



BEACONHOUSE NATIONAL UNIVERSITY

ShotSense

PRJ-F23/337

Implementation Documentation

EXTERNAL SUPERVISOR

Mirza Danial Masood

INTERNAL SUPERVISORS

Huda Sarfraz

GROUP MEMBERS

Umair Ali Khan

F2020-532

Muhammad Fahad

F2020-157

SCHOOL OF COMPUTER & IT

January 2024

Table of Contents

Table of Contents.....	2
Introduction.....	3
ML Development.....	3
Data Gathering.....	3
Training Data.....	3
Testing Data.....	3
Data Preprocessing.....	4
Video Frame Generator.....	6
Feature Engineering with Pretrained Models.....	7
Google Cloud Datasets.....	8
Experimentation.....	9
Custom Models.....	9
EfficientNet.....	9
VGG16.....	9
TensorFlow Lite.....	10
Google's AutoML.....	11
Mobile Application Development.....	12
Development Platform and Tools.....	12
Flutter.....	12
Firebase.....	12
Cloud Services.....	12
ML Model Selection.....	12
Mobile Application Flow.....	13
Functionality.....	13
Data Management.....	14
Test Cases.....	15
View a session from history.....	15
Add new balls to session from gallery.....	16
Record balls in session from camera.....	17
Display shot history by shot type.....	18
View shot analysis per ball.....	19
Conclusion.....	20
References.....	21

Introduction

The Implementation Document provides a practical roadmap for turning our design concepts into a fully functioning application. This document offers a detailed exploration of the ML development process, data procurement, and preprocessing steps. We will walk through the specifics of feature engineering using pretrained models, the utilization of Google Cloud Datasets, and our experimentation phase for model selection. Moving forward, the document will cover the development of the mobile application, discussing the platforms and tools, including Flutter, Firebase, and cloud services. We will present the mobile application's flow, functionality, and data management, supported by concrete test cases. This straightforward guide aims to document our implementation journey, making it accessible and understandable for all stakeholders involved.

ML Development

Data Gathering

The initial phase of ML development begins with data gathering, a pivotal step that lays the foundation for robust model training. In this stage, we obtained a high quality dataset with over 1800+ videos of 10 different shot types from a research paper by Anik Sen[1]. These videos capture shots performed by professional cricketers in international broadcasts, establishing a benchmark for the expected shot variations. The emphasis on both quality and diversity within our dataset is paramount, as it significantly contributes to refining the model's capacity to generalize and adeptly classify cricket shots within real-world scenarios.

Training Data

Instances: 1800+ videos

Format: .AVI

Description: This dataset encompasses a diverse range of cricket shots, comprising 10 distinct classes and featuring over 1800 videos. Notably, these videos were meticulously curated from international broadcasts, offering a rich and varied collection of shots played by professional cricketers.

Testing Data

Instances: 50+ videos

Format: .AVI, .mp4

Description: Our unseen testing data consisted of 5 videos for each of the 10 classes. A deliberate decision was made to maintain the testing data in its raw form, extracted directly from YouTube Shorts. This strategic choice allowed us to replicate authentic instances of application use, considering the multitude of angles and qualities encountered in real-world scenarios. Our testing dataset features a diverse array of cricket shots, mirroring the challenges and variations users may encounter during live gameplay.

Data Preprocessing

In the preprocessing phase of training videos, we employed a two-step approach to refine the dataset for optimal training of our shot classification model. Initially, we utilized ffmpeg to crop the videos, removing excess background and adjusting them to the aspect ratios commonly found in mobile-recorded videos. The provided script showcases the use of ffmpeg[2] to achieve this, creating an output directory and processing each '.avi' file by cropping and scaling it appropriately.

```
mkdir output
for file in ./*.avi; do
    ffmpeg -i "$file" -vf "crop=in_w*0.5625:in_h,scale=1080:1920,setsar=1:1" -c:a copy -pix_fmt yuv420p
    ./output/${basename "$file"}_vertical.avi"
```

Figure 1 ffmpeg script to crop the video in 9:16 format

Subsequently, we leveraged OpenCV[3] and moviepy to further enhance the dataset. The Python script provided demonstrates how we utilized moviepy to load each video, calculate its duration, and intelligently trim unnecessary segments from the beginning and end, focusing solely on the cricket shots. This process helps eliminate additional information and ensures that our model is trained on relevant shot sequences.

```
from moviepy.video.io.VideoFileClip import VideoFileClip
import os

def crop_video(input_path, output_path):
    # Load the video clip
    video_clip = VideoFileClip(input_path)

    # Get the duration of the video
    duration = video_clip.duration


    # Calculate the time to trim from the beginning and end based on video duration
    if duration < 2:
        start_time = 0.25
        end_time = 0.25
    elif duration < 3:
        start_time = 0.25
        end_time = 0.5
    else:
        start_time = 0.25
        end_time = 1.0

    # Remove specified duration from the beginning and end
    cropped_clip = video_clip.subclip(start_time, duration - end_time)

    # Write the cropped clip to the output folder without audio
    cropped_clip.write_videofile(output_path, codec="libx264", audio=False)

    # Close the video clip to free up resources
    video_clip.close()
```

Figure 2 Crop Video Function



```
def bulk_crop_videos(input_folder, output_folder):
    for split in ['train', 'test', 'val']:
        split_input_folder = os.path.join(input_folder, split)
        split_output_folder = os.path.join(output_folder, split)

        # Create the output folder for the split if it doesn't exist
        os.makedirs(split_output_folder, exist_ok=True)


        # Iterate through class folders
        for class_folder in os.listdir(split_input_folder):
            class_input_folder = os.path.join(split_input_folder, class_folder)
            class_output_folder = os.path.join(split_output_folder, class_folder)

            # Create the output folder for the class if it doesn't exist
            os.makedirs(class_output_folder, exist_ok=True)

            # Iterate through video files
            for file_name in os.listdir(class_input_folder):
                if file_name.endswith('.avi'):
                    input_path = os.path.join(class_input_folder, file_name)
                    output_path = os.path.join(class_output_folder, file_name)

                    # Call the function to crop the video
                    crop_video(input_path, output_path)
```

Figure 3 Function to bulk crop in folders



```
if __name__ == "__main__":
    # Replace with the actual path to your dataset2 folder
    input_folder = "dataset2"
    # Replace with the desired output path
    output_folder = "dataset2_cropped"

    # Call the function to bulk crop videos
    bulk_crop_videos(input_folder, output_folder)
```

Figure 4 Calling the function

The script efficiently processes videos in bulk, categorizing them into train, test, and validation sets, and outputs the cropped videos to a designated folder. This comprehensive preprocessing workflow plays a crucial role in refining our training dataset.

