- **Links:**
    - **Github:** https://github.com/M1ion/AdvProg.git
    - **Youtube:** https://youtu.be/DjGHxV9PZ7c
    - **Database:** https://www.kaggle.com/datasets/gpiosenka/balls-image-classification
- **Introduction**
    - **Problem**

      In this report, we address the problem of classifying images of different types of balls in a dataset of 30 classes. Sports ball identification and classification is a common task in the sports industry, but it can be challenging due to the variety of shapes, sizes, and colors of different types of balls. Automating this task can improve efficiency and accuracy in sports equipment inventory management, coaching, and training. In this work, we apply deep learning to solve the problem, using the VGG16 model and data augmentation.
    - **Literature Review**

      There are many existing solutions for image classification, such as Convolutional Neural Networks (CNNs) and transfer learning. CNNs are a type of deep neural network that have been shown to be effective in image classification tasks. Transfer learning is a technique where a pre-trained model is used as a starting point for training a new model on a different task. The pre-trained model can be fine-tuned to the new task by adding new layers on top of it.

      One of the most popular pre-trained models for image classification is VGG16, which is a CNN with 16 layers developed by the Visual Geometry Group at Oxford University. It has achieved state-of-the-art performance on various image recognition challenges:
        - Theoretical part: https://towardsdatascience.com/a-demonstration-of-transf

er-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a

- Practical part: https://keras.io/api/applications/vgg/

Also, Sweta Shaw developed a deep learning-based solution for ball classification on the same dataset, as used in this study. In her work, titled "Ball classification from scratch - 95% acc," she achieved an accuracy of 95% using MobileNetV2 (source: https://www.kaggle.com/code/swetash/ball-classification-from-scratch-95-acc). MobileNetV2 is a neural network architecture that is optimized for mobile devices, with a small memory footprint and fast computation times. The model was trained using a batch size of 20 for 30 epochs, with the categorical cross-entropy loss function and the RMSprop optimizer. Her model was also able to identify the classes with high accuracy, including soccer ball, tennis ball, and volleyball. Overall, her work demonstrated the effectiveness of deep learning models for ball classification and the importance of data preprocessing and model optimization for achieving high accuracy.

➢ Current Work

In this work, I use the VGG16 model as a starting point and add new layers on top of it to perform classification on the dataset. I also use data augmentation to increase the size of the training set and improve the generalization of the model. The model trained using batch size of 16, image size 224x224 for 10 epochs, with the categorical cross-entropy loss function and the Adam optimizer.

❖ Data and Methods

➢ Data Analysis

Dataset obtained from kaggle: https://www.kaggle.com/datasets/gpiosenka/balls-image-classification. It consists of 30 classes of images with varying sizes and aspect ratios. However, the training set of the dataset is unbalanced, with some classes having much fewer samples than others. As the author of the dataset said: “This was done intentionally so notebook developers could test methods to deal with unbalanced data.”

