# Deep Learning Using Python

by

Dayananda Kumar N. C

Project Manager, AI Team

Samsung Electro Mechanics Co. Ltd

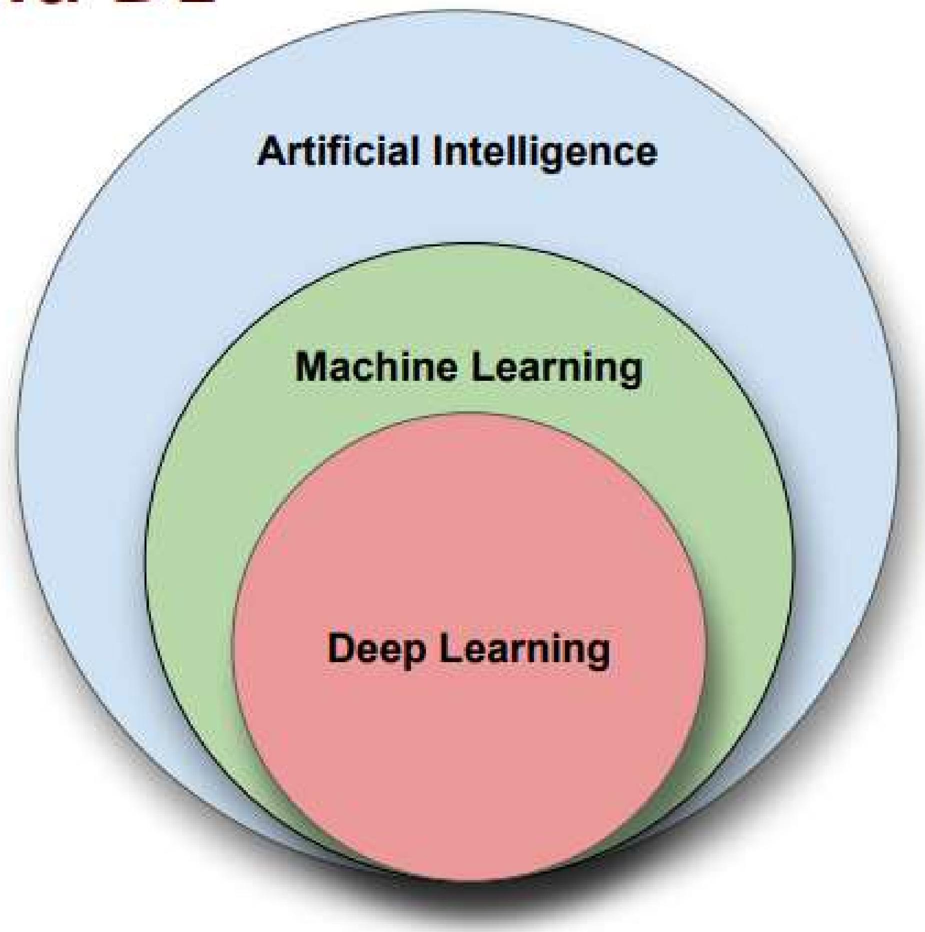
Bengaluru

# OUTLINE

- Introduction to Deep Learning
- CNN Modules
- Python Implementation
- Discussion

