

## Unit-III: Neural Networks

---

**Unit-III:** Neural Networks: Anatomy of Neural Network, Introduction to Keras: Keras, TensorFlow, Theano and CNTK, setting up Deep Learning Workstation, Classifying Movie Reviews: Binary Classification, Classifying newswires: Multiclass Classification.

---

### Anatomy of a Neural Network (Simplified Explanation with Examples)

A **neural network** is like a team of problem solvers working together to understand and predict things based on the data they receive. It mimics the human brain, where neurons (small processing units) work together to process information.

Let us break it down step by step.

---

### 1. Layers of a Neural Network

A neural network has three main types of layers:

#### 1.1 Input Layer

- This layer is like the **eyes and ears** of the network.
- It takes in the raw data (numbers, images, text, etc.) and passes it to the next layer.
- Each neuron in the input layer represents one feature of the data.

#### ◆ Example:

If we are building a neural network to predict house prices, the input layer might have neurons representing:

- Size of the house (sq. ft.)
  - Number of bedrooms
  - Location rating
  - Age of the house
- 

#### 1.2 Hidden Layers

- These layers process the data by learning patterns and relationships.
- The more hidden layers a network has, the more complex patterns it can learn.
- Each neuron in a hidden layer takes input, applies a mathematical transformation, and passes it to the next layer.

### ◆ Example:

For house price prediction, hidden layers might learn:

- How house size affects price.
- How location influences value.
- How different features interact.

### ◆ Real-Life Example:

Think of hidden layers like the **chefs in a restaurant**.

- The **input layer** is the waiter taking orders.
  - The **hidden layers (chefs)** prepare the meal based on the order details.
  - The **output layer** delivers the final dish.
- 

## 1.3 Output Layer

- This layer provides the final prediction or result.
- The number of neurons in this layer depends on the type of problem.

### Types of Output Layers:

#### 1. Binary Classification (Yes/No, True/False):

- Example: Spam detection (Spam or Not Spam)
- Uses **one neuron** with a **sigmoid activation function** (output is between 0 and 1).

#### 2. Multiclass Classification (Multiple Categories):

- Example: Recognizing different types of animals (Cat, Dog, Horse).
- Uses **multiple neurons**, one for each category, with a **softmax activation function**.

#### 3. Regression (Numerical Prediction):

- Example: Predicting house prices.
- Uses **one neuron** with a **linear activation function** (output is a real number).

### ◆ Real-Life Example:

The output layer is like the **cashier at a restaurant** who gives the final bill based on the order and what the chefs prepared.

---

## 2. Neurons (Processing Units)

- Each neuron receives input, processes it, and passes it forward.

- Neurons are connected by **weights**, which determine the importance of each input.

◆ **Example:**

In a spam detection email classifier:

- A neuron might check if the subject line contains the word "FREE".
  - Another neuron might check if the email contains too many links.
  - If both neurons detect spam-like features, they pass a strong signal forward.
- 

### 3. Weights and Biases (How the Network Learns)

- **Weights:** Control the strength of connections between neurons.
- **Bias:** Helps adjust the output, making the network more flexible.

◆ **Example:**

- If a house's **location** is very important in determining its price, the network will assign a **high weight** to that input.
- If a house's **paint color** is not important, the network will assign a **low weight** to that input.

◆ **Real-Life Example:**

Think of weights like **teacher grading criteria**:

- Homework (20%)
  - Tests (50%)
  - Class participation (30%)
- Each factor is **weighted** differently in the final grade calculation.
- 

### 4. Activation Functions (Decision Makers in the Network)

- Activation functions decide whether a neuron should be "activated" (send information forward) or not.
- Without activation functions, the network would just be performing basic linear calculations.

◆ **Types of Activation Functions:**

1. **Sigmoid:**

- Used for binary classification (Yes/No).
- Example: Detecting if an email is spam or not.

2. **ReLU (Rectified Linear Unit):**

- Used in hidden layers because it helps train deep networks efficiently.
- Example: Image recognition (detecting edges, colors, shapes).

### 3. Softmax:

- Used for multi-class classification problems.
- Example: Identifying different types of animals (dog, cat, bird).