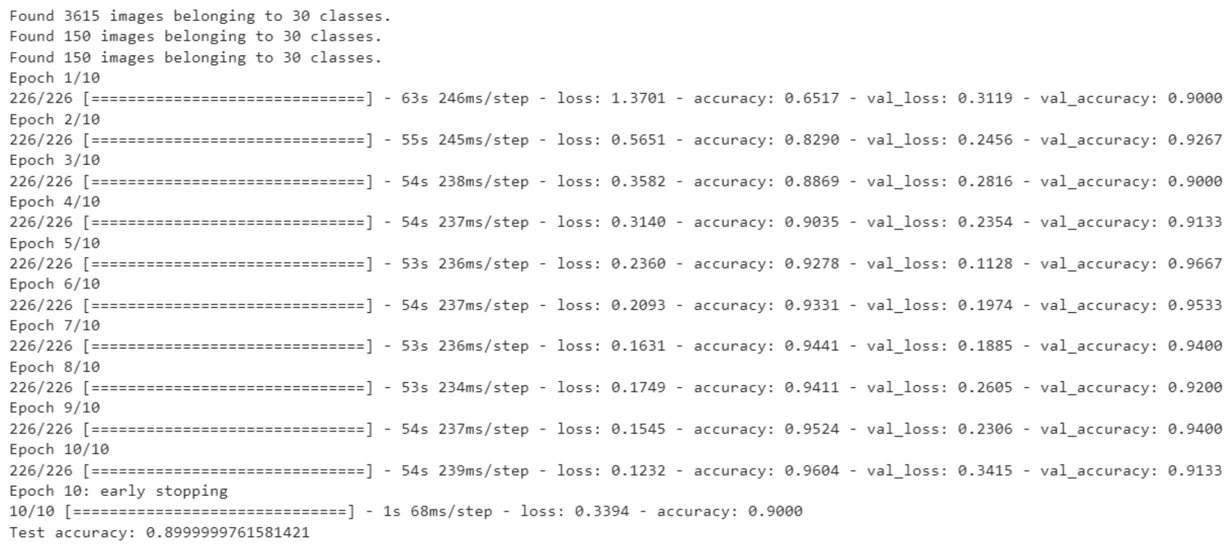
- ➢ **Model Description**

  The VGG16 model was used as a starting point and added our own fully connected layers on top of it. The VGG16 model has 16 layers, including 13 convolutional layers and 3 fully connected layers. I froze the pre-trained layers of the VGG16 model during the initial training to prevent overfitting and speed up the training process. In addition, I used the categorical cross-entropy loss and the Adam optimizer with a learning rate of 0.001. Model was trained for 10 epochs with early stopping if the validation loss did not improve for 5 consecutive epochs.

  Also data augmentation was used to increase the size of the training set and improve the generalization of the model. I applied random transformations such as rotation, shear, zoom, and horizontal flipping to the images during training.