# Relationship of AI, ML and DL

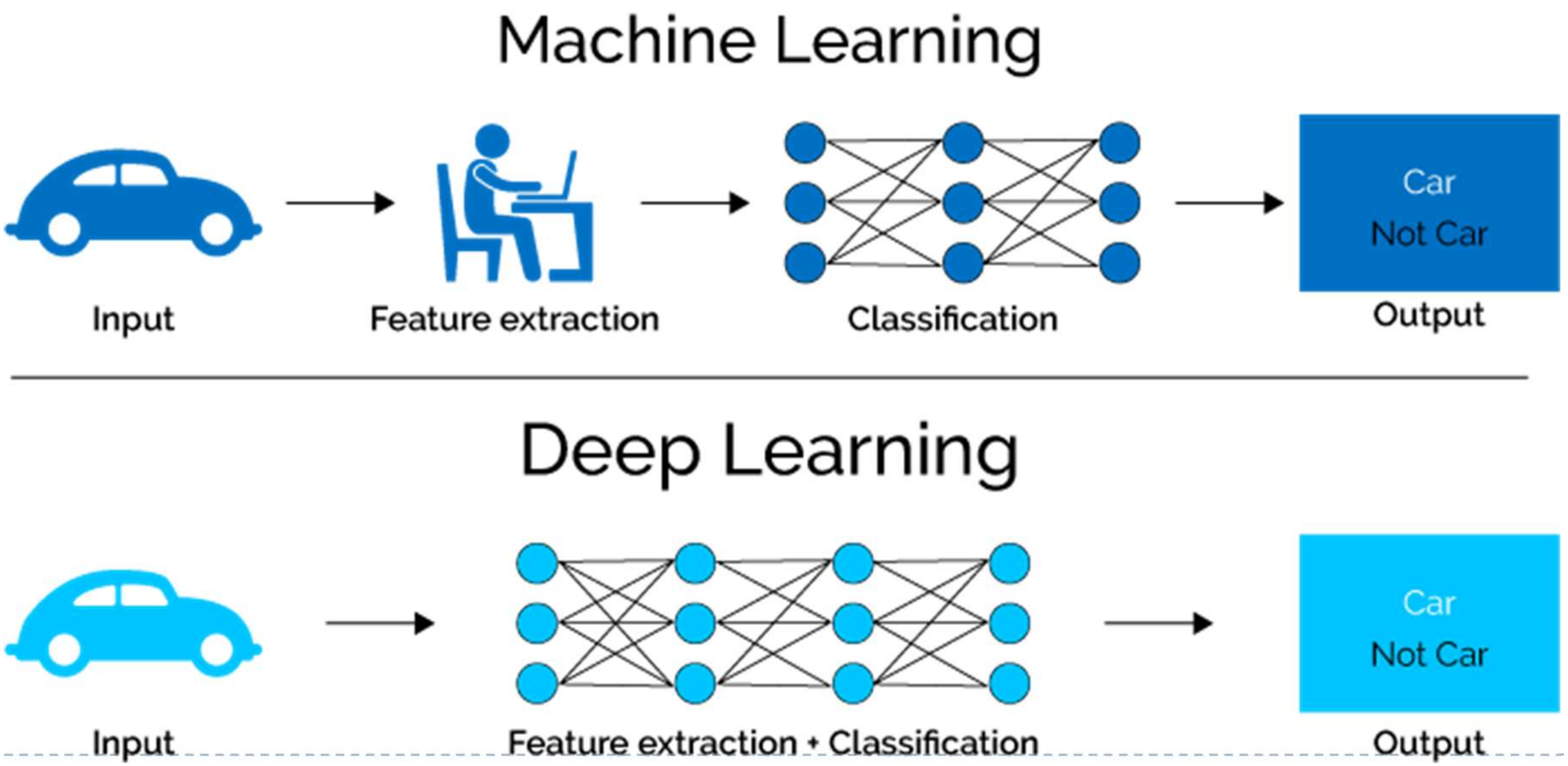
- **Artificial Intelligence (AI)** is anything about man-made intelligence exhibited by machines.
- **Machine Learning (ML)** is an approach to achieve **AI**.
- **Deep Learning (DL)** is one technique to implement **ML**.



# What is Deep Learning (DL) ?

A machine learning subfield of learning **representations** of data.

Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**



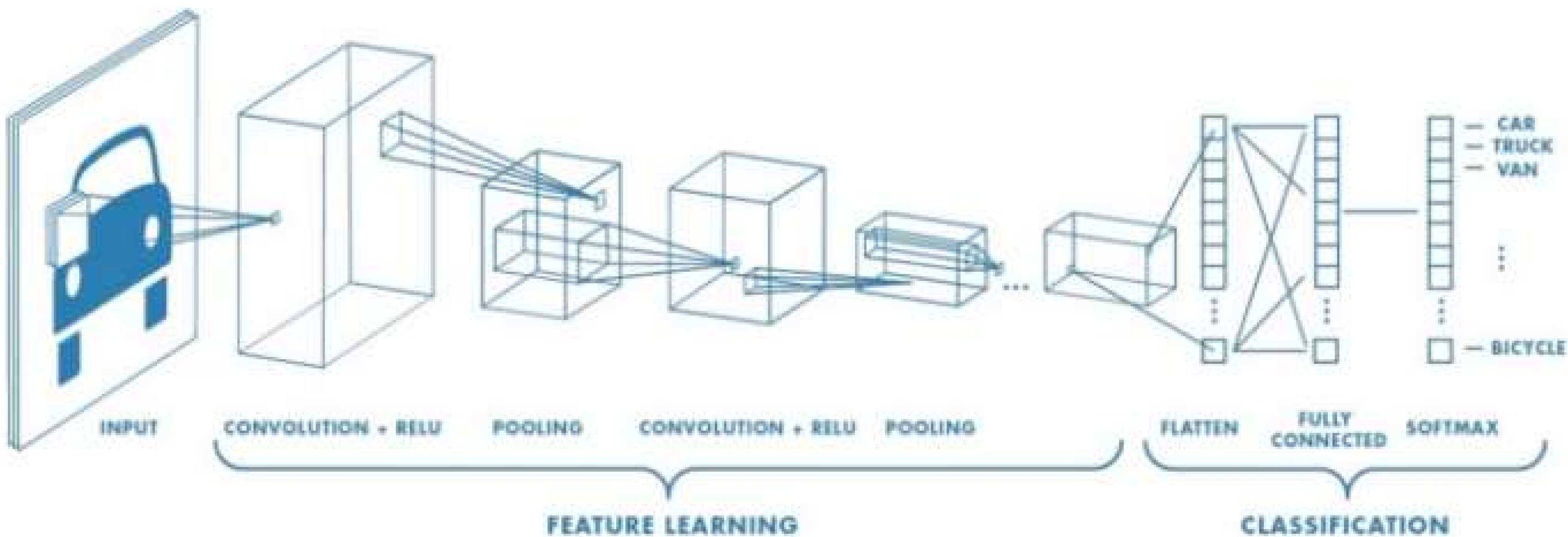
## Why Deep Learning?

