Pattern Recognition and Machine Learning
Major Project
"Football Club Logo Recognizer"

Team Members:
Ashudeep Dubey (B21EE090)
Tanish Pagaria (B21AI040)
Vinay Vaishnav (B21EE084)

Problem Statement:
To classify the images of the club logos of the teams in the top 5 football leagues.

Kaggle Dataset Link:
https://www.kaggle.com/datasets/alexteboul/top-5-football-leagues-club-logos

Drive link to the files used in our project:
https://drive.google.com/drive/folders/1yLwtZHtW8RIrDzQjHPvNgoedEtZYGMwe?usp=share_link

Libraries used:
`os` (for file-handling)
`NumPy`, `Pandas`, `Matplotlib`, `Seaborn` (for data-handling, mathematical operations, and visualization)
- `cv2` (OpenCV-Python for image operations)
- `Pillow` (image file handling)
- `rembg` (for background-removal)

`Tensorflow`, `Keras` (for Convolutional Neural Network and Image Augmentation)
`PyTorch` (for Artificial Neural Network (MLP))
`Scikit-Learn` (for classification models and other machine learning operations)
`streamlit` (for model-deployment)

Dataset Generation:
The "top-5-football-leagues" dataset contained images of 100 clubs (20 clubs each in 5 different leagues). Hence, we had only 100 different images in the dataset.
We had to create training, validation, and test datasets using these logo images.
For this purpose, we used the image augmentation technique.
The `ImageDataGenerator` module from `Keras` was used to perform the image augmentations. We applied augmentations, including random rotations, width shift, height shift, rescaling, shear range, random zoom, etc., to the images and generated a new dataset.
For this project, we chose a size of 64 x 64 pixels for the generated images. We generated 75 augmented images for each team.

Example images from the generated dataset:-

| Original Image	Images generated (after Image Augmentation)
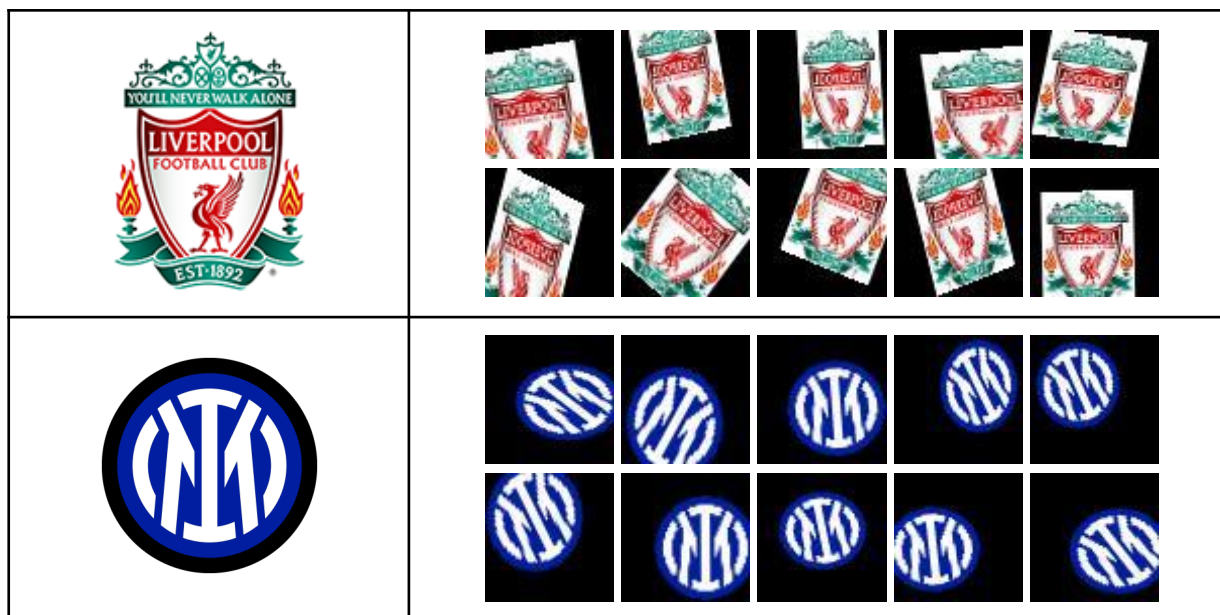
Loading the generated dataset:

The images after augmentation were automatically stored in custom created folders, and the paths to these images were added to a Pandas dataframe, which was also stored for accessing the images later. The team-names (class labels) were categorically-encoded.
The image-handling modules Pillow and OpenCV were used for loading the images in the required format (RGB or grayscale).

Creating data loaders for the dataset:

We split the dataset into training-testing-validation sets in the ratio 70:10:20 (stratified splits were performed to ensure that our models are trained on a balanced and representative sample of the original dataset).
The data loaders were then created for all the splits. We performed this operation to create data loaders for both RGB and grayscale images.
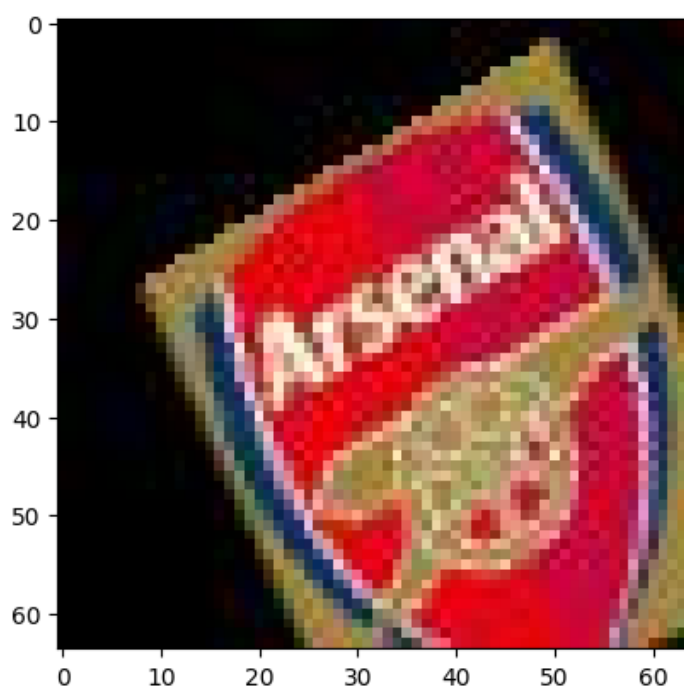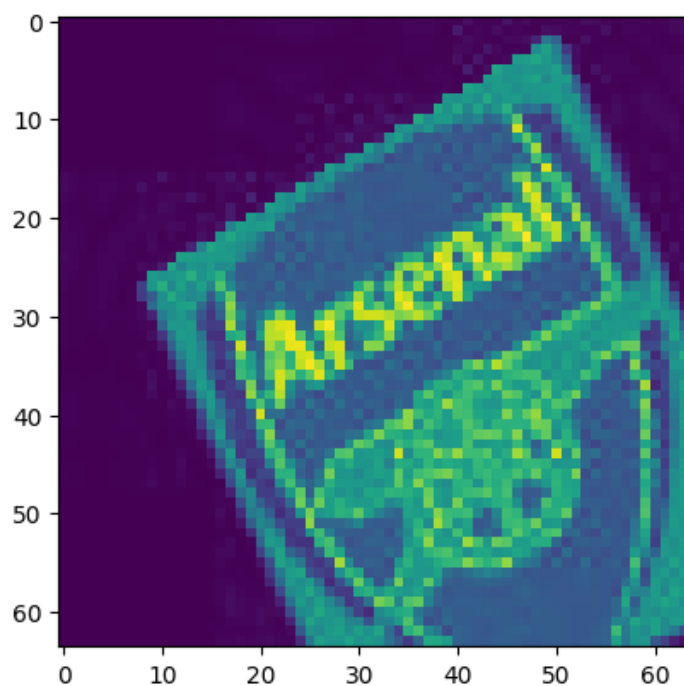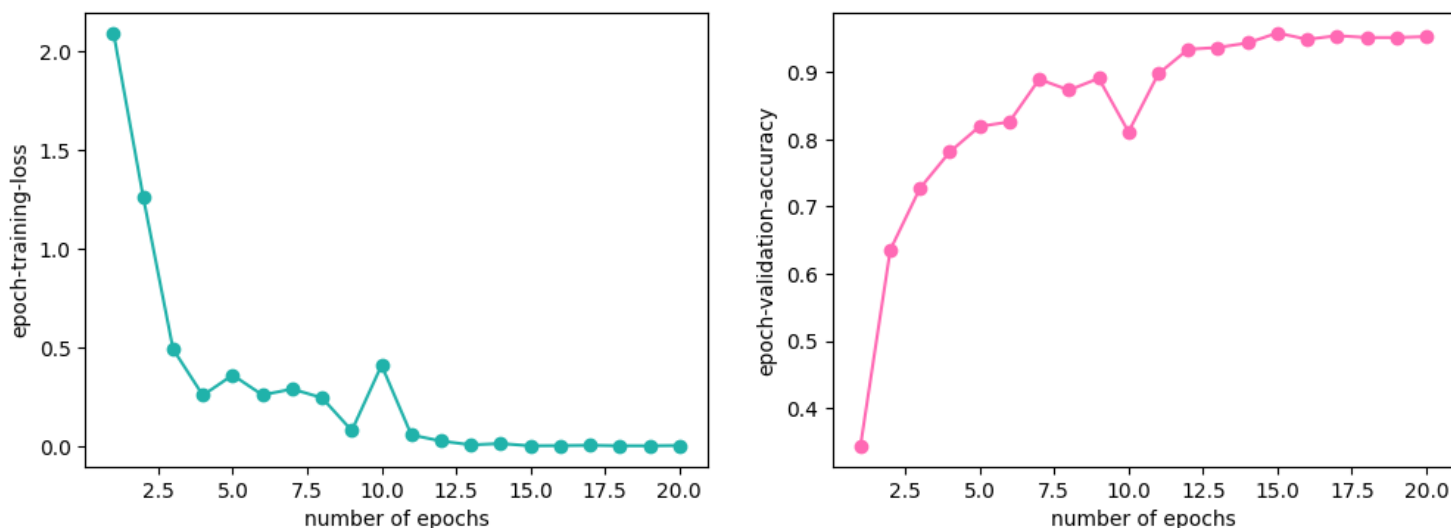