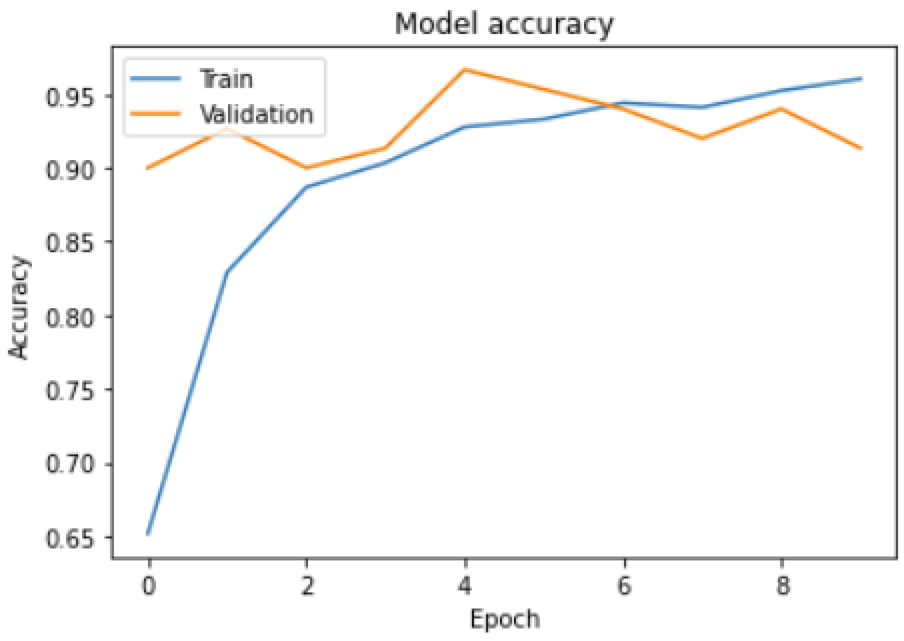
❖ Results

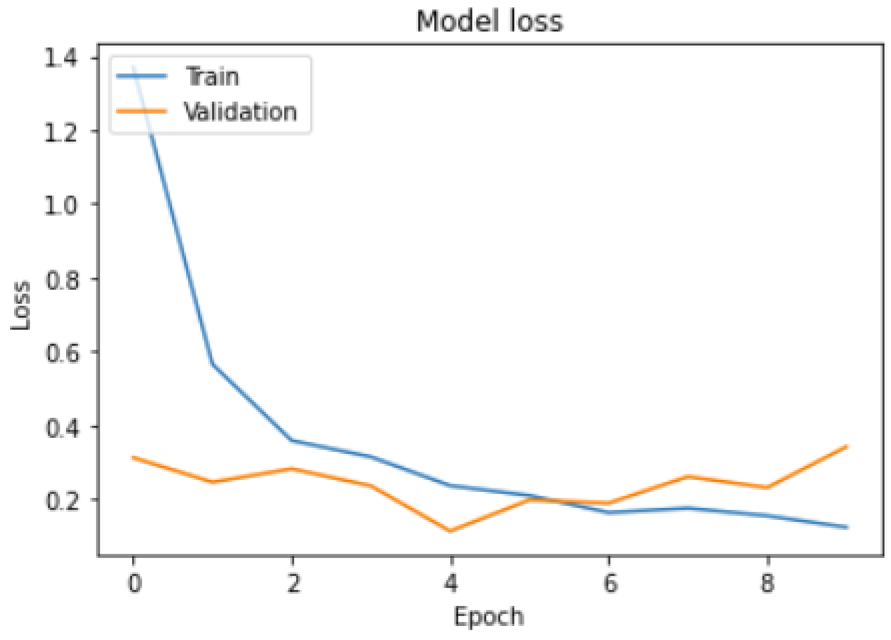
After training the model for 10 epochs, we achieved accuracy of 96.04%, validation accuracy of 91.33% and a test accuracy of 90%, indicating that the model was able to generalize well to new, unseen images:



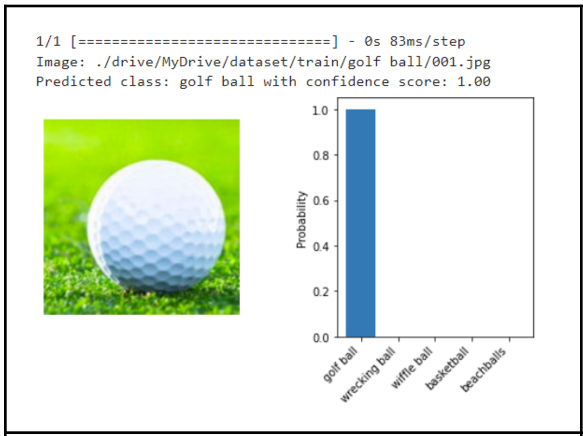
```
Found 3615 images belonging to 30 classes.
Found 150 images belonging to 30 classes.
Found 150 images belonging to 30 classes.
Epoch 1/10
226/226 [==============================] - 63s 246ms/step - loss: 1.3701 - accuracy: 0.6517 - val_loss: 0.3119 - val_accuracy: 0.9000
Epoch 2/10
226/226 [==============================] - 55s 245ms/step - loss: 0.5651 - accuracy: 0.8290 - val_loss: 0.2456 - val_accuracy: 0.9267
Epoch 3/10
226/226 [==============================] - 54s 238ms/step - loss: 0.3582 - accuracy: 0.8869 - val_loss: 0.2816 - val_accuracy: 0.9000
Epoch 4/10
226/226 [==============================] - 54s 237ms/step - loss: 0.3140 - accuracy: 0.9035 - val_loss: 0.2354 - val_accuracy: 0.9133
Epoch 5/10
226/226 [==============================] - 53s 236ms/step - loss: 0.2360 - accuracy: 0.9278 - val_loss: 0.1128 - val_accuracy: 0.9667
Epoch 6/10
226/226 [==============================] - 54s 237ms/step - loss: 0.2093 - accuracy: 0.9331 - val_loss: 0.1974 - val_accuracy: 0.9533
Epoch 7/10
226/226 [==============================] - 53s 236ms/step - loss: 0.1631 - accuracy: 0.9441 - val_loss: 0.1885 - val_accuracy: 0.9400
Epoch 8/10
226/226 [==============================] - 53s 234ms/step - loss: 0.1749 - accuracy: 0.9411 - val_loss: 0.2605 - val_accuracy: 0.9200
Epoch 9/10
226/226 [==============================] - 54s 237ms/step - loss: 0.1545 - accuracy: 0.9524 - val_loss: 0.2306 - val_accuracy: 0.9400
Epoch 10/10
226/226 [==============================] - 54s 239ms/step - loss: 0.1232 - accuracy: 0.9604 - val_loss: 0.3415 - val_accuracy: 0.9133
Epoch 10: early stopping
10/10 [==============================] - 1s 68ms/step - loss: 0.3394 - accuracy: 0.9000
Test accuracy: 0.8999999761581421
```

