**UNIVERSITY OF RIJEKA  
FACULTY OF INFORMATICS   
AND DIGITAL TECHNOLOGIES**

**Graduate study of computer science**

**Project from the course**

**Machine and Deep Learning**

**Cinematic Analytics: Investigating the Correlation Between Model Complexity and Score**

**Author: Nikita Ferents**

Mentors: mag. inf. Kristina Host

Rijeka, 2024

Contents

[1. Project Definition 2](#_Toc169558291)

[1.1 Objective: 2](#_Toc169558292)

[1.2 Scope: 2](#_Toc169558293)

[1.3 Methodology: 2](#_Toc169558294)

[2. Working with Source 3](#_Toc169558295)

[2.1 Papers with Code Overview: 3](#_Toc169558296)

[2.2 Navigating the Platform 3](#_Toc169558297)

[2.3 Dataset Selection 4](#_Toc169558298)

[2.4 Exploring the Dataset 5](#_Toc169558299)

[2.4.1 Benchmark Results: 5](#_Toc169558300)

[2.4.2 Research Papers: 7](#_Toc169558301)

[2.4.3 Code Implementations: 8](#_Toc169558302)

[2.4.4 Evaluation Tables: 8](#_Toc169558303)

[3. Pretrained Model Choice 9](#_Toc169558304)

[3.1 Introduction 9](#_Toc169558305)

[3.2 Overview of the YOLO Model 9](#_Toc169558306)

[3.3 Key Features 10](#_Toc169558307)

[3.4 Models 10](#_Toc169558308)

[3.5 Advantages: 10](#_Toc169558309)

[4. Working with AI 11](#_Toc169558310)

[4.1 Full code Overview 11](#_Toc169558311)

[4.2 TensorFlow Models 18](#_Toc169558312)

[4.2.1 Simple CNN 18](#_Toc169558313)

[4.2.2 VGG-16 Inspired Model 22](#_Toc169558314)

[4.3 Ultralytics YOLO Model 25](#_Toc169558315)

[4.3.1 Installing and Setting Up YOLO 25](#_Toc169558316)

[5. Conclusion 26](#_Toc169558317)

[5.1 Summary 26](#_Toc169558318)

[5.2 Key Takeaways: 27](#_Toc169558319)

[6. References: 27](#_Toc169558320)

# Project Definition

## Objective:

The objective of this project is to investigate how various factors of Artificial Intelligence (AI) models, such as complexity, type, source, training time, and size, influence the accuracy of image classification tasks. The study aims to provide insights into optimizing these factors to enhance model performance.

## Scope:

Data Acquisition: Utilize the Web resourses such as [paperswithcode](https://paperswithcode.com/) to collect data on the relevant field, such as: Datasets focusing on metrics such as popularity scores and viewer ratings.

Data Processing: Implement a data engineering pipeline to clean, preprocess, and structure the dataset for analysis.

Data Storage: Employ appropriate storage technologies to manage the streaming or batch data efficiently.

Analytical Model: Develop regression models (linear and polynomial) to analyze the data and extract insights into the correlation between movie popularity and ratings.

## Methodology:

Data Collection: Gather datasets from various sources, primary [paperswithcode](https://paperswithcode.com/) for training and testing the models. Ensure diversity in data to cover different scenarios and complexities.

Model Selection and Development: Choose a range of AI models varying in complexity, type, size, and source. Using both pretrained and precompiled models and new, compiled my myself. Implement or source pre-trained models as required.

Training and Evaluation:Train selected models using different training durations and datasets. Evaluate the models using standard accuracy metrics.

Analysis: Analyze the data to determine how each factor influences classification accuracy. Use statistical methods to validate findings.

# Working with Source

## Papers with Code Overview:

The mission of Papers with Code is to create a free and open resource with Machine Learning papers, code, datasets, methods and evaluation tables.

This is best done together with the community, supported by NLP and ML.

To ensure high quality of data, all edits are monitored on Slack on the #recentchanges channel. This is an open channel and everyone is invited to follow and review contributions.