- Limitations of traditional machine learning algorithms
  - not good at handling high dimensional data.
  - difficult to do feature extraction and object recognition.

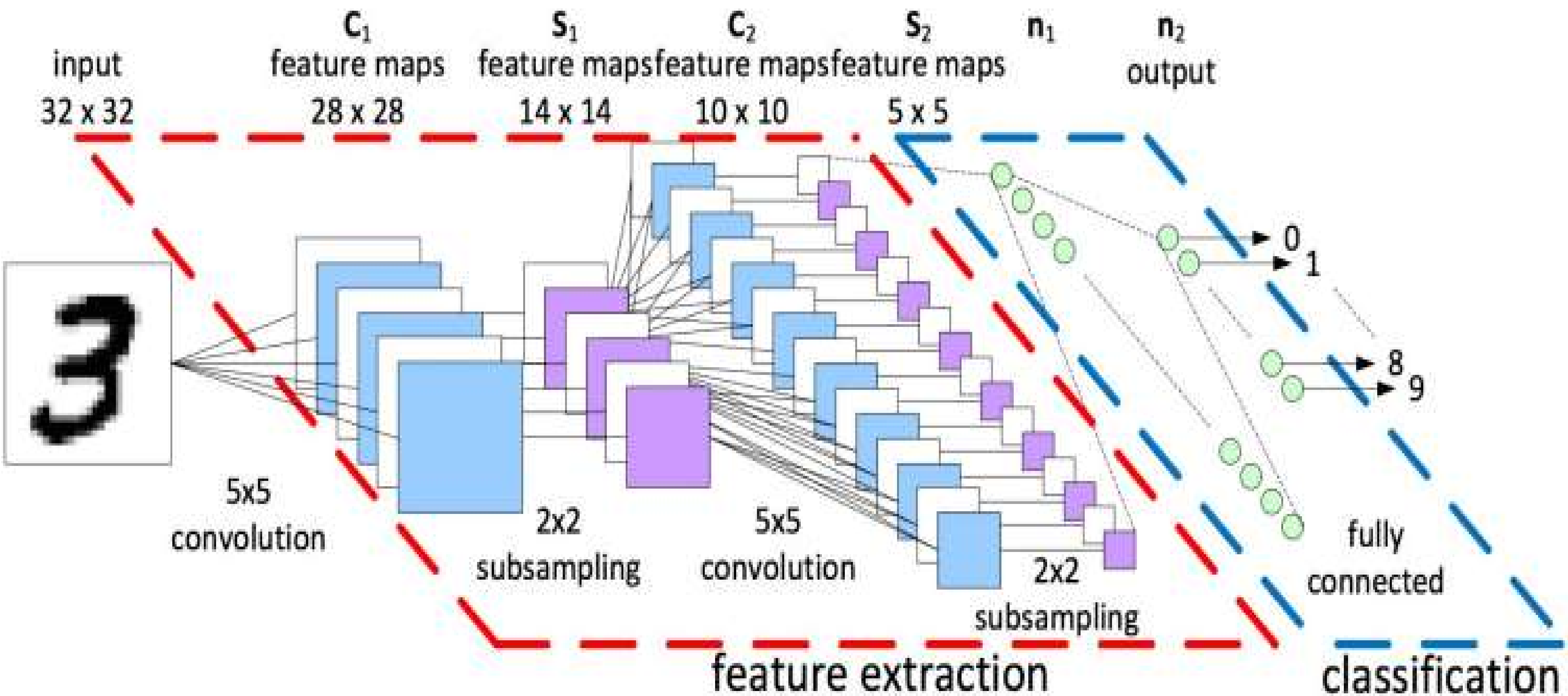
- Advantages of deep learning
  - DL is computationally expensive, but it is capable of handling high dimensional data.
  - feature extraction is done automatically.

# Feature Extraction and Classification using CNN

- Deep Convolutional Neural Network (CNN) consists series of convolution layers with learnable filters (Kernels), activation, pooling, fully connected layers and apply SoftMax function to classify an object with probabilistic values between 0 and 1.
- The below figure is a complete flow of CNN to process an input image for classification task.



