Studenti: Stefano Arasi, Marko P’yeshchyk

Github progetto: https://github.com/ArasiStefanoISS/Sistemi-Digitali

# Android

## Introduzione

Questa applicazione sviluppata in Kotlin per android ha come scopo l’utilizzo della telecamera per la ripresa di un video di un utente e l’applicazione di azioni su di un Video Player per permettere la visione senza bisogno dell’utilizzo di un input diretto da parte dell’utente.

Una volta che il video registrato è stato ripreso viene analizzato da una rete neurale per riconoscere quale dei movimenti tra quelli utilizzati per il suo addestramento è stato eseguito e verrà eseguita una azione a seconda del risultato.

Le azioni disponibili con la corrispondente azione sono:

* Fare il pollice in su: aumenta il volume del dispositivo.
* Fare il pollice in giù: diminuisce il volume del dispositivo.
* Fare un simbolo di stop con il palmo della mano: mettere in pausa il video (o riprendere la riproduzione se già in pausa)
* Muovere la mano verso destra: manda il video avanti di 10 secondi.
* Muovere la mano verso sinistra: manda il video indietro di 10 secondi.

## Implementazione

### Pattern di design

L’applicazione è stata sviluppata secondo il pattern di design MVC (Model View Controller) perché è un'architettura software molto usata, specialmente quelle con interfaccia utente. Il suo scopo principale è separare la logica di presentazione dalla logica di business, migliorando la manutenibilità, testabilità e riusabilità del codice.

### Controller

Il controller rappresenta il punto di connessione tra dati ed interfaccia grafica dell’applicazione, nel caso della nostra applicazione esegue anche l’inizializzazione del sistema.

L’inizializzazione dell’applicazione viene eseguito nel seguente modo:

class MainActivity : AppCompatActivity() {  
 private lateinit var volumeManager:VolumeManager;  
 private lateinit var cameraManager: CameraManager;  
 private lateinit var videoManager: VideoManager;  
  
 private val REQUIRED\_PERMISSIONS = *arrayOf*(  
 Manifest.permission.*CAMERA*,  
 Manifest.permission.*READ\_MEDIA\_VIDEO*  
)  
 private val REQUEST\_CODE\_PERMISSIONS = 10  
  
 //set up  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 setContentView(R.layout.*activity\_main*)  
  
 *requestedOrientation* = ActivityInfo.*SCREEN\_ORIENTATION\_PORTRAIT*  
  
volumeManager = VolumeManager(*applicationContext*)  
  
 cameraManager = CameraManager(this, findViewById(R.id.*previewView*))  
  
 videoManager = VideoManager(this, findViewById(R.id.*videoView*));  
  
  
 if (allPermissionsGranted()) {  
 cameraManager.startCamera()  
 } else {  
 ActivityCompat.requestPermissions(this, REQUIRED\_PERMISSIONS, REQUEST\_CODE\_PERMISSIONS)  
 }  
  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main*)) **{** v, insets **->**  
val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}**  
}  
  
 private fun allPermissionsGranted() = REQUIRED\_PERMISSIONS.*all* **{**  
ContextCompat.checkSelfPermission(this, **it**) == PackageManager.*PERMISSION\_GRANTED*  
**}**  
  
  
override fun onRequestPermissionsResult(  
 requestCode: Int,  
 permissions: Array<out String>,  
 grantResults: IntArray  
 ) {  
 super.onRequestPermissionsResult(requestCode, permissions, grantResults)  
  
 if (requestCode == REQUEST\_CODE\_PERMISSIONS) {  
 if (grantResults.*isNotEmpty*() && grantResults[0] == PackageManager.*PERMISSION\_GRANTED*  
&& grantResults[1] == PackageManager.*PERMISSION\_GRANTED*) {  
 // Both permissions granted: Start Camera & Play Video  
 //startCamera()  
 } else {  
 // If any permission is denied, show a toast message and close the app  
 Toast.makeText(this, "All permissions are required", Toast.*LENGTH\_SHORT*).show()  
 finish()  
 }  
 }  
 }

}

Inizialmente vengono definiti gli oggetti che faranno riferimento alle classi del Model per la gestione della camera, del video player e del volume.

Poi vengono definiti i permessi da richiedere, CAMERA per accedere alla camera, READ\_MEDIA\_VIDEO.

La funzione onCreate() viene richiamata come prima cosa nell’applicazione.

Questa prepara il layout dell’applicazione e fissa il layout dell’applicazione a verticale.

Poi vengono inizializzati i tre manager descritti precedentemente.

Viene poi controllato se i permessi sono già stati concessi, se sono concessi viene attivata la camera, altrimenti richiede i permessi all’utente.

Nel caso i permessi vengano negati l’applicazione viene chiusa.

Come ultima cosa fa in modo che l’applicazione non venga coperta dalla barra di stato, barra di navigazione, etc.

