100 SPORTS IMAGE CLASSIFICATION USING EFFICIENTNETB0 AND RESNET50

Exploring Ensemble Learning and Fine-Tuning

By
John Yare
& Joshua Adegbola

OVERVIEW

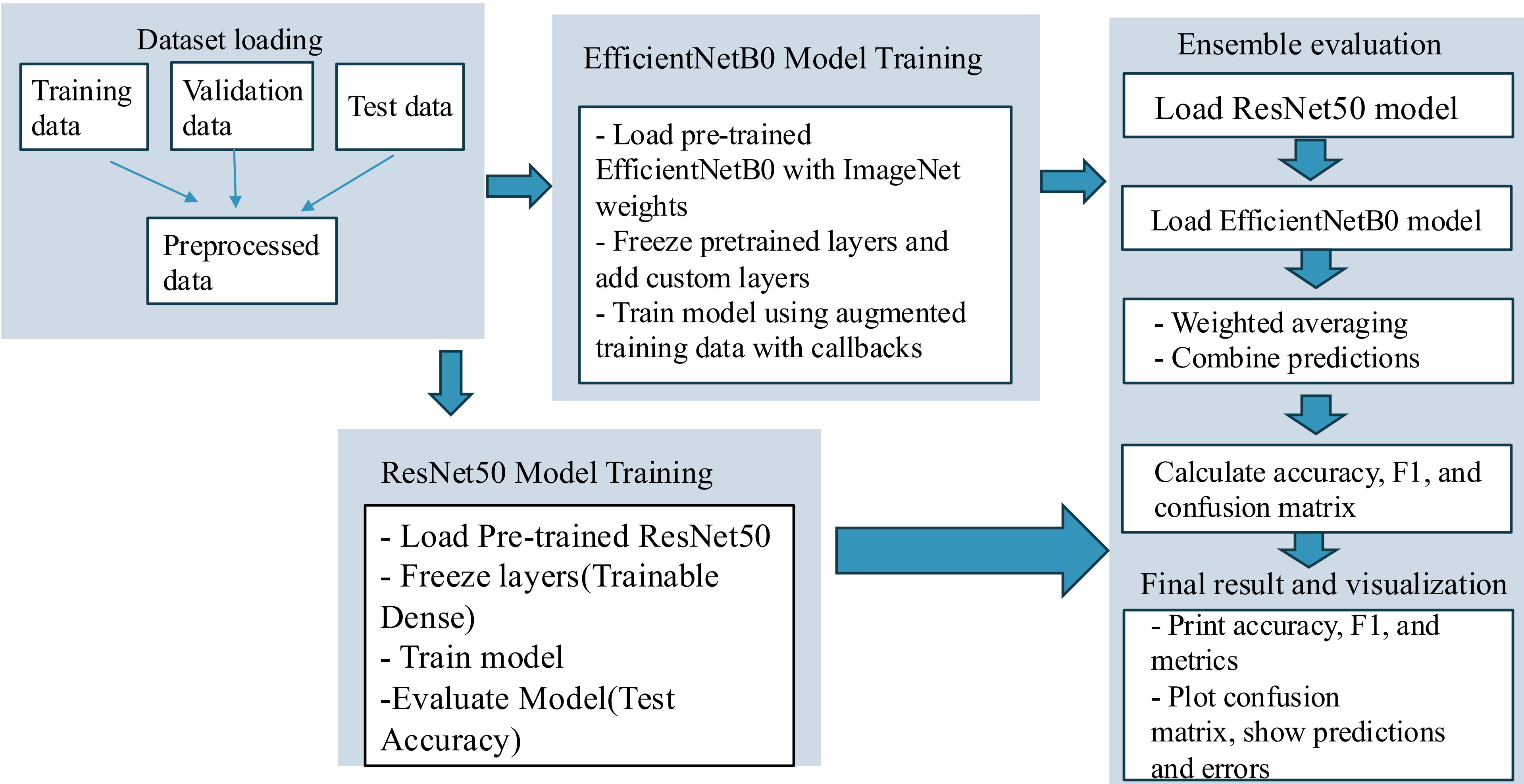
- Objective: Classify 100 sports image categories using an ensemble model by combining predictions of a Vision Transformer alternative model available in keras (EfficientNetB0) and ResNet50 model.

