Face Mask Recognition using Deep Learning

A Micro Project Report

Submitted by

E.Manideep Reddy
Reg.no: 99220041175

B.Tech – CSE,AIML

Kalasalingam Academy of Research and Education

(Deemed to be University)

Anand Nagar, Krishnankoil - 626 126

February 2024

KALASALINGAM
ACADEMY OF RESEARCH AND EDUCATION
(DEEMED TO BE UNIVERSITY)
Under sec. 3 of UGC Act 1956. Accredited by NAAC with "A" Grade

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Bonafide record of the work done by E.Manideep Reddy – 99220041175 in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Specialization of the Computer Science and Engineering, during the Academic Year Even Semester (2023-24)

Mr.R.Mari Selvan
Project Guide
Assistant Professor
CSE
Kalasalingam Academy of Research and Education
Krishnan kovil - 626126

Mrs.S.Reshni
Faculty In charge
Assistant Professor
CSE
Kalasalingam Academy of Research and Education
Krishnan kovil - 626126

Mr.B.ShanmugaRaja
Evaluator
Assistant Professor
CSE
Kalasalingam Academy of Research and Education
Krishnan kovil - 626126

Abstract

worldwide epidemic COVID-19There were 19 events that led to a global outbreak of a deadly illness. Put on a face mask to stop the infection from spreading and to shield the wearer from any infectious airborne bacteria. One can keep an eye on whether or not people are wearing masks by using the Face Mask Detection System. The HAAR-CASACADE method is employed in this instance to detect images. When combined with other current methods, this classifier yields a minimal variety of false positive features, an effective feature selection process, and a high identification rate even with changing expressions. With a recognition rate of 85–95%, the HAAR feature-based cascade classifier system uses just 200 features out of 6000 features. We demand mask detection as a distinct and public health service system based on this rationale. during the COVID-19 worldwide pandemic. Both face mask and non-facial mask images are used to train the model.

Keywords: COVID-19 epidemic, HAAR-CASACADE algorithm, mask detection, face mask image, non-face mask image

Contents

List of Figures

Chapter 1

Introduction

Ace mask detection is the process of determining if someone is wearing a mask or not. As a matter of fact, the issue is in reverse engineering face detection, wherein various machine learning techniques are employed to detect faces for the sake of security, authentication, and surveillance. A crucial area of study in computer vision and pattern recognition is face detection. In the past, a sizable corpus of research has advanced algorithms 9999 for facial detection. The first studies on face detection were conducted in 2001, and they trained efficient classifiers for detection and classification by designing handcrafted features and applying conventional machine learning techniques. acknowledgement. The technique presents several challenges, such as low detection accuracy and significant feature design complexity (15 15). Deep convolutional neural networks (CNN)-based face identification techniques have been extensively researched recently to enhance detection performance. According to the literature that is currently available, very little study has been done to detect masks over 111111 faces.

Therefore, the goal of our work is to create a method for properly detecting face masks in public spaces in order to stop the transmission of coronavirus and improve public health. Furthermore, the dataset for identifying masks on human faces is comparatively tiny, making it difficult to train the algorithm to detect faces with or without masks in public. Thus, the learned kernels from networks trained for a comparable face detection task are transferred here using the idea of transfer learning on an

vast dataset. The dataset includes a variety of face images, such as mask-containing and mask-free faces, mask-containing and mask-free faces combined into one image, and mask-free confused images. Using a large dataset of 45,000 photos, our method 9999 obtains an exceptional accuracy of 98.2%. The following summarizes the planned work's main contribution:

1. Using 9999 transfer learning on the back end, create a unique object identification technique that combines one stage and two-stage 111111 detectors to accurately recognize objects in real-time from video streams.

2. Create a brand-new approach to object identification that uses a combination of one- and two-stage 111111 detectors to precisely identify objects in real-time from video streams while using 9999 transfer learning on the back end.

3. Development of an impartial facemask dataset with an imbalance ratio close to unity. 111111

4. Because the suggested model uses less memory, embedded devices used for surveillance can readily implement it. .

1.1 Deep Learning Methodologies for Face Mask Recognition

These methods for deep learning offer a basis for creating accurate and efficient face mask recognition systems capable of detecting masked faces in various real-world scenarios, contributing to public health and safety initiatives. Evaluate the effectiveness of transfer learning compared to training CNNs from scratch, considering factors such as accuracy, training time, and model complexity.