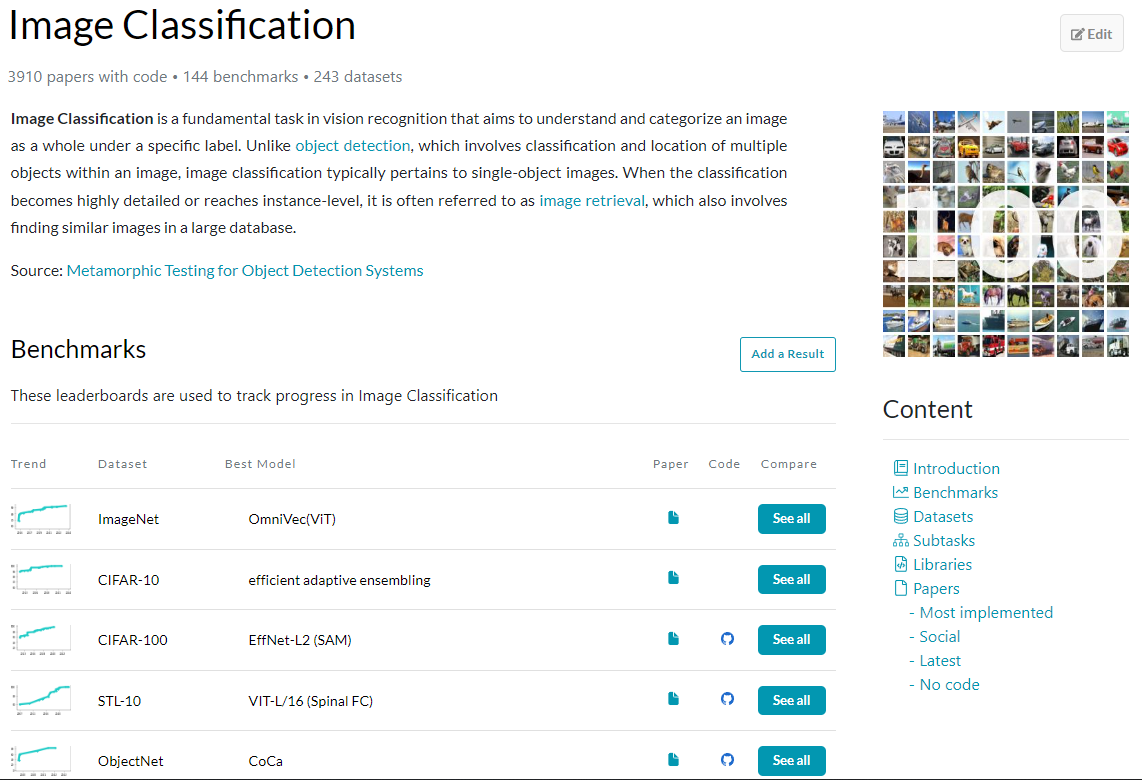
For a result to be included as a benchmark result we require that the paper is published as pre-print, in a conference or a journal. Having code is strongly encouraged but not required so we can capture the latest published results even before the code has been released.

All content on this website is openly licenced under CC-BY-SA (same as Wikipedia) and everyone can contribute

They also operate [specialized portals](https://portal.paperswithcode.com/) for papers with code in astronomy, physics, computer sciences, mathematics and statistics.