#### ◆ Real-Life Example:

Think of activation functions like **light switches** in a house:

- Some switches turn on a little (dim light – sigmoid).
  - Some turn on fully or not at all (on/off – ReLU).
- 

## 5. Loss Function (How Wrong is the Prediction?)

- The loss function tells the neural network how far off its prediction is from the actual value.
- The goal is to minimize this error during training.

#### ◆ Types of Loss Functions:

- **Binary Crossentropy:** Used for Yes/No classification.
- **Categorical Crossentropy:** Used for multi-class classification.
- **Mean Squared Error (MSE):** Used for predicting numbers (e.g., house prices).

#### ◆ Real-Life Example:

- A loss function is like **Google Maps estimating your travel time**.
  - If it predicts you will take 30 minutes but you actually take 40, the loss function tells it to improve future estimates.
- 

## 6. Optimizers (How the Network Learns from Mistakes)

- Optimizers adjust the weights and biases based on the loss function to improve predictions.

#### ◆ Common Optimizers:

1. **Stochastic Gradient Descent (SGD):** Updates weights gradually.
2. **Adam Optimizer:** Adapts learning speed dynamically (most popular).

#### ◆ Real-Life Example:

- Think of an optimizer as a **coach helping an athlete improve**.

- If a runner is too slow, the coach suggests changes in training.
  - The optimizer tweaks the neural network similarly.
- 

## 7. Backpropagation (Learning from Mistakes)

- The network calculates how wrong it was, then updates weights to improve.
- This is done using **gradient descent**.

### ◆ Real-Life Example:

- Imagine a student taking a math test.
  - If they get a question wrong, the teacher shows them their mistake.
  - Next time, they try not to repeat the mistake.
- 

## Final Real-Life Example: Self-Driving Cars

A self-driving car uses a neural network to make decisions:

1. **Input Layer:** Sensors collect data (speed, road signs, objects).
2. **Hidden Layers:** Process patterns (detect red lights, pedestrians).
3. **Output Layer:** Decides whether to stop, slow down, or turn.
4. **Weights & Biases:** Adjust importance (pedestrians > lane markings).
5. **Loss Function:** If the car makes a bad decision, it learns from it.
6. **Optimizer:** Adjusts to improve future driving performance.

## Introduction to Keras: Understanding Keras in AI & Deep Learning

### 1. What is Keras?

Keras is an **open-source deep learning framework** that provides a simple and intuitive interface for building neural networks. It is written in **Python** and is widely used in **Artificial Intelligence (AI) and Machine Learning (ML)** applications.

#### 1.1 Why Use Keras?

- ✓ **User-Friendly** – Simple and easy to use for beginners.
- ✓ **Modular & Flexible** – Provides a high-level API for rapid development.
- ✓ **Runs on Multiple Backends** – Supports **TensorFlow, Theano, and Microsoft CNTK**.
- ✓ **Supports CPUs & GPUs** – Efficient computation for deep learning.
- ✓ **Pretrained Models** – Includes VGG, ResNet, MobileNet, etc., for transfer learning.

#### ◆ Real-Life Analogy:

Keras is like a **modern kitchen with pre-built tools** that help you cook faster, while TensorFlow is like a **professional kitchen** where you have to do everything manually.

---

### 2. Why is Keras Important?

Keras **simplifies deep learning** by providing an easy-to-use interface while maintaining the power of advanced frameworks like **TensorFlow**.

#### ◆ Key Benefits:

1. **Rapid Prototyping** – Build and test models quickly.
  2. **Scalability** – Works on small and large datasets.
  3. **Community Support** – Large user base and active development.
  4. **Integration with TensorFlow** – Keras is now part of TensorFlow as **tf.keras**.
- 

### 3. How Keras Works?

Keras acts as a **high-level wrapper** that runs on top of deep learning frameworks such as:

Backend	Description
<b>TensorFlow</b>	Google's powerful deep learning framework (Most popular backend)
<b>Theano</b>	Early framework for deep learning (Now deprecated)
<b>Microsoft CNTK</b>	Deep learning framework by Microsoft (Less popular)

--> **Keras now primarily uses TensorFlow as its backend.**

---

## 4. Setting Up Keras

Before using Keras, we need to install it along with TensorFlow.