# Conv-Net Representation

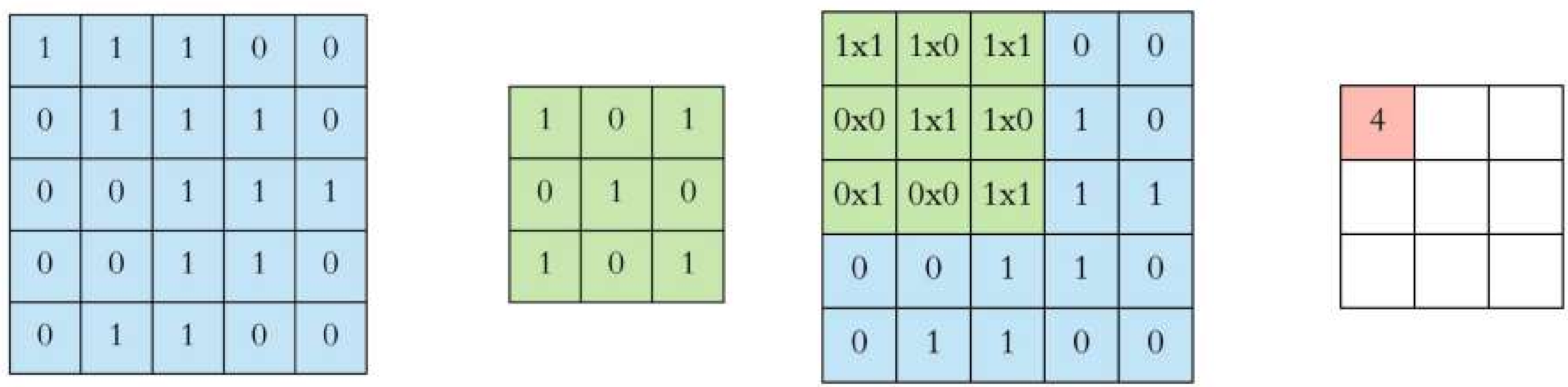


# Data Initialization Code

```
# Imports and Setup

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

# Create a sample input image (6x6 grayscale)

image = np.array([
    [1, 2, 1, 0, 0, 0],
    [4, 5, 2, 0, 0, 0],
    [1, 2, 1, 0, 0, 0],
    [0, 1, 3, 2, 2, 0],
    [0, 1, 4, 4, 4, 0],
    [0, 1, 2, 3, 3, 0]
], dtype=np.float32)


# Reshape for TF model: (batch_size, height, 
width, channels)

image_tf = image.reshape((1, 6, 6, 1))
```

# Convolution

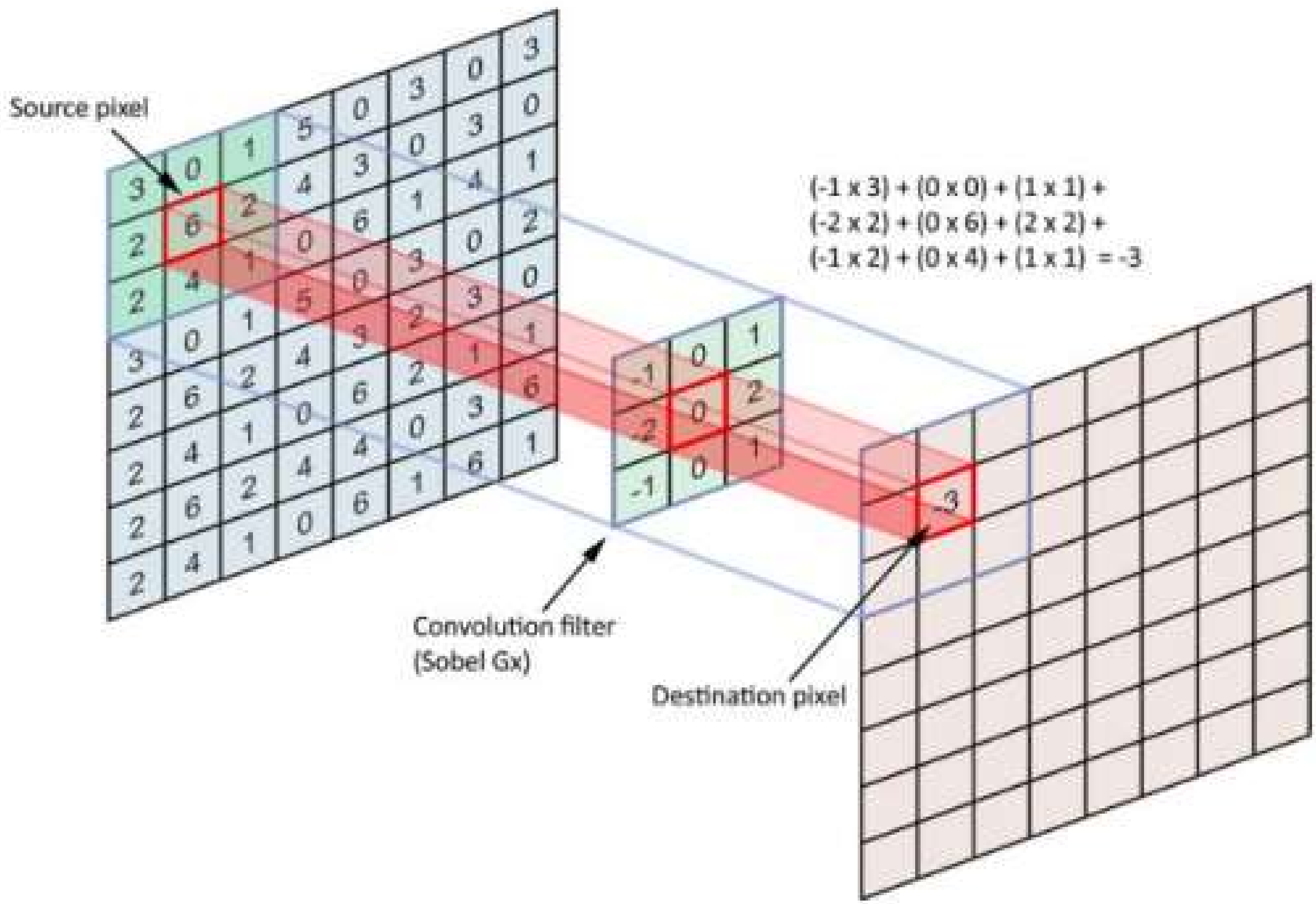
- Convolution layer - extract features from an input image. Preserves the relationship between pixels by learning image features using small patch of input data.




