EEP 523:
MOBILE APPLICATIONS FOR SENSING AND CONTROL

SUMMER 2021
Lecture 5

Tamara Bonaci

tbonaci@uw.edu

Agenda

- Canvas - lecture notes
- GitHub repo

HW1
HW2

HW1 - HW6

HW6 becomes optional as well

- HW4, 5 -

Aug 23rd

Aug 22nd -HW5

MLkit

Review: Sensors in Android

The Android platform supports three broad categories of sensors:

Motion sensors

- Measure: acceleration forces and rotational forces along three axes.
- Types: accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

Environmental sensors

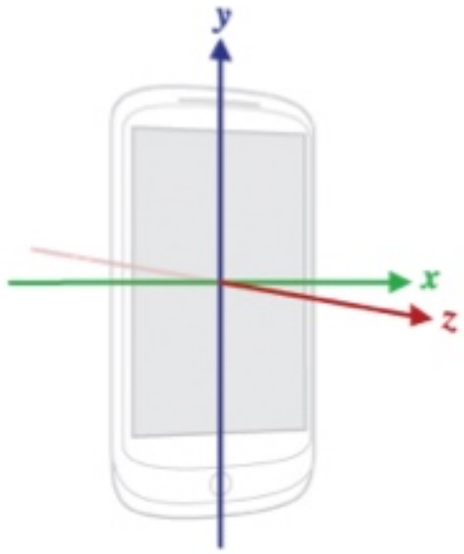
- Measure: ambient air temperature and pressure, illumination, and humidity.
- Types: barometers, photometers, and thermometers.

Position sensors

- Measure: physical position of a device.
- Types: orientation sensors and magnetometers.

Review: Coordinate System

Coordinate system
(relative to a mobile device)
used by the Sensor API.



- The axes are not swapped when the device's screen orientation changes—**the sensor's coordinate system never changes as the device moves**

- your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And **the sensor coordinate system is always based on the natural orientation of a device**.

Review: Accelerometer

- Acceleration of the device along the 3 sensor axes.
- The measured acceleration includes both the physical acceleration (change of velocity) and the gravity.
- The measurement is reported in the x, y and z fields of *sensors_event_t.acceleration.*
- All values are in SI units **(m/s^2)** and measure the acceleration of the device minus the force of gravity along the 3 sensor axes.

Review: Accelerometer in Android

Sensor.TYPE_ACCELEROMETER:

SensorEvent.values[0]	Acceleration force along the x axis (including gravity).
SensorEvent.values[1]	Acceleration force along the y axis (including gravity).
SensorEvent.values[2]	Acceleration force along the z axis (including gravity).

The force of gravity is always influencing the measured acceleration

- Device sitting on a table (and not accelerating), the accelerometer reads a magnitude of $g = 9.81\ m/s^2$
- Device in free fall and therefore rapidly accelerating toward the ground at $9.81\ m/s^2$, its accelerometer reads a magnitude of $g = 0\ m/s^2$

Review: Accelerometer in Android

To measure the real acceleration of the device, the contribution of the force of gravity must be removed from the accelerometer data.

- Use a low-pass filter to isolate the force of gravity
- Apply a high-pass filter to remove the force of gravity

```
override fun onSensorChanged(event: SensorEvent) {
    // alpha is calculated as t / (t + dT)
    // with t, the low-pass filter's time-constant
    // and dT, the event delivery rate

    val alpha = 0.8f

    gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0]
    gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1]
    gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2]

    linear_acceleration[0] = event.values[0] - gravity[0]
    linear_acceleration[1] = event.values[1] - gravity[1]
    linear_acceleration[2] = event.values[2] - gravity[2]
}
```

t: rough representation of the latency that the filter adds to the sensor events,

dt: the sensor's event delivery rate

Review: Accelerometer in Android

- Not necessary to get accelerometer events at a very high rate
- By using a slower rate (SENSOR_DELAY_UI), we get an automatic low-pass filter, which "extracts" the gravity component of the acceleration.
- As an added benefit, we use less power and CPU resources

```
mSensorManager.registerListener(this, mSensor, SensorManager.SENSOR_DELAY_GAME)
```

Gyroscope

EE P 523, Lecture 5

- DBS - Jeff's work
- Haptic password

Gyroscope

Measure rotational (angular) velocity in 1, 2, or 3 directions

- ❑ 3-axis gyroscopes often implemented with a 3-axis accelerometer to provide a 6 degree-of-freedom motion tracking system

- ❑ Basic types of gyroscope:
  - Rotary (classical) gyroscopes
  - Vibrating Structure Gyroscope
  - Optical Gyroscopes

Gyroscope

Applications

- The gyroscope helps the accelerometer out with understanding which way your phone is orientated—it adds another level of precision, so those 360-degree photo spheres really look as impressive as possible.

- Gyroscopes aren’t exclusive to phones.
  They’re used in altimeters inside aircraft to determine altitude and position, for example, and to keep cameras steady on the move.

Gyroscope

Gyroscope vs Accelerometer

- So, putting in simplest of the words – **gyroscope** provides velocity, and **accelerometer** provides speed.
- Gyroscope is capable of providing precision motion inside the app functionality.
- User can execute majority of the tasks with the motion of the device only.

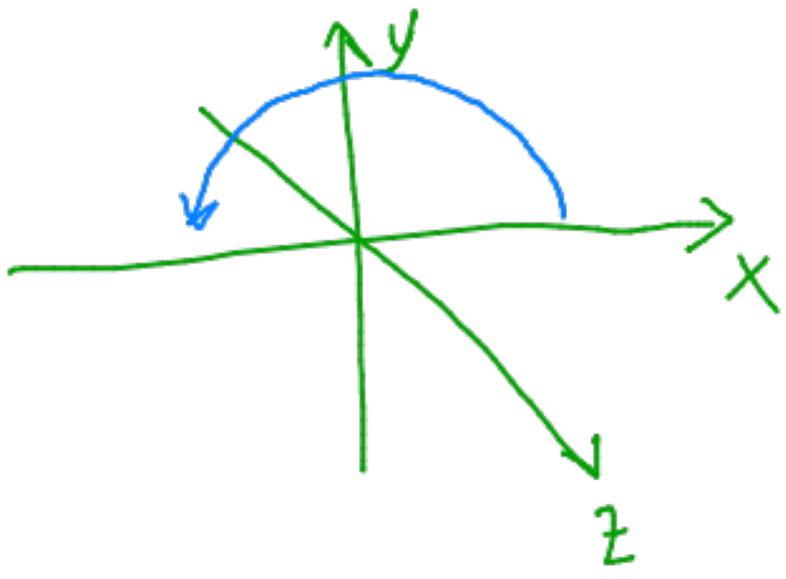
Gyroscope in Android

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
```

• All values are in radians/second and measure the rate of rotation around the device's local X, Y and Z axis.

•Rotation is positive in the counter-clockwise direction. That is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter-clockwise.

values[0]: Angular speed around the x-axis
values[1]: Angular speed around the y-axis
values[2]: Angular speed around the z-axis



Gyroscope in Android

```
Override fun onSensorChanged(event: SensorEvent) {
    // This time step's delta rotation to be multiplied by the current rotation
    // after computing it from the gyro sample data.
    if (timestamp !== 0) {
        val dT = (event.timestamp - timestamp) * NS2S
        // Axis of the rotation sample, not normalized yet.
        var axisX = event.values[0]
        var axisY = event.values[1]
        var axisZ = event.values[2]

        // Calculate the angular speed of the sample
        val omegaMagnitude = sqrt((axisX * axisX + axisY * axisY + axisZ * axisZ).toDouble())

        // Normalize the rotation vector if it's big enough to get the axis
        if (omegaMagnitude > EPSILON) {
            axisX /= omegaMagnitude.toFloat()
            axisY /= omegaMagnitude.toFloat()
            axisZ /= omegaMagnitude.toFloat()
        }

        // Integrate around this axis with the angular speed by the time step
        // in order to get a delta rotation from this sample over the time step
        // We will convert this axis-angle representation of the delta rotation
        // into a quaternion before turning it into the rotation matrix.
        val thetaOverTwo = omegaMagnitude * dT / 2.0f
        val sinThetaOverTwo = sin(thetaOverTwo)
        val cosThetaOverTwo = cos(thetaOverTwo)
        deltaRotationVector[0] = (sinThetaOverTwo * axisX).toFloat()
        deltaRotationVector[1] = (sinThetaOverTwo * axisY).toFloat()
        deltaRotationVector[2] = (sinThetaOverTwo * axisZ).toFloat()
        deltaRotationVector[3] = cosThetaOverTwo.toFloat()
    }
    timestamp = event.timestamp.toInt()
    val deltaRotationMatrix = FloatArray(9)
    SensorManager.getRotationMatrixFromVector(deltaRotationMatrix, deltaRotationVector)
    // User code should concatenate the delta rotation we computed with the current
    // rotation in order to get the updated rotation.
    // rotationCurrent = rotationCurrent * deltaRotationMatrix;
}
```

Typically, the output of the gyroscope is integrated over time to calculate a rotation describing the change of angles over the time step

WWU - Shriranga More

Additional Base sensors

- **Compass**

- **Proximity sensor**:
  When you lift phone to your ear, the proximity sensor can turn off the display to save power and prevent accidental dialing.

- **Ambient light sensor:**
  Automatically brightens the display when you're in sunlight or a bright room and dims it in darker places.

Composite Sensors

EE P 523, Lecture 5

Composite Sensors

Generates data by processing and/or fusing data from one or several physical sensors -> Any sensor that is not a base sensor is called a composite sensor

Examples of composite sensors include:

- **Step detector** and **Significant motion**
usually based on an accelerometer, but could be based on other sensors as well, if the power consumption and accuracy was acceptable.

- **Game rotation vector,** based on an accelerometer and a gyroscope.

- **Uncalibrated gyroscope**, similar to the gyroscope base sensor,
but with the bias calibration being reported separately instead of being corrected in the measurement.

Composite Sensors

Sensor type	Category	Underlying physical sensors	Reporting mode
Game rotation vector	Attitude	Accelerometer, Gyroscope MUST NOT USE Magnetometer	Continuous
Geomagnetic rotation vector	Attitude	Accelerometer, Magnetometer, MUST NOT USE Gyroscope	Continuous
Glance gesture	Interaction	Undefined	One-shot
Gravity	Attitude	Accelerometer, Gyroscope	Continuous
Gyroscope uncalibrated	Uncalibrated	Gyroscope	Continuous
Linear acceleration	Activity	Accelerometer, Gyroscope (if present) or Magnetometer (if gyro not present)	Continuous
Magnetic field uncalibrated	Uncalibrated	Magnetometer	Continuous
Orientation (deprecated)	Attitude	Accelerometer, Magnetometer PREFERRED Gyroscope	Continuous
Pick up gesture	Interaction	Undefined	One-shot
Rotation vector	Attitude	Accelerometer, Magnetometer, AND (when present) Gyroscope	Continuous
Significant motion	Activity	Accelerometer (or another as long as very low power)	One-shot
Step counter	Activity	Accelerometer	On-change
Step detector	Activity	Accelerometer	Special
Tilt detector	Activity	Accelerometer	Special
Wake up gesture	Interaction	Undefined	One-shot

Reading Data from Sensors

1. Class must implement interface ***SensorEventListener***

```
class MainActivity : AppCompatActivity(), SensorEventListener {
```

1)

2. Override two functions

```
override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {}
```

```
override fun onSensorChanged(event: SensorEvent) {
    if (event.sensor.type != Sensor.TYPE_ACCELEROMETER)
        return
  . . .
}
```

Reading Data from Sensors

3. Create a ***SensorManager*** and ***Sensor*** variable

```
private lateinit var mSensorManager: SensorManager

private val mAccelerometer: Sensor
```

4. Initiate an instance of the SensorManager and Sensor

```
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

mAccelerometer = mSensorManager!!.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
```

5. Register a listener for the particular sensor

```
mSensorManager!!.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_GAME)
```

Sensor Best Practices

Be sure to unregister a sensor's listener when you are done using the sensor or when the sensor activity pauses.

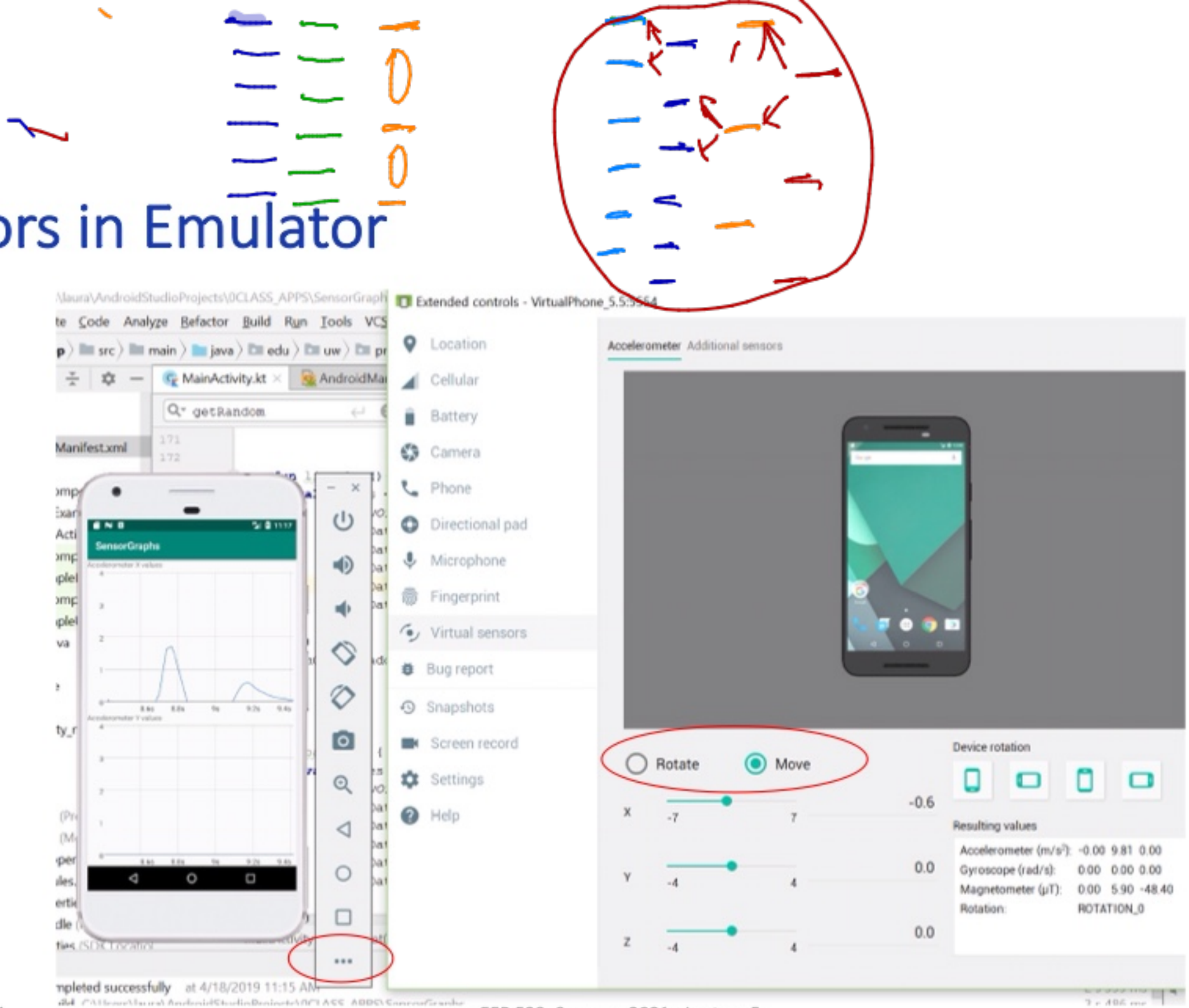
If a listener is registered and its activity is paused, the sensor will continue to acquire data and use the battery resources

```
override fun onPause() {
    super.onDestroy()
    mSensorManager.unregisterListener(this)
}
```

Sensor Best Practices

- Only gather sensor data in the foreground
- On devices running Android 9 (API level 28) or higher, apps running in the background have the following restrictions:
  - Sensors that use the continuous reporting mode (link), such as accelerometers and gyroscopes, don't receive events.
  - Sensors that use the on-change or one-shot reporting modes don't receive events.
- Given these restrictions, it's best to detect sensor events either when your app is in the foreground or as part of a foreground service.

Sensors in Emulator



Android Supported Sensors

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.

Android Supported Sensors

`TYPE_MAGNETIC_FIELD`	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
`TYPE_ORIENTATION`	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the `getRotationMatrix()` method.	Determining device position.
`TYPE_PRESSURE`	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

Android Supported Sensors

TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

Libraries in Android

EE P 523, Lecture 5

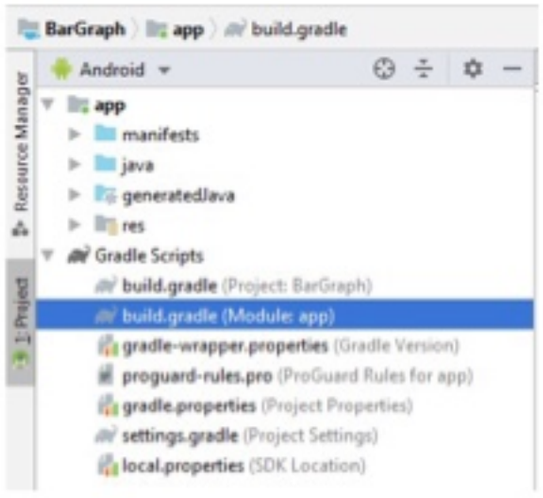
Libraries

- Many Android developers have produced useful **libraries**.
- There is a **Maven repository** to store various libraries.
- This makes it easy to add them to your Android Studio projects.
- Most libraries use permissive licenses so that you can use them for free and can include them in the code of commercial apps/products.
- (Some libraries must be downloaded as .JARs and added manually to your project)

Adding a Library to Your Project

- Edit the **build.gradle** file for your 'app' module and add lines to the following section at the bottom.

- You can usually find out what file name to write below by going to various libraries' home pages / GitHub pages.