- Models: EfficientNetB0 and ResNet50.

- Dataset:
  - Initial manipulated dataset had less training data.
  - Reverted to baseline dataset for better performance and more variety of data in training.

- Achievements:
  - Achieved 92.6% test accuracy with EfficientNetB0 model.
  - Achieved 96.2% test accuracy with ResNet50 model.
  - Combined predictions from EfficientNetB0 and ResNet50 by optimizing weights for each model, based on each model's validation performances, into an ensemble model to improve the model's performance.
  - Achieved 90% test accuracy with ensemble model.

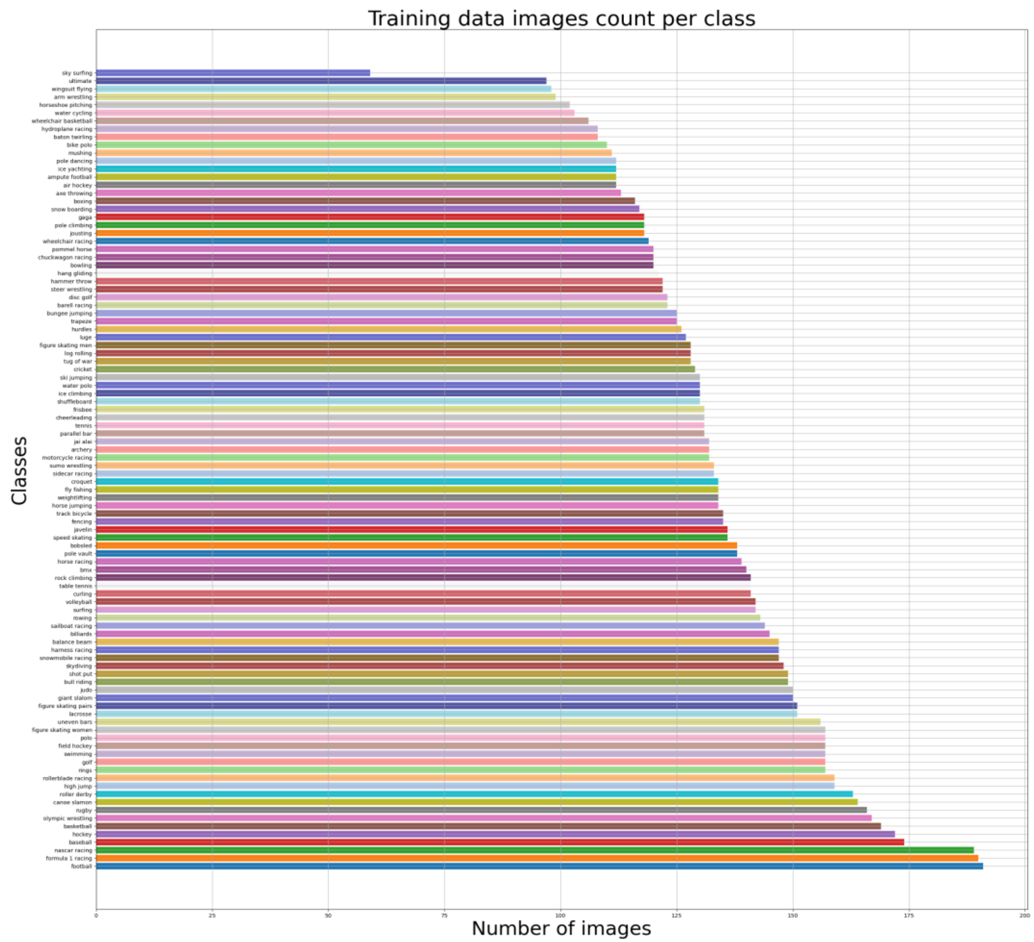
BLOCK DIAGRAMS



Block Diagram: Dataset Loading to Ensemble Evaluation

DATASET PREPARATION

- ❖Dataset paths for training, validation, and testing.
- ❖Used Label Encoder for class label encoding.
- ❖Visualized class distribution in training data.



