EE4745 Final Project: *Defending LSU's Sports AI*

1. Project Background and Motivation

Figure 1: LSU Football Team - LSU Tigers.

Louisiana State University is more than a world-class academic institution; it is a powerhouse of collegiate sports. From the roar of a Saturday night in **Tiger Stadium** to the rhythm of the PMAC basketball court and the artistry of the gymnastics arena, athletics is deeply woven into LSU's culture and identity. As the university continues to modernize its athletic analytics, the LSU Athletics Department is envisioning a new artificial intelligence platform:

> *"What if we could automatically recognize, tag, and organize every image and video captured from our games — from football and basketball to baseball and gymnastics — all using AI?"*

Thus, the idea of the **Sports Image Classification System** was born — a neural network model that can automatically identify the type of sport shown in a photo. This system could help in multiple real-world applications:

- Generating highlight reels automatically from thousands of video frames.
- Tagging and archiving sports media for social media and digital repositories.
- Assisting journalists, scouts, and fans in quickly retrieving sports moments.

Problem A: Sports Image Classification System

Objective: Develop and analyze an image classification model that can accurately distinguish between **ten sports categories** using a small dataset of approximately 1,000 labeled images (around 100 images per class). This task emphasizes practical model design under limited data conditions, interpretability, and reproducible experimentation.

Dataset Description: The *Sports Image Dataset* contains ten distinct classes, each representing a different sport. Each class folder includes about 100 color images collected from public sports media sources. Example samples are shown in Figure 2 Because the dataset is limited in size, you are encouraged to use appropriate augmentation and regularization techniques.

Instructions:

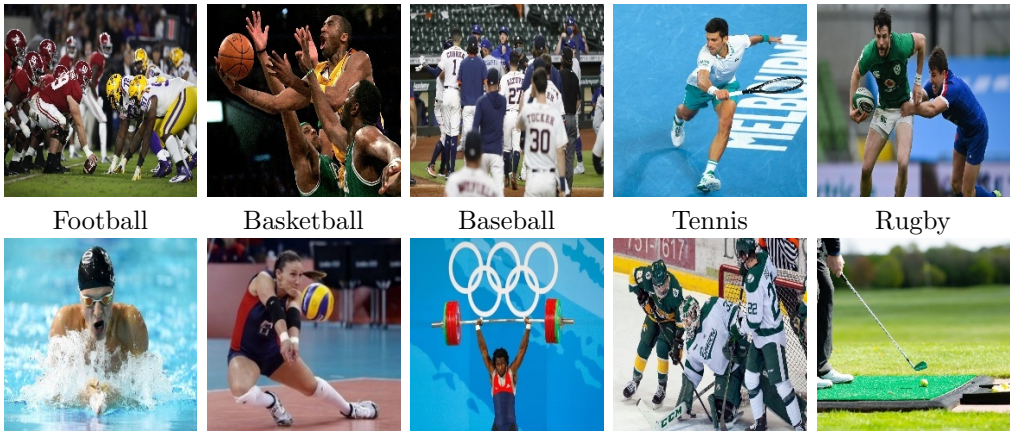

Figure 2: Ten sports classes used in this project. Each image corresponds to one class in the Sports Image Dataset.

1. **Data Preparation**
   - Download the provided dataset package on Moodle, the dataset is already split into train and validation.
   - Resize all images to **32-by-32** or **64-by-64**. Using a smaller resolution will provide or higher training speed.
2. **Model Design and Training**
   - Implement **at least two different architectures**.
   - Ensure both models are **trainable on CPU**.
   - Report your configurations: total number of parameters and training time per epoch, the number of epoch, etc.
3. **Monitoring Training Behavior (You may use TensorBoard)**
   - Log training and validation performance.
   - Use TensorBoard (or matplotlib) to visualize these curves.
   - In your reports:
     - Did the model overfit due to limited data?
     - Which model generalizes better?
4. **Model Evaluation**
   - Save your model checking points 0in the format of **Your-model-name-original.pt** using "torch.save()".
   - Evaluate each trained model on the validation dataset
   - Report:
     - Final **validation accuracy**.
     - Per-class performance.
     - **Example misclassifications**, think about the reason of misprediction.
   - Compare both models in a table including:
     - Accuracy (train/val/test)
     - Parameter count

- Training time
- Qualitative interpretability notes

5. **Model Interpretability**

   - Apply the following techniques to visualize and interpret model decisions:
     (a) **Saliency Map:** pixel-level gradient visualization.
     (b) **Grad-CAM:** feature-level attention heatmap.
   - Select at least **three different classes** and visualize:
     - One correctly classified example per class.
     - One misclassified example (you can find one sample in the training dataset as well).
   - Briefly discuss the results and add a visualization to your report

6. **Reporting Requirements**

   - Each group must submit a concise but complete section for Problem A in the final project report containing:
     - Dataset overview and preprocessing pipeline.
     - Model architecture descriptions and hyperparameters.
     - Training and validation curves (TensorBoard figures).
     - Evaluation metrics and per-class performance.
     - Saliency and Grad-CAM visualizations with discussion.
     - Summary comparing both models and key findings.

