Sports Image Classification using Transfer Learning

100-Class Classification with MobileNetV2

Project Objective

- Goal: Build a Deep Learning model to accurately classify images into 100 different sports categories.
- Approach: Utilize Transfer Learning to leverage pre-existing knowledge from large datasets (ImageNet) for a specific domain task.
- Tech Stack: Python, TensorFlow, Keras, Matplotlib, Scikit-Learn.

Dataset Overview

Source: Kaggle Sports Classification Dataset.

Structure:

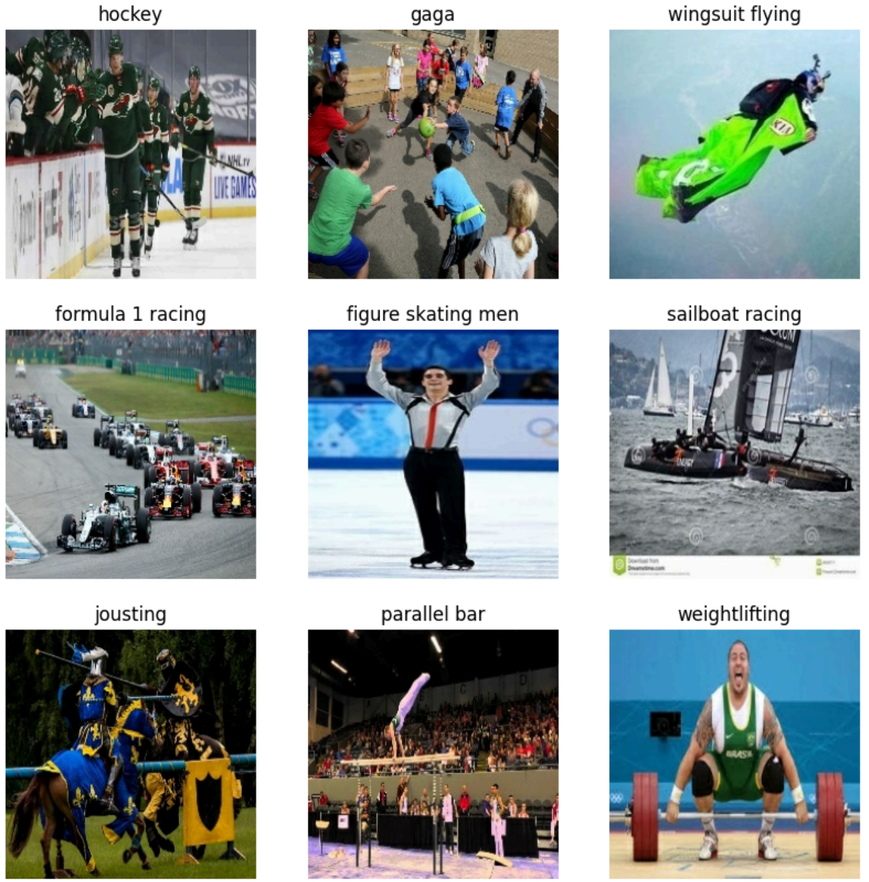
- Total Classes: 100 (e.g., archery, baseball, formula 1, swimming).
- Image Dimensions: 224 x 224 pixels (RGB).

Data Split:

- Training Set: 13,492 images.
- Validation Set: 500 images (5 per class).
- Test Set: 500 images (5 per class).

Data Exploration

- Diversity: The dataset covers a wide range of sports, from "Air Hockey" to "Baton Twirling."
- Class Names: Extracted directly from folder directory structure.
- Visual Inspection:
  - Images are colored and varied in angles/lighting.
  - Labels are mapped to integer indices (0-99).



Data Preprocessing Pipeline

1. NORMALIZATION: PIXEL VALUES RESCALE FROM [0, 255] TO [0, 1] (DIVIDING BY 255.0).

2. Performance Optimization:
   - Caching: Stores data in memory to prevent disk bottlenecks.
   - Shuffling: Randomizes training order (buffer size 100).
   - Prefetching (AUTOTUNE): Prepares the next batch of data while the GPU is processing the current one.

3. BATCH SIZE: 16 IMAGES PER STEP.

Model Architecture Strategy

Base Model: MobileNetV2

- Selected for efficiency and speed.
- Pre-trained on ImageNet weights.
- Frozen State: Base layers set to trainable = False to preserve feature extractors.