Bar Chart: Image Count Per Class

DATA PREPROCESSING AND AUGMENTATION

```python
# Data augmentation layer for applying random transformations to input images
augment = tf.keras.Sequential([
    RandomRotation(0.1),  # Randomly rotate images by 10%
    RandomZoom(0.1),  # Randomly zoom into images
    RandomContrast(0.1),  # Randomly adjust image contrast
    RandomFlip("horizontal_and_vertical") # Randomly flip images horizontally and vertically
], name='AugmentationLayer')

# Define the input layer for the model
inputs = layers.Input(shape = (224,224,3), name='inputLayer')

# Pass inputs through the augmentation layer
x = augment(inputs)

# Pass the augmented inputs through the pretrained model
pretrain_out = pretrained_model(x, training = False)

# Add a fully connected dense layer for feature extraction
x = layers.Dense(350)(pretrain_out)
x = layers.Activation(activation="relu")(x)  # Apply ReLU activation
x = BatchNormalization()(x) # Normalize activations to speed up training
x = layers.Dropout(0.25)(x) # Add dropout to reduce overfitting

# Add an output layer with the number of classes (num_classes)
x = layers.Dense(num_classes)(x)
outputs = layers.Activation(activation="softmax", dtype=tf.float32, name='activationLayer')(x) # mixed_precision need separated Dense and Activation layers

# Create the final model
model = Model(inputs=inputs, outputs=outputs)
```

Diagram: Augmentation Applied to Training Data Only

- Challenges with direct augmentation:
  - Reduced training and validation accuracy.
  - Increased losses indicated no learning.
- Solution:
  - Augmentation applied after the input layer for training data only.

EFFICIENTNETB0 MODEL ARCHITECTURE

- Base Model: Pretrained on ImageNet.
- Custom Layers:
  - Dense layers for feature extraction.
  - Dropout for regularization.
  - Batch Normalization for stability.
  - Softmax activation for classification.
- Fine-tuning with frozen Batch Normalization layers to avoid instability.

Model: "functional_1"

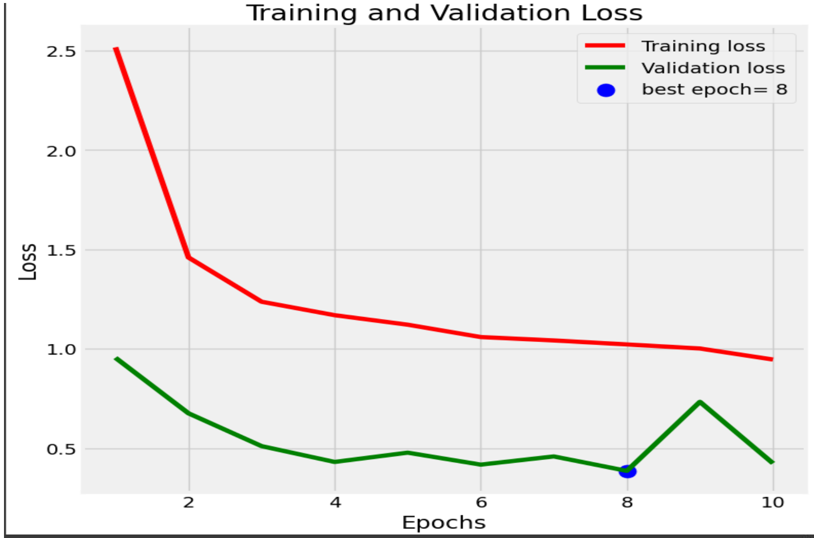
Layer (type)	Output Shape	Param #
inputLayer (InputLayer)	(None, 224, 224, 3)	0
cast_1 (Cast)	(None, 224, 224, 3)	0
AugmentationLayer (Sequential)	(None, 224, 224, 3)	0
efficientnetb0 (Functional)	(None, 1280)	4,049,571
dense (Dense)	(None, 350)	448,350
activation (Activation)	(None, 350)	0
batch_normalization (BatchNormalization)	(None, 350)	1,400
dropout (Dropout)	(None, 350)	0
dense_1 (Dense)	(None, 100)	35,100
cast_2 (Cast)	(None, 100)	0
activationLayer (Activation)	(None, 100)	0