```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.android.support:support-v4:28.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    implementation 'com.jjoe64:graphview:4.2.2'
}
```

Picasso Library

- **Picasso** is a powerful library for manipulating images.
  - written by Square, inc.
  - http://square.github.io/picasso/

- To add Picasso to your project:

MAVEN

```
<dependency>
  <groupId>com.squareup.picasso</groupId>
  <artifactId>picasso</artifactId>
  <version>(insert latest version)</version>
</dependency>
```

GRADLE

```
implementation 'com.squareup.picasso:picasso:(insert latest version)'
```

Picasso Library

In your app's Kotlin code, write:

```
Picasso.with(this) .load("url") .into(ImageView)
```

Example

```
ImageView img = (ImageView) findViewById(R.id.photo);
Picasso.with(this)
      .load("http://i.imgur.com/DvpvklR.png")
      .into(img);
```

Picasso Library

Method	Description
`centerCrop()`	center and crop image inside view
`centerInside()`	resize image proportionally inside view
`error(id)`	show given drawable as error
`fetch()`	download image in the background
`fit()`	resize image to fit view bounds
`get()`	return image as a Bitmap
`into(view)`	puts image into given view
`placeholder(id)`	show given drawable while loading
`resize(width, height)`	change image size in pixels
`rotate(degrees)`	rotate clockwise
`tag("tag")`	attaches a "tag" to a loading image (useful for bulk operations shown later)
`transform(trans)`	apply complex transformations

Picasso Library

Method	Description
`cancelRequest(`*`view`*`)`	abort any image loading in that view
`cancelTag("`*`tag`*`")`	cancel all images with given tag
`invalidate("`*`url`*`")` `invalidate(`*`File`*`)`	flush out cache of given image, so it will be re-downloaded the next time

`load("`*`url`*`")`
`load(`*`id`*`)`
`load(`*`File`*`)`
	load an image from various sources

`pauseTag("`*`tag`*`")`	pause all image loads for given tag
`resumeTag("`*`tag`*`")`	unpause all image loads for given tag
`shutdown()`	stop entire Picasso system
`with(`*`context`*`)`	use given activity/fragment as context

Libraries of Interest

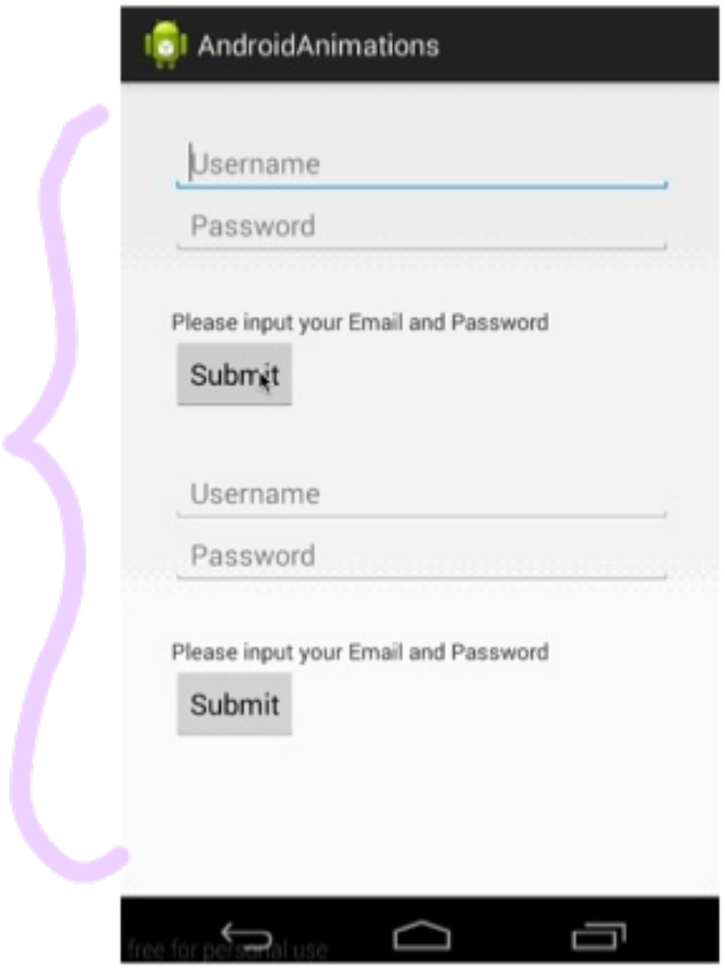
- **Android-Bootstrap**
  Customizable widgets not normally available in Android
  https://github.com/Bearded-Hen/Android-Bootstrap

- **Ion**
  make it easier to download files from the web.
  https://github.com/koush/ion

- **ButterKnife**
  Popular library intended to simplify usage of Android widgets and events in Java code.
  http://jakewharton.github.io/butterknife/

- Ambitious Android user named ***daimajia*** has created several libraries, including one to do animation effects on Views.
  https://github.com/daimajia/AndroidViewAnimations

Animation Library



Graph View

Open-source graph plotting library for Android

- Adding GraphView to your project:
  Add that line to your ***build.gradle*** file under your **app** directory into the dependencies block:

```
implementation 'com.jjoe64:graphview:4.2.2'
```

Graph View: Line graph

```
fun lineplot(){
   val series = LineGraphSeries(
        arrayOf(
            DataPoint(0.0, 1.0),
            DataPoint(1.0, 5.0),
            DataPoint(2.0, 3.0),
            DataPoint(3.0, 2.0),
            DataPoint(4.0, 6.0)
        )
    )
    mGraphX.addSeries(series)
}
```

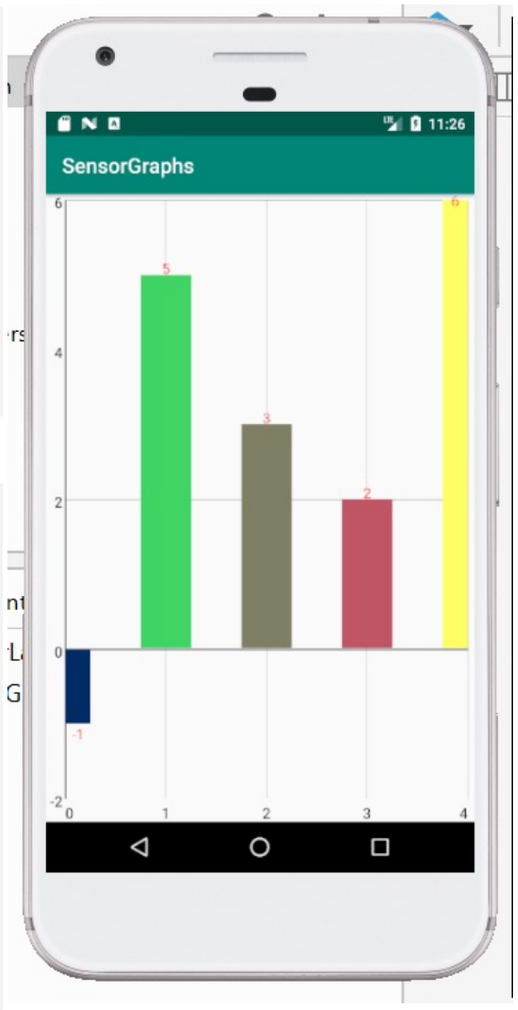
Graph View: Bar graph

```
fun barplot(){
    val series = BarGraphSeries(
        arrayOf(
            DataPoint(0.0, -1.0),
            DataPoint(1.0, 5.0),
            DataPoint(2.0, 3.0),
            DataPoint(3.0, 2.0),
            DataPoint(4.0, 6.0)
        )
    )
    mGraphX.addSeries(series)

      // styling
    series.setValueDependentColor {data ->Color.rgb(data.x.toInt() * 255 / 4,
            Math.abs(data.y * 255 / 6).toInt(),100
        )
    }
    series.spacing = 50

   // draw values on top
    series.isDrawValuesOnTop = true
    series.valuesOnTopColor = Color.RED
}
```

Graph View

```
fun barplot(){ //call this function anywhere, on the onCreate for example
    val series = BarGraphSeries(
        arrayOf(
            DataPoint(0.0, -1.0),
            DataPoint(1.0, 5.0),
            DataPoint(2.0, 3.0),
            DataPoint(3.0, 2.0),
            DataPoint(4.0, 6.0)     )
    )
   mGraphX.addSeries(series)
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
              android:orientation="vertical"
        tools:context=".MainActivity">
    <com.jjoe64.graphview.GraphView
            android:layout_weight="1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/mGraphX"/>
</LinearLayout>
```



Graph View: Real Time Graph

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    mSeriesXaccel = LineGraphSeries()
    mSeriesYaccel = LineGraphSeries()
    initGraphRT(mGraphX,mSeriesXaccel!!)
    initGraphRT(mGraphY,mSeriesYaccel!!)
}
```

```
override fun onSensorChanged(event: SensorEvent) {

    if (event.sensor.type != Sensor.TYPE_ACCELEROMETER)
        return
    linear_acceleration[0] = event.values[0]
    linear_acceleration[1] = event.values[1]
    linear_acceleration[2] = event.values[2]


    val xval = System.currentTimeMillis()/1000.toDouble()//graphLastXValue += 0.1
    mSeriesXaccel!!.appendData(DataPoint(xval, linear_acceleration[0].toDouble()),
true, 50)
    mSeriesYaccel!!.appendData(DataPoint(xval, linear_acceleration[1].toDouble()),
true, 50)
}
```

Face Detection with MLKit

EE P 523, Lecture 5

Firebase

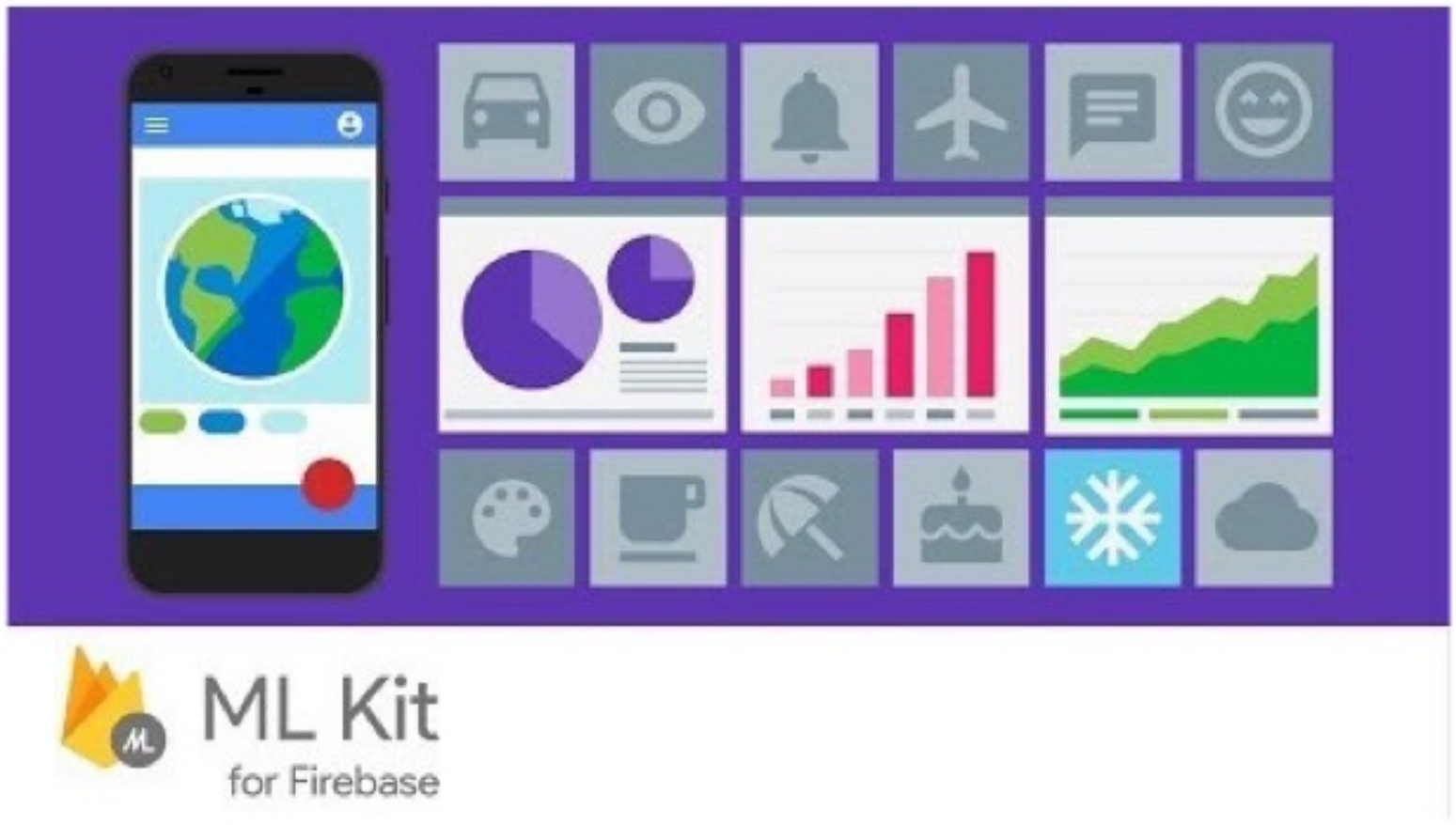
Mobile and web application development platform

In this course we will learn about and use

- Remote databases (Lecture 6-7)
- Machine Learning tools -> **Face Detection**

ML Kit

Face Detection: *MLkit*



Use Firebase for Android

Add Firebase you your Android App:

1. Create a project in your Firebase console

2. Add an Android app
   a) from Android Studio OR
   b) from firebase cosole

https://firebase.google.com/docs/android/setup#assistant

Add Firebase to Android App

1. Go to Firebase online and create a new account

2. Add new project

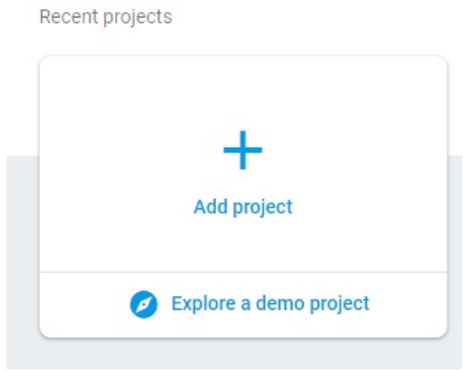
3. Create and Android App where you are going to use Firebase