Custom Classification Head:

- GlobalAveragePooling2D: To reduce spatial dimensions.
- Dropout(0.3): To prevent overfitting.
- Dense(100, activation='softmax'): Output layer for 100 distinct classes.

TRAINING CONFIGURATION

1. Optimizer: Adam (learning_rate=0.01).
2. Loss Function: Sparse Categorical Crossentropy (ideal for non-one-hot encoded integer labels).
3. Metrics: Accuracy.
4. Epochs: Set to 50 max, but controlled by callbacks.

Callbacks & Training Control

TO ENSURE OPTIMAL CONVERGENCE AND PREVENT OVERFITTING:

EarlyStopping:

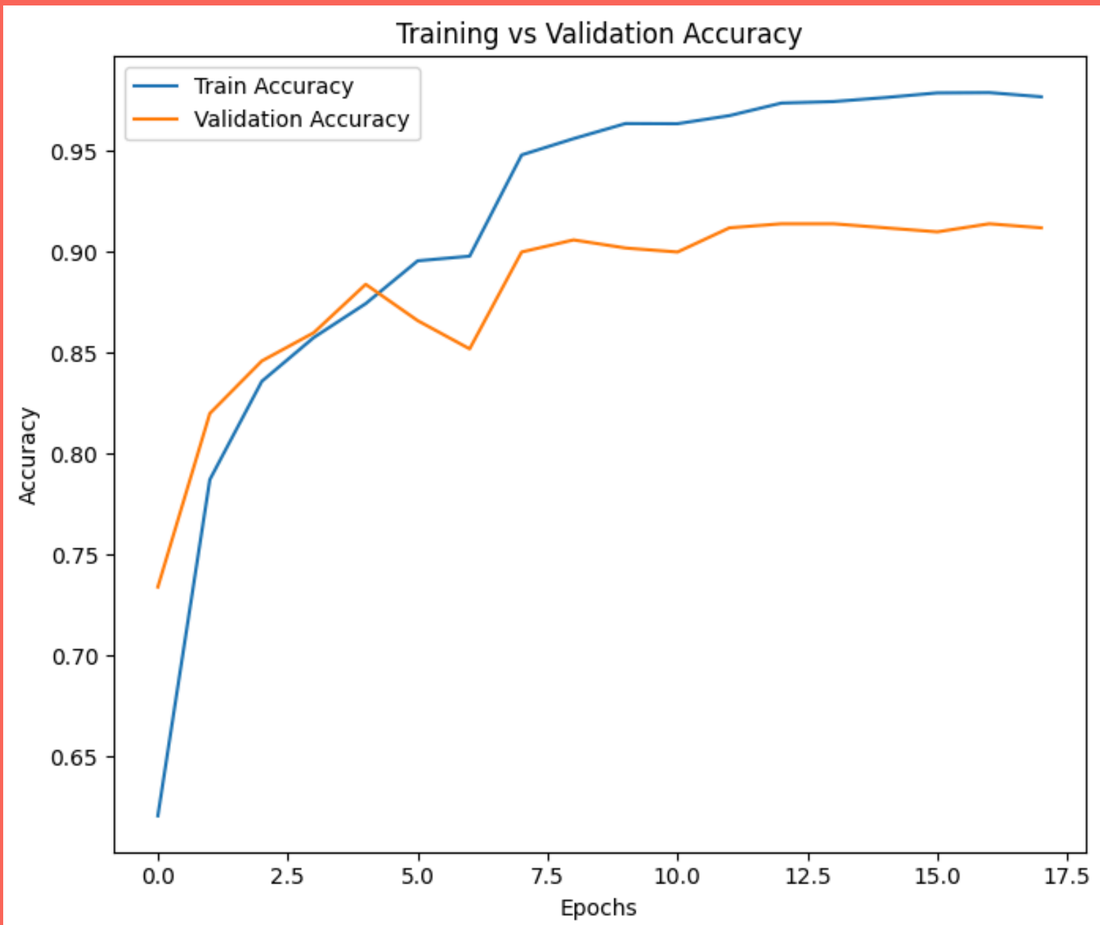
- Monitors val_accuracy.
- Stops training if no improvement for 5 epochs.
- Restores best weights automatically.