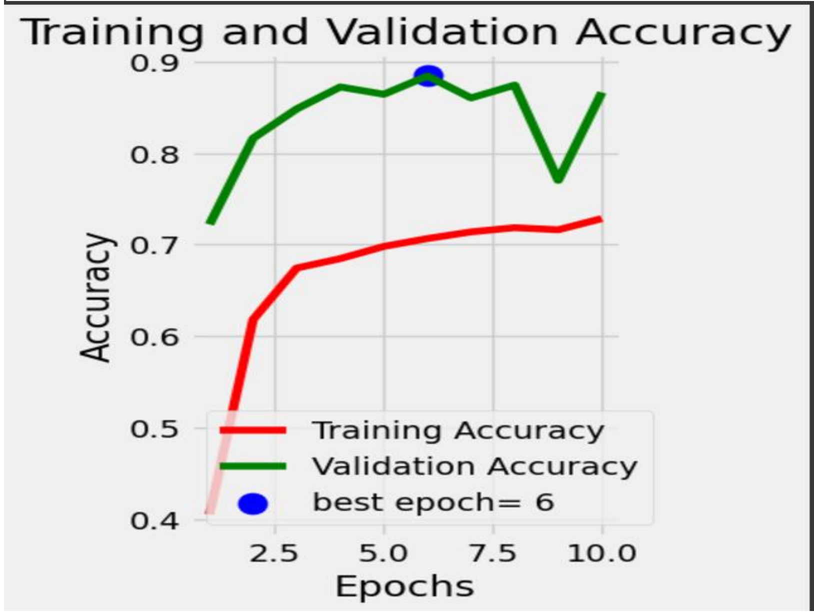
Total params: 4,534,421 (17.30 MB)
Trainable params: 484,150 (1.85 MB)
Non-trainable params: 4,050,271 (15.45 MB)

Diagram: Model summary

TRAINING THE EFFICIENTNETB0 MODEL

- Training settings:
  - Optimizer: Adam (learning rate: 0.0005).
  - Batch size: 10.
  - Epochs: 10.
- Callbacks:
  - EarlyStopping to prevent overfitting.
  - ReduceLROnPlateau for dynamic learning rate adjustment.
- Results:
  - Improved training and validation performance.





Plots: Training and Validation Loss/Accuracy

FINE-TUNING THE EFFICIENTNETB0 MODEL

- Unfroze EfficientNetB0 layers while freezing BatchNormalization.
- Reduced learning rate to 0.0001 for stability.
- Reused EarlyStopping and ReduceLROnPlateau callbacks.
- Results:
  - Further improvement in validation accuracy and loss reduction.

Fine-tuned Training and Validation Performance Plots

Test accuracy performanxe for EfficientNetB0 model

Evaluating the Model

Evaluate the model on the test set

```
[ ] results = model.evaluate(test_images, verbose=0)

    print("    Test Loss: {:.5f}".format(results[0]))
    print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
    Test Loss: 0.28631
Test Accuracy: 92.60%
```

MODEL ARCHITECTURE: RESNET50

Base Model: ResNet-50 pretrained on ImageNet.

Output Layer: Dense Layer with 100 softmax nodes for classification

Freezing ResNet-50 layers for transfer learning.

