed with the deepfake faces detected, we encode the cv2 object to PNG format. This PNG image can be returned from the post request as a Streaming Response using "io.BytesIO". For better response time we encode and return the image as base64 from the post request of the API. We finalize our API by allowing Cross-Origin Resource Sharing (CORS) since our frontend runs on JavaScript.

C. React JS frontend

React is a JavaScript-based frontend development library. It becomes easier to develop dynamic web applications using it. React JS is chosen considering these three main advantages:

- Component based reusability of code
- Multiple UI libraries supporting React
- Better performance with virtual DOM and functionalities like hooks

We create and setup our react app with the help of npx-create-react-app we install react-router-dom for routing between various components. For the ease of request response, we use the 'Axios' package. We define an empty state of title, content, image and response image. The upload button on submit invokes the handleSubmit event. The handleSubmit event accepts the image and stores it in an image state with title state storing the file name. Using the FormData() the image file is sent as a request to the "/predict" endpoint of the API. After the API process the uploaded image and returns the base64 type image. The base64 data from the response image state is passed to Buffer and toString methods and stored in response_image state. The data in the response image state is rendered in the div using <img> tag with the specified ID.

IV. Results and Conclusion

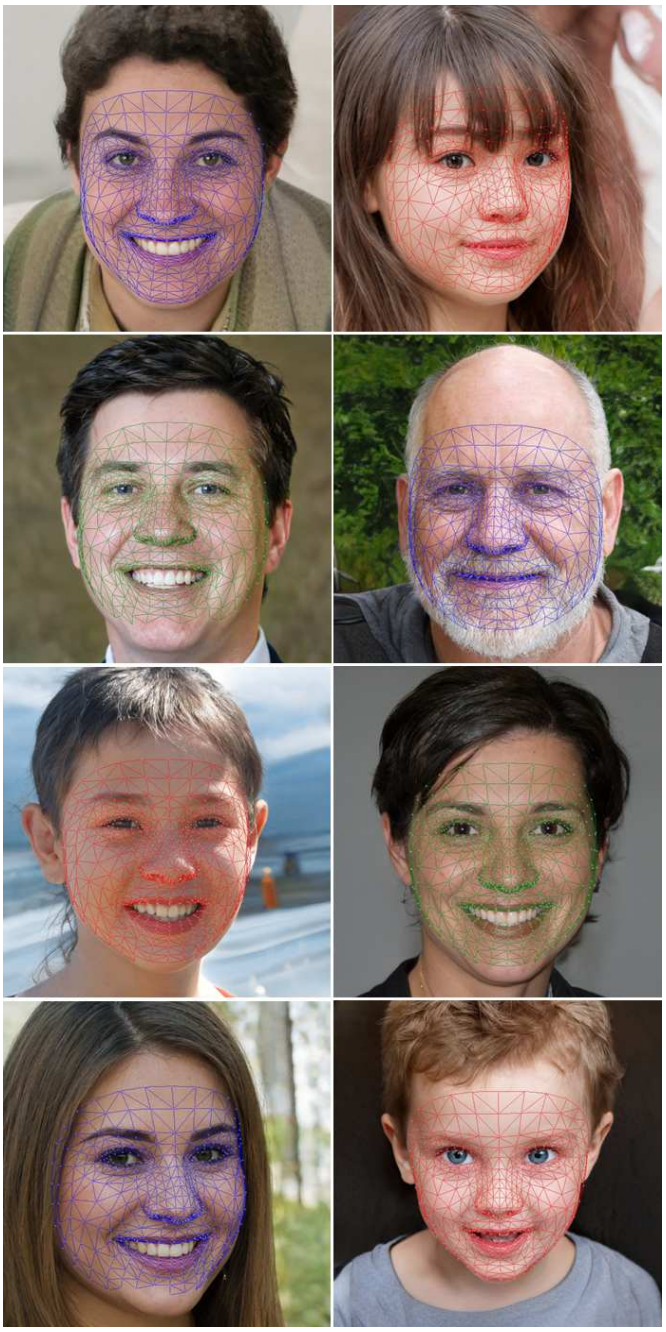

Fig. 5. Single-face deepfake image detected by the model.

We download images from [10] which gives a GAN generated deepfake. These images are then uploaded to our web-application client. The image gets stored in state and is passed to the deep learning model's Rest API. Upon processing the backend sends the output image to our frontend which updates our state and triggers the render method on our client. We observe a mesh created on the deepfake which concludes it is able to detect GAN [11] generated deepfakes. To further test our model. We upload an image with multiple faces and our web interface accepts it. The model segregates and processes every individual face and classifies them. Each classified face is again combined to form the original image with deepfakes having a face mesh over them. Figures 5 and 6 show the output generated by the model when a deepfake, photoshopped or manipulated images are detected.

Fig. 6. Multi-face deepfake images detected by the model. Only deepfake faces have face mesh drawn on them. Real faces remain intact.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun(2015). Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.

[2] Koonce, Brett. (2021). ResNet 50. 10.1007/978-1-4842-6168-2_6.

[3] Nguyen, Luong & Lee, Kwangjin & Shim, Byonghyo. (2021). Stochasticity and Skip Connection Improve Knowledge Transfer. 1537-1541. 10.23919/Eusipco47968.2020.9287227.

[4] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 661–670. ACM, 2014.

[5] Yang You, Igor Gitman, Boris Ginsburg (2017). Large Batch Training of Convolutional Networks. arXiv:1708.03888.

[6] Rössler, Andreas & Cozzolino, Davide & Verdoliva, Luisa & Riess, Christian & Thies, Justus & Nießner, Matthias. (2019). FaceForensics++: Learning to Detect Manipulated Facial Images.

[7] Lanham, Micheal. (2021). Generating a New Reality: From Autoencoders and Adversarial Networks to Deepfakes. 10.1007/978-1-4842-7092-9.

[8] Thies, Justus & Zollhofer, Michael & Stamminger, Marc & Theobalt, Christian & NieBner, Matthias. (2016). Face2Face: Real-Time Face Capture and Reenactment of RGB Videos. 2387-2395. 10.1109/CVPR.2016.262.

[9] Graphics, S., 2022. *Benchmark Results - FaceForensics Benchmark*. [online] Kaldir.vc.in.tum.de. Available at: <http://kaldir.vc.in.tum.de/faceforensics_benchmark/> [Accessed 10 April 2022].

[10] Thispersondoesnotexist.com. n.d. *This Person Does Not Exist*. [online] Available at: <https://thispersondoesnotexist.com/> [Accessed 10 April 2022].

[11] Sangyup Lee, Shahroz Tariq, Youjin Shin, Simon S. Woo. (2021). Detecting handcrafted facial image manipulations and GAN-generated facial images using Shallow-FakeFaceNet.