Fusion vs Sequential Architectures for Image Classification

Nicolas Acevedo
Juan Camilo Niño

Abstract

This project develops and trains two deep learning models for image classification using pre-trained convolutional neural network backbones. The first model employs a parallel fusion strategy, combining MobileNetV2 and VGG16 features, while the second uses a sequential configuration to offer a different way of feature integration. Both models undergo a two-phase training process with frozen backbones followed by fine-tuning. The dataset is stratified to preserve class distributions, and the models are evaluated using accuracy, macro-F1, and confusion matrices.

1 Introduction

The main goal of this work is to compare two strategies for leveraging pre-trained convolutional neural networks for image classification tasks: parallel fusion and sequential integration. Pre-trained models such as MobileNetV2 and VGG16 provide strong feature extractors. By either combining their outputs in parallel or arranging them in sequence, the project aims to assess the impact of architectural design on performance.

2 Dependencies and Imports

The implementation uses TensorFlow for model building and training, NumPy for array manipulation, Matplotlib for visualization, and Hugging Face Datasets for loading the image dataset. Scikit-learn provides tools for stratified data splitting and metric computation.

3 Hyperparameters and Utilities

Hyperparameters such as seed values, image size (224×224), batch size, learning rates, and number of epochs are defined to ensure reproducibility and control of the training process. Utility functions handle data visualization, stratified splitting, and metric computation (accuracy, F1-score, confusion matrices).

4 Dataset Loading and Stratified Splitting

The dataset is loaded and partitioned using a stratified splitting strategy. This ensures that each subset (training, validation, and testing) preserves the original class distribution, which is especially important in the presence of class imbalance.

5 Data Preprocessing and Pipelines

Images are resized to 224×224 pixels and preprocessed according to the requirements of each backbone. TensorFlow data pipelines (`tf.data`) are constructed with `map`, `batch`, and `prefetch` to provide efficient GPU data feeding during training.

6 Fusion Model Construction

The fusion model uses two pre-trained backbones (MobileNetV2 and VGG16) without their top classification layers. Feature maps from both networks are globally pooled and concatenated. The resulting feature vector is passed through dense layers and a final softmax output layer.

```
def build_fusion_model(num_classes, freeze_backbones=True):
    mobilenet = tf.keras.applications.MobileNetV2(
        include_top=False, weights='imagenet', input_shape
   =(224,224,3))
    vgg = tf.keras.applications.VGG16(
        include_top=False, weights='imagenet', input_shape
   =(224,224,3))

    if freeze_backbones:
```

```
        mobilenet.trainable = False
        vgg.trainable = False

    x1 = tf.keras.layers.GlobalAveragePooling2D()(mobilenet.output)
    x2 = tf.keras.layers.GlobalAveragePooling2D()(vgg.output)
    x = tf.keras.layers.Concatenate()([x1, x2])
    x = tf.keras.layers.Dense(256, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    outputs = tf.keras.layers.Dense(num_classes, activation='softmax')(x)

    model = tf.keras.Model(inputs=[mobilenet.input, vgg.input], outputs=outputs)
    return model
```

Listing 1: Fusion model construction

7 Sequential Model Construction

The sequential model follows an alternative architectural strategy, where feature extraction is structured differently to compare with the fusion approach. While details differ, the training procedure remains the same for fair evaluation.

```
def build_sequential_model(num_classes, freeze_backbone=True):
    base_model = tf.keras.applications.MobileNetV2(
        include_top=False, weights='imagenet', input_shape=(224,224,3))
    if freeze_backbone:
        base_model.trainable = False

    x = base_model.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(256, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    outputs = tf.keras.layers.Dense(num_classes, activation='softmax')(x)

    model = tf.keras.Model(inputs=base_model.input, outputs=outputs)
    return model
```

Listing 2: Sequential model construction

8 Training Strategy

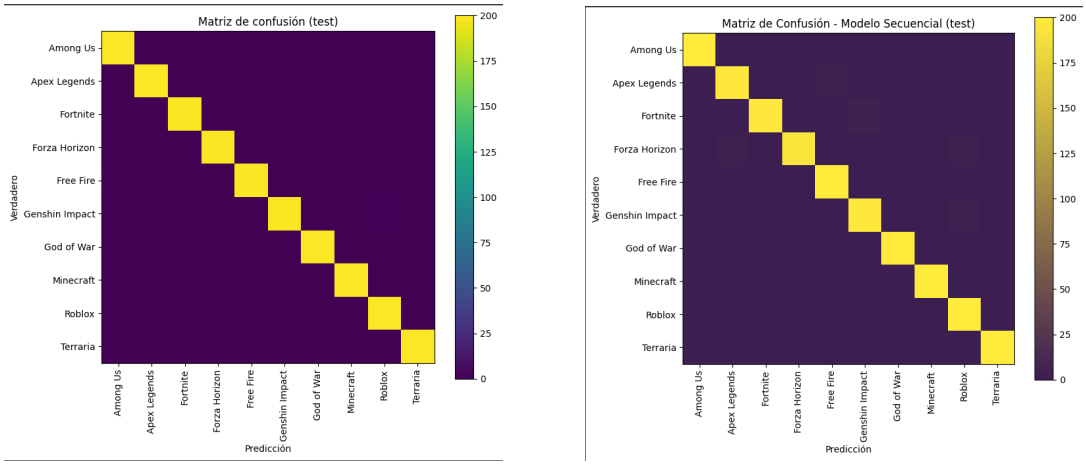
Training is performed in two phases:

1. **Phase 1 (Frozen backbones):** The pre-trained feature extractors are frozen and only the new dense layers are trained. This prevents large weight updates from damaging the pre-trained representations.

2. **Phase 2 (Fine-tuning):** The last layers of the backbones are unfrozen and trained with a reduced learning rate. This allows the models to adapt more finely to the specific dataset.

9 Evaluation and Metrics

Both models are evaluated on a held-out test set using accuracy, macro-F1 score, and confusion matrices. The fusion model achieved an accuracy of 0.9975 and a macro-F1 score of 0.9975, outperforming the sequential model, which obtained an accuracy of 0.9895 and a macro-F1 score of 0.9895.



(a) Confusion matrix — Fusion model (b) Confusion matrix — Sequential model

Figure 1: Confusion matrices for both models on the held-out test set.

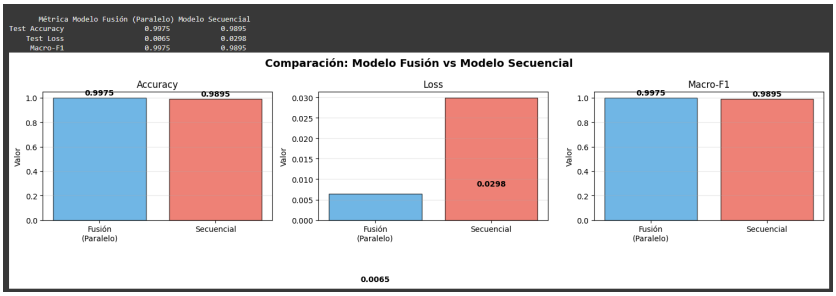


Figure 2: Comparison of main metrics (Accuracy, Macro-F1, Loss) between the fusion and sequential models.

10 Conclusion and Future Work

The parallel fusion approach combining MobileNetV2 and VGG16 features demonstrated superior performance compared to the sequential configuration. This suggests that combining complementary feature representations enhances classification accuracy. Future work includes exploring more diverse backbones, applying stronger regularization techniques, and experimenting with data augmentation strategies to further improve robustness.