Development of hand gestures in poor light and low resolution.

Author: Brahath Shet

Supervisor: Dr Georgios Mastorakis

Table of Contents

Chapter 5 Implementation

5.1 Final implementation

Popular uses for computer vision and machine learning include the recognition that can capture and decode image of hand gestures. The project entails live capture of a person's hand motions using a camera, GPU acceleration, and computer vision libraries to analyse the image. Such that it can identify and track the hand's landmarks, using pre-built functions and algorithms in these libraries gives computers the recognising the gestures using machine learning techniques. A system that can distinguish a wide range of task, including gesture recognition, feature representation, and segmentation techniques of hand gestures, including those used in sign languages is to aim in filtering deprived illumination. In this project, I will confer the final implementation of a hand gesture recognition system using Python free and open-source libraries and repos [2][3][4].

During the trial the hand gesture recognition algorithm, a specific number of stages must be achieved. I have produced a datasets of hand movements for the machine learning model to be trained on throughout the data gathering stage. I would extract and store the pixel values from images performing hand gestures using Python tools like OpenCV and Mediapipe which allows us to obtain the x and y pixel positions of the parameter when capturing the data [2].

The final implementation of the project involves the following steps:

Data Collection: The first step in building any machine learning project is to collect the data required for training the model. In this project, I have created new data of different hand gestures using a webcam. There are collective data of my hand with different positionings to make the model more robust. Likewise, there are different hand labeled data stored in a format that can be used for training. In terms of inputting the data Mediapipe' s hand tracking solution does not explicitly use grayscale images as input instead it requires a single RGB camera image that contain 3 colour channel for its hand tracking solution to predict the hand skeleton [2].

Data preprocessing: I used a variety of approaches to improve the data's accuracy at this stage, in addition to noise reduction the data is standardized and made things simpler to examine. Scaling, inversion, and the application of gamma correction are some of the featured techniques utilised to enhance the illumination of the images.

After pre-processing the data, I can move on to the next stage, which is hand detection. This involves using computer vision techniques to identify and track the landmarks of the hand, such as the fingertips, wrist, and palm. There are several computer vision libraries available, such as Mediapipe, which provide pre-trained models for hand landmark detection. I have used these models to extract the hand landmarks from each frame of the video.

The next stage is featuring extraction, where I extract relevant features from the hand landmarks to recognize hand gestures in real-time, the system needs to be able to detect pattern of the hand. Feature extraction is an important step in machine learning, as it helps to reduce the dimensionality of the data and extract meaningful patterns. There are several convolutional layers that are used to extract features from an input image or video frame. The hand detection model uses a feature extractor based on MobileNetV2, which is a lightweight convolutional neural network architecture designed for mobile and embedded devices [2].

The Mediapipe model uses supervised learning projected ground-truth 3D joints for each hand pose. These 3D joints are projected onto a 2D plane to create synthetic training data. The data combined with personally created datasets, which are generated by logging hand key points and during

training the datasets I have dividing the data into 70% training and 30% testing. Therefore, enhancing the model's performance hyperparameters such as batch size, early stopping, activation function. Upon identifying the ideal hyperparameters, the model is trained utilizing an algorithm, for instance, stochastic gradient descent, which aims to reduce the loss function and boost the precision of hand gesture identification.

Evaluating the model: The model must be examined once it has been trained to gauge its effectiveness. But I have given some testing with different noises added to accomplished by putting the model to the experiment on a different visual to classify the break point for which there is no training data. The model's performance can be evaluated using several criteria, including loss and accuracy.

Final implementation: Finally, the hand gesture recognition system is implemented using the trained machine learning model and the hand landmark detection techniques. The system takes input from a webcam, detects the hand landmarks, and classifies the gesture using the trained model. The recognized gesture is then displayed on the screen, allowing the user to interact with the system using hand gestures.

5.2 Detailed technical description.

The Hand Gesture Recognition code is an implementation of hand gesture recognition using a machine learning model. The code is implemented in Python and is designed to run on a machine with a webcam. It uses a pre-trained machine learning model to classify the hand gestures. The code saves the training data in a CSV file that can be used to train a new model.

Technical Description:

The best ways to run these files to use PyCharm IDE with python 3.9 & above and installing libraries.

