# Module 2

**Advanced CNNs in Keras**

**Advanced CNNs and Data Augmentation**

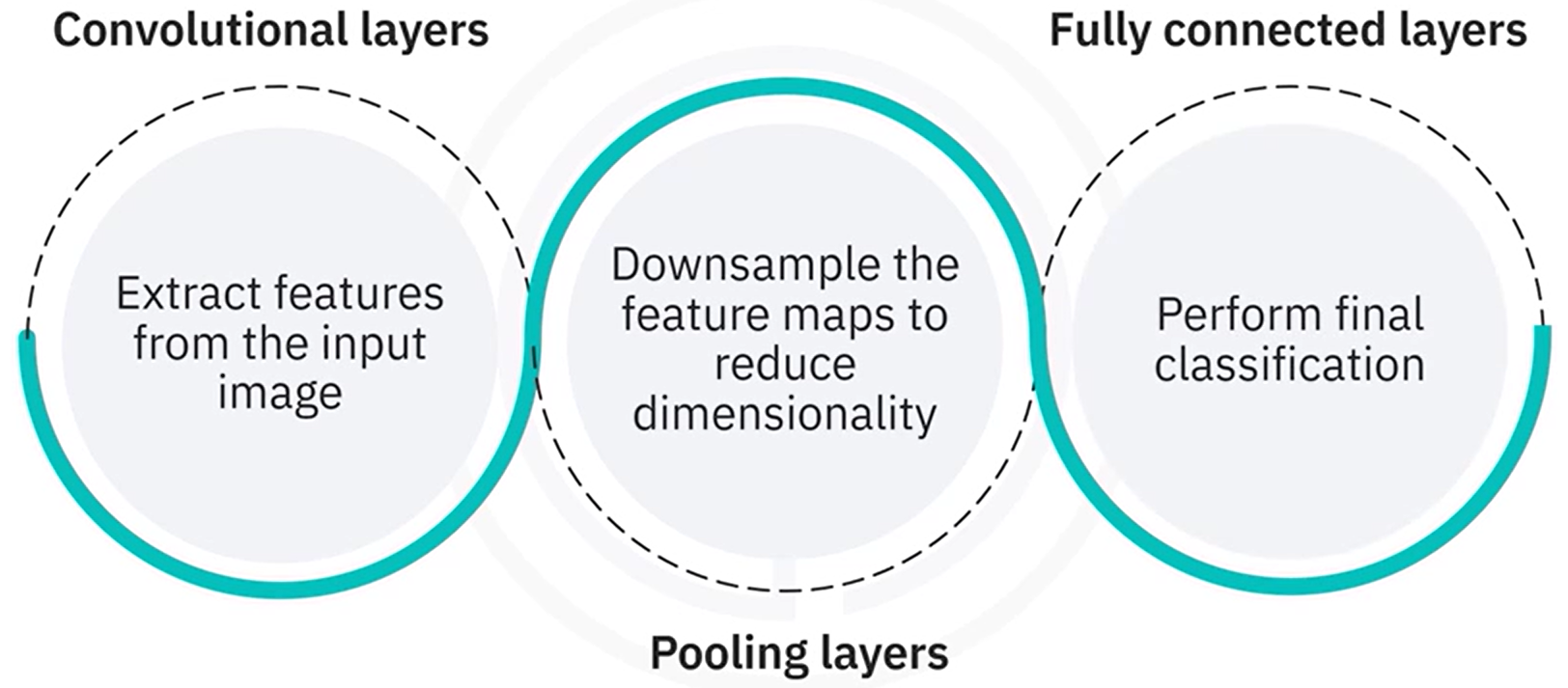
## 📌 Advance CNNs in keras

This section focuses on implementing **advanced convolutional neural network architectures** in Keras to improve performance on complex visual tasks. It builds on basic CNN knowledge and introduces architectural innovations used in state-of-the-art models like **VGG** and **ResNet**..

