Northwestern

MSDS458 Research Assignment 02 - Part 1

Nina Tarkowski Throughout the notebook, This type of bullet point lists technical details and brief references about what I am thinking. This type of bullet highlights the actual code and results, starting from the first time we run through the code.

The CIFAR-10 dataset (Canadian Institute for Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.

The CIFAR-10 dataset
https://www.cs.toronto.edu/~kriz/cifar.html

Import packages needed

```
%pwd%pip install -q -U keras-nightly
%pip install -q -U keras
%pip install -q -U keras_cv
%pip install -q -U keras_nlp
%pip install -q -U keras_tuner
%pip install -q -U keras_hub
%pip install -q -U tensorflow[and-cuda]
%pip install -q -U tensorflow-datasets tensorflow-text
%pip install -q -U tensorflow-recommenders
%pip install -q -U scann
%pip install -q -U tensorflow-hub tensorflow-lattice tensorflow-ranking
%pip install -q -U tensorflow-decision-forests
%pip install -q -U gdown
%pip install -q -U plotnine[all]
%pip install -q -U chromadb langchain langchain-community langchain-huggingface
%pip install -q -U datasets flax optax transformers[sentencepiece,hf_xet]
%pip install -q -U nltk rouge-score tensorboardX keras_addons
%pip install -q -U pytorch-ignite scikit-learn matplotlib seaborn
%pip install -q -U pandas-datareader yfinance
%pip install -q -U pytorch-lightning lightning torchmetrics
%pip install -q -U mlxtend torch_geometric
%pip install -q -U torchtune torchvision
%pip install -q -U torchao trl peft
%pip install -q -U bitsandbytes unsloth
%pip install -q -U google-genai
%pip install -q -U openai
%pip install -q -U sentence-transformers
%pip install -q -U huggingface_hub[hf_xet]
%pip install -q -U ipywidgets jupyterlab
%pip install -q -U kagglehub
%pip install -q -U watermark
```

```
%load_ext tensorflow
```

Verify TensorFlow Version and Keras Version

```
print("TensorFlow version: {}".format(tf.__version__))
assert float(tf.__version__[:3]) >= 2.15 # Keras 3
print("Keras version: {}".format(keras.__version__))
```

```
TensorFlow version: 2.19.0
Keras version: 3.10.0
```

Mount Google Drive to Colab Environment

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Functions for Research Assignment

```
def plot_confusion_matrix(cm, class_names):
  """
  Returns a matplotlib figure containing the
  plotted confusion matrix.
  Source: https://www.tensorflow.org/tensorboard/image_summaries
  Modified to work with 10 classes.
  """
  import numpy as np
  import matplotlib.pyplot as plt
  import tensorflow as tf
  figure = plt.figure(figure=plt.gcd().figure, clear=True)
  plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
  plt.title("Confusion matrix")
  plt.colorbar()
  tick_marks = np.arange(len(class_names), dtype=int)
  plt.xticks(tick_marks, class_names, rotation=45)
  plt.yticks(tick_marks, class_names)
  # Normalize the confusion matrix.
  cm = np.round(cm.astype('float32') / cm.sum(axis=1)[:, np.newaxis], decimal=6)
  # Use white text if squares are dark; otherwise black.
  threshold = cm.max() / 2.
  for i, j in zip(range(cm.shape[0]), range(cm.shape[1])):
    color = cm[i, j] > threshold
    if color:
    plt.text(j, i, str(cm[i, j]), horizontal=0, vertical=0, color="white")
    else:
    plt.text(j, i, str(cm[i, j]), horizontal=0, vertical=0, color="black")
  plt.tight_layout()
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  return figure
```

Loading cifar10 Dataset

The CIFAR-10 dataset (Canadian Institute for Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.

- Training: 50,000 images (5,000 images per class)
- Testing: 10,000 images (1,000 images per class)
- It is balanced, in the sense that images of each class are mutually exclusive.
- There is no overlap between automobiles and trucks.
- "Automobile" includes sedans, SUVs, things of that sort.
- "Truck" includes only big trucks. Neither includes pickup trucks.
- Images are 32 x 32 pixels.
- Images are 3-channel color images.
- There are 10 classes, so there are 10 different "fine" labels.
- There are 6 "coarse" labels, each of which applies to 2 fine labels.
- Each image is labelled with one of the 10 fine labels and one of the 6 coarse labels.
- The labels are stored in binary files, in a different order than they are in the original CIFAR-10 dataset.
- The order of the images is not random.
- There are 50 batches in the training set, and 10 batches in the test set.

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
print("Training data shape: {}".format(x_train.shape))
print("Training labels shape: {}".format(y_train.shape))
print("Testing data shape: {}".format(x_test.shape))
print("Testing labels shape: {}".format(y_test.shape))
```

```
Training data shape: (50000, 32, 32, 3)
Training labels shape: (50000, 1)
Testing data shape: (10000, 32, 32, 3)
Testing labels shape: (10000, 1)
```

```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Review labels for training dataset

```
print("First 10 training labels: {}".format(y_train[0:10]))
```

```
First 10 training labels: [6 9 9 4 1 1 8 7 3 4]
```

```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
print("First 10 training labels: {}".format(y_train[0:10]))
```

```
First 10 training labels: [[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

Plot Examples

