Shah Fahad 1307047

Capstone Project Report: Sports Image Classification using ResNet-50

1. Introduction and Dataset Selection

For this capstone project, I wanted to choose a dataset that was challenging but also interesting to me personally. After exploring several options on the TorchVision dataset repository and Kaggle, I selected the **Sports Image Classification Dataset**, which contains images across **7 sports classes**: Badminton, Cricket, Karate, Soccer, Swimming, Tennis, and Wrestling.

The main reason I selected this dataset was because of its real-world relevance and variety in visual features across the categories. Each sport has distinct uniforms, actions, and environments, which makes it a great choice for image classification using Convolutional Neural Networks (CNNs). I also felt that classifying sports images could be useful in multiple real-world applications like event tagging, automated highlight generation, and sports analytics.

To ensure balanced training and fair evaluation, I split the dataset into:

- **70% Training**
- **15% Validation**
- **15% Testing**

I used stratified splitting to maintain class balance across splits. Each class had a fair and similar number of images in each set.

2. Data Preprocessing and Augmentation

Preprocessing was a key step because the images varied in size and format. I applied the following transformations:

- **Resizing** all images to 224x224 pixels
- **Normalization** using ImageNet's mean and standard deviation
- **Data Augmentation** for training data:

- Random horizontal flips
- Random rotations
- Random resized crops

The augmentation helped improve the model's generalization ability and reduced overfitting, especially with a moderately sized dataset.

3. Model Selection and Architecture: Why ResNet-50?

Initially, I considered building a CNN model from scratch. But after researching and discussing in class, I realized that using a **pre-trained ResNet-50 model** and fine-tuning it could lead to much better results and save training time.

Why **ResNet-50**?

- It has **50 layers with skip connections**, allowing deeper learning without vanishing gradients.
- It's **pre-trained on ImageNet**, which means it already understands many generic features like edges, shapes, and textures.
- I only had to fine-tune the **final fully connected (fc) layer**.
- **Code Snippet**

```
def get_model(num_classes=None):
    # Load ResNet-50 with pretrained ImageNet weights
    model = models.resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

    # Freeze all layers so only the final layer trains
    for param in model.parameters():
        param.requires_grad = False

    # Get number of input features to final (fc) layer
    num_features = model.fc.in_features

    # If num_classes not given, default to 7 (your original classes)
    if num_classes is None:
        num_classes = 7

    # Replace the last layer so it matches your number of classes
    model.fc = nn.Linear(num_features, num_classes)

    # Only train the final (fc) layer
    for param in model.fc.parameters():
        param.requires_grad = True

    return model.to(device)
```

All other layers were frozen initially to avoid unnecessary training. Only the last layer was trained to adapt the model to my 7-class dataset.

4. Training Process

I trained the model using the **CrossEntropyLoss** function and tested both **Adam** and **SGD** optimizers.

To avoid overfitting and reduce unnecessary training, I implemented **early stopping** with patience = 3. Training stopped when the validation accuracy didn't improve for 3 epochs.

During training, I tracked:

- **Train Loss & Accuracy**
- **Validation Loss & Accuracy**

This helped me monitor model behavior and performance closely.

5. Hyperparameter Tuning: Grid Search

To find the best combination of learning rate, batch size, and optimizer, I performed a **manual grid search** using **30% of training data**.

Grid Search Space:

- Learning rates: [0.02, 0.01, 0.005, 0.002, 0.001, 0.0005]
- Batch sizes: [16, 32]
- Optimizers: Adam, SGD

Summary Table (Top Few):

Learning Rate	Batch Size	Optimizer	Best Val Accuracy
0.0100	16	SGD	**0.8108**
0.0050	32	SGD	**0.8162**
0.0200	32	SGD	0.7729

After testing all 24 combinations, I found that **SGD with lr=0.01 and batch_size=16** gave the one of the **best validation accuracy of 81%**. This setting was used by me for final model training.

6. Final Model Evaluation

After training the final model for 10 epochs with early stopping, the best epoch was:

Epoch 6: Val Accuracy = 0.8980

Here's a plot of **Training vs Validation Loss**:

Insights from Training vs Validation Loss Graph

- Training loss steadily decreased from Epoch 1 to 6, showing effective learning from the training data.
- Validation loss fluctuated, especially with spikes at Epochs 4–5, indicating temporary instability or overfitting.