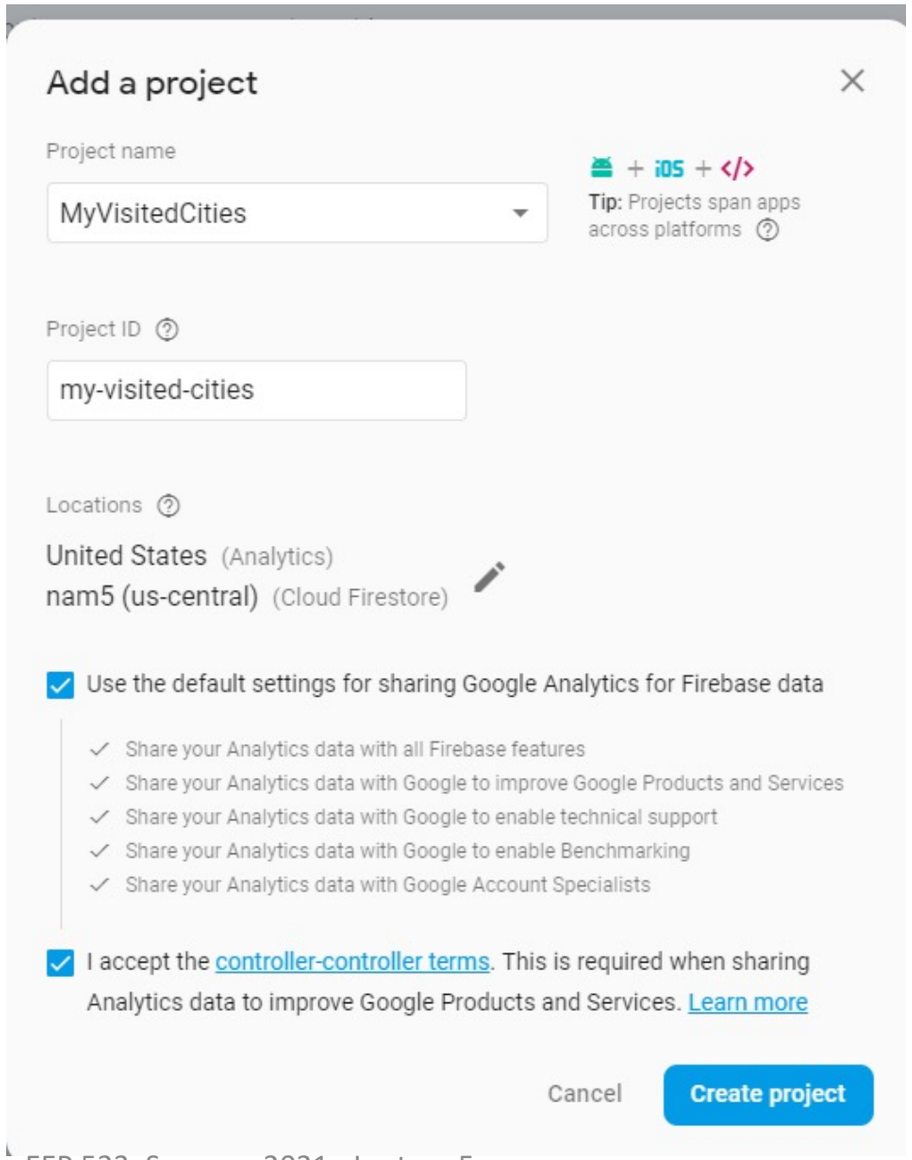
4. Add Firebase to your Android App

5. Connect your App to Firebase
   In Android IDE, go to Tools -> Firebase ->

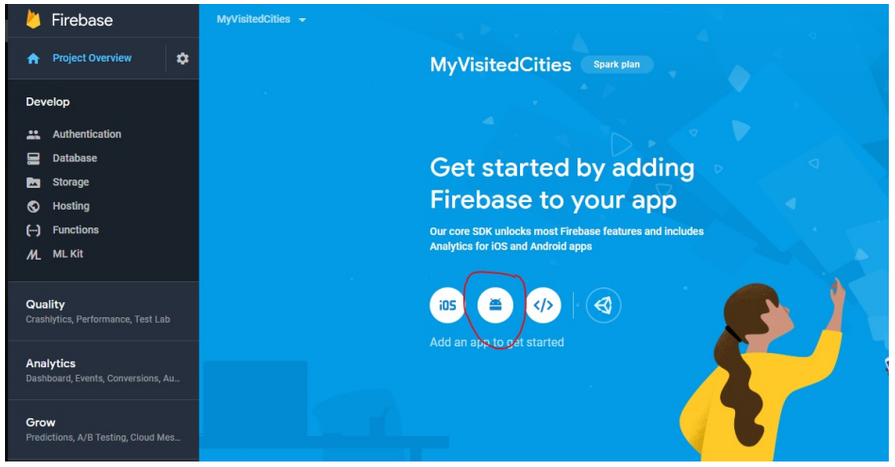
6. Follow the steps

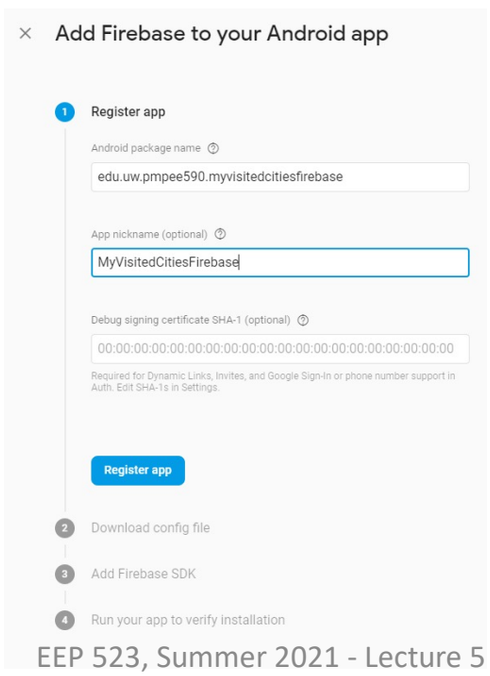
Add Project in Firebase console



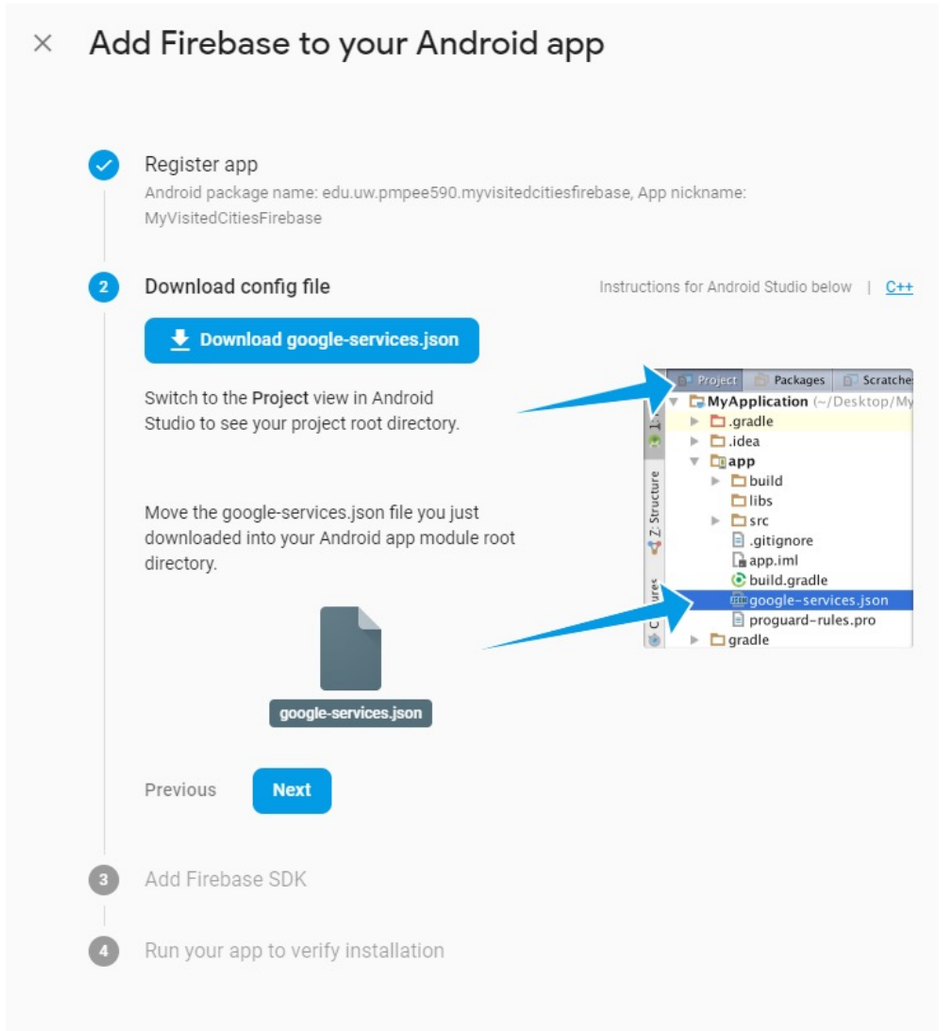


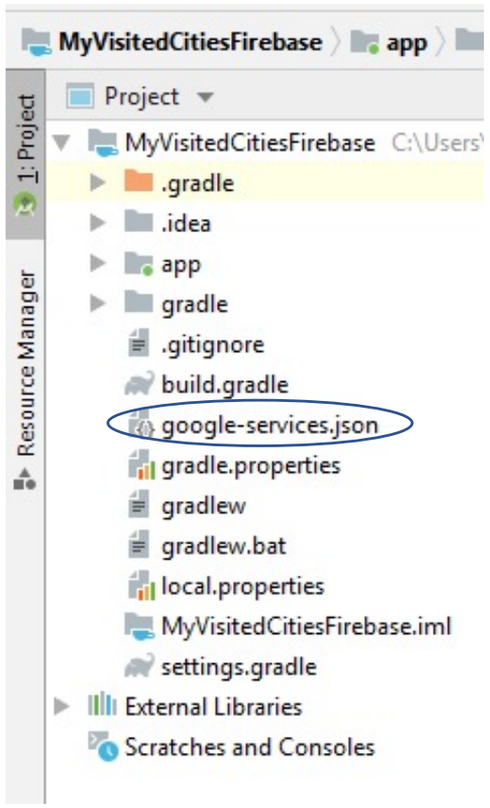
Add App to Firebase project



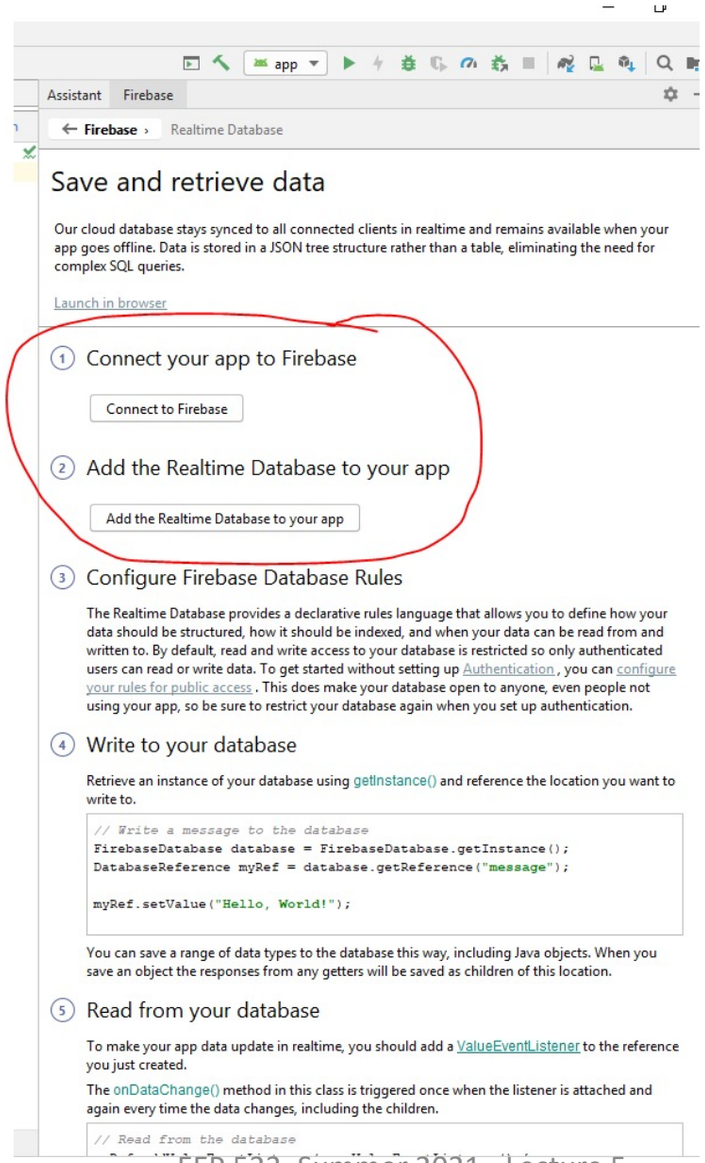


Add App to Firebase project





From AndroidStudio Assistant



Face Detection: *Contour points*

Each feature contour that ML Kit detects is represented by a fixed number of points:

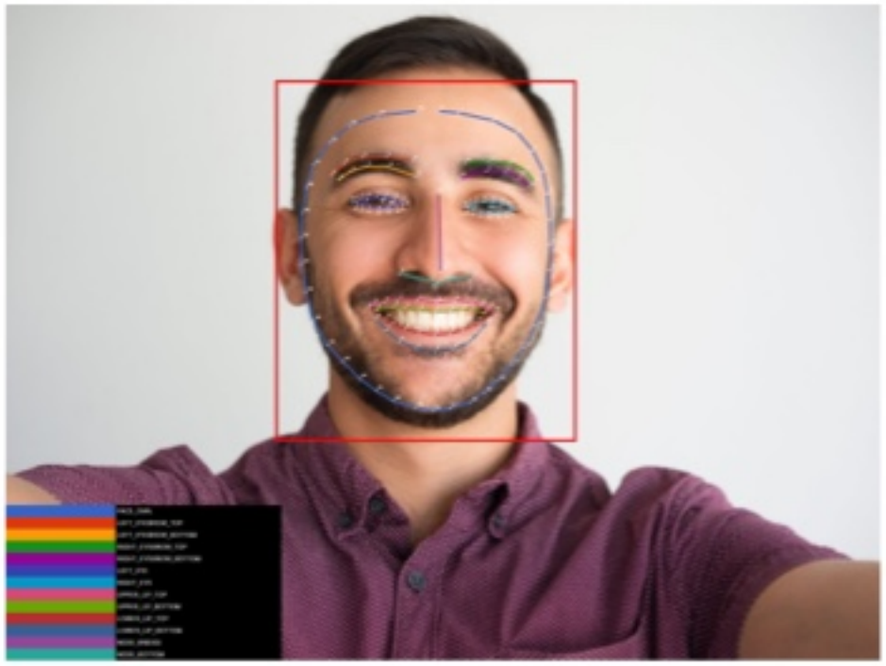
Face oval	36 points	Upper lip (top)	11 points
Left eyebrow (top)	5 points	Upper lip (bottom)	9 points
Left eyebrow (bottom)	5 points	Lower lip (top)	9 points
Right eyebrow (top)	5 points	Lower lip (bottom)	9 points
Right eyebrow (bottom)	5 points	Nose bridge	2 points
Left eye	16 points	Nose bottom	3 points
Right eye			
Left cheek (center)	1 point		
Right cheek (center)	1 points		

Indexes of feature contours	
0-35	Face oval
36-40	Left eyebrow (top)
41-45	Left eyebrow (bottom)
46-50	Right eyebrow (top)
51-55	Right eyebrow (bottom)
56-71	Left eye
72-87	Right eye
88-96	Upper lip (bottom)
97-105	Lower lip (top)
106-116	Upper lip (top)
117-125	Lower lip (bottom)
126, 127	Nose bridge
128-130	Nose bottom (note that the center point is at index 128)
131	Left cheek (center)
132	Right cheek (center)

Face Detection: *Landmarks*

A landmark is a point of interest within a face.

right eye,
left eye,
nose base,
left cheek,
right cheek,
left mouth,
right mouth,
bottom mouth



Face Detection: *Classification*

- Classification determines whether a certain facial characteristic is present.
- ML Kit currently supports two classifications: **eyes open** and **smiling**.
- Classification is expressed as a certainty value, indicating the **confidence** that the facial characteristic is present.
- For example, a value of 0.7 or more for the smiling classification indicates that it is likely that a person is smiling.
- Both of these classifications rely upon landmark detection.
- Also note that "eyes open" and "smiling" classification only works for frontal faces,

Face Detection: *Firebase MLkit*

Setting up your project to use the face detector of MLkit

1. If you haven't already, add Firebase to your Android project.

2. In your project-level build.gradle file, make sure to include Google's Maven repository in both your buildscript and allprojects sections.

3. Add the dependencies for the ML Kit Android libraries to your module (app-level) Gradle file (**app/build.gradle**)

```
apply plugin: 'com.google.gms.google-services'

dependencies {
  // ...

  implementation 'com.google.firebase:firebase-ml-vision:24.0.2'
  // If you want to detect face contours (landmark detection and classification
  // don't require this additional model):
  implementation 'com.google.firebase:firebase-ml-vision-face-model:20.0.0’

}
```

Face Detection: *Firebase MLkit*

Bitmap image

```
private fun faceDetection() {

    val image = FirebaseVisionImage.fromBitmap(mSelectedImage!!)
    val options = FirebaseVisionFaceDetectorOptions.Builder()
        .setPerformanceMode(FirebaseVisionFaceDetectorOptions.FAST)
        .setLandmarkMode(FirebaseVisionFaceDetectorOptions.ALL_LANDMARKS)
        .setContourMode(FirebaseVisionFaceDetectorOptions.ALL_CONTOURS)
        .setClassificationMode(FirebaseVisionFaceDetectorOptions.ALL_CLASSIFICATIONS)
        .build()
    val detector = FirebaseVision.getInstance().getVisionFaceDetector(options)
    detector.detectInImage(image)
        .addOnSuccessListener { faces ->
            processFaces(faces)
        }
        .addOnFailureListener { e -> // Task failed with an exception
            e.printStackTrace()
        }
}
```

Face Detection: *Firebase MLkit*

Landmarks Example: get the pixel positions (x,y) of the *face center* and eyes

```
private fun processFaces(faces: List<FirebaseVisionFace>) {
// Task completed successfully
    if (faces.size == 0) {
       return
    }

    val face = faces[0]
    val center_x = (face.boundingBox.centerX().toInt())
    val center_y = (face.boundingBox.centerY().toInt())



    var leftEyePosX = 0
    var leftEyePosY = 0
    val leftEye = face.getLandmark(FirebaseVisionFaceLandmark.LEFT_EYE)
    leftEye?.let {
          leftEyePosX = leftEye.position.x.toInt()
          leftEyePosY = leftEye.position.y.toInt()
    }

}
```

Face Detection: *Firebase MLkit*

Contour Example: get the pixel positions of the contour points

```
val contour = face.getContour(FirebaseVisionFaceContour.ALL_POINTS)
  for (point in contour.points) {
          val px = point.x
          val py = point.y }
```

int	ALL_POINTS	All points of a face contour.
int	FACE	The outline of the subject's face.
int	LEFT_EYE	The outline of the subject's left eye cavity.
int	LEFT_EYEBROW_BOTTOM	The bottom outline of the subject's left eyebrow.
int	LEFT_EYEBROW_TOP	The top outline of the subject's left eyebrow.
int	LOWER_LIP_BOTTOM	The bottom outline of the subject's lower lip.
int	LOWER_LIP_TOP	The top outline of the subject's lower lip.
int	NOSE_BOTTOM	The outline of the subject's nose bridge.
int	NOSE_BRIDGE	The outline of the subject's nose bridge.
int	RIGHT_EYE	The outline of the subject's right eye cavity.
int	RIGHT_EYEBROW_BOTTOM	The bottom outline of the subject's right eyebrow.
int	RIGHT_EYEBROW_TOP	The top outline of the subject's right eyebrow.
int	UPPER_LIP_BOTTOM	The bottom outline of the subject's upper lip.
int	UPPER_LIP_TOP	The top outline of the subject's upper lip.