Libraries: The code uses several libraries to perform its functions, including:

argparse: for parsing command-line arguments.

csv: for reading and writing CSV files that have been used for collecting for hand gesture.

copy: for copying objects.

itertools: This Python module offers a selection of tools for interacting with iterators and iterable objects. In addition to functions for working with infinite iterables, it also has functions for filtering and grouping data as well as functions for creating combinations, permutations, and other kinds of iterables [5].

collections: for working with collections of data.

NumPy: Large, multi-dimensional arrays and matrices are supported by NumPy, along with a vast variety of sophisticated mathematical operations that may be performed on these arrays and matrices.

cv2: for processing images and video streams.

Video Capture: The code captures the video feed from the webcam attached to the machine. It's efficient performance allowing the user to set the width and height of the video stream. The video stream is then processed to detect the hand landmarks. In this part OpenCV is a class that provides a convenient way to capture manipulate image and video data efficiently and process video frames from cameras, video files, or image sequences [3].

TensorFlow Lite XNNPACK delegate is an optimization feature for TensorFlow Lite, a machine learning framework for deploying lightweight models on mobile and embedded devices. This delegate leverages XNNPACK, a highly optimized library for neural network inference, to improve CPU performance. By using the XNNPACK delegate, TensorFlow Lite can accelerate model execution on devices with limited resources, providing faster and more efficient inferencing [2]. During the process of training and testing the custom datasets converting to TensorFlow Lite will provide a compact model performance with efficient use of resources rather than using TensorFlow GPU power that will performance hungry effective on runtime process.

Mediapipe: for hand landmark detection and tracking.

According to the Mediapipe, it uses a combination of two models to palm detects and tracks the hand and finger landmarks in an image or video which applies pre-processing steps to prepare input. It can detect 21 3D landmarks on a hand from just a single frame. The pipeline consisting of multiple models working together: A palm detection model that operates on the full image and returns an oriented hand bounding box and a hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand key points [2].

Landmark Classification: The KeyPointClassifier models used to classify the landmarks and hand gestures. The class (hand gesture) of the input feature vector is predicted using the classify () method of the KeyPointClassifier. The process accepts a feature vector and outputs the anticipated class label. For example, stop, fist, peace, thumbs up, thumbs down. Based on the identified gestures, the data can then be used to execute specified actions or initiate events in your application while the models are trained using collected dataset, and the code reads the labels for the models from a CSV file.

Data Logging: The code logs the hand and finger landmark data to a CSV file for training the models. The logging can be enabled or disabled based on the mode set by the user. The user can also set the number associated with the gesture for logging. The keypoint.csv has all the data starting from 0-7 following the label's structure, for instance if STOP sign would be logged "0" is associated which marks the coordinates of x and y position values when registering the key point of hand marker. Below is the list of data that are logged into the CSV file.

Fig5.0 final list of key points

Accuracy Measurement: The code uses the Counter function from the collections library to measure the precision and stability of detection. it is keeping track of the most recent hand gesture predictions in the past and choosing the most prevalent gesture from that history to serve as the current classification of hand gestures which can be less vulnerable to temporary errors or fluctuations in the input data.

The probability that a hand is present and reasonably aligned in the input image, as indicated by the confidence score. This score is determined by the hand tracker model and depends on several parameters, including the source image's quality and thus the precision of the hand landmark predictions [2].

User Interface: The code has a user interface that displays the video stream with the bounding box and landmarks for the detected hand which is drawn using OpenCV library to create a landmark of the hand when video feed-forwarded to processing it also displays the hand and finger gesture IDs getting the accurate detection.

5.3 Evaluation & Testing

Hand tracking solution is capable of working with both video streams and still images, but the pipeline applied to these different input sources has some differences. The pipeline applies GPU a series of optimizations and heuristics to improve the performance and accuracy of hand detection when processing a video stream. These optimizations include using temporal information from previous frames to improve hand detection in the current frame.

However, these optimizations may not be as effective when processing a single still image, making it more challenging to detect hands accurately. In the folder “still image test”, I have 8 different images and test results in “Jupyter notebook” format which show the model may identify the hand. For example, infrared image of hands in file “0_01” Mediapipe does not recognize the hand as the finger are too close and of hand gets bright in the middle so the algorithm may detect as a white circle. On the other part, file “02_02” gets detected when pointed at the camera as it runs several layers to classify the hand as right with score of 0.80 and identify the finger. There are many more images that I have provided the results shown in the folder.