Building, Compiling and Training the model

```
model1 = Sequential()

model1.add(ResNet50(include_top = False, pooling = 'avg',
                    weights = resnet_weights_path))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
```

```
model1.add(Dense(100, activation = 'softmax'))
```

```
model1.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 2048)	23,587,712
dense_2 (Dense)	(None, 100)	204,900

Total params: 23,792,612 (90.76 MB)
Trainable params: 23,739,492 (90.56 MB)
Non-trainable params: 53,120 (207.50 KB)

TRAINING SETUP

Loss Function: Categorical Crossentropy.

Optimizer: SGD (Stochastic Gradient Descent).

Metrics: Accuracy.

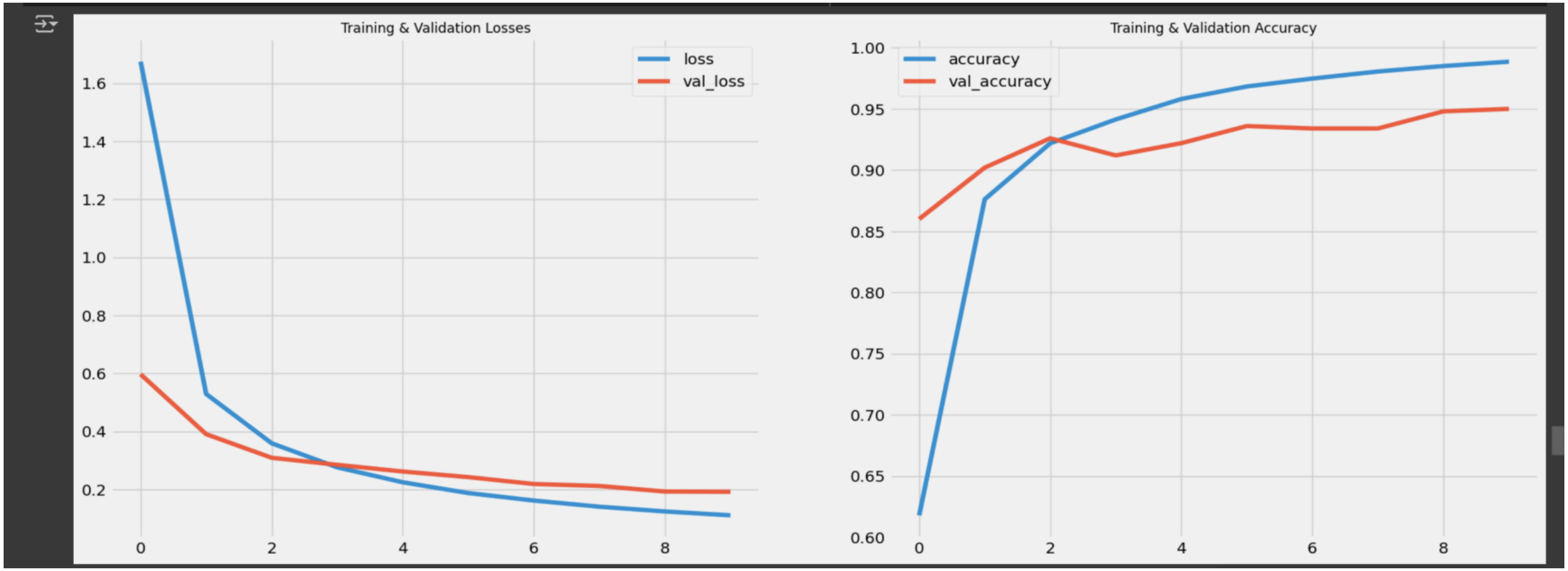
Training for 10 epochs using train and validation datasets.

```
[ ] model1.compile(optimizer= 'sgd',
                   loss = 'categorical_crossentropy',
                   metrics = ['accuracy'])

history1 = model1.fit(train_generator,
                      validation_data = val_generator, epochs = 10)