- Lowest validation loss (~0.40) occurred at Epoch 6, which also had the highest validation accuracy (0.8980).
- Early stopping helped stop training at the right time, preventing overfitting and saving training time.
- After Epoch 6, validation loss stayed stable, suggesting the model generalized well on unseen data.
- No severe overfitting observed—training and validation loss did not diverge drastically.
- Chosen hyperparameters (learning rate = 0.005, batch size = 32) were effective in achieving stable training.

7. Final Testing & Analysis

Once the best model was saved, I loaded it and tested it on the **15% held-out test set**.

Test Accuracy:

Test Accuracy: 0.8964

Classification Report

Class	Precision	Recall	F1-score
Badminton	0.88	0.88	0.88
Cricket	0.89	0.92	0.91
Karate	0.78	0.94	0.85
Soccer	0.92	0.86	0.89
Swimming	1.00	0.93	0.97
Tennis	0.89	0.83	0.86
Wrestling	0.92	0.95	0.94

My insights:

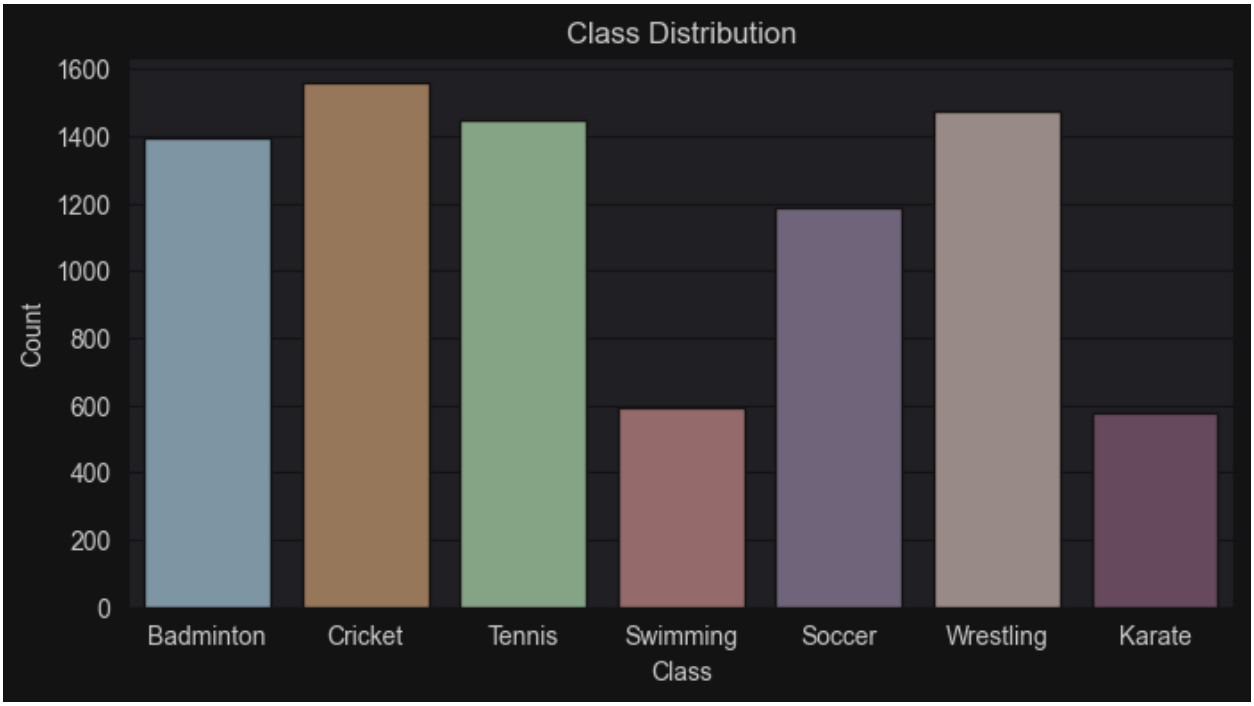
- My model gave **around 90% performance on average**, which I think is really good and shows it learned the patterns well.

