Bonus_Lab2_Intro_DL_NN

Sharon Paruwani

2024-11-20

```r
library(reticulate)
```

```
Warning: package 'reticulate' was built under R version 4.4.2
```

```r
use_condaenv("r-tensorflow", required = TRUE)
```

#3.1 Exercise 1: Building a CNN Image Classier with Fashion MNIST Data

```r
library(keras)
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.4.2
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(gridExtra)
```

```
Warning: package 'gridExtra' was built under R version 4.4.2
```

```
Attaching package: 'gridExtra'

The following object is masked from 'package:dplyr':

    combine
```

```r
library(reshape2)
```

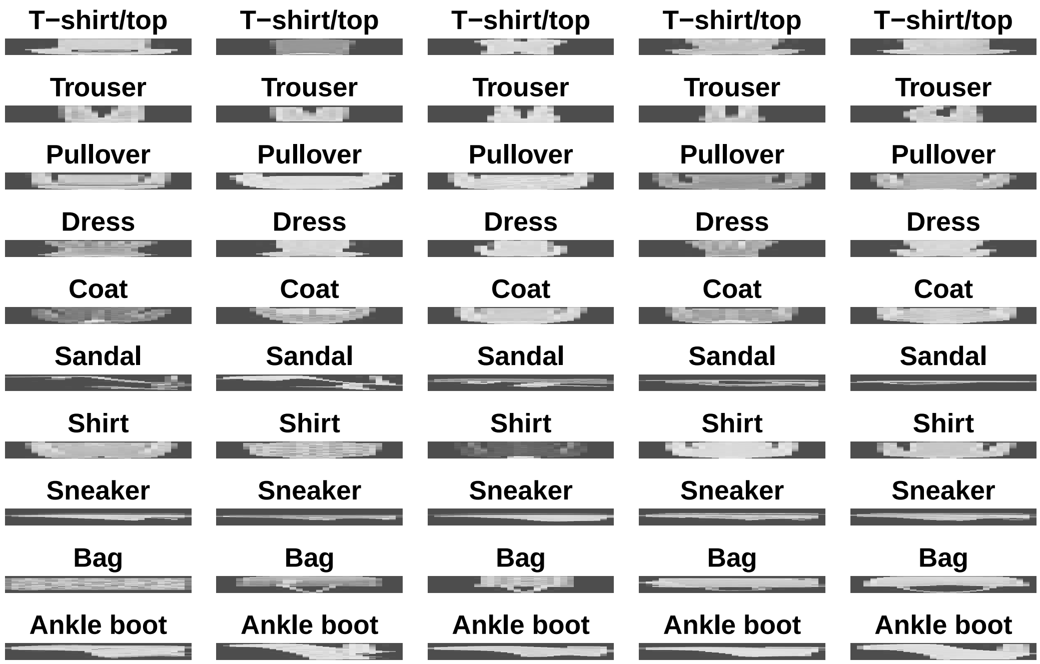
```
Warning: package 'reshape2' was built under R version 4.4.2


Attaching package: 'reshape2'

The following object is masked from 'package:tidyr':

    smiths
```

```r
# Load Fashion MNIST dataset
fashion_mnist <- dataset_fashion_mnist()
x_train <- fashion_mnist$train$x
y_train <- fashion_mnist$train$y
x_test <- fashion_mnist$test$x
y_test <- fashion_mnist$test$y

# Define class labels
fashion_labels <- c(
  "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
  "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"
)

# Explore data distribution
plot_fashion_distribution <- function(y_train, labels) {
  tibble(
    class = factor(y_train, labels = labels),
    count = 1
  ) %>%
    count(class) %>%
    ggplot(aes(x = reorder(class, n), y = n)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    coord_flip() +
```