1.2 Dataset Acquisition and Preprocessing

Effective dataset acquisition and preprocessing are crucial steps in developing a robust face mask recognition system. By carefully curating and preparing the dataset, researchers can improve model performance, enhance generalization capabilities, and facilitate The advancement of precise and dependable mask detection models for real-world applications. Address potential class imbalances in the dataset by applying techniques such as oversampling, undersampling, or class-weighted loss functions to ensure equal representation of mask and non-mask instances during model training.

1.2.1 Transfer Learning Techniques in Mask Recognition Models

Transfer learning techniques in mask recognition models involve leveraging pre-trained models for deep learning that were originally trained on large-scale datasets for generic object recognition tasks, and fine-tuning them for the specific task of mask detection. method provides a number of benefits, like as reduced training time, improved performance with limited labeled data, and the ability to capture complex features relevant to mask detection.

1. **Selection of Pre-trained Models**: Evaluate the suitability of different pre-trained models for the mask recognition task based on factors like model complexity, accuracy, and resource requirements.
2. **Fine-tuning Strategies**: Explore methods for setting learning rates, batch sizes, and other hyperparameters during fine-tuning to achieve optimal performance without overfitting.

1.2.2 Data Augmentation and Cleaning Strategies for Improved Model Generalization

Introduce random rotations of images by certain degrees to simulate variations in pose and orientation. Introduce random rotations of images by certain degrees to simulate variations in pose and orientation. Resize images to different scales or crop them to varying sizes to introduce variations in object sizes and positions. Inject Gaussian noise into images to mimic real-world noise and improve model robustness to environmental variations. Change the saturation, brightness, contrast, and color at random. of images to simulate changes in lighting conditions. Apply elastic deformations to images to mimic distortions caused by facial movements or occlusions.

1. **Preprocessing Steps**:
    - Resizing and Normalization: Resize images to a fixed resolution and normalize pixel values to a standard range to ensure consistency across the dataset.
    - Grayscale Conversion: Convert images to grayscale to reduce computational complexity and remove color-based variations that are irrelevant to mask detection.
2. **Class Imbalance Handling**:
    - Oversampling: Duplicate samples from minority classes to balance class distribution and prevent bias in favor of the dominant class.
    - Undersampling: Randomly remove samples from the majority class to achieve a balanced dataset, reducing computational overhead and preventing overfitting.

Chapter 2

Tool and Technology Used

2.1 Deep Learning Frameworks and Libraries

In order to effectively construct, train, and implement neural network models, deep learning frameworks and libraries are vital resources for artificial intelligence practitioners, researchers, and developers. Two notable players in this space are TensorFlow and Keras. Google's TensorFlow provides a rich ecosystem and strong assistance for creating and refining deep neural networks. Because of its adaptability, it may be easily implemented on a variety of platforms, such as distributed systems and mobile devices, which makes it a popular option for both research and production settings. In contrast, Keras offers an intuitive user interface that is based on TensorFlow.

2.1.1 TensorFlow and Keras:

- **TensorFlow**: TensorFlow is an open-source deep learning framework developed by Google that provides comprehensive tools and libraries for building and training neural networks. It offers flexibility and scalability, allowing developers to deploy models in various environments, including mobile devices and cloud servers.
- **Keras:** Keras is a high-level neural networks API written in Python that serves as an interface for TensorFlow, allowing for fast experimentation and deep learning model prototype. It offers an interface that is easy to use. for building and training neural networks with minimal code, making it ideal for beginners and researchers.

2.1.2 Python :

Python is a high-level, interpreted, object-oriented language with dynamic semantics. Its dynamic typing and dynamic binding, along with its high-level built-in data structures, make it an appealing language for Rapid Application Development and for usage as a scripting or glue language to join existing components.

2.2.1 Matplotlib

A complete Python visualization toolkit for static, animated, and interactive graphics is called Matplotlib. Matplotlib enables both difficult and easy tasks. Make plots fit for a publishing. Create dynamic figures with zoom, pan, and update capabilities. Personalize the layout and visual design. Export in a variety of file formats.Integrate graphical user interfaces and Jupyter Lab. Utilize a wide range of third-party packages constructed using Matplotlib.