Vengono poi definite le seguenti funzioni per gestire l’interazione dell’utente con l’ui:

private val PICK\_VIDEO\_REQUEST = 1  
  
fun onPickVideo(view: View) {  
 val intent = Intent(Intent.*ACTION\_PICK*)  
 intent.*type* = "video/\*"  
 startActivityForResult(intent, PICK\_VIDEO\_REQUEST)  
}  
  
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
 super.onActivityResult(requestCode, resultCode, data)  
  
 if (requestCode == PICK\_VIDEO\_REQUEST && resultCode == *RESULT\_OK* && data != null) {  
 val videoUri: Uri? = data.*data*  
videoUri?.*let* **{** videoManager.loadAndPlayVideo(**it**) **}**  
}  
}

Viene inizialmente richiesto all’utente di scegliere un video, una volta scelto viene ricavato il suo uri e passato al [CameraManager](#_CameraManager) per riprodurre quel video.

fun startOrStopRecording(view: View) {  
 UpdateButton();  
  
 cameraManager.startOrStopRecording(  
 onVideoSaved = **{** uri **->**  
if (uri == null) {  
 Toast.makeText(this, "Something went wrong while trying to record the video", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}**,  
 onClassificationDone = **{** result **->**  
val classificationResult = result  
 // Now you can use the result  
 Log.i("Classification Result","${classificationResult}");  
  
  
 when (classificationResult) {  
 1 -> behindVideo();  
 2 -> aheadVideo();  
 3 -> pauseVideo();  
 4 -> lowerVolume();  
 5 -> raiseVolume();  
 }  
 **}**  
)  
}

Questa funzione fa partire o ferma la registrazione avviata attraverso [CameraManager](#_CameraManager), nel caso ci sia un errore viene fatto visualizzare all’utente.

Questa funzione restituirà anche il risultato della classificazione della rete neurale utilizzata per riconoscere l’azione registrata.

A seconda del risultato viene applicata una delle varie seguenti funzioni:

//Neural Network applicable functions  
  
fun pauseVideo(view: View) {  
 videoManager.togglePlayPause()  
}  
  
fun aheadVideo(view: View) {  
 videoManager.seekForward()  
}  
  
fun behindVideo(view: View) {  
 videoManager.seekBackward()  
}  
  
  
fun lowerVolume(){  
 volumeManager.lowerVolume();  
}  
  
fun raiseVolume(){  
 volumeManager.raiseVolume();  
}

Ciascuna richiama la propria classe per gestire l’esecuzione ed eseguono rispettivamente:

* Mettere in pausa/riprendere la riproduzione del video;
* Mandare avanti il video di 10 secondi;
* Mandare indietro il video di 10 secondi;
* Diminuire il volume di una unità;
* Aumentare il volume di una unità;

fun UpdateButton(){  
 val button: Button = findViewById(R.id.*recordButton*)  
 if(isRecording){  
 button.*text* = "Record";  
 isRecording=false;  
 } else {  
 button.*text* = "Stop Recording";  
 isRecording=true;  
 }  
}

La funzione UpdateButton cambia il testo del bottone per rispecchiare lo stato dell’applicazione.

fun onFlipCamera(view: View){  
 cameraManager.FlipCamera();  
}

Questa funzione viene eseguita quando l’utente clicca sull’icona per cambiare li lato in cui la telecamera sta guardando, da lato interno a lato esterno o viceversa.

### Model

Il modello rappresenta i dati e la logica di business dell’applicazione.

Nel nostro caso comprende 3 classi: CameraManager, VideoManager, VolumeManager.

#### CameraManager

È la classe che gestisce l’utilizzo della camera nell’applicazione.

La libreria utilizzata per la camera è CameraX, ufficiale di android.

Per poterla utilizzare nell’applicazione è necessario implementare la libreria nel file gradle in questo modo:

// CameraX core library using the camera2 implementation  
val camerax\_version = "1.5.0-alpha06"  
// The following line is optional, as the core library is included indirectly by camera-camera2  
*implementation*("androidx.camera:camera-core:${camerax\_version}")  
*implementation*("androidx.camera:camera-camera2:${camerax\_version}")  
// If you want to additionally use the CameraX Lifecycle library  
*implementation*("androidx.camera:camera-lifecycle:${camerax\_version}")  
// If you want to additionally use the CameraX VideoCapture library  
*implementation*("androidx.camera:camera-video:${camerax\_version}")  
// If you want to additionally use the CameraX View class  
*implementation*("androidx.camera:camera-view:${camerax\_version}")  
// If you want to additionally add CameraX ML Kit Vision Integration  
*implementation*("androidx.camera:camera-mlkit-vision:${camerax\_version}")  
// If you want to additionally use the CameraX Extensions library  
*implementation*("androidx.camera:camera-extensions:${camerax\_version}")

Inoltre, per poterlo utilizzare è necessario ricevere i permessi di utilizzare la telecamera da parte dell’utente per motivi di sicurezza.

Questa classe utilizza anche il la neural network per discriminare quale delle azioni riconosciute è stata fatta, importando ed utilizzando il suo modello.