### 🔹 Understanding CNN Architecture

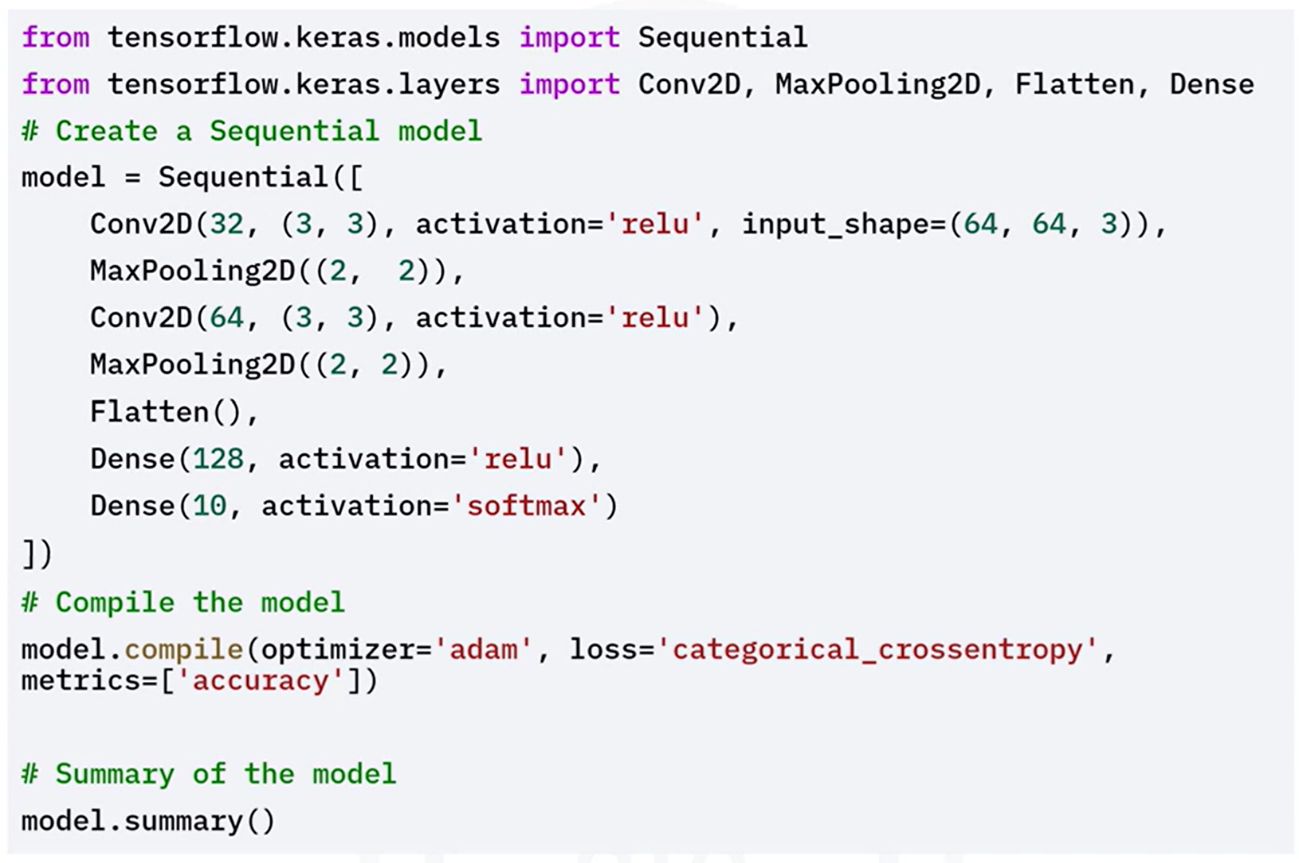
Convolutional Neural Networks (CNNs) mimic the human visual system by processing images through multiple layers:

* **Convolutional Layers**: When designing new algorithms or techniques that are not yet part of the Keras API.
* **Pooling Layers**: When an application requires a specific mathematical operation or architectural element that isn’t supported natively.
* **Fully Connected Layers**: Tailoring layers to better handle specific types of data distributions, memory constraints, or latency-sensitive applications.



The structure below demonstrates a standard CNN stack:



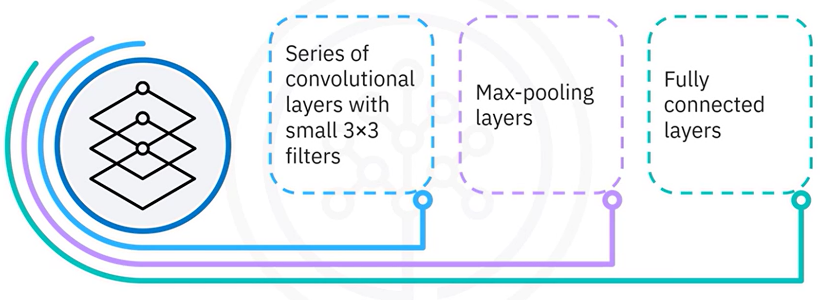


### 🔹 Advanced CNN Architecture

While the basic CNN is a powerful tool, complex tasks demand deeper and more efficient architectures. Popular advanced architectures include:

#### ⚙️VGG-Like Architecture

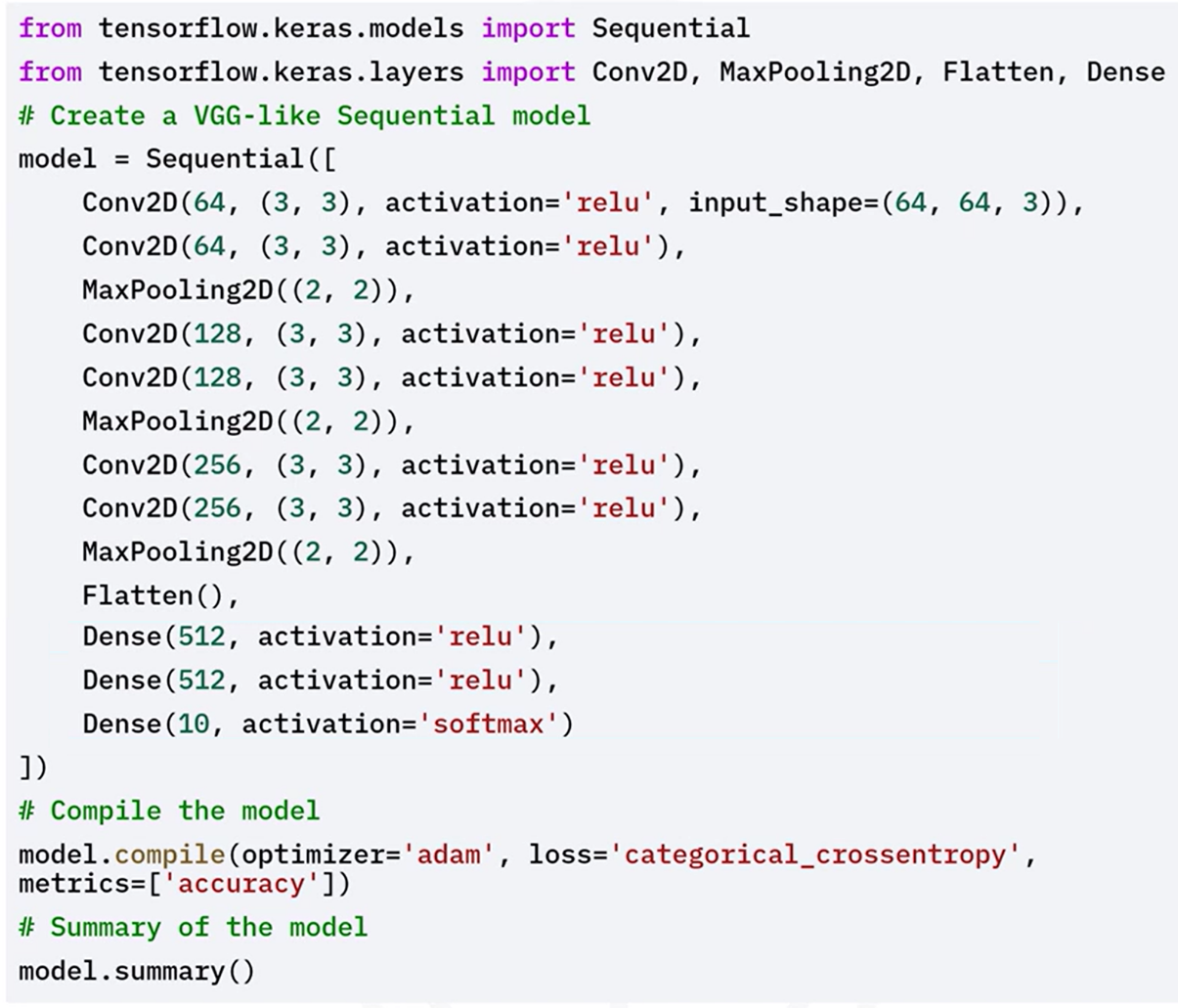
The **VGG architecture** is known for its **depth and simplicity**, relying on small 3×3 convolutional filters and stacking them in increasing depth.