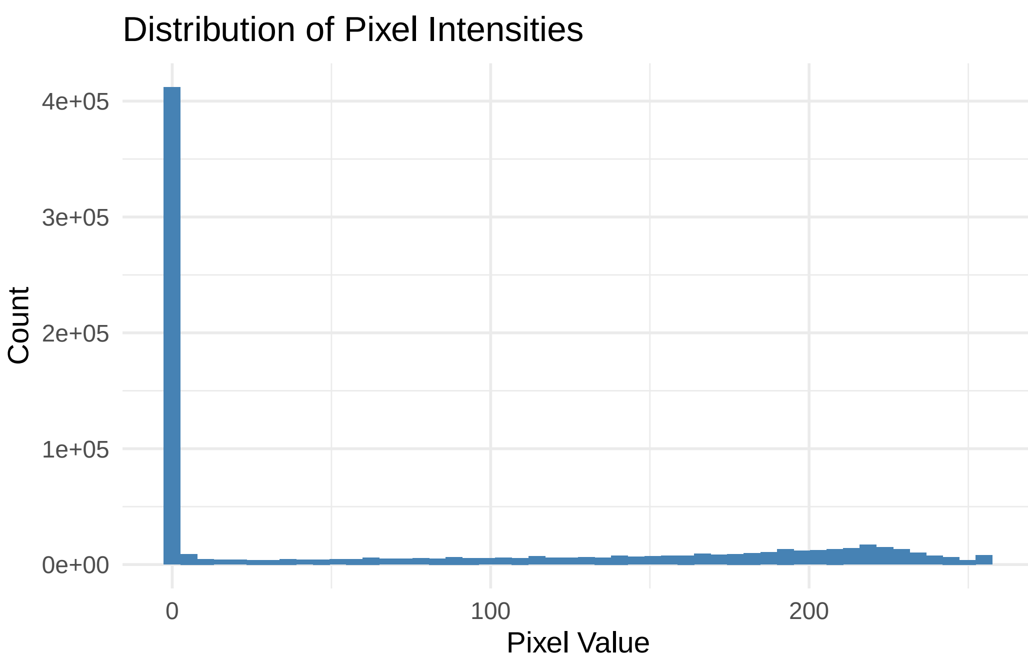
```r
    theme_minimal() +
    labs(
      title = "Distribution of Fashion MNIST Classes",
      x = "Class",
      y = "Count"
    )
}

# Visualize sample images with labels
plot_fashion_samples <- function(x_train, y_train, labels, samples_per_class = 5) {
  par(mfrow = c(length(unique(y_train)), samples_per_class),
      mar = c(0.5, 0.5, 1.5, 0.5))
  for (class in 0:9) {
    class_indices <- which(y_train == class)[1:samples_per_class]
    for (idx in class_indices) {
      image(t(x_train[idx, , ]),
            col = gray.colors(256),
            axes = FALSE,
            main = labels[class + 1])
    }
  }
}

# Plot pixel distribution
plot_pixel_distribution <- function(x_train) {
  # Check dimensions
  dims <- dim(x_train)
  if (length(dims) == 4) {
    # If 4D array, remove the channel dimension
    sample_images <- x_train[1:1000, , , 1]
  } else if (length(dims) == 3) {
    # If 3D array, take as is
    sample_images <- x_train[1:1000, , ]
  } else {
    stop("Input must be 3D or 4D array")
  }
  # Convert to vector
  pixel_values <- as.vector(sample_images)
  # Create plot
  ggplot(data.frame(pixel = pixel_values), aes(x = pixel)) +
    geom_histogram(bins = 50, fill = "steelblue") +
    theme_minimal() +
```

```r
    labs(
      title = "Distribution of Pixel Intensities",
      x = "Pixel Value",
      y = "Count"
    )
}

# Test the functions
# Plot the distribution of classes
plot_fashion_distribution(y_train, fashion_labels)
```


```r
# Plot sample images with labels
plot_fashion_samples(x_train, y_train, fashion_labels)
```




```
# Plot pixel intensity distribution
plot_pixel_distribution(x_train)
```