Plot of the training and validation accuracy and loss:

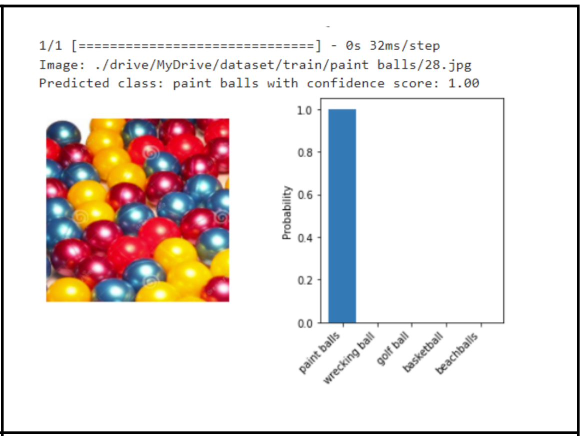




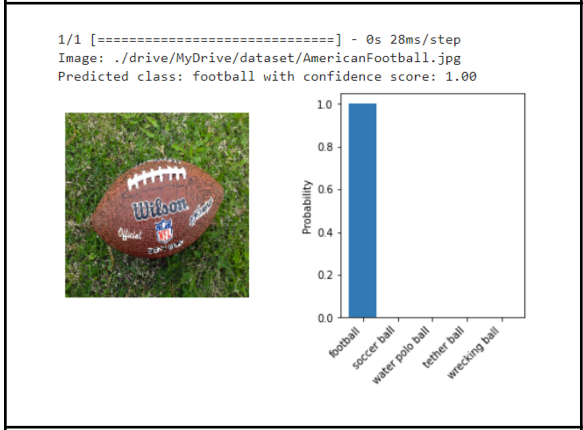
Testing the model on a few individual images, and the model was able to correctly classify the images with the confidence score of 1.00:



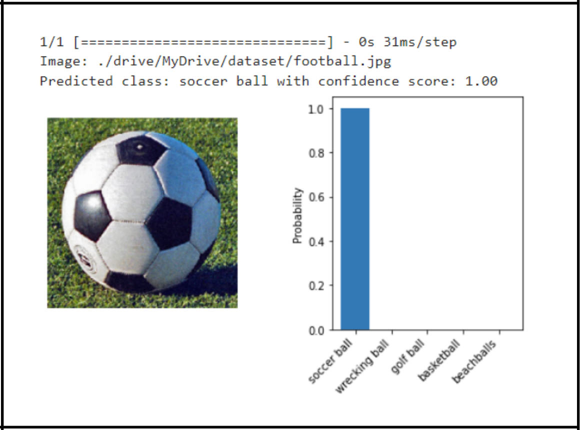
```
1/1 [==============================] - 0s 83ms/step
Image: ./drive/MyDrive/dataset/train/golf ball/001.jpg
Predicted class: golf ball with confidence score: 1.00
```



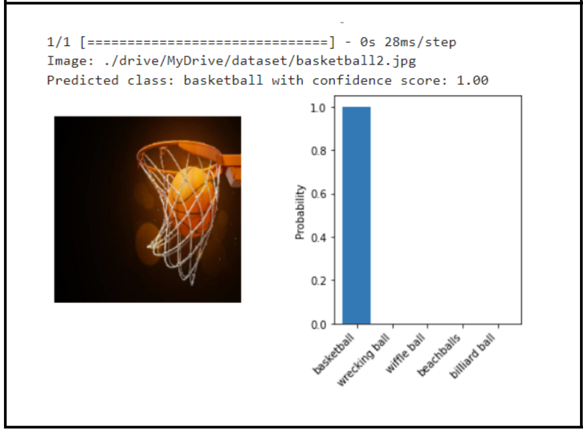
```
1/1 [==============================] - 0s 32ms/step
Image: ./drive/MyDrive/dataset/train/paint balls/28.jpg
Predicted class: paint balls with confidence score: 1.00
```