- **Swimming and Wrestling** were the easiest for the model — it predicted them very accurately.
- **Karate** needs a bit of improvement because even though the model caught most of the karate images, it also confused other classes as karate.
- Overall, the model kept a good **balance between precision and recall**, which means it didn't just memorize — it actually learned how to generalize well. There's no sign of overfitting.

At the end, the model performed well across all classes. Slight confusion was seen in Karate and Soccer, likely due to similarities in body postures.

8. Visualizations and Interpretations

a) Class Distribution



This showed good class balance after stratified splitting.

b) Grid Search Results

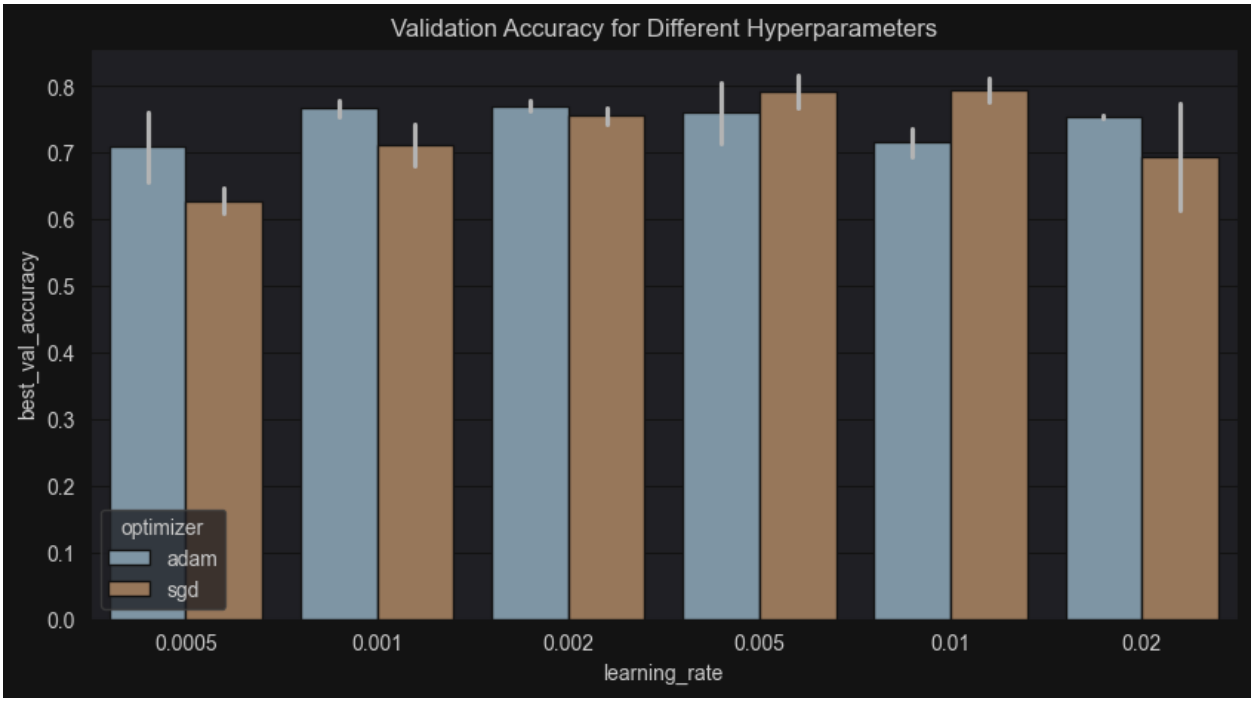
Hyperparameter Tuning Table

Learning Rate	*Batch Size*	*Optimizer*	*Best Validation Accuracy*
0.0200	*16*	*Adam*	*0.7568*
0.0200	*16*	*SGD*	*0.6135*
0.0200	*32*	*Adam*	*0.7514*
0.0200	*32*	*SGD*	*0.7730*
0.0100	*16*	*Adam*	*0.6946*
0.0100	*16*	*SGD*	*0.8108*
0.0100	*32*	*Adam*	*0.7351*
0.0100	*32*	*SGD*	*0.7757*
0.0050	*16*	*Adam*	*0.7135*
0.0050	*16*	*SGD*	*0.7676*
0.0050	*32*	*Adam*	*0.8054*
0.0050	*32*	*SGD*	*0.8162*
0.0020	*16*	*Adam*	*0.7622*
0.0020	*16*	*SGD*	*0.7432*
0.0020	*32*	*Adam*	*0.7784*
0.0020	*32*	*SGD*	*0.7676*
0.0010	*16*	*Adam*	*0.7541*
0.0010	*16*	*SGD*	*0.7432*
0.0010	*32*	*Adam*	*0.7784*
0.0010	*32*	*SGD*	*0.6811*