### 4.1 Installation

```
pip install tensorflow keras
```

### 4.2 Importing Keras

```
import keras
```

```
import tensorflow as tf
```

```
print(keras.__version__) # Check Keras version
```

```
print(tf.__version__) # Check TensorFlow version
```

---

## 5. Building a Neural Network with Keras

Keras provides an easy way to build **deep learning models** using a **Sequential API**.

### 5.1 Step-by-Step Guide to Building a Simple Neural Network

We will create a **basic neural network** to classify **handwritten digits (MNIST dataset)**.

---

#### Step 1: Import Libraries

```
from keras.models import Sequential
```

```
from keras.layers import Dense, Flatten
```

```
from keras.datasets import mnist
```

```
from keras.utils import to_categorical
```

---

#### Step 2: Load & Preprocess Data

```
# Load the MNIST dataset
```

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
# Normalize images (scale pixel values between 0 and 1)
```

```
train_images = train_images / 255.0
```

```
test_images = test_images / 255.0
```

```
# Convert labels to categorical format (one-hot encoding)
```

```
train_labels = to_categorical(train_labels, 10)
```

```
test_labels = to_categorical(test_labels, 10)
```

---

### Step 3: Create the Model

```
model = Sequential([
    Flatten(input_shape=(28, 28)), # Convert 2D images to 1D
    Dense(128, activation='relu'), # Hidden layer with 128 neurons
    Dense(10, activation='softmax') # Output layer with 10 neurons (for digits 0-9)
])
```

---

### Step 4: Compile the Model

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

---

### Step 5: Train the Model

```
model.fit(train_images, train_labels, epochs=5, batch_size=32, validation_split=0.2)
```

---

### Step 6: Evaluate the Model

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_acc}')
```

---

## 6. Keras Model APIs

Keras provides different ways to build models:

### 6.1 Sequential API (Easy & Quick)

- Builds models layer-by-layer in a **linear stack**.
- **Example:**

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(784,)),
    Dense(10, activation='softmax')
])
```

---



## 6.2 Functional API (More Flexible)

- Allows creating complex architectures (multiple inputs/outputs).
- **Example:**

```
from keras.models import Model
```

```
from keras.layers import Input, Dense
```

```
input_layer = Input(shape=(784,))
```

```
hidden_layer = Dense(128, activation='relu')(input_layer)
```

```
output_layer = Dense(10, activation='softmax')(hidden_layer)
```

```
model = Model(inputs=input_layer, outputs=output_layer)
```

## 7. Key Components of Keras

Component	Description	Example
<b>Layers</b>	Building blocks of neural networks	Dense(128, activation='relu')
<b>Optimizers</b>	Adjust weights to minimize loss	optimizer='adam'
<b>Loss Functions</b>	Measure prediction error	loss='categorical_crossentropy'
<b>Metrics</b>	Evaluate model performance	metrics=['accuracy']

## 8. Pretrained Models in Keras (Transfer Learning)

Keras provides **pretrained models** that can be used for **image classification, object detection, and feature extraction**.

### 8.1 Using Pretrained Model (MobileNetV2)


```
from keras.applications import MobileNetV2
```

```
# Load MobileNetV2 model with pretrained weights
```

```
model = MobileNetV2(weights='imagenet')
```

✓ Used for Image Classification & Transfer Learning.

## 9. Applications of Keras

 Keras is widely used in various AI applications:

- ✓ **Image Recognition (Face Detection, Object Recognition)** – Used in **Self-driving Cars, Medical Imaging.**
- ✓ **Natural Language Processing (NLP)** – Used in **Chatbots, Speech Recognition.**
- ✓ **Recommendation Systems** – Used by **Netflix, Amazon, YouTube.**
- ✓ **Healthcare & Drug Discovery** – Used in **AI-based disease detection.**
- ✓ **Finance & Fraud Detection** – Used in **predicting stock prices & fraud detection.**

Here are some key topics:

- 1 **Keras Layers** – Understanding Dense, Convolutional, Recurrent layers, etc.
- 2 **Keras Optimizers** – Adam, SGD, RMSprop, and how they affect training.
- 3 **Keras Loss Functions** – How to choose the right loss function for classification & regression.
- 4 **Keras Callbacks** – How to use EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, etc.
- 5 **Keras Functional API** – Building complex models with multiple inputs/outputs.
- 6 **Keras Transfer Learning** – Using pretrained models like VGG, ResNet, MobileNet.
- 7 **Keras for NLP** – Text processing, embedding layers, and LSTMs.
- 8 **Keras for Computer Vision** – Convolutional Neural Networks (CNNs).

Great! Let's go through all the important **Keras features** in detail, step by step.

### 1. Keras Layers

Layers are the **building blocks** of a neural network in Keras.

#### 1.1 Types of Layers in Keras

Layer Type	Purpose	Example Usage
<b>Dense (Fully Connected)</b>	Basic neural network layer	Dense(128, activation='relu')
<b>Conv2D (Convolutional Layer)</b>	Extracts features from images	Conv2D(32, kernel_size=3, activation='relu')
<b>MaxPooling2D (Pooling Layer)</b>	Reduces spatial dimensions in CNNs	MaxPooling2D(pool_size=2)
<b>Flatten</b>	Converts multi-dimensional input to 1D	Flatten()
<b>Dropout</b>	Prevents overfitting	Dropout(0.5)

Layer Type	Purpose	Example Usage
<b>LSTM/GRU (Recurrent Layer)</b>	Used in sequence models (NLP, Time Series)	LSTM(64)

## 1.2 Example: Using Different Layers in Keras