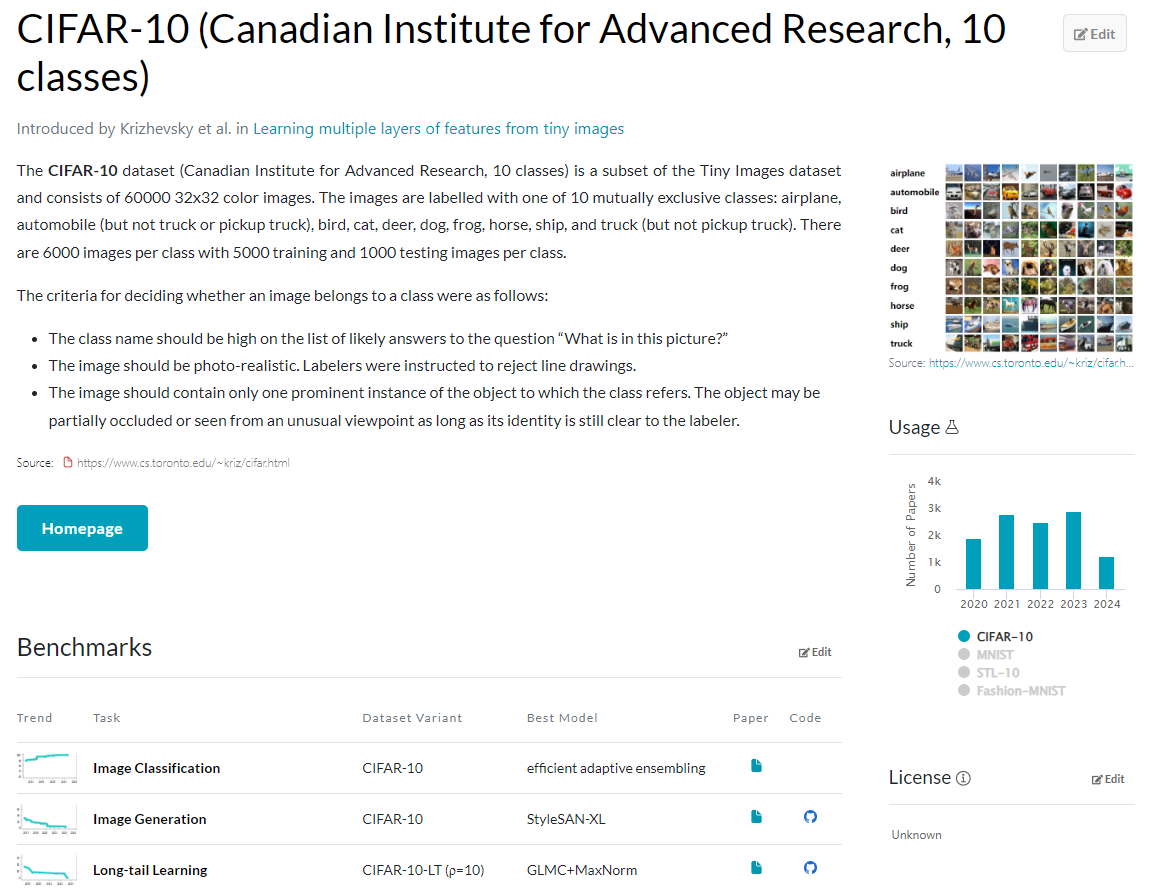
## Navigating the Platform

Upon visiting the "Papers with Code" website, the user is greeted with a clean interface featuring a search bar and various categories of AI tasks. For our project, I focused on the "Image Classification" task, which is prominently listed among the available categories.



## ****Dataset Selection****

"Papers with Code" lists datasets used in the evaluation of these task. For my project, it was important to use diverse datasets to understand the impact of data source on model performance. I found dataset such as CIFAR-10, which is commonly used in image classification research. The website provided links to download these datasets and detailed descriptions, including the number of classes, image resolution, and size of the datasets.