I had taken images of my hand files “my_01” and “my_03” to check if the Mediapipe discovers the hand in the normal lighting condition with the hand angled. In which “my_01” was recognized conversely, “my_03” was not able to detect because with still image angled into a symmetry position and not considered the hand taken as an object.

Therefore, to improve the accuracy of hand detection in still images, adjustments to some of the pipeline's parameters or preprocessing steps may be necessary. For instance, one can adjust the threshold for hand detection or apply additional image preprocessing steps, such as contrast enhancement or noise reduction.

Furthermore, performing in real-time I discovered some results that were tested to understand the datasets that were trained in extreme lighting condition showing as proof of how Mediapipe tries to recognize in poor lighting settings of gamma 4.0 and above as it would need more data that can be trained using different visual conditions to make it robust.

In fig5.1 and 5.2 image processing of low light makes it difficult to detect the hand for this model. Given that there are downfalls in succeeding it would need to added max pooling and retrain on hidden layers for the hand to detected. Also, finding out the Mediapipe uses RGB camera as input and dark skin pigment the machine take some time and would need more computing power to recognize the hand.

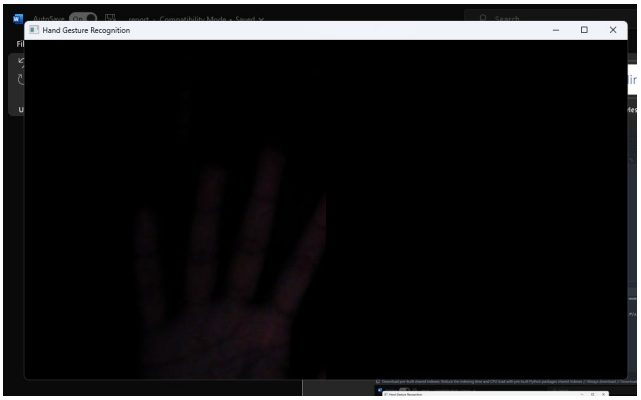

Fig 5.1 in a dark part the hand is not detected.

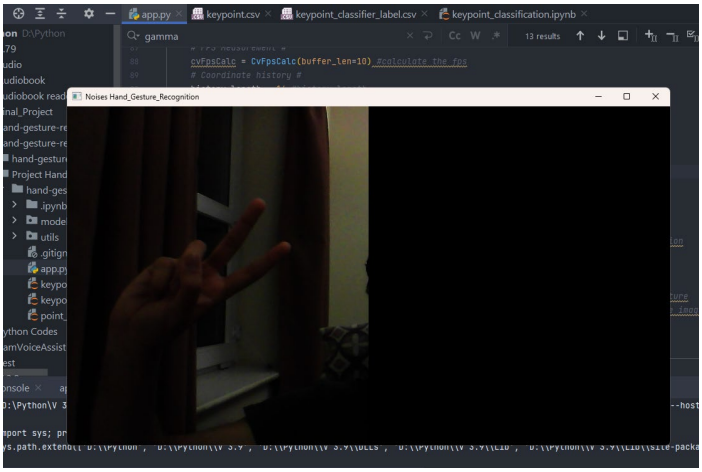

Fig5.2 in the 4.0 gamma correction setting the hand is not detected.

During the process of certain datasets giving false information seen in fig 5.3 the data collected as it is not perfect data and required data augmentation to regularize the hand in certain position. Given the lighting condition was natural. After data augmentation in fig5.4 the data was calibrated to identify the sign.