Video Frame Generator

Prior to the application of our custom VGG16 model, a crucial preprocessing step involved the segmentation of videos in our datasets into sequential frames. This process was essential to enable the model to capture and understand temporal dependencies inherent in the dynamics of cricket shots. Utilizing Keras video[4] capabilities, we employed a batch size of 3 to simultaneously process multiple videos, while the "nb frames" parameter determined the number of frames extracted for each sequence. To enhance the diversity of our training data, we incorporated shuffling of video sequences within each batch during training, a step facilitated by setting the "shuffle" parameter to True. Additionally, the "transformation" parameter offered the flexibility to apply transformations such as resizing and cropping, ensuring the robustness of our model to variations in video content.

```
import keras_video
from keras_video import utils as ku

train = keras_video.VideoFrameGenerator(
    batch_size=3,
    nb_frames=15,
    shuffle=True,
    transformation=None,
    glob_pattern='dataset1/train/{classname}/*.avi'
)
```

Figure 5 Code for extracting video frames for training data



Figure 6 Example of frames extracted from each set of videos

Feature Engineering with Pretrained Models

In the realm of computer vision, feature engineering with pretrained models takes center stage, and one stalwart in this domain is VGG16[5]. As a convolutional deep neural network, VGG16 boasts a formidable architecture comprising 13 convolutional layers and 3 fully connected layers. This model has earned its stripes by being trained on the extensive ImageNet dataset, a repository of millions of labeled images spanning 1000 classes. What makes VGG16 particularly potent is its role as a pre-trained model, widely adopted across various computer vision tasks. The essence of Feature Engineering lies in harnessing the learned features from VGG16's deep layers, effectively utilizing them as powerful feature extractors. Transfer learning capitalizes on this approach, enhancing generalization and propelling the performance of computer vision tasks by transferring knowledge from one domain to another.

```
model = keras.models.Model(  
    inputs=base_model.input,  
    outputs=base_model.get_layer(  
        'block1_pool'  
    ).output  
)
```

Figure 7 Code to extract features from the first block pool

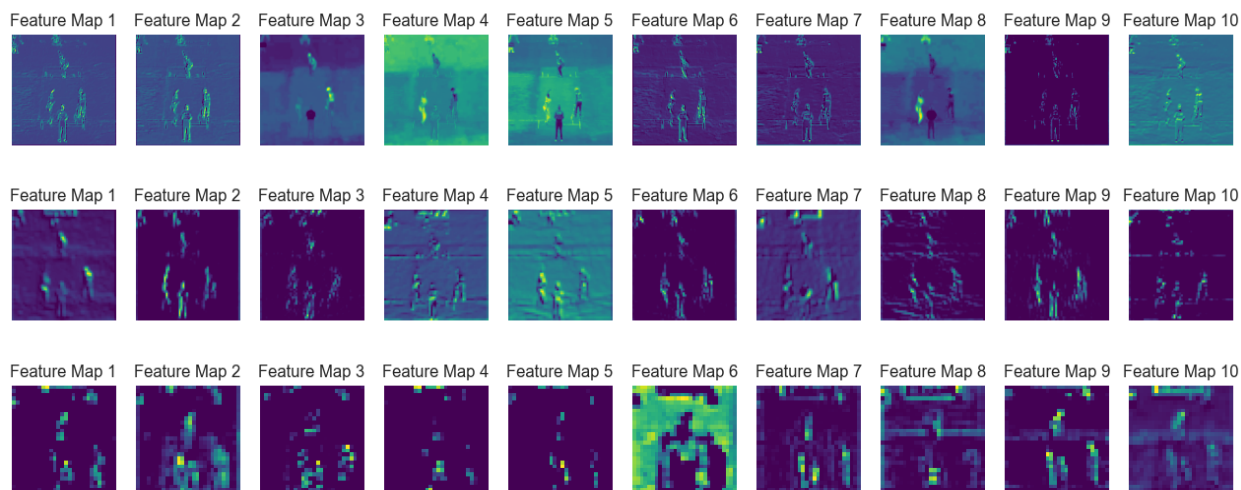


Figure 8 Features extracted for a single instance of video from block pool 1, 2 & 3

Google Cloud Datasets

To integrate our dataset seamlessly with Google Vertex AI, our initial step involved the creation of a dedicated dataset[6] on Google Cloud. This process entailed the upload of videos to Google Cloud Storage, followed by the creation of a dataset with a specified name. To populate this dataset, we utilized a JSON Lines file, containing URI links to each video residing in the cloud storage buckets, along with associated labels and timestamps. Subsequently, leveraging this prepared dataset, we proceeded to train our video classification AutoML model on Google Vertex AI, ensuring the integration of a tailored machine learning solution into our mobile application.

```
{
  "content":
    "gs://unseen_test/cover_03.mp4",
  "mimeType":
    "video/mp4",
  "timeSegmentStart":
    "0.0s",
  "timeSegmentEnd":
    "Infinity"
}
```

Figure 9 The format for the input file to import videos to the dataset

Add videos to your dataset

Before you begin, review the data guide to make sure your data is formatted correctly and optimized for the best results. Supported video file formats: MOV, MPEG-4, MP4, AVI

[VIEW DATA GUIDE](#)

Select an import method

- **Upload videos:** Recommended if you don't have labels yet
- **Import files:** Recommended if you already have labels. An import file is a list of Cloud Storage URIs to your videos and optional data, like labels. [Learn how to create an import file](#)

- ☐ Upload videos from your computer
- ☐ Upload import files from your computer
- ☒ Select import files from Cloud Storage

Select import files from Cloud Storage

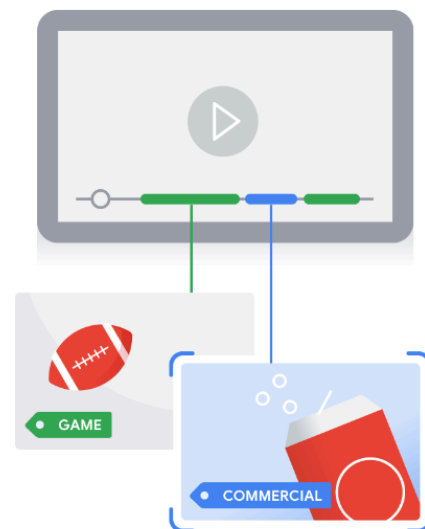
Videos referenced in the import files will be preprocessed and stored in a new Cloud Storage bucket ([charges apply](#))

Import file path *
☒ gs://unseen_test/unseen_test.jsonl

BROWSE

Data split
Default

Summary



Video classification models let you predict labels for video segments.