```
from keras.models import Sequential
```

```
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
```

```
# Define model
```

```
model = Sequential([
```

```
    Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)),
```

```
    MaxPooling2D(pool_size=(2,2)),
```

```
    Flatten(),
```

```
    Dense(128, activation='relu'),
```

```
    Dropout(0.5),
```

```
    Dense(10, activation='softmax')
```

```
])
```

```
# Compile model
```

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

```
# Model Summary
```

```
model.summary()
```

✓ Used in Image Classification, NLP, and other Deep Learning applications.

## 2. Keras Optimizers

Optimizers **adjust the weights** of the neural network to minimize the loss function.

### 2.1 Common Optimizers in Keras

Optimizer	Description	Best Use Case
<b>SGD (Stochastic Gradient Descent)</b>	Simple, slow but stable	Small datasets, linear regression
<b>Adam (Adaptive Moment Estimation)</b>	Most widely used, fast convergence	Works well on most problems
<b>RMSprop</b>	Designed for RNNs, adapts learning rates	Time-series, speech recognition
<b>Adagrad</b>	Adjusts learning rates dynamically	Sparse data, NLP tasks

### 2.2 Example: Using Different Optimizers

```
from keras.optimizers import Adam, SGD, RMSprop
```

```
# Compile model with different optimizers
```

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',  
metrics=['accuracy'])
```

✓ Adam is recommended for most deep learning applications.

## 3. Keras Loss Functions

Loss functions measure **how well the model is performing**.

### 3.1 Choosing the Right Loss Function

Problem Type	Loss Function
<b>Binary Classification (Yes/No)</b>	binary_crossentropy
<b>Multiclass Classification</b>	categorical_crossentropy
<b>Regression (Predicting Numbers)</b>	mean_squared_error

### 3.2 Example: Choosing a Loss Function

# Binary Classification

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

# Multiclass Classification

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

# Regression

```
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```

✓ **Choosing the right loss function improves model accuracy.**

## 4. Keras Callbacks

Callbacks are used to **monitor and control the training process**.

### 4.1 Common Callbacks

Callback	Purpose
<b>EarlyStopping</b>	Stops training if the model stops improving
<b>ModelCheckpoint</b>	Saves the best model during training
<b>ReduceLROnPlateau</b>	Reduces learning rate when the model stops improving

### 4.2 Example: Using Callbacks

```
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
```

```
callbacks = [
```

```
    EarlyStopping(monitor='val_loss', patience=5),
```

```
    ModelCheckpoint('best_model.h5', save_best_only=True),
```

```
    ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3)
```

```
]
```

```
model.fit(train_images, train_labels, epochs=50, batch_size=32, validation_split=0.2,
callbacks=callbacks)
```

✓ **Callbacks help prevent overfitting and speed up training.**

## 5. Keras Functional API

The **Functional API** allows building complex models with **multiple inputs/outputs**.

### 5.1 Example: Functional API Model