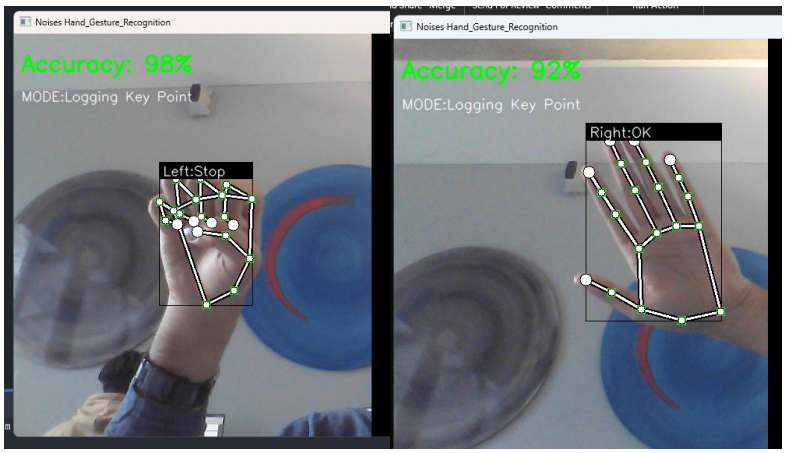

Fig5.3 these are false recognition.

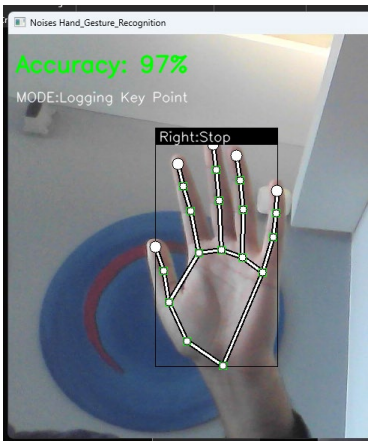

Fig 5.4 stops sign.

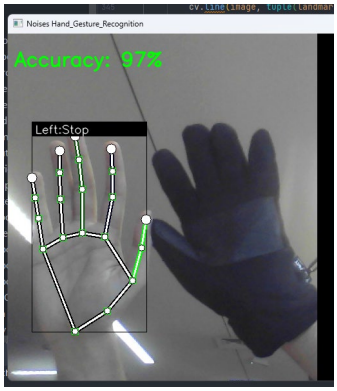

Fig 5.6 object obstruction for hand

in fig5.6 the hand detection does not identifies the hand wearing a glove when it has interfered to the model. This would show there is an element of colour that differentiates the hand with the Mediapipe model. The model needs to be trained in hand detecting in both conditions to notice the hands.

While working on the processing and training the data, I ran a few tests to check the accuracy of hand detection running the datasets to check how well it will perform. In fig 5.8 during processing new data there was a `loss: 0.4451 - accuracy: 0.7753` given that data is underfitting after 23 epochs, an early stopping added to check these results. The reason to do this test was to get the worst case with 5 gestures stored into the datasets of fig5.7. This means that the model lacks data, poor data quality, nonrepresentative data, uninformative features, excessively simple models. Hence, more training data or performing data augmentation to increase the size and variety of the dataset can show some sign of improvement.

Fig5.7 tested on list of labels.