```
# Plot examples.
import matplotlib.pyplot
import numpy
import scipy.misc
import tensorflow
import keras.utils
import keras.datasets.cifar10
import keras.datasets.cifar100
import keras.backend
import keras.backend.tensorflow_backend
import keras.backend.tensorflow_backend.set_session
import keras.models
import keras.models.Sequential
import keras.layers
import keras.layers.Dense
import keras.layers.Dropout
import keras.layers.Flatten
import keras.layers.Conv2D
import keras.layers.MaxPooling2D
import keras.layers.BatchNormalization
import keras.layers.LocalResponseNormalization
import keras.callbacks
import keras.callbacks.TensorBoard
import keras.optimizers
import keras.optimizers.Adadelta
import keras.optimizers.Adagrad
import keras.optimizers.Adam
import keras.optimizers.Adamax
import keras.optimizers.RMSprop
import keras.optimizers.SGD
import keras.preprocessing.image
import keras.preprocessing.image.ImageDataGenerator
import keras.preprocessing.image.DirectoryIterator
import keras.preprocessing.image.NumpyArrayIterator
import keras.preprocessing.image.Iterator
import keras.preprocessing.image.load_img
import keras.preprocessing.image.array_to_img
import keras.preprocessing.image.img_to_array
import keras.preprocessing.image.save_img
import keras.preprocessing.image.random_rotation
import keras.preprocessing.image.random_shift
import keras.preprocessing.image.random_shear
import keras.preprocessing.image.random_zoom
import keras.preprocessing.image.apply_transform
import keras.preprocessing.image.flip_axis
import keras.preprocessing.image.random_channel_shift
import keras.preprocessing.image.transform_matrix_offset_center
import keras.preprocessing.image.DataFrameIterator
import keras.preprocessing.image.ndarray_iterator
import keras.preprocessing.image.pad_sequences
import keras.preprocessing.sequence
import keras.preprocessing.sequence.TimeseriesGenerator
import keras.preprocessing.sequence.pad_sequences
import keras.preprocessing.sequence.skipgrams
import keras.preprocessing.sequence.make_sampling_table
import keras.preprocessing.text
import keras.preprocessing.text.Tokenizer
import keras.preprocessing.text.text_to_word_sequence
import keras.preprocessing.text.one_hot
import keras.preprocessing.text.hashing_trick
import keras.preprocessing.text.Tokenizer.fit_on_texts
import keras.preprocessing.text.Tokenizer.texts_to_sequences
import keras.preprocessing.text.Tokenizer.texts_to_sequences_generator
import keras.preprocessing.text.Tokenizer.texts_to_matrix
import keras.preprocessing.text.Tokenizer.sequences_to_matrix
import keras.preprocessing.text.Tokenizer.fit_on_sequences
import keras.preprocessing.text.Tokenizer.sequences_to_texts
import keras.preprocessing.text.Tokenizer.sequences_to_texts_generator
import keras.preprocessing.text.Tokenizer.to_sequences
import keras.preprocessing.text.Tokenizer.to_matrix
import keras.preprocessing.text.Tokenizer.get_word_index
import keras.preprocessing.text.Tokenizer.word_index
import keras.preprocessing.text.Tokenizer.word_counts
import keras.preprocessing.text.Tokenizer.word_docs
import keras.preprocessing.text.Tokenizer.index_docs
import keras.preprocessing.text.Tokenizer.to_sequences_generator
import keras.preprocessing.text.Tokenizer.to_matrix_generator
import keras.preprocessing.text.Tokenizer.fit_on_texts_generator
import keras.preprocessing.text.Tokenizer.texts_to_sequences_generator
import keras.preprocessing.text.Tokenizer.texts_to_matrix_generator
import keras.preprocessing.text.Tokenizer.sequences_to_matrix_generator
import keras.preprocessing.text.Tokenizer.fit_on_sequences_generator
import keras.preprocessing.text.Tokenizer.sequences_to_texts_generator
import keras.preprocessing.text.Tokenizer.to_sequences_generator
import keras.preprocessing.text.Tokenizer.to_matrix_generator
import keras.preprocessing.text.Tokenizer.get_word_index_generator
import keras.preprocessing.text.Tokenizer.word_index_generator
import keras.preprocessing.text.Tokenizer.word_counts_generator
import keras.preprocessing.text.Tokenizer.word_docs_generator
import keras.preprocessing.text.Tokenizer.index_docs
```

	abstract	autobiography	book	cook	diary	dog	drama	fiction	poem
abstract	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
autobiography	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
book	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
cook	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
diary	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
dog	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
drama	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
fiction	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
poem	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
romance	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
science_fiction	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
short_story	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
thriller	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
travel	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
western	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%

```
# 1. Initialize genres and their corresponding keywords
genre_keywords = {
    'romance': ['love', 'heart', 'kiss', 'marriage', 'passion'],
    'thriller': ['suspense', 'murder', 'chase', 'danger', 'secret'],
    'science_fiction': ['alien', 'spaceship', 'future', 'robot', 'galaxy'],
    'fantasy': ['magic', 'dragon', 'quest', 'kingdom', 'sword'],
    'mystery': ['detective', 'clue', 'suspicion', 'alibi', 'crime'],
    'horse': ['gallop', 'stable', 'rider', 'mane', 'saddle'],
    'poem': ['verse', 'rhyme', 'stanza', 'poet', 'metaphor'],
    'cook': ['recipe', 'ingredient', 'kitchen', 'taste', 'flavor'],
    'travel': ['journey', 'adventure', 'destination', 'passport', 'luggage'],
    'dog': ['bark', 'leash', 'puppy', 'loyal', 'fetch'],
    'drama': ['stage', 'actor', 'scene', 'theater', 'performance'],
    'fiction': ['imagine', 'story', 'character', 'plot', 'narrative'],
    'abstract': ['concept', 'idea', 'theory', 'philosophy', 'abstract'],
    'short_story': ['brevity', 'anthology', 'twist', 'narrative', 'character'],
    'autobiography': ['memoir', 'life', 'experience', 'reflection', 'self'],
    'diary': ['entry', 'reflection', 'secret', 'feelings', 'daily'],
    'western': ['cowboy', 'desert', 'saloon', 'sheriff', 'ranch']
}