Epoch 1/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 75s 91ms/step - accuracy: 0.4013 - loss: 2.7818 - val_accuracy: 0.8600 - val_loss: 0.5968
Epoch 2/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 52s 76ms/step - accuracy: 0.8665 - loss: 0.5784 - val_accuracy: 0.9020 - val_loss: 0.3903
Epoch 3/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 52s 76ms/step - accuracy: 0.9206 - loss: 0.3680 - val_accuracy: 0.9260 - val_loss: 0.3090
Epoch 4/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 51s 75ms/step - accuracy: 0.9440 - loss: 0.2819 - val_accuracy: 0.9120 - val_loss: 0.2847
Epoch 5/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 52s 76ms/step - accuracy: 0.9602 - loss: 0.2244 - val_accuracy: 0.9220 - val_loss: 0.2616
Epoch 6/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 53s 78ms/step - accuracy: 0.9705 - loss: 0.1863 - val_accuracy: 0.9360 - val_loss: 0.2423
Epoch 7/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 52s 76ms/step - accuracy: 0.9738 - loss: 0.1628 - val_accuracy: 0.9340 - val_loss: 0.2186
Epoch 8/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 52s 76ms/step - accuracy: 0.9832 - loss: 0.1368 - val_accuracy: 0.9340 - val_loss: 0.2118
Epoch 9/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 52s 76ms/step - accuracy: 0.9868 - loss: 0.1197 - val_accuracy: 0.9480 - val_loss: 0.1926
Epoch 10/10
675/675 ━━━━━━━━━━━━━━━━━━━━ 52s 76ms/step - accuracy: 0.9893 - loss: 0.1092 - val_accuracy: 0.9500 - val_loss: 0.1917
```

RESULT VISUALIZATION



❖Results:

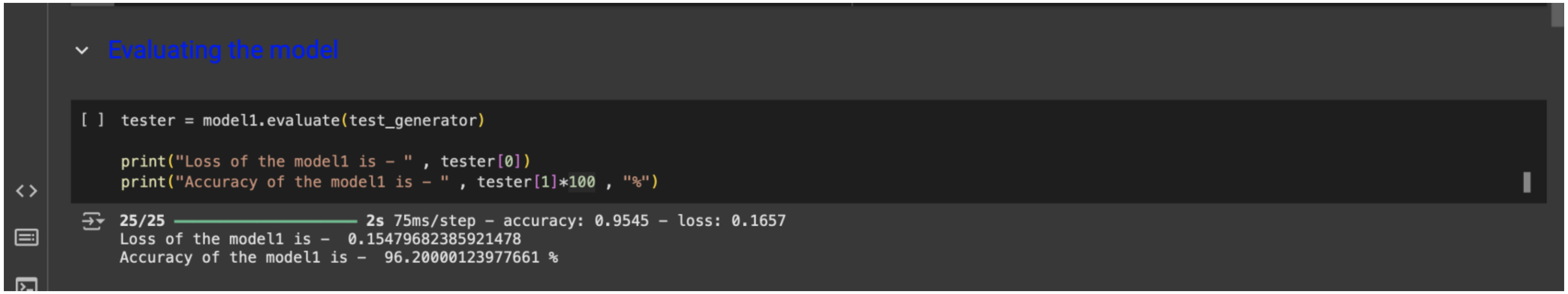
- Improved training and validation performance accuracy

MODEL EVALUATION

Metrics Used

1. **Loss**: Measures the error during the evaluation process.
    1. Lower loss values indicate better model predictions.
    2. Example from the evaluation:
        1. **Loss = 0.1547**

2. **Accuracy**: Percentage of correct predictions made by the model.
    1. High accuracy indicates the model is correctly classifying most of the input images.
    2. Example from the evaluation:
        1. **Accuracy = 96.20%**


Evaluating the model

```
tester = model1.evaluate(test_generator)

print("Loss of the model1 is - " , tester[0])
print("Accuracy of the model1 is - " , tester[1]*100 , "%")
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - accuracy: 0.9545 - loss: 0.1657
Loss of the model1 is -  0.15479682385921478
Accuracy of the model1 is -  96.20000123977661 %
```

COMBINING EFFICIENTNETB0 AND RESNET50 PREDICTIONS

- Ensemble Method:
  - Weighted averaging of predictions from both models.
  - EfficientNetB0 and ResNet50 assigned weights based on each model's validation and prediction performances.
- Results:
  - Achieved a high performance from combined predictions in ensemble model.
- Advantages:
  - Leverages strengths of both models.
  - Reduces individual model biases and overfitting in general.
  - Improves generalization, which makes the combined model likely perform better on unseen data compared to a single model.

DIAGRAM: ENSEMBLE PREDICTION PROCESS

```
# Assuming `test_images` is already defined and properly preprocessed
resnet_predictions = model_resnet.predict(test_images)
#resnet_predictions = model_resnet.predict(test_images)
efficientnet_predictions = model_efficientnet.predict(test_images)

