What is Deep Learning?

Deep learning is a subfield of machine learning that uses **artificial neural networks** with many layers (hence "deep") to model complex patterns in data.

It's inspired by the brain's network of neurons, where layers of artificial neurons transform inputs into meaningful outputs.

□ Core Concepts

1. Neural Networks

- Composed of **layers** of nodes (neurons).
- Each neuron receives inputs, applies weights, adds a bias, passes it through an activation function, and outputs a value.
- Basic types of layers:
  - **Input layer**: receives raw data.
  - **Hidden layers**: transform inputs into features.
  - **Output layer**: produces final prediction.

---

2. Weights and Biases

- Learnable parameters adjusted during training.
- Weights control importance of each input.
- Bias allows shifting activation.

---

3. Activation Functions

- Add non-linearity so networks can model complex data.
- Common examples:
  - **ReLU (Rectified Linear Unit)**: outputs zero if input < 0, else outputs input.
  - **Sigmoid**: outputs value between 0 and 1 (good for binary classification).
  - **Softmax**: converts outputs to probability distribution over classes.

---

4. Forward Propagation

- Input data passes through network layers.

- Each layer transforms data and passes it on until output is produced.

---

5. Loss Function

- Measures how far off the network's prediction is from the actual target.
- Examples:
  - **Mean Squared Error** (for regression)
  - **Cross-Entropy Loss** (for classification)

---

6. Backpropagation and Gradient Descent

- Core training method.
- Backpropagation computes how to adjust weights to reduce loss.
- Gradient descent updates weights by moving in the direction that decreases loss.

---

7. Epochs and Batches

- **Epoch**: One full pass over the training data.
- **Batch**: Subset of data used to compute gradients (faster training).

□ Why Deep Learning?

- Automatically learns features (no manual feature engineering).
- Excels with large amounts of data and complex patterns.
- Powers applications like image recognition, language translation, speech recognition, and more.

□ Summary Diagram

```
Input Data --> [Layer 1: Weights + Activation] --> [Layer 2: Weights +
Activation] --> ... --> Output Prediction
           <-- Backpropagation (Adjust weights using loss)
```

🧠 Basics of Deep Learning + Types of Neural Networks

---

1. What is Deep Learning?

Deep Learning uses **multi-layered neural networks** to model complex patterns in data. It automatically extracts features and makes predictions.

---

2. Core Components Recap

- **Neurons**: Units that process input to output using weights, bias, and activation functions.
- **Layers**: Stack of neurons (input, hidden, output).
- **Weights & Biases**: Learnable parameters.
- **Activation functions**: Introduce non-linearity (ReLU, Sigmoid, Softmax).
- **Loss Function**: Measures prediction error.
- **Backpropagation**: Adjusts weights to minimize loss.
- **Gradient Descent**: Optimization algorithm to update weights.

---

3. Types of Neural Networks

A. Feedforward Neural Networks (FNN) / Multi-Layer Perceptron (MLP)

- **Structure**: Information flows forward from input to output layer.
- **Use**: Basic classification and regression on tabular or simple data.
- **Limitations**: Doesn't consider sequence or spatial data well.

B. Convolutional Neural Networks (CNN)

- Designed for **image and spatial data**.
- Uses **convolutional layers** to detect spatial features like edges, textures.
- Great for image classification, object detection.
- Can also be used in audio and video processing.

C. Recurrent Neural Networks (RNN)

- Designed for **sequential data** (e.g., text, speech).
- Has loops allowing information to persist across steps.
- Good for language modeling, time series.
- Variants:
    - **LSTM (Long Short-Term Memory)**: Handles long-term dependencies better.
    - **GRU (Gated Recurrent Unit)**: Simplified LSTM, faster to train.

D. Autoencoders

- Used for **unsupervised learning**, especially **dimensionality reduction** or **anomaly detection**.
- Network learns to compress input to a lower dimension and then reconstruct it.

E. Generative Adversarial Networks (GANs)

- Consists of **two networks**: Generator and Discriminator.
- Generator creates fake data; Discriminator tries to detect fake vs real.
- Used for image generation, style transfer, data augmentation.

F. Transformer Networks

- Designed for **sequence modeling** without recurrence.
- Uses **attention mechanisms** to weigh importance of parts of input data.
- State-of-the-art in NLP (e.g., BERT, GPT models).