If you want to recognize a vast number of objects, places, and actions in your videos with Google's pre-trained models, try the Video Intelligence API. [Learn more](#)

Figure 10 The interface to add videos to your datasets on Google Cloud

Experimentation

This section delves into ML Experimentation for the shot classification model. We fine-tuned and tailored models to our needs, sharing results for the experimented models. Additionally, TensorFlow Lite was integrated to explore on-device processing, optimizing our machine learning efforts for local inferences. We also explored Google's AutoML, delving into automated machine learning to enhance our capabilities. This section unveiled insights gained through our iterative experimentation process.

Custom Models

Date	Model Architecture	Pre-Trained Model	Trainable Parameters	Input Shape	Training Time/GPU	Training Time/CPU	Accuracy/ Train	Accuracy/ Test
16-Nov	EfficientNetB2	Yes	No	(2, 10, 224, 224, 3)	NA	1 hr 12 mins	0.6128	0.525
19-Nov	EfficientNetB5	Yes	No	(2, 10, 224, 224, 3)	NA	1 hr 02 mins	0.625	0.455
20-Nov	EfficientNetB7	Yes	No	(3, 15, 224, 224, 3)	NA	1 hr 30 mins	0.5397	0.41
20-Nov	VGG16	Yes	Yes	(3, 15, 224, 224, 3)	NA	approx 2 hr	0.12	0.11
22-Nov	EfficientNetB7	Yes	No	(3, 15, 224, 224, 3)	NA	approx 3 hr	0.705	0.495
26-Nov	VGG16	Yes	No	(3, 15, 224, 224, 3)	19 mins	NA	1	0.707

Figure 11 Results from our model experimentation

EfficientNet

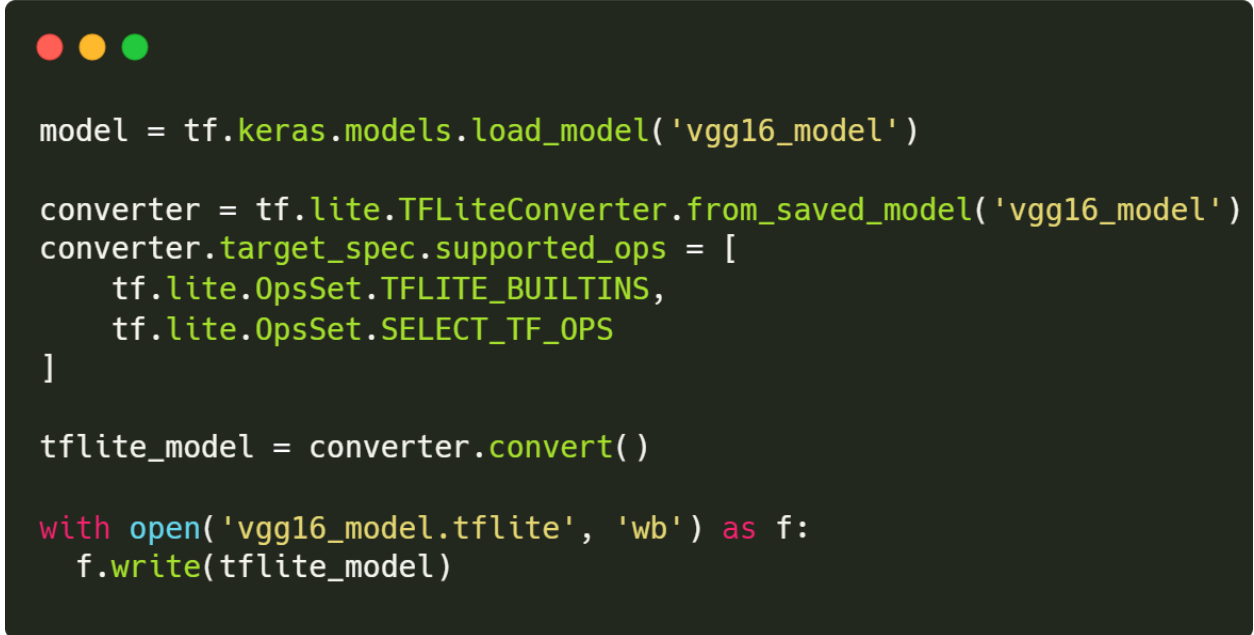
EfficientNet[7], known for its efficiency and compact architecture, plays a crucial role in enabling fast on-device inference. Its significance in computer vision lies in striking a balance between model size and performance, making it ideal for resource-constrained environments. This lightweight yet powerful model ensures that ShotSense can deliver robust shot classification capabilities with minimal computational resources, enhancing the user experience by enabling swift and efficient processing on mobile devices. EfficientNetB7 achieved an accuracy of 49.5% showing proof of concept with room for improvement.

VGG16

Leveraging its deep architecture and pretrained knowledge from the ImageNet dataset, VGG16 brings strong feature extraction capabilities to our application. Despite its larger size compared to some newer models, VGG16's well-established performance and versatility contribute significantly to accurate shot analysis. We kept the trainable layers frozen and used the deep learning layers for feature extraction. The VGG16 model we trained achieved an impressive accuracy of 70.7% on the test set.

TensorFlow Lite

TensorFlow Lite (TFLite)[8] serves as a crucial component in enabling efficient on-device inference for ShotSense. This lightweight version of TensorFlow is tailored for mobile and edge devices, ensuring swift and resource-efficient execution of machine learning models. TFLite supports our goal of delivering real-time shot classification on users' devices, enhancing the application's responsiveness and user experience. Below we can see the code implemented to optimize our model for on device inference and then exporting it as a TFLite file.



```
model = tf.keras.models.load_model('vgg16_model')

converter = tf.lite.TFLiteConverter.from_saved_model('vgg16_model')
converter.target_spec.supported_ops = [
    tf.lite.OpsSet.TFLITE_BUILTINS,
    tf.lite.OpsSet.SELECT_TF_OPS
]

tflite_model = converter.convert()

with open('vgg16_model.tflite', 'wb') as f:
    f.write(tflite_model)
```

Figure 12 Code for converting SavedModel to TFLite

The process involves loading the pretrained VGG16 model and using the TFLiteConverter to convert it to a compressed format suitable for deployment on mobile devices. The resulting 'vgg16_model.tflite' file represents a streamlined version of the original model, ensuring efficient execution.

Google’s AutoML

Google AutoML[9] is a user-friendly and efficient tool that streamlines machine learning model training without requiring extensive expertise. Our integration with AutoML facilitated easy customization of model parameters and training processes, accelerating development and democratizing machine learning capabilities.

