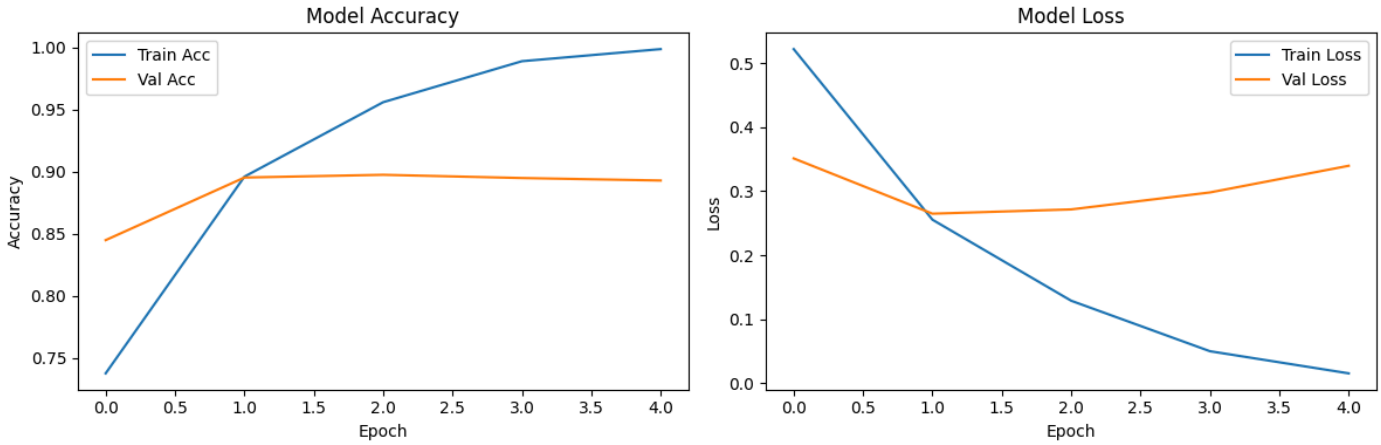
```
# Step 1: Import Libraries
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
import matplotlib.pyplot as plt

# Step 2: Set Parameters
vocab_size = 10000      # Use top 10,000 words
maxlen = 500            # Pad & truncate all sequences to length 500
embedding_dim = 128     # Embedding output dimension

# Step 3: Load IMDB Dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Step 4: Pad Sequences
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# Step 5: Build 1D CNN Model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxlen),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(10, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Step 6: Compile Model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Step 7: Train Model
history = model.fit(x_train, y_train,
                    epochs=5,
                    batch_size=128,
                    validation_split=0.2)

# Step 8: Evaluate Model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {accuracy:.4f}")

# Step 9: Plot Accuracy and Loss
plt.figure(figsize=(12, 4))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

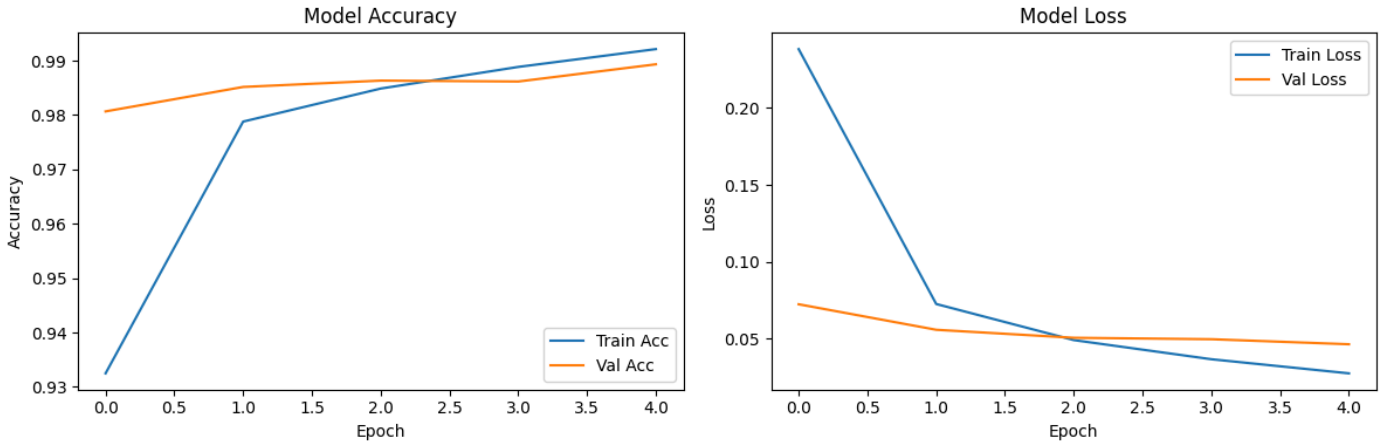
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just
  warnings.warn(
Epoch 1/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 132s 825ms/step - accuracy: 0.6497 - loss: 0.6228 - val_accuracy: 0.8450 - val_loss: 0.3513
Epoch 2/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 139s 807ms/step - accuracy: 0.8871 - loss: 0.2769 - val_accuracy: 0.8954 - val_loss: 0.2650
Epoch 3/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 143s 815ms/step - accuracy: 0.9573 - loss: 0.1338 - val_accuracy: 0.8976 - val_loss: 0.2717
Epoch 4/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 148s 853ms/step - accuracy: 0.9894 - loss: 0.0524 - val_accuracy: 0.8950 - val_loss: 0.2982
Epoch 5/5
157/157 ━━━━━━━━━━━━━━━━━━━━ 138s 825ms/step - accuracy: 0.9992 - loss: 0.0153 - val_accuracy: 0.8930 - val_loss: 0.3398
782/782 ━━━━━━━━━━━━━━━━━━━━ 46s 59ms/step - accuracy: 0.8859 - loss: 0.3577

Test Accuracy: 0.8861
```




```
# Step 1: Import Libraries
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# Step 2: Load and Preprocess Data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize pixel values to [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape to (28, 28, 1) for CNN input
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Step 3: Build CNN Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Step 5: Train Model
history = model.fit(x_train, y_train, epochs=5, batch_size=128, validation_split=0.1)
```

```
# Step 6: Evaluate Model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {accuracy:.4f}")

# Step 7: Plot Accuracy and Loss
plt.figure(figsize=(12, 4))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 28s 62ms/step - accuracy: 0.8672 - loss: 0.4835 - val_accuracy: 0.9807 - val_loss: 0.0723
Epoch 2/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 40s 61ms/step - accuracy: 0.9779 - loss: 0.0755 - val_accuracy: 0.9852 - val_loss: 0.0557
Epoch 3/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 26s 61ms/step - accuracy: 0.9849 - loss: 0.0514 - val_accuracy: 0.9863 - val_loss: 0.0505
Epoch 4/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 26s 62ms/step - accuracy: 0.9890 - loss: 0.0350 - val_accuracy: 0.9862 - val_loss: 0.0495
Epoch 5/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 25s 60ms/step - accuracy: 0.9927 - loss: 0.0258 - val_accuracy: 0.9893 - val_loss: 0.0463
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9799 - loss: 0.0592

Test Accuracy: 0.9845
```