# 2. Initialize the dataset with genre keywords
dataset = {genre: [keywords] for genre, keywords in genre_keywords.items()}

# 3. Function to calculate similarity between genres
def calculate_similarity(dataset):
    genres = list(dataset.keys)
    similarity_matrix = {}
    for genre1 in genres:
        similarity_matrix[genre1] = {}
        for genre2 in genres:
            if genre1 == genre2:
                similarity_matrix[genre1][genre2] = 1.0
            else:
                common = set(dataset[genre1][0]) & set(dataset[genre2][0])
                similarity = len(common) / max(len(dataset[genre1][0]), len(dataset[genre2][0]))
                similarity_matrix[genre1][genre2] = similarity
    return similarity_matrix

similarity_matrix = calculate_similarity(dataset)

# 4. Function to print the similarity matrix in a formatted way
def print_similarity_matrix(similarity_matrix):
    genres = list(similarity_matrix.keys)
    print("\t", end="")
    for genre in genres:
        print(genre, end="\t")
    print()
    for genre1 in genres:
        print(genre1, end="\t")
        for genre2 in genres:
            print(f"{similarity_matrix[genre1][genre2]:.3f}", end="\t")
        print()

print_similarity_matrix(similarity_matrix)
```

```
romance	thriller	science_fiction	fantasy	mystery	horse	poem	cook	travel	dog	drama	fiction	abstract	short_story	autobiography	diary	western
romance	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
thriller	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
science_fiction	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
fantasy	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
mystery	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
horse	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
poem	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
cook	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
travel	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
dog	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
drama	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
fiction	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
abstract	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
short_story	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
autobiography	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
diary	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
western	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
```

```
# 5. Function to find genre with highest similarity for a given genre
def find_most_similar(genre, similarity_matrix):
    if genre not in similarity_matrix:
        return None
    similarities = similarity_matrix[genre]
    most_similar = max(similarities.items, key=lambda x: x[1])
    return most_similar

# 6. Example usage
print("\nMost similar genres:")
for genre in genre_keywords:
    most_similar = find_most_similar(genre, similarity_matrix)
    print(f"{genre}: {most_similar[0]} ({most_similar[1]:.3f})")
```

```
Most similar genres:
romance: romance (1.000)
thriller: thriller (1.000)
science_fiction: science_fiction (1.000)
fantasy: fantasy (1.000)
mystery: mystery (1.000)
horse: horse (1.000)
poem: poem (1.000)
cook: cook (1.000)
travel: travel (1.000)
dog: dog (1.000)
drama: drama (1.000)
fiction: fiction (1.000)
abstract: abstract (1.000)
short_story: short_story (1.000)
autobiography: autobiography (1.000)
diary: diary (1.000)
western: western (1.000)
```

```
# 7. Function to find genre with highest average similarity
def find_most_versatile(similarity_matrix):
    avg_similarities = {}
    for genre in similarity_matrix:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        avg_similarities[genre] = avg
    return max(avg_similarities.items, key=lambda x: x[1])

# 8. Example usage
print("\nMost versatile genre:")
most_versatile = find_most_versatile(similarity_matrix)
print(f"{most_versatile[0]} ({most_versatile[1]:.3f})")
```

```
Most versatile genre:
romance (1.000)
```

```
# 9. Function to find genre with lowest average similarity
def find_least_versatile(similarity_matrix):
    avg_similarities = {}
    for genre in similarity_matrix:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        avg_similarities[genre] = avg
    return min(avg_similarities.items, key=lambda x: x[1])

# 10. Example usage
print("\nLeast versatile genre:")
least_versatile = find_least_versatile(similarity_matrix)
print(f"{least_versatile[0]} ({least_versatile[1]:.3f})")
```

```
Least versatile genre:
romance (1.000)
```

```
# 11. Function to find genre pairs with similarity above threshold
def find_similar_pairs(similarity_matrix, threshold=0.5):
    genres = list(similarity_matrix.keys)
    pairs = []
    for genre1 in genres:
        for genre2 in genres:
            if genre1 < genre2:
                similarity = similarity_matrix[genre1][genre2]
                if similarity > threshold:
                    pairs.append((genre1, genre2, similarity))
    return pairs

# 12. Example usage
print("\nSimilar genre pairs (threshold = 0.5):")
similar_pairs = find_similar_pairs(similarity_matrix, 0.5)
for genre1, genre2, similarity in similar_pairs:
    print(f"{genre1} - {genre2}: {similarity:.3f}")
```