#Looking at CNN Architecture and Feature Maps

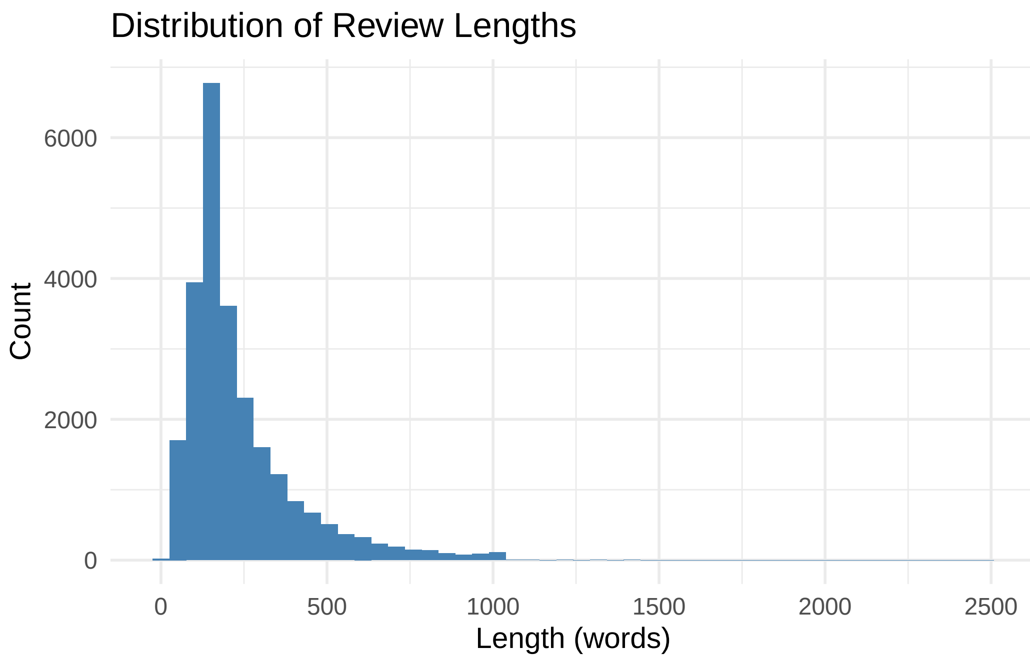
```
# Define the model
build_model <- function() {
  model <- keras_model_sequential() %>%
    # Convolutional and pooling layers
    layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", name = "conv1", i
    layer_max_pooling_2d(pool_size = c(2, 2), name = "pool1") %>%
    layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu", name = "conv2") %
    layer_max_pooling_2d(pool_size = c(2, 2), name = "pool2") %>%
    # Dense layers
    layer_flatten(name = "flatten") %>%
    layer_dense(units = 128, activation = "relu", name = "dense1") %>%
    layer_dropout(rate = 0.5, name = "dropout") %>%
    layer_dense(units = 10, activation = "softmax", name = "output")
  return(model)
}

# Function to visualize feature maps
visualize_feature_maps <- function(model, image) {
  # Create model that outputs feature maps
  layer_outputs <- lapply(1:length(model$layers), function(i) model$layers[[i]]$output)
  activation_model <- keras_model(inputs = model$input, outputs = layer_outputs)

  # Get activations
  activations <- activation_model %>% predict(image)

  # Find convolutional layers
  conv_layers <- which(sapply(model$layers, function(x) inherits(x, "keras.layers.convolutio

  # Visualize feature maps for each convolutional layer
  for (i in seq_along(conv_layers)) {
    layer_name <- model$layers[[conv_layers[i]]]$name
    n_features <- dim(activations[[conv_layers[i]]])[4]

    # Plot first 16 feature maps (or all if less than 16)
    n_cols <- min(4, n_features)
    n_rows <- ceiling(n_features / n_cols)
    par(mfrow = c(n_rows, n_cols), mar = c(0.5, 0.5, 2, 0.5))

    for (j in 1:min(16, n_features)) {
      feature_map <- activations[[conv_layers[i]]][1, , , j]
      image(t(feature_map), main = paste(layer_name, "- Filter", j),
            col = viridis::viridis(100), axes = FALSE)
```

```
    }
  }
}
```

