1. What is Keras?

- **Open-source library** for deep learning in Python.
- Developed by **François Chollet**.
- Provides a **high-level API** → easy to build and train deep learning models.
- Originally supported multiple backends (Theano, CNTK, TensorFlow).
- Now, **TensorFlow's official high-level API (tf.keras)**.

Think of Keras as a shortcut to deep learning. Instead of writing long math-heavy code, you just "stack layers like Lego blocks."

2. Keras Architecture

User → Keras API (Frontend) → Backend (TensorFlow)

- **Frontend**: Clean functions & classes (Dense, Conv2D, Sequential).
- **Backend**: Handles heavy computations (matrix multiplications, GPU/TPU).

3. Model Building Approaches

(a) Sequential API – Simple & Linear

Layers are stacked in order. Best for beginners.

```
from tensorflow.keras import Sequential, layers

model = Sequential([

    layers.Dense(64, activation='relu', input_dim=100),

    layers.Dense(10, activation='softmax')

])
```

(b) Functional API – Flexible

Supports multiple inputs, outputs, and non-linear connections.

```
from tensorflow.keras import layers, Model

inputs = layers.Input(shape=(100,))

x = layers.Dense(64, activation='relu')(inputs)

outputs = layers.Dense(10, activation='softmax')(x)

model = Model(inputs, outputs)
```

(c) Model Subclassing – Full Control

For custom research models.

```
from tensorflow.keras import Model, layers

class MyModel(Model):

    def __init__(self):

        super().__init__()

        self.d1 = layers.Dense(64, activation='relu')

        self.d2 = layers.Dense(10, activation='softmax')

    def call(self, x):

        return self.d2(self.d1(x))
```

4. Model Workflow in Keras

1. **Build Model** → Add layers.
2. **Compile** → Select optimizer, loss, and metrics.
3. **Train (fit)** → Learn from training data.
4. **Evaluate** → Test on unseen data.
5. **Predict** → Classify new inputs.
6. **Save/Load** → Reuse trained models.

5. Example: Training a Classifier

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(X_train, y_train,
                    epochs=10,
                    validation_data=(X_val, y_val))
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Accuracy:", test_acc)
```

6. Common Layers

Layer	**Purpose**	**Example**
Dense	Fully connected layer	Dense(64, activation='relu')
Conv2D	Extract features from images	Conv2D(32, (3,3), activation='relu')
MaxPooling2D	Reduce spatial size	MaxPooling2D((2,2))
Dropout	Prevent overfitting	Dropout(0.5)
LSTM/GRU	Sequence (text, time-series)	LSTM(128)

7. Model Saving

(a) HDF5 Format (Single file)

```
model.save("model.h5")
loaded = keras.models.load_model("model.h5")
```

(b) SavedModel Format (Folder with metadata)

```
model.save("my_model")
loaded = tf.saved_model.load("my_model")
```

💡 *HDF5 is simple, but SavedModel is better for deployment (mobile, web, TensorFlow Serving).*

8. Pre-trained Models (Transfer Learning)

```
from tensorflow.keras.applications import ResNet50

resnet = ResNet50(weights='imagenet', include_top=False)
```

- **Transfer Learning**: Freeze base layers, train only new layers.
- **Fine-tuning**: Train entire network on your dataset.

Saves time & works well with small datasets.

9. Metrics in Keras

- **Classification**:
  - Accuracy → % correct predictions.
  - Precision → How many predicted positives are correct.
  - Recall → How many actual positives were captured.
  - F1-score → Balance of precision & recall.
  - AUC-ROC → Ability to distinguish classes.
- **Regression**:
  - MAE (Mean Absolute Error).
  - MSE (Mean Squared Error).
  - RMSE (Root Mean Squared Error).

Custom Accuracy Metric

Sometimes we want to define our own metric (instead of using built-ins).

We can do this by subclassing keras.metrics.Metric.