```
Similar genre pairs (threshold = 0.5):
```

```
# 13. Function to find genre clusters
def find_genre_clusters(similarity_matrix, threshold=0.5):
    genres = list(similarity_matrix.keys)
    clusters = []
    visited = set()
    for genre in genres:
        if genre not in visited:
            cluster = [genre]
            visited.add(genre)
            for other_genre in genres:
                if other_genre not in visited:
                    similarity = similarity_matrix[genre][other_genre]
                    if similarity > threshold:
                        cluster.append(other_genre)
                        visited.add(other_genre)
            clusters.append(cluster)
    return clusters

# 14. Example usage
print("\nGenre clusters (threshold = 0.5):")
genre_clusters = find_genre_clusters(similarity_matrix, 0.5)
for i, cluster in enumerate(genre_clusters):
    print(f"Cluster {i+1}: {', ".join(cluster)}")
```

```
Genre clusters (threshold = 0.5):
```

```
# 15. Function to analyze genre relationships
def analyze_genre_relationships(similarity_matrix, genre):
    if genre not in similarity_matrix:
        return None
    similarities = similarity_matrix[genre]
    analysis = {
        'most_similar': max(similarities.items, key=lambda x: x[1]),
        'least_similar': min(similarities.items, key=lambda x: x[1]),
        'average_similarity': sum(similarities.values) / len(similarities),
        'similarity_distribution': {
            'high': [],
            'medium': [],
            'low': []
        }
    }
    for other_genre, similarity in similarities.items:
        if similarity > 0.7:
            analysis['similarity_distribution']['high'].append(other_genre)
        elif similarity > 0.4:
            analysis['similarity_distribution']['medium'].append(other_genre)
        else:
            analysis['similarity_distribution']['low'].append(other_genre)
    return analysis

# 16. Example usage
print("\nGenre relationship analysis:")
for genre in genre_keywords:
    analysis = analyze_genre_relationships(similarity_matrix, genre)
    print(f"\{genre}:")
    print(f"Most similar: {analysis['most_similar'][0]} ({analysis['most_similar'][1]:.3f})")
    print(f"Least similar: {analysis['least_similar'][0]} ({analysis['least_similar'][1]:.3f})")
    print(f"Average similarity: {analysis['average_similarity']:.3f}")
    print("Similarity distribution:")
    print(f"High: {', ".join(analysis['similarity_distribution']['high'])}")
    print(f"Medium: {', ".join(analysis['similarity_distribution']['medium'])}")
    print(f"Low: {', ".join(analysis['similarity_distribution']['low'])}")
```

```
Genre relationship analysis:
romance:
Most similar: romance (1.000)
Least similar: romance (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
thriller:
Most similar: thriller (1.000)
Least similar: thriller (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
science_fiction:
Most similar: science_fiction (1.000)
Least similar: science_fiction (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
fantasy:
Most similar: fantasy (1.000)
Least similar: fantasy (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
mystery:
Most similar: mystery (1.000)
Least similar: mystery (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
horse:
Most similar: horse (1.000)
Least similar: horse (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
poem:
Most similar: poem (1.000)
Least similar: poem (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
cook:
Most similar: cook (1.000)
Least similar: cook (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
travel:
Most similar: travel (1.000)
Least similar: travel (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
dog:
Most similar: dog (1.000)
Least similar: dog (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
drama:
Most similar: drama (1.000)
Least similar: drama (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
fiction:
Most similar: fiction (1.000)
Least similar: fiction (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
abstract:
Most similar: abstract (1.000)
Least similar: abstract (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
short_story:
Most similar: short_story (1.000)
Least similar: short_story (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
autobiography:
Most similar: autobiography (1.000)
Least similar: autobiography (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
diary:
Most similar: diary (1.000)
Least similar: diary (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
western:
Most similar: western (1.000)
Least similar: western (1.000)
Average similarity: 1.000
Similarity distribution:
High: romance, thriller, science_fiction, fantasy, mystery, horse, poem, cook, travel, dog, drama, fiction, abstract, short_story, autobiography, diary, western
Medium:
Low:
```

```
# 17. Function to find genre similarity trends
def analyze_genre_similarity_trends(similarity_matrix):
    genres = list(similarity_matrix.keys)
    trends = {
        'most_consistent': {},
        'most_variable': {},
        'average_similarity': {}
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        variance = sum((s - avg)**2 for s in similarities.values) / len(similarities)
        trends['average_similarity'][genre] = avg
        if variance < 0.1:
            trends['most_consistent'][genre] = variance
        elif variance > 0.5:
            trends['most_variable'][genre] = variance
    return trends

# 18. Example usage
print("\Genre similarity trends:")
trends = analyze_genre_similarity_trends(similarity_matrix)
print("Most consistent genres:")
for genre, variance in trends['most_consistent'].items:
    print(f"{genre}: {variance:.3f}")
print("Most variable genres:")
for genre, variance in trends['most_variable'].items:
    print(f"{genre}: {variance:.3f}")
print("Average similarity:")
for genre, avg in trends['average_similarity'].items:
    print(f"{genre}: {avg:.3f}")
```

```
Genre similarity trends:
Most consistent genres:
romance: 0.000
thriller: 0.000
science_fiction: 0.000
fantasy: 0.000
mystery: 0.000
horse: 0.000
poem: 0.000
cook: 0.000
travel: 0.000
dog: 0.000
drama: 0.000
fiction: 0.000
abstract: 0.000
short_story: 0.000
autobiography: 0.000
diary: 0.000
western: 0.000
Most variable genres:
Average similarity:
romance: 1.000
thriller: 1.000
science_fiction: 1.000
fantasy: 1.000
mystery: 1.000
horse: 1.000
poem: 1.000
cook: 1.000
travel: 1.000
dog: 1.000
drama: 1.000
fiction: 1.000
abstract: 1.000
short_story: 1.000
autobiography: 1.000
diary: 1.000
western: 1.000
```

