



MEiM

MASTER IN ENTREPRENEURSHIP
INNOVATION MANAGEMENT
IN COLLABORATION WITH **MIT SLOAN**

IN COLLABORATION WITH
MIT MANAGEMENT
SLOAN SCHOOL

Introduction to Machine Learning programming on Apple devices using CoreML kit.

Apple Foundation Program



UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARthenoPE



Foundation
Program



MEIM

MASTER IN ENTREPRENEURSHIP
INNOVATION MANAGEMENT
IN COLLABORATION WITH **MIT SLOAN**

IN COLLABORATION WITH
MIT MANAGEMENT
SLOAN SCHOOL



Emanuel Di Nardo, PhD

- Many courses (ScuolaSIS - Parthenope)
- Teacher at Apple Foundation Program (Parthenope)
- Contacts: emanuel.dinardo@uniparthenope.it



MEiM

MASTER IN ENTREPRENEURSHIP
INNOVATION MANAGEMENT
IN COLLABORATION WITH MIT SLOAN

IN COLLABORATION WITH
MIT MANAGEMENT
SLOAN SCHOOL

Agenda

- Application Scope
- How to build a dataset
- CreateML in details
- Integrate AI in devices applications



UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE



Foundation
Program

Application Scope

What's our aim?

- Learn how to collect data
 - Images, videos, texts, ...
 - How many data?
 - Diversification
 - Augmentation
- Learn how to train and evaluate a model

What's our aim?

- Learn how to train and evaluate a model
 - How many iteration?
 - There is a specific algorithm?
 - Learning curves...what?
 - Results preview

What's our aim?

- Integrate the model in an iOS application
 - Which steps are required before the model can be used in a real-world application?
 -

Hand pose estimation and classification

- Our aim is to estimate the hand pose
- The estimation make us able to perform some tasks
- We want to send commands to an application with our hands



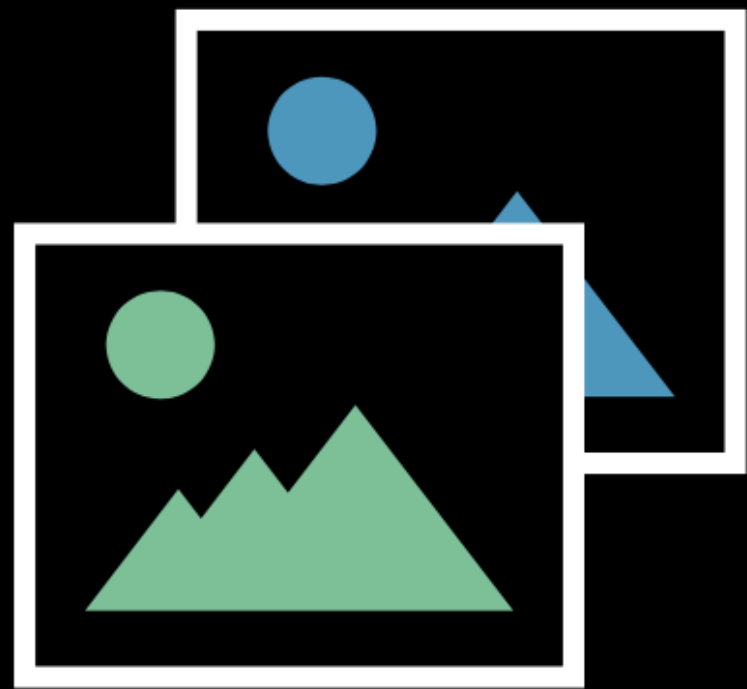
UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE



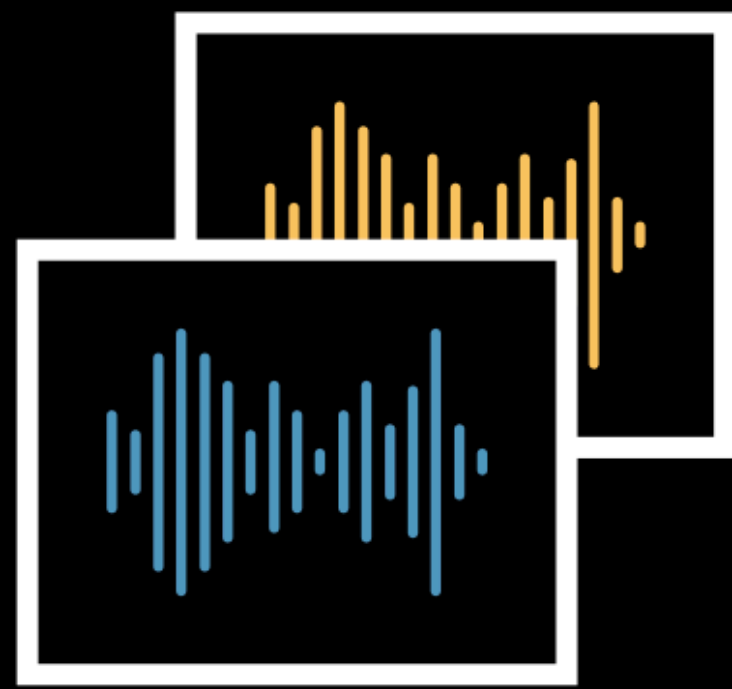
Foundation
Program

Data collection

Which data can I process in my app?



Image



Sound



Activity



Text



Tabular

We need classes!

- Which classes are required in our application?
 - Class 1 Finger
 - Class 2 Fingers
 - Class Thumb Up

Class 1 Finger



Class 2 Fingers



Class Thumb Up



Is it enough?

Is it enough?

NO!

Another class

- We also need background
- It is important to avoid distractions and reduce misclassification
- It is composed
 - Random hand position
 - Transition hand position

Example

Random hand positions



Example

Transition hand positions (1 Finger)



Example

Transition hand positions (1 Finger)



Example

Transition hand positions (1 Finger)



Example

Transition hand positions (2 Fingers)



Example

Transition hand positions (2 Fingers)



Example

Transition hand positions (2 Fingers)



Example

Transition hand positions (Thumb Up)



Example

Transition hand positions (Thumb Up)



Example

Transition hand positions (Thumb Up)



How many data?

- Deep learning is data hungry
- Many data as possible
- Thankfully Apple uses transfer learning
 - It reduces the number of required samples

How many data?

- We try with using about 25/50 samples per class
- To have the same number of data is very important
 - Balanced data is a foundation of machine learning

Diversification

- Try to have data that are different from multiple point of view
 - Geometrically:
 - Scale
 - Position
 - Rotation

Diversification

- Try to have data that are different from multiple point of view
 - Subject:
 - Multiple people
 - Skin color
 - Light condition
 - Poses

Is is enough?

- Unfortunately not...
- It is the best to augment data
 - It automatically applies many random transformations on the input each at each iteration
- When a neural network sees the same image with a little difference, it appears like a totally new and different image!

Some other tricks?

- A lot, but the most important and famous is to shuffle data



- We simply swiped the **second** image with the **first**
- It is like a totally new set of images to process!

Challenge (1h)

- Splits in groups and with the help of your device and using the image search features of web search engine try to collect as many data as possible for each class (*each group can collect a different class to speed up the process*)

- Tips:

- Images should have the same size and orientation
(we can be flexible for now)

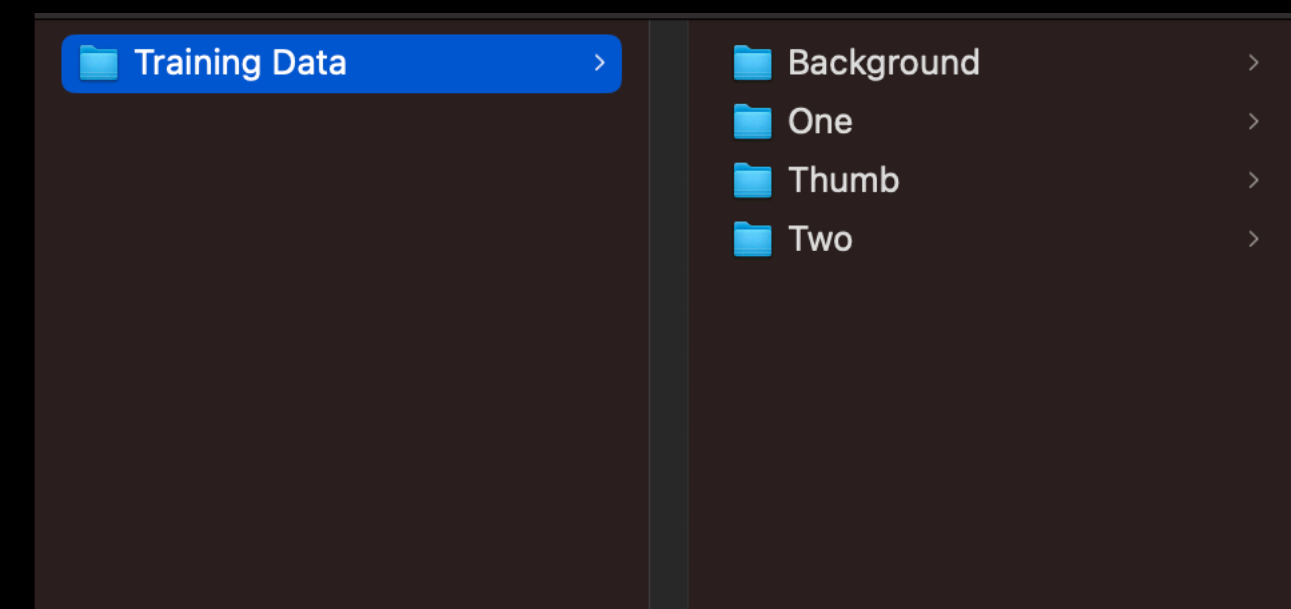


- Images should have the same number of colors
(avoid to mix grayscale and color images)



Split data

- Collect data and split the images by class
- Create a *Training Data* folder
 - Inside create a folder for each class
 - One
 - Two
 - Thumb
 - Background





UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE



Foundation
Program

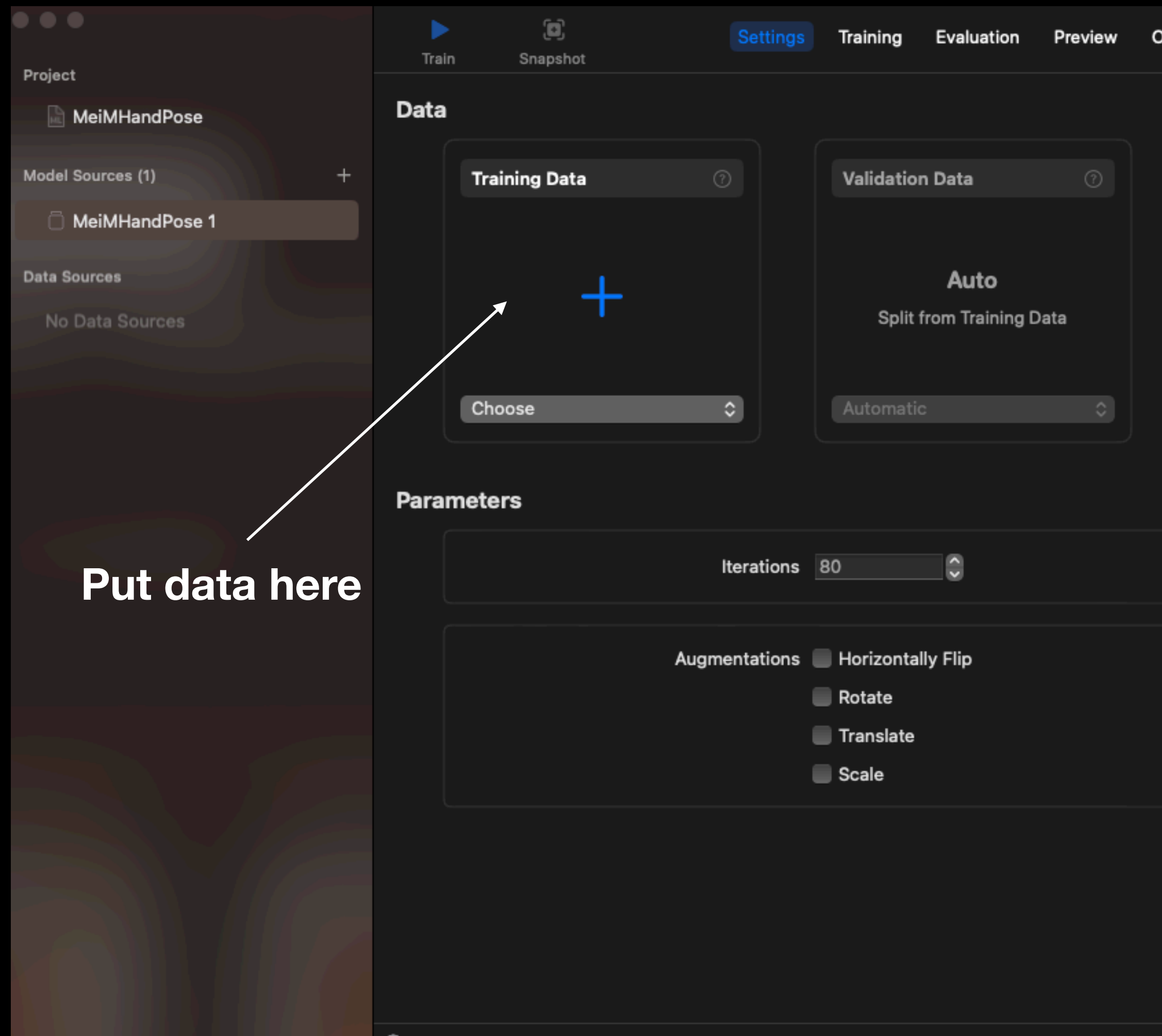
Train the model

CreateML

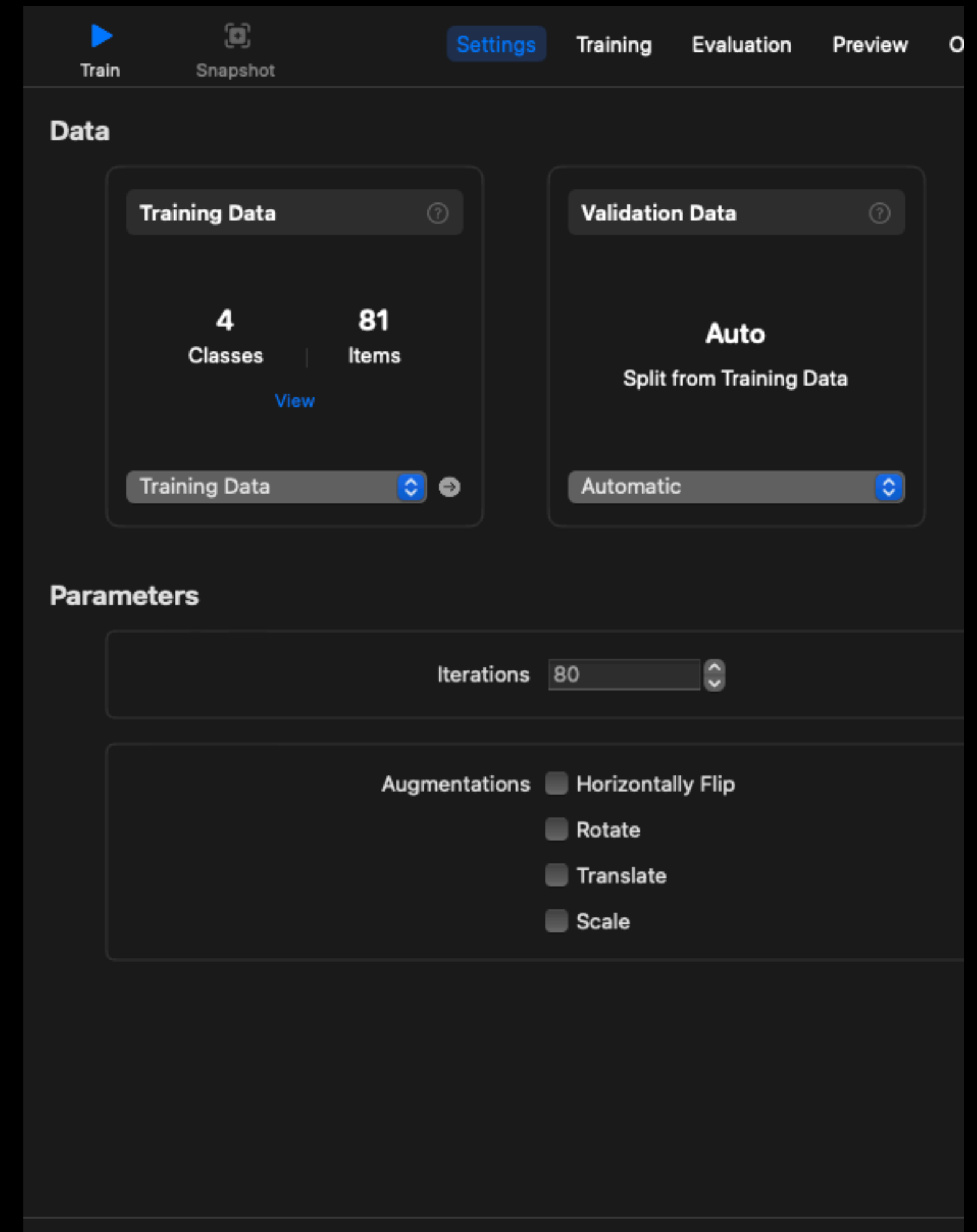
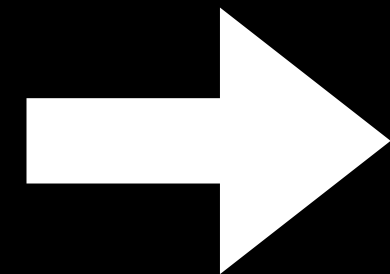
- Open CreateML
 - New Project
- Select Hand Pose Classification
- Choose a name
ex. MeiMHandPose