RGB image   *Grayscale image*

Classification Techniques Used:

Artificial Neural Network (4-layered Multi-Layer Perceptron in our case)

It was implemented using PyTorch. We defined a 4-layered MLP, which would take the number of nodes in each of the layers as the argument. The optimizer used was Adam, and the loss used was Cross Entropy Loss. The model had the provision of saving the best model at the end of all the epochs, which would be used in the testing.

We checked several permutations for the number of hidden nodes and the learning rate.

Epoch-curves for ANN (RGB images)
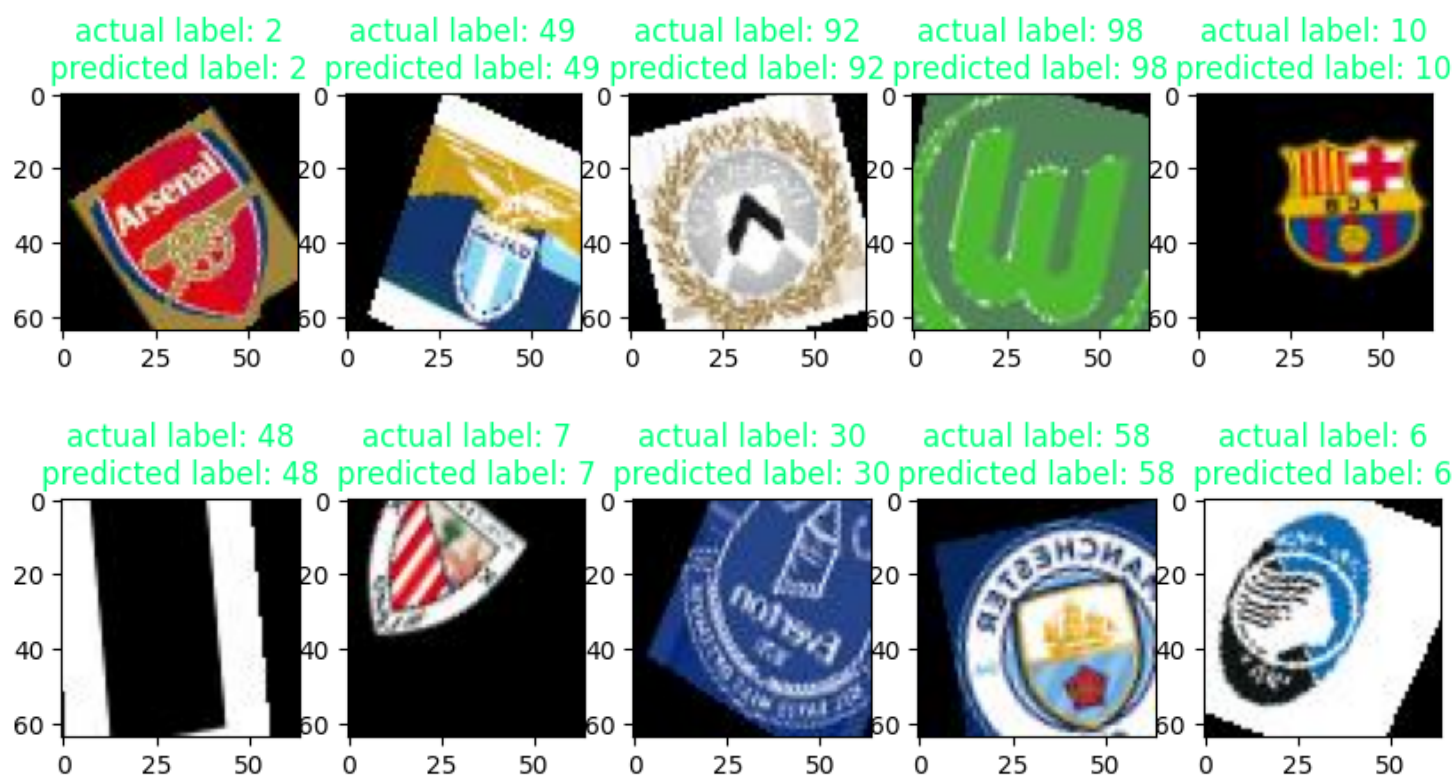
Some of the correctly classified logos