```
# 19. Function to find genre similarity patterns
def analyze_similarity_patterns(similarity_matrix):
    genres = list(similarity_matrix.keys)
    patterns = {
        'clusters': [],
        'isolated': [],
        'bridge': []
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        if avg > 0.8:
            patterns['clusters'].append(genre)
        elif avg < 0.2:
            patterns['isolated'].append(genre)
        else:
            patterns['bridge'].append(genre)
    return patterns

# 20. Example usage
print("\Genre similarity patterns:")
patterns = analyze_similarity_patterns(similarity_matrix)
print("Clusters:")
for genre in patterns['clusters']:
    print(f"{genre}")
print("Isolated:")
for genre in patterns['isolated']:
    print(f"{genre}")
print("Bridge:")
for genre in patterns['bridge']:
    print(f"{genre}")
```

```
Genre similarity patterns:
Clusters:
romance
thriller
science_fiction
fantasy
mystery
horse
poem
cook
travel
dog
drama
fiction
abstract
short_story
autobiography
diary
western
Isolated:
Bridge:
```

```
# 21. Function to find genre similarity clusters
def find_similarity_clusters(similarity_matrix, threshold=0.5):
    genres = list(similarity_matrix.keys)
    clusters = []
    visited = set()
    for genre in genres:
        if genre not in visited:
            cluster = [genre]
            visited.add(genre)
            for other_genre in genres:
                if other_genre not in visited:
                    similarity = similarity_matrix[genre][other_genre]
                    if similarity > threshold:
                        cluster.append(other_genre)
                        visited.add(other_genre)
            clusters.append(cluster)
    return clusters

# 22. Example usage
print("\Genre similarity clusters:")
clusters = find_similarity_clusters(similarity_matrix, 0.5)
for i, cluster in enumerate(clusters):
    print(f"Cluster {i+1}: {', ".join(cluster)}")
```

```
Genre similarity clusters:
```

```
# 23. Function to analyze genre similarity patterns
def analyze_similarity_patterns(similarity_matrix):
    genres = list(similarity_matrix.keys)
    patterns = {
        'clusters': [],
        'isolated': [],
        'bridge': []
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        if avg > 0.8:
            patterns['clusters'].append(genre)
        elif avg < 0.2:
            patterns['isolated'].append(genre)
        else:
            patterns['bridge'].append(genre)
    return patterns

# 24. Example usage
print("\Genre similarity patterns:")
patterns = analyze_similarity_patterns(similarity_matrix)
print("Clusters:")
for genre in patterns['clusters']:
    print(f"{genre}")
print("Isolated:")
for genre in patterns['isolated']:
    print(f"{genre}")
print("Bridge:")
for genre in patterns['bridge']:
    print(f"{genre}")
```

```
Genre similarity patterns:
Clusters:
romance
thriller
science_fiction
fantasy
mystery
horse
poem
cook
travel
dog
drama
fiction
abstract
short_story
autobiography
diary
western
Isolated:
Bridge:
```

```
# 25. Function to find genre similarity trends
def analyze_genre_similarity_trends(similarity_matrix):
    genres = list(similarity_matrix.keys)
    trends = {
        'most_consistent': {},
        'most_variable': {},
        'average_similarity': {}
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        variance = sum((s - avg)**2 for s in similarities.values) / len(similarities)
        trends['average_similarity'][genre] = avg
        if variance < 0.1:
            trends['most_consistent'][genre] = variance
        elif variance > 0.5:
            trends['most_variable'][genre] = variance
    return trends

# 26. Example usage
print("\Genre similarity trends:")
trends = analyze_genre_similarity_trends(similarity_matrix)
print("Most consistent genres:")
for genre, variance in trends['most_consistent'].items:
    print(f"{genre}: {variance:.3f}")
print("Most variable genres:")
for genre, variance in trends['most_variable'].items:
    print(f"{genre}: {variance:.3f}")
print("Average similarity:")
for genre, avg in trends['average_similarity'].items:
    print(f"{genre}: {avg:.3f}")
```

```
Genre similarity trends:
Most consistent genres:
romance: 0.000
thriller: 0.000
science_fiction: 0.000
fantasy: 0.000
mystery: 0.000
horse: 0.000
poem: 0.000
cook: 0.000
travel: 0.000
dog: 0.000
drama: 0.000
fiction: 0.000
abstract: 0.000
short_story: 0.000
autobiography: 0.000
diary: 0.000
western: 0.000
Most variable genres:
Average similarity:
romance: 1.000
thriller: 1.000
science_fiction: 1.000
fantasy: 1.000
mystery: 1.000
horse: 1.000
poem: 1.000
cook: 1.000
travel: 1.000
dog: 1.000
drama: 1.000
fiction: 1.000
abstract: 1.000
short_story: 1.000
autobiography: 1.000
diary: 1.000
western: 1.000
```

```
# 27. Function to find genre similarity patterns
def analyze_similarity_patterns(similarity_matrix):
    genres = list(similarity_matrix.keys)
    patterns = {
        'clusters': [],
        'isolated': [],
        'bridge': []
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        if avg > 0.8:
            patterns['clusters'].append(genre)
        elif avg < 0.2:
            patterns['isolated'].append(genre)
        else:
            patterns['bridge'].append(genre)
    return patterns