Una volta che l’utente permette l’utilizzo della camera una istanza di CameraX viene creata attraverso la seguente funzione:

class CameraManager(private val context: Context, private val previewView: PreviewView) {  
 private var videoCapture:VideoCapture<Recorder>?=null;  
 private var currentRecording:Recording?=null;  
 private val model = Model.newInstance(context);

private var lensFacing = CameraSelector.*LENS\_FACING\_BACK*;

fun startCamera() {  
 val cameraProviderFuture = ProcessCameraProvider.getInstance(context)  
  
 cameraProviderFuture.addListener(**{**  
val cameraProvider = cameraProviderFuture.get()  
  
 val preview = Preview.Builder().build().*also***{**  
 **it**.setSurfaceProvider(previewView.*surfaceProvider*)  
 **}**  
  
val recorder = Recorder.Builder()  
 .setQualitySelector(QualitySelector.from(Quality.*HIGHEST*))  
 .build()  
  
 videoCapture = VideoCapture.withOutput(recorder)  
  
 val cameraSelector = CameraSelector.Builder()  
 .requireLensFacing(lensFacing)  
 .build()  
  
 try {  
 cameraProvider.unbindAll()  
 cameraProvider.bindToLifecycle(  
 context as androidx.lifecycle.LifecycleOwner, cameraSelector, preview, videoCapture  
 )  
 } catch (exc:Exception){  
 Log.e("VideoManager", "Use case binding failed", exc)  
 }  
 **}**,ContextCompat.getMainExecutor(context))  
}

Innanzi tutto, viene creata un’istanza di ProcessCameraProvider che gestisce i casi d’uso di CameraX e restituirà una CameraProvider una volta pronto.

Poi viene creato l’oggetto preview che crea il caso d’uso preview video e lo collega alla PreviewView, in modo che la fotocamera venga mostrata a schermo.

Vengono poi creati un recorder per poter registrare un video e un videoCapture che lo utilizza.

Viene poi selezionata la camera posteriore.

Infine, disconnette qualsiasi precedente caso d’uso e collega la fotocamera al ciclo di vita di context, avviando la preview e preparando la registrazione.

La prossima funzione viene richiamata quando il pulsante per registrare viene premuto, per poter salvare il video che verrà registrato l’utente dovrà garantire i permessi di accesso alla memoria del dispositivo:

fun startOrStopRecording(onVideoSaved:(Uri?) ->Unit?, onClassificationDone: (Int?) -> Unit?) {  
 val videoCapture = videoCapture ?: return ;  
  
 if (currentRecording != null) {  
 currentRecording ?.stop()  
 currentRecording = null  
 Toast.makeText(context, "Recording stopped", Toast.*LENGTH\_SHORT*).show()  
 } else {  
 val name = "video\_${System.currentTimeMillis()}.mp4"  
  
 val contentValues = ContentValues().*apply* **{**  
put(MediaStore.MediaColumns.*DISPLAY\_NAME*, name)  
 put(MediaStore.MediaColumns.*MIME\_TYPE*, "video/mp4")  
 if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*Q*) {  
 put(MediaStore.MediaColumns.*RELATIVE\_PATH*, "Movies/MyAppVideos")  
 }  
 **}**  
  
val mediaStoreOutputOptions = MediaStoreOutputOptions.Builder(  
 context.*contentResolver*,  
 MediaStore.Video.Media.*EXTERNAL\_CONTENT\_URI*  
)  
 .setContentValues(contentValues)  
 .build()  
  
 currentRecording = videoCapture.*output*  
.prepareRecording(context as androidx.activity.ComponentActivity, mediaStoreOutputOptions)  
 .start(ContextCompat.getMainExecutor(context)) **{**  
recordEvent **->**  
when(recordEvent) {  
 is VideoRecordEvent.Start -> {  
 Toast.makeText(context, "Recording started", Toast.*LENGTH\_SHORT*).show()  
 onClassificationDone(null);  
 }  
 is VideoRecordEvent.Finalize -> {  
 if (!recordEvent.hasError()) {  
 val uri = recordEvent.*outputResults*.*outputUri*  
onVideoSaved(uri)  
 val result=ExtractFrames(uri);  
  
 val byteBuffer:ByteBuffer=bitmapsToByteBuffer(result,112,112)  
  
 val inputFeature0 = TensorBuffer.createFixedSize(*intArrayOf*(1, 30, 112, 112, 3), DataType.*FLOAT32*)  
 inputFeature0.loadBuffer(byteBuffer)  
  
 val outputs = model.process(inputFeature0)  
 val outputFeature0 = outputs.*outputFeature0AsTensorBuffer*  
  
val modelResult: MutableList<Float> = *mutableListOf*(outputFeature0.getFloatValue(0),outputFeature0.getFloatValue(1),outputFeature0.getFloatValue(2),outputFeature0.getFloatValue(3),outputFeature0.getFloatValue(4))  
  
 Log.i("Result","Result: ${modelResult}")  
  
 onClassificationDone(modelResult.*indexOf*(modelResult.*maxOrNull*())+1);  
  
 } else {  
 Toast.makeText(context, "Error saving video: ${recordEvent.*error*}", Toast.*LENGTH\_SHORT*).show()  
 onClassificationDone(null);  
 }  
 currentRecording = null  
 }  
 }  
 **}**  
}  
}

Questa funzione serve a gestire l’avvio o l’arresto della registrazione video usando la libreria CameraX, se la registrazione è in corso, la ferma; se non è in corso, la avvia.

Come prima cosa controlla se ci sia già una registrazione in corso, in quel caso la ferma.

Altrimenti, viene creato un nome unico per il video registrato e prepara i metadati per salvare il file.