Face Detection: *Firebase MLkit*

Classification Example: get the probability of smile and eyes open

```
if (face.smilingProbability != FirebaseVisionFace.UNCOMPUTED_PROBABILITY) {
    val smileProb = face.smilingProbability
}
if (face.rightEyeOpenProbability != FirebaseVisionFace.UNCOMPUTED_PROBABILITY) {
    val rightEyeOpenProb = face.rightEyeOpenProbability
}
```

Speech to Text

EE P 523, Lecture 5

Speech-to-Text

User talks; Android records, turns into a String.

- Similar to the camera, Android has a built-in activity for capturing speech into text.
- You can call it using an Intent and wait for the result.

```
fun record(){
    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH )
    intent.putExtra( RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault()    )
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
    // prompt text is shown on screen to tell user what to say
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "text")
    startActivityForResult(intent, KEY)
}
```

Receiving the Speech-to-text()

When the speech-to-text activity comes back, its `Intent` gives you all text spoken by the user in an `ArrayList`.

- Usually the first element (index 0) contains the string you want.

```
override fun onActivityResult(requestCode: Int,resultCode: Int, intent: Intent?) {
    super.onActivityResult(requestCode, resultCode, intent)
    val list = intent!!.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS  )
    val spokenText = list[0]
    val duration = Toast.LENGTH_SHORT

    val toast = Toast.makeText(applicationContext, spokenText, duration)
    toast.show()
    // ...
}
```

Robust Speech-to-text()

- Some devices do not have speech-to-text capability.
  - In these cases, it will throw an exception when you try to use it.
  - To handle such situations, you can **`try/catch`** the exception.

```
fun record(){
    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH )
    intent.putExtra( RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault()   )
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
    // prompt text is shown on screen to tell user what to say
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "text")

    try {
        startActivityForResult(intent, KEY)}
    catch (anfe: ActivityNotFoundException) {
        //code to handle exception
    }
}
```

Text to Speech

EE P 523, Lecture 5

Text-To-Speech

text-to-speech:
Allows Android device to speak an audible message based on a text string.

- Not installed by default on some devices.
  To install, click
  Settings → Language and Input → Text to speech output
  → Google textto-speech engine "settings" icon → Install voice data → Languages

- In general, text-to-speech on Android is simple:
  1. create a TextToSpeech object
  2. call its speak method

Text-To-Speech Class

TextToSpeech constructor and methods:

- TextToSpeech(***activity***, ***listener***) - constructor
- getVoice, getVoices, setVoice - change speaking voice
- getLanguage, setLanguage - sets language used
- getPitch, setPitch - sets vocal tone used
- isSpeaking - returns true if speaking
- shutdown - kills TTS engine
- speak(***text***, ***mode***, ***params***) - speaks given text aloud
- stop - halts any speech
- synthesizeToFile(***text***, ***params***, ***filename***)-speaks to file

The speak() method

```
speak(CharSequence text, int queueMode, Bundle params, String utteranceId)
```

The speak method accepts 4 parameters *:

- the text to speak aloud, as a String.
- the mode to use for speaking, one of:
  TextToSpeech.*QUEUE_ADD* : Speak after any other text is done.
  TextToSpeech.*QUEUE_FLUSH* : Stop any other text and speak immediately.
- a Map of parameters (we don't need any, so we can pass null).
- an ID of the utterance

** (Android 5 introduces a different version of speak(), but we'll ignore it.)*

```
private fun speakOut() {
    val myText1 = "Did you sleep well?"
    val myText2 = "I hope so, because it's time to wake up."
    tts.speak(myText1, TextToSpeech.QUEUE_FLUSH, null, null)
    tts.speak(myText2, TextToSpeech.QUEUE_ADD, null, null)
```

Text-to-Speech

```
class MainActivity : AppCompatActivity(),TextToSpeech.OnInitListener {
       private lateinit var  tts: TextToSpeech;

       override fun onCreate(savedInstanceState: Bundle?) {
           super.onCreate(savedInstanceState)
           setContentView(R.layout.activity_main)

           tts = TextToSpeech(this, this) }
```

```
override fun onInit(status: Int) {
    if (status == TextToSpeech.SUCCESS) {
        val result = tts!!.setLanguage(Locale.US) // set US English as language for tts
        if (result == TextToSpeech.LANG_MISSING_DATA || result == TextToSpeech.LANG_NOT_SUPPORTED)
{           Log.e("TTS","The Language specified is not supported!")
        } else {
}
    } else {
        Log.e("TTS", "Initilization Failed!")
    }

}
```

```
private fun speakOut() {
    val myText1 = "Did you sleep well?"
    val myText2 = "I hope so, because it's time to wake up."
    tts.speak(myText1, TextToSpeech.QUEUE_FLUSH, null, null)
    tts.speak(myText2, TextToSpeech.QUEUE_ADD, null, null)}
```

Tiny Deep Learning: Practical Introduction

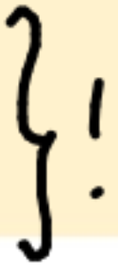
EE P 523, Lecture 5

Motivation and Goals for This Topic

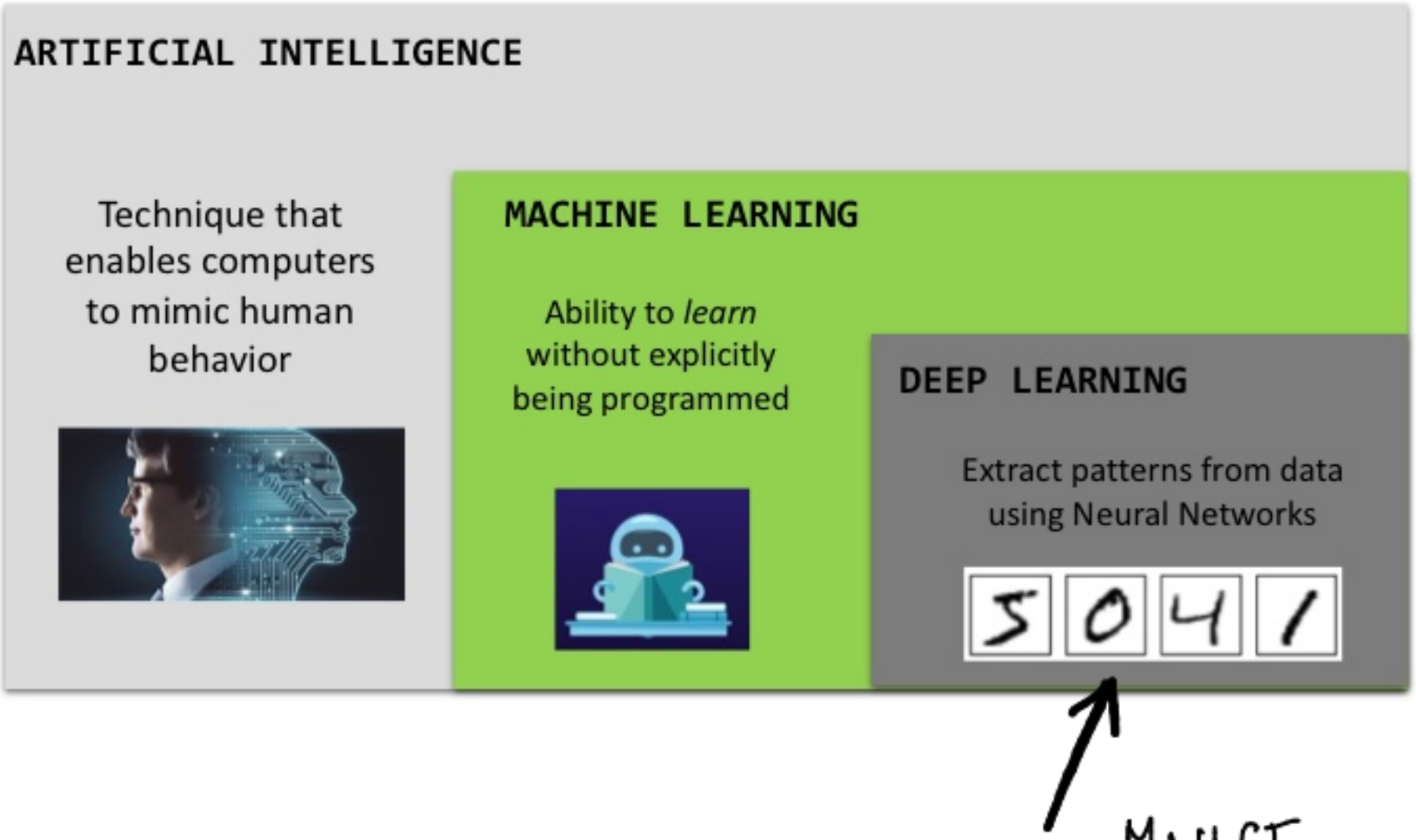
- Detect your own custom gesture (not just known patterns)
- Accurate and ~fast data classification and prediction

There are many books and courses that explain the science behind deep learning, so we won't be doing that here (encourage you to explore!)

Remember:
you don't need all the theory to start building useful things



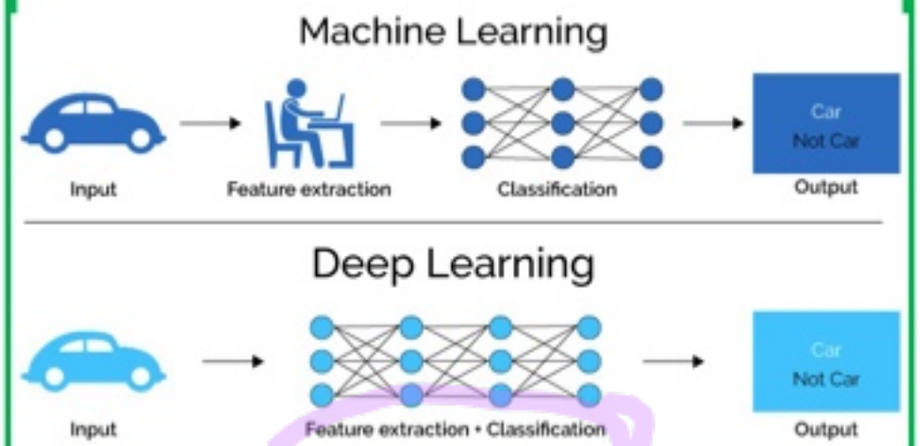
What is *AI/ML/DL*



Deep Learning

WHY ?

- **What is it**: pattern from
  Extract useful ~~patters form~~ data
- **How:**
  neural network + optimization
- **How in practice**:
  Python + TensorFlow
- **Why now**:
  data, hardware, community,
  tools, investment



WHY NOT ?

Unintended consequences

Three Classes of Learning Problems

Supervised

Data (x, y)

x: data

y: labels

Goal learn function to map x -> y

Example:

This is a digit 3

Unsupervised

Data x (no labels!)

Goal: learn underlying structure

Example:

This thing is like the other thing

Reinforcement

Data state-action pairs

Goal: maximize future rewards

Example:

Select this number because it will give make you go forward

Getting Up to Speed on Machine Learning

Creating a machine learning program is different from the usual process of writing code.

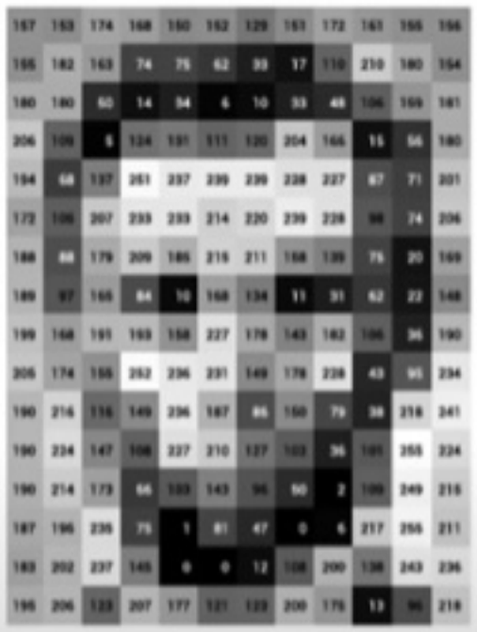
- In a traditional piece of software, a programmer designs an algorithm that takes an input, applies various rules, and returns an output.
- The algorithm's internal operations are planned out by the programmer and implemented explicitly through lines of code

- To create a machine learning program, a programmer feeds data into a special kind of algorithm and lets the algorithm discover the rules.
- This means that, as programmers, we can create programs that make predictions based on complex data without having to understand of all the complexity ourselves

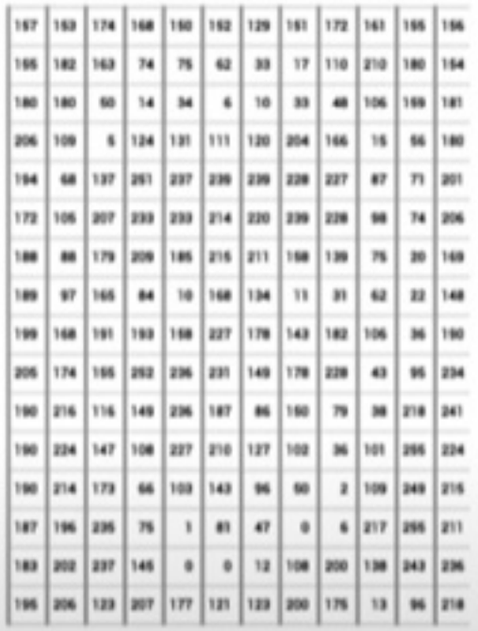
Deep Learning Workflow in 7 steps

1. Decide on a goal
2. Collect a dataset
3. Design a model architecture
4. Train the model
5. Convert the model
6. Run inference
7. Evaluate and troubleshoot

Images are Numbers

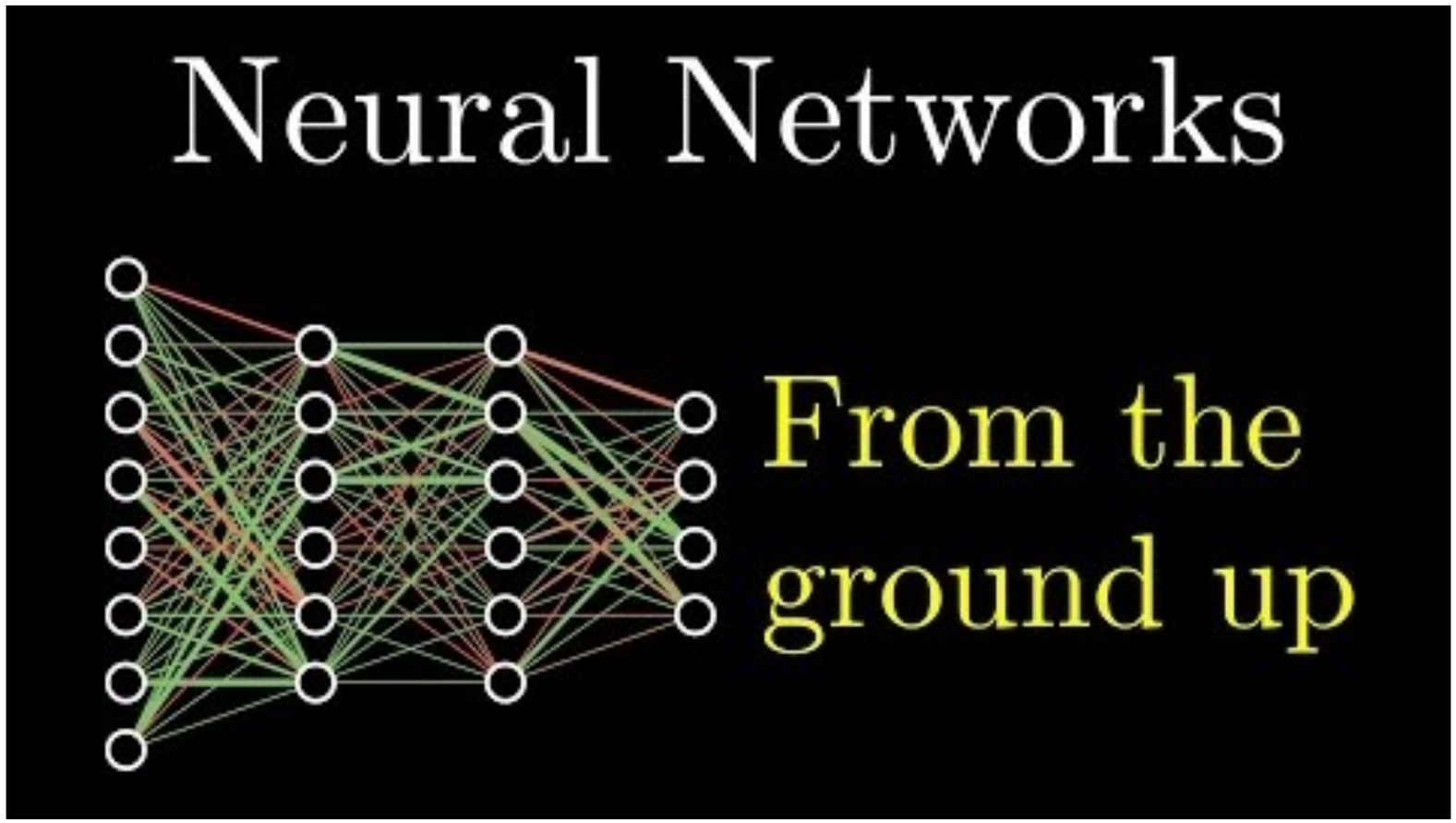


What the computer sees



An image is just a matrix of numbers [0,255]!
i.e., 1080×1080×3 for an RGB image

1. Decide on a goal



1. Decide on the goal

- We can express this as a *classification* problem.
- Classification is a machine learning task that takes a set of input data and returns the probability that this data fits each of a set of known *classes*.