Some of the incorrectly classified logos

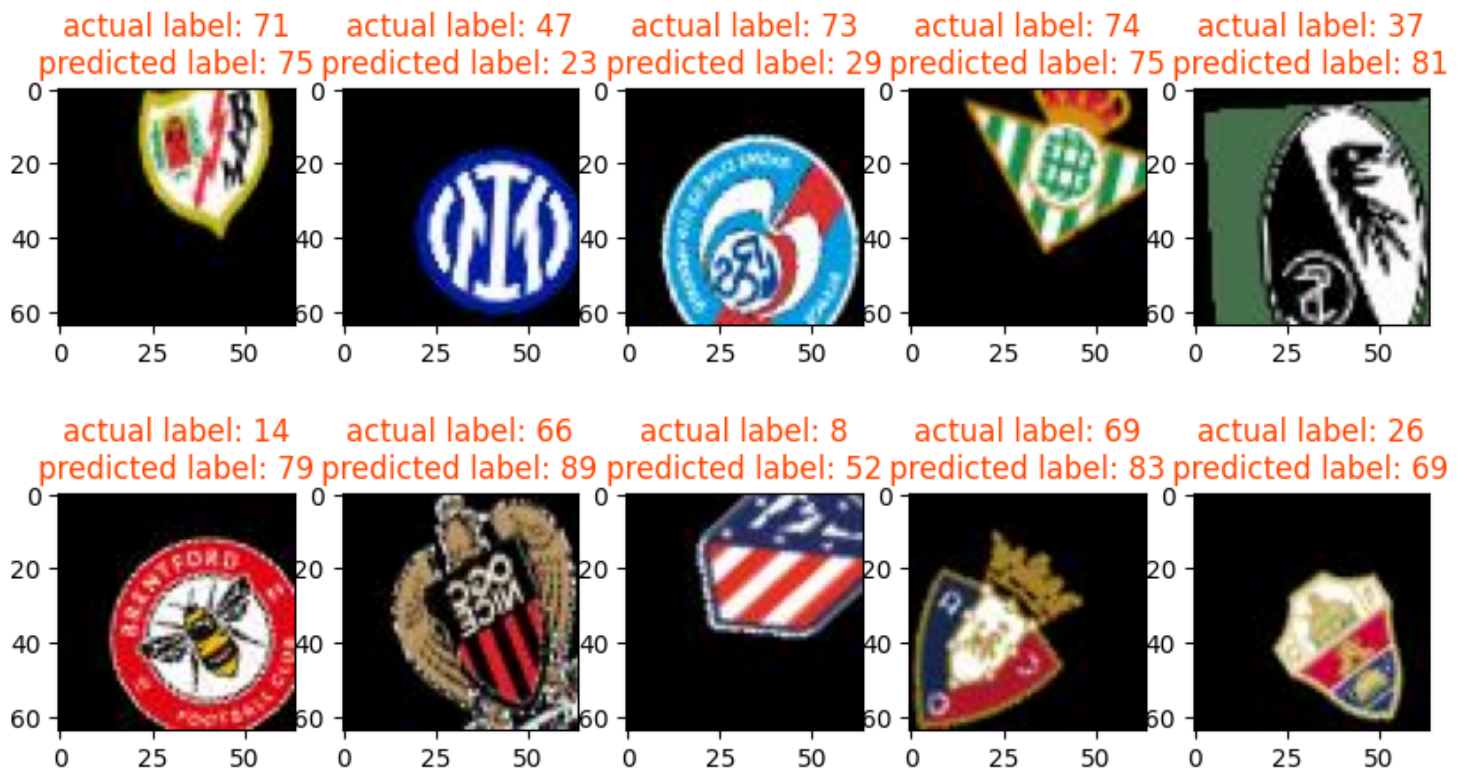| Test accuracy for RGB images	Test accuracy for grayscale data
94.74%	40.64%

Convolutional Neural Network (CNN)

CNN, or Convolutional Neural Network, is a type of neural network that is particularly well-suited for image classification tasks. It works by breaking down the image into smaller pieces called "features" and analyzing them to identify patterns and relationships that can help identify what the image represents.

It was implemented using `tensorflow`.

Basic working of CNN:-

A CNN starts with a series of convolutional layers that extract low-level features from the input image. Each convolutional layer applies a set of filters to the input image, producing a set of output feature maps. To reduce the spatial dimensions of the output feature maps and make the model more computationally efficient, a pooling layer is often applied after each convolutional layer. The output of the last pooling layer is then flattened into a 1-dimensional feature vector, and passed through one or more fully connected layers. These layers perform a series of non-linear transformations on the feature vector, and produce a final output that corresponds to the predicted class probabilities.