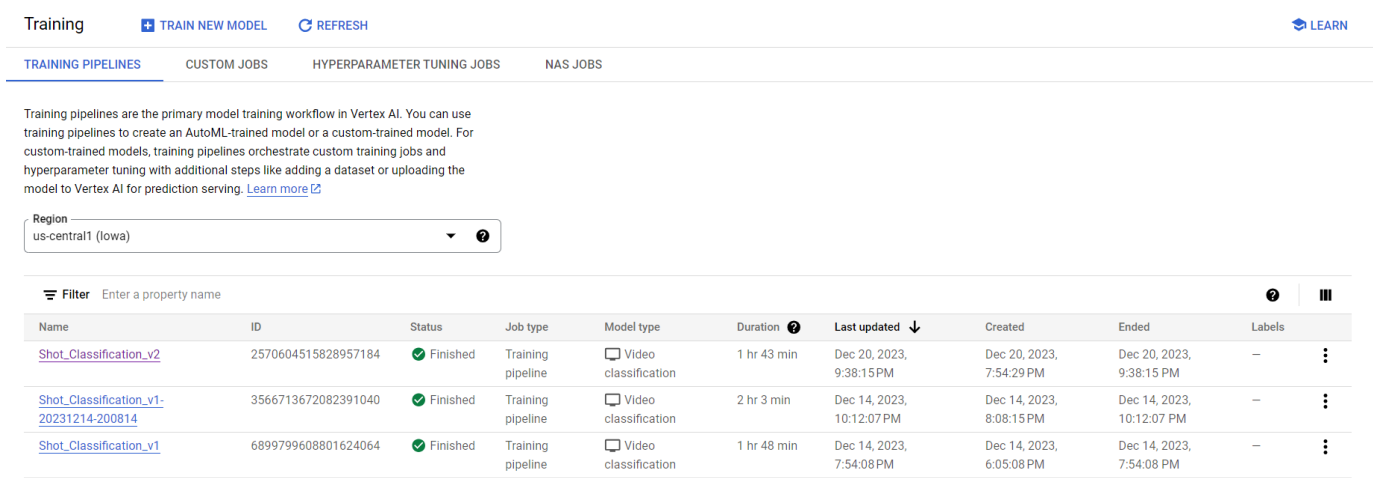


Figure 13 Implementing training pipelines on Google Vertex AI

Our AutoML video classification model performed far better than any of our custom models. They managed to achieve a F1 score of 88.5% with average precision under the curve of 94.3%.

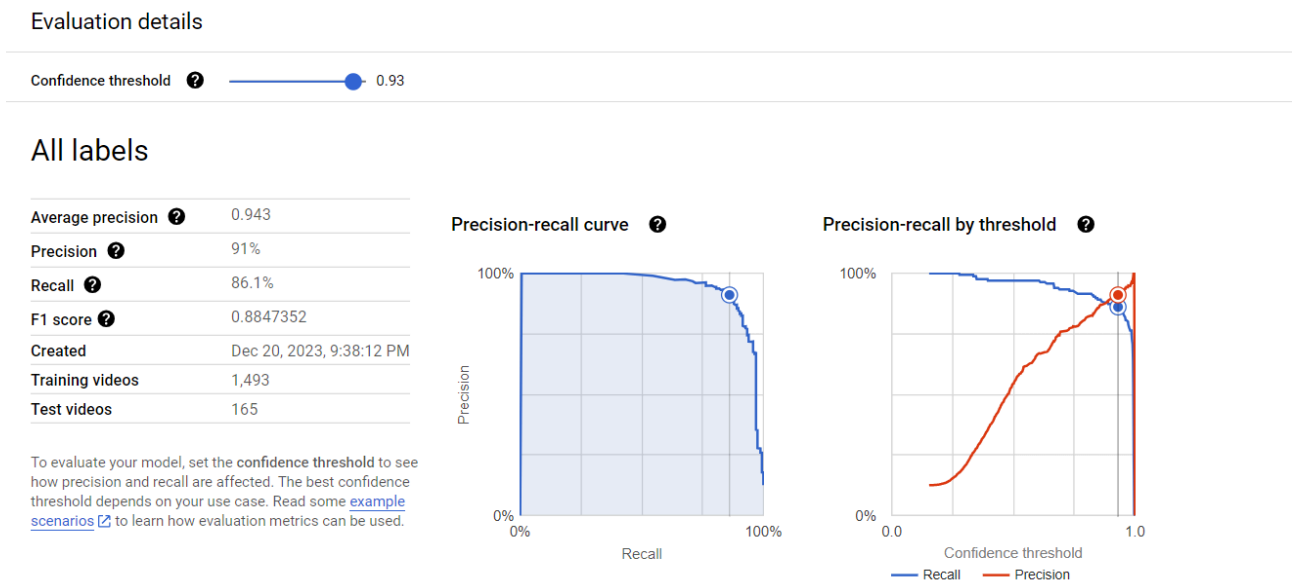


Figure 14 Evaluation details on Vertex AI

Mobile Application Development

In this section, we delve into the chosen development platform and tools that form the backbone of our application's functionality.

Development Platform and Tools

Flutter

The Development platform for mobile application is chosen to be versatile, working for both android and ios platforms. By using cloud based storage and AI processing, accessing the native features for processing in these platforms is on a low priority.

Flutter[10] is chosen to be the tool for creating the mobile application as it supports all the major mobile operating systems and has massive support from google itself creating a smooth developer experience. To encourage clean code, standard development practices in flutter have been adapted separating widgets, APIs and screens for each component of the application.

Firebase

Firebase[11] is an excellent tool for handling all the backend functionalities and seamlessly integrates with Flutter, our chosen development framework. Firebase provides a comprehensive suite of services, including real-time database, authentication, cloud functions, and cloud storage, addressing our backend requirements effectively.

Cloud Services

Our development strategy emphasizes cloud-based storage and AI processing to maximize scalability and minimize platform-specific dependencies. By leveraging cloud-based solutions, we ensure seamless data storage and retrieval, facilitating efficient sharing and processing of user-uploaded videos. The chosen cloud infrastructure offers a centralized repository, enhancing accessibility and collaboration across platforms.

ML Model Selection

After a comprehensive evaluation of various models through experimentation, the final choice for the ShotSense application is Google's AutoML. This decision is grounded in the multifaceted advantages that Google Cloud Platform offers, providing an end-to-end ML pipeline. Leveraging Google Cloud Storage ensures secure and efficient data storage, while Vertex AI streamlines the entire process, encompassing training, development, and deployment of the model.

The decision to opt for Google AutoML is also influenced by its capability to provide faster inferences compared to on-device predictions. This feature aligns seamlessly with our goal of delivering a responsive and efficient shot classification experience to users. The integration with AutoML not only enhances the application's performance but also signifies a commitment to leveraging state-of-the-art technologies for optimal results. This strategic choice positions ShotSense on a trajectory of continuous improvement and scalability in its machine learning capabilities.