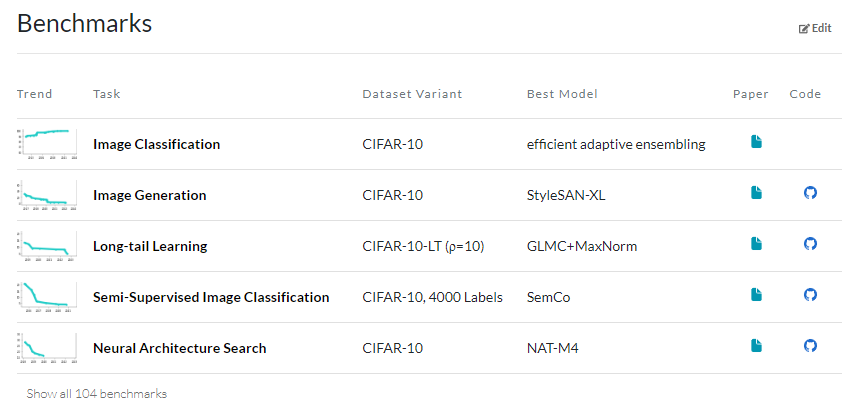
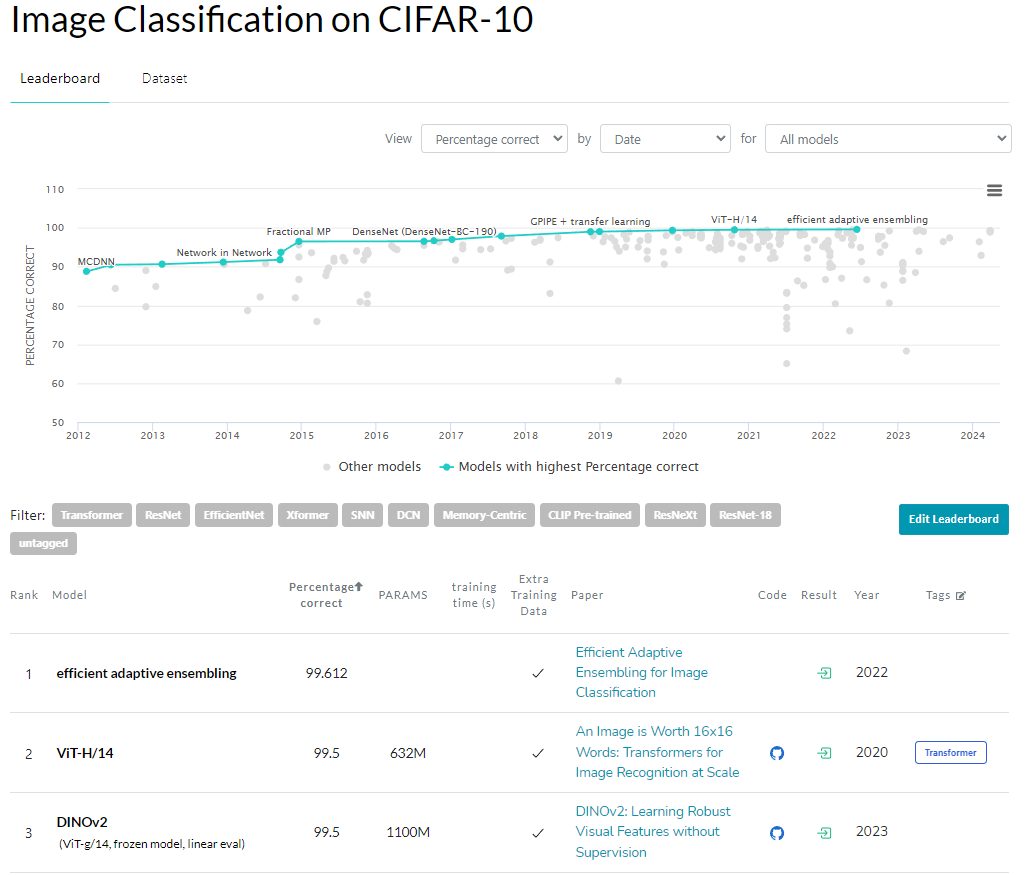