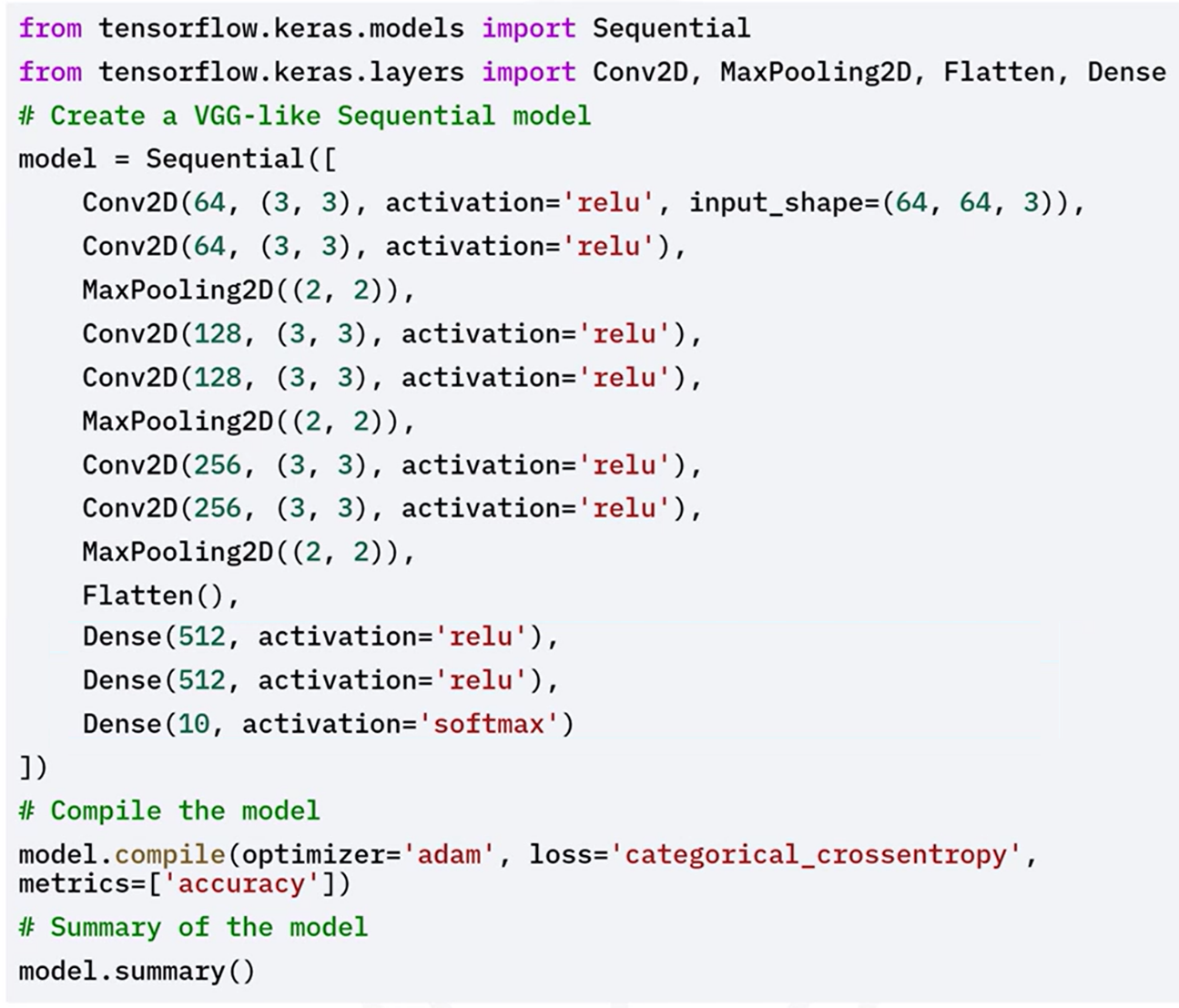
It consists of a series of convolutional layers with small three by three filters, followed by Max-pooling layers and fully connected layers.

**VGG Principles:**

* Use of small filters (3×3)
* Deep stacking of layers
* MaxPooling after each convolutional block

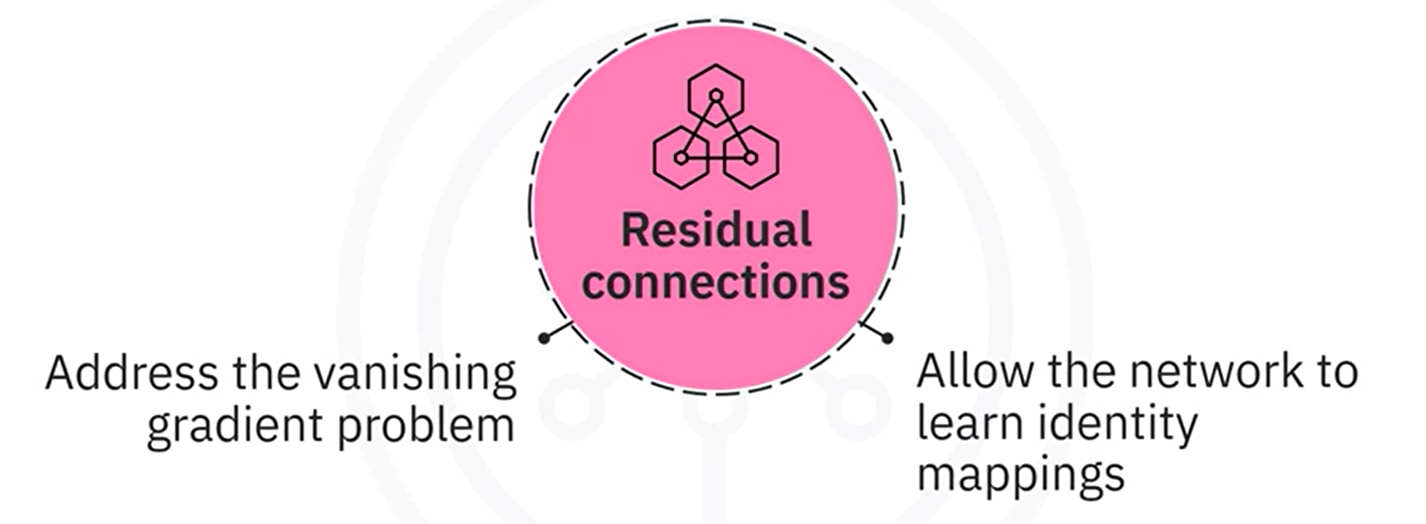
Implementing a VGG-Like Architecture in Keras:





#### ⚙️ResNet-Like Architecture

**ResNet (Residual Network)** introduces **residual connections** to allow deeper network training by solving the **vanishing gradient problem**.