Poi definisce un oggetto MediaStoreOutputOptions per definire dove e come salvare il video.

Viene poi avviata la registrazione, in particolare:

* Usa videoCapture.output.prepareRecording(...) per preparare la registrazione.
* Chiama start(...) per iniziarla.
* Registra recordEvent per reagire agli eventi della registrazione.

Nel caso la registrazione venga iniziata notificato all’utente di ciò.

Nel caso la registrazione venga terminata:

* Se è andato tutto correttamente, viene mostrato all’utente dove il file viene salvato e viene richiamato onVideoSaved() per salvarlo.
* Se c’è stato un errore l’utente viene notificato di ciò.
* Poi si richiama la funzione ExtractFrames per estrarre i frame da dare in imput al modello.
* Si converte il risultato ottenuto nel formato leggibile dal modello.
* Si prepara la grandezza del buffer di input del modello e gli si inseriscono i dati.
* La rete neurale processa un risultato.
* Si prende la classificazione con risultato maggiore e lo si restituisce attraverso una callback al [Controller](#_Controller).

fun ExtractFrames(uri: Uri): List<Bitmap> {  
 val retriever = MediaMetadataRetriever()  
 val frameDiffs = *mutableListOf*<Pair<Bitmap, Double>>() // Pair of resized frame and its motion score  
 val targetWidth = 112  
 val targetHeight = 112  
  
 try {  
 retriever.setDataSource(context, uri)  
  
 val durationMs = retriever.extractMetadata(MediaMetadataRetriever.*METADATA\_KEY\_DURATION*)?.*toLongOrNull*()  
 ?: return *emptyList*()  
  
 val totalFrames = 100  
 val intervalMs = durationMs / totalFrames  
  
 var lastFrame: Bitmap? = null  
  
 for (i in 0 *until* totalFrames) {  
 val timeUs = i \* intervalMs \* 1000  
 val frame = retriever.getFrameAtTime(timeUs, MediaMetadataRetriever.*OPTION\_CLOSEST*) ?: continue  
  
 // Resize the frame to 112x112  
 val resizedFrame = Bitmap.createScaledBitmap(frame, targetWidth, targetHeight, true)  
  
 val diffScore = if (lastFrame != null) {  
 calculateFrameDifference(lastFrame!!, resizedFrame)  
 } else {  
 0.0  
 }  
  
 frameDiffs.add(Pair(resizedFrame, diffScore))  
 lastFrame = resizedFrame  
 }  
  
 } catch (e: Exception) {  
 Log.e("MotionDetect", "Error processing video", e)  
 } finally {  
 retriever.release()  
 }  
  
 return frameDiffs  
 .*sortedByDescending* **{ it**.second **}** // sort by motion score  
 .*take*(30) // take top 30  
 .*map* **{ it**.first **}** // return bitmaps only  
}

Questa è la funzione che estrae i frames dal video, estrae 30 frames con più variazione tra quello ed il suo frame precedente per prendere i frame più significativi e li imposta all’aspect ratio letto dalla neural network.

fun calculateFrameDifference(b1: Bitmap, b2: Bitmap): Double {  
 val scaledB1 = Bitmap.createScaledBitmap(b1, 64, 64, true)  
 val scaledB2 = Bitmap.createScaledBitmap(b2, 64, 64, true)  
  
 var diff = 0L  
  
 for (x in 0 *until* scaledB1.*width*) {  
 for (y in 0 *until* scaledB1.*height*) {  
 val pixel1 = scaledB1.getPixel(x, y)  
 val pixel2 = scaledB2.getPixel(x, y)  
  
 val r1 = (pixel1 shr 16) and 0xFF  
 val g1 = (pixel1 shr 8) and 0xFF  
 val b1c = pixel1 and 0xFF  
  
 val r2 = (pixel2 shr 16) and 0xFF  
 val g2 = (pixel2 shr 8) and 0xFF  
 val b2c = pixel2 and 0xFF  
  
 diff += kotlin.math.*abs*(r1 - r2)  
 diff += kotlin.math.*abs*(g1 - g2)  
 diff += kotlin.math.*abs*(b1c - b2c)  
 }  
 }  
  
 return diff.toDouble() / (scaledB1.*width* \* scaledB1.*height*)  
}

Questa funzione è quella usata per calcolare quanto due frame consecutivi siano diversi, per farlo compara i colori di ciascun pixel e ne accumula la differenza, restituendo una sorta di “distanza” totale.