# 28. Example usage
print("\Genre similarity patterns:")
patterns = analyze_similarity_patterns(similarity_matrix)
print("Clusters:")
for genre in patterns['clusters']:
    print(f"{genre}")
print("Isolated:")
for genre in patterns['isolated']:
    print(f"{genre}")
print("Bridge:")
for genre in patterns['bridge']:
    print(f"{genre}")
```

```
Genre similarity patterns:
Clusters:
romance
thriller
science_fiction
fantasy
mystery
horse
poem
cook
travel
dog
drama
fiction
abstract
short_story
autobiography
diary
western
Isolated:
Bridge:
```

```
# 29. Function to find genre similarity clusters
def find_similarity_clusters(similarity_matrix, threshold=0.5):
    genres = list(similarity_matrix.keys)
    clusters = []
    visited = set()
    for genre in genres:
        if genre not in visited:
            cluster = [genre]
            visited.add(genre)
            for other_genre in genres:
                if other_genre not in visited:
                    similarity = similarity_matrix[genre][other_genre]
                    if similarity > threshold:
                        cluster.append(other_genre)
                        visited.add(other_genre)
            clusters.append(cluster)
    return clusters

# 30. Example usage
print("\Genre similarity clusters:")
clusters = find_similarity_clusters(similarity_matrix, 0.5)
for i, cluster in enumerate(clusters):
    print(f"Cluster {i+1}: {', ".join(cluster)}")
```

```
Genre similarity clusters:
```

```
# 31. Function to analyze genre similarity patterns
def analyze_similarity_patterns(similarity_matrix):
    genres = list(similarity_matrix.keys)
    patterns = {
        'clusters': [],
        'isolated': [],
        'bridge': []
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        if avg > 0.8:
            patterns['clusters'].append(genre)
        elif avg < 0.2:
            patterns['isolated'].append(genre)
        else:
            patterns['bridge'].append(genre)
    return patterns

# 32. Example usage
print("\Genre similarity patterns:")
patterns = analyze_similarity_patterns(similarity_matrix)
print("Clusters:")
for genre in patterns['clusters']:
    print(f"{genre}")
print("Isolated:")
for genre in patterns['isolated']:
    print(f"{genre}")
print("Bridge:")
for genre in patterns['bridge']:
    print(f"{genre}")
```

```
Genre similarity patterns:
Clusters:
romance
thriller
science_fiction
fantasy
mystery
horse
poem
cook
travel
dog
drama
fiction
abstract
short_story
autobiography
diary
western
Isolated:
Bridge:
```

```
# 33. Function to find genre similarity trends
def analyze_genre_similarity_trends(similarity_matrix):
    genres = list(similarity_matrix.keys)
    trends = {
        'most_consistent': {},
        'most_variable': {},
        'average_similarity': {}
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        variance = sum((s - avg)**2 for s in similarities.values) / len(similarities)
        trends['average_similarity'][genre] = avg
        if variance < 0.1:
            trends['most_consistent'][genre] = variance
        elif variance > 0.5:
            trends['most_variable'][genre] = variance
    return trends

# 34. Example usage
print("\Genre similarity trends:")
trends = analyze_genre_similarity_trends(similarity_matrix)
print("Most consistent genres:")
for genre, variance in trends['most_consistent'].items:
    print(f"{genre}: {variance:.3f}")
print("Most variable genres:")
for genre, variance in trends['most_variable'].items:
    print(f"{genre}: {variance:.3f}")
print("Average similarity:")
for genre, avg in trends['average_similarity'].items:
    print(f"{genre}: {avg:.3f}")
```

```
Genre similarity trends:
Most consistent genres:
romance: 0.000
thriller: 0.000
science_fiction: 0.000
fantasy: 0.000
mystery: 0.000
horse: 0.000
poem: 0.000
cook: 0.000
travel: 0.000
dog: 0.000
drama: 0.000
fiction: 0.000
abstract: 0.000
short_story: 0.000
autobiography: 0.000
diary: 0.000
western: 0.000
Most variable genres:
Average similarity:
romance: 1.000
thriller: 1.000
science_fiction: 1.000
fantasy: 1.000
mystery: 1.000
horse: 1.000
poem: 1.000
cook: 1.000
travel: 1.000
dog: 1.000
drama: 1.000
fiction: 1.000
abstract: 1.000
short_story: 1.000
autobiography: 1.000
diary: 1.000
western: 1.000
```

```
# 35. Function to find genre similarity patterns
def analyze_similarity_patterns(similarity_matrix):
    genres = list(similarity_matrix.keys)
    patterns = {
        'clusters': [],
        'isolated': [],
        'bridge': []
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        if avg > 0.8:
            patterns['clusters'].append(genre)
        elif avg < 0.2:
            patterns['isolated'].append(genre)
        else:
            patterns['bridge'].append(genre)
    return patterns

# 36. Example usage
print("\Genre similarity patterns:")
patterns = analyze_similarity_patterns(similarity_matrix)
print("Clusters:")
for genre in patterns['clusters']:
    print(f"{genre}")
print("Isolated:")
for genre in patterns['isolated']:
    print(f"{genre}")
print("Bridge:")
for genre in patterns['bridge']:
    print(f"{genre}")
```