---

4. Summary Table

Neural Network Type	Use Case	Key Feature
Feedforward (MLP)	Tabular data, simple tasks	Fully connected layers, no loops
CNN	Images, videos, spatial data	Convolutional layers, spatial filters
RNN / LSTM / GRU	Text, time series, sequences	Recurrence, memory of past inputs
Autoencoder	Compression, anomaly detection	Encoder-decoder structure

Neural Network Type	Use Case	Key Feature
GAN	Data generation	Adversarial training between two nets
Transformer	NLP, sequences	Attention mechanism, no recurrence

---

5. Visual Simplification

```
Input --> [Layer 1] --> [Layer 2] --> ... --> Output
            ↑                ↑
       (weights)         (activation)
```

- **CNN** layers slide filters over images.
- **RNN** layers loop through sequence elements.

What is TensorFlow?

TensorFlow is an **open-source deep learning framework** developed by Google. It allows you to build and train **machine learning** and **deep learning** models for tasks such as:

- Image classification
- Text classification (NLP)
- Time series prediction
- Object detection
- Reinforcement learning
- And more

It supports both high-level APIs like **Keras** for easy model building and low-level APIs for flexibility.

How to Use TensorFlow (Beginner Guide)

Here's a simplified guide to get started with TensorFlow in **5 key steps**.

Step 1: Install TensorFlow

```
pip install tensorflow
```

Step 2: Import TensorFlow

```
import tensorflow as tf
print(tf.__version__)  # Check version
```

Step 3: Build a Simple Neural Network

Let's build a model to classify handwritten digits (MNIST dataset).

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Load dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize
```

```
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build model
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),      # Flatten 28x28 images
    layers.Dense(128, activation='relu'),      # Hidden layer
    layers.Dense(10, activation='softmax')     # Output layer (10 classes)
])

# Compile model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train model
model.fit(x_train, y_train, epochs=5)

# Evaluate model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")
```

□ Step 4: Make Predictions

```
predictions = model.predict(x_test)
print(tf.argmax(predictions[0]))  # Predicted class of the first image
```

□ Step 5: Save and Load Model

```
# Save model
model.save('my_model.h5')

# Load model
loaded_model = tf.keras.models.load_model('my_model.h5')
```

□ What Can You Do with TensorFlow?

Task	TensorFlow Tools
Image Classification	CNNs, Transfer Learning
Text Classification	Tokenizers, Embeddings, LSTMs
Object Detection	TensorFlow Object Detection API
Time Series Forecast	RNNs, LSTMs
Chatbots / NLP	Transformers, BERT
Deployment	TensorFlow Lite, TensorFlow.js

□ High-Level vs Low-Level APIs

API Type	Description
Keras	Easy-to-use, beginner-friendly
TF Core	Low-level operations for custom models

Below is a **Complete Python Script** for Text Classification Using Tensorflow and your **own CSV file**, along with a sample format for the CSV data.

□ 1. Sample CSV File (`data.csv`)

Your CSV file should look like this (2 columns: `text`, `label`):

```
text,label
"This movie was fantastic! I loved it.",1
"Terrible movie. It was a waste of time.",0
"Great performances and story.",1
"Awful script and bad acting.",0
"Best film of the year!",1
"I don't recommend it at all.",0
"Amazing experience!",1
"Completely boring and predictable.",0
```

Save this file as `data.csv` in the same folder as your Python script.

□ 2. Full Python Code (`text_classifier.py`)

```
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense
```

```
import matplotlib.pyplot as plt

# Step 1: Load CSV
df = pd.read_csv('data.csv')

# Step 2: Clean Data (drop any missing values)
df.dropna(inplace=True)

# Step 3: Split Data
texts = df['text'].astype(str).tolist()
labels = df['label'].tolist()

x_train_texts, x_test_texts, y_train, y_test = train_test_split(
    texts, labels, test_size=0.2, random_state=42)

# Step 4: Tokenization and Padding
vocab_size = 1000
maxlen = 50
tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
tokenizer.fit_on_texts(x_train_texts)

x_train_seq = tokenizer.texts_to_sequences(x_train_texts)
x_test_seq = tokenizer.texts_to_sequences(x_test_texts)

x_train_pad = pad_sequences(x_train_seq, maxlen=maxlen, padding='post',
truncating='post')
x_test_pad = pad_sequences(x_test_seq, maxlen=maxlen, padding='post',
truncating='post')