2.2.2 Numpy and Scipy

Numpy : A library for the Python programming language, NumPy (pronounced /ˈnómpaɪ/ (NUM-py) or occasionally /ˈnŌmpi/ (NUM-pee)) adds support for big, multi-dimensional arrays and matrices. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Num array into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. NumPy is a NumFOCUS fiscally sponsored project.

Scipy : Algorithms for eigenvalue problems, algebraic equations, differential equations, statistics, optimization, integration, interpolation, and many other problem types are available in Scipy. Because of its high level syntax, SciPy is useful and approachable to programmers with varying backgrounds and levels of experience. Modules for signal and image processing, linear algebra, integration, interpolation, special functions, FFT, optimization, ODE solvers, and other common science and engineering tasks are included in SciPy.

2.3 Development Environment and Collaboration Tools

Development environments and collaboration tools play a crucial role in facilitating efficient software development workflows, particularly in the realm of deep learning and AI. PyCharm is a standout development environment renowned for its robust features tailored specifically for Python programming. Its integrated interface encompasses a rich set of tools such as code editor, debugger, and version control integration, streamlining the development process and enhancing productivity. With PyCharm, developers can write, debug, and manage Python projects seamlessly, ensuring code quality and consistency.

2.3.1 PyCharm

PyCharm Dedicated to Python development, PyCharm is an Integrated Development Environment (IDE) that offers a comprehensive set of tools that are tightly integrated to create a pleasant environment for efficient Python, web, and data science development.productive environment for writing, debugging, and managing Python projects, suitable for individual developers and teams.

2.3.2 Version Control and Collaboration Tools

1. **Git:** A distributed version control system called git it is employed for tracking changes in code and coordinating work among multiple developers. It allows for efficient collaboration, code review, and project management, enabling teams to maintain a history of changes and easily revert to previous versions if needed.

2. **GitHub:** GitHub is an online hosting platform for version control that uses Git, providing features for code hosting, collaboration, and project management. It offers tools for code review, issue tracking, and continuous integration, facilitating collaborative software development workflows.