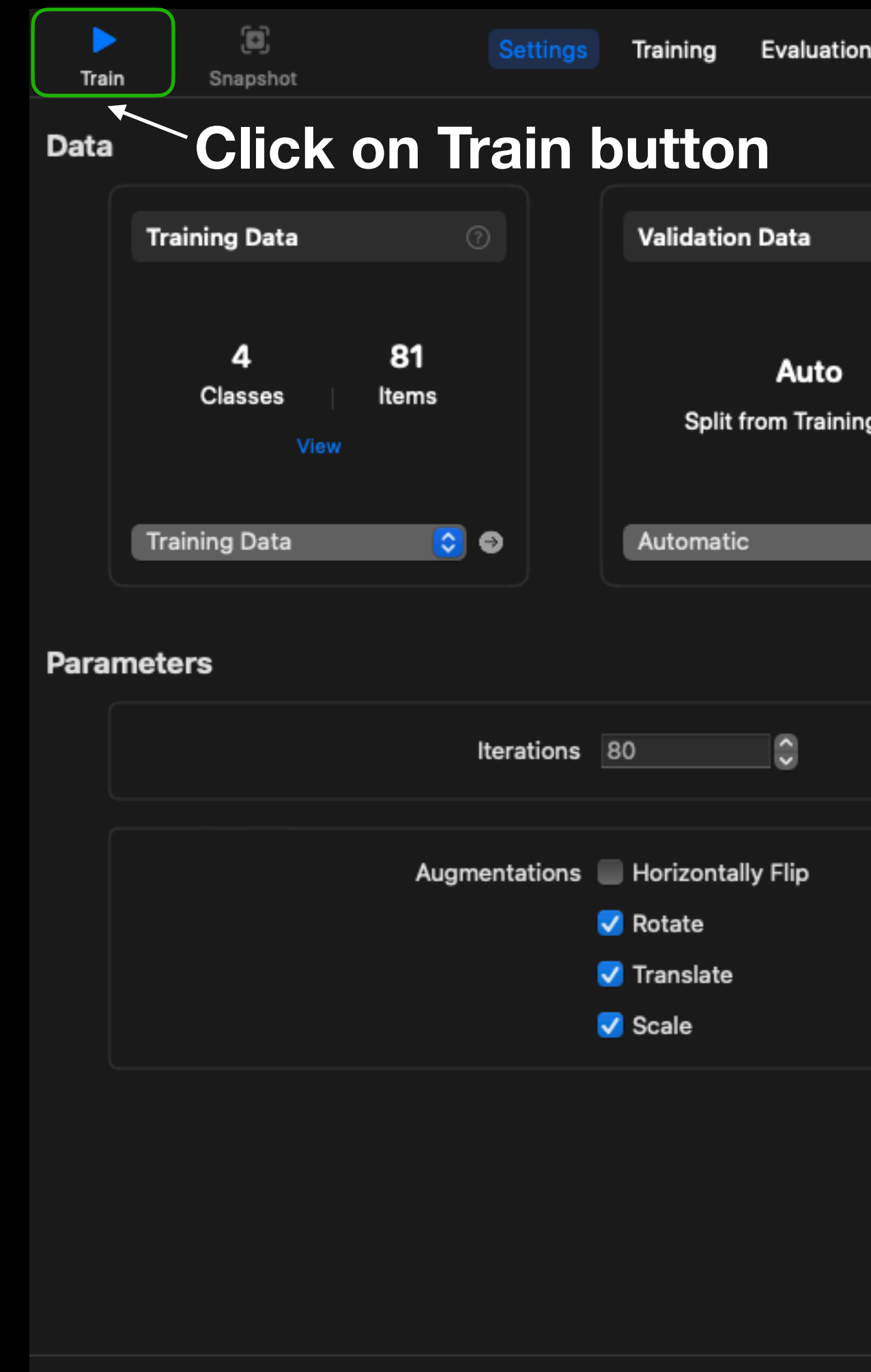
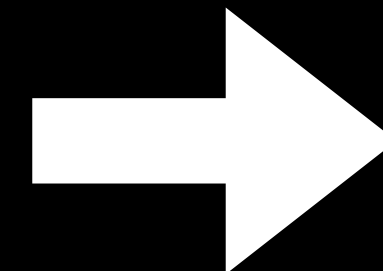
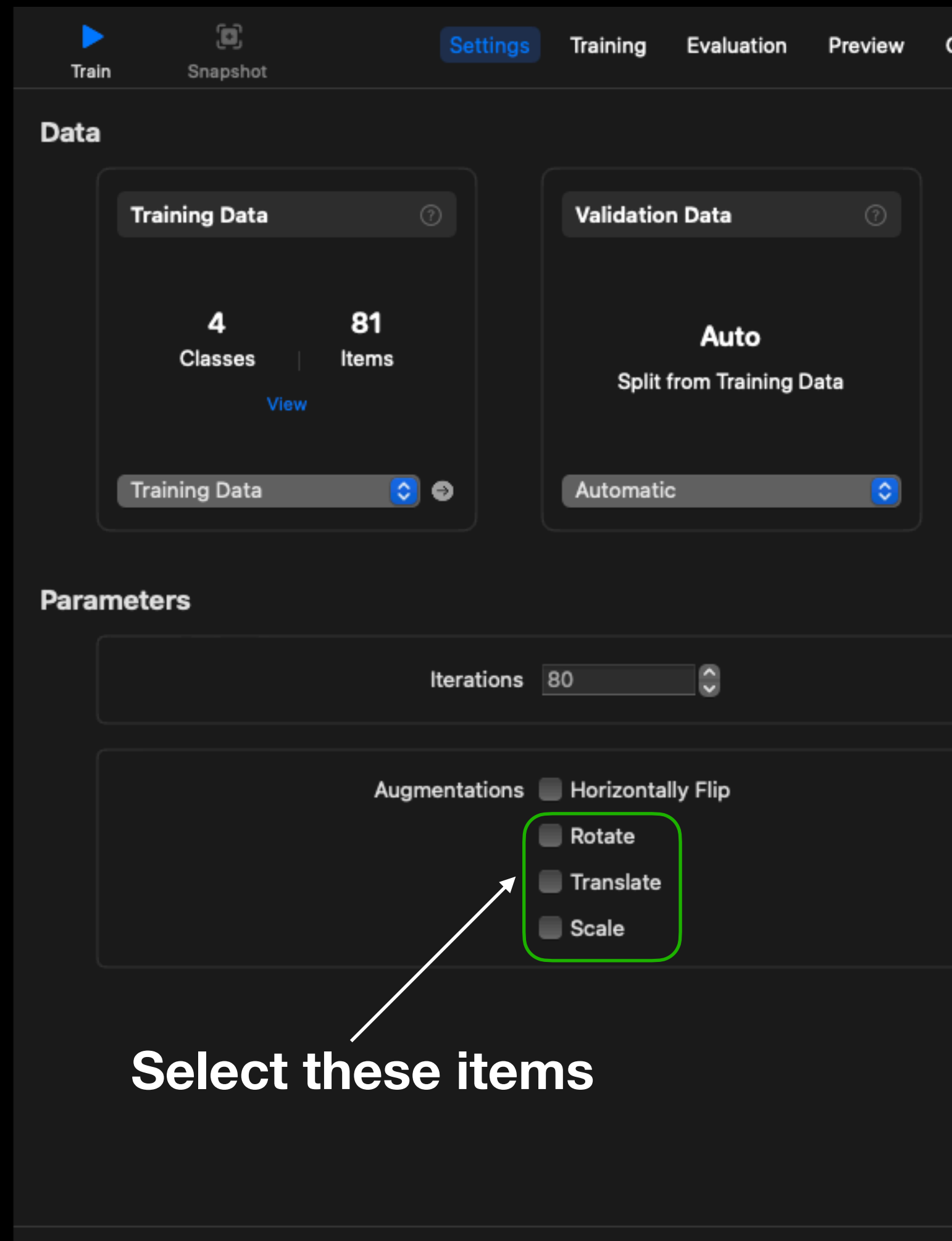
CreateML - Dataset



Put data here



CreateML - Augmentation



CreateML - Train

The screenshot displays the 'Train' interface in CreateML. At the top, there are tabs for 'Settings', 'Training', and 'Evaluation'. The 'Train' button is highlighted with a green box and a white arrow pointing to it, with the text 'Click on Train button' next to it. Below the 'Train' button is a 'Data' section containing two panels: 'Training Data' and 'Validation Data'. The 'Training Data' panel shows '4 Classes' and '81 Items' with a 'View' link. The 'Validation Data' panel shows 'Auto' and 'Split from Training Data'. Below the 'Data' section is a 'Parameters' section with a slider for 'Iterations' set to 80 and a list of 'Augmentations' including 'Horizontally Flip' (unchecked), 'Rotate' (checked), 'Translate' (checked), and 'Scale' (checked).

Train Snapshot Settings Training Evaluation

Data Click on Train button

Training Data ?