## Exploring the Dataset

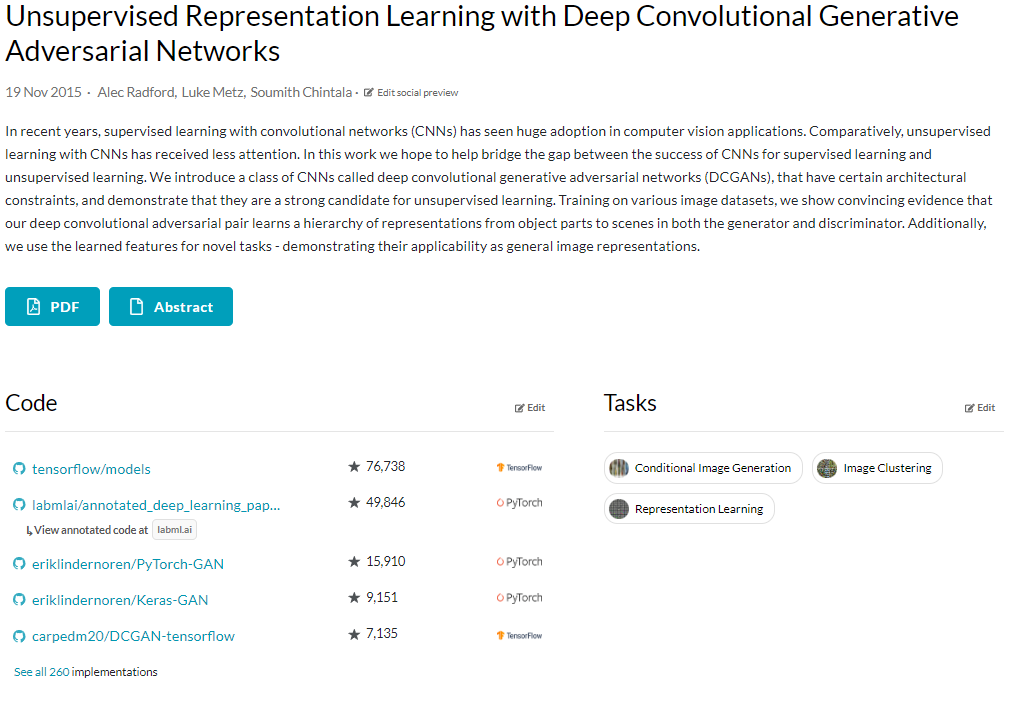
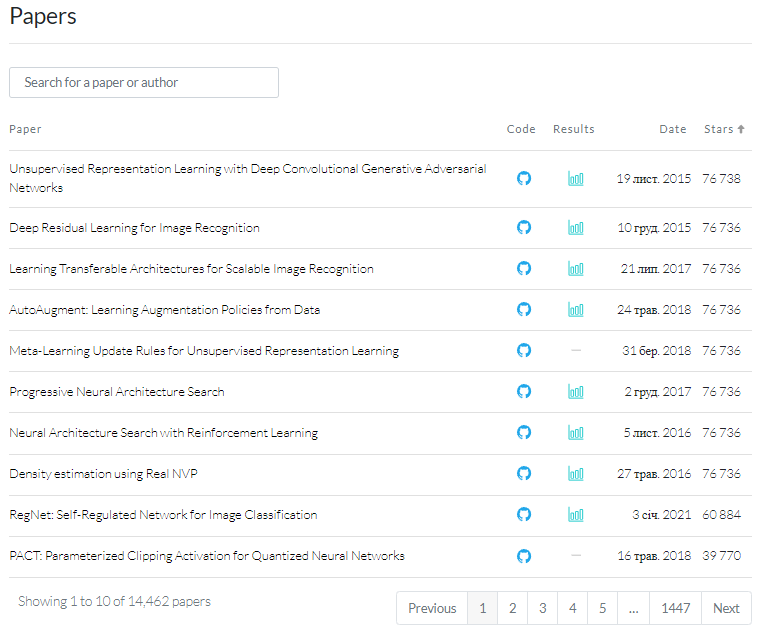
**Dataset Overview**: The CIFAR-10 dataset is a well-established image classification dataset that includes 60,000 color images divided into 10 different classes, with each class containing 6,000 images. It is split into 50,000 training images and 10,000 test images. This dataset is commonly used to benchmark machine learning models in image classification tasks.

**Key Features Available**:

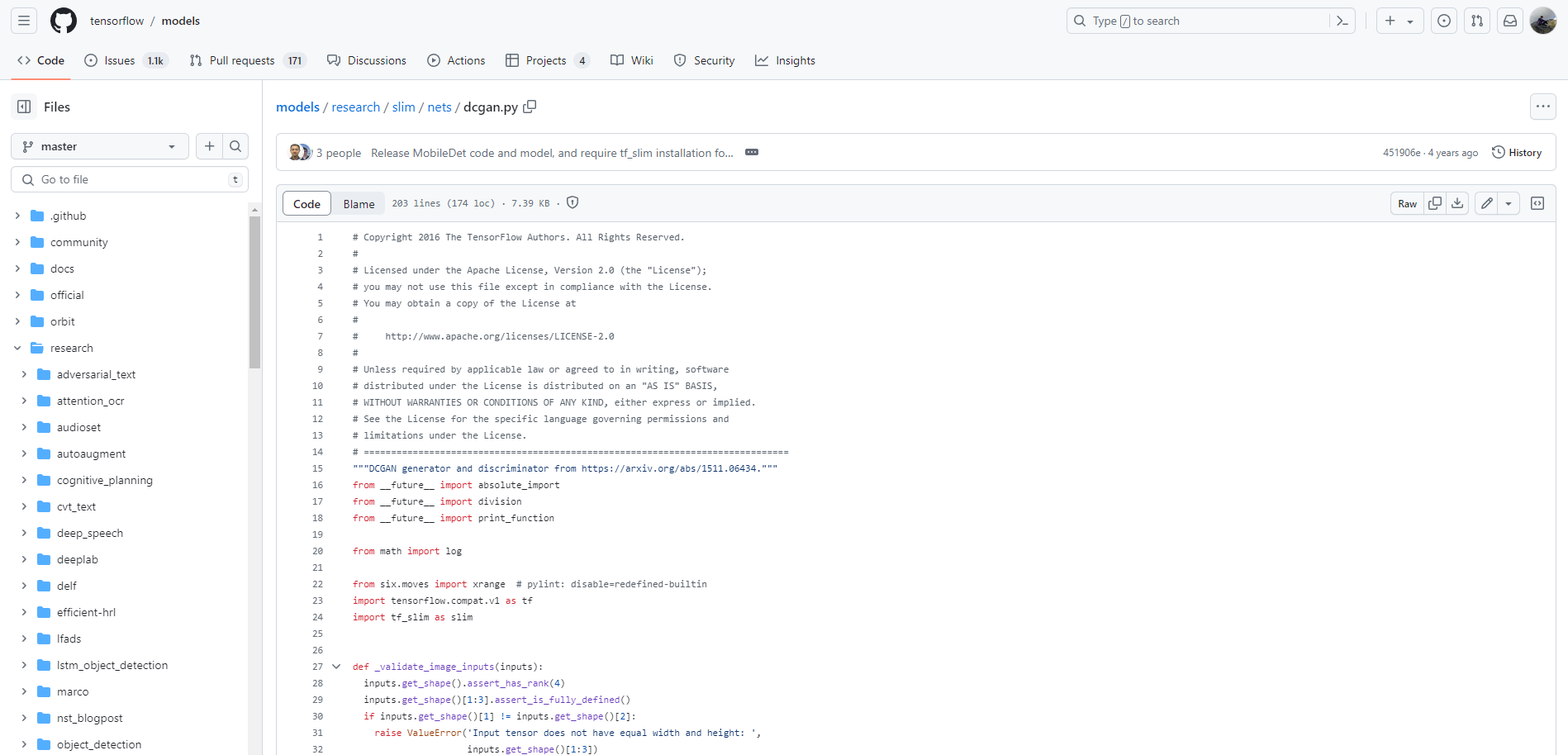
### Benchmark Results:

* + The website provides a list of models that have been evaluated on the CIFAR-10 dataset, along with their performance metrics, such as accuracy. This list includes state-of-the-art models and their respective scores, which helps in identifying the most effective models.
  + Using the detailed evaluation tables, I was able to compare the performance of different models under similar conditions. This comparison helped me draw conclusions about which types of models were more effective for image classification tasks and the reasons behind their performance.

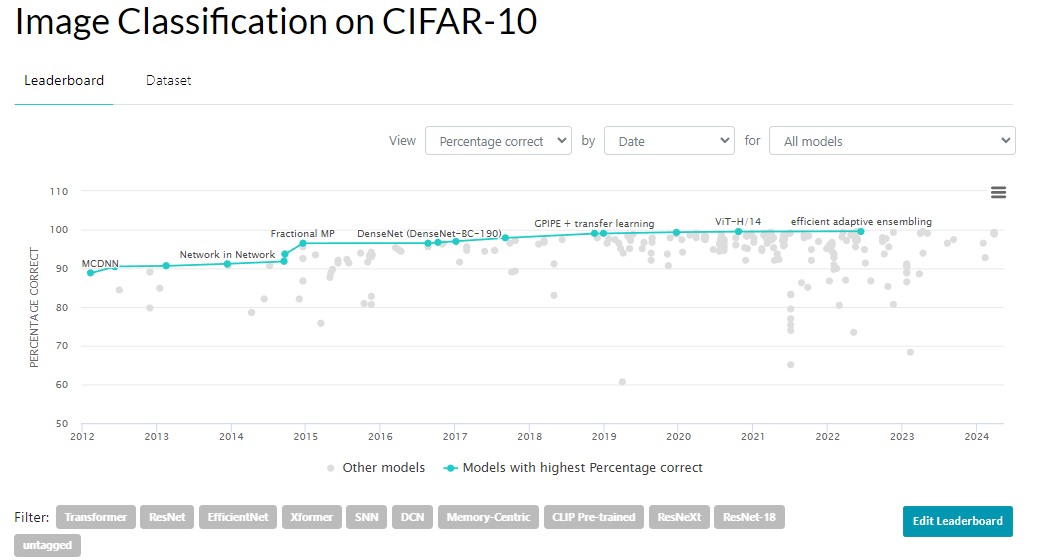
### Research Papers:

* + "Papers with Code" links to research papers associated with each model. These papers describe the methodologies and experiments conducted, offering in-depth insights into how different models perform on CIFAR-10.
  + By reviewing the associated research papers, I gained insights into the training processes and data preprocessing techniques used by different models. This information was crucial for setting up my experiments.

### Code Implementations:

* + For each model listed, the website provides links to GitHub repositories where the code implementation can be found. This is invaluable for replicating the results and understanding the nuances of each model's training process.