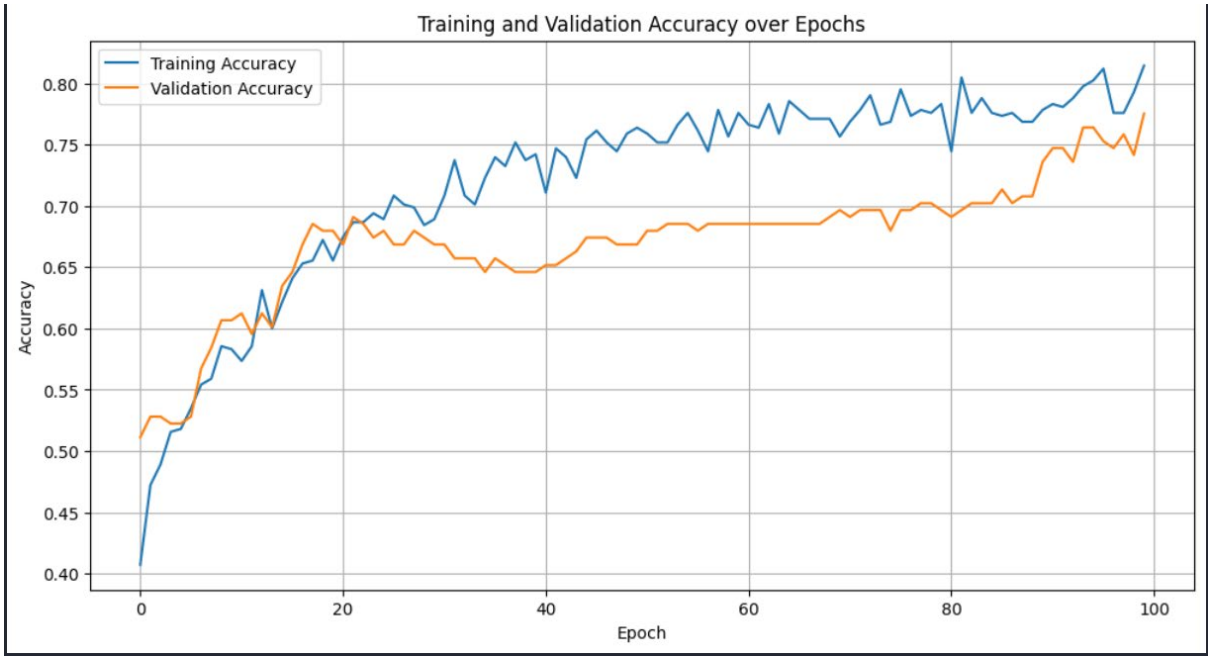

Fig 5.8 data of the false recognition.

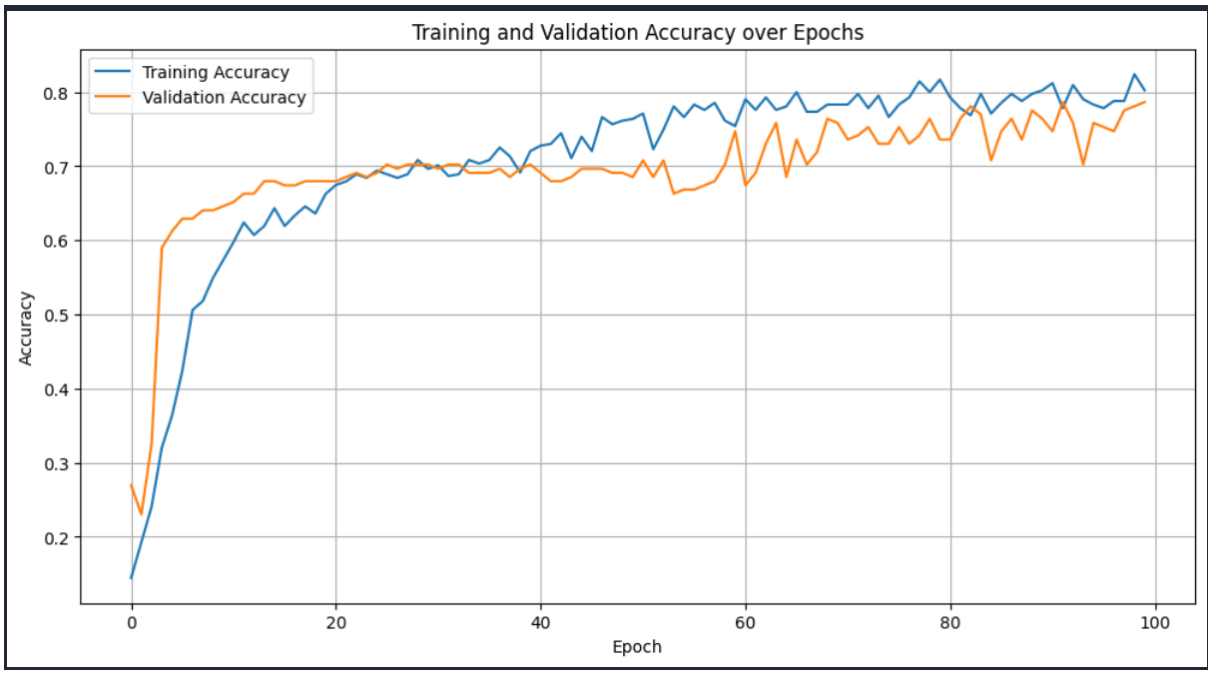

Fig5.9 adding more data and use of early stopping on false recognition.

In fig 5.9, it was retraining on the data augmentation by configuring the stop and ok sign given the `loss: 0.4070 - accuracy: 0.7865` and the data shows it is overfitting at the beginning because the noise and random fluctuations present in the data, at epochs 0 to 20. However, at epoch 20-30 it gives the good fit showing it perform better than previous fig 5.8. While this data was split into 70% trained and 30% tested. Therefore, I would focus on tuning the model for further improvement to get better results in the future [6].