4 Classes | 81 Items
View

Training Data

Validation Data

Auto
Split from Training Data

Automatic

Parameters

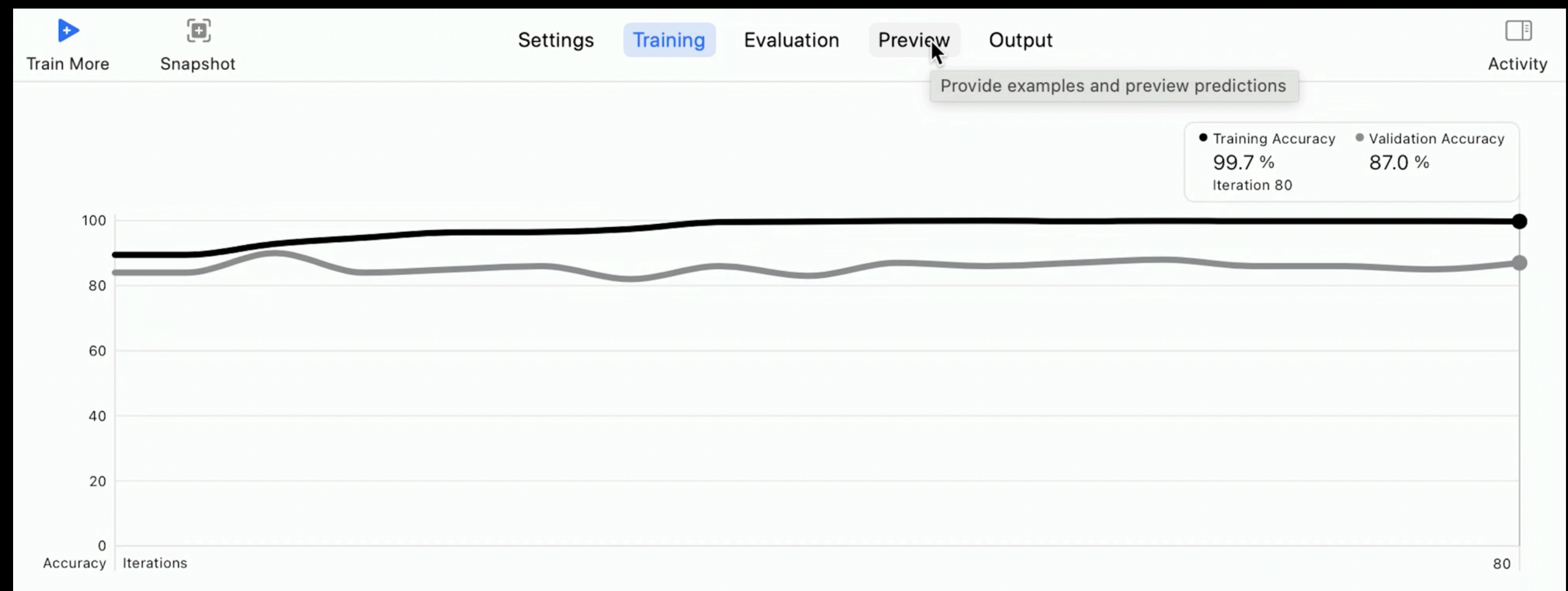
Iterations 80

Augmentations

- Horizontally Flip
- Rotate
- Translate
- Scale

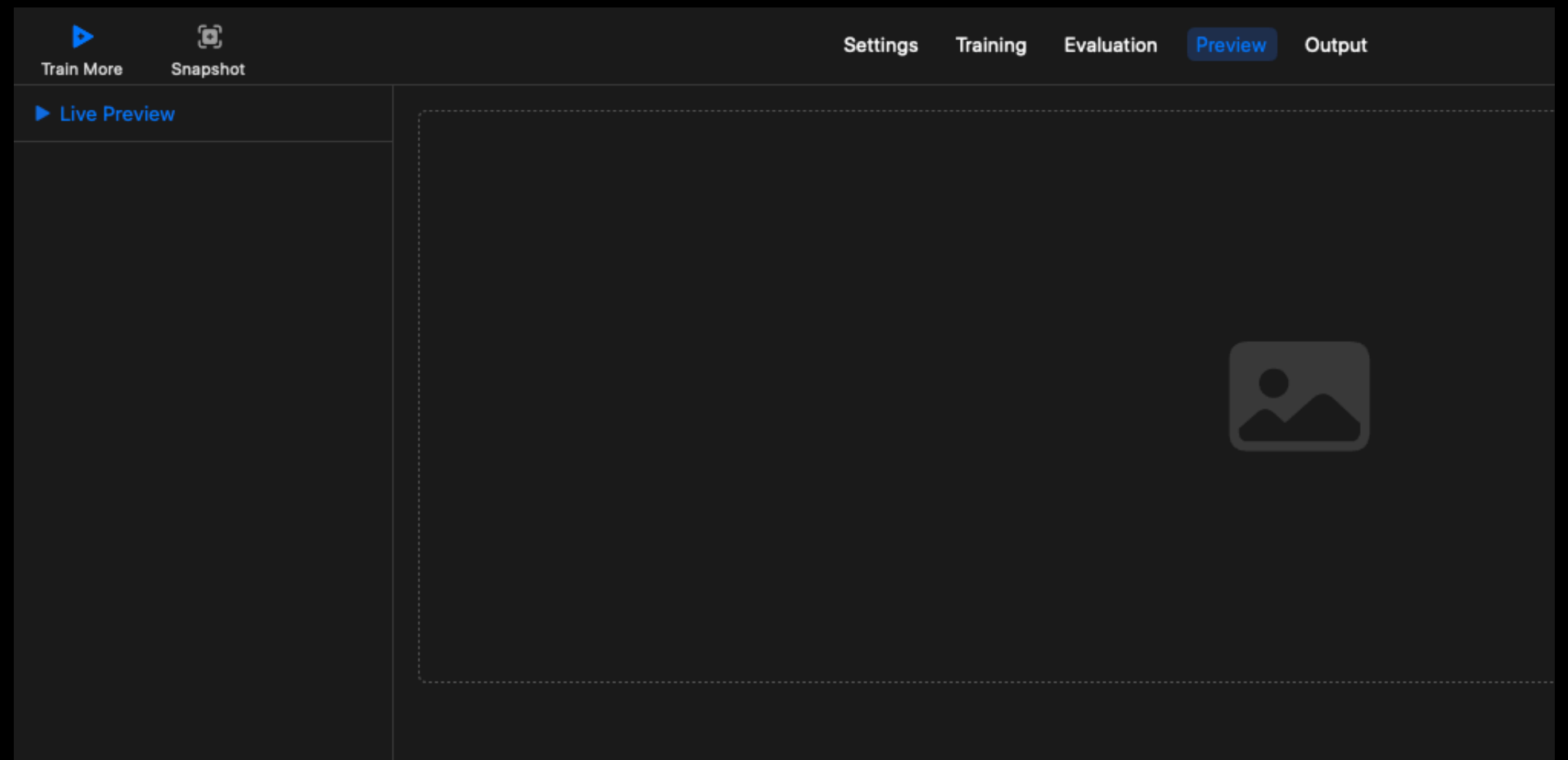
CreateML - Learning curves

- Curves tell us what happened
 - In this case we just have Accuracy (usually it is better to check the *loss* function)
 - Training accuracy represents how good the model have learned from training images
 - Validation accuracy tells us how the model is able to understand images that it never saw before (this concept is called *generalization*)



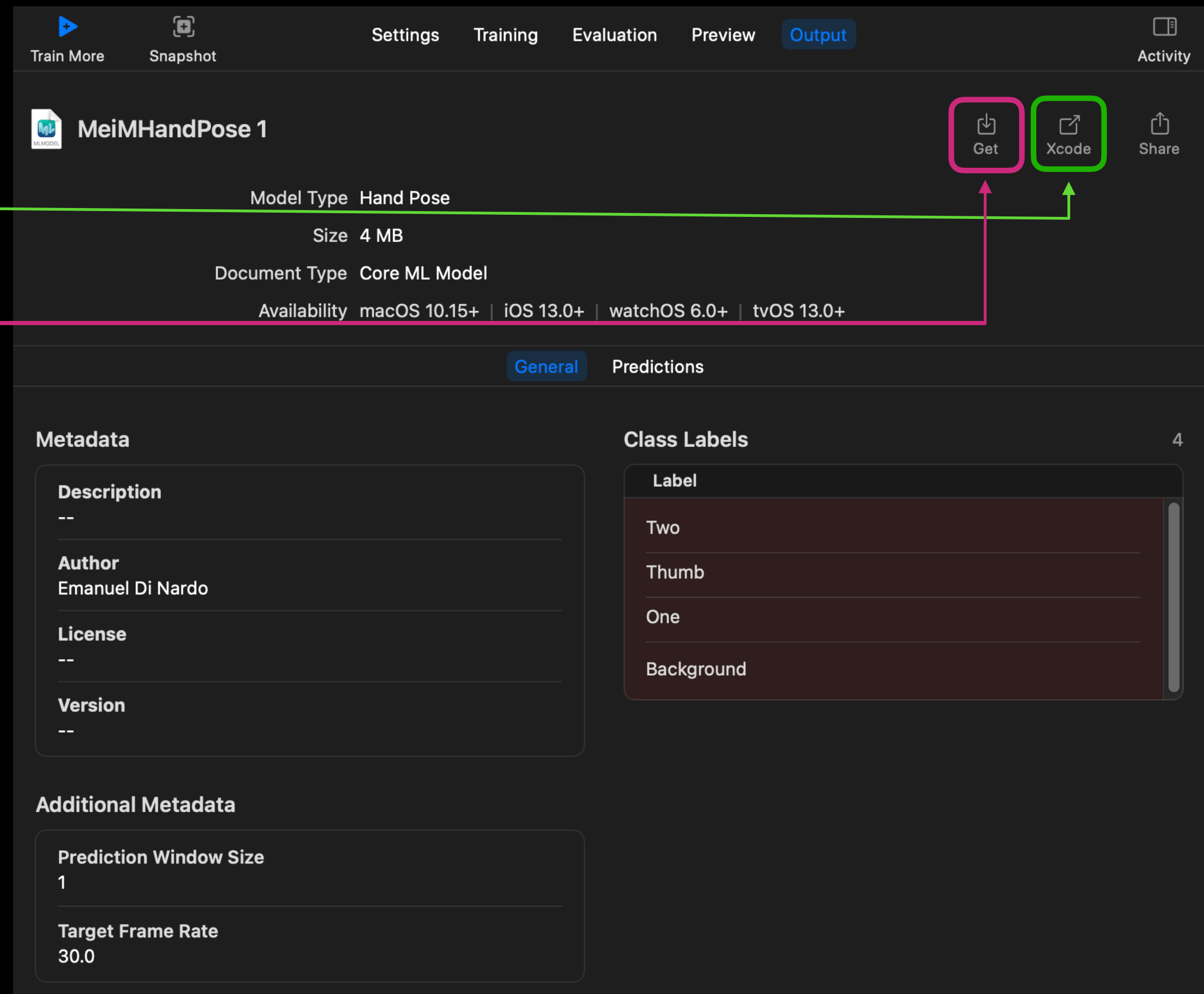
CreateML - Preview

- Preview helps us to test the machine learning model with new data
- We can also click on “Live Preview” and use FaceTime HD Camera to process the input video stream
- It prints out the results of each frame with the relative score of each action



CreateML - Output

- Click on Output menu
- Try **Xcode** button
- To export the model use **Get** button



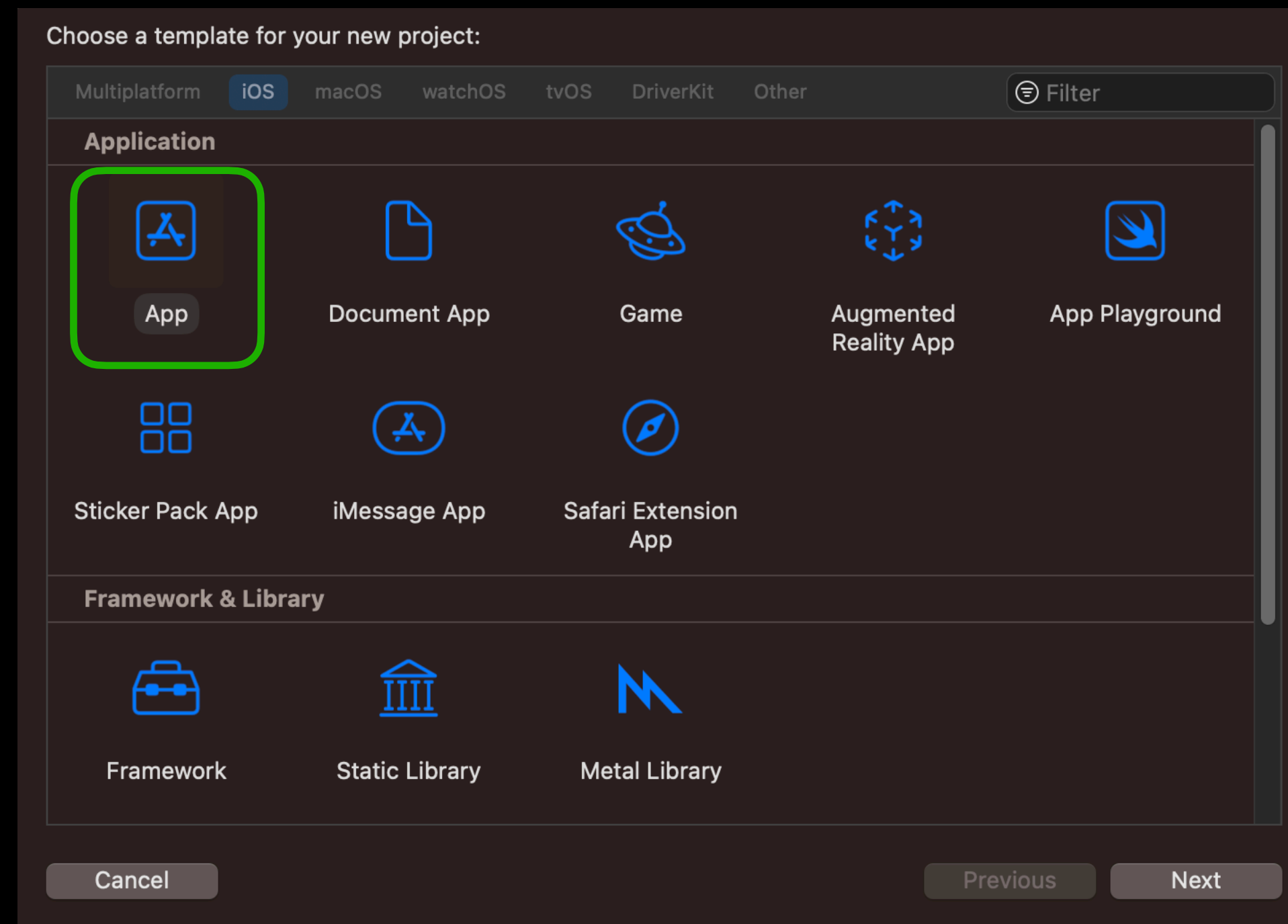
Xcode

- Xcode is the tools that is used to develop application for any Apple devices
- It is installed on your foundation kit
- Open it and create a new project