# Weighted averaging for ensemble predictions
weights_resnet = 0.6
weights_efficientnet = 0.4
ensemble_predictions = (weights_resnet * resnet_predictions) + (weights_efficientnet * efficientnet_predictions)
```

```
from sklearn.metrics import accuracy_score

# Convert averaged predictions into class labels
ensemble_labels = np.argmax(ensemble_predictions, axis=1)

# Ground truth labels from test data
true_labels = test_images.classes

# Calculate ensemble accuracy
ensemble_accuracy = accuracy_score(true_labels, ensemble_labels)
print(f"Ensemble Model Accuracy: {ensemble_accuracy * 100:.2f}%")

# Calculate ensemble f1 score
ensemble_f1_score = f1_score(true_labels, ensemble_labels, average='macro')
print(f"Ensemble Model F1 Score: {ensemble_f1_score:.4f}")
```

```
Ensemble Model Accuracy: 90.00%
Ensemble Model F1 Score: 0.8976
```

EVALUATION METRICS FOR ENSEMBLE MODEL

- Achieved Accuracy: 90%.
- Metrics explained:
  - Precision: True positives among predicted positives.
  - Recall: True positives among actual positives.
  - F1-Score: Balances precision and recall.
  - Confusion Matrix: Breakdown of true vs. predicted classes.

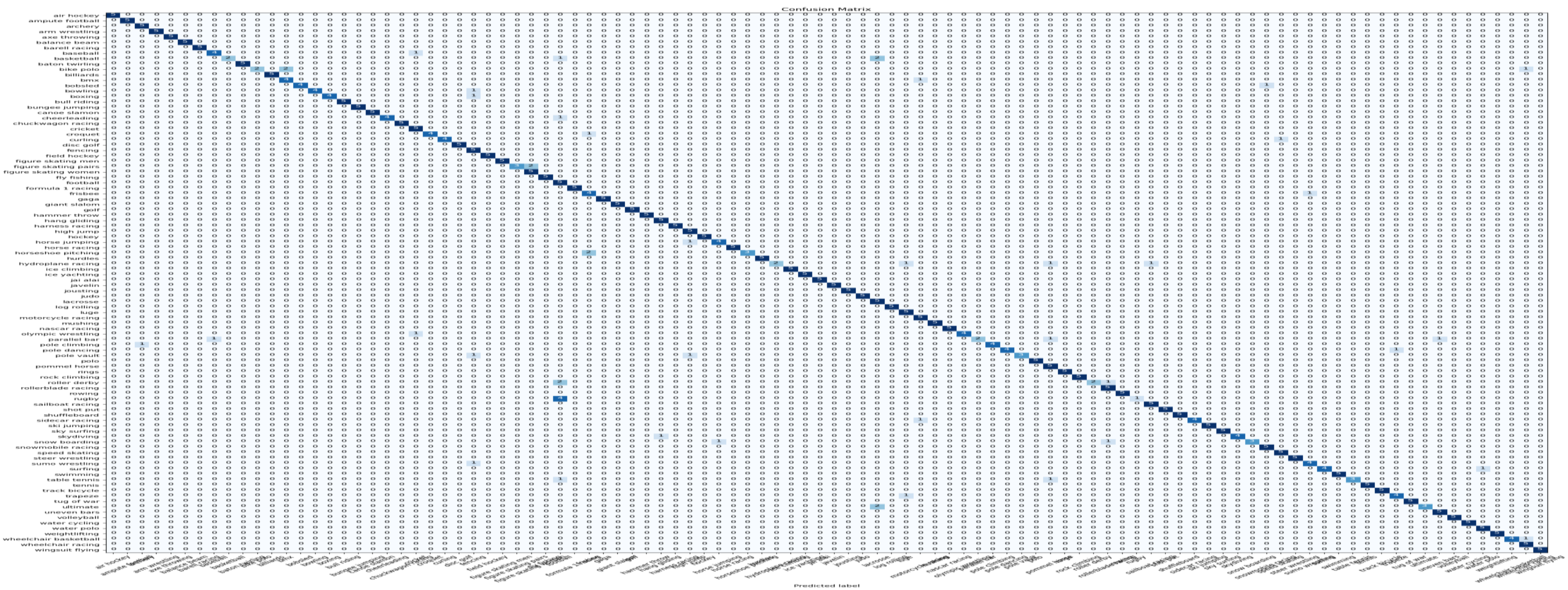
Classification Report:

	precision	recall	f1-score	support
air hockey	1.00	1.00	1.00	5
ampute football	1.00	1.00	1.00	5
archery	0.83	1.00	0.91	5
arm wrestling	1.00	1.00	1.00	5
axe throwing	1.00	1.00	1.00	5
balance beam	1.00	1.00	1.00	5
barell racing	1.00	1.00	1.00	5
baseball	0.80	0.80	0.80	5
basketball	1.00	0.40	0.57	5
baton twirling	1.00	1.00	1.00	5
bike polo	1.00	0.40	0.57	5
billiards	1.00	1.00	1.00	5
bmx	0.67	0.80	0.73	5
bobsled	1.00	0.80	0.89	5
bowling	1.00	0.80	0.89	5
boxing	1.00	0.80	0.89	5
bull riding	1.00	1.00	1.00	5
bungee jumping	1.00	1.00	1.00	5
canoe slamon	1.00	1.00	1.00	5
cheerleading	1.00	0.80	0.89	5
surfing	1.00	0.80	0.89	5
swimming	1.00	1.00	1.00	5
table tennis	1.00	0.60	0.75	5
tennis	1.00	1.00	1.00	5
track bicycle	1.00	1.00	1.00	5
trapeze	0.80	0.80	0.80	5
tug of war	1.00	1.00	1.00	5
ultimate	1.00	0.60	0.75	5