2. Collect Dataset

1. Select data

It's really important that the data you choose will also be available when you want to make predictions

2. Collect data

It's difficult to know exactly how much data is required to train an effective model. It depends on many factors, such as the complexity of the relationships between variables, the amount of noise, and the ease with which classes can be distinguished. However, there's a rule of thumb that is always true: the more data, the better!

3. Label data

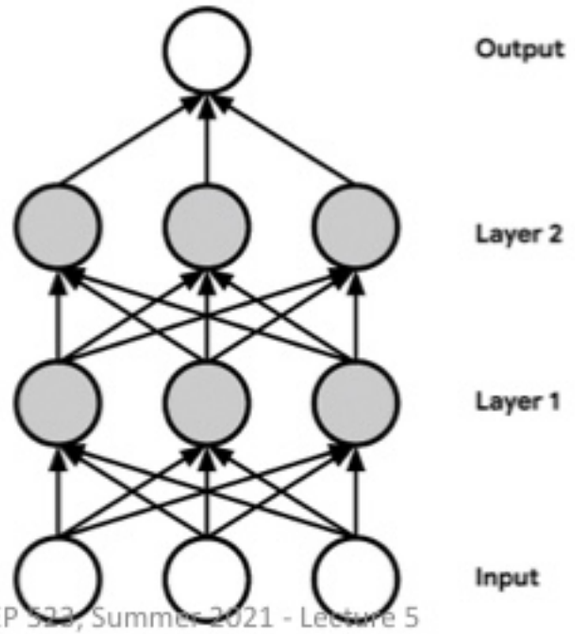
The process of associating data with classes is called labeling, and the "normal" and "abnormal" classes are our labels.

3. Design a Model Architecture

Model

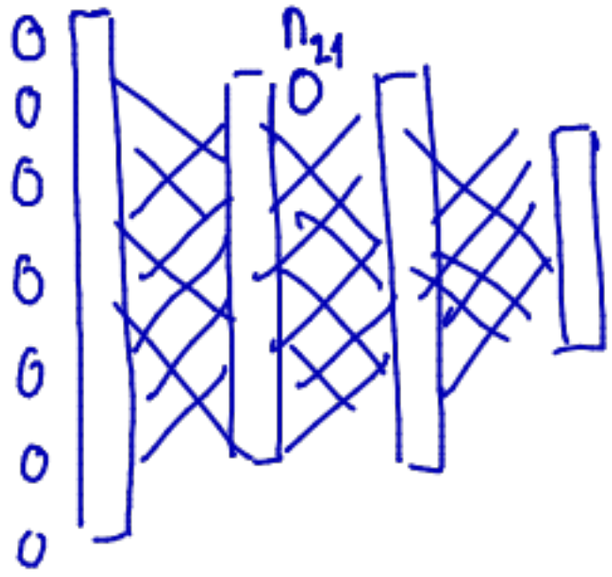
- network of simulated neurons represented by arrays of numbers arranged in layers.
- These numbers are known as ***weights*** and ***biases***, or collectively as the network's *parameters*.

When data is fed into the network, it is transformed by successive mathematical operations that involve the weights and biases in each layer. The output of the model is the result of running the input through these operations.

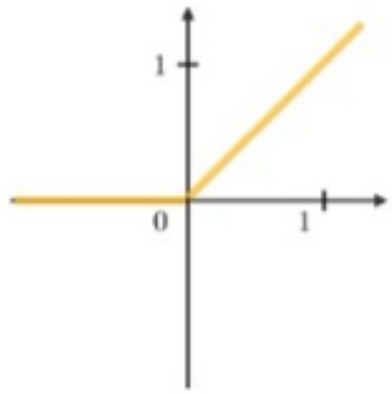
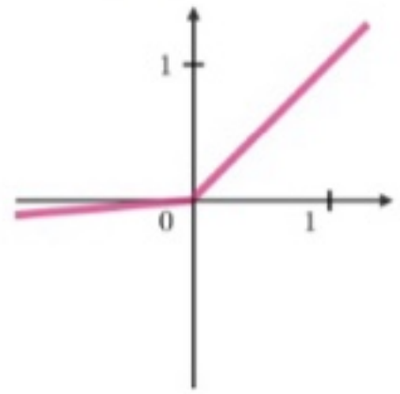
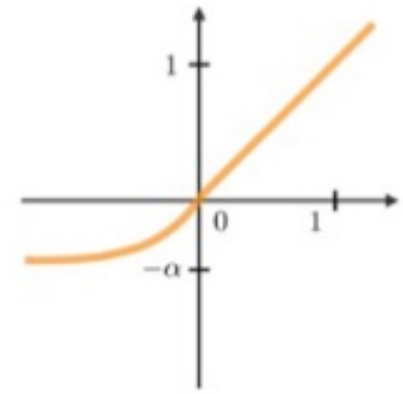


Single Neuron (Perceptron)

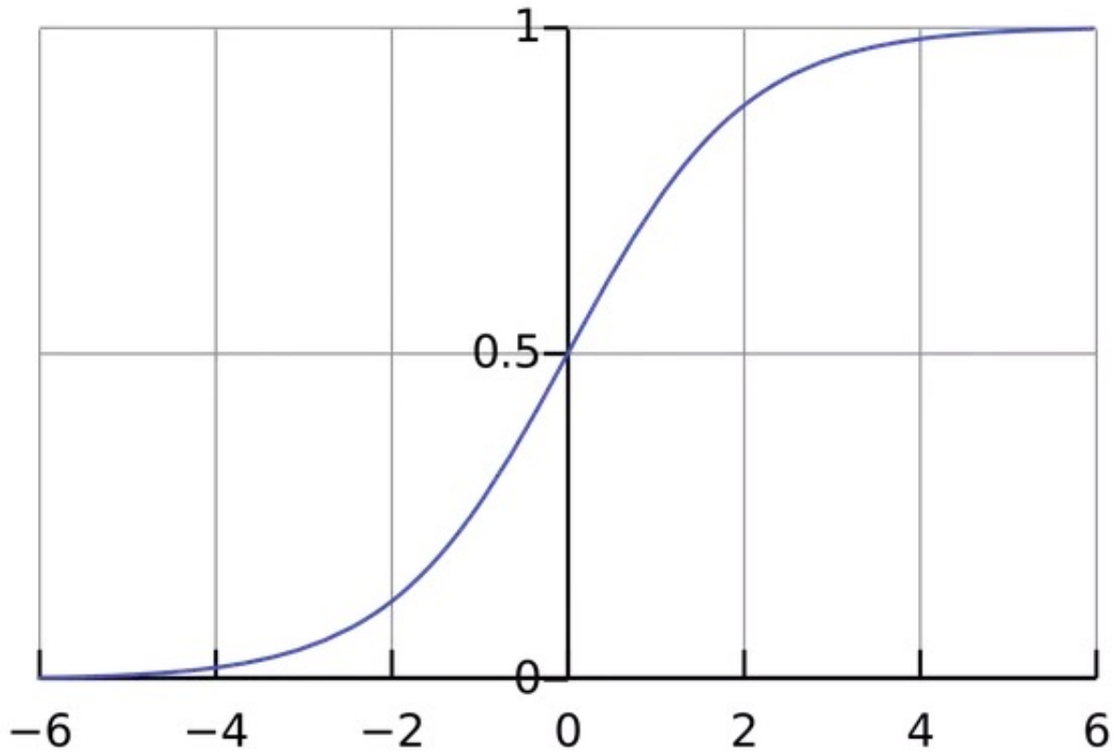
$n_{21} = f(n_{1i}, w)$



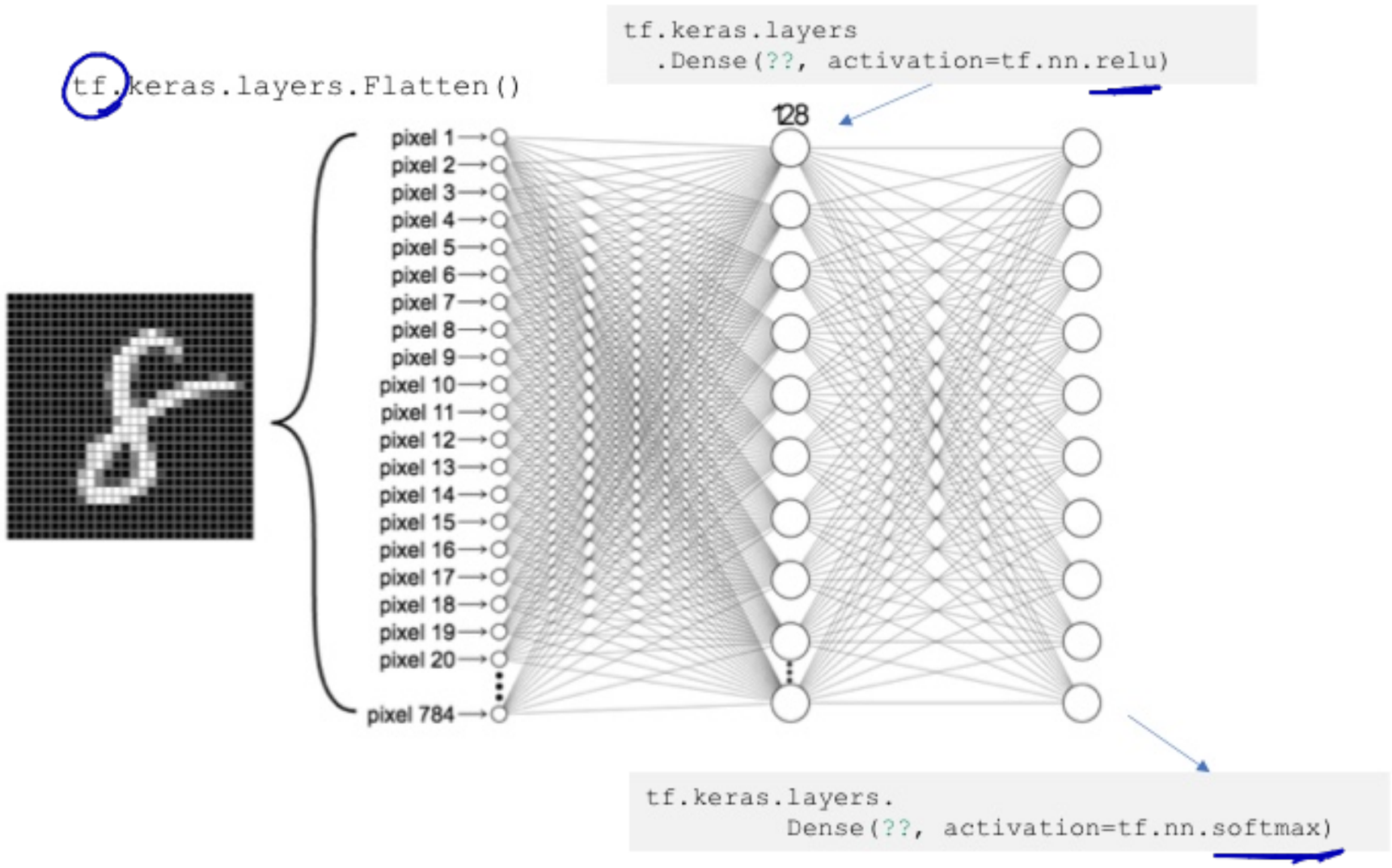
Activation Function: ReLU

ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
• Non-linearity complexities biologically interpretable	• Addresses dying ReLU issue for negative values	• Differentiable everywhere

Activation Function: SoftMax



Our model


tf.keras.layers.Flatten()
tf.keras.layers
.Dense(??, activation=tf.nn.relu)
128
pixel 1
pixel 2
pixel 3
pixel 4
pixel 5
pixel 6
pixel 7
pixel 8
pixel 9
pixel 10
pixel 11
pixel 12
pixel 13
pixel 14
pixel 15
pixel 16
pixel 17
pixel 18
pixel 19
pixel 20
pixel 784
tf.keras.layers.
Dense(??, activation=tf.nn.softmax)

3. Design a Model Architecture

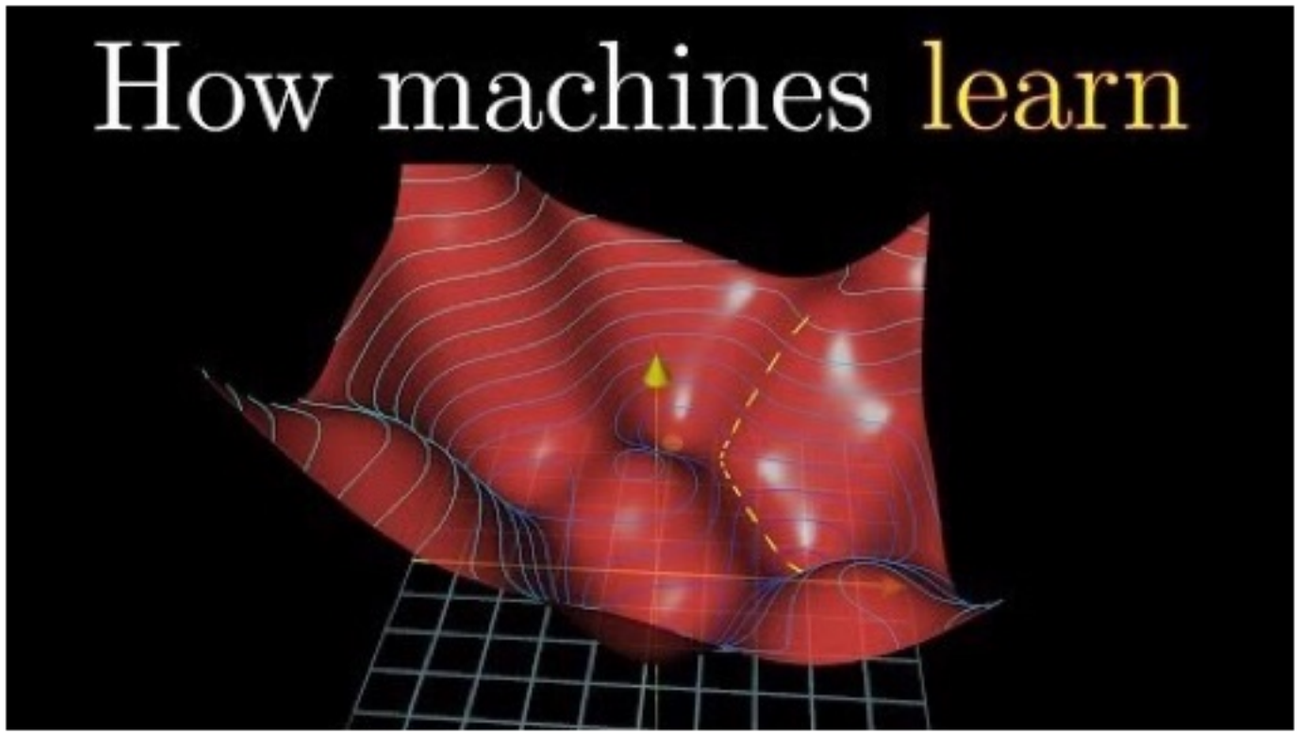
- For many common problems, you can find pretrained models available online for free.