#Looking at LSTM for Text Classification

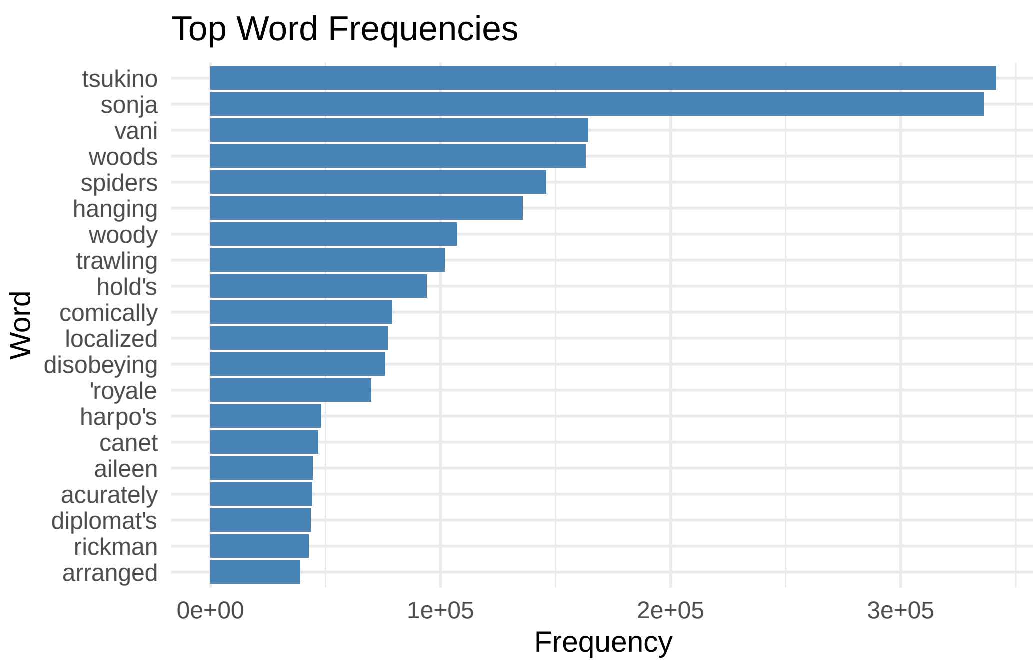
```
# Load IMDB dataset with preprocessing
max_features <- 10000
max_len <- 500
imdb <- dataset_imdb(num_words = max_features)

# Explore text data
analyze_text_lengths <- function(sequences) {
  lengths <- sapply(sequences, length)
  ggplot(data.frame(length = lengths), aes(x = length)) +
    geom_histogram(bins = 50, fill = "steelblue") +
    theme_minimal() +
    labs(
      title = "Distribution of Review Lengths",
      x = "Length (words)",
      y = "Count"
    )
}

# Visualize word frequency
plot_word_frequency <- function(sequences, word_index, top_n = 20) {
  # Count word frequencies
  word_counts <- table(unlist(sequences))
  # Get word labels
  reverse_word_index <- names(word_index)[1:length(word_index)]
  # Create frequency dataframe
  freq_df <- data.frame(
    word = reverse_word_index[as.numeric(names(word_counts))],
    count = as.numeric(word_counts)
  ) %>%
    arrange(desc(count)) %>%
    head(top_n)
  ggplot(freq_df, aes(x = reorder(word, count), y = count)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    coord_flip() +
    theme_minimal() +
    labs(
```

```
        title = "Top Word Frequencies",
        x = "Word",
        y = "Frequency"
    )
}

# Show text length distribution
analyze_text_lengths(imdb$train$x)
```