Mobile Application Flow

The application operates seamlessly, offering users a comprehensive platform for analyzing their cricket shots. Upon video upload to the cloud, a robust backend, powered by a Node server and Google's Vertex AI, orchestrates the processing through batch prediction of vertex ai. This AI system discerns shot classifications from videos capturing an entire over (6 balls) played by the user. Using shot classification AI, predictions for each ball are meticulously generated and stored in a Firebase Firestore collection. The user is then presented with a user-friendly display, showcasing detailed shot predictions derived from the processed videos.

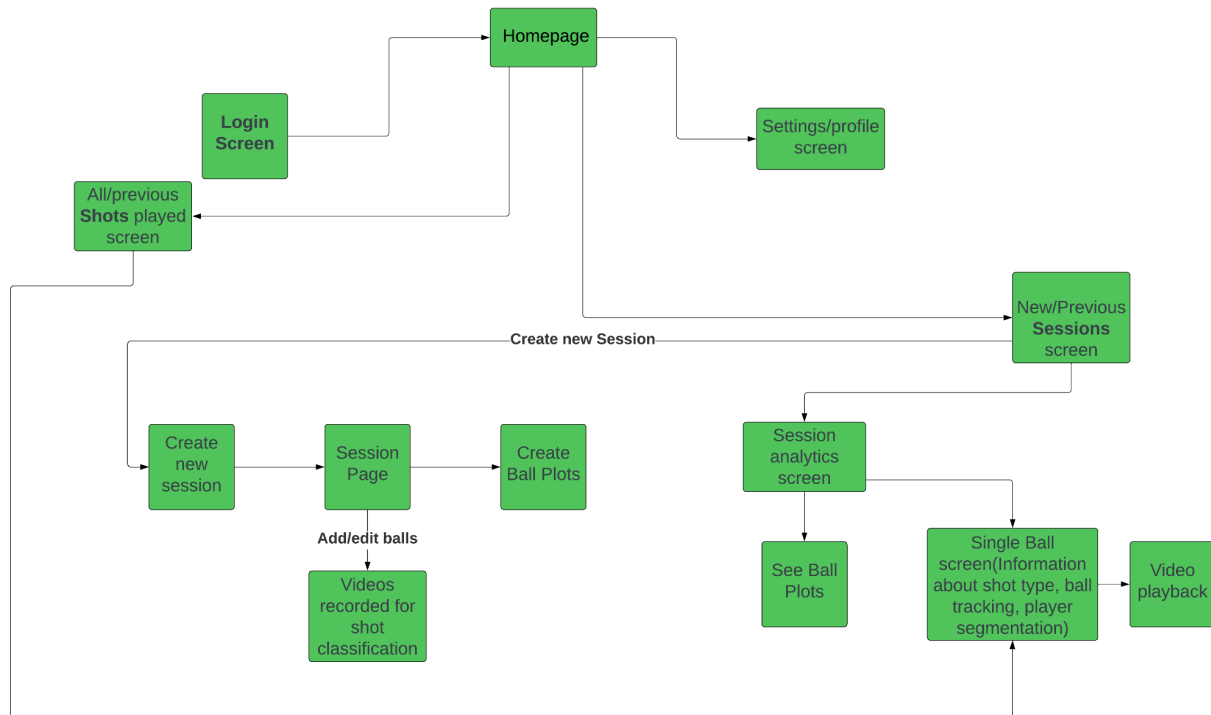


Figure 15 Mobile Application flow diagram

Functionality

The mobile application is designed to gain visualization for a player's cricket performance. Separating player's playing sessions allows players to gain insight into their playing patterns and encourage them to use the application to accumulate their results over time.

Having recording and uploading options right through the application makes it convenient and shows the adaptive nature of the application encouraging anyone novice with a camera to record themselves and gain insights.

The application's innovative features, such as pose tracking and player segmentation, add an exciting dimension to the user experience. These cutting-edge tools elevate the user experience by providing detailed insights into body movements, form, and player positioning during each shot.

With shot classification the players can learn the different types of shots, this application uses a shot classification AI model to predict up to 8 different shot types. Giving players an opportunity to practice and perfect their shots and be ready for the field.

Data Management

Effective data management is pivotal for providing players with meaningful insights and facilitating continuous improvement. Our mobile application employs robust data management strategies to ensure seamless handling and insightful analysis.

Shotsense uses Google Cloud Storage[12] to store the video cloud storage allows us to seamlessly integrate with vertex AI and making it easier to deploy the application to all platforms.

←

Bucket details

REFRESH

LEARN

unseen_test

Location

Storage class

Public access

Protection

us-central1 (Iowa)

Standard

Not public

None

OBJECTS

CONFIGURATION

PERMISSIONS

PROTECTION

LIFECYCLE

OBSERVABILITY

INVENTORY REPORTS

Buckets > unseen_test > predictions > prediction-Shot_Classification_v2-2023-12-21T17:46:02.174247Z