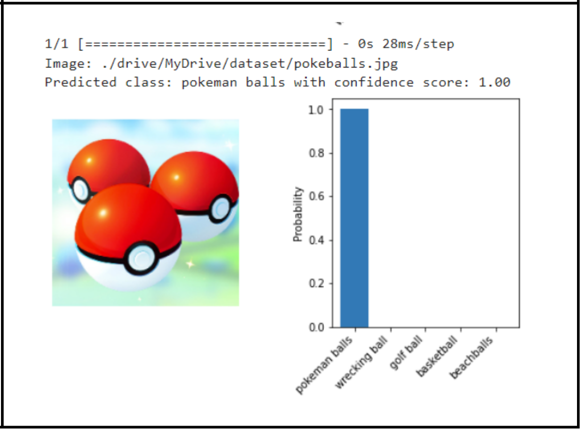
```
1/1 [==============================] - 0s 28ms/step
Image: ./drive/MyDrive/dataset/AmericanFootball.jpg
Predicted class: football with confidence score: 1.00
```



```
1/1 [==============================] - 0s 31ms/step
Image: ./drive/MyDrive/dataset/football.jpg
Predicted class: soccer ball with confidence score: 1.00
```



```
1/1 [==============================] - 0s 28ms/step
Image: ./drive/MyDrive/dataset/basketball2.jpg
Predicted class: basketball with confidence score: 1.00
```

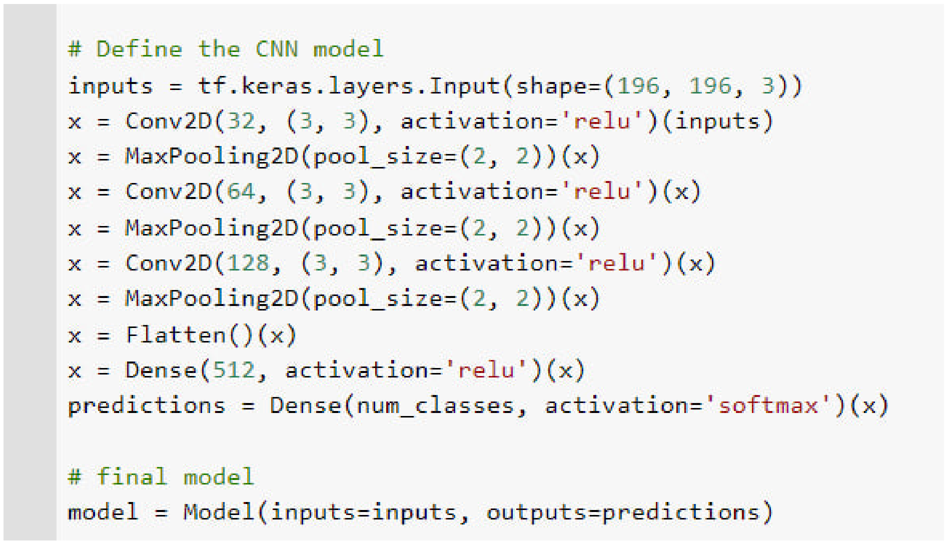


```
1/1 [==============================] - 0s 28ms/step
Image: ./drive/MyDrive/dataset/pokeballs.jpg
Predicted class: pokeman balls with confidence score: 1.00
```

❖ Discussion

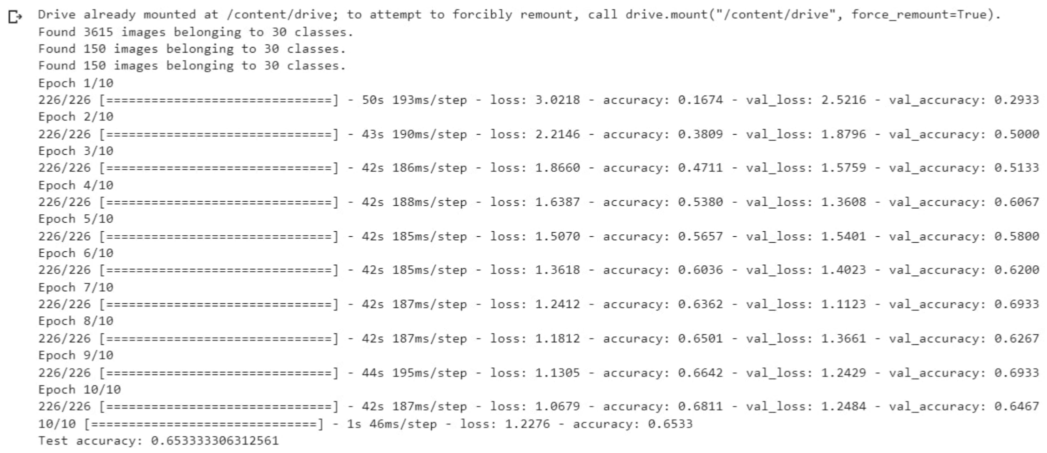
Results show that the VGG16 model with added layers and data augmentation is effective in classifying images of objects in this particular dataset.