- Deep learning models accept input and generate output in the form of tensors.
- For the purposes of this course, a tensor is essentially a list that can contain either numbers or other tensors; you can think of it as similar to an array.

4. Train the model

- The model's weights start out with random values, and biases typically start with a value of 0.
- During training, batches of data are fed into the model, and the model's output is compared with the desired output (which in our case is the correct label, "normal" or "abnormal").
- An algorithm called *backpropagation* adjusts the weights and biases incrementally so that over time, the output of the model gets closer to matching the desired value.
- Training, which is measured in epochs (meaning iterations), continues until we decide to stop.

4. How our Models Learn?



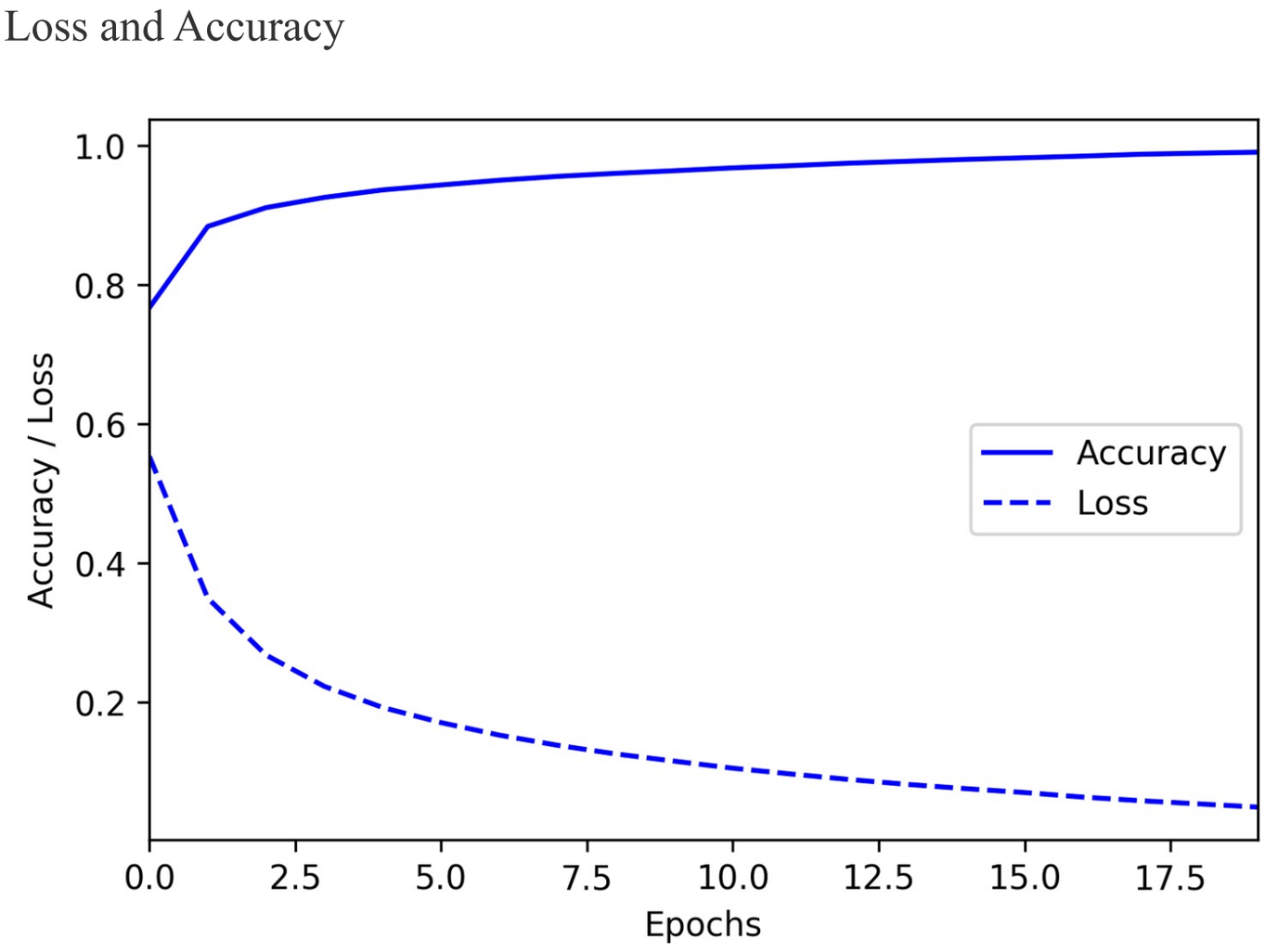
4. Train the Model: When to Stop?

- We generally stop training when a model's performance stops improving. It is said to have ***converged***.

- Common performance metrics are ***loss*** and ***accuracy***.
  - **Loss**: numerical estimate of how far the model is from producing the expected answers
  - **Accuracy**: percentage of the time that it chooses the correct prediction.

- A perfect model would have a loss of 0.0 and an accuracy of 100%, but real models are rarely perfect.

4. Train the Model:



4. Train the Model: Hyperparameters

- To attempt to improve the model's performance, we can change our model architecture, and we can adjust various values used to set up the model and moderate the training process.

- These values are collectively known as *hyperparameters*, and they include variables such as the number of training **epochs** to run and the number of neurons in each layer.

- Each time we make a change, we can retrain the model, look at the metrics, and decide whether to optimize further.

- Hopefully, time and iterations will result in a model with acceptable accuracy!

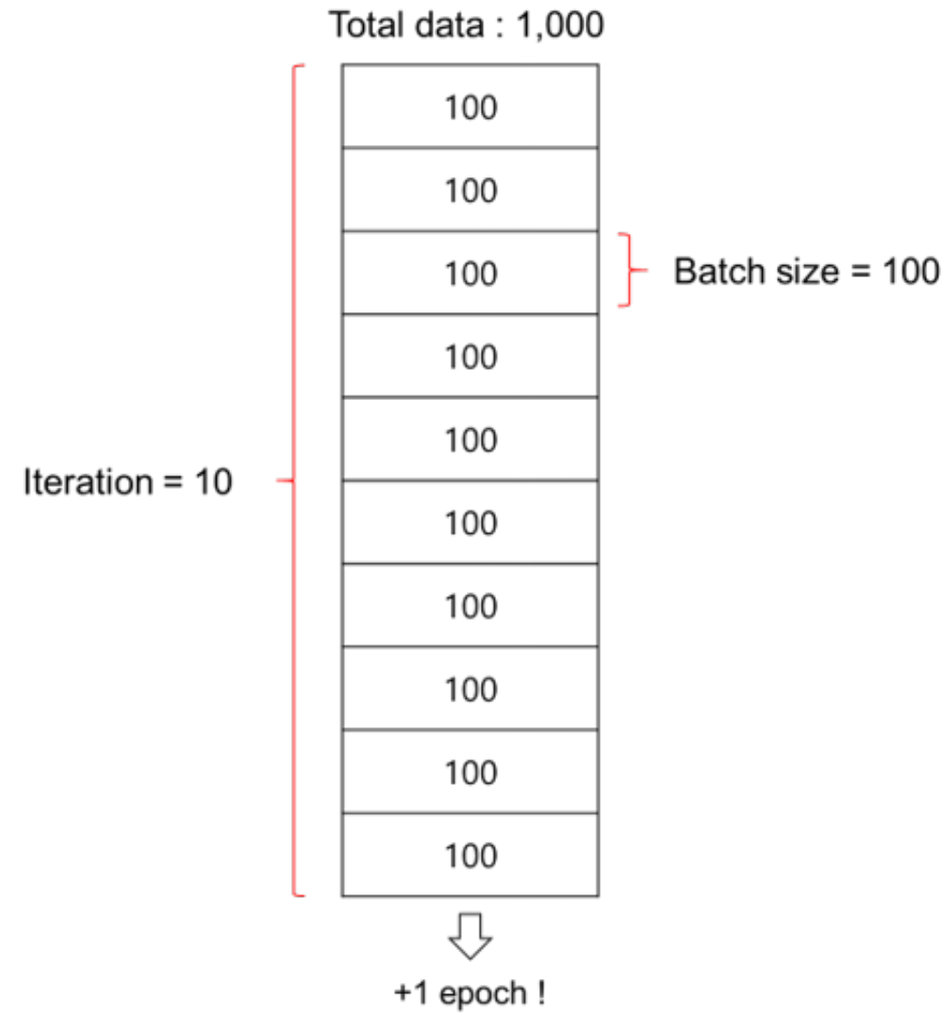
4. Train the Model: Hyperparameters

- **Batch size**

how many pieces of training data to feed into the network before measuring its accuracy and updating its weights and biases

- **Epochs**

how many times our entire training set will be run through the network during training. The more epochs, the more training will occur.



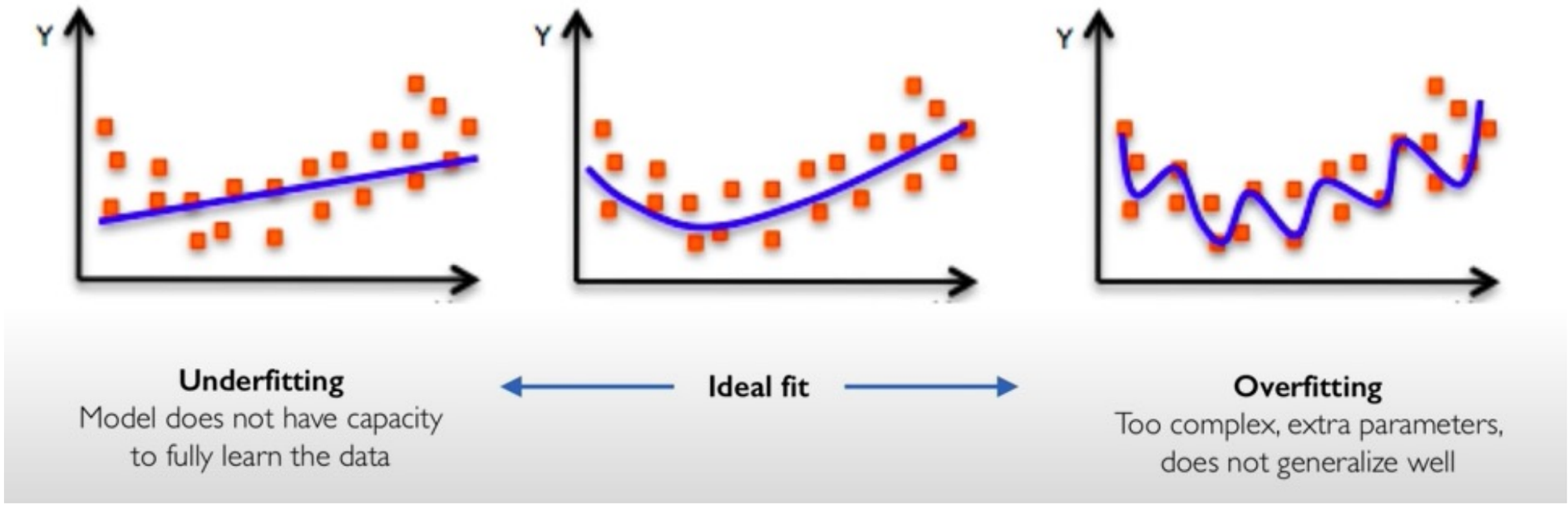
4. Train the Model: Why the Model Fails?

A neural network learns to *fit* its behavior to the patterns it recognizes in data

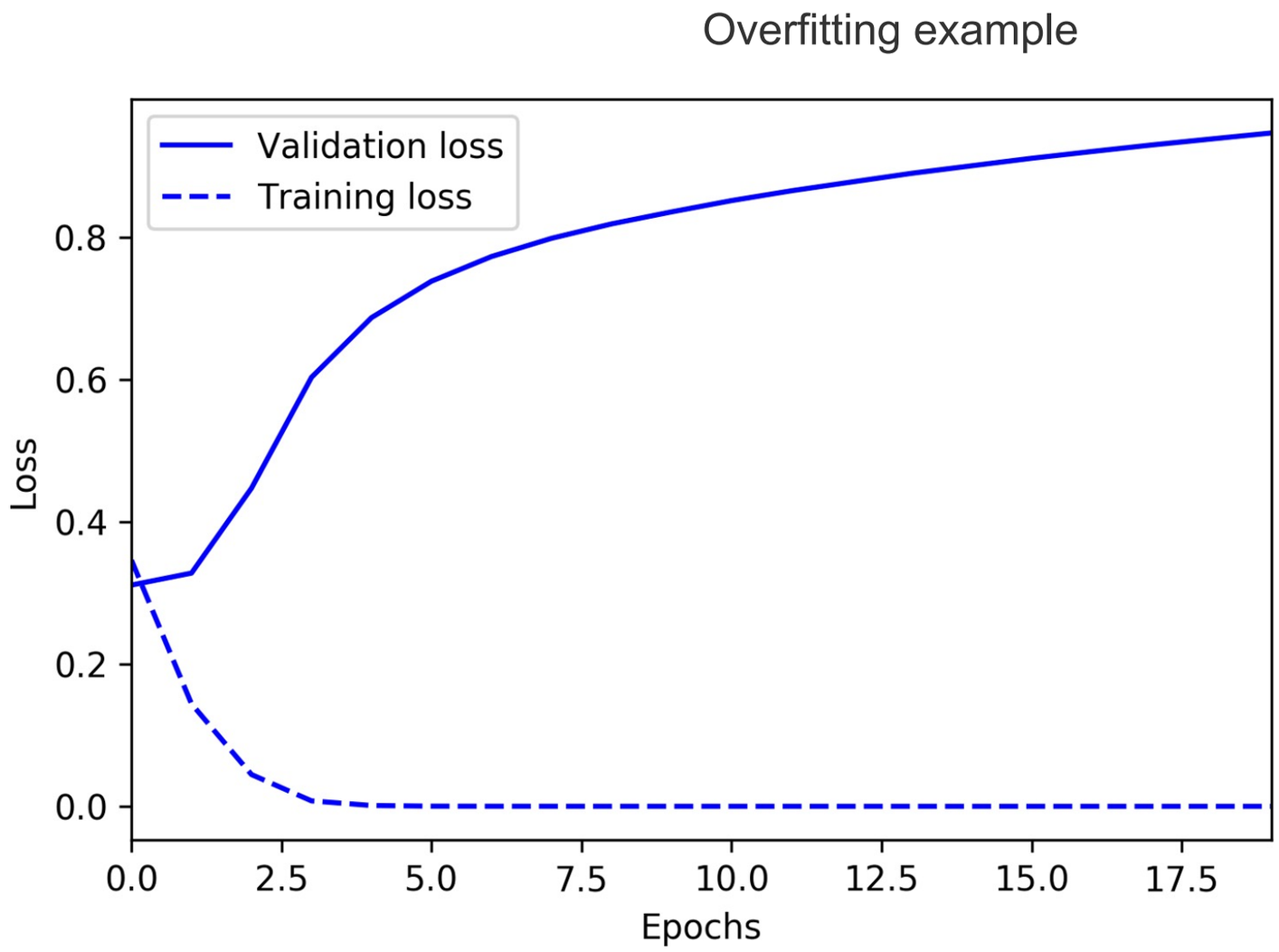
- ***Underfit***
  model has not yet been able to learn a strong enough representation of these patterns to be able to make good predictions.
  - X The architecture is too small to capture the complexity of the system it is supposed to model
  - X The model has not been trained on enough data.

- ***Overfit***
  it has learned its training data too well. The model is able to exactly predict the minutiae of its training data, but it is not able to generalize its learning to data it has not previously seen.
  - X the model has managed to entirely memorize the training data
  - X it has learned to rely on a shortcut present in the training data but not in the real world.

4. Train the model: Why the Model Fails?



4. Train the Model: Why the Model Fails?



4. Train the model: Preventing the model from failing

Goal: make deep learning models **less likely to overfit their training data**.

Regularization

- generally involve constraining the model in some way in order to prevent it from perfectly memorizing the data that it's fed during training.
- Example: *dropout* -> randomly cutting the connections between neurons during training.

Data augmentation,

- artificially expand the size of a training dataset.
- creating multiple additional versions of every training input, each transformed in a way that preserves its meaning but varies its exact composition

4. Train the model: train-validate-test

- need to *validate* the model using new data that wasn't used in training. It's common to split a dataset into three parts—***training*, *validation*, and *test*.**

- A typical split is **60% training data, 20% validation, and 20% test**. This splitting must be done so that each part contains the same distribution of information, and in a way that preserves the structure of the data.

1) During training, the *training* dataset is used to train the model.

2) Periodically, data from the *validation* dataset is fed through the model, and the loss is calculated. Because the model has not seen this data before, its loss score is a more reliable measure of how the model is performing.

3) By comparing the training and validation loss (and accuracy, or whichever other metrics are available) over time, you can see whether the model is overfitting.