```
Genre similarity patterns:
Clusters:
romance
thriller
science_fiction
fantasy
mystery
horse
poem
cook
travel
dog
drama
fiction
abstract
short_story
autobiography
diary
western
Isolated:
Bridge:
```

```
# 37. Function to find genre similarity clusters
def find_similarity_clusters(similarity_matrix, threshold=0.5):
    genres = list(similarity_matrix.keys)
    clusters = []
    visited = set()
    for genre in genres:
        if genre not in visited:
            cluster = [genre]
            visited.add(genre)
            for other_genre in genres:
                if other_genre not in visited:
                    similarity = similarity_matrix[genre][other_genre]
                    if similarity > threshold:
                        cluster.append(other_genre)
                        visited.add(other_genre)
            clusters.append(cluster)
    return clusters

# 38. Example usage
print("\Genre similarity clusters:")
clusters = find_similarity_clusters(similarity_matrix, 0.5)
for i, cluster in enumerate(clusters):
    print(f"Cluster {i+1}: {', ".join(cluster)}")
```

```
Genre similarity clusters:
```

```
# 39. Function to analyze genre similarity patterns
def analyze_similarity_patterns(similarity_matrix):
    genres = list(similarity_matrix.keys)
    patterns = {
        'clusters': [],
        'isolated': [],
        'bridge': []
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        if avg > 0.8:
            patterns['clusters'].append(genre)
        elif avg < 0.2:
            patterns['isolated'].append(genre)
        else:
            patterns['bridge'].append(genre)
    return patterns

# 40. Example usage
print("\Genre similarity patterns:")
patterns = analyze_similarity_patterns(similarity_matrix)
print("Clusters:")
for genre in patterns['clusters']:
    print(f"{genre}")
print("Isolated:")
for genre in patterns['isolated']:
    print(f"{genre}")
print("Bridge:")
for genre in patterns['bridge']:
    print(f"{genre}")
```

```
Genre similarity patterns:
Clusters:
romance
thriller
science_fiction
fantasy
mystery
horse
poem
cook
travel
dog
drama
fiction
abstract
short_story
autobiography
diary
western
Isolated:
Bridge:
```

```
# 41. Function to find genre similarity trends
def analyze_genre_similarity_trends(similarity_matrix):
    genres = list(similarity_matrix.keys)
    trends = {
        'most_consistent': {},
        'most_variable': {},
        'average_similarity': {}
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        variance = sum((s - avg)**2 for s in similarities.values) / len(similarities)
        trends['average_similarity'][genre] = avg
        if variance < 0.1:
            trends['most_consistent'][genre] = variance
        elif variance > 0.5:
            trends['most_variable'][genre] = variance
    return trends

# 42. Example usage
print("\Genre similarity trends:")
trends = analyze_genre_similarity_trends(similarity_matrix)
print("Most consistent genres:")
for genre, variance in trends['most_consistent'].items:
    print(f"{genre}: {variance:.3f}")
print("Most variable genres:")
for genre, variance in trends['most_variable'].items:
    print(f"{genre}: {variance:.3f}")
print("Average similarity:")
for genre, avg in trends['average_similarity'].items:
    print(f"{genre}: {avg:.3f}")
```

```
Genre similarity trends:
Most consistent genres:
romance: 0.000
thriller: 0.000
science_fiction: 0.000
fantasy: 0.000
mystery: 0.000
horse: 0.000
poem: 0.000
cook: 0.000
travel: 0.000
dog: 0.000
drama: 0.000
fiction: 0.000
abstract: 0.000
short_story: 0.000
autobiography: 0.000
diary: 0.000
western: 0.000
Most variable genres:
Average similarity:
romance: 1.000
thriller: 1.000
science_fiction: 1.000
fantasy: 1.000
mystery: 1.000
horse: 1.000
poem: 1.000
cook: 1.000
travel: 1.000
dog: 1.000
drama: 1.000
fiction: 1.000
abstract: 1.000
short_story: 1.000
autobiography: 1.000
diary: 1.000
western: 1.000
```



```
# 43. Function to find genre similarity patterns
def analyze_similarity_patterns(similarity_matrix):
    genres = list(similarity_matrix.keys)
    patterns = {
        'clusters': [],
        'isolated': [],
        'bridge': []
    }
    for genre in genres:
        similarities = similarity_matrix[genre]
        avg = sum(similarities.values) / len(similarities)
        if avg > 0.8:
            patterns['clusters'].append(genre)
        elif avg < 0.2:
            patterns['isolated'].append(genre)
        else:
            patterns['bridge'].append(genre)
    return patterns

# 44. Example usage
print("\Genre similarity patterns:")
patterns = analyze_similarity_patterns(similarity_matrix)
print("Clusters:")
for genre in patterns['clusters']:
    print(f"{genre}")
print("Isolated:")
for genre in patterns['isolated']:
    print(f"{genre}")
print("Bridge:")
for genre in patterns['bridge']:
    print(f"{genre}")
```

```
Genre similarity patterns:
Clusters:
romance
thriller
science_fiction
fantasy
mystery
horse
poem
cook
travel
dog
drama
fiction
abstract
short_story
autobiography
diary
western
Isolated:
Bridge:
```

```
In [14]: [item.shape[0] for item in X_train] # Series dimensions before train_data
```