Xcode

- Select “App”



Xcode

- Fill all inputs like in the image
- Press Next two times

Choose options for your new project:

Product Name:

Team:

Organization Identifier:

Bundle Identifier:

Interface:

Language:

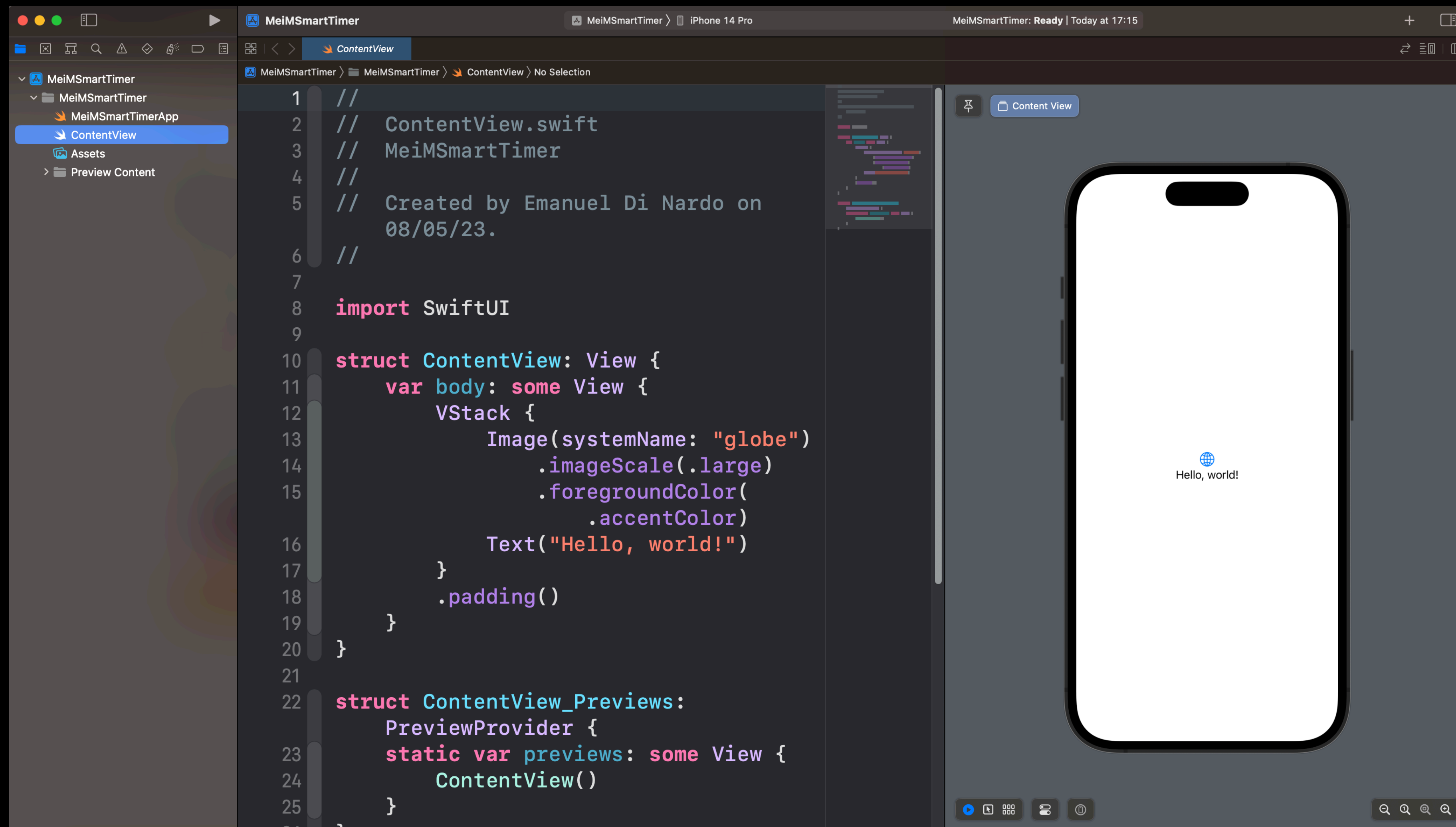
Use Core Data

Host in CloudKit

Include Tests

Xcode

Your project is ready!



Setup the Augmented Reality Camera

- Create a new Swift file (right click on menu list) and call it ARViewContainer
 - It will used to show the camera and use the machine learning model
- Remove “Foundation” and import:
 - SwiftUI (the base UI library)
 - RealityKit
 - ARKit (both used for AR)
 - Vision (it is needed to recognize hands)

ARViewContainer

- We get an error, but it is normal

```
import SwiftUI
import RealityKit
import ARKit
import Vision

struct ARViewContainer: UIViewRepresentable {

    func makeUIView(context: Context) -> ARView {

    }

    func updateUIView(_ uiView: ARView, context: Context) {

    }

}
```

Missing return in instance method expected to return 'ARView'

Setup the Augmented Reality Camera

- To remove the error, and get the desired result we have to declare an Augmented Reality View (ARView)
 - It is responsible to show camera and do AR stuffs
 - We can set a configuration that take care of human body

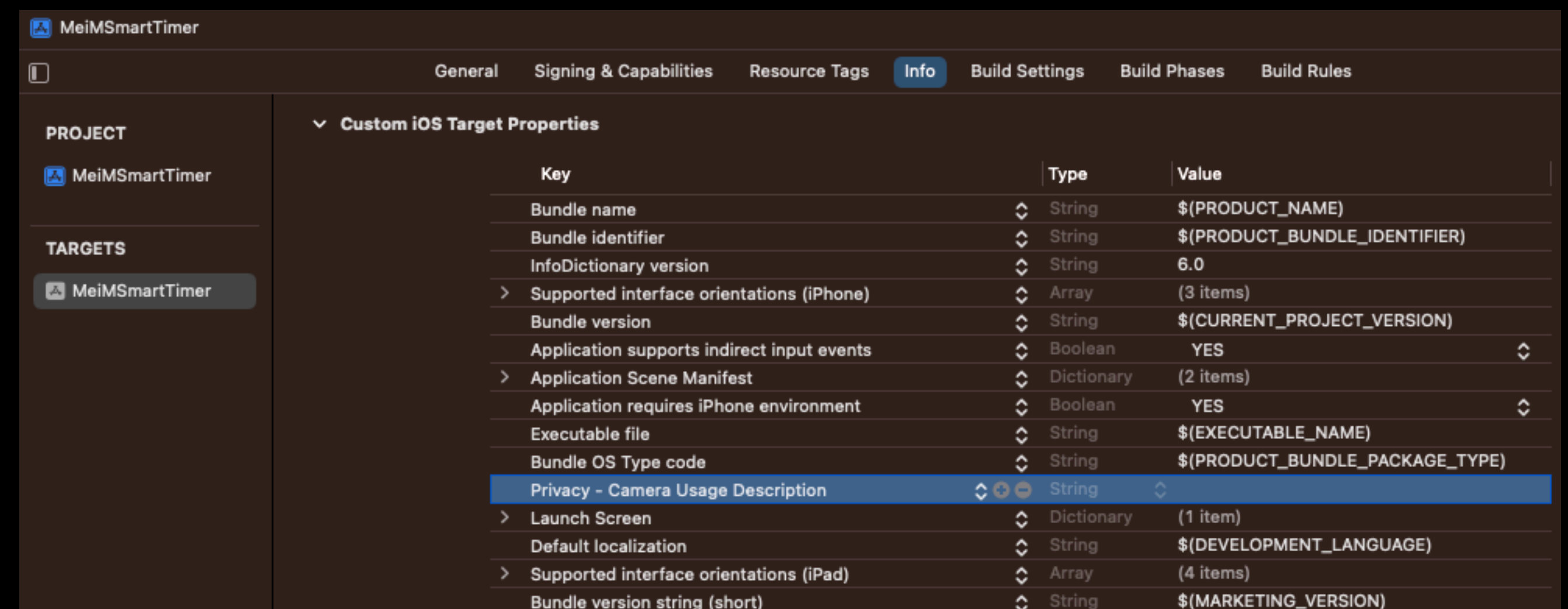
ARViewContainer

- Finally the error disappeared!

```
struct ARViewContainer: UIViewRepresentable {  
  
    func makeUIView(context: Context) -> ARView {  
        let arView = ARView(frame: .zero)  
  
        let configuration = ARBodyTrackingConfiguration()  
        arView.session.run(configuration)  
  
        return arView  
    }  
  
    func updateUIView(_ uiView: ARView, context: Context) {  
  
    }  
  
}
```


ARViewContainer

- Good we are near to test our application for the first time!
- Before to try the app it is important to tell to iOS that our choice is to use the camera
- Click on blu icon, it is the Project Settings file
- Select Info tab and add following text:
 - Privacy - Camera Usage Description



ARViewContainer

- Are we ready to test the app?
- Unfortunately there is another step
- We need to set the iPhone to be used for Development authorize our profile to deploy application on it
- Do this step together! (~ 10 minutes)

ARHandCaptureCoordinator

- We need to intercept the camera output (also known as *frame*)
- We need to realize a *Coordinator* that is able to interface between the camera output and what we want to obtain