# Step 5: Build the Model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=32, input_length=maxlen),
    GlobalAveragePooling1D(),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Step 6: Train the Model
history = model.fit(x_train_pad, y_train,
                    epochs=10,
                    batch_size=4,
                    validation_split=0.2)

# Step 7: Evaluate the Model
loss, acc = model.evaluate(x_test_pad, y_test)
print(f"Test Accuracy: {acc:.2f}")

# Step 8: Visualize Accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.show()

# Step 9: Predict on New Samples

sample_texts = ["I really enjoyed the movie!", "It was terrible and boring."]
sample_seq = tokenizer.texts_to_sequences(sample_texts)
sample_pad = pad_sequences(sample_seq, maxlen=maxlen, padding='post')

predictions = model.predict(sample_pad)

for i, text in enumerate(sample_texts):
    sentiment = 'Positive' if predictions[i] > 0.5 else 'Negative'
    print(f"Text: {text} => Prediction: {sentiment}
({predictions[i][0]:.2f})")
```

▶□ How to Run:

1. Save the sample data as `data.csv` in your working directory.
2. Save the Python code above as `text_classifier.py`.
3. In terminal or command prompt:

```
python text_classifier.py
```

□ Output:

- Console will show training/validation accuracy
- Final test accuracy
- A plot of training performance
- Sample predictions like:

```
Text: I really enjoyed the movie! => Prediction: Positive (0.92)
Text: It was terrible and boring. => Prediction: Negative (0.12)
```

Here's the **full working code** for an **image classification model** using **TensorFlow + Keras**, based on the **CIFAR-10** dataset. It's self-contained — you can copy and paste it into a Python file and run it.

We'll use the built-in **CIFAR-10 dataset**, which contains 60,000 images (32x32 pixels) in 10 categories like airplanes, cars, birds, etc.

□ Full Code: Image Classification with CIFAR-10

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np

# Step 1: Load CIFAR-10 Dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Normalize pixel values (0 to 1)
x_train = x_train / 255.0
x_test = x_test / 255.0

# Class labels
class_names = ['airplane', 'car', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Step 2: Build the CNN Model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes
])

# Step 3: Compile the Model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Step 4: Train the Model
history = model.fit(x_train, y_train,
                    epochs=10,
                    validation_data=(x_test, y_test),
```

```
                    batch_size=64)

# Step 5: Evaluate the Model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"\n Test Accuracy: {test_acc:.2f}")

# Step 6: Plot Accuracy Curves
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training vs Validation Accuracy')
plt.grid(True)
plt.show()

# Step 7: Make Predictions and Visualize
predictions = model.predict(x_test[:5])

for i in range(5):
    plt.figure(figsize=(2, 2))
    plt.imshow(x_test[i])
    pred_label = class_names[np.argmax(predictions[i])]
    true_label = class_names[y_test[i][0]]
    plt.title(f"Predicted: {pred_label}\nTrue: {true_label}")
    plt.axis('off')
    plt.show()
```

□ Requirements

Make sure you have the required libraries:

```
pip install tensorflow matplotlib numpy
```

□ Output

- **Training progress** with accuracy and validation
- **Test accuracy score**
- **Graph** showing training vs validation accuracy
- **Images** from the test set with predicted and true labels

Here's a **complete image classification script** using **your own image folders** (e.g., `cats/`, `dogs/`) with **TensorFlow and Keras**.

□ Folder Structure

Organize your images like this:

```
dataset/
├── cats/
│   ├── cat1.jpg
│   ├── cat2.jpg
│   └── ...
├── dogs/
│   ├── dog1.jpg
│   ├── dog2.jpg
│   └── ...
```

Each subfolder name (e.g., `cats`, `dogs`) is automatically treated as a **class label**.