```
from keras.models import Model

from keras.layers import Input, Dense

# Input Layer
input_layer = Input(shape=(784,))

# Hidden Layers
hidden_layer = Dense(128, activation='relu')(input_layer)

# Output Layer
output_layer = Dense(10, activation='softmax')(hidden_layer)

# Create Model
model = Model(inputs=input_layer, outputs=output_layer)

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

model.summary()
```

✓ Used in advanced deep learning applications like multi-task learning.

---

## 6. Keras Transfer Learning

Transfer learning uses **pretrained models** to improve performance on new datasets.

### 6.1 Using a Pretrained Model (MobileNetV2)

```
from keras.applications import MobileNetV2
from keras.models import Model
from keras.layers import Dense, Flatten

# Load MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224,224,3))

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Add new layers
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
output_layer = Dense(10, activation='softmax')(x)

# Create model
model = Model(inputs=base_model.input, outputs=output_layer)

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

model.summary()
```

✓ Used for Image Recognition & Object Detection.

---

## 7. Keras for NLP (Text Processing)

Keras provides tools for **Natural Language Processing (NLP)**.

### 7.1 Example: Using Embedding Layer for NLP

```
from keras.layers import Embedding, LSTM

model = Sequential([
    Embedding(input_dim=10000, output_dim=128, input_length=100),
    LSTM(64),
    Dense(1, activation='sigmoid')
])

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

✓ Used in Chatbots, Sentiment Analysis, and Speech Recognition.

---

## 8. Keras for Computer Vision (CNNs)

CNNs are used for **image classification and object detection**.

### 8.1 Example: CNN Model

```
model = Sequential([
    Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(64,64,3)),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

✓ Used in Face Recognition, Medical Imaging, and Self-Driving Cars.

---



- 1 **Keras Sequential API** – How to build deep learning models step by step.
- 2 **Keras Functional API** – Creating complex models with multiple inputs/outputs.
- 3 **Keras CNNs (Computer Vision)** – Building and training a Convolutional Neural Network (CNN).
- 4 **Keras NLP (Text Processing & LSTMs)** – Using Keras for text classification, sentiment analysis, and RNNs.
- 5 **Keras Transfer Learning** – Using pretrained models like VGG16, ResNet, and MobileNet.
- 6 **Keras Hyperparameter Tuning** – Optimizing model performance with Keras Tuner.
- 7 **Keras Callbacks & Model Saving** – Using EarlyStopping, ModelCheckpoint, and saving/loading models.

Awesome! Let's go through **all key Keras topics** in detail with **step-by-step tutorials** and **Python examples**. 🚀

---

## 1 Keras Sequential API

The **Sequential API** is the easiest way to build deep learning models in Keras. It allows stacking layers in a **linear fashion** (one after another).

### 1.1 Example: Build a Simple Neural Network (Sequential API)

```
from keras.models import Sequential

from keras.layers import Dense

# Create a Sequential model
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)), # Hidden layer with 128 neurons
    Dense(10, activation='softmax') # Output layer for 10 classes
])

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

# Summary of the model
model.summary()

✓ Best for simple feedforward networks.
```

---

## ❏ Keras Functional API

The **Functional API** allows creating **complex architectures**, like multiple inputs/outputs and shared layers.

### 2.1 Example: Multi-Input Model (Functional API)

python

CopyEdit

```
from keras.models import Model
from keras.layers import Input, Dense, concatenate
```

```
# Input layers
```

```
input_a = Input(shape=(64,))
```

```
input_b = Input(shape=(32,))
```

```
# Hidden layers
```

```
hidden_a = Dense(32, activation='relu')(input_a)
```

```
hidden_b = Dense(32, activation='relu')(input_b)
```

```
# Merge both branches
```

```
merged = concatenate([hidden_a, hidden_b])
```

```
output = Dense(1, activation='sigmoid')(merged)
```

```
# Create model
```

```
model = Model(inputs=[input_a, input_b], outputs=output)
```

```
# Compile the model
```

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

```
# Summary of the model
```

```
model.summary()
```

✓ **Best for models with multiple inputs and outputs.**

### 3 Keras CNNs (Computer Vision)

Convolutional Neural Networks (CNNs) are used for **image classification, object detection, and facial recognition**.

#### 3.1 Example: CNN for Image Classification