UPLOAD FILES

UPLOAD FOLDER

CREATE FOLDER

TRANSFER DATA

MANAGE HOLDS

DOWNLOAD

DELETE

Filter by name prefix only

Filter

Filter objects and folders

Show deleted data

<input type="checkbox"/>	Name	Size	Type	Created	Storage class	Last modified	Public access	
<input type="checkbox"/>	errors.jsonl	0 B	application/octet-stream	Dec 21, 2023, 10:49:05 PM	Standard	Dec 21, 2023, 10:49:05 PM	Not public	
<input type="checkbox"/>	predictions_00001.jsonl	627 B	application/octet-stream	Dec 21, 2023, 10:49:05 PM	Standard	Dec 21, 2023, 10:49:05 PM	Not public	
<input type="checkbox"/>	predictions_00002.jsonl	618 B	application/octet-stream	Dec 21, 2023, 10:49:16 PM	Standard	Dec 21, 2023, 10:49:16 PM	Not public	
<input type="checkbox"/>	predictions_00003.jsonl	150 B	application/octet-stream	Dec 21, 2023, 10:49:11 PM	Standard	Dec 21, 2023, 10:49:11 PM	Not public	
<input type="checkbox"/>	predictions_00004.jsonl	630 B	application/octet-stream	Dec 21, 2023, 10:49:07 PM	Standard	Dec 21, 2023, 10:49:07 PM	Not public	
<input type="checkbox"/>	predictions_00005.jsonl	964 B	application/octet-stream	Dec 21, 2023, 10:49:14 PM	Standard	Dec 21, 2023, 10:49:14 PM	Not public	
<input type="checkbox"/>	predictions_00006.jsonl	969 B	application/octet-stream	Dec 21, 2023, 10:49:06 PM	Standard	Dec 21, 2023, 10:49:06 PM	Not public	
<input type="checkbox"/>	predictions_00007.jsonl	659 B	application/octet-stream	Dec 21, 2023, 10:49:10 PM	Standard	Dec 21, 2023, 10:49:10 PM	Not public	
<input type="checkbox"/>	predictions_00008.jsonl	957 B	application/octet-stream	Dec 21, 2023, 10:49:09 PM	Standard	Dec 21, 2023, 10:49:09 PM	Not public	
<input type="checkbox"/>	predictions_00009.jsonl	813 B	application/octet-stream	Dec 21, 2023, 10:49:09 PM	Standard	Dec 21, 2023, 10:49:09 PM	Not public	
<input type="checkbox"/>	predictions_00010.jsonl	975 B	application/octet-stream	Dec 21, 2023, 10:49:13 PM	Standard	Dec 21, 2023, 10:49:13 PM	Not public	
<input type="checkbox"/>	predictions_00011.jsonl	810 B	application/octet-stream	Dec 21, 2023, 10:49:16 PM	Standard	Dec 21, 2023, 10:49:16 PM	Not public	
<input type="checkbox"/>	predictions_00012.jsonl	323 B	application/octet-stream	Dec 21, 2023, 10:49:08 PM	Standard	Dec 21, 2023, 10:49:08 PM	Not public	
<input type="checkbox"/>	predictions_00013.jsonl	808 B	application/octet-stream	Dec 21, 2023, 10:49:11 PM	Standard	Dec 21, 2023, 10:49:11 PM	Not public	
<input type="checkbox"/>	predictions_00014.jsonl	800 B	application/octet-stream	Dec 21, 2023, 10:49:18 PM	Standard	Dec 21, 2023, 10:49:18 PM	Not public	
<input type="checkbox"/>	predictions_00015.jsonl	800 B	application/octet-stream	Dec 21, 2023, 10:49:11 PM	Standard	Dec 21, 2023, 10:49:11 PM	Not public	
<input type="checkbox"/>	predictions_00016.jsonl	402 B	application/octet-stream	Dec 21, 2023, 10:49:12 PM	Standard	Dec 21, 2023, 10:49:12 PM	Not public	

Figure 16 Vertex AI predictions for the videos uploaded

Firebase Firestore a NoSql document database is used to store the users information including sessions, user data, shot predictions and more. Firestore grants flexibility in the database schema, allowing for easy adaptation to changing application requirements. This flexibility is particularly valuable as the application evolves, ensuring that Firestore can seamlessly accommodate new features, user attributes, and performance metrics without compromising data integrity or system stability.

Test Cases

The test cases are referred from the requirements analysis document and presented here for the implemented functionality of the application.

View a session from history

This test case refers to TC-06 from the requirement analysis document. Please refer to it for complete details.

Objective: To verify that the system provides users with a complete analytical summary of a selected session, including key statistics, graphical representations, and access to individual video clips for in-depth analysis.

Expected Results: The "Sessions" tab is accessible within the user interface.

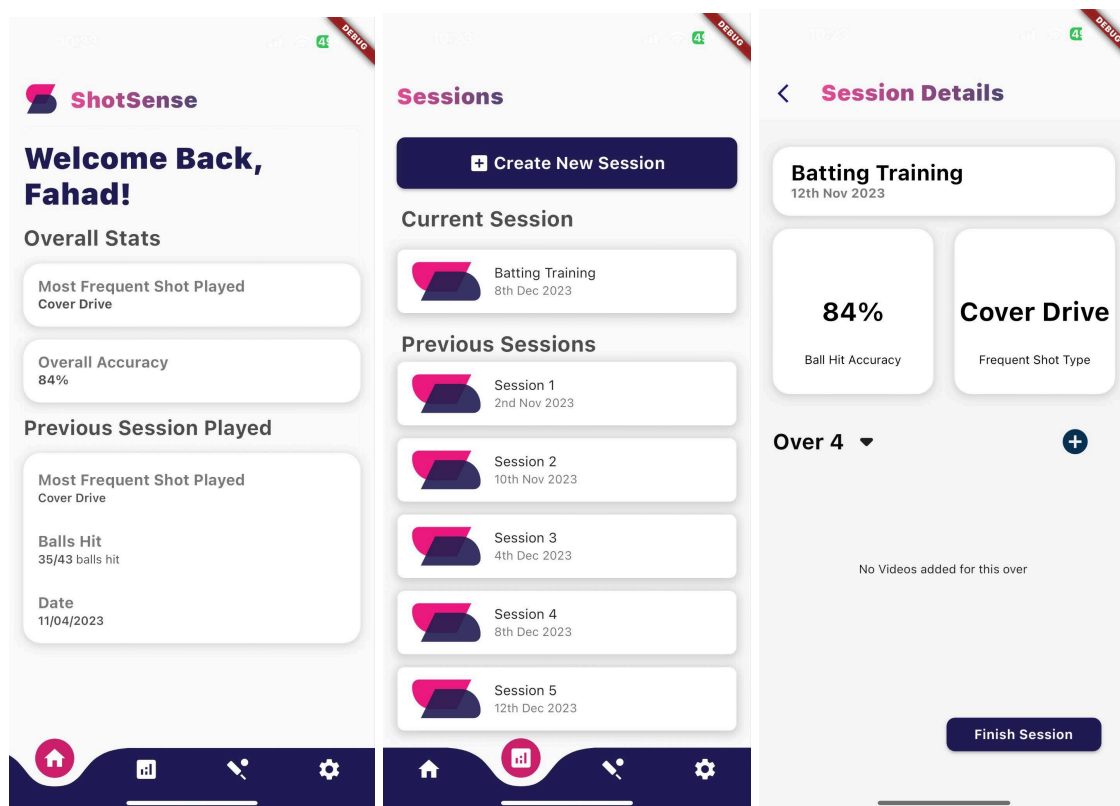


Figure 17 Screens showing homepage, sessions tab and a session detail/summary

Description: Within the application, users have the option to navigate to the "Sessions" tab, where both ongoing and past sessions are conveniently listed. Upon selecting a specific session, users gain access to pertinent information, as illustrated in the screenshots above. The provided details encompass relevant statistics, graphical data representations, and the ability to view individual video clips for a thorough analysis of the chosen session.

Add new balls to session from gallery

This test case refers to TC-04 from the requirement analysis document. Please refer to it for complete details.

Objective: To verify that the system allows players to successfully select videos from gallery and upload video clips for a ball, associating them with the specified over in the current session.

Expected Results: The video is uploaded and predictions are visible to the user in the over.