□ Full Code: Image Classification from Folder

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
import os

# Parameters
image_size = (180, 180)
batch_size = 32
data_dir = 'dataset'  # path to your image dataset folder

# Step 1: Load Dataset from Folder
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=image_size,
    batch_size=batch_size
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=image_size,
```

```
    batch_size=batch_size
)

# Step 2: Optimize Data Pipeline
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)

class_names = train_ds.class_names
print("Class names:", class_names)

# Step 3: Build the CNN Model
model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(180, 180, 3)),
    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(128, 3, activation='relu'),
    layers.MaxPooling2D(),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')  # number of classes
])

# Step 4: Compile the Model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Step 5: Train the Model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)

# Step 6: Plot Accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Step 7: Predict on One Batch
for images, labels in val_ds.take(1):
    predictions = model.predict(images)
    for i in range(5):  # Show first 5 predictions
        plt.figure(figsize=(2, 2))
        plt.imshow(images[i].numpy().astype("uint8"))
        pred_label = class_names[np.argmax(predictions[i])]
        true_label = class_names[labels[i]]
        plt.title(f"Pred: {pred_label} | True: {true_label}")
        plt.axis('off')
        plt.show()
```

□ How to Use

1. Create a folder named `dataset/`
2. Add one subfolder per class (e.g., `cats`, `dogs`)
3. Place relevant images in each folder
4. Save the code above as `image_classifier.py`
5. Run it:

```
python image_classifier.py
```

□ Notes

- You can add more classes (e.g., `horses`, `birds`) — just add more subfolders.
- Images will be resized to `180x180` automatically.
- Make sure your dataset isn't too small (at least 100+ images per class recommended).

Here's a curated list of **TensorFlow projects for beginners** that will help you build practical skills while learning core concepts of deep learning and machine learning.

□ Beginner TensorFlow Projects (Hands-On)

1. Image Classification (CIFAR-10 or MNIST)

- **Goal**: Classify images into categories (e.g., digits, objects).
- **Concepts**: CNNs, softmax, accuracy, overfitting.
- **Dataset**: Built-in (`tf.keras.datasets.cifar10` or `mnist`).

□ **Skills Learned**:

- CNN architecture
- Model training and evaluation
- Visualization of predictions

---

2. Text Sentiment Classification

- **Goal**: Classify movie reviews (positive/negative).
- **Concepts**: Tokenization, Embedding, LSTM, RNN.
- **Dataset**: IMDB dataset (built-in)

 Skills Learned:

- Working with text
- Embedding layers
- Sequence modeling

---

3. Custom Image Classification from Folder

- **Goal**: Classify your own images (e.g., cats vs dogs).
- **Concepts**: Data loading, preprocessing, CNNs.
- **Dataset**: From local folders (like `cats/`, `dogs/`).

 Skills Learned:

- Custom dataset handling
- Transfer learning (optional)
- Fine-tuning

---

4. Digit Recognition App (MNIST + GUI)

- **Goal**: Draw a digit and let the model predict it.
- **Concepts**: MNIST + simple GUI (Tkinter).
- **Bonus**: Integrate TensorFlow model with GUI.

 Skills Learned:

- Digit image preprocessing
- Model inference
- GUI interaction

---

5. Binary Classification with CSV (e.g., Titanic)

- **Goal**: Predict survival from structured data.
- **Concepts**: Dense layers, tabular data, feature columns.

- **Dataset**: Titanic dataset (`.csv`)

 Skills Learned:

- Feature engineering
- Model evaluation (accuracy, precision)
- Loading and preprocessing CSV data

---

6. Real-Time Object Detection (Using pre-trained model)

- **Goal**: Detect objects from webcam using TensorFlow Lite or MobileNet.
- **Concepts**: Transfer learning, object detection APIs.

 Skills Learned:

- Loading pre-trained models
- Working with video frames
- Real-time ML

---

7. House Price Prediction (Regression)

- **Goal**: Predict house prices based on features.
- **Concepts**: Regression, normalization.
- **Dataset**: Boston housing dataset (or your own CSV).

 Skills Learned:

- Regression loss functions (MSE)
- Feature normalization
- Dense network for regression

---

8. Fashion Image Classification

- **Goal**: Classify fashion items (T-shirt, bag, etc.).
- **Dataset**: Fashion MNIST (built-in)
- **Concepts**: CNNs, softmax, accuracy

 Skills Learned:

- Lightweight CNNs

- Evaluating multi-class models

---

9. Text Generation with LSTM (Character-based RNN)

- **Goal**: Train a model to generate text (like Shakespeare).
- **Dataset**: Any text (e.g., a book or poem)
- **Concepts**: RNN, LSTM, sequence modeling

 Skills Learned:

- Character embeddings
- Sequential prediction
- Sampling generated text

---

10. Image Caption Generator (Intermediate)

- **Goal**: Generate captions for images using CNN + RNN
- **Concepts**: Encoder-decoder, attention (advanced)
- **Dataset**: MS COCO or Flickr8k

 Skills Learned:

- Combining vision and language
- Advanced deep learning workflows

Tools You'll Practice

Tool/Concept	Use
`tf.keras`	Model building & training
`Tokenizer`	Preprocessing text
`ImageDataGenerator`	Augmenting image data
`ModelCheckpoint`	Saving best models
`TensorBoard`	Monitoring training visually

What is CNN?

□ CNN = Convolutional Neural Network

CNNs are **neural networks designed to process data with a grid-like structure**, such as images.

□ Best for:

- Image classification
- Object detection
- Facial recognition
- Medical image analysis

□ How It Works:

Instead of looking at the entire image all at once, CNNs **slide small filters** over the image to learn **local features** like edges, textures, or patterns.

Typical CNN Layers:

1. **Convolution Layer** – detects features (e.g., edges)
2. **ReLU Activation** – introduces non-linearity
3. **Pooling Layer** – reduces dimensionality (e.g., MaxPooling)
4. **Fully Connected Layer** – final classification

□ Example:

For a cat image, early layers detect:

- Edges → Shapes → Eyes/Ears → "Cat" □

□ What is RNN?

□ RNN = Recurrent Neural Network

RNNs are designed to process **sequential data**, where **order matters**, like sentences, time series, or audio.

□ Best for:

- Text/speech processing
- Sentiment analysis
- Time series forecasting
- Music/audio generation
- Language translation

□ How It Works:

RNNs **loop over sequences**, remembering what they saw before — like reading a sentence **word by word**, using past context to interpret current input.

□ *Example:*

Sentence: “I love deep learning.”

The RNN reads:

- “I” → remembers it
- “love” → combines with memory of “I”
- “deep” → adds more context
- and so on...

□ RNN Limitation:

- Can struggle with **long sequences** due to vanishing gradients.
- Solved by better variants like **LSTM** (Long Short-Term Memory) and **GRU** (Gated Recurrent Unit).

□ CNN vs RNN: Comparison

Feature	CNN	RNN
Input Type	Images (2D), spatial data	Text, time series, sequences
Memory	No memory	Has memory (uses previous data)
Use Case	Image recognition/classification	Text, speech, time-series
Architecture	Uses filters/kernels	Uses loops over sequences

□ Summary:

- **CNN**: Best for *spatial patterns* (images)
- **RNN**: Best for *temporal/sequential patterns* (text, time series)

Perfect! Below are two **simple TensorFlow code examples**:

- A **CNN** for image classification (using MNIST)
- An **RNN** for text classification (using IMDB movie reviews)

---

1. CNN Example— Image Classification (MNIST)

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Load MNIST dataset (handwritten digits)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalize and reshape (28x28 grayscale → 28x28x1)
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0

# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D(2, 2),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 digits
])

# Compile and train
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, validation_split=0.1)

# Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")
```

---

2. RNN Example — Text Classification (IMDB)

```
import tensorflow as tf
from tensorflow.keras import layers

# Load IMDB dataset (movie reviews)
(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.imdb.load_data(num_words=10000)

# Pad sequences to same length
x_train = tf.keras.preprocessing.sequence.pad_sequences(x_train, maxlen=200)
x_test = tf.keras.preprocessing.sequence.pad_sequences(x_test, maxlen=200)

# Build RNN model
model = tf.keras.Sequential([
    layers.Embedding(input_dim=10000, output_dim=64, input_length=200),
    layers.SimpleRNN(64),  # Try LSTM() for better performance
    layers.Dense(1, activation='sigmoid')  # Binary classification
])

# Compile and train
```

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, validation_split=0.1)

# Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")
```

Summary:-

Model	Dataset	Task	Layers Used
CNN	MNIST	Digit recognition	Conv2D, MaxPooling, Dense
RNN	IMDB	Sentiment analysis	Embedding, SimpleRNN/LSTM, Dense