Feature	1D CNN	2D CNN
Input Type	Sequences (e.g., text, time series)	Images (height × width × channels)
Kernel Shape	1D (e.g., size 3 over sequence)	2D (e.g., 3×3 over image grid)
Use Case	NLP, audio, time series	Computer vision, image classification
Example Layer	`Conv1D(filters=64, kernel_size=3)`	`Conv2D(filters=64, kernel_size=(3,3))`

2. Why is padding used in convolutional layers? Padding maintains the spatial dimensions (height/width) of the input after convolution, preventing shrinkage of feature maps. Common padding:

'valid' → no padding (shrinks output)

'same' → pads input so output has same size

This helps preserve edge information and allows deeper models without losing too much data.

3. What does a filter/kernel do in a CNN? A filter/kernel is a small matrix (e.g., 3×3) that:

Slides across input data

Performs element-wise multiplication and summation

Extracts specific patterns (edges, textures, shapes)

Each filter learns to detect a specific feature from the input.

4. How does a max pooling layer affect the feature map? Max Pooling:

Reduces spatial dimensions (downsampling)

Keeps the strongest activation (most important feature)

Helps reduce overfitting and computation

E.g., a 2×2 max pooling cuts height and width by half.

5. Why are CNNs preferred over Feedforward Neural Networks for spatial/temporal data? CNNs are preferred because they:

Use local connections → learn spatial features

Share weights → fewer parameters, more efficient

Handle translation invariance

Can scale better to high-dimensional inputs (e.g., images)

Feedforward networks ignore spatial structure and would require too many parameters.