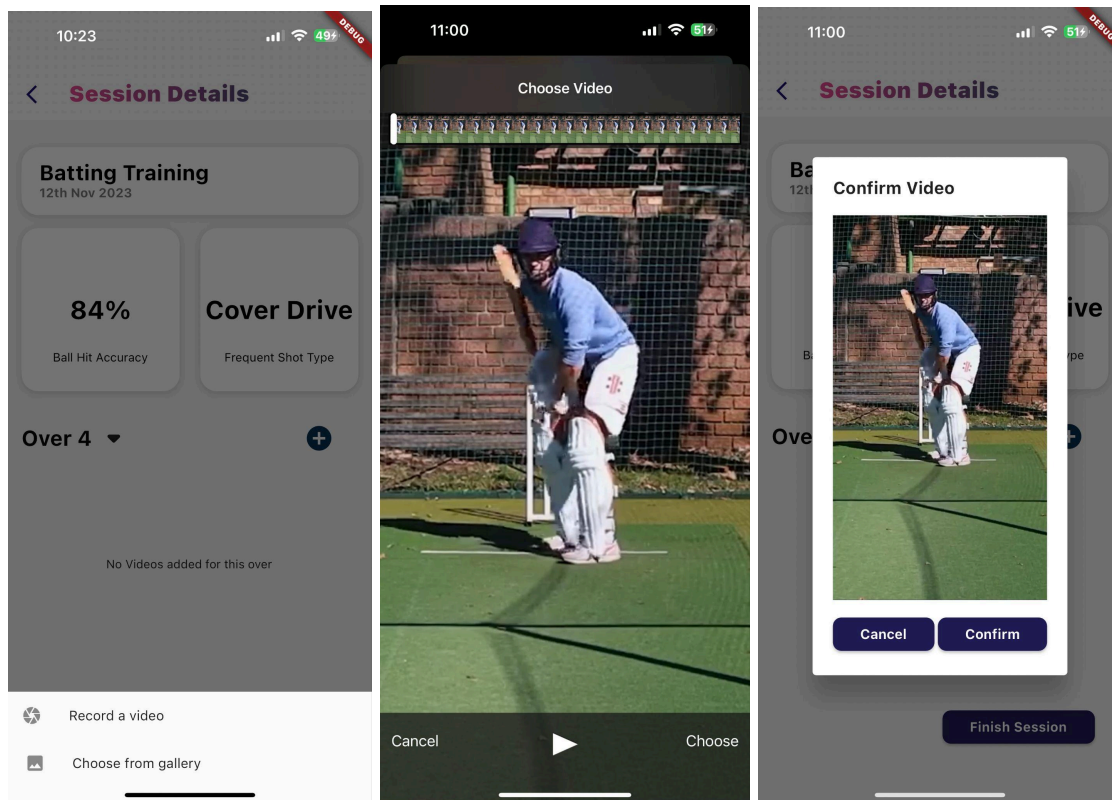


Figure 18 showing the process of uploading a video from gallery

Description: Users can seamlessly commence the video uploading process for a specific ball by engaging with the "Upload from gallery" button available within the current session interface. Following the upload, the system promptly displays the video to the user. To finalize the ball and initiate processing, the user simply presses the "Confirm" button, thus indicating their satisfaction with the uploaded video. This streamlined process ensures user-friendly interaction and efficient handling of uploaded content within the application.

Record balls in session from camera

This test case refers to TC-03 from the requirement analysis document. Please refer to it for complete details.

Objective: To verify that the system allows players to successfully record and upload video clips for an over, associating them with the specified over in the session summary.

Expected Results: The video is uploaded and predictions and visible to the user in the over

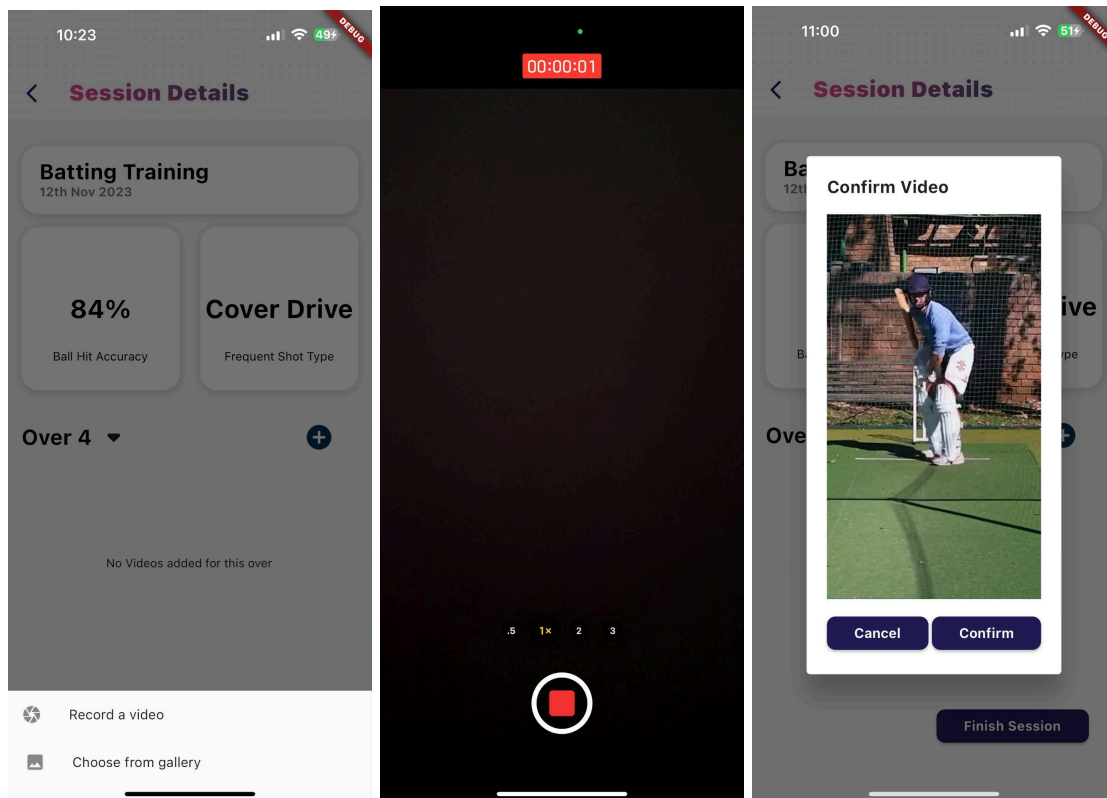


Figure 19 showing the process of uploading a video from camera

Description: Within the session interface, users can commence video recording for a specific over by selecting the "Record" button. Following the recording, the system promptly displays the video to the user for review. To confirm the ball and initiate processing, the user simply presses the "Confirm" button, indicating their satisfaction with the recorded video. This straightforward process ensures a seamless user experience and efficient management of recorded content within the application.