ARHandCaptureCoordinator

- Let's create an **ARHandCaptureCoordinator** inside the struct
- Very important is to add the **ARSessionDelegate** to our new class

```
struct ARViewContainer: UIViewRepresentable {  
  
    func makeUIView(context: Context) -> ARView {  
        let arView = ARView(frame: .zero)  
  
        let configuration = ARBodyTrackingConfiguration()  
        arView.session.run(configuration)  
  
        return arView  
    }  
  
    func updateUIView(_ uiView: ARView, context: Context) {  
  
    }  
  
    class ARHandCaptureCoordinator: NSObject, ARSessionDelegate {  
  
        override init() {  
            super.init()  
        }  
  
    }  
  
}
```


ARHandCaptureCoordinator

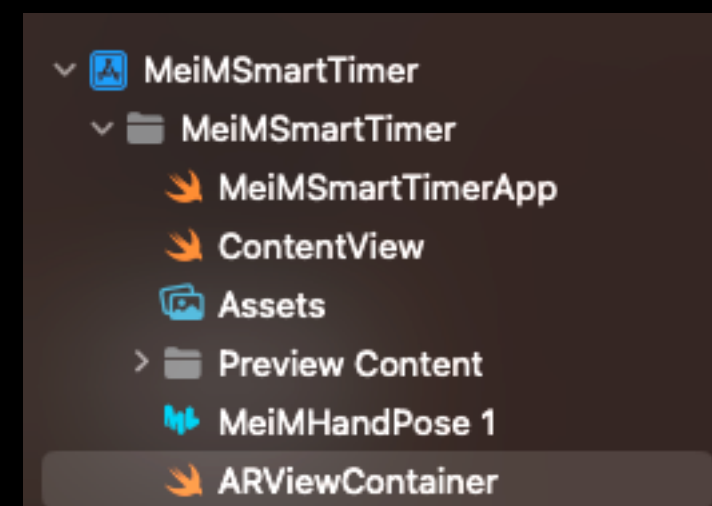
MLModel

- Here we can do a lot of things
- First we have to build the right structure
- Import the mlmodel that you extracted from CreateML
 - It is very simple, just drag and drop the file in the file list
 - ATTENTION! When it tells you to confirm the operation you need to select the checkbox ***Copy item if needed***

ARHandCaptureCoordinator

MLModel

- It should be safe
- In the example image it is called ***MeiMHandPose 1***
 - The icon is different from other files, it shows the CreateML icon



ARHandCaptureCoordinator

Variables declaration

- Now we can add three variables to the Coordinator that represent
 - The model prediction update interval
 - Used to get a smoother user experience
 - The current frame counter
 - It is needed to know how much frames we processed
 - The mlmodel

ARHandCaptureCoordinator

Variables declaration

- Now we can add three variables to the Coordinator

```
class ARHandCaptureCoordinator: NSObject, ARSessionDelegate {  
    var handPosePredictionInterval: Int = 30  
    var frameCounter: Int = 0  
  
    var model: MeiMHandPose_1
```


ARHandCaptureCoordinator

Initialization

- Modify the initializer
- It configures the model to use all computational units (CPU, GPU, Neural Engine)
- Take care of errors

```
override init() {  
    do {  
        let modelConfig = MLModelConfiguration()  
        modelConfig.computeUnits = .all  
  
        model = try MeiMHandPose_1(configuration: modelConfig)  
  
    } catch {  
        fatalError("Model not found")  
    }  
    super.init()  
}
```

ARHandCaptureCoordinator

Session updates

- Do you remember the ARSessionDelegate?
 - It is time to use it
- We need just one function from it
 - The one that handles the frames captured by ARKit

ARHandCaptureCoordinator

Session updates

- Add it at the end of the *init*
- We need to add many more things to use our artificial intelligence model

```
func session(_ session: ARSession, didUpdate frame: ARFrame) {  
    frameCounter += 1  
    let pixelBuffer = frame.capturedImage  
}
```

ARHandCaptureCoordinator

Requests hand pose

- We need to ask the iPhone that we want to detect human hand pose
 - We ask for a maximum of 1 hand
 - Plus we set the algorithm that we want to use
 - There is only one algorithm for this task

```
func session(_ session: ARSession, didUpdate frame: ARFrame) {  
    frameCounter += 1  
    let pixelBuffer = frame.capturedImage  
    let handPoseRequest = VNDetectHumanHandPoseRequest()  
    handPoseRequest.maximumHandCount = 1  
    handPoseRequest.revision = VNDetectHumanHandPoseRequestRevision1  
}
```


ARHandCaptureCoordinator

Requests hand pose

- Our request is done. It's time to submit and handle this task
- When performed it is possible that some errors occur
 - We have to be ready to manage this situation

```
func session(_ session: ARSession, didUpdate frame: ARFrame) {
    frameCounter += 1
    let pixelBuffer = frame.capturedImage
    let handPoseRequest = VNDetectHumanHandPoseRequest()
    handPoseRequest.maximumHandCount = 1
    handPoseRequest.revision = VNDetectHumanHandPoseRequestRevision1

    let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, options: [:])
    do {
        print("Perform Hand Pose Estimation")
        try handler.perform([handPoseRequest])
    } catch {
        assertionFailure("Human Hand Pose Request Failed: \(error)")
    }
}
```

ARHandCaptureCoordinator

Retrieves hand pose

- As result we should obtain the hand pose. Why should?
 - It can happen that not all frames have hands
 - Exit if it happens

```
let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, options: [:])
do {
    print("Perform Hand Pose Estimation")
    try handler.perform([handPoseRequest])
} catch {
    assertionFailure("Human Hand Pose Request Failed: \(error)")
}
```

```
guard let handPoses = handPoseRequest.results, !handPoses.isEmpty else {
    print("[Hand Pose Estimation] isEmpty")
    return
}
```

```
let handObservation = handPoses.first
```

ARHandCaptureCoordinator

Retrieves hand pose

- Now we have to check if enough time is elapsed to perform a prediction

```
guard let handPoses = handPoseRequest.results, !handPoses.isEmpty else {
    print("[Hand Pose Estimation] isEmpty")
    return
}

let handObservation = handPoses.first

if self.frameCounter % self.handPosePredictionInterval == 0 {
    self.frameCounter = 0
    print("[Hand Pose Estimation] frame counter \(self.frameCounter)")
}
```


ARHandCaptureCoordinator

Retrieves hand pose

- Check check check
- It could appear annoying, but it save you and your clients from lost money and patience

```
if self.frameCounter % self.handPosePredictionInterval == 0 {  
    self.frameCounter = 0  
    print("[Hand Pose Estimation] frame counter \ \(self.frameCounter)")  
  
    guard let keypointsMultiArray = try? handObservation?.keypointsMultiArray() else {  
        fatalError()  
    }  
    print("[Hand Pose Estimation] Found keypoints")  
}
```

ARHandCaptureCoordinator

Retrieves hand pose

- Finally the prediction!
- It uses a multi dimensional array and returns a dictionary with a score for each class
 - It is the common behavior of any classifier
- Sure we are interested only on very confident estimation

```
guard let keypointsMultiArray = try? handObservation?.keypointsMultiArray() else {
    fatalError()
}
print("[Hand Pose Estimation] Found keypoints")

let handPosePrediction = try! self.model.prediction(poses: keypointsMultiArray)

let confidence = handPosePrediction.labelProbabilities[handPosePrediction.label]!

if confidence > 0.9 {
    print("[Hand Pose Estimation] confidence: \(confidence)")
    print("[Hand Pose Estimation] class label: \(handPosePrediction.label)")
}
```

ARHandCaptureCoordinator

Almost done

- Before to test it, we need to make coordinator accessible

```
func updateUIView(_ uiView: ARView, context: Context) {  
}  
  
func makeCoordinator() -> ARHandCaptureCoordinator {  
    ARHandCaptureCoordinator()  
}
```


ARHandCaptureCoordinator

Almost done

- Also, we have to notify that we want to use it

```
func makeUIView(context: Context) -> ARView {
    let arView = ARView(frame: .zero)

    let configuration = ARBodyTrackingConfiguration()
    arView.session.run(configuration)

    arView.session.delegate = context.coordinator

    return arView
}
```

ARHandCaptureCoordinator

First review

TEST IT!

The output can be read from the terminal log

ARHandCaptureCoordinator

Communication

- It's time that this view can start talking with the ContentView
 - It is the main container where all components will be rendered
- Go up to the start of the ARViewContainer and add two Bindings

ARHandCaptureCoordinator

State/Binding and beyond

- Quick tip:
 - State and Binding are two fundamental functionalities that make the view able to update the state of its graphical elements from itself (using @State) and inside another linked view (using @Binding)
 - It will be more clear in the next steps

ARHandCaptureCoordinator

Communication

- It's time that this view can start talking with the ContentView
- Add the following to lines (take care to add @Binding)

```
struct ARViewContainer: UIViewRepresentable {  
  
    @Binding var confidenceScore: Double  
    @Binding var classLabel: String  
  
    func makeUIView(context: Context) -> ARView {  
  
        // MARK: 3. Create the view  
        let arView = ARView(frame: .zero)
```

ARHandCaptureCoordinator

Communication

- We need to create an interface from ARViewContainer to the coordinator
- We need to add similar variables also in coordinator

```
class ARHandCaptureCoordinator: NSObject, ARSessionDelegate {  
  
    var confidenceScore: Binding<Double>  
    var classLabel: Binding<String>  
  
    var handPosePredictionInterval: Int = 30  
    var frameCounter: Int = 0
```


ARHandCaptureCoordinator

Communication

- Modify the initializer
- Remove the override keyword and add variables

```
override init() {
    do {
        let modelConfig = MLModelConfiguration()
        modelConfig.computeUnits = .all

        model = try MeiMHandPose_1(configuration: modelConfig)
    } catch {
        fatalError("Model not found")
    }
    super.init()
}
```

Before

```
init(confidenceScore: Binding<Double>, classLabel:
    Binding<String>) {
    do {
        let modelConfig = MLModelConfiguration()
        modelConfig.computeUnits = .all

        model = try MeiMHandPose_1(configuration: modelConfig)
    } catch {
        fatalError("Model not found")
    }

    self.confidenceScore = confidenceScore
    self.classLabel = classLabel
}
```

