# Image Classification with R Shiny

## Connecting R and Python

SeongYeon Park

June 6, 2025

Sungshin Women's University

# Outline

1 Introduction

2 CNN

3 CNN Training with CIFAR-10 (Python)

4 Connecting Python Model in R

# Deep Learning

- Learns features automatically from raw data.
- Inspired by the structure of the human brain.
- Consists of:
  - Input Layer – receives raw data
  - Hidden Layers – extract and transform features
  - Output Layer – makes the final prediction
- Each ”neuron” multiplies inputs with weights, applies an activation function, and passes to the next layer.

# Outline

1 Introduction

2 CNN

3 CNN Training with CIFAR-10 (Python)

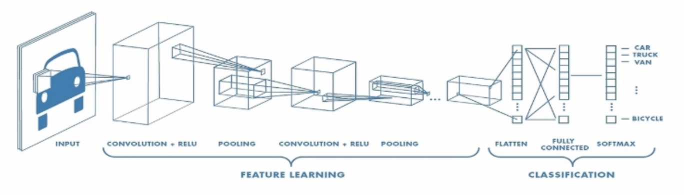
4 Connecting Python Model in R

## CNN (Convolutional Neural Network)

- A type of deep neural network specialized for image data.
- Learns patterns like edges, textures, and shapes directly from images.
- Typically consists of two main stages:
  - **Feature Extraction:** Convolutional and pooling layers learn hierarchical image features.
  - **Classification:** Fully connected layers classify the extracted features into categories.

# CNN (Convolutional Neural Network)

- Key components:
  - **Convolutional Layer** – extracts local features using filters and activation functions (e.g. ReLU)
  - **Pooling Layer** – reduces spatial dimensions and removes redundant information (optional but common)
  - **Fully Connected Layer** – performs the final classification using softmax activation



## Convolutional Layer

- Applies filters (kernels) to extract features from the input image.
- The filter slides over the image (**stride**) — larger strides reduce output size and speed up computation.
- To preserve image size and include edge features, **padding** (often zero-padding) is used.
- As we stack more convolutional layers, feature maps become smaller and more abstract.

# Pooling Layer

- **Pooling layers** reduce the size of feature maps while preserving important information.
- This helps lower computational cost and prevents overfitting.
- Pooling also makes the model more robust to small translations in the image.
- Common types of pooling include:
  - Max Pooling: Selects the maximum value.
  - Average Pooling: Computes the average.

# Fully Connected Layer

- Also called a **Dense Layer**.
- Every neuron is connected to all outputs from the previous layer.
- Combines features to make the final decision, usually at the end of CNNs.
- Flatten layer: Converts feature maps into a 1D vector for classification.
- Dense layer: Each neuron processes all inputs and makes predictions.
- Softmax activation: Outputs class probabilities.

# Outline

1 Introduction

2 CNN

3 CNN Training with CIFAR-10 (Python)

4 Connecting Python Model in R

# CIFAR-10 Dataset

- CIFAR-10 is a widely used benchmark dataset for image classification tasks.
- It contains 60,000 color images,each of size 32 × 32 pixels and with 3 RGB channels.
- The dataset is divided into:
  - 50,000 training images
  - 10,000 test images
- Images are labeled across 10 different categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

# Load Data and Define Model

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import numpy as np

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0


model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
    # Conv2D → MaxPooling2D → Flatten → Dense → Output (Softmax)
])
```

# Compile, Train, Evaluate and Save

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

model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test))

test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)

model.save('cnn_model.h5')
```

# Outline

1 Introduction

2 CNN

3 CNN Training with CIFAR-10 (Python)

4 Connecting Python Model in R

# Load and Use CNN Model via reticulate

```r
library(reticulate)
keras <- import("tensorflow.keras")
np <- import("numpy")

cifar10_labels <- c("airplane", "automobile", "bird", "cat", "deer",
                    "dog", "frog", "horse", "ship", "truck")

model <- keras$models$load_model("cnn_model.h5")

predict_cnn <- function(input_array) {
  # reshape input: 1 sample of 32 x 32 RGB image
  input_array <- array_reshape(input_array, c(1, 32, 32, 3))

  preds <- model$predict(input_array)
  class_id <- np$argmax(preds)
  confidence <- round(np$max(preds) * 100, 2)
  list(class_id = class_id, confidence = confidence)}
```

# Shiny App

```r
library(shiny)
library(magick)

ui <- fluidPage(
  titlePanel("CNN 이미지 분류"),
  sidebarLayout(
    sidebarPanel(
      fileInput("image", "이미지 업로드(jpg/png)",
                accept = c(".png", ".jpg", ".jpeg")),
      actionButton("predict", "예측 실행")
    ),
    mainPanel(
      h4("업로드된 이미지"),
      imageOutput("uploaded_img", height = "200px"),
      hr(),
      h4("예측 결과"),
      verbatimTextOutput("result")
    )))
```

# Shiny app – Image Upload and Prediction Trigger

```r
server <- function(input, output, session) {

  output$uploaded_img <- renderImage({
    req(input$image)
    list(
      src = input$image$datapath,
      contentType = input$image$type,
      width = 200
    )
  }, deleteFile = FALSE)

  observeEvent(input$predict, {
    if (is.null(input$image)) {
      output$result <- renderText({
        "이미지를 먼저 업로드해주세요."
      })
      return()}
```

# Shiny app - Image Preprocessing and Prediction Output

```r
    tryCatch({

      img <- image_read(input$image$datapath)
      img <- image_resize(img, "32x32")

      img_array <- as.integer(img[[1]])
      img_array <- img_array / 255
      dim(img_array) <- c(32, 32, 3)
      img_array <- array(img_array, dim = c(1, 32, 32, 3))

      pred <- predict_cnn(img_array)
      class_id <- pred$class_id
      confidence <- pred$confidence
      class_name <- cifar10_labels[class_id + 1]
```

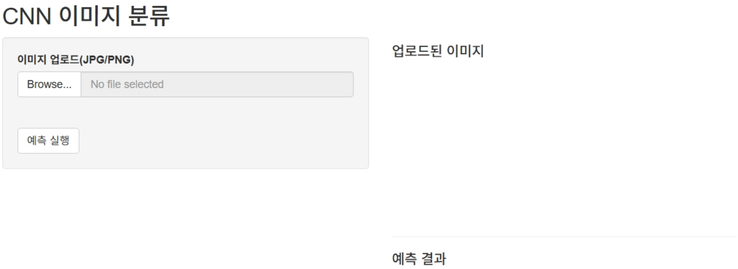
# Shiny app - Image Preprocessing and Prediction Output

```
      output$result <- renderText({
        paste0("예측된 클래스: ", class_id, " (", class_name, ")\n",
               "신뢰도: ", confidence, "%")
      })

    }, error = function(e) {
      output$result <- renderText({
        paste("오류 발생:", e$message)
      })
    })
  })
}

shinyApp(ui, server)
```

# Shiny App - Final Interface



CNN 이미지 분류

이미지 업로드(JPG/PNG)

업로드된 이미지

예측 결과

Users can upload an image, click the predict button, and receive classification results instantly.

# Q & A

# Thank You