Deploying machine learning models on mobile and IoT devices

TensorFlow

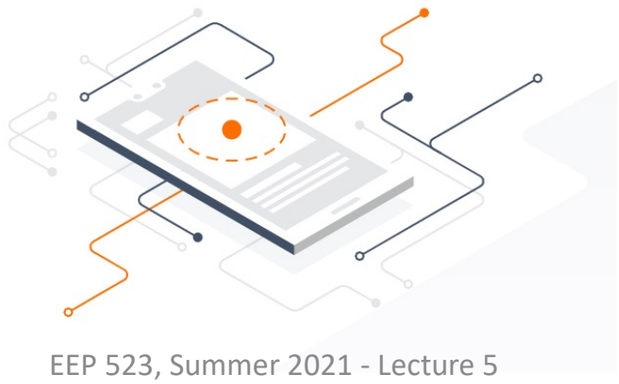
- a set of tools for building, training, evaluating, and deploying machine learning models.
- Originally developed at Google - now an open source project
- most popular and widely used framework for machine learning.

TensorFlow Lite

- set of tools for deploying TensorFlow models to mobile and embedded devices

Keras

- TensorFlow's high-level API that makes it easy to build and train deep learning networks. We'll also use.



5. Convert the model: Lite model

TensorFlow model:

- set of instructions that tell an interpreter how to transform data in order to produce an output.
- When we want to use our model, we just load it into memory and execute it using the **TensorFlow interpreter**.
- TensorFlow's interpreter is designed to run models on powerful desktop computers and servers
- TensorFlow Lite Converter. The converter can also apply special optimizations aimed at reducing the size of the model and helping it run faster, often without sacrificing performance.

6. Run inference

```
private fun classifyDrawing() {
      val bitmap = drawView?.getBitmap()

      if ((bitmap != null) && (digitClassifier.isInitialized)) {
         digitClassifier
               .classifyAsync(bitmap)
               .addOnSuccessListener { resultText -> predictedTextView?.text = resultText }
               .addOnFailureListener { e ->
                  predictedTextView?.text = "classification error"
                  Log.e(TAG, "Error classifying drawing.", e)
               }
      }
   }
```

6. Run inference

STEPS TO USE THE TFLITE MODEL IN OUR APP

1. Load the TFLITE model from the asset folder – `loadModelFile()`

1. Initialize the interpreter - `initializeInterpreter()`

2. Prepare the data input in a byte buffer array

3. Run inference: interpreter.run(input, output)

See provided app for function implementation details

TensorFlow Lite

EE P 523, Lecture 5

TensorFlow *Lite*

TensorFlow: end-to-end open-source platform for machine learning.

TensorFlow Lite: lightweight solution for mobile and embedded devices.

- It enables on-device machine learning inference with low latency and a small binary size.

- TensorFlow Lite also supports hardware acceleration with the Android Neural Networks API.

TensorFlow *Lite*

Why do we need a new mobile-specific library?

The next wave of machine learning applications will have significant processing on mobile and embedded devices.

- Machine Learning is changing the computing paradigm.
- Consumer expectations are also trending toward natural, human-like interactions with their devices, driven by the camera and voice interaction models.

TensorFlow *Lite*

Why do we need a new mobile-specific library?

- Innovation at the *silicon layer* is enabling new possibilities for hardware acceleration, and frameworks such as the Android Neural Networks API make it easy to leverage these.

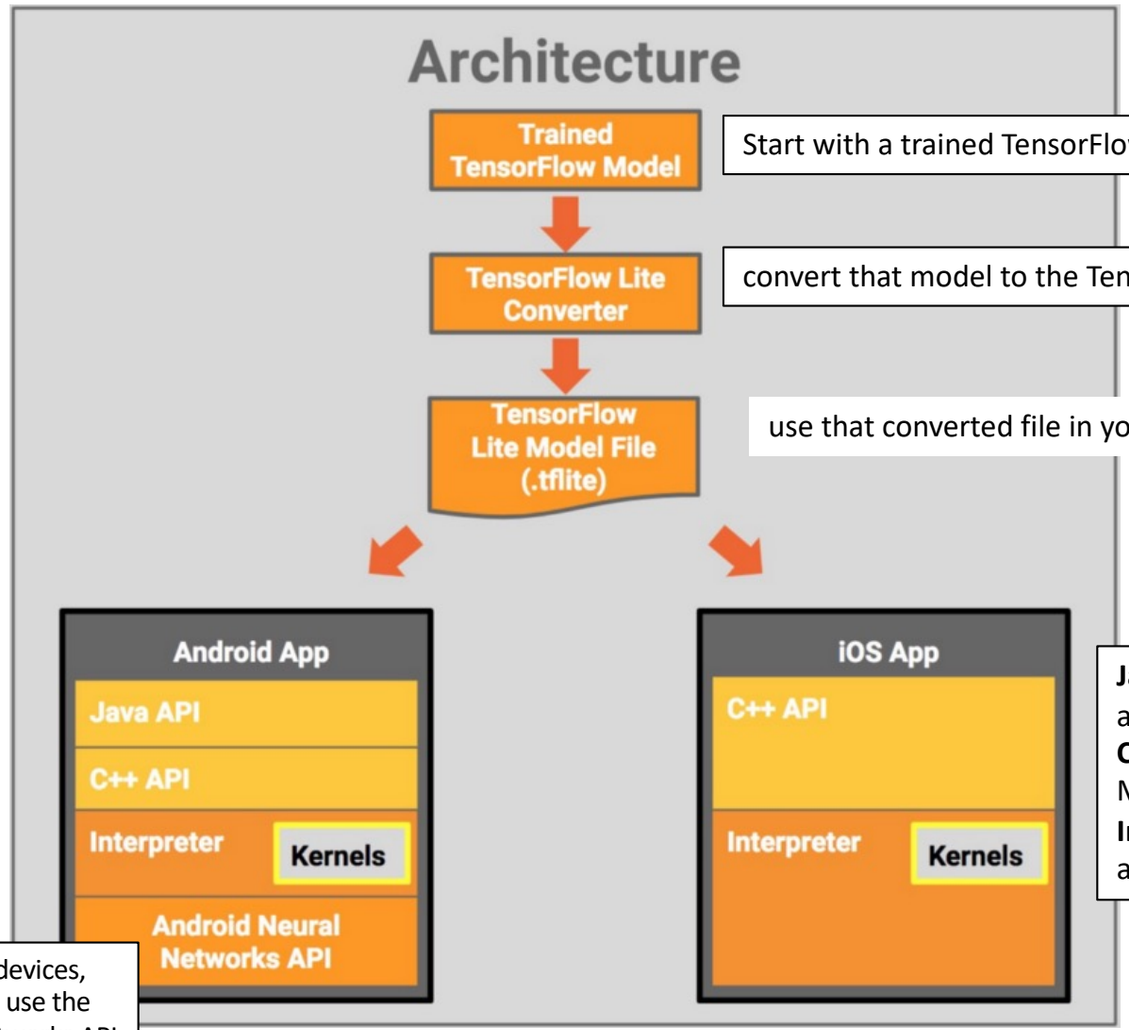
- Recent advances in real-time computer-vision and spoken language understanding have led to mobile-optimized benchmark models being open sourced (e.g., MobileNets, SqueezeNet).

- Widely-available smart appliances create new possibilities for on-device intelligence.

- Interest in stronger user data privacy paradigms where user data does not need to leave the mobile device.

- Ability to serve 'offline' use cases, where the device does not need to be connected to a network.

TensorFlow *Lite*



Start with a trained TensorFlow model on disk

convert that model to the TensorFlow Lite file format (.tflite)

use that converted file in your mobile application

Java API: A convenience wrapper around the C++ API on Android.
C++ API: Loads the TensorFlow Lite Model File and invokes the Interpreter
Interpreter: Executes the model using a set of kernels

On select Android devices, the Interpreter will use the Android Neural Networks API for hardware acceleration, or default to CPU execution if none are available.

Why is it Called TensorFlow?

- TensorFlow is called 'TensorFlow' because it handles the flow (node/mathematical operation) of tensors, which are data structures that you can think of as multi-dimensional arrays.

- Tensors are represented as n-dimensional arrays of base datatypes, such as a string or integer -- they provide a way to generalize vectors and matrices to higher dimensions.

- The ```shape``` of a Tensor defines its number of dimensions, and the size of each dimension.

- The ```rank``` of a Tensor provides the number of dimensions (n-dimensions) -- you can also think of this as the Tensor's order or degree.Let's first look at 0-d Tensors, of which a scalar is an example:

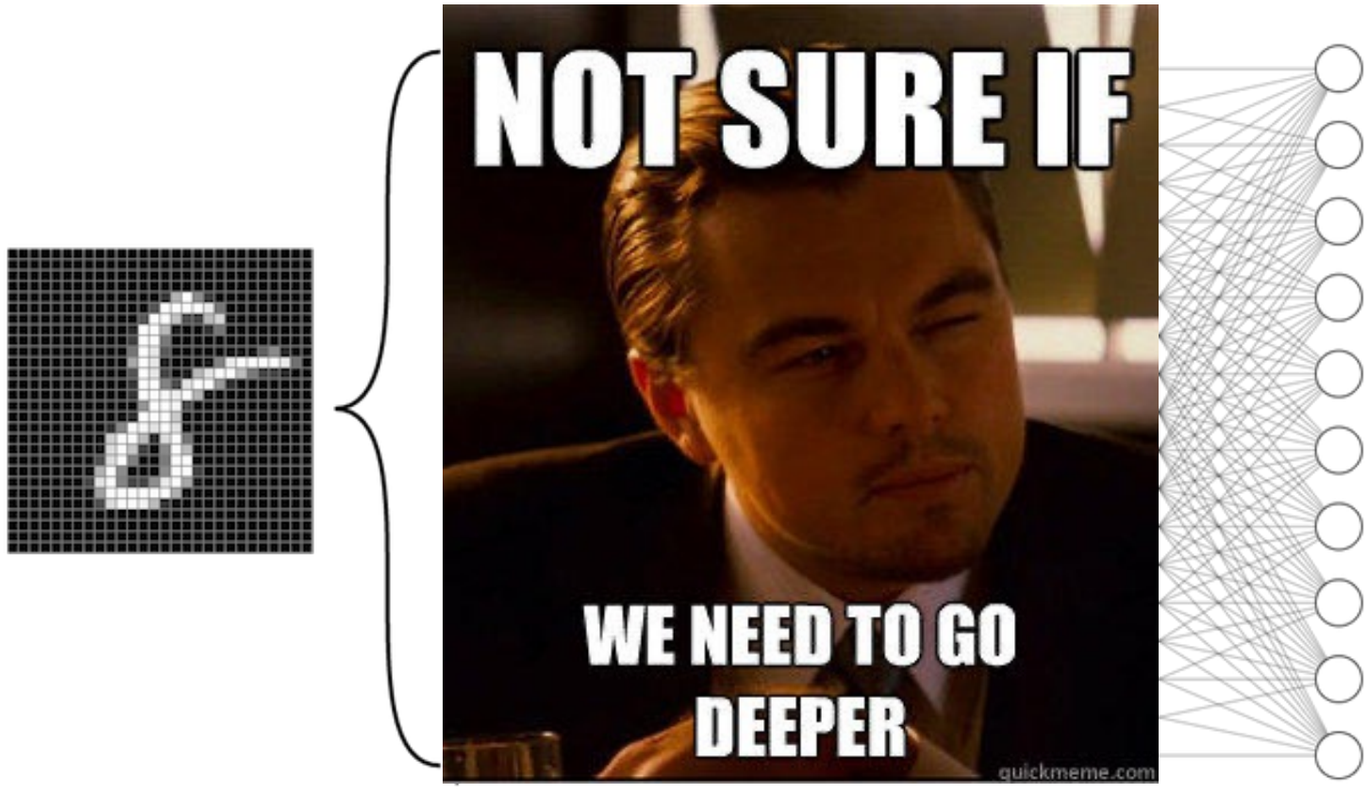
6. Run inference

- Since this is the part where our model meets our application code
- we need to write some code **that takes raw input data from our sensors and transforms it into the same form that our model was trained on**.
- We then pass this transformed data into our model and run inference.

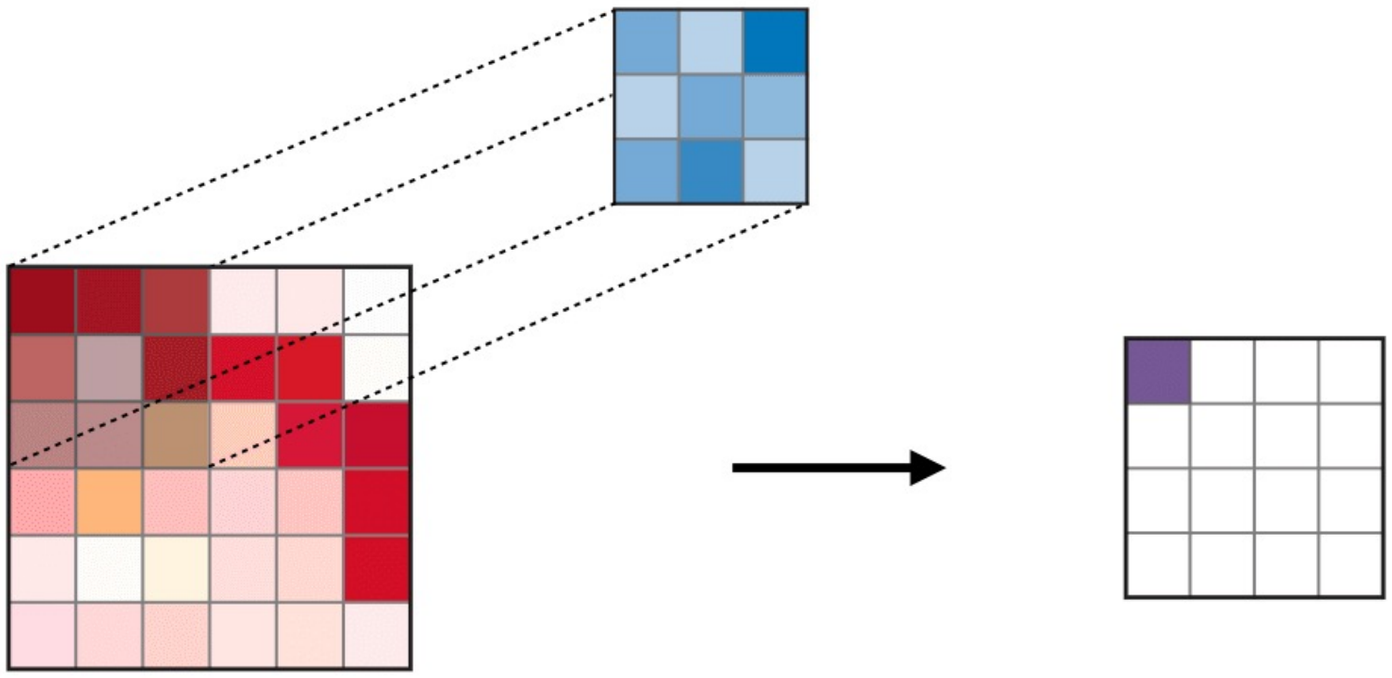
Improving Our Model

EE P 523, Lecture 5

How Do We improve the model?