Show word frequencies

```
plot_word_frequency(imdb$train$x, dataset_imdb_word_index())
```



Create and visualize LSTM model

```
create_lstm_model <- function(max_features, max_len) {
  model <- keras_model_sequential()
  model %>%
    layer_embedding(
      input_dim = max_features,
      output_dim = 128,
      input_length = max_len,
      name = "embedding"
    ) %>%
    layer_lstm(
      units = 64,
      return_sequences = TRUE,
      name = "lstm1"
    ) %>%
    layer_lstm(
      units = 32,
      name = "lstm2"
    ) %>%
```

```
    layer_dense(
      units = 1,
      activation = "sigmoid",
      name = "output"
    )
  return(model)
}
```

#Comparing

```
compare_architectures <- function(cnn_model, lstm_model) {
  # Extract layer information
  get_layer_info <- function(model) {
    tibble(
      layer = sapply(model$layers, function(x) x$name),
      type = sapply(model$layers, function(x) class(x)[1]),
      parameters = sapply(model$layers, function(x) x$count_params())
    )
  }
  cnn_info <- get_layer_info(cnn_model)
  lstm_info <- get_layer_info(lstm_model)

  # Plot comparisons
  p1 <- ggplot(cnn_info, aes(x = layer, y = parameters)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    labs(title = "CNN Architecture", x = "Layer", y = "Parameters")

  p2 <- ggplot(lstm_info, aes(x = layer, y = parameters)) +
    geom_bar(stat = "identity", fill = "coral") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    labs(title = "LSTM Architecture", x = "Layer", y = "Parameters")

  grid.arrange(p1, p2, ncol = 2)
}

# Compare training metrics
compare_training_histories <- function(cnn_history, lstm_history) {
  # Combine histories
  cnn_df <- data.frame(
```

```r
    epoch = 1:length(cnn_history$metrics$accuracy),
    accuracy = cnn_history$metrics$accuracy,
    model = "CNN"
  )

  lstm_df <- data.frame(
    epoch = 1:length(lstm_history$metrics$accuracy),
    accuracy = lstm_history$metrics$accuracy,
    model = "LSTM"
  )

  combined_df <- rbind(cnn_df, lstm_df)

  ggplot(combined_df, aes(x = epoch, y = accuracy, color = model)) +
    geom_line() +
    theme_minimal() +
    labs(title = "Training Progress Comparison", x = "Epoch", y = "Accuracy")
}
```

#Looking at buiding an image classifer in Python using the Fashion MNIST dataset.

```python
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# Class labels
fashion_labels = [
    'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
    'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'
]

# Data visualization functions
def plot_fashion_distribution(y_train, labels):
    plt.figure(figsize=(10, 6))
    sns.countplot(y=pd.Series(y_train).map(lambda x: labels[x]))
    plt.title('Distribution of Fashion MNIST Classes')
```

```
    plt.xlabel('Class')
    plt.ylabel('Count')
    plt.tight_layout()
    plt.show()

def plot_fashion_samples(x_train, y_train, labels, samples_per_class=5):
    fig, axes = plt.subplots(len(labels), samples_per_class, figsize=(samples_per_class * 2,
    for i, label in enumerate(range(len(labels))):
        indices = np.where(y_train == label)[0][:samples_per_class]
        for j, idx in enumerate(indices):
            axes[i, j].imshow(x_train[idx], cmap='gray')
            axes[i, j].axis('off')
            if j == 0:
                axes[i, j].set_ylabel(labels[label], rotation=0, labelpad=30, va='center')
    plt.tight_layout()
    plt.show()

# Create CNN model
def create_cnn_model(input_shape=(28, 28, 1)):
    model = keras.Sequential([
        keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape, name='co
        keras.layers.MaxPooling2D((2, 2), name='pool1'),
        keras.layers.Conv2D(64, (3, 3), activation='relu', name='conv2'),
        keras.layers.MaxPooling2D((2, 2), name='pool2'),
        keras.layers.Flatten(name='flatten'),
        keras.layers.Dense(128, activation='relu', name='dense1'),
        keras.layers.Dropout(0.5, name='dropout'),
        keras.layers.Dense(10, activation='softmax', name='output')
    ])
    return model

# Visualize feature maps
def visualize_feature_maps(model, image):
    # Create a model that will output feature maps
    layer_outputs = [layer.output for layer in model.layers if isinstance(layer, keras.layers
    activation_model = keras.Model(inputs=model.input, outputs=layer_outputs)

    # Get feature maps
    activations = activation_model.predict(np.expand_dims(image, 0))

    # Plot feature maps
    for i, activation in enumerate(activations):
```

```
        n_features = activation.shape[-1]
        size = activation.shape[1]
        n_cols = min(n_features, 8)
        n_rows = (n_features + n_cols - 1) // n_cols  # Ensure enough rows for all features
        fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 2, n_rows * 2))
        for j in range(n_features):
            row, col = divmod(j, n_cols)
            ax = axes[row, col] if n_rows > 1 else axes[col]
            ax.imshow(activation[0, :, :, j], cmap='viridis')
            ax.axis('off')
        plt.suptitle(f'Feature maps for layer {model.layers[i * 2].name}')
        plt.tight_layout()
        plt.show()
```