```
from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Define CNN model
model = Sequential([
    Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(64,64,3)),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Model summary
model.summary()
```

✓ **Used in Face Recognition, Object Detection, and Medical Imaging.**

### 4 Keras NLP (Text Processing & LSTMs)

Keras provides tools for **Natural Language Processing (NLP)** such as **word embeddings and LSTMs**.

#### 4.1 Example: Using Embedding Layer for Text Classification

```
python
CopyEdit

from keras.models import Sequential

from keras.layers import Embedding, LSTM, Dense
```

```
# Define NLP model
model = Sequential([
    Embedding(input_dim=10000, output_dim=128, input_length=100),
    LSTM(64),
    Dense(1, activation='sigmoid')
])

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

# Model summary
model.summary()

✓ Used in Chatbots, Sentiment Analysis, and Speech Recognition.
```

---

## 5 Keras Transfer Learning

Transfer learning allows **using pretrained models** like **VGG16, ResNet, MobileNet** for image classification.

### 5.1 Example: Using MobileNetV2 for Transfer Learning

```
python
CopyEdit
from keras.applications import MobileNetV2
from keras.models import Model
from keras.layers import Flatten, Dense

# Load pretrained MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224,224,3))

# Freeze the base model layers
for layer in base_model.layers:
```

```

layer.trainable = False

# Add new layers
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
output_layer = Dense(10, activation='softmax')(x)

# Create model
model = Model(inputs=base_model.input, outputs=output_layer)

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

# Model summary
model.summary()

```

✓ **Used in Image Recognition, Object Detection, and Medical Diagnosis AI.**

---

## ❏ Keras Hyperparameter Tuning

Tuning hyperparameters like **learning rate, batch size, and number of layers** improves model performance.

### 6.1 Example: Hyperparameter Tuning with Keras Tuner

```

python
CopyEdit
import keras_tuner as kt
from keras.models import Sequential
from keras.layers import Dense

# Function to build the model
def build_model(hp):
    model = Sequential()

```

```

    model.add(Dense(hp.Int('units', min_value=32, max_value=256, step=32),
activation='relu', input_shape=(784,)))

    model.add(Dense(10, activation='softmax'))

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

    return model

```

```
# Define tuner
```

```
tuner = kt.RandomSearch(build_model, objective='val_accuracy', max_trials=5)
```

```
# Run the search
```

```
tuner.search(train_images, train_labels, epochs=10, validation_split=0.2)
```

✓ Used to find the best hyperparameters automatically.

## 📌 Keras Callbacks & Model Saving

Callbacks help **control training** and **save the best model**.

### 7.1 Example: Using Callbacks

```
python
```

```
CopyEdit
```

```
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

```

callbacks = [

    EarlyStopping(monitor='val_loss', patience=5),

    ModelCheckpoint('best_model.h5', save_best_only=True)

]

```

```
# Train model with callbacks
```

```

model.fit(train_images, train_labels, epochs=50, batch_size=32, validation_split=0.2,
callbacks=callbacks)

```

✓ Used to prevent overfitting and save the best model.

### 7.2 Saving and Loading a Model

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

# Load model
from keras.models import load_model
loaded_model = load_model('my_model.h5')

# Check model summary
loaded_model.summary()
```

✓ Ensures models can be reused without retraining.

---

### 8 Full Example: Building & Training a Keras Model

Let's build and train a complete deep learning model using **Keras Sequential API**.

```
python
CopyEdit
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.datasets import mnist
from keras.utils import to_categorical

# Load the dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize images
train_images, test_images = train_images / 255.0, test_images / 255.0

# Convert labels to categorical format
train_labels = to_categorical(train_labels, 10)
```

```
test_labels = to_categorical(test_labels, 10)

# Define model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Train model
model.fit(train_images, train_labels, epochs=5, batch_size=32, validation_split=0.2)

# Evaluate model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_acc}')
```

✓ Used for digit classification in the MNIST dataset.

---

## 📌 Conclusion

- 🚀 Keras makes deep learning easy and efficient.
- 🚀 Supports Sequential & Functional API for different needs.
- 🚀 Used in NLP, Computer Vision, and Transfer Learning.
- 🚀 Helps in Hyperparameter Tuning, Callbacks, and Model Saving.

🚀 Keras is the go-to deep learning library for AI & ML!

Would you like a tutorial on **deploying a Keras model in production**? 😊🚀

O