```
[1000,
 1000,
 1000,
 
```

[2019-03-25 17:53:20.353]# C:\Users\Administrator\Desktop\ftp\ftpserver\server\tmp_group\tmp_gameserver_requests\1-127.0.0.1-7002
0.00/0MB(0.00/0MB),peak:0.00 ,set:50,used:1
DATE:2019-03-25 RUNTIME:00:00
CMD:D:\Erlang/erl.exe +A64 +P65536 -pa ./ebin ./deps/*/ebin -boot start_sasl -config ./priv/app.config -s lager -s ranch -s crypto -s mysql -s cowboy -s game_server -s game_server_app -s cowlib -s ssl -s public_key -s asn1 -s inets -s goldrush -s gen_logger -s tcp_acceptor_pool -s rfc4627_jsonrpc -s erlang_memcached -s erlang_redis_client -s poolboy -s eredis -s mnesia -s game_server_sup -s cowboy_sup -s game_server_cowboy_app -s game_server_tcp_client_sup -s tcp_client_sup -s tcp_acceptor_sup -s tcp_listener_sup -s game_server_tcp_listener_sup -s tcp_listener_sup -s game_server_tcp_listener_sup -s tcp_listener_sup -s game_server_tcp_listener

```
Epoch 25/50 val_loss improved from inf to 0.7874,
Epoch 26/50 val_loss improved from 0.7874 to 0.7574,
Epoch 27/50 val_loss improved from 0.7574 to 0.7334, saving model to /content/best_weights.h5
Epoch 28/50 val_loss improved from 0.7334 to 0.7156,
Epoch 29/50 val_loss improved from 0.7156 to 0.7000,
Epoch 30/50 val_loss improved from 0.7000 to 0.6884,
Epoch 31/50 val_loss improved from 0.6884 to 0.6784,
Epoch 32/50 val_loss improved from 0.6784 to 0.6696,
Epoch 33/50 val_loss improved from 0.6696 to 0.6616, saving model to /content/best_weights.h5
Epoch 34/50 val_loss improved from 0.6616 to 0.6544,
Epoch 35/50 val_loss improved from 0.6544 to 0.6479, saving model to /content/best_weights.h5
Epoch 36/50 val_loss improved from 0.6479 to 0.6420, saving model to /content/best_weights.h5
Epoch 37/50 val_loss improved from 0.6420 to 0.6367, saving model to /content/best_weights.h5
Epoch 38/50 val_loss improved from 0.6367 to 0.6318, saving model to /content/best_weights.h5
Epoch 39/50 val_loss improved from 0.6318 to 0.6273, saving model to /content/best_weights.h5
Epoch 40/50 val_loss improved from 0.6273 to 0.6232, saving model to /content/best_weights.h5
Epoch 41/50 val_loss improved from 0.6232 to 0.6194,
Epoch 42/50 val_loss improved from 0.6194 to 0.6159, saving model to /content/best_weights.h5
Epoch 43/50 val_loss improved from 0.6159 to 0.6126,
Epoch 44/50 val_loss improved from 0.6126 to 0.6095,
Epoch 45/50 val_loss improved from 0.6095 to 0.6065,
Epoch 46/50 val_loss improved from 0.6065 to 0.6037,
Epoch 47/50 val_loss improved from 0.6037 to 0.6010,
Epoch 48/50 val_loss improved from 0.6010 to 0.5984,
Epoch 49/50 val_loss improved from 0.5984 to 0.5959,
Epoch 50/50 val_loss improved from 0.5959 to 0.5935,
```

```
[ 0.7143 0.7143 0.7143 0.7143 0.7143 0.7143 0.7143 0.7143
```

```
def main():
    # Load data
    train = pd.read_csv('train.csv')
    test = pd.read_csv('test.csv')

    # Prepare features
    X = train.drop(['id', 'Id', 'SalePrice'], axis=1)
    y = np.log1p(train['SalePrice'])  # Using log1p to handle zero values
    X_test = test.drop(['id', 'Id'], axis=1)

    # Fill missing values with median
    for col in X.columns:
        if X[col].dtype == 'object':
            X[col] = X[col].fillna('missing')
            X_test[col] = X_test[col].fillna('missing')
        else:
            median = X[col].median()
            X[col] = X[col].fillna(median)
            X_test[col] = X_test[col].fillna(median)

    # Convert categorical variables to dummies
    X = pd.get_dummies(X)
    X_test = pd.get_dummies(X_test)

    # Align test set with training set
    X, X_test = X.align(X_test, join='left')

    # Split data for validation
    X_train, X_val = train_test_split(X, test_size=0.2, random_state=42)
    y_train, y_val = train_test_split(y, test_size=0.2, random_state=42)

    # Create and train model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predict on validation set
    val_pred = model.predict(X_val)

    # Calculate RMSE
    rmse = np.sqrt(mean_squared_error(y_val, val_pred))
    print(f"Validation RMSE: {rmse}")

    # Predict on test set
    test_pred = model.predict(X_test)

    # Create submission file
    submission = pd.DataFrame({'Id': test['Id'], 'SalePrice': np.expm1(test_pred)})
    submission.to_csv('submission.csv', index=False)

    print("Submission file created!")

if __name__ == "__main__":
    main()
```

```
python3 train.py
/home/agent/conda/envs/mlrc/lib/python3.11/site-packages/sklearn/linear_model/_base.py:59: FutureWarning:
'DataFrame.fillna(lineage_id_1, lineage_id_2, ...)' is deprecated and will raise in a future version.
Use 'obj.fillna(lineage_id_1, lineage_id_2, ...)' instead.
```