The VGG16 model performed much better than a simple CNN model:



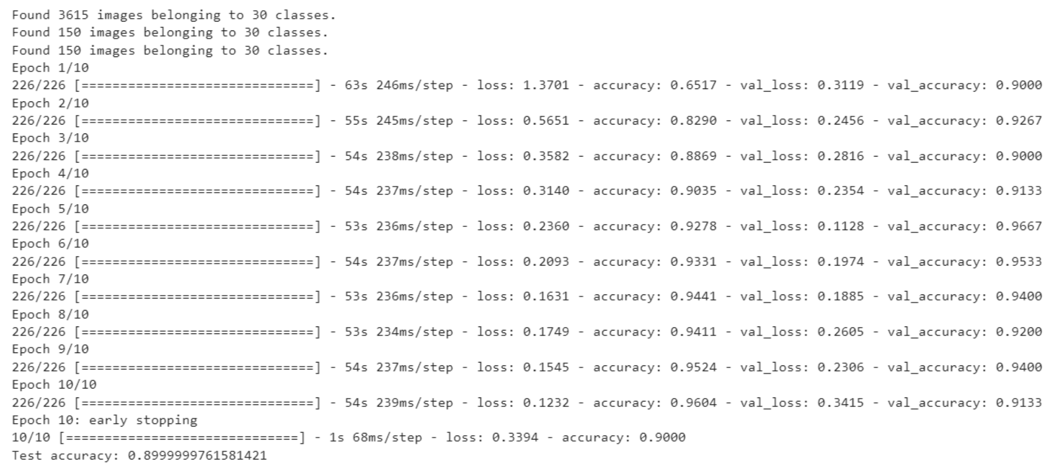
```
# Define the CNN model
inputs = tf.keras.layers.Input(shape=(196, 196, 3))
x = Conv2D(32, (3, 3), activation='relu')(inputs)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(128, (3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

# final model
model = Model(inputs=inputs, outputs=predictions)
```

While the CNN model took over 30 minutes to train and only achieved a test accuracy of 65%:



```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 3615 images belonging to 30 classes.
Found 150 images belonging to 30 classes.
Found 150 images belonging to 30 classes.
Epoch 1/10
226/226 [==============================] - 50s 193ms/step - loss: 3.0218 - accuracy: 0.1674 - val_loss: 2.5216 - val_accuracy: 0.2933
Epoch 2/10
226/226 [==============================] - 43s 190ms/step - loss: 2.2146 - accuracy: 0.3809 - val_loss: 1.8796 - val_accuracy: 0.5000
Epoch 3/10
226/226 [==============================] - 42s 186ms/step - loss: 1.8660 - accuracy: 0.4711 - val_loss: 1.5759 - val_accuracy: 0.5133
Epoch 4/10
226/226 [==============================] - 42s 188ms/step - loss: 1.6387 - accuracy: 0.5380 - val_loss: 1.3608 - val_accuracy: 0.6067
Epoch 5/10
226/226 [==============================] - 42s 185ms/step - loss: 1.5070 - accuracy: 0.5657 - val_loss: 1.5401 - val_accuracy: 0.5800
Epoch 6/10
226/226 [==============================] - 42s 185ms/step - loss: 1.3618 - accuracy: 0.6036 - val_loss: 1.4023 - val_accuracy: 0.6200
Epoch 7/10
226/226 [==============================] - 42s 187ms/step - loss: 1.2412 - accuracy: 0.6362 - val_loss: 1.1123 - val_accuracy: 0.6933
Epoch 8/10
226/226 [==============================] - 42s 187ms/step - loss: 1.1812 - accuracy: 0.6501 - val_loss: 1.3661 - val_accuracy: 0.6267
Epoch 9/10
226/226 [==============================] - 44s 195ms/step - loss: 1.1305 - accuracy: 0.6642 - val_loss: 1.2429 - val_accuracy: 0.6933
Epoch 10/10
226/226 [==============================] - 42s 187ms/step - loss: 1.0679 - accuracy: 0.6811 - val_loss: 1.2484 - val_accuracy: 0.6467
10/10 [==============================] - 1s 46ms/step - loss: 1.2276 - accuracy: 0.6533
Test accuracy: 0.653333306312561
```