Input Filter / Kernel

- Stride - number of pixels shifts over the input matrix. when the stride is 1 then move the filters to 1 pixel at a time. When the stride is 2 then move the filters to 2 pixels at a time and so on.

# Convolution in 2D

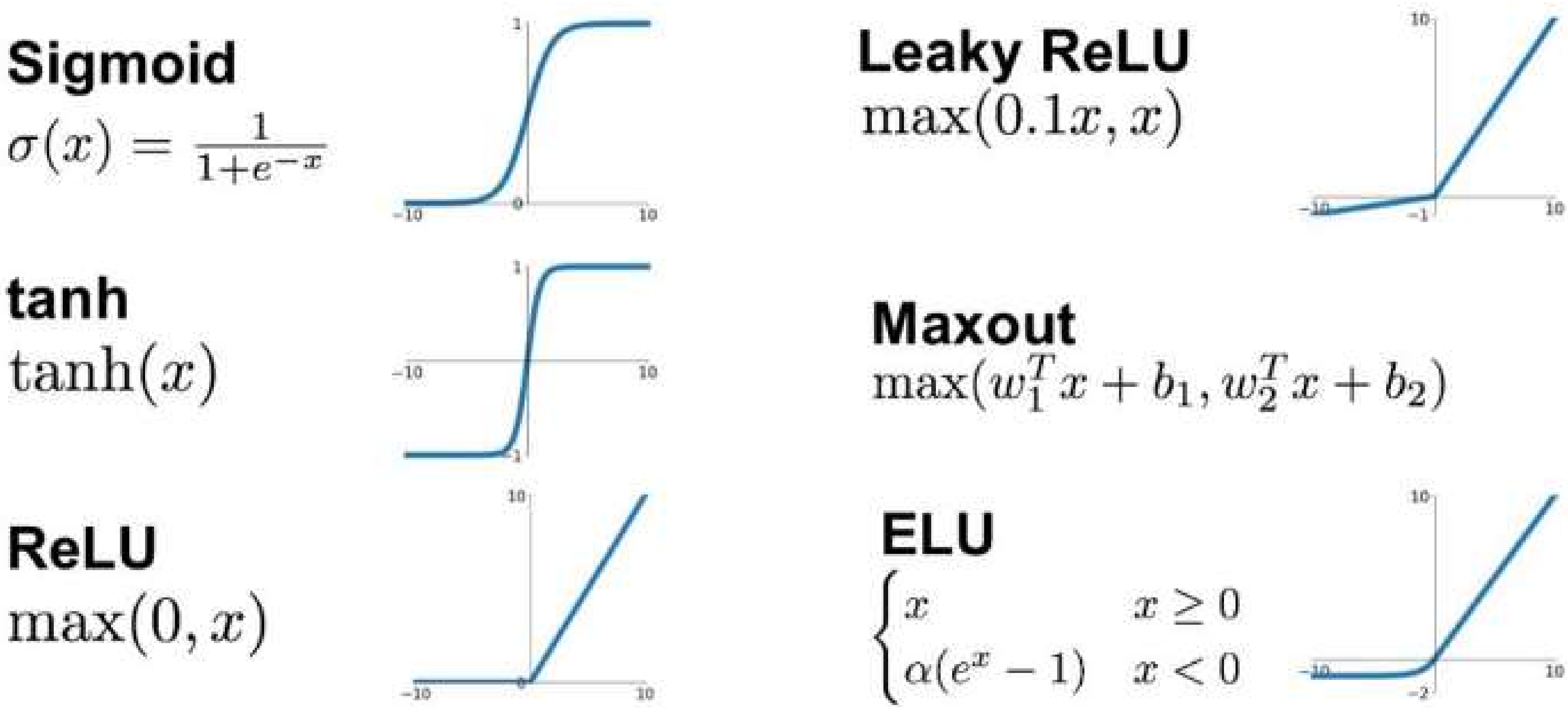


# Conv2D Code

```
# Conv2D Layer

conv_layer = layers.Conv2D(filters=1,
kernel_size=3, strides=1, padding='valid',
use_bias=False)

model_conv = models.Sequential([conv_layer])

conv_output = model_conv(image_tf)

plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plt.title("Input Image")
plt.imshow(image, cmap='gray')

plt.subplot(1, 2, 2)
plt.title("Conv2D Output")
plt.imshow(conv_output[0, :, :, 0], cmap='gray')
plt.show()
```

# Activation

- ReLU introduce non-linearity in Conv Net. Since, the real world data would be non-negative linear values.
- ReLU stands for Rectified Linear Unit for a non-linear operation.