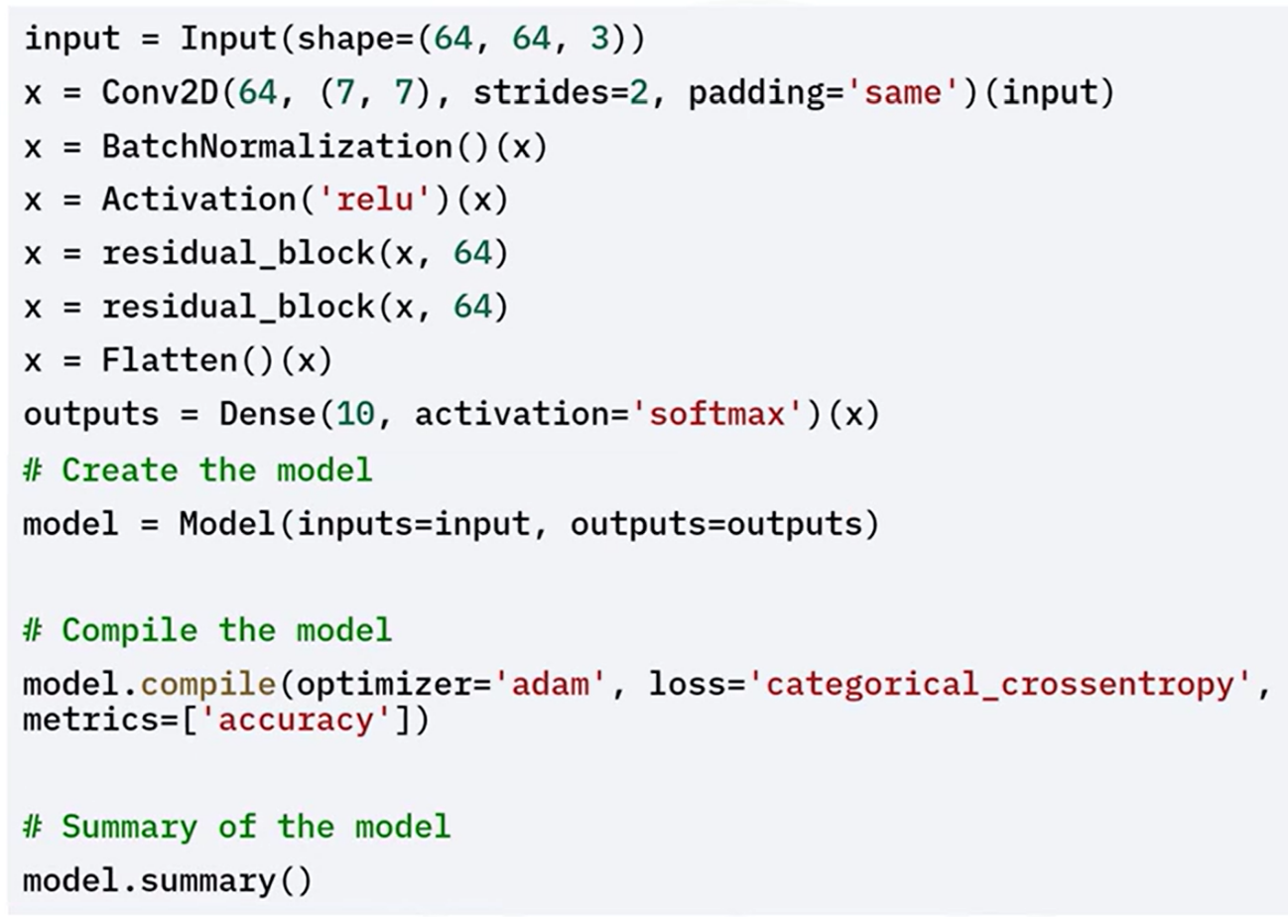
**💡 Key Concepts:**

* **Residual Blocks**: Learn identity mappings using shortcut connections.
* **Improves gradient flow**, enabling very deep networks to converge.

Here’s an example of implementing a ResNet-Like architecture in Keras:

The residual block function defines a residual block with two convolutional layers and a shortcut connection.





### ✅ Takeaways

✅ CNNs consist of convolutional, pooling, and fully connected layers that together extract and interpret visual features.

✅ **Basic CNNs** are foundational, but advanced tasks benefit from deeper and more specialized architectures.

✅ **VGG** networks use small filters with deep stacks for hierarchical feature learning.

✅ **ResNet** networks introduce **residual connections** to overcome training difficulties in deep networks.