### Evaluation Tables:

* + The evaluation tables on the site summarize the performance of different models. They provide a clear comparative analysis of various models based on metrics like accuracy, enabling a better understanding of which models are more effective and why.

# Pretrained Model Choice

## Introduction

For my project on investigating the relationship between AI model characteristics and image classification accuracy, I decided to include the YOLO (You Only Look Once) model due to its innovative approach to real-time object detection and classification. This section describes the YOLO model, its relevance to image classification, and the steps I took to integrate it into my project.

## Overview of the YOLO Model

**YOLO Model Basics**: YOLO is a popular deep learning model designed for real-time object detection. Unlike traditional methods that use a sliding window approach, YOLO frames object detection as a single regression problem, directly predicting bounding boxes and class probabilities from the entire image in one evaluation. This makes YOLO extremely fast and efficient compared to other object detection models.



## Key Features

* **Speed**: YOLO processes images in real-time, achieving high frame rates.
* **Unified Architecture**: It uses a single neural network for the entire image, making it simpler and more straightforward.
* **Accuracy**: Despite its speed, YOLO maintains high accuracy, especially in detecting large objects and those with distinct features.

## ****Models****

While YOLO is primarily known for object detection, its architecture can be adapted for image classification tasks by focusing on the class probabilities output. This adaptation makes YOLO a versatile model for both detection and classification tasks.

**YOLOv8 Classification Models**: YOLOv8 offers several models for image classification, each varying in size and complexity, denoted by suffixes such as -cls. These models are pretrained on the ImageNet dataset and include:

* **YOLOv8n-cls**: Smallest and fastest, with lower accuracy.
* **YOLOv8s-cls**: Slightly larger with improved accuracy.
* **YOLOv8m-cls**: Medium-sized, balancing speed and accuracy.
* **YOLOv8l-cls**: Large model with high accuracy but slower.
* **YOLOv8x-cls**: Largest and most accurate, but slowest.

**Performance Metrics**:

* **Top-1 and Top-5 Accuracy**: Measures of the model’s prediction accuracy on the ImageNet validation set.
* **Speed**: Inference time on CPU and GPU, indicating the model's efficiency.
* **Parameters and FLOPs**: Indicators of model complexity and computational requirements.

## Advantages:

* **Efficiency**: Leveraging YOLO’s speed for classification can lead to faster training and inference times.
* **Scalability**: YOLO’s architecture can handle various image sizes and complexities, making it suitable for different classification tasks.

# Working with AI

## Full code Overview

<https://colab.research.google.com/drive/1x7U0m-r4Ma-FdIHlIm9UDiHsOsZWpJQu?usp=sharing>

import tensorflow as tf

from tensorflow import keras

from keras.datasets import cifar10

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

y\_train\_one\_hot = keras.utils.to\_categorical(y\_train, 10)

y\_test\_one\_hot = keras.utils.to\_categorical(y\_test, 10)

x\_train = x\_train.astype('float32')/255

x\_test = x\_test.astype('float32')/255

modeleasy= keras.models.Sequential()

modeleasy.add(keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input\_shape=(32,32,3)))

modeleasy.add(keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))

modeleasy.add(keras.layers.MaxPooling2D(pool\_size=(2, 2)))

modeleasy.add(keras.layers.Dropout(0.25))

modeleasy.add(keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))

modeleasy.add(keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))

modeleasy.add(keras.layers.MaxPooling2D(pool\_size=(2,2)))

modeleasy.add(keras.layers.Dropout(0.25))

modeleasy.add(keras.layers.Flatten())

modeleasy.add(keras.layers.Dense(512, activation='relu'))

modeleasy.add(keras.layers.Dropout(0.5))

modeleasy.add(keras.layers.Dense(10, activation='softmax'))

modeleasy.summary()

modeleasy.compile(loss='categorical\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

history\_of\_model = modeleasy.fit(x\_train, y\_train\_one\_hot,

batch\_size=32, epochs=20, #stavi max 15

validation\_split=0.2)

import pandas as pd

pd.DataFrame(history\_of\_model.history).plot(figsize=(8,5))

plt.grid(True)

plt.xlabel('Epoch')

plt.show()

modeleasy.save('my\_easy\_model.keras')

modeleasy.evaluate(x\_test, y\_test\_one\_hot)

import numpy as np

probabilities = modeleasy.predict(np.array([x\_test[150]]))

probabilities

class\_names = ['airplane', 'car', 'bird', 'cat', 'dear', 'dog', 'frog',

'horse', 'ship', 'truck']

index = np.argsort(probabilities[0,:])

for i in range (9,5,-1): #prvih nekoliko probabilities

print(class\_names[index[i]], ":", probabilities[0,index[i]])