# Activation Code

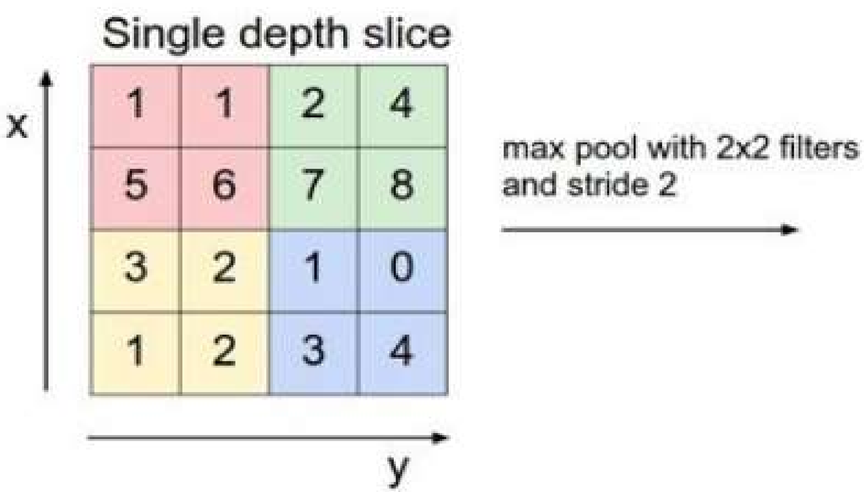
```
# ReLU Activation Layer
relu_layer = layers.ReLU()
model_relu = models.Sequential([conv_layer,
relu_layer])
relu_output = model_relu(image_tf)
plt.title("ReLU Output")
plt.imshow(relu_output[0, :, :, 0], cmap='gray')
plt.show()
```

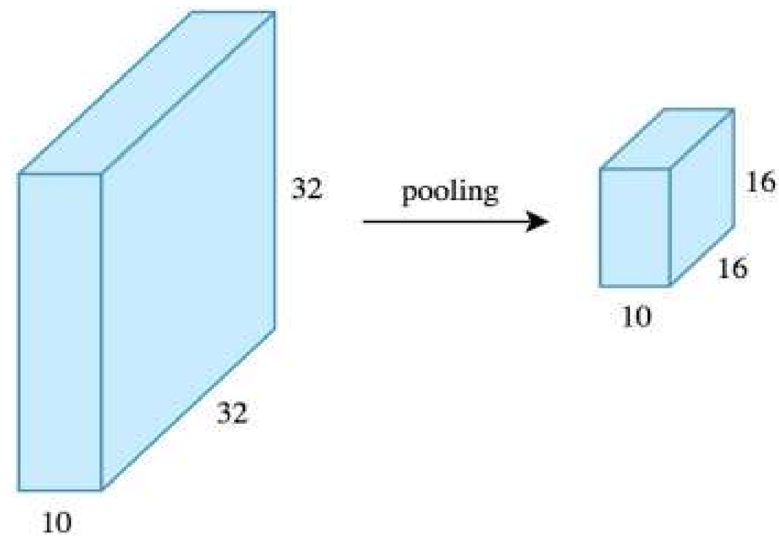
# Pooling

- Pooling layers reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains the important information.

Spatial pooling can be of different types:

- Max Pooling - largest element from the rectified feature map
- Average Pooling - Average of all elements in the feature map
- Sum Pooling - Sum of all elements in the feature map