After

ARHandCaptureCoordinator Communication

- Set these variables to pass from ARViewContainer to Coordinator
 - Take care to insert the dollar symbol \$

```
func updateUIView(_ uiView: ARView, context: Context) {  
}  
  
func makeCoordinator() -> ARHandCaptureCoordinator {  
    ARHandCaptureCoordinator()  
}
```

Before

```
func makeCoordinator() -> ARHandCaptureCoordinator {  
    ARHandCaptureCoordinator(confidenceScore: $confidenceScore,  
                             classLabel: $classLabel)  
}
```

After

ARHandCaptureCoordinator

Communication

- Set their values in the session function

```
if confidence > 0.9 {  
    print("[Hand Pose Estimation] confidence: \((confidence)")  
    print("[Hand Pose Estimation] class label:  
        \((handPosePrediction.label)")  
  
    confidenceScore.wrappedValue = confidence  
    classLabel.wrappedValue = handPosePrediction.label  
}
```


ContentView

Communication

- Come back to ContentView
- Start to style it

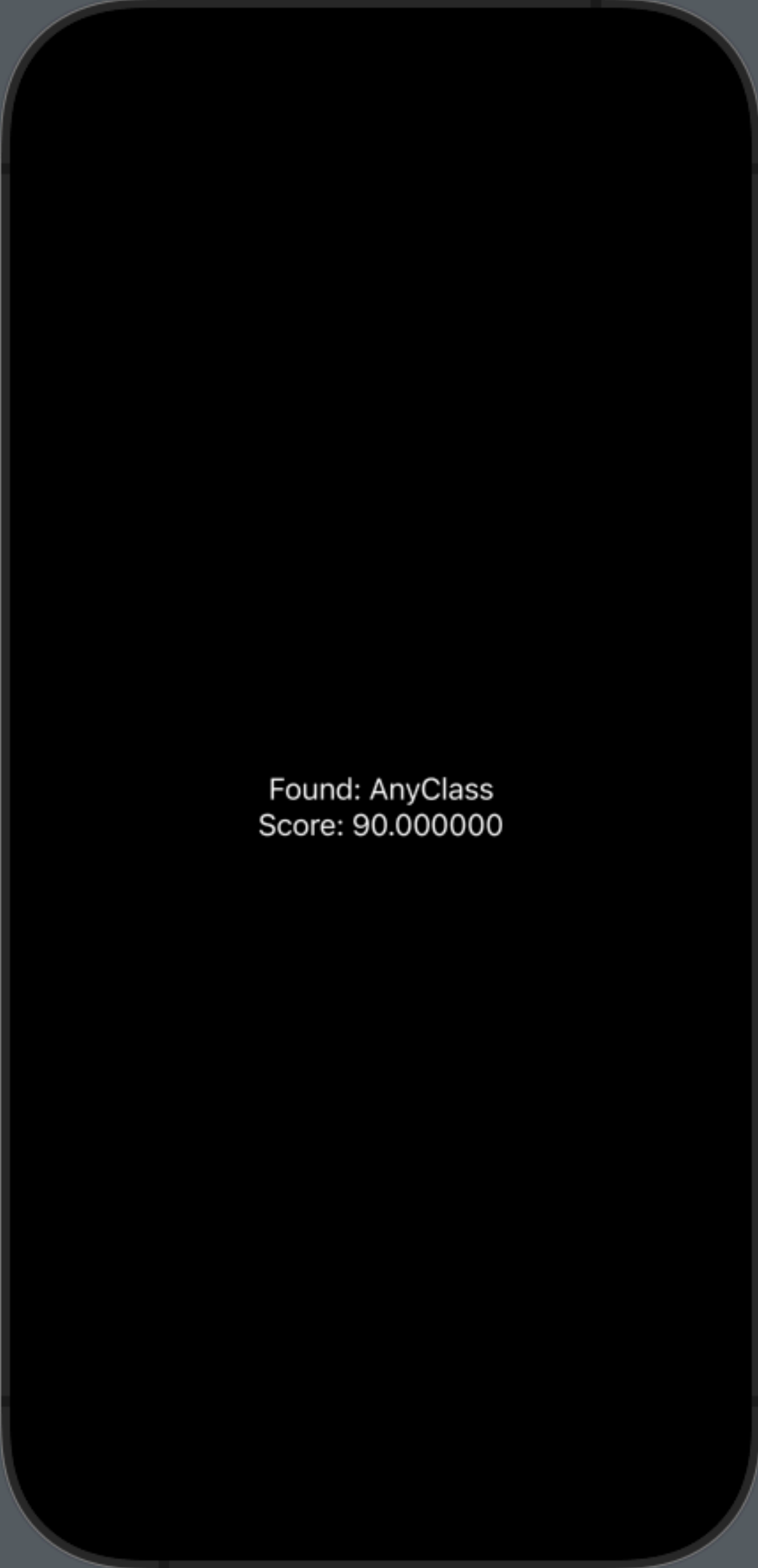
```
struct ContentView: View {  
    @State private var confidenceScore: Double = 0.0  
    @State private var confidenceLabel: String = ""  
  
    var body: some View {
```

ContentView

Communication

- Add the binding of ARViewContainer
- Add a text with label and score

```
ARViewContainer(confidenceScore: $confidenceScore,  
                classLabel: $confidenceLabel)  
    .edgesIgnoringSafeArea(.all)  
    .overlay {  
        VStack {  
            if !confidenceLabel.isEmpty {  
                Text("Found: \$(confidenceLabel)")  
                    .foregroundColor(.white)  
                Text("Score: \$(confidenceScore)")  
                    .foregroundColor(.white)  
            }  
        }  
    }  
}
```



Found: AnyClass
Score: 90.000000

ContentView

Communication

- We want to increase a timer based on gestures

```
struct ContentView: View {  
    @State private var confidenceScore: Double = 0.0  
    @State private var confidenceLabel: String = ""  
    @State private var timeRemaining: Int = 0  
  
    private var timer = Timer.publish(every: 1, on: .main, in:  
        .common).autoconnect()
```


ContentView

Timer

- It is needed to decrease the timer automatically

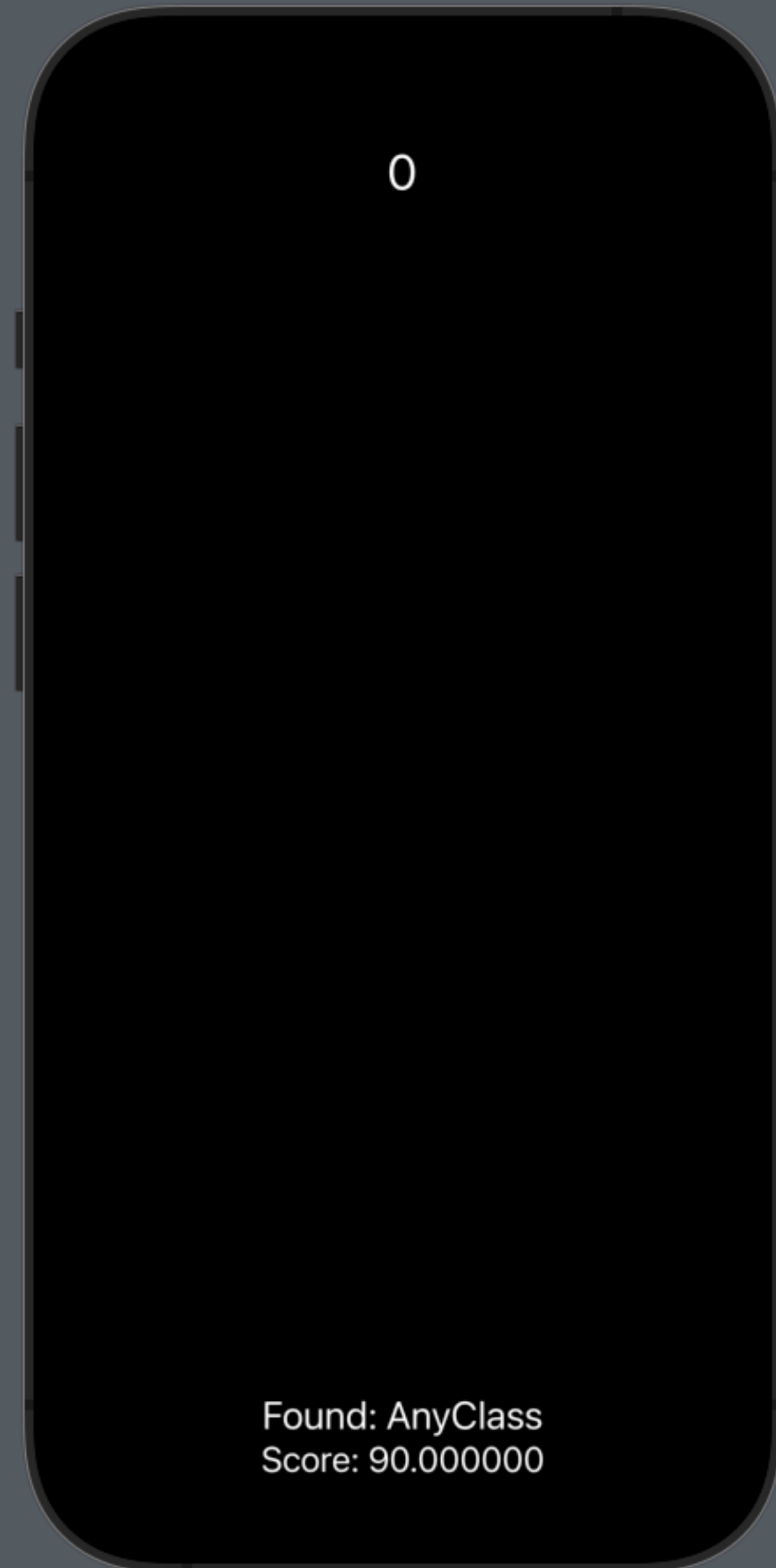
```
ARViewContainer(confidenceScore: $confidenceScore,
    classLabel: $confidenceLabel)
    .edgesIgnoringSafeArea(.all)
    .overlay {
        VStack {
            if !confidenceLabel.isEmpty {
                Text("Found: \($confidenceLabel)")
                    .foregroundColor(.white)
                Text("Score: \($confidenceScore)")
                    .foregroundColor(.white)
            }
        }
    }
    }.onReceive(timer) {
        _ in
        if timeRemaining > 0 {
            timeRemaining -= 1
        }
    }
}
```