fun bitmapsToByteBuffer(  
 bitmaps: List<Bitmap>,  
 modelInputWidth: Int,  
 modelInputHeight: Int,  
 isQuantized: Boolean = false // true if your model uses uint8 instead of float32  
): ByteBuffer {  
  
 val inputSize = modelInputWidth \* modelInputHeight \* 3 // Assuming RGB  
 val bytesPerChannel = if (isQuantized) 1 else 4  
 val buffer = ByteBuffer.allocateDirect(bitmaps.size \* inputSize \* bytesPerChannel)  
 buffer.order(ByteOrder.nativeOrder())  
  
 for (bitmap in bitmaps) {  
 val resized = Bitmap.createScaledBitmap(bitmap, modelInputWidth, modelInputHeight, true)  
 for (y in 0 *until* modelInputHeight) {  
 for (x in 0 *until* modelInputWidth) {  
 val pixel = resized.getPixel(x, y)  
  
 val r = Color.red(pixel)  
 val g = Color.green(pixel)  
 val b = Color.blue(pixel)  
  
 if (isQuantized) {  
 // uint8 values  
 buffer.put(r.toByte())  
 buffer.put(g.toByte())  
 buffer.put(b.toByte())  
 } else {  
 // float32 values normalized to [0, 1]  
 buffer.putFloat(r / 255.0f)  
 buffer.putFloat(g / 255.0f)  
 buffer.putFloat(b / 255.0f)  
 }  
 }  
 }  
 }  
  
 buffer.rewind()  
 return buffer  
}

Questa funzione è usata per convertire il risultato della funzione ExtractFrames(), descritta precedentemente, che restituisce una lista di Bitmap, in un ByteBuffer utilizzato dalla rete neurale per leggere l’input.

Nel nostro caso il modello non può leggere i valori quantizzati in quanto lavora con numeri FLOAT32.

fun FlipCamera(){  
 lensFacing = if (lensFacing == CameraSelector.*LENS\_FACING\_BACK*) {  
 CameraSelector.*LENS\_FACING\_FRONT*  
} else {  
 CameraSelector.*LENS\_FACING\_BACK*  
}  
  
 startCamera()  
  
}

Questa funzione viene richiamata dal Controller quando l’utente clicca l’icona per cambiare la telecamera utilizzata, da interna a esterna o viceversa.

#### VideoManager

È la classe per la gestione del Video Player dato l’uri del video da riprodurre.

class VideoManager (private val context: Context, private val videoView: VideoView) {  
  
 fun loadAndPlayVideo(uri: Uri) {  
 val mediaController = MediaController(context)  
 mediaController.setAnchorView(videoView)  
 videoView.setMediaController(mediaController)  
  
 videoView.setVideoURI(uri)  
 videoView.setOnPreparedListener **{** mp: MediaPlayer **->**  
  
val videoWidth = mp.*videoWidth*  
val videoHeight = mp.*videoHeight*  
  
if (videoWidth == 0 || videoHeight == 0) {  
 videoView.start()  
 return@setOnPreparedListener  
 }  
  
 val parent = videoView.*parent* as View  
 val parentWidth = parent.*width*  
  
val aspectRatio = videoHeight.toFloat() / videoWidth.toFloat()  
 var calculatedHeight = (parentWidth \* aspectRatio).toInt()  
  
 // Calculate 40% of screen height in pixels  
 val displayMetrics = context.*resources*.*displayMetrics*  
val maxHeightPx = (displayMetrics.heightPixels \* 0.4).toInt()  
  
 // Cap height to 40% of screen height  
 if (calculatedHeight > maxHeightPx) {  
 calculatedHeight = maxHeightPx  
 }  
  
 val layoutParams = videoView.*layoutParams*  
layoutParams.width = parentWidth  
 layoutParams.height = calculatedHeight  
 videoView.*layoutParams* = layoutParams  
  
 videoView.start()  
 **}**  
}

MediaController aggiunge i controlli tipici dei Video Player sopra il video (pausa, play, ...).

Viene poi impostato l’uri del video e preparato un listener per quando il MediaPlayer sarà pronto.

Una volta pronto e controllato che il video abbia dimensioni valide, le dimensioni di VideoView vengono ridimensionate per impostare il corretto aspect ratio al video mettendo come limite di altezza un quarto dello schermo.

Infine, il video viene avviato.

Le seguenti funzioni vengono richiamate dal Controller per eseguire le azioni riconosciute dalla Camera:

fun togglePlayPause() {  
 if (videoView.*isPlaying*) {  
 videoView.pause()  
 } else {  
 videoView.start()  
 }  
}  
  
fun seekForward(ms: Int = 10\_000) {  
 val newTime = videoView.*currentPosition* + ms  
 videoView.seekTo(newTime.*coerceAtMost*(videoView.*duration*))  
}  
  
fun seekBackward(ms: Int = 10\_000) {  
 val newTime = videoView.*currentPosition* - ms  
 videoView.seekTo(newTime.*coerceAtLeast*(0))  
}

La prima mette il video in pausa o riprende la riproduzione del video a seconda dello stato corrente del video.

Le altre due funzioni aggiungono o tolgono un certo numero di millisecondi (di default 10 secondi) al video in riproduzione rispettivamente.

#### VolumeManager

È la classe per la gestione del volume del dispositivo.