Problem B: Accuracy Isn't Enough

But before this system can be deployed, the engineering team faces a challenge that every AI system in the real world must confront: **Is the model truly reliable, explainable, and secure?** A high validation accuracy does not guarantee safety. A model might perform well under normal conditions, yet make catastrophic mistakes when faced with subtle distortions or manipulations.

I want you to act as a passionate basketball enthusiast, **James**, who is working with the media analytics team. You make **THE DECISION** to confuse the system with adversarial samples.

> *"What if I could make every single image — football, baseball, tennis — be classified as **basketball**?"*

It sounds harmless, but if successful, this would cause massive confusion in the automatic highlight generator: The wrong clips would be posted, tags would be incorrect, and sponsors might even misattribute ad impressions.
This scenario mirrors real-world AI security threats — where malicious actors exploit neural networks through **adversarial attacks**. To deploy an AI system safely, we must understand how and why it fails under these attacks, and how to make it more robust and efficient.

Objective: Systematically evaluate the robustness of your best-performing classifier(s) from Problem A by generating and analyzing both **targeted** and **untargeted** adversarial examples. Emphasize careful experimental design, clear visualizations, and insightful interpretation (report quality is the primary grading factor).

Threat Model (default) Assume a **white-box** threat model: the attacker has access to model weights, architecture, and gradients.

Required Elements (summary)

- Implement both **untargeted** and **targeted** attacks using two attack methods.
- Generate adversarial examples for several representative samples (for each type of attack, generate 10 adversarial samples).
- For each adversarial example, provide: original vs adversarial image, predicted labels and logits, perturbation norm, attack success or failure, and interpretability maps (Saliency and Grad-CAM) for both clean and adversarial inputs.
- Analyze transferability: test how adversarial examples crafted on one model affect other models you trained in Problem A.
- Document experimental settings (seeds, exact hyperparameters) so results are reproducible.

Attack Methods

- Untargeted attacks: aim to cause *any* incorrect label .
- Targeted attacks: aim to force a *specific* target label ("basketball").

Transferability test:

- For adversarial examples crafted on Model A, evaluate them on Model B without modification (Two models you built in Probelm A.
- Report cross-model success rates and analyze which attacks transfer better and why.

Optional (5 bonus pts) Defense: Implement a simple defense (e.g., input preprocessing, Gaussian smoothing, or a small adversarial training) and evaluate how it affects success rates and interpretability.

Problem C: Faster Inference (Unstructured Pruning)

Objective: Improve the deployment efficiency of your sports classifier by applying **unstructured weight pruning** while preserving as much accuracy as possible. Assess changes in **model size**, **sparsity**, **inference speed**, and **robustness to adversarial examples** (crafted in Problem B).

Setup & Scope

- Select **one trained model** from Problem A (preferably the better or more interpretable one).
- Apply **unstructured pruning** to convolutional and linear layers.
- Evaluate multiple sparsity levels (e.g., **20%**, **50%**, **80%** of weights pruned).
- Brief **fine-tuning** after pruning to recover accuracy.

Detailed Instructions

1. **Baseline Checkpoint**
   - Save a reference checkpoint of the *unpruned* model and record:
     - Test accuracy (and validation accuracy).
     - Parameter count and on-disk model size (MB).
     - Inference latency (see step 4.).
2. **Apply Unstructured Pruning**

- Use magnitude-based pruning to remove the smallest-magnitude weights in selected layers.
- Recommended sparsity targets: $s \in \{0.2, 0.5, 0.8\}$ (i.e., 20%, 50%, 80% of weights set to zero).
- Maintain identical evaluation protocol across sparsity levels for fair comparison.
- *Implementation hint (PyTorch)*:

```
# Example (per layer) using L1 magnitude:
import torch.nn.utils.prune as prune
prune.l1_unstructured(module, name='weight', amount=0.5)  # 50% prune
# To make pruning permanent (remove buffers, keep zeros in tensor):
prune.remove(module, 'weight')
```