The VGG16 model took less than 10 minutes to train and achieved a test accuracy of 90%:



```
Found 3615 images belonging to 30 classes.
Found 150 images belonging to 30 classes.
Found 150 images belonging to 30 classes.
Epoch 1/10
226/226 [==============================] - 63s 246ms/step - loss: 1.3701 - accuracy: 0.6517 - val_loss: 0.3119 - val_accuracy: 0.9000
Epoch 2/10
226/226 [==============================] - 55s 245ms/step - loss: 0.5651 - accuracy: 0.8290 - val_loss: 0.2456 - val_accuracy: 0.9267
Epoch 3/10
226/226 [==============================] - 54s 238ms/step - loss: 0.3582 - accuracy: 0.8869 - val_loss: 0.2816 - val_accuracy: 0.9000
Epoch 4/10
226/226 [==============================] - 54s 237ms/step - loss: 0.3140 - accuracy: 0.9035 - val_loss: 0.2354 - val_accuracy: 0.9133
Epoch 5/10
226/226 [==============================] - 53s 236ms/step - loss: 0.2360 - accuracy: 0.9278 - val_loss: 0.1128 - val_accuracy: 0.9667
Epoch 6/10
226/226 [==============================] - 54s 237ms/step - loss: 0.2093 - accuracy: 0.9331 - val_loss: 0.1974 - val_accuracy: 0.9533
Epoch 7/10
226/226 [==============================] - 53s 236ms/step - loss: 0.1631 - accuracy: 0.9441 - val_loss: 0.1885 - val_accuracy: 0.9400
Epoch 8/10
226/226 [==============================] - 53s 234ms/step - loss: 0.1749 - accuracy: 0.9411 - val_loss: 0.2605 - val_accuracy: 0.9200
Epoch 9/10
226/226 [==============================] - 54s 237ms/step - loss: 0.1545 - accuracy: 0.9524 - val_loss: 0.2306 - val_accuracy: 0.9400
Epoch 10/10
226/226 [==============================] - 54s 239ms/step - loss: 0.1232 - accuracy: 0.9604 - val_loss: 0.3415 - val_accuracy: 0.9133
Epoch 10: early stopping
10/10 [==============================] - 1s 68ms/step - loss: 0.3394 - accuracy: 0.9000
Test accuracy: 0.8999999761581421
```

I also tried to use other pre-trained models, such as ResNet50, VGG19, instead of VGG16, but with ResNet50 I got the accuracy of 25%, and with VGG19 the accuracy of 88%. Although the performance was on the same level. Therefore, I settled on VGG16.

However, there is still room for improvement, especially for the classes with fewer samples. One possible next step is to use transfer learning with **other** pre-trained models and compare their performance with the VGG16 model. Another possible next step is to collect more data for the classes with fewer samples or use techniques such as data synthesis to balance the dataset.

❖ Conclusion

In this project, VGG16 was used as a pre-trained model for image classification of 30 different sports balls types. Also used data augmentation techniques such as rotation, zooming and flipping to increase the diversity of the dataset. After training the model, I achieved a validation accuracy of 91.33% and a test accuracy of 90%. This model was able to correctly classify new, unseen images with a high degree of accuracy, indicating that the model was able to generalize well to new data. Overall, the use of pre-trained models such as VGG16 can greatly improve the accuracy of image classification tasks, while also reducing the amount of time needed for training.

- **Sources:**
    - https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a
    - https://keras.io/api/applications/vgg/
    - https://www.kaggle.com/code/swetash/ball-classification-from-scratch-95-acc