Chapter 3

System Development Overview

3.1 Design

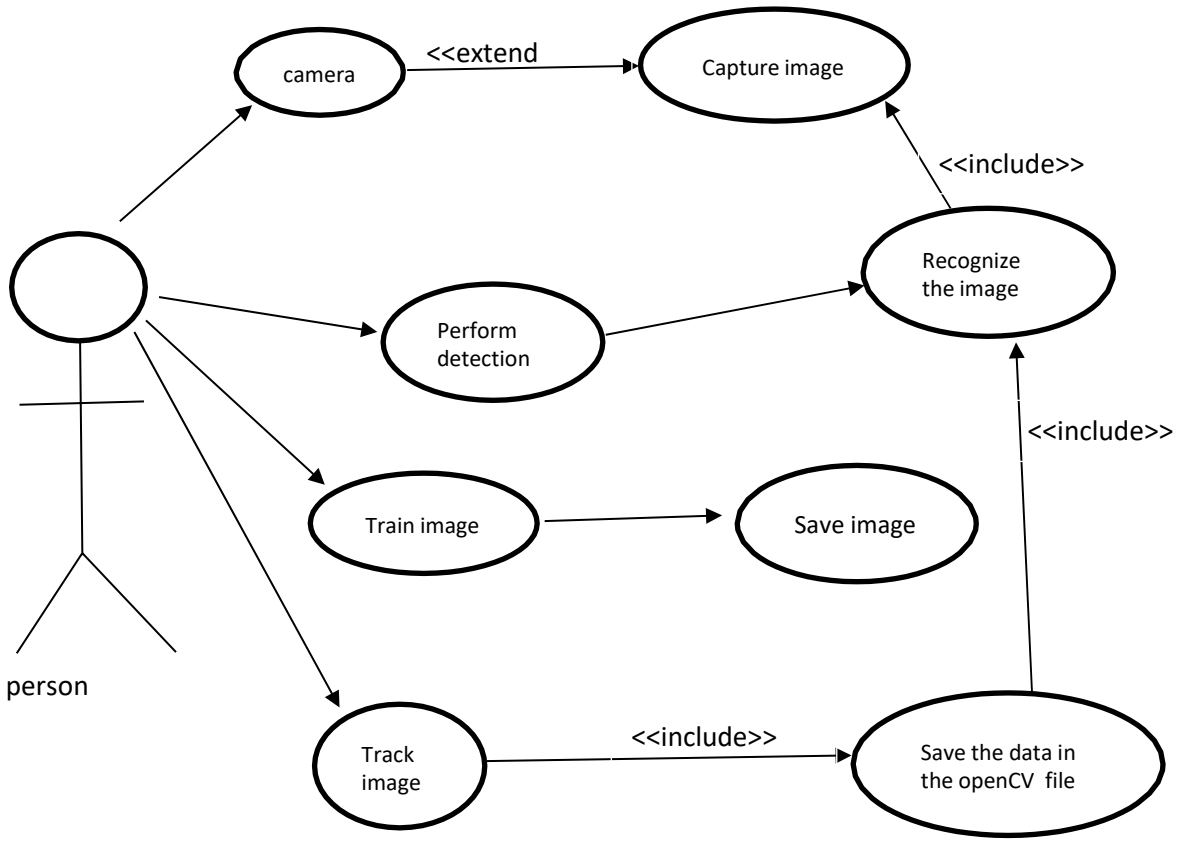
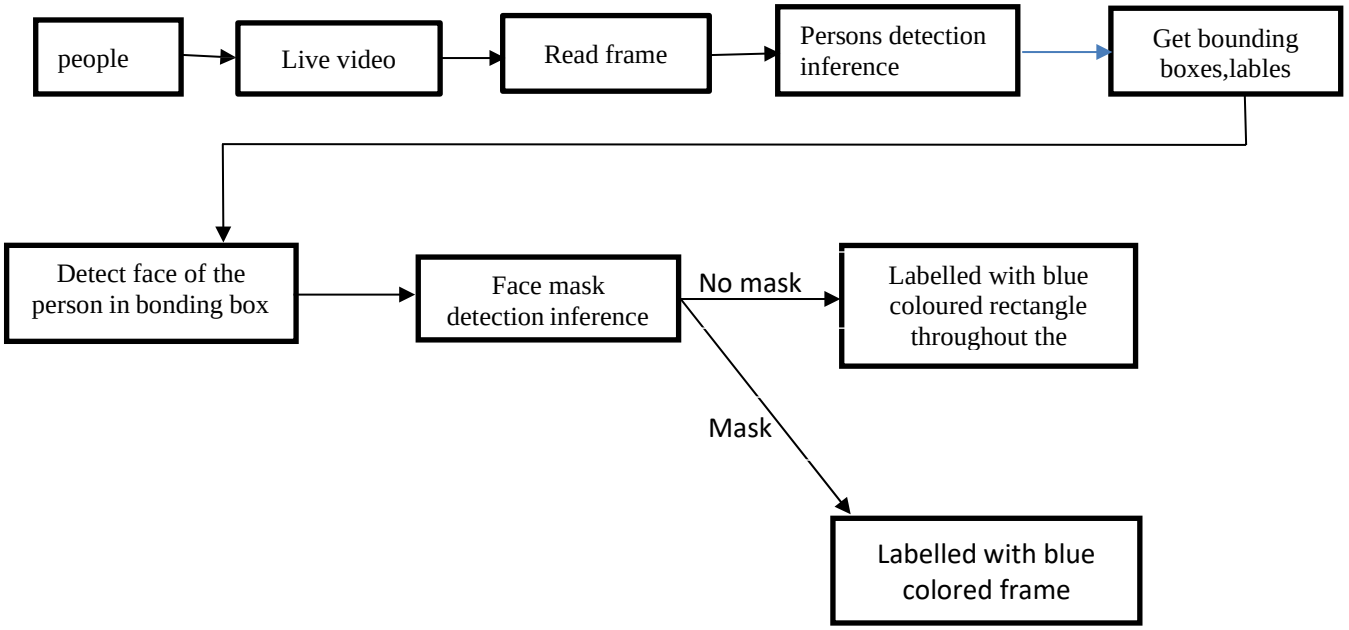


Figure 1.1 use case diagram

3.1.1 Block diagram



Face Mask Detection in webcam stream:

The process of determining whether the individual in the camera is wearing a face mask. This is

a two-step process.