uneven bars	0.83	1.00	0.91	5
volleyball	1.00	1.00	1.00	5
water cycling	1.00	1.00	1.00	5
water polo	0.83	1.00	0.91	5
weightlifting	1.00	1.00	1.00	5
wheelchair basketball	1.00	0.80	0.89	5
wheelchair racing	0.71	1.00	0.83	5
wingsuit flying	1.00	1.00	1.00	5
accuracy			0.90	500
macro avg	0.93	0.90	0.90	500
weighted avg	0.93	0.90	0.90	500

Snippet of Ensemble model Classification Report

VISUALIZING PREDICTIONS

- ❖Visualized predictions from EfficientNetB0, ResNet50, and ensemble model.
- ❖Highlighted misclassifications with high confidence.



Grid: True Labels vs. Predictions

SUMMARY AND KEY TAKEAWAYS

- EfficientNetB0 and ResNet50 proved effective for sports image classification.
- Ensemble learning improved generalization, and robustness.
- Future Work:
  - Experiment with different ensemble strategies.
  - Test on larger datasets.

REFERENCES

100 Sports Image Classification. (2023, May 3). Kaggle. https://www.kaggle.com/datasets/gpiosenka/sports-classification/data

Affonso, C., Rossi, A. L. D., Vieira, F. H. A., & de Leon Ferreira, A. C. P. (2017). Deep learning for biological image classification. *Expert systems with applications*, *85*, 114-122.

Hamedghorbani. (2023b, July 17). *100 Sports Classification - EfficientNetB0 | 97.8%*. Kaggle. https://www.kaggle.com/code/hamedghorbani/100-sports-classification-efficientnetb0-97-8

Ishaparanjpe. (2023, April 19). *sports classification RESNET transfer learning*. Kaggle. https://www.kaggle.com/code/ishaparanjpe/sports-classification-resnet-transfer-learning/notebook

Liu, X. (2024). Comparison of Four Convolutional Neural Network-Based Algorithms for Sports Image Classification. In *Advances in intelligent systems research/Advances in Intelligent Systems Research* (pp. 178–186). https://doi.org/10.2991/978-94-6463-370-2_20

Podgorelec, V., Pečnik, Š., & Vrbančič, G. (2020). Classification of Similar Sports Images Using Convolutional Neural Network with Hyper-Parameter Optimization. *Applied Sciences*, *10*(23), 8494. https://doi.org/10.3390/app10238494