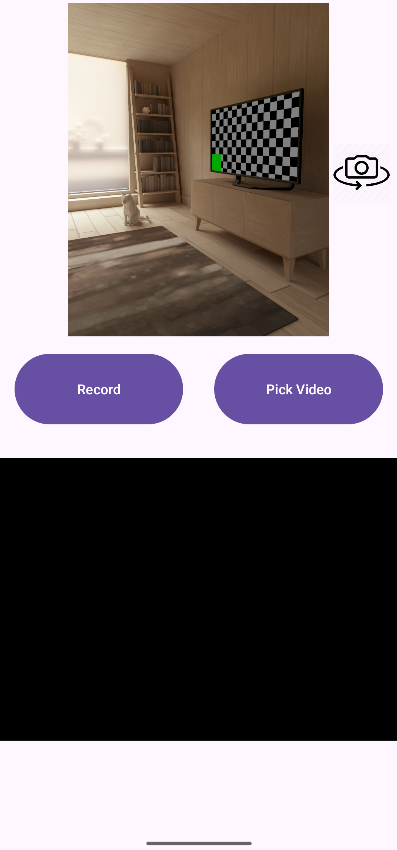
E contiene le seguenti funzioni che verranno richiamate dal Controller se viene riconosciuta la corrispondente azione:

class VolumeManager(private val context: Context) {  
  
 private val audioManager = context.getSystemService(Context.*AUDIO\_SERVICE*) as AudioManager  
  
 fun raiseVolume() {  
 audioManager.adjustStreamVolume(  
 AudioManager.*STREAM\_MUSIC*,  
 AudioManager.*ADJUST\_RAISE*,  
 AudioManager.*FLAG\_SHOW\_UI*  
)  
 }  
  
 fun lowerVolume() {  
 audioManager.adjustStreamVolume(  
 AudioManager.*STREAM\_MUSIC*,  
 AudioManager.*ADJUST\_LOWER*,  
 AudioManager.*FLAG\_SHOW\_UI*  
)  
 }  
}

Queste funzioni rispettivamente aumentano o diminuiscono il volume del dispositivo di una unità mostrando l’ui del volume del dispositivo a schermo attraverso l’utilizzo del servizio di gestione dell’audio di sistema.

### View

Questo rappresenta il modo in cui l’applicazione si presenta all’utente ed è visualizzato in questo modo:



Definito nel seguente file xml:

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
  
 <LinearLayout  
 android:id="@+id/linearLayout"  
 android:layout\_width="411dp"  
 android:layout\_height="127dp"  
 android:layout\_marginTop="340dp"  
 android:gravity="center\_horizontal"  
 android:orientation="horizontal"  
 android:padding="16dp"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="@+id/previewView">  
  
 <Button  
 android:id="@+id/recordButton"  
 android:layout\_width="0dp"  
 android:layout\_height="80dp"  
 android:layout\_marginEnd="16dp"  
 android:layout\_weight="1"  
 android:onClick="startOrStopRecording"  
 android:text="Record" />  
  
 <Button  
 android:id="@+id/pickVideoButton"  
 android:layout\_width="0dp"  
 android:layout\_height="80dp"  
 android:layout\_marginStart="16dp"  
 android:layout\_weight="1"  
 android:onClick="onPickVideo"  
 android:text="Pick Video" />  
  
 </LinearLayout>  
  
 <androidx.camera.view.PreviewView  
 android:id="@+id/previewView"  
 android:layout\_width="268dp"  
 android:layout\_height="342dp"  
 android:layout\_margin="16dp"  
 android:background="#000000"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.496"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_bias="0.0" />  
  
 <ImageButton  
 android:id="@+id/flipCamera"  
 android:layout\_width="60dp"  
 android:layout\_height="60dp"  
 android:layout\_marginStart="343dp"  
 android:layout\_marginTop="160dp"  
 android:background="@android:color/transparent"  
 android:scaleType="fitCenter"  
 android:src="@drawable/flip\_camera\_icon"  
 android:onClick="onFlipCamera"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <FrameLayout  
 android:id="@+id/videoContainer"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:paddingBottom="16dp"  
 app:layout\_constraintTop\_toBottomOf="@+id/linearLayout"> <!-- Add padding here -->  
  
 <VideoView  
 android:id="@+id/videoView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="290dp"  
 android:layout\_gravity="center" />  
 </FrameLayout>  
  
  
</androidx.constraintlayout.widget.ConstraintLayout>

## Risultati e considerazioni finali

L’applicazione è in grado di sfruttare la rete neurale per campiere le varie azioni possibili.

Nonostante questo, non è molto ottimizzata per quanto riguarda la parte di elaborazione di frames, probabilmente sarebbe stato più efficiente fare eseguire questa manipolazione in Python.

Una possibilità alternativa forse più efficiente nello scopo della nostra applicazione potrebbe essere l’utilizzo di ImageAnalysis al posto di CameraX per un'elaborazione in tempo reale del video invece che al termine della registrazione.

# Modello Neural Network

## IL DATASET

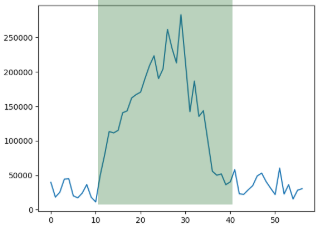
Il dataset (preso da Keggle) su cui è stato trainato un modello è composto di circa 600 video strutturati come 30 immagini di dimensioni quadrate e 100 video per il test set. Il vantaggio sta nella quantità di dati su cui si può trainare il modello. Dall’altra parte non c’è troppa varietà per quanto riguarda persone o luoghi in cui sono state scattate le immagini.