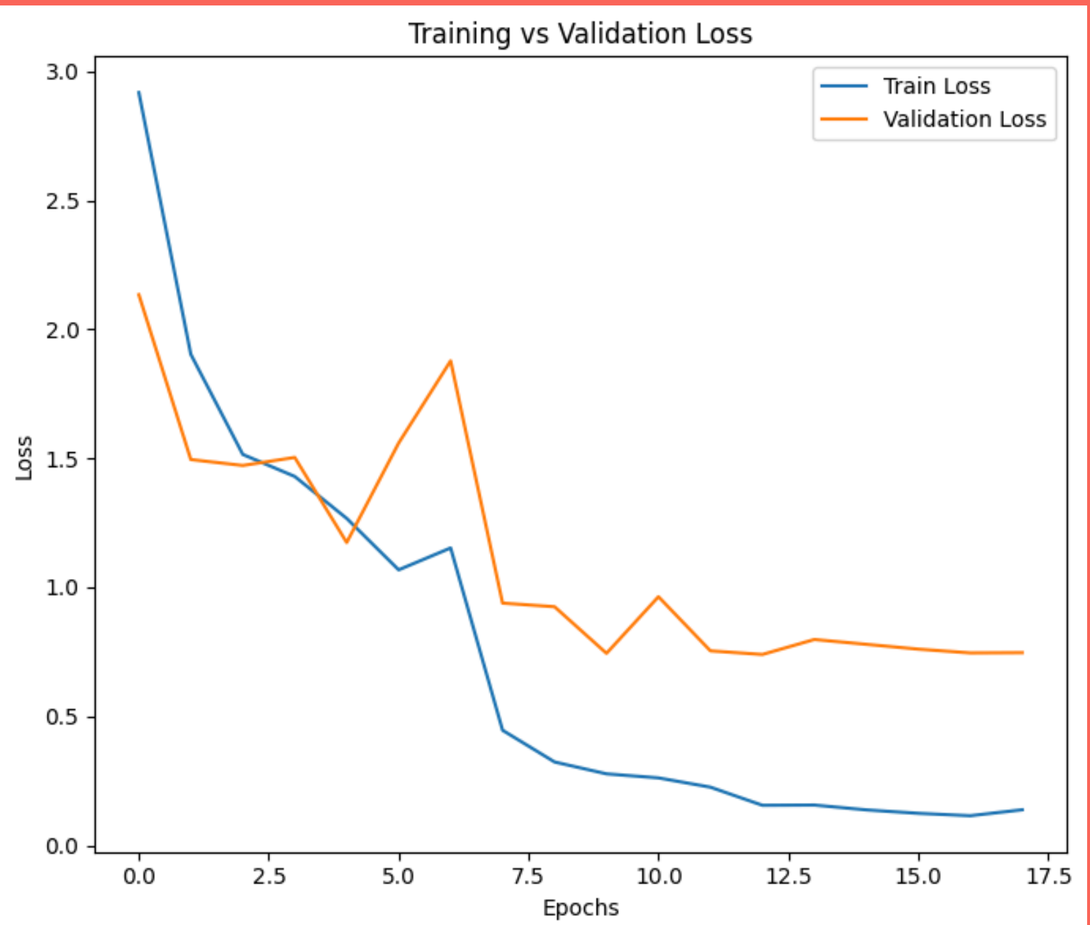
ReduceLROnPlateau:

- Monitors val_loss.
- Reduces learning rate by factor of 0.2 if performance stalls for 2 epochs.

TqdmCallback: For progress visualization.

Training vs. Validation Curves





- Training accuracy peaked at 97.29%.
- Validation accuracy plateaued around 91.40%.

Visualizing Predictions

Final Model Performance

The best model weights were restored and evaluated on unseen data:

Metric	Score
Best Validation Accuracy	9.140%
Best Validation Loss	7.397
Final Test Accuracy	9.420%
Final Test Loss	6.490

Conclusion

- Success: Successfully fine-tuned MobileNetV2 for a 100-class problem.
- Efficiency: Achieved >94% accuracy in under 20 epochs.
- Robustness: The implementation of dynamic learning rates and early stopping prevented wasted computational resources.
- Deliverable: Model saved as mobilenet_best.keras for deployment.

THANK YOU