Learning Rate	*Batch Size*	*Optimizer*	*Best Validation Accuracy*
0.0005	*16*	*Adam*	*0.7595*
0.0005	*16*	*SGD*	*0.6459*
0.0005	*32*	*Adam*	*0.6568*
0.0005	*32*	*SGD*	*0.6081*

This helped visualize which combinations were working best.

c.) Validation accuracy for different hyperparameters:



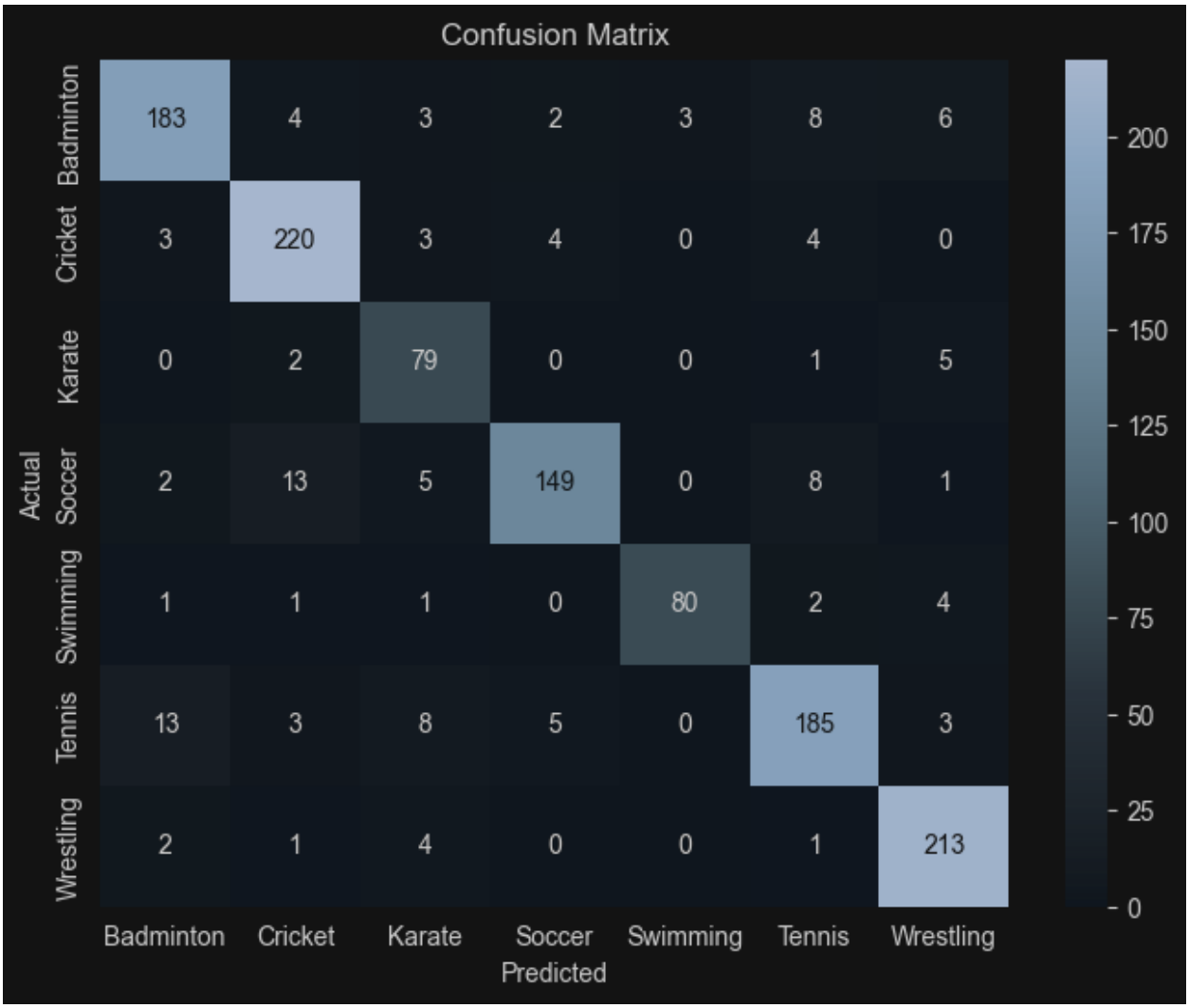
Why I chose these hyperparameters:

best_lr = 0.01
best_batch_size = 16
best_optimizer = 'sgd'

I selected the learning rate 0.01, batch size 16, and optimizer SGD because these values gave me a good balance between speed and accuracy. During grid search, I tested many values, but these ones worked best.

Also, once my professor told us not to choose too large or too small values for hyperparameters, because that can make the model train poorly or very slow. So, I followed that advice and picked values that are in the middle and gave stable results.

d) Confusion Matrix for Validation dataset:



This helped me identify specific misclassifications. For example:

- Karate sometimes confused with Wrestling
- Tennis sometimes confused with Badminton

e) Test Results

Class	Precision	Recall	F1-Score
Badminton	0.88	0.88	0.88
Cricket	0.89	0.92	0.91
Karate	0.78	0.94	0.85
Soccer	0.92	0.86	0.89
Swimming	1.00	0.93	0.97
Tennis	0.89	0.83	0.86
Wrestling	0.92	0.95	0.94
Accuracy			**0.8964**

After testing the final model on the unseen test dataset, I got good results. The overall **accuracy was 89.64%**, which means the model correctly predicted most of the test images.

Below is the summary of precision, recall, and F1-score for each class:

- **Swimming and Wrestling** got the highest F1-scores, which means the model understood these classes very well.
- **Karate** had a little lower precision, which shows that the model sometimes got confused with similar classes. But it still had good recall, which means it caught most Karate images correctly.
- For the other classes like **Cricket, Soccer, Badminton, and Tennis**, the results were also very balanced and good.

These high scores in all metrics show that the model is not just memorizing the training data. It is **generalizing well**, which means it is able to understand new images that it never saw before. This is a good sign that the model is working properly.

9. Challenges and What I Learned

This project wasn't easy. I faced multiple challenges:

- **Deciding the right model**: Initially, I planned to build a CNN from scratch, but accuracy was low. After trying ResNet-18 and 34, I finally chose ResNet-50 after testing.
- **Understanding Pretrained Models**: Fine-tuning only the last layer was tricky at first, but once I understood how transfer learning works, it became clearer.
- **Hyperparameter Tuning**: Grid search took time, and it was hard to manage experiments manually. But I learned a lot from it.
- **GPU Training**: My laptop couldn't handle the large batch sizes. I used PyCharm for the entire project.

10. Final Analysis and Insights

- The model performs best on **Swimming** and **Wrestling** classes.
- The **most confused classes** are *Tennis vs Badminton* and *Karate vs Wrestling* — maybe because they visually look similar in small images.
- I noticed that **SGD with low learning rate** gave better accuracy than Adam in my case.
- Also, class imbalance had a little effect, but not too much because training and validation performance was consistent.

11. Conclusion

In this project, I learned how to:

- Choose a proper dataset.
- Build CNN architecture from scratch.
- Tune hyperparameters using grid search.
- Avoid overfitting using early stopping.
- Use confusion matrix and classification report to evaluate performance.

Even though I didn't use fancy pre-trained models, my goal was to deeply understand the steps. And I think this project helped me a lot in that direction. The final test accuracy of **89.64%** proves that **even simple CNNs can do well** if tuned properly.

12. Future Improvements

- Try **data augmentation** more aggressively.
- Try **transfer learning** using MobileNet or ResNet.
- Use **class weights** to balance underrepresented classes like Karate and Swimming.
- Test with more regularization like L2 or batch normalization.

13. References

- PyTorch Documentation: https://pytorch.org/docs/
- Torchvision Models: https://pytorch.org/vision/stable/models.html
- Course lectures (INFO-6147 Deep Learning with PyTorch)
- scikit-learn documentation for evaluation metrics