In fig 5.10 this is the final data distributed to all classes presented for analyzing the distribution of classes can help to identify the areas that have imbalance data which classes may require more data.

For some classes such as class 5 & 6 it has fewer dataset, it might consider collecting more data for those classes to balance the dataset and improve model performance.

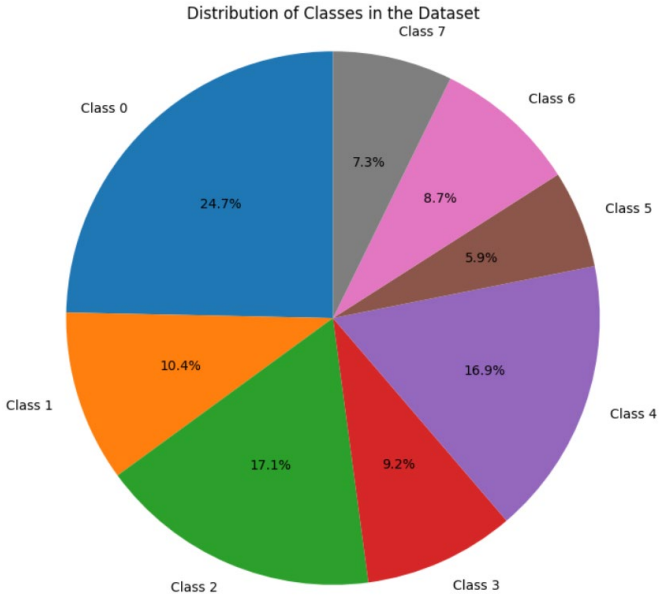

Fig 5.10 of distribution of dataset that are trained.

Overall to conclude with all the points, there are improvement needed to make it robust and precision in all types of visual condition would be getting to think over the parts of baseline. Hand Gesture Recognition code is an implementation using a machine learning model. The code is designed to run on a machine with a webcam and uses the Mediapipe and OpenCV libraries to capture and process the video stream. The code uses pre-trained models to classify the hand and finger gestures and logs the data to a CSV file for training new models. The user interface displays the video stream and the hand and finger gesture landmark.

Due to the poor image quality and loss of crucial visual information, hand gesture detection in low-resolution situations is a difficult undertaking. Nonetheless, there are several methods and algorithms that may be applied to raise hand gesture recognition's precision in these circumstances.

To increase the precision of gesture recognition, one method is to apply deep learning techniques such as convolutional neural networks (CNNs). Datasets of hand gesture photographs are used to train, which can recognise patterns and features in the data that are crucial for precise categorization. As a result, the network performs better on live feed with low quality and resolution since it can learn to identify patterns even when some information is absent. However, in the still images from Kaggle dataset and my own datasets the Mediapipe library does not identify certain images. It would be less likely to detect the hand in certain angle and light settings when the source of input was changed.

Before submitting the input photos to the recognition system, it is crucial to preprocess them in addition to these methods. The quality of the images can be improved and made more suited for precise recognition by using preprocessing techniques including contrast enhancement, noise reduction, and image standardization.

Generally, a mix of techniques and algorithms is needed to obtain high accuracy in hand gesture detection in low and poor resolution. Even in difficult situations, reliable recognition can be accomplished by utilising deep learning techniques, feature extraction, and the right preprocessing.

Reference

1. Brahath, S(2023). Final-project [Computer vision]. https://github.com/BrahathS/finalProject.git
2. Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C.L., Yong, M.G., Lee, J. and Chang, W.T., 2019. Mediapipe: A framework for building perception pipelines. arXiv preprint arXiv:1906.08172. GitHub , https://google.github.io/mediapipe/
3. Bradski, G. and Kaehler, A., 2008. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.".
4. Takahashi, S. (2020). hand-gesture-recognition-using-mediapipe [Computer software]. https://github.com/Kazuhito00/hand-gesture-recognition-using-mediapipe
5. Van Rossum, G. & Drake, F.L., 2009. *Python 3 Reference Manual*, Scotts Valley, CA: CreateSpace.
6. Géron, A. (2023) *Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools, and techniques to build Intelligent Systems*. Sebastopol (CA): O'Reilly Media.