Test accuracy for RGB images	Test accuracy for grayscale data
98.07%	87.7%

Random Forest Classifier

We decided to use Random Forest Classifier for the problem because it can handle a wide range of visual features, and can also handle noisy or missing data. It is also fast and efficient, making it a practical choice for large datasets.

We performed the grid-search algorithm for finding the optimal hyperparameters for the classifier model.

Best hyperparameters: **{ max_depth: 20, n_estimators: 200 }**

Test accuracy for RGB data	Test accuracy for grayscale data
80.57%	**67.79%**

KNearestNeighbors (KNN) Classifier

KNN is a simple and intuitive algorithm used in image classification tasks because it compares the image that needs to be classified with other images that are already labeled and known.

We performed the grid-search algorithm for finding the optimal hyperparameters for the classifier model.

Best Hyperparameters: **{ n_neighbors=3, weights="distance", p=1 }**

Test accuracy for RGB images: **72.25%**

Support Vector Machine (SVM) Classifier

SVM is a popular machine learning algorithm used for image classification due to its ability to handle high-dimensional data, non-linear decision boundaries, imbalanced data, and a strong theoretical foundation. Its effectiveness in finding the best decision boundary between different classes of images makes it a reliable and robust algorithm for image classification tasks.

We performed the grid-search algorithm for finding the optimal hyperparameters for the classifier model.

Best Hyperparameters: **{ kernel= "rbf", gamma= "auto",   C=10 }**

Test accuracy for RGB data	Test accuracy for grayscale data
90.2%	**63.33%**

Other models (failed attempt)

KMeans Algorithm

We tried to use K–means but the dataset we had generated contained rotations; hence, we got cluster centroids that were very uncertain, and that led to poor performance of our K-means model.
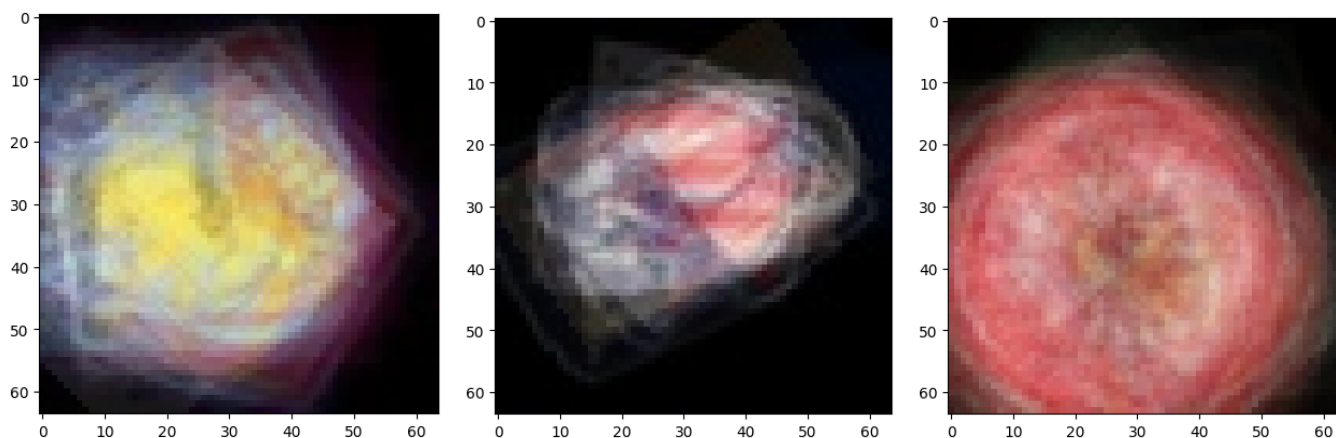
Centroids generated

Saving the "Best" model:

On the basis of our test accuracy scores, we found the **CNN Classifier Model** to be our best model, which gave **98.07%** accuracy on our generated dataset.