3. **Post-Pruning Evaluation**
   - Save your model checking points 0in the format of **Your-model-name-pruned-ratio.pt** using "torch.save()".
   - Recalculate:
     - **Sparsity**: sparsity = $\frac{\#\{w=0\}}{\#\text{total weights}}$.
     - **Parameter count** and **file size** (MB).
     - **Accuracy**: test accuracy (and validation accuracy).
   - **Fine-tuning**: run a few epochs (e.g., 3–10) with a smaller learning rate to recover performance. Report "before vs. after fine-tuning" performance for each sparsity level.
4. **Measure Inference Latency**
   - Report latency on the same hardware used in Problem A.
   - Use batch size = 1 (and optionally = 16) and average over ≥ 100 runs.
   - Discard the first 10 "warm-up" runs; then average the remainder.
   - If using GPU, synchronize between timings; if CPU-only, use `time.perf_counter()`.
   - Report: mean latency (ms) and standard deviation for each sparsity level.
5. **Robustness Check with Adversarial Examples**
   - Reuse a subset of adversarial examples from Problem B (both targeted and untargeted).
   - Evaluate whether adversarial examples *still fool* the pruned model.
   - Report changes in success rate.

What to Report

A. Summary Table (per sparsity level)

- % Pruned (sparsity), Accuracy (pre-/post-finetune), #Params, Model Size (MB), Latency (ms).

B. Plots

- **Accuracy vs. Sparsity** curve (with/without fine-tuning).
- **Latency vs. Sparsity** (and/or Model Size vs. Sparsity).

C. Robustness Table

- For a fixed set of adversarial samples from Problem B, report success rates on:
  (a) Original (unpruned) model,
  (b) Pruned @ 20%,
  (c) Pruned @ 50%,
  (d) Pruned @ 80%.
- Separate targeted vs. untargeted outcomes; include mean perturbation norms and average confidence of incorrect predictions.

D. Short Discussion

- Trade-offs among accuracy, sparsity, size, and speed.
- Which layers were most sensitive to pruning?
- Did pruning make attacks easier or harder (transferability, needed ε, iterations)? Why?
- Deployment takeaway: a recommended sparsity point for your model.

E. (Optional) (5 Bonus Points) Additional pruning method. Do a literature review, other than the unstructured pruning we have introduced in the class, and implement another pruning method. For instance, structured pruning or adversarial pruning.

Project Deliverables

Team Policy: Each project may be completed individually or in teams of **at most two students**. Both teammates must contribute equally, and the final report must clearly state each member's contribution. Duplicate submissions or identical write-ups across different teams are not permitted.

The final submission must include all mandatory parts (**Problem A–C**) and may include the optional bonus task for extra credit. A concise summary of deliverables is shown in Table 1.

Part	Title	Main Deliverables and Requirements	Submission Format
A	**Sports Image Classification System**	• Train at least two CPU-trainable models (e.g., CNN, VGG, ResNet). • Monitor loss/accuracy with TensorBoard. • Evaluate on 10-class dataset (100 images/class). • Visualize and discuss interpretability (Saliency Map & Grad-CAM). • Compare architectures and report performance summary.	• trained model's checking points. Clearly state how to load it. • Report section with results and analysis • Source code (.py/.ipynb) • TensorBoard or matplotlib figures
B	**Adversarial Attack & Analysis**	• Implement both **targeted** and **untargeted** attacks (two algorithms). • Generate at least 10 adversarial examples per type. • Compare original vs. adversarial predictions & interpretability maps. • Test transferability across models. • Discuss results; focus on analysis rather than raw success rate.	• Report section with tables & visualizations • Attack scripts/notebooks • Figures showing Saliency & Grad-CAM before/after attack
C	**Faster Inference (Unstructured Pruning)**	• Apply unstructured pruning (20%, 50%, 80% sparsity). • Measure model size, sparsity, accuracy, and inference speed. • Evaluate adversarial robustness of pruned models. • Plot accuracy vs. sparsity and latency vs. sparsity curves. • Provide short discussion on trade-offs.	• trained model's checking points. Clearly state how to load it. • Report section with summary tables & plots • Code for pruning and evaluation • Timing logs or measurement scripts
(Opt)	**Bonus: AE defense/Structured Pruning (5 pts)**	• Implement an additional pruning technique (structured, adversarial, or literature-based). • Compare with unstructured pruning baseline. • Discuss performance and efficiency differences.	• trained model's checking points. Clearly state how to load it. • Report subsection with explanation & comparison • Supporting code

Table 1: Summary of deliverables for each project component. Each team (≤ 2 students) must submit a single joint report and corresponding source code files.