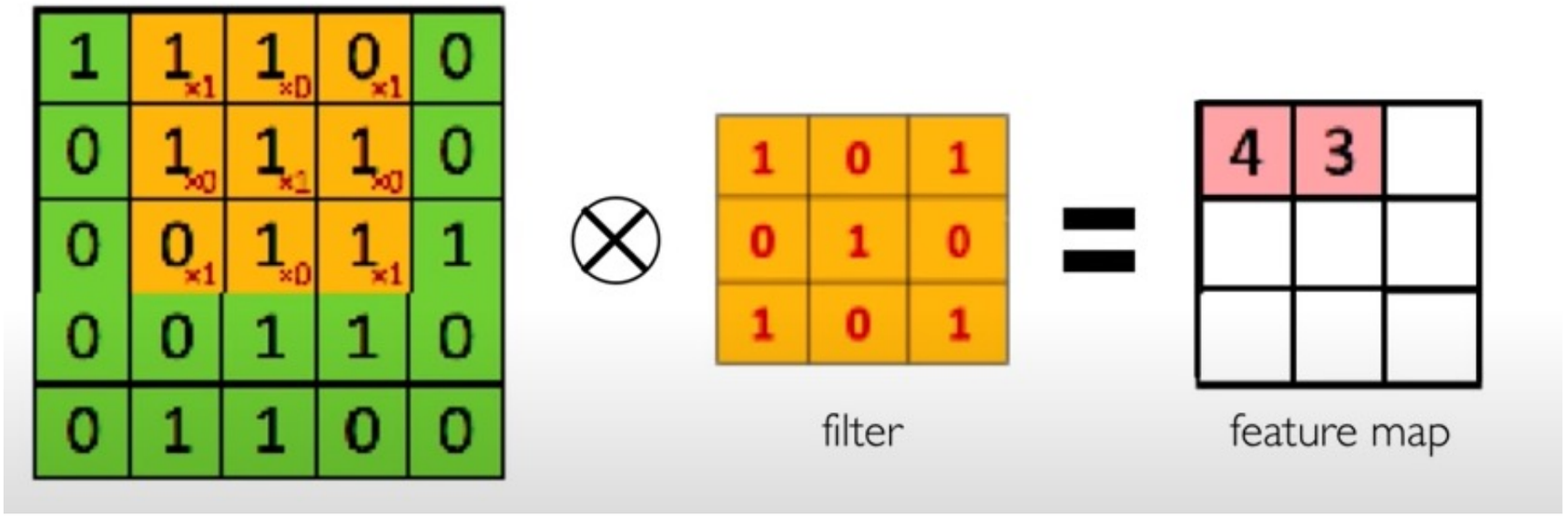
Convolution Operation



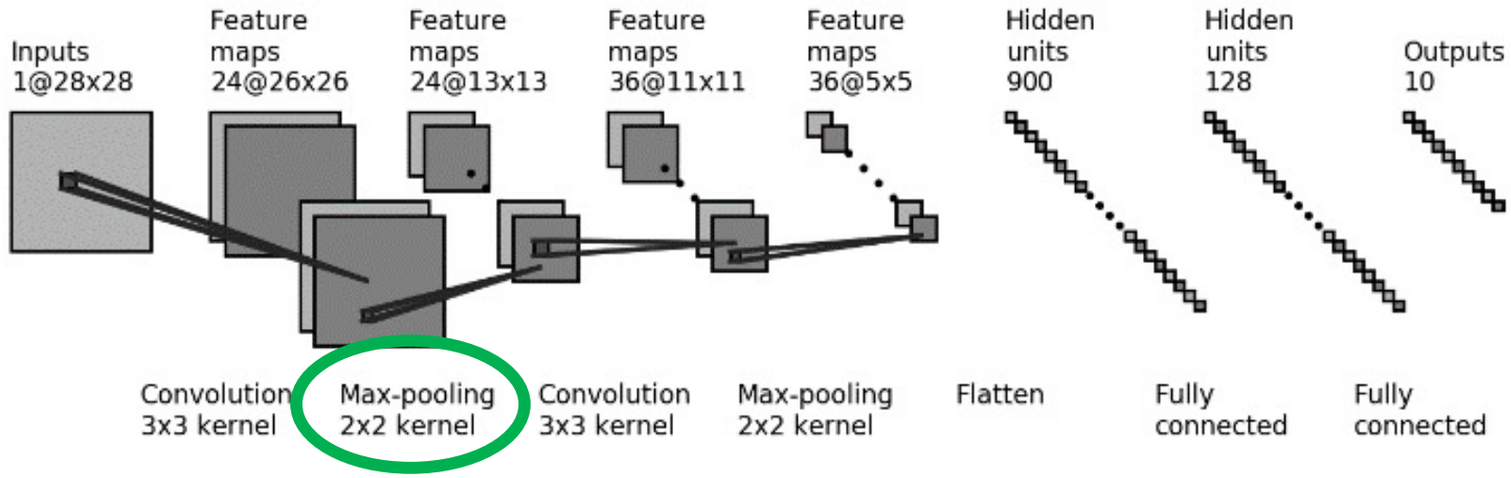
- Convolutional layers are used for **spotting 2D patterns in input images**.
- Each filter is a rectangular array of values that is moved as a sliding window across the input, and the output image is a representation of how closely

Convolution Operatoin

1)Slide a 3x3 filter 2)Element-wise multiply 3)Add the numbers



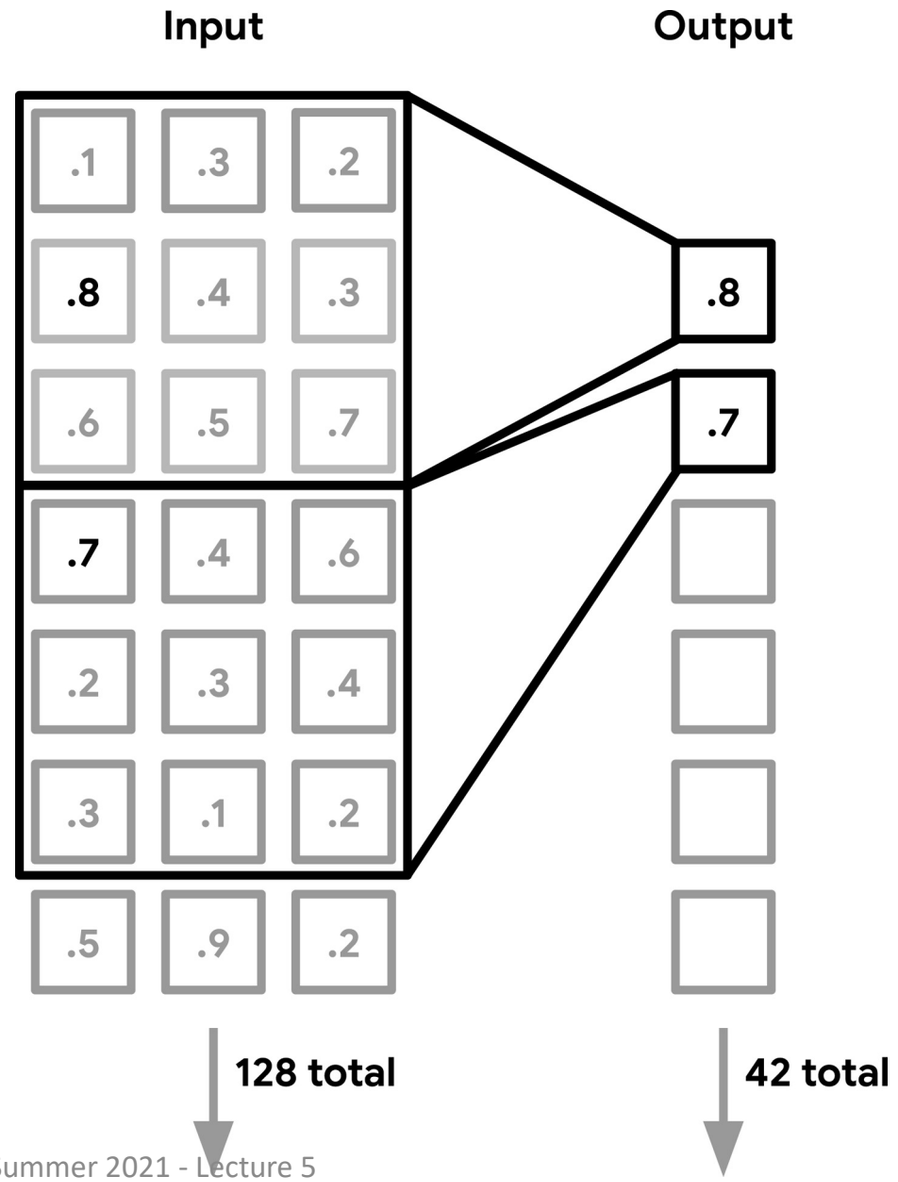
Convolutional Neural Network for digits recognition



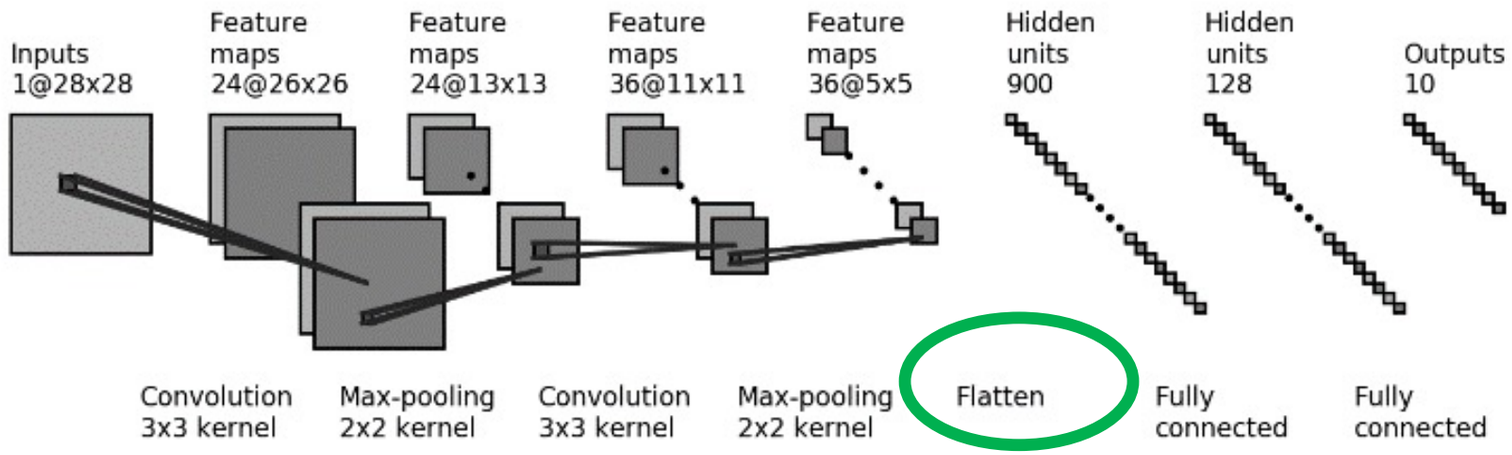
Max-pooling

- The pooling layer (POOL) is a **downsampling operation**, typically applied after a convolution layer, which does some spatial invariance.

- In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively

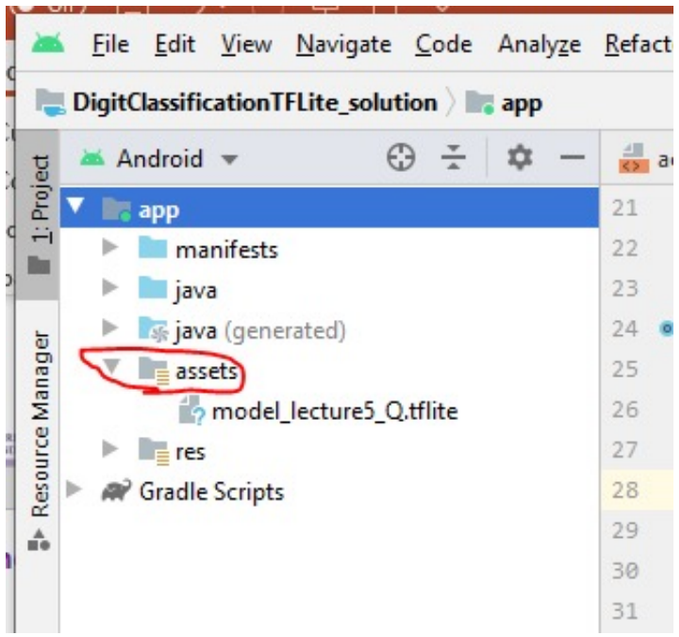


Convolutional Neural Network for digits recognition



- The **Flatten layer** is used to **transform a multidimensional tensor into one with a single dimension**.
- In this case, our (36, 5, 5) tensor is squished down into a single dimension with shape (900).

Include the model into our Android App



Include in the build.gradle(App)

```
android{

aaptOptions {
    noCompress "tflite"
}

}
```

```
dependencies {

    // Task API - required only if you are running inference in a background task
    implementation "com.google.android.gms:play-services-tasks:17.0.0"

    // TF Lite
    implementation 'org.tensorflow:tensorflow-lite:0.0.0-nightly'
}
```

HOMEWORK HINTS

Assignment: Face Detection

In your main activity:

1. Create the Face Detector

```
val detector = FaceDetector.Builder(applicationContext)
    .setTrackingEnabled(false)
    .setLandmarkType(FaceDetector.ALL_LANDMARKS)
    .setClassificationType(FaceDetector.ALL_CLASSIFICATIONS)
    .build()
```

2. Create a frame from the bitmap and run the face detection on the frame

```
val frame = Frame.Builder().setBitmap(bitmap).build()
val faces = detector.detect(frame)
```

Assignment: Face Detection

In you View Class

- Receive the faces and the bitmap from the main activity
- Process the bitmap based on the landmarks
- Pain the result on a Canvas

```
class myViewClass(context: Context, attrs: AttributeSet) : View(context, attrs)
{

     private var mBitmap: Bitmap? = null
     private var mFaces: SparseArray<Face>? = null

           . . . . .
```

Assignment: Face Detection

In you View Class

- Scale the Bitmap to the size of the Canvas

```
val viewWidth = canvas.width.toDouble()
val viewHeight = canvas.height.toDouble()
val imageWidth = mBitmap!!.width.toDouble()
val imageHeight = mBitmap!!.height.toDouble()
val scale = Math.min(viewWidth / imageWidth, viewHeight / imageHeight)
```

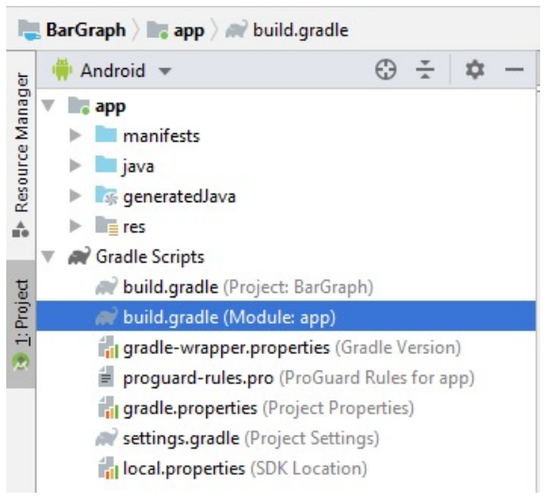
Assignment: Face Detection

In you View Class

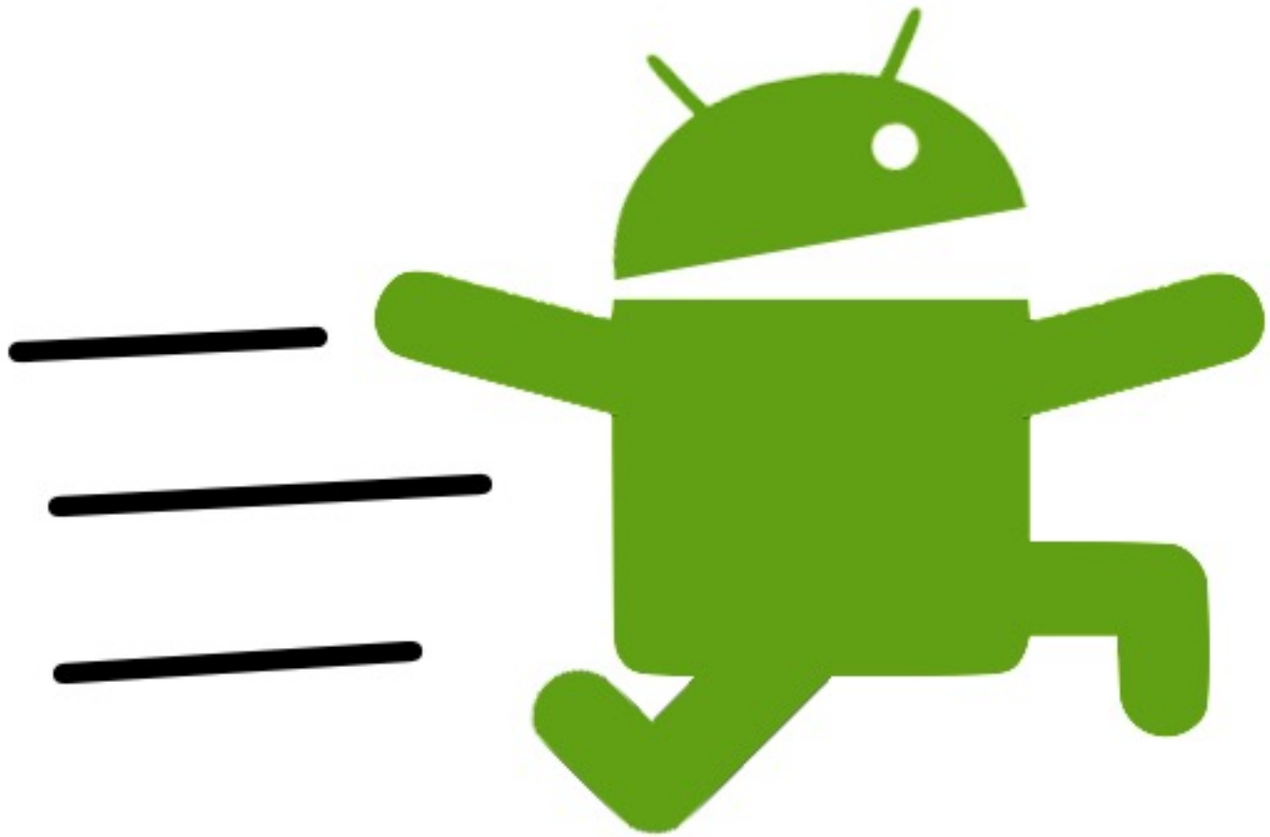
- Iterate over the faces, and recover the landmarks

```
for (i in 0 until mFaces!!.size()) {
    val face = mFaces!!.valueAt(i)
    var smileProb = face.isSmilingProbability
    for (landmark in face.landmarks) {
        val cx = (landmark.position.x * scale).toInt()
        val cy = (landmark.position.y * scale).toInt()
        //Example of plotting a circle in the landmark positions
        canvas.drawCircle(cx.toFloat(), cy.toFloat(), 10f, paint)

    }
}
```

Assignment: Face Detection




```
dependencies {
     . . . . .
    implementation 'com.google.android.gms:play-services-vision:9.4.0’
          . . . . .
}
```



Your Questions

?