MODEL DOUBLE Very Deep Convolutional Networks for Large-Scale Image Recognition(VGG-16)

modeldouble= keras.models.Sequential()

modeldouble.add(keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same', input\_shape=(32,32,3)))

modeldouble.add(keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.MaxPooling2D(pool\_size=(2, 2)))

modeldouble.add(keras.layers.Dropout(0.25))

modeldouble.add(keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.MaxPooling2D(pool\_size=(2,2)))

modeldouble.add(keras.layers.Dropout(0.25))

modeldouble.add(keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.MaxPooling2D(pool\_size=(2,2)))

modeldouble.add(keras.layers.Dropout(0.25))

modeldouble.add(keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.MaxPooling2D(pool\_size=(2,2)))

modeldouble.add(keras.layers.Dropout(0.25))

modeldouble.add(keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same'))

modeldouble.add(keras.layers.MaxPooling2D(pool\_size=(2,2)))

modeldouble.add(keras.layers.Dropout(0.25))

modeldouble.add(keras.layers.Flatten())

modeldouble.add(keras.layers.Dense(4096, activation='relu'))

modeldouble.add(keras.layers.Dense(4096, activation='relu'))

modeldouble.add(keras.layers.Dense(4096, activation='relu'))

modeldouble.add(keras.layers.Dropout(0.5))

modeldouble.add(keras.layers.Dense(10, activation='softmax'))

modeldouble.summary()

modeldouble.compile(loss='categorical\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

history\_of\_model = modeldouble.fit(x\_train, y\_train\_one\_hot,

batch\_size=32, epochs=20, #stavi max 15

validation\_split=0.2)

import pandas as pd

pd.DataFrame(history\_of\_model.history).plot(figsize=(8,5))

plt.grid(True)

plt.xlabel('Epoch')

plt.show()

modeldouble.evaluate(x\_test, y\_test\_one\_hot)

modeldouble.save('my\_double\_model.keras')

import numpy as np

probabilities = modeldouble.predict(np.array([x\_test[150]]))

for i in range (9,5,-1): #prvih nekoliko probabilities

print(class\_names[index[i]], ":", probabilities[0,index[i]])

keras.backend.clear\_session()

!pip install ultralytics

from ultralytics import YOLO

# Load a model

model = YOLO("yolov8n-cls.pt") # load a pretrained model (recommended for training)

# Train the model

results = model.train(data="cifar10", epochs=5, imgsz=32)

result = model(x\_test[150])

result = model(x\_test)

y\_test[160]

import matplotlib.pyplot as plt

plt.imshow(x\_test[150])

model.save('my\_cifar10\_model.h5')

model.eval()

results = model.train(data="imagenet10", epochs=100, imgsz=32)

model.eval()

model.save('my\_imagenet10\_model.h5')

## TensorFlow Models

### Simple CNN

The first model is a straightforward CNN using Keras, designed to classify CIFAR-10 images.

**Loading and Preprocessing Data:**

The CIFAR-10 dataset is loaded and preprocessed by normalizing the pixel values and converting the labels to one-hot encoded format.

from keras.datasets import cifar10

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

y\_train\_one\_hot = keras.utils.to\_categorical(y\_train, 10)

y\_test\_one\_hot = keras.utils.to\_categorical(y\_test, 10)

x\_train = x\_train.astype('float32')/255

x\_test = x\_test.astype('float32')/255

* **Normalization**: Converts pixel values from 0-255 to 0-1.
* **One-Hot Encoding**: Converts class labels to binary class matrices.

**Building the Model:**

A Sequential model is constructed with multiple convolutional, pooling, and dropout layers, culminating in a dense softmax layer for classification.

modeleasy = keras.models.Sequential()

modeleasy.add(keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input\_shape=(32,32,3)))

modeleasy.add(keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))

modeleasy.add(keras.layers.MaxPooling2D(pool\_size=(2, 2)))

modeleasy.add(keras.layers.Dropout(0.25))

modeleasy.add(keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))

modeleasy.add(keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'))

modeleasy.add(keras.