✅ Keras enables flexible implementation of both basic and advanced CNNs with clean and modular APIs.

## 📌 Data Augmentation Techniques in Keras

Data augmentation is a **crucial technique** for training **robust and generalizable models** in computer vision. By artificially increasing the diversity of training data through image transformations, models learn to better identify invariant patterns and avoid overfitting.

**🎯 Purpose of Data Augmentation:**

* Introduces **variation** to training inputs without requiring new data.
* Helps models **generalize** to unseen examples by simulating real-world conditions.
* Reduces the risk of **overfitting** by exposing the model to multiple versions of the same image.

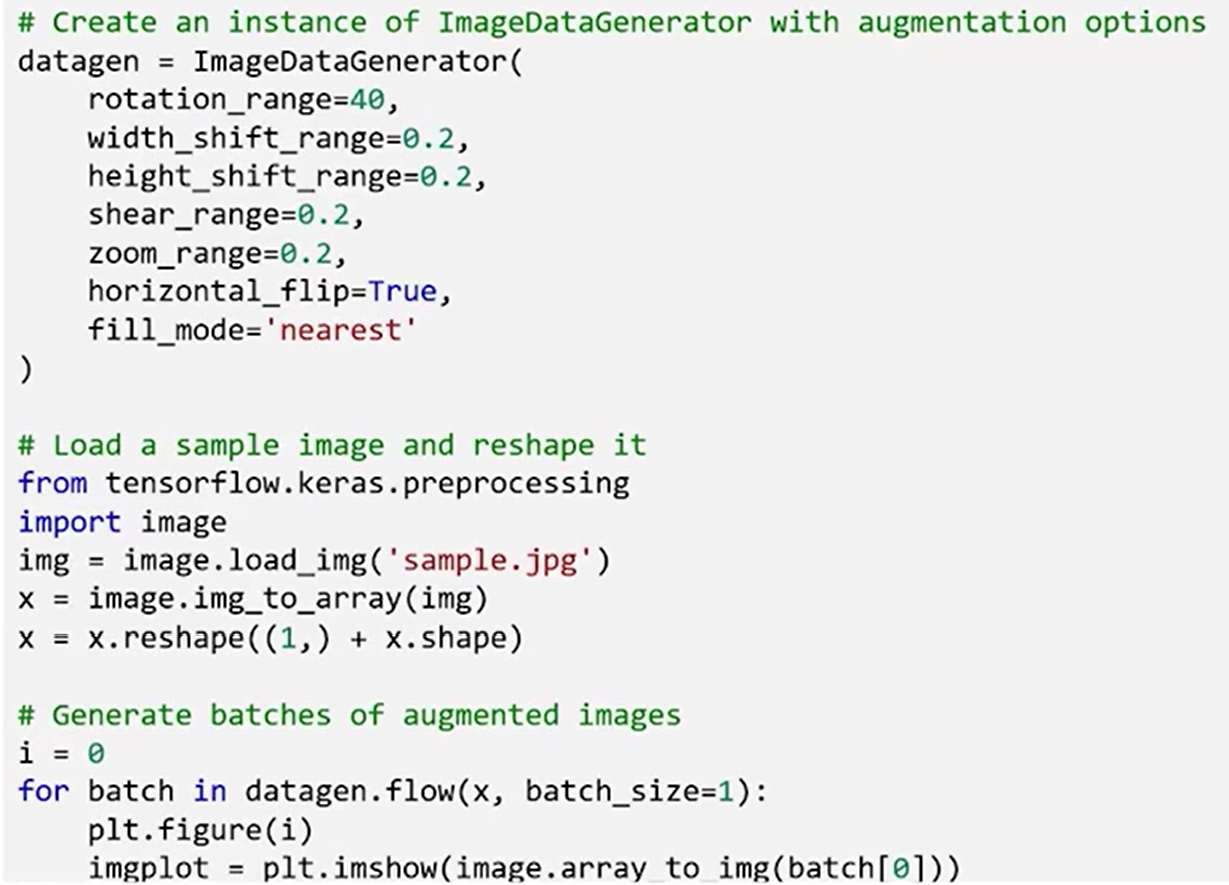
### 🔹 Basic Augmentation Techniques

Keras provides the ***ImageDataGenerator*** class to apply a variety of common image augmentations.

**✅ Available Transformations:**

* **Rotation**: Rotates image by a random angle.
* **Width Shift / Height Shift**: Randomly moves the image along horizontal or vertical axis.
* **Shear**: Applies geometric distortion (shearing).
* **Zoom**: Random zoom-in or zoom-out.
* **Horizontal Flip**: Flips the image left-to-right.
* **Rescaling**: Normalizes pixel values.