- To recognize the faces captured on camera.
- Utilize the mask to categorize the faces.

Recognize the Face on the Webcam:

An OpenCV framework-provided pre-trained model was utilized to recognize the faces. Web photos were used to train the model. Two models are available for this face detector in OpenCV.

code

```
# USAGE
# python detect_mask_video.py

# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils
import time
```

```
import cv2
import os

def detect_and_predict_mask(frame, faceNet, maskNet):
	# grab the dimensions of the frame and then construct a blob
	# from it
	(h, w) = frame.shape[:2]
	blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
		(104.0, 177.0, 123.0))

	# pass the blob through the network and obtain the face detections
	faceNet.setInput(blob)
	detections = faceNet.forward()

	# initialize our list of faces, their corresponding locations,
	# and the list of predictions from our face mask network
	faces = []
	locs = []
	preds = []

	# loop over the detections
	for i in range(0, detections.shape[2]):
		# extract the confidence (i.e., probability) associated with
		# the detection
		confidence = detections[0, 0, i, 2]

		# filter out weak detections by ensuring the confidence is
		# greater than the minimum confidence
		if confidence > args["confidence"]:
			# compute the (x, y)-coordinates of the bounding box for
			# the object
			box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
			(startX, startY, endX, endY) = box.astype("int")

			# ensure the bounding boxes fall within the dimensions of
			# the frame
			(startX, startY) = (max(0, startX), max(0, startY))
			(endX, endY) = (min(w - 1, endX), min(h - 1, endY))

			# extract the face ROI, convert it from BGR to RGB channel
			# ordering, resize it to 224x224, and preprocess it
			face = frame[startY:endY, startX:endX]
			if face.any():
				face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
				face = cv2.resize(face, (224, 224))
				face = img_to_array(face)
				face = preprocess_input(face)

				# add the face and bounding boxes to their respective
				# lists
				faces.append(face)
				locs.append((startX, startY, endX, endY))
```

```
		# only make a predictions if at least one face was detected
		if len(faces) > 0:
			# for faster inference we'll make batch predictions on *all*
			# faces at the same time rather than one-by-one predictions
			# in the above `for` loop
			faces = np.array(faces, dtype="float32")
			preds = maskNet.predict(faces, batch_size=32)

		# return a 2-tuple of the face locations and their corresponding
		# locations
		return (locs, preds)

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
		default="face_detector",
		help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
		default="mask_detector.model",
		help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
