1. 🔧 **Cómo se define el modelo** (capas)
2. ⚙️ **Qué hace el compilador**
3. 📚 **Cómo se entrena**
4. 💾 **Cómo se guarda y cómo descargarlo**

**🧠 1. CAPAS DEL MODELO (model.add(...))**

Aquí definimos una red neuronal **convolucional (CNN)**. Cada .add(...) agrega una "capa" que transforma los datos de entrada hasta producir una salida útil (en este caso, una clasificación de fruta).

js

CopiarEditar

const model = tf.sequential();

Esto crea un modelo secuencial: las capas se conectan una tras otra.

**🔹 Conv2D: detección de características (bordes, formas)**

js

CopiarEditar

model.add(tf.layers.conv2d({

inputShape: [64, 64, 3], // imagen RGB de 64x64 píxeles

filters: 16, // cantidad de filtros detectores

kernelSize: 3, // tamaño del filtro 3x3

activation: 'relu' // activa solo señales fuertes

}));

Esta capa examina pequeñas regiones (3x3 píxeles) de la imagen buscando patrones visuales.

**🔹 MaxPooling2D: reducción de tamaño**

js

CopiarEditar

model.add(tf.layers.maxPooling2d({ poolSize: 2 }));

Reduce la resolución (metiendo menos datos al sistema), manteniendo lo más importante.

**🔹 Flatten: aplanamiento para pasar a capas densas**

js

CopiarEditar

model.add(tf.layers.flatten());

Convierte la imagen "aplanada" en un vector largo para enviar a una red neuronal tradicional.

**🔹 Dense oculta: aprendizaje de patrones complejos**

js

CopiarEditar

model.add(tf.layers.dense({ units: 64, activation: 'relu' }));

Capa de 64 neuronas. Aprende combinaciones no lineales de las características visuales.

**🔹 Dense final: salida con tantas neuronas como clases**

js

CopiarEditar

model.add(tf.layers.dense({ units: etiquetas.length, activation: 'softmax' }));

Produce una **probabilidad para cada clase**. Ejemplo: [0.05, 0.02, 0.93] → “pera”

**🛠️ 2. COMPILACIÓN DEL MODELO**

js

CopiarEditar

model.compile({

optimizer: 'adam',

loss: 'categoricalCrossentropy',

metrics: ['accuracy']

});

**¿Qué significa?**

* optimizer: 'adam': algoritmo de optimización **inteligente y rápido** para ajustar los pesos del modelo.
* loss: 'categoricalCrossentropy': se usa cuando **las clases son exclusivas** (frutas). Penaliza predicciones erróneas.
* metrics: ['accuracy']: mide qué tan seguido acierta el modelo.

**📚 3. ENTRENAMIENTO**

js

CopiarEditar

await model.fit(xsStacked, ysOneHot, {

epochs: 10,

callbacks: {

onEpochEnd: (epoch, logs) => {

...

}

}

});

**¿Qué pasa aquí?**

* xsStacked: tensor de todas las imágenes (normalizadas a valores entre 0 y 1)
* ysOneHot: etiquetas convertidas a vectores "one-hot", ej. "pera" → [0, 0, 1]
* epochs: 10: el modelo ve el conjunto de entrenamiento 10 veces
* onEpochEnd: se ejecuta después de cada ciclo para mostrar pérdida y precisión

**💾 4. GUARDAR Y DESCARGAR EL MODELO**

js

CopiarEditar

await model.save('indexeddb://modelo-frutas');

Esto guarda el modelo **localmente en el navegador (IndexedDB)**. Pero **sí, podés descargarlo** para usarlo en otra app o en otro navegador:

**✅ ¿Cómo descargarlo?**

Agregá esta línea:

js

CopiarEditar

await model.save('downloads://modelo-frutas');

Esto:

* Genera automáticamente un archivo .zip con:
  + model.json (estructura del modelo)
  + group1-shard1of1.bin (pesos aprendidos)
* El navegador te va a pedir descargarlo

**📂 ¿Qué podés hacer con eso?**

* Usarlo en otro proyecto web con tf.loadLayersModel('ruta/model.json')
* Usarlo en Node.js
* Subirlo a una página o un backend para predicción

**✅ Resumen Final**

| **Componente** | **Qué hace** |
| --- | --- |
| conv2d | Detecta características visuales |
| maxPooling | Reduce el tamaño conservando la información importante |
| flatten | Aplana la imagen 2D en un vector 1D |
| dense (64) | Aprende combinaciones complejas |
| dense (N) | Da probabilidades por clase |
| compile | Define cómo se entrena y evalúa el modelo |
| fit | Entrena el modelo |
| save() | Guarda local o descarga el modelo |