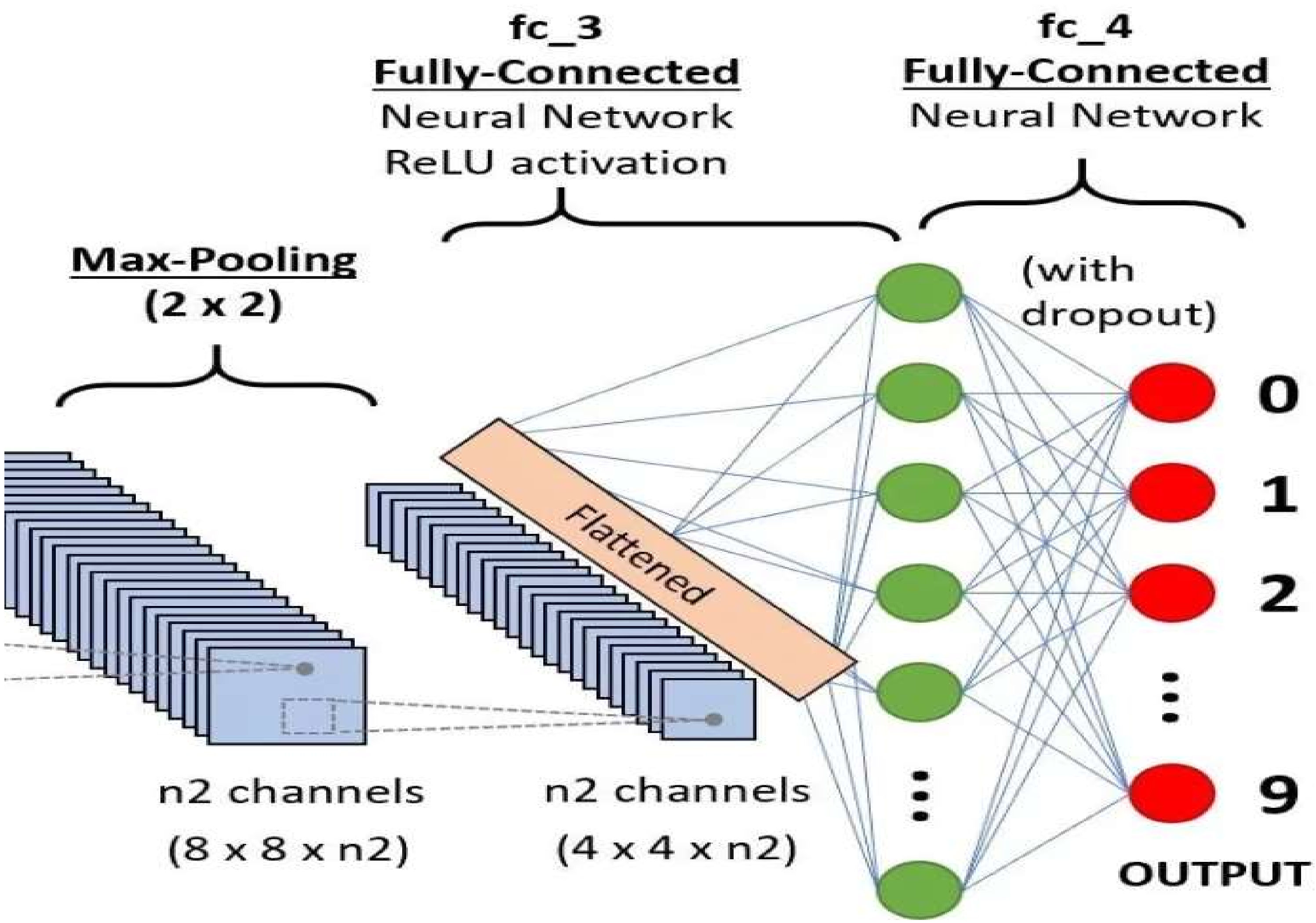


# Pooling Code

```
# MaxPooling2D Layer

pool_layer = layers.MaxPooling2D(pool_size=(2,
2), strides=2)

model_pool = models.Sequential([conv_layer,
relu_layer, pool_layer])

pool_output = model_pool(image_tf)

plt.title("Max Pooling Output")
plt.imshow(pool_output[0, :, :, 0], cmap='gray')
plt.show()
```