Display shot history by shot type

This test case refers to TC-07 from the requirement analysis document. Please refer to it for complete details.

Objective: To verify that the system allows users to view their shots from all sessions combined, filtered by specific shot types.

Expected Results: The system compiles shots from all sessions and displays them in their respective category.

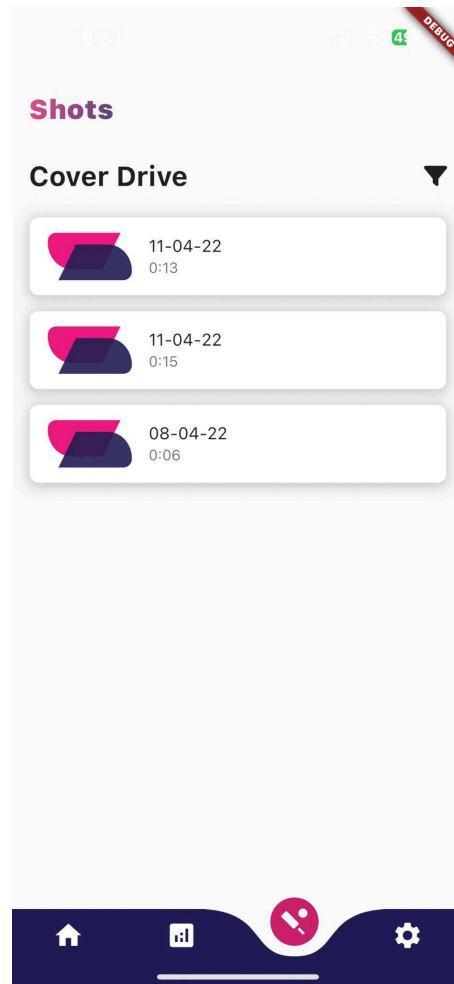


Figure 20 Screen of “Shots” tab in the mobile application

Description: The system consolidates shot videos from all sessions and presents them to users for easy access and analysis. Users have the option to apply filters such as "Cover drive," "Pull shots," or other shot types to refine the displayed content. The shots are then categorized according to the chosen shot type filter, providing users with a targeted and organized view of specific shot categories. This feature enhances the user's ability to review and compare shots of interest across various sessions, contributing to a more insightful analysis of their gameplay.

View shot analysis per ball

This test case refers to TC-08 from the requirement analysis document. Please refer to it for complete details.

Objective: To verify that the system allows users to access in-depth shot analysis for each ball played within a selected over.

Expected Results: The system should present a list of balls played within the over for user selection.

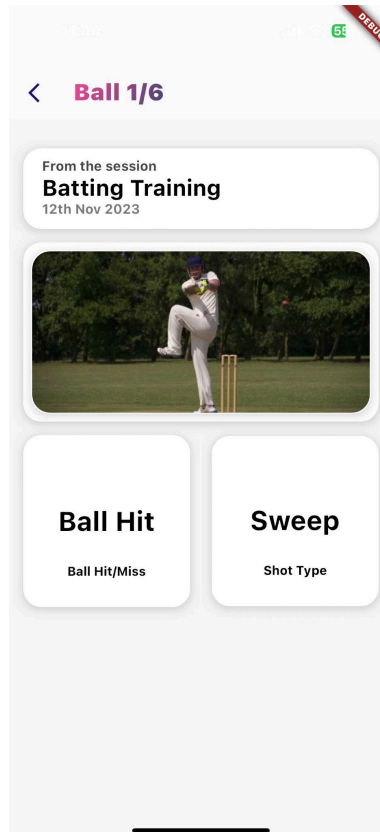


Figure 21 Screen showing analytics in a single page

Description: The system facilitates a user-friendly interface by presenting a list of balls played within the selected over. Users can choose a specific ball for in-depth analysis. Upon selecting a ball, the system provides a comprehensive shot analysis, encompassing shot classification (shot type), video playback (featuring ball tracking and pose detection), and a determination of whether the shot was a hit or a miss. This detailed analysis is accessible in the "Shot" tab or within any of the current or previous session pages. Users can initiate this analysis by clicking on a specific ball of interest, enabling them to scrutinize and understand the nuances of each shot with ease.

Conclusion

The Implementation Document serves as a comprehensive guide to transforming design concepts into a fully functional ShotSense application. It meticulously details the ML development process, data procurement, preprocessing steps, and the chosen platforms and tools, including Flutter, Firebase, and cloud services. The document also provides a glimpse into the experimentation phase, showcasing the exploration of custom models, TensorFlow Lite, and Google's AutoML. From the intricate data management strategies to the effective utilization of Google Cloud Datasets, the implementation journey is thoroughly documented, ensuring accessibility and understanding for all stakeholders. This transparent and informative guide sets the foundation for the successful development and deployment of the ShotSense application.

The robust mobile application, developed on Flutter and integrated seamlessly with Firebase and cloud services, offers users an intuitive platform for cricket shot analysis. With features like pose tracking, player segmentation, and shot classification, ShotSense provides a holistic experience for players to visualize and enhance their performance. The implementation document's conclusion encapsulates the essence of the chosen ML model, Google's AutoML, and the rationale behind leveraging Google Cloud Platform, affirming ShotSense's commitment to cutting-edge technologies. The document underlines the strategic decisions made throughout the development, ensuring scalability, efficiency, and a continual pursuit of optimal user experience.

References

1. Sen, Anik, et al. "CricShotClassify: an approach to classifying batting shots from cricket videos using a convolutional neural network and gated recurrent unit." *Sensors* 21.8 (2021): 2846.
2. FFMpeg Documentation. (n.d.). <https://ffmpeg.org/documentation.html>
3. OpenCV. (n.d.). OpenCV. <https://forum.opencv.org/>
4. Metal3d, 'Keras Sequence Video Generators', Github (Source Code), <https://github.com/metal3d/keras-video-generators>
5. Rohini G, 'Everything you need to know about VGG16', <https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918>
6. Vertex AI Documentation, Create a dataset for training video classification models, <https://cloud.google.com/vertex-ai/docs/video-data/classification/create-dataset>
7. Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." *International conference on machine learning*. PMLR, 2019.
8. TensorFlow lite. (n.d.). TensorFlow. <https://www.tensorflow.org/lite/guide>
9. AutoML Google Cloud. <https://cloud.google.com/vertex-ai/docs/beginner/beginners-guide>
10. Flutter documentation. (n.d.). Flutter. <https://docs.flutter.dev/>
11. Documentation. (n.d.-b). Firebase. <https://firebase.google.com/docs>
12. Cloud Storage documentation. (n.d.). Google Cloud. <https://cloud.google.com/storage/docs>