Final pipeline implementation:

A pipeline was implemented to chain together the multiple image preprocessing and machine learning steps into a single object, making it easier to build and deploy the final complex machine learning model.

The pipeline object would take in the image as input from the user. The background of the image would get removed in the backend and replaced with black. The changed images would get resized to the size at which our model was trained (64x64 pixels in this case).

The image would then get converted to a tensor and passed to our final classifier model, and the resulting prediction would get returned as the output.

Model deployment:

The best model was saved, and the pipeline was fed to a `streamlit` web application.

The title of the webpage was set, and a facility to upload the test image was given to the user.

On clicking the detection button, the model would then evaluate the image and print the best matched logo accordingly.

Testing our pipeline on logo-images not from the dataset:

We randomly downloaded team logo images from the internet, cropped them a bit, and tested them with our final pipeline.

The following results were obtained:-

```
Entered image: test-bayern-2.jpg
1/1 [==============================] - 0s 20ms/step
Predicted team: losc-lille

Entered image: test-chelsea-1.png
1/1 [==============================] - 0s 17ms/step
Predicted team: manchester-city

Entered image: test-clermont-foot-1.jpg
1/1 [==============================] - 0s 18ms/step
Predicted team: clermont-foot-63

Entered image: test-inter-1.png
1/1 [==============================] - 0s 18ms/step
Predicted team: inter

Entered image: test-liverpool-1.png
1/1 [==============================] - 0s 17ms/step
Predicted team: liverpool

Entered image: test-liverpool-2.jpg
1/1 [==============================] - 0s 19ms/step
Predicted team: athletic

Entered image: test-manchester-united-1.png
1/1 [==============================] - 0s 17ms/step
Predicted team: manchester-united

Entered image: test-newcastle-1.png
1/1 [==============================] - 0s 17ms/step
Predicted team: newcastle-united
```

```
Entered image: test-angers-1.png
1/1 [==============================] - 0s 74ms/step
Predicted team: angers-sco

Entered image: test-arsenal-1.png
1/1 [==============================] - 0s 24ms/step
Predicted team: arsenal

Entered image: test-atletico-madrid-2.png
1/1 [==============================] - 0s 19ms/step
Predicted team: athletic

Entered image: test-barcelona-1.png
1/1 [==============================] - 0s 17ms/step
Predicted team: barcelona

Entered image: test-barcelona-2.jpg
1/1 [==============================] - 0s 19ms/step
Predicted team: barcelona

Entered image: test-barcelona-3.jpeg
1/1 [==============================] - 0s 27ms/step
Predicted team: barcelona

Entered image: test-bayern-1.png
1/1 [==============================] - 0s 17ms/step
Predicted team: bayern

Entered image: test-bayern-2.jpg
1/1 [==============================] - 0s 20ms/step
Predicted team: losc-lille
```

We observe that our model performed fairly well for the test-images (11/16).

Check out our model deployment:

https://ashddftw-logo-detection-logo-detect-web-0s4lfu.streamlit.app/

Github Repo link:

https://github.com/yorozuya-2003/Football-Club-Logo-Recognizer

Contributions:

→ Ashudeep Dubey (B21EE090)
- Proposed idea for image augmentation for dataset generation.
- Model deployment on web-server.
- Implemented SVM and KMeans models with hyperparameter tuning and used them for RGB and grayscale images.

→ Tanish Pagaria (B21AI040)
- Researched about and performed Image Augmentation on the images.
- Created workflow for saving the augmented images after the generation in respective directories based on the corresponding team and league.
- Implemented ANN using PyTorch and also performed visualizations.
- Implemented the final pipeline.

→ Vinay Vaishnav (B21EE084)
- Generated grayscale data from the RGB generated images.
- Implemented KNN and Random Forest classifier models, performed hyperparameter tuning with gridsearch, and used them for both RGB and grayscale images.
- Implemented CNN from Tensorflow and trained it for both RGB and grayscale data.