		help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
		"res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
maskNet = load_model(args["model"])

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)

# loop over the frames from the video stream
while True:
		# grab the frame from the threaded video stream and resize it
		# to have a maximum width of 400 pixels
		frame = vs.read()
		frame = imutils.resize(frame, width=400)

		# detect faces in the frame and determine if they are wearing a
		# face mask or not
		(locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

		# loop over the detected face locations and their corresponding
		# locations
```

```
	for (box, pred) in zip(locs, preds):
		# unpack the bounding box and predictions
		(startX, startY, endX, endY) = box
		(mask, withoutMask) = pred

		# determine the class label and color we'll use to draw
		# the bounding box and text
		label = "Mask" if mask > withoutMask else "No Mask"
		color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

		# include the probability in the label
		label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

		# display the label and bounding box rectangle on the output
		# frame
		cv2.putText(frame, label, (startX, startY - 10),
			cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
		cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

	# show the output frame
	cv2.imshow("Frame", frame)
	key = cv2.waitKey(1) & 0xFF

	# if the `q` key was pressed, break from the loop
	if key == ord("q"):
		break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

3.1.2 Functional requirements & Non-Functional requirements

1. **FUNCTIONAL REQUIREMENTS**

Delivering processing results to users is the main goal of computer results. They are also used to keep an archive of the outcomes for later reference. Broadly speaking, the following are various kinds of outcomes:

- Outcomes exported outside of the organization are referred to as external results.
- Internal results, which are displayed primarily on computers and by users and have a place in the company.
- Using operating results exclusively for the department of computers.
- User-interface outcomes that let users speak with the system directly.
- Gaining insight into the user's inclinations, technological proficiency, and commercial requirements via an amicable questionnaire.

2. **NON-FUNCTIONAL REQUIREMENTS**

SYSTEM CONFIGURATION

Commodity hardware can be used to run this project. Our AMD Ryzen5 CPU, which has six cores operating at 1.7 GHz and 2.1 GHz, 16 GB of RAM, and a 4 GB Nvidia graphics processor, powered the entire project. The training phase takes ten to fifteen minutes, and the testing phase takes only a few seconds to complete the accuracy calculations and make predictions.

HARDWARE REQUIREMENTS

• Memory: 4 GB

• 500 GB of storage

• Processor: 2 GHz or more