## GLI INPUT AL MODELLO

Poiché nella nostra applicazione di base abbiamo un video come dato, una funzione fondamentale per il riconoscimento della gesture è stata “load\_most\_moving\_frames\_center\_crop”. Il suo scopo è selezionare automaticamente dai video registrati un set di 30 frame che catturano la parte del video con più movimento. Questo consente di isolare le parti del video più rilevanti per l'analisi del gesto, eliminando porzioni più statiche. Inoltre viene fatto un crop su ogni frame tenendo solamente il quadrato centrale e scartando le fasce sopra e sotto.

Tutto ciò è stato fatto sfruttando la libreria OpenCV con lo scopo di estrarre i frame dai video nel modo più simile possibile al dataset su cui il modello è stato trainato.

<Immagine: grafico movimento->video->30 frame->crop centrale>



## IL MODELLO

Il modello è una rete neurale convoluzionale 3D pensata per riconoscere gesti a partire da video.

Ogni input è una sequenza di 30 frame RGB, e l'obiettivo è classificare il gesto presente nel video, sfruttando sia le informazioni spaziali (nei singoli frame) che temporali (nel movimento tra i frame).

Nel modello Vengono usate le ResNet ovvero una particolare architettura per reti neurali utile nel caso di CNN profonde.

3D CNN e (2+1)D CNN

Per creare il un modello che potesse analizzare video, quindi una sequenza di frame, è stata usata una rete convoluzionale. Di solito una CNN viene usata per classificare immagini singole e quindi viene fatta una interpolazione 2D. Per i video è diverso poiché presentano un concetto in più: sequenza temporale di frame. Per questo serve un kernel 3D che è capace non solo di interpretare frame singoli ma anche una sequenza di essi. Infatti, a differenza di un filtro 2D, il filtro 3D è capace di muoversi anche nella dimensione temporale.

2D altezza larghezza -> classifica immagini

3D altezza larghezza tempo -> classifica video

Perciò se immaginiamo di avere un set di 30 frame uno affianco al altro creerebbero un volume (array) di dimensioni: (30, altezza, larghezza). Perciò un filtro 3D funzionerebbe esattamente come uno 2D.

Le reti convoluzionali tridimensionali (3D CNN) estendono il concetto delle CNN 2D alle sequenze video, aggiungendo la dimensione temporale al processo di convoluzione. Mentre nelle CNN 2D i filtri operano su altezza e larghezza di un’immagine per estrarne caratteristiche spaziali, le CNN 3D applicano filtri volumetrici di forma (T × H × W), dove T è la profondità temporale, per analizzare blocchi di più fotogrammi consecutivi. Questo permette al modello non solo di riconoscere strutture spaziali all’interno di ogni singolo frame, ma anche di catturare dinamiche e movimenti nel tempo, fondamentali per comprendere il contenuto di un video. L’input viene trattato come un volume tridimensionale, ad esempio con forma (30, altezza, larghezza), dove 30 rappresenta il numero di frame. Un kernel 3D scorre su questo volume e, in ogni posizione, aggrega le informazioni di un piccolo cubo, generando una nuova rappresentazione volumetrica. Sebbene le CNN 3D siano molto efficaci nell’analisi spaziotemporale, esse richiedono un numero elevato di parametri: ad esempio, un filtro 3D di dimensione (3 × 3 × 3) con C canali in ingresso e uscita necessita di 27 · C² pesi. Questo le rende computazionalmente costose.

Per ovviare a questo problema, sono state introdotte le reti convoluzionali (2+1)D, che scompongono la convoluzione 3D in due operazioni separate: una convoluzione spaziale 2D applicata ai singoli frame (1 × 3 × 3), seguita da una convoluzione temporale 1D (3 × 1 × 1) che elabora la sequenza dei frame nel tempo. Questa fattorizzazione riduce significativamente il numero di parametri (ad esempio 12 · C² invece di 27 · C² per un filtro equivalente) e migliora l’efficienza dell’apprendimento, pur mantenendo la capacità di estrarre informazioni sia spaziali che temporali.

## RESNET

Le ResNet sono architetture di NN progettate per risolvere un grosso problema che emerge quando le reti diventano molto profonde: il degrado delle prestazioni.

Questo è dovuto a:

* Vanishing/exploding gradients
* Difficoltà nel propagare informazioni utili all’indietro (backpropagation)

Una ResNet permette di trasportare l’input lungo tutta la rete, in questo modo si evita che durante la rete venga “scordato” l’input. Infatti, funzioni come la ReLU rischiano di cancellare informazioni (ad esempio valori del neurone minore di 0) utili mentre i residual block del modello risolvono anche questo problema.

Le residual connections risolvono questi problemi creando una scorciatoia che trasmette direttamente l’input di un blocco alla sua uscita, facendo sì che il blocco debba imparare solo la differenza (il residuo) rispetto a ciò che riceve in ingresso.

Immagine che contiene Carattere, schermata, calligrafia, simbolo

Il contenuto generato dall'IA potrebbe non essere corretto., Picture Immagine che contiene Carattere, testo, schermata, Elementi grafici

Il contenuto generato dall'IA potrebbe non essere corretto., Picture

La somma x + f(x) serve a non perdere l’informazione iniziale di x. In pratica ad ogni passo modifichiamo leggermente x con le nuove informazioni apprese dalla convoluzione.