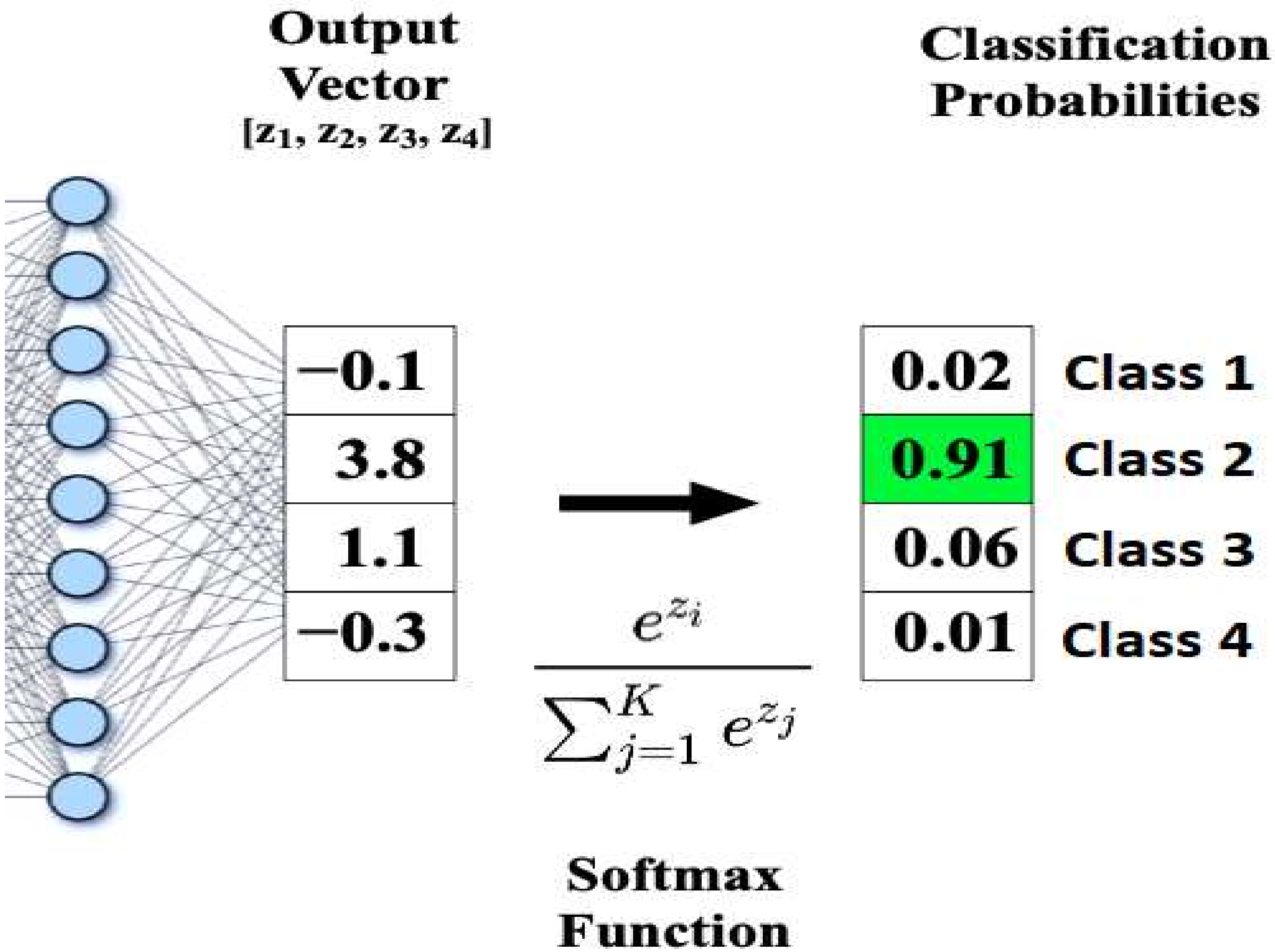
# Fully Connected Layer



# Fully Connected Code

```
# Fully Connected Layer

flatten_layer = layers.Flatten()

dense_layer = layers.Dense(units=3)  # 3 classes

model_fc = models.Sequential([conv_layer,
relu_layer, pool_layer, flatten_layer,
dense_layer])

fc_output = model_fc(image_tf)

print("Fully Connected Output:",
fc_output.numpy())
```

# SoftMax Layer



# Softmax Code

```
# Softmax Layer
softmax_layer = layers.Softmax()

model_softmax = models.Sequential([
    conv_layer, relu_layer, pool_layer,
    flatten_layer, dense_layer, softmax_layer
])

softmax_output = model_softmax(image_tf)

print("Softmax Probabilities:",
softmax_output.numpy())

# Visualize class probabilities
plt.bar(range(3), softmax_output.numpy()[0])
plt.title("Softmax Output")
plt.xlabel("Class")
plt.ylabel("Probability")
plt.show()
```

# MNIST - Digit classification

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

# 1. Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()

# Normalize images to the range [0,1]
x_train, x_test = x_train / 255.0, x_test / 255.0

# Add channel dimension (batch, height, width, channel)
x_train = x_train[..., np.newaxis]
x_test = x_test[..., np.newaxis]

# One-hot encode labels
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
```

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

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

model.summary()

# 3. Train model
history = model.fit(x_train, y_train_cat, epochs=5,
                    validation_split=0.1, batch_size=64)
```

```python
plt.plot(history.history['accuracy'], label='Train
Acc')
plt.plot(history.history['val_accuracy'], label='Val
Acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training History')
plt.show()
```

```
# 4. Evaluate model on test set
test_loss, test_acc = model.evaluate(x_test, y_test_cat)
print(f"\nTest accuracy: {test_acc:.4f}")

# 5. Run inference on sample test images
pred_probs = model.predict(x_test[:10])
pred_classes = np.argmax(pred_probs, axis=1)

# 6. Visualize predictions
plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test[i].squeeze(), cmap='gray')
    plt.title(f"Pred: {pred_classes[i]}\nTrue: {y_test[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

# IRIS - Classification

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# 1. Load and preprocess the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
class_names = iris.target_names

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# One-hot encode target
y_train_cat = to_categorical(y_train, num_classes=3)
y_test_cat = to_categorical(y_test, num_classes=3)

# 2. Build the MLP model
model = Sequential([
    Dense(10, activation='relu', input_shape=(4,)),
    Dense(8, activation='relu'),
    Dense(3, activation='softmax')
])

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

model.summary()

# 3. Train the model
history = model.fit(X_train, y_train_cat,
                    validation_split=0.1,
                    epochs=50,
                    batch_size=8,
                    verbose=1)
```

```
# 4. Evaluate on test data
test_loss, test_acc = model.evaluate(X_test,
y_test_cat)
print(f"\nTest Accuracy: {test_acc:.4f}")

# Predict
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# 5. Visualization - Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay
(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```

```
# 6. Visualization - Feature Scatter Plot (First 2
features)
plt.figure(figsize=(8, 5))
for i in range(3):
    idx = np.where(y_test == i)
    plt.scatter(X_test[idx, 0], X_test[idx, 1],
label=class_names[i], edgecolors='k')
plt.title("Iris Test Data (First 2 Features)")
plt.xlabel("Sepal Length (standardized)")
plt.ylabel("Sepal Width (standardized)")
plt.legend()
plt.grid(True)
plt.show()
```

# THANK YOU