Let's see an example of how to apply common augmentations like rotation, width shift, height shift and horizontal flip



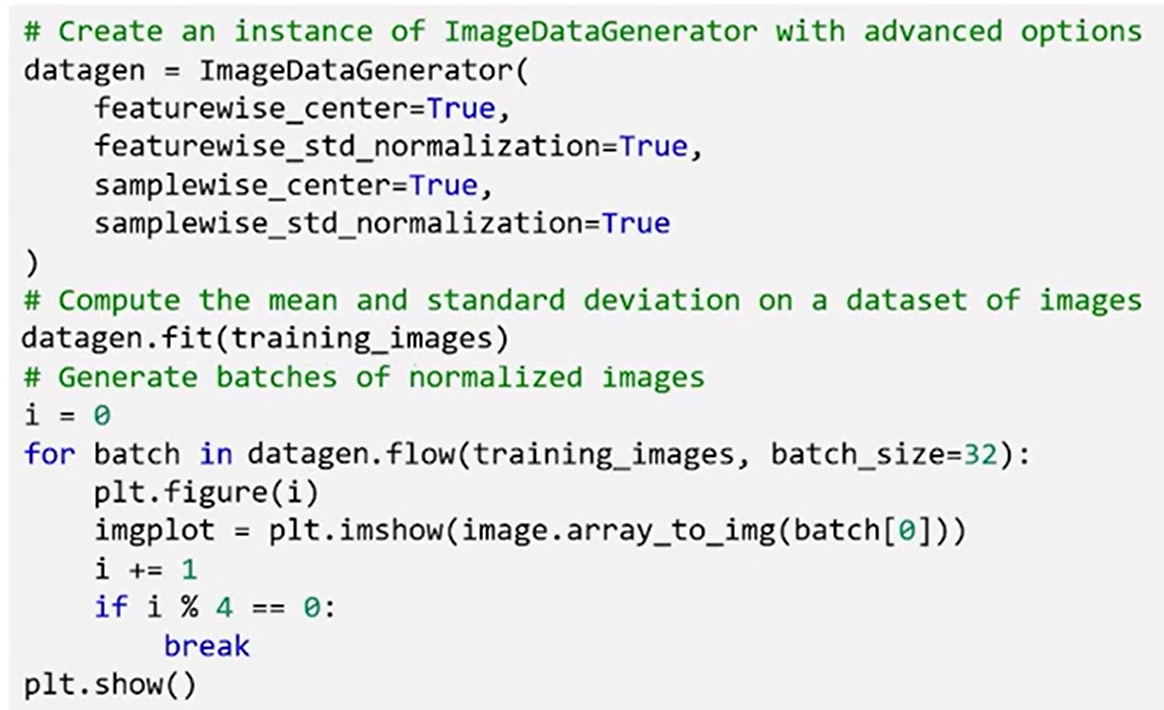
### 🔹 Normalization-Based Augmentation

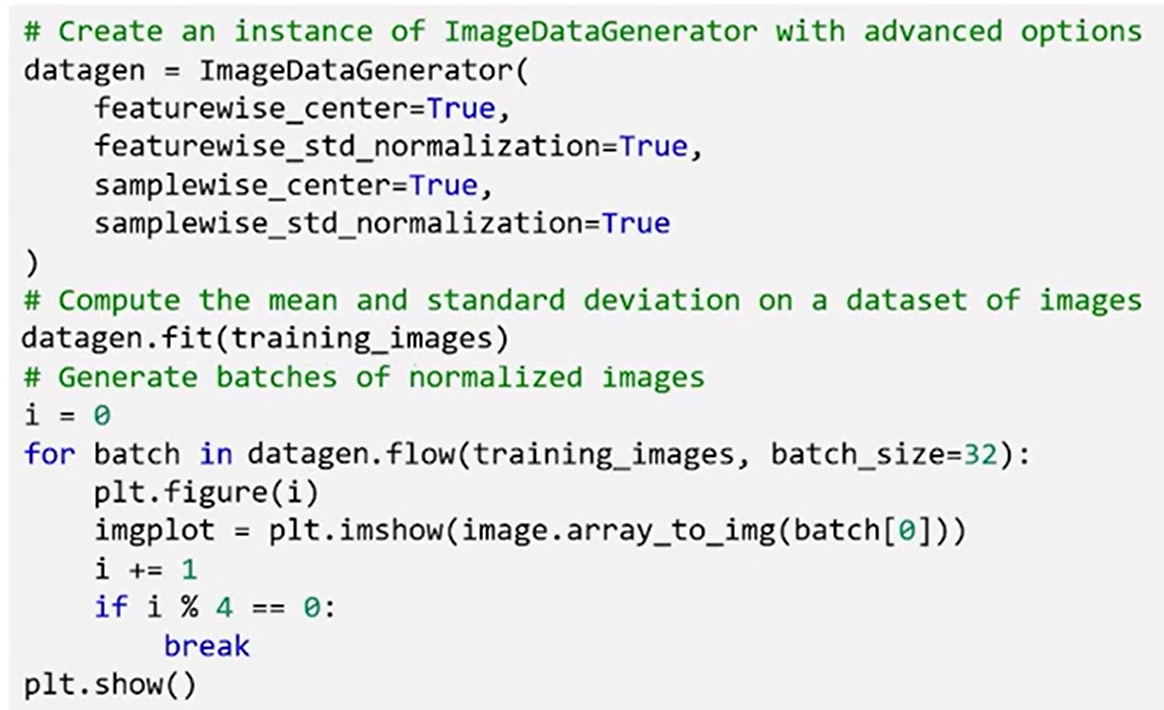
Keras also allows **feature-wise and sample-wise normalization**, which standardizes input images either globally across the dataset or locally for each image.