```
import tensorflow as tf

from tensorflow.keras.metrics import Metric
```

```
class AccuracyMetric(Metric):
    def __init__(self, name='accuracy', **kwargs):
        super().__init__(name=name, **kwargs)
        self.correct = self.add_weight(name='correct', initializer='zeros')
        self.total = self.add_weight(name='total', initializer='zeros')

    def update_state(self, y_true, y_pred, sample_weight=None):
        y_true_idx = tf.argmax(y_true, axis=-1)
        y_pred_idx = tf.argmax(y_pred, axis=-1)
        matches = tf.cast(tf.equal(y_true_idx, y_pred_idx), self.dtype)
        self.correct.assign_add(tf.reduce_sum(matches))
        self.total.assign_add(tf.cast(tf.size(y_true_idx), self.dtype))

    def result(self):
        return self.correct / self.total

    def reset_states(self):
        self.correct.assign(0.0)
        self.total.assign(0.0)
```

Usage:

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

This works exactly like accuracy, but you have full control (can customize for weighted accuracy, top-k accuracy, etc.).

10. Visualization

(a) Training Curves

```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label="Train Loss")

plt.plot(history.history['val_loss'], label="Val Loss")

plt.legend()

plt.show()
```

(b) TensorBoard (Interactive)

```
from tensorflow.keras.callbacks import TensorBoard

tb = TensorBoard(log_dir='./logs')

model.fit(X_train, y_train, epochs=5, callbacks=[tb])
```

Run in terminal: tensorboard --logdir=./logs

\

Q&A – Quick Revision

Q1: Why was Keras chosen as TensorFlow's official high-level API?

A. Because it is simple, consistent, and user-friendly while still allowing advanced customization.

Q2: What is the difference between Sequential API and Functional API?

A. Sequential is for linear models (one input, one output).

Functional API supports complex architectures (multiple inputs/outputs, branching).

Q3: Why do we need to compile a model before training?

A. To define optimizer, loss function, and metrics. Without this, the model doesn't know how to update weights.

Q4: When should we use a Dense layer vs. a Convolutional layer?

A. Dense → tabular/classification tasks where all inputs matter equally.

Convolutional → image/video tasks where spatial features matter.

Q5: What is the difference between training loss and validation loss?

A. Training loss = model performance on training data.

Validation loss = model performance on unseen validation data.

Q6: Why is F1-score better than Accuracy for imbalanced datasets?

A. Because it balances precision and recall, while accuracy can be misleading if one class dominates.

Q7: What's the advantage of Transfer Learning?

A. Uses pre-trained models, reduces training time, improves accuracy with limited data.

Q8: Why do we use Dropout layers?

A. To prevent overfitting by randomly disabling neurons during training.

Keras Practice Exercises

Exercise 1: Common Functional Modules

Build a simple feed-forward neural network using **Sequential API** with the following layers:

- Input: 20 features
- Hidden Layer: 64 neurons (ReLU)
- Output Layer: 3 classes (Softmax)

Tasks:

1. Print the **model summary**.
2. Train it on dummy data (X_train, y_train) for 5 epochs.
3. Evaluate accuracy on a test split.

Exercise 2: Model Configuration, Training, and Testing

Use the **MNIST dataset** (handwritten digits).

Tasks:

1. Load the dataset from keras.datasets.mnist.
2. Preprocess the data (normalize images, one-hot encode labels).
3. Build and **compile** a Sequential CNN with:
    - Conv2D → MaxPooling → Flatten → Dense layers.
4. Train for 10 epochs, plot **training vs validation loss** using Matplotlib.
5. Report final **test accuracy**.

Exercise 3: Model Saving & Loading

Extend Exercise 2.

Tasks:

1. Save the trained CNN model in both:
    - model.h5 (HDF5 format)
    - SavedModel format
2. Delete the current model and reload it from disk.
3. Show that predictions are the same before and after saving.

Exercise 4: Custom Network & Accuracy Metric

Create a **custom accuracy metric** using keras.metrics.Metric.

Tasks:

1. Implement a metric class MyAccuracy that computes accuracy.
2. Use it in model.compile() along with the optimizer & loss.
3. Train a small model (like on Iris dataset).
4. Compare results between built-in accuracy vs custom MyAccuracy.

Exercise 5: Model Zoo & Visualization

Use a **pre-trained ResNet50 model** from keras.applications.

Tasks:

1. Load ResNet50 with include_top=False.
2. Freeze its layers and add your own classifier (Dense layer with 5 classes).
3. Train on a small image dataset (CIFAR-10 / any available).
4. Use **TensorBoard** to visualize training progress.
5. Experiment with **fine-tuning** (unfreeze last 10 layers) and compare accuracy.