ContentView

Timer

- We have the algorithm to decrease the timer, but we does not have a timer interface
- We want to obtain a UI with found class and score on bottom and timer on top

```
ARViewContainer(confidenceScore: $confidenceScore,
  classLabel: $confidenceLabel)
  .edgesIgnoringSafeArea(.all)
  .overlay {
    VStack {
      Text("\(timeRemaining)")
        .foregroundColor(.white)
        .font(.title)
      Spacer()
      if !confidenceLabel.isEmpty {
        Text("Found: \((confidenceLabel)")
          .foregroundColor(.white)
          .font(.title2)
        Text("Score: \((confidenceScore)")
          .foregroundColor(.white)
          .font(.title3)
      }
    }
  }
}
```

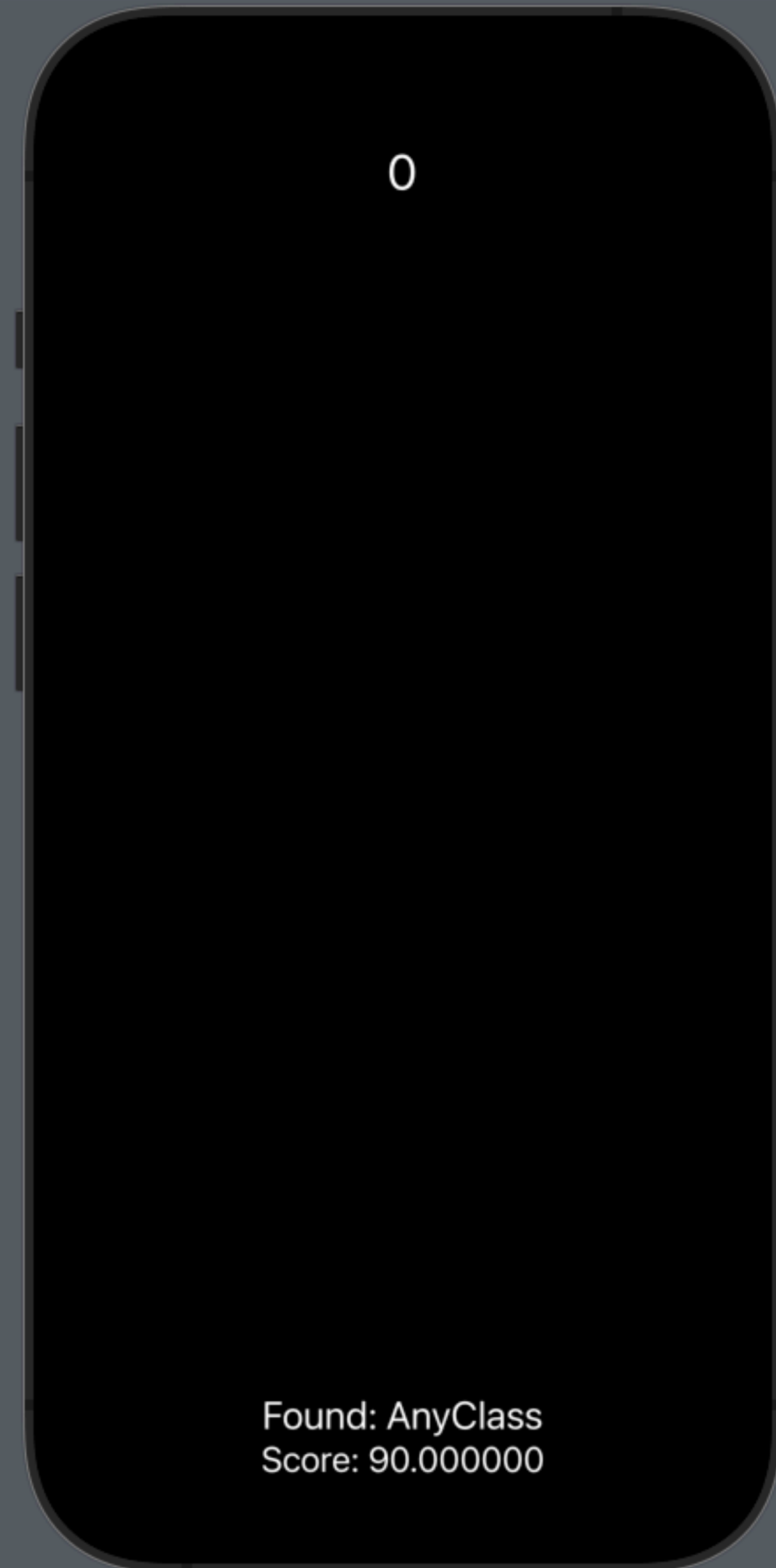


ContentView

Timer

- Just a number is not so common for a timer
- We can do better
 - From 0
 - To 00:00

```
        }  
        }.onReceive(timer) {  
            in  
            if timeRemaining > 0 {  
                timeRemaining -= 1  
            }  
        }  
    }  
}  
  
func getTimeRepresentation() -> String {  
    let formatter = DateComponentsFormatter()  
    formatter.zeroFormattingBehavior = .pad  
    formatter.allowedUnits = [.minute, .second]  
    return formatter.string(from:  
        TimeInterval(timeRemaining))!  
}  
}
```

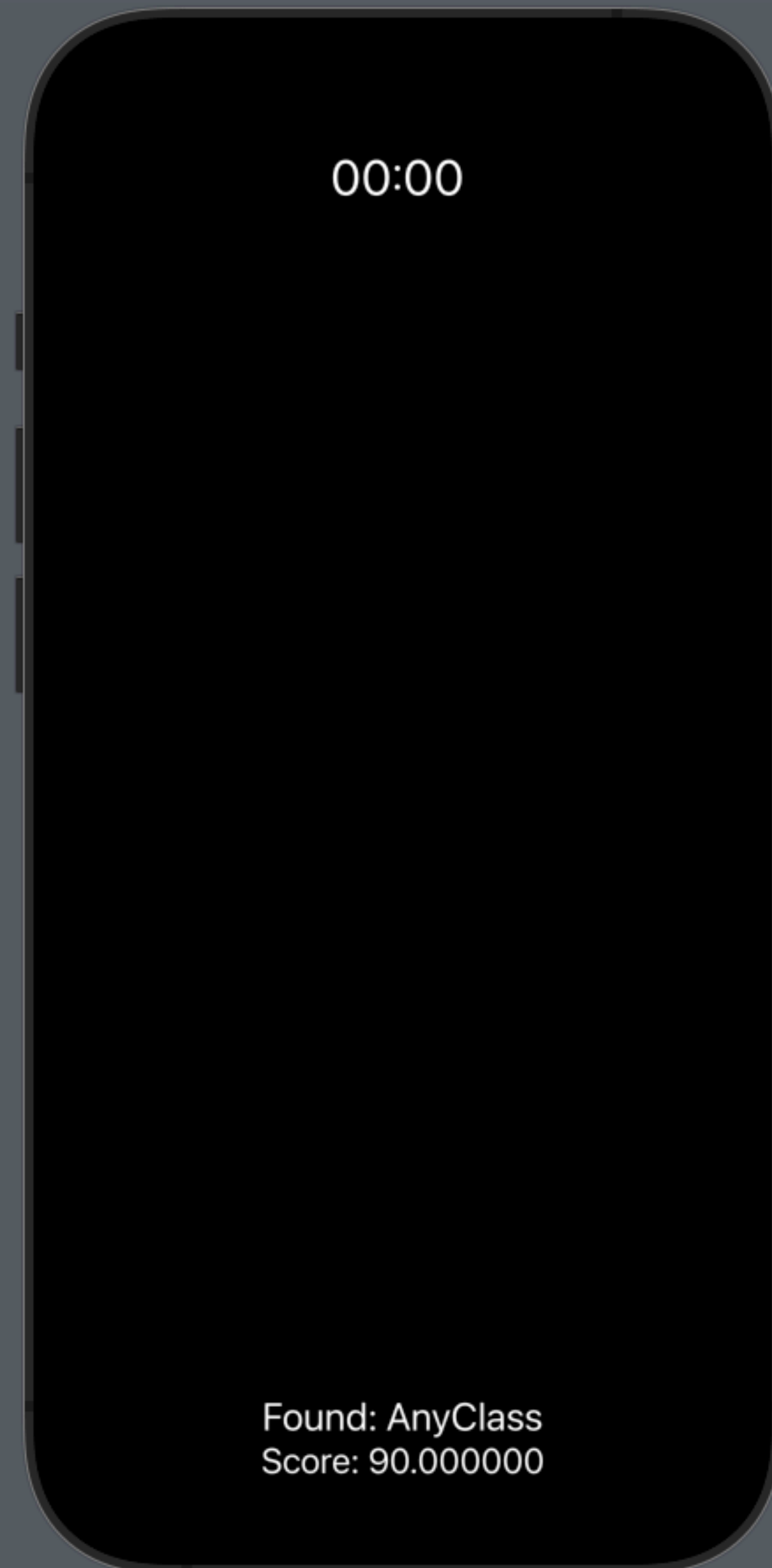


ContentView

Timer

- Just a number is not so common for a timer
- We can do better
 - From 0
 - To 00:00

```
ARViewContainer(confidenceScore: $confidenceScore,  
                classLabel: $confidenceLabel)  
    .edgesIgnoringSafeArea(.all)  
    .overlay {  
        VStack {  
            Text(getTimeRepresentation())  
                .foregroundColor(.white)  
                .font(.title)  
            Spacer()  
            if !confidenceLabel.isEmpty {
```



ContentView

Autoincrement Timer

- For now our timer is not so smart
- We can use the output of the Machine Learning model to add time automatically
- Add a new functionality when the *confidenceLabel* is updated

```
}.onReceive(timer) {
    _ in
    if timeRemaining > 0 {
        timeRemaining -= 1
    }
}

}.onChange(of: confidenceLabel) {
    newLabel in
    switch newLabel {
    case "One":
        timeRemaining += 60
    case "Two":
        timeRemaining += 120
    case "Thumb":
        timeRemaining += Int.random(in: 0..100)
    default:
        print("background, we don't need it")
    }
}
}
```

The End

**Try the
application**

Q&A

- Comments?
- Questions?
- Curiosity?
-