• Bit or 64-bit architecture

SOFTWARE NEEDS

• Python 3.5 is utilized in PyCharm for pre-processing data, training models, and making predictions.

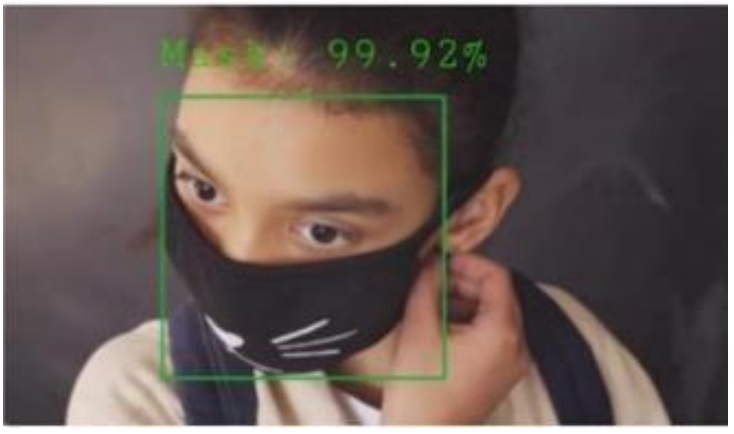
• Operating System: Windows 7 and higher, Linux-based operating systems, or Mac OS

• Python is the coding language.

• Graphic Processor Nvidia

Chapter 4

Result



4.1 Future enhancements

Jingdong has a greater than 99 percent recognition accuracy. We produced the RMFRD, SMFRD, and MFDD datasets along with an innovative algorithm built upon them. The algorithm will enable face authentication without physical touch in environments including university governance, community access, and enterprise resumption. The globe now has greater scientific and technological might thanks to our research.

4.1 Conclusion and Future Work

Emerging advances in technology have made it possible for us to have a revolutionary face 333 mask detector, which may have applications in public health. With Mobile Net serving as its foundation, the design is applicable to both high and low computation scenarios. We use transfer learning to adopt weights from a related task, face detection, which is trained on a massive dataset, in order to extract more robust features. TensorFlow, NN, and OpenCV were utilized to determine whether or not persons were wearing face masks or not. Both still photos and live video streams were used to test the models. By adjusting the hyperparameters, we are able to attain the model's correctness and continue to optimize it in order to create an extremely accurate solution. There may be a use case for edge analytics using this particular paradigm. Also, using a dataset of public face masks, the suggested approach produces state-of-the-art outcomes. As face mask recognition technology advances, we will be able to identify face mask wearers and grant them admission, which will benefit society greatly.

References

1. S. V. Militante and N. V. Alarm Dionisio (2020). Deep Learning-Based Real-Time System. Research Colloquium 11th (ICSGRC) 2020, Facial Mask Recognition of the Shah with IEEE Control and System Alam, Graduate Malaysia.. https://doi.org/10.1109/ ICSGRC49013.2020.9232610 Integer nec odio. Praesent libero.Sed cursus ante dapibus diam. Sed nisi.

2. In 2020, Guillermo, M., Pascua, Dadios, E. A. R. A., and Billones, R. K. The COVID-19 Risk Sybingco, E., Fillone, Assessment via A., and Multiple Face Mask Detection with MobileNetV2 Computational DNN are reported.https:// isciia2020.bit.edu.cn/docs/20201114082420135149. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi.

3. In 2018, Boyko, N., Basystiuk, O., & of Software, IEEE Second (DSMP), International Lviv, Ukrain e. https://doi.org/10.1109/DSMP.2018.8478556

4. Cousera

Chapter 6

Certification

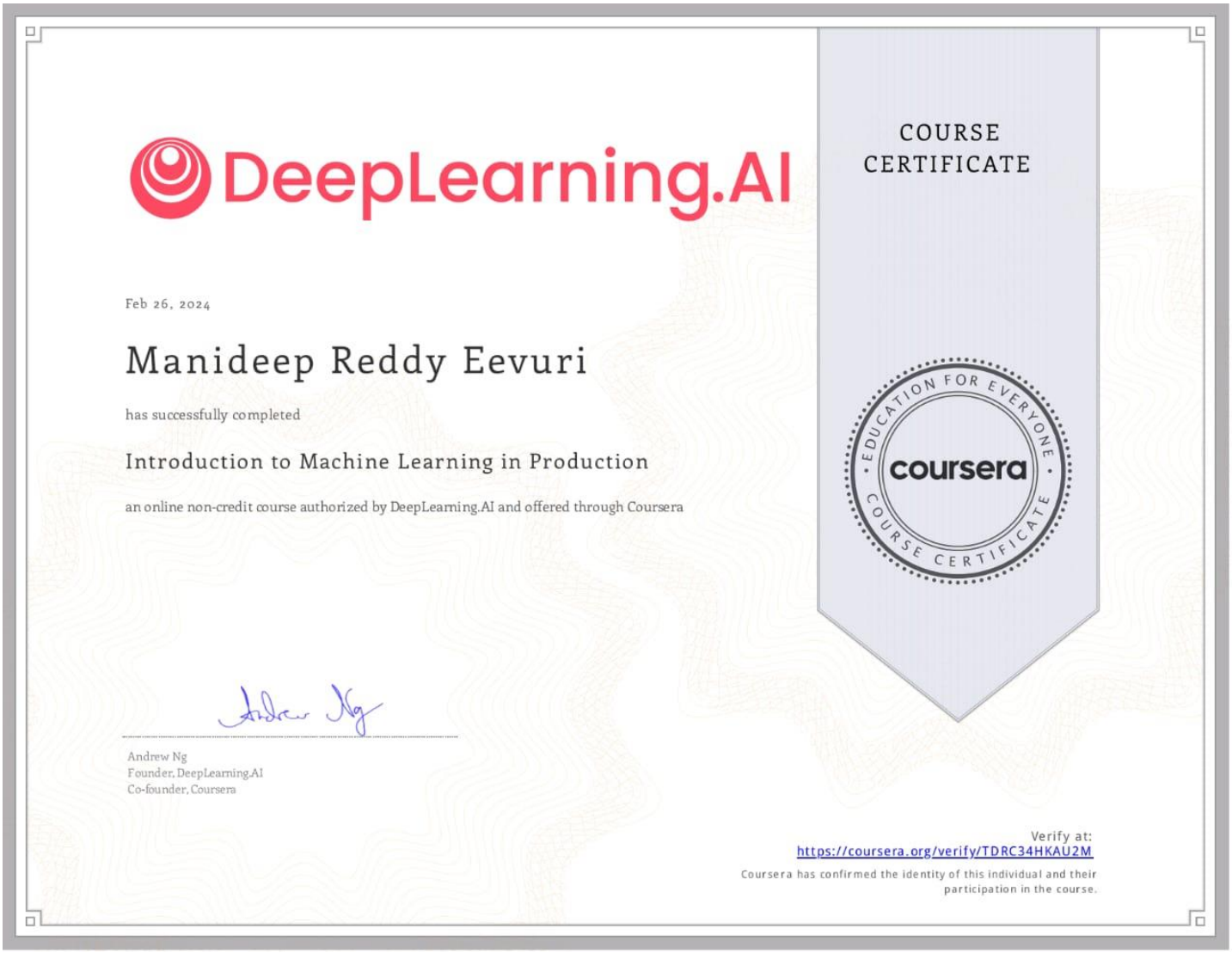


Figure 6.1: Certification details