Nel nostro caso la f(x) è proprio la convoluzione 3D spiegata di sopra.

Di solito una rete prende una immagine con 3 canali RGB e ne estrae n (ad esempio 16) feature: bordi verticali, orizzontali, curve… quindi si fa del pooling e l’immagine (le 16 feature) viene ridotta di dimensioni (questa riduzione di dimensioni simula il catturare feature sempre più estese e grandi dell’immagine, noi riduciamo l’immagine ma in realtà è come se stessimo rendendo i filtri più grandi: Aumenta il campo visivo e ogni unità nel layer successivo guarda un’area più ampia dell’immagine originale) e così via. Si passa dai pixel → a bordi → a texture → a parti del viso → a una testa. All’ultimo passo si prendono tutte le feature estratte (teste, zampe…) e con “Flatten + Dense + Softmax”(capire) si decide se è un cane o un gatto. Con l’aumentare dei layer Il gradiente si disperde → l’input originale non viene più "sentito" e La rete diventa difficile da ottimizzare. Invece le reti con residual block funzionano portandosi la rappresentazione x dell’immagine originale lungo tutta la rete e i vari layer potenziano le aree dove trovano certe feature come occhi o orecchie ma mantenendo comunque x che viene ritoccata ad ogni layer. Evitando di alterare completamente l’immagine iniziale come farebbero le classiche CNN ma modificando l’input pian piano.

Un fattore importante è appunto rimanere costanti con le dimensioni dell’input e output di ogni residual block, insomma x e f(x) devono avere le stesse dimensioni di altezza e larghezza. Altrettanto importante è ridurre la dimensione dei canali e quindi dopo ogni residual block i canali vengono riscalati (tipicamente di un fattore 2) e ridotti di dimensioni, dopodiché si può riapplicare un residual block (che comprende la convoluzione 3D). Questo è il principio che sta alla base del nostro modello.

## ARCHITETTURA DEL MODELLO

È un modello di deep learning che usa Keras ed è composto da diversi livelli grazie a Functional API: classi Python che ereditano da keras.layers.Layer che poi vanno a specializzarsi; quindi, vengono assemblate e riutilizzate per comporre il modello finale. Qui un punto fondamentale, che poi verrà approfondito, è il fatto di poter costruire un grafo anzi che il classico modello lineare di keras.sequential. Infatti, c’è la possibilità di mettere i layer in sequenza ma anche in parallelo.

## PIPELINE MODELLO

Immagine che contiene testo, Carattere, ricevuta, documento

Il contenuto generato dall'IA potrebbe non essere corretto., Picture

In generale la rete neurale sfrutta la combinazione di convoluzioni 2+1D e residual block per apprendere in modo efficiente movimenti complessi, applicando più Resize il che consente di catturare informazioni a diverse scale spaziali.

Una pipeline di input al modello serve innanzitutto per inserire l’immagine e passarla attarverso i nodi. Un punto fondamentale è un layer di pre-processing che usa data augmentation per migliorare la generalizzazione del modello. Questo layer applica ad ogni video una variazione di luminosità costante su tutti i frame simulando condizioni di luce diverse. Questa strategia rende il modello meno prono all’overfitting rispetto alle condizioni di luminosità viste durante l’addestramento.

Immagine che contiene persona, Viso umano, interno, Selfie

Il contenuto generato dall'IA potrebbe non essere corretto., PictureImmagine che contiene persona, Viso umano, occhiale, Forte

Il contenuto generato dall'IA potrebbe non essere corretto., PictureImmagine che contiene persona, Viso umano, vestiti, Forte

Il contenuto generato dall'IA potrebbe non essere corretto., Picture

Di seguito un singolo layer che applica una convoluzione 2+1D.

Quindi i dati vengono normalizzati e passati ad una ReLu.

Dopo questa fase iniziale ci sono una serie di blocchi con lo scopo di ridurre man mano la risoluzione dei video e nel frattempo inserire blocchi di ResNet come descritto di sopra.

Dopodiché viene fatto un pooling sul tensore finale che aggrega i dati lungo tutte e 3 le dimensioni e ne fa una media. Ogni feature è rappresentato da un unico valore. Infine, c’è un Dense layer semplicemente per la classificazione del gesto.

## RISULTATI

Il modello si comporta bene sia durante l’addestramento che nella fase di test. È abbastanza generalizzato e non ci sono segni di overfitting, riesce a predirre le classi in modo eguale senza prediligenre una e raggiunge una una accuracy del 90%.Immagine che contiene schermata, testo, quadrato, Rettangolo

Il contenuto generato dall'IA potrebbe non essere corretto., PictureImmagine che contiene testo, diagramma, schermata, linea

Il contenuto generato dall'IA potrebbe non essere corretto., Picture

Detto ciò, è importante fare una considerazione sul dataset utilizzato: i dati con cui il modello è stato addestrato contengono un gruppo omogeneo di persone e questo limita la capacità del modello di generalizzare quando viene testato su soggetti con caratteristiche differenti. Si può dire che è un overfitting indotto dal dataset. Dai risultati la CNN non sembra essere il problema ed è plausibile aspettarsi risultati migliori con una ampliazione del dataset introducendo appunto diversità nel set di training.