**🔹 Options:**

* **featurewise\_center:** Set global dataset mean to 0
* **featurewise\_std\_normalization:** Normalize dataset to std = 1
* **samplewise\_center:** Set each image's mean to 0
* **samplewise\_std\_normalization:** Normalize each image to std = 1

Let's see an example of using feature-wise and sample-wise normalization.



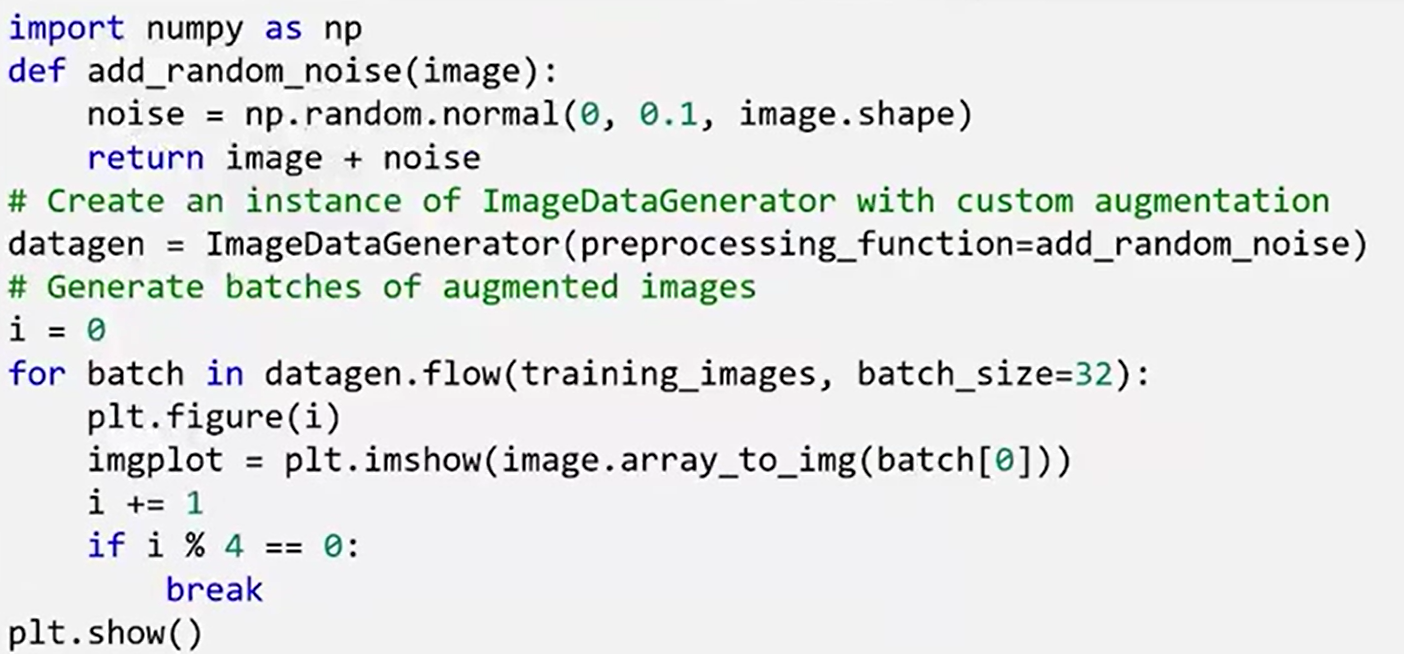


### 🔹 Custom Augmentation Functions

Keras allows developers to define **custom augmentation logic** using the ***preprocessing\_function*** parameter.

This gives complete control over how each image is transformed.

Custom functions can implement domain-specific distortions, simulate hardware artifacts, or apply stochastic noise patterns for robustness.



### ✅ Takeaways

✅ Data augmentation improves **model generalization** by simulating diverse visual conditions.

✅ **ImageDataGenerator** supports essential transformations like rotation, translation, shear, and flip.

✅ **Normalization techniques** (feature-wise or sample-wise) standardize image inputs to stabilize training.

✅ **Custom augmentation functions** unlock flexibility and allow full control over data preprocessing.

✅ These augmentation techniques are critical for **computer vision tasks** involving limited data or high variability.