Using LSTM for Text Classification in Python

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load IMDB dataset
max_features = 10000
max_len = 500
(x_train, y_train), (x_test, y_test) = keras.datasets.imdb.load_data(num_words=max_features)

# Text analysis functions
def analyze_text_lengths(sequences):
    lengths = [len(seq) for seq in sequences]
    plt.figure(figsize=(10, 6))
    plt.hist(lengths, bins=50, color="steelblue")
    plt.title('Distribution of Review Lengths')
    plt.xlabel('Length (words)')
    plt.ylabel('Count')
    plt.show()
```

```python
def plot_word_frequency(sequences, word_index, top_n=20):
    # Count word frequencies
    word_freq = {}
    for seq in sequences:
        for word_id in seq:
            if word_id not in word_freq:
                word_freq[word_id] = 0
            word_freq[word_id] += 1

    # Sort and plot
    word_freq_sorted = sorted(word_freq.items(), key=lambda x: x[1], reverse=True)
    words = [list(word_index.keys())[list(word_index.values()).index(id)]
             for id, _ in word_freq_sorted[:top_n]]
    freqs = [freq for _, freq in word_freq_sorted[:top_n]]
    plt.figure(figsize=(12, 6))
    sns.barplot(x=freqs, y=words, palette="viridis")
    plt.title('Top Word Frequencies')
    plt.xlabel('Frequency')
    plt.show()

# Create LSTM model
def create_lstm_model(max_features, max_len):
    model = keras.Sequential([
        keras.layers.Embedding(max_features, 128, input_length=max_len, name='embedding'),
        keras.layers.LSTM(64, return_sequences=True, name='lstm1'),
        keras.layers.LSTM(32, name='lstm2'),
        keras.layers.Dense(1, activation='sigmoid', name='output')
    ])
    return model

# Architecture comparison visualization
def compare_architectures(cnn_model, lstm_model):
    def get_model_info(model):
        return pd.DataFrame({
            'layer': [layer.name for layer in model.layers],
            'parameters': [layer.count_params() for layer in model.layers]
        })

    cnn_info = get_model_info(cnn_model)
    lstm_info = get_model_info(lstm_model)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
```

```
sns.barplot(data=cnn_info, x='layer', y='parameters', ax=ax1, color="steelblue")
ax1.set_title('CNN Architecture')
ax1.tick_params(axis='x', rotation=45)

sns.barplot(data=lstm_info, x='layer', y='parameters', ax=ax2, color="coral")
ax2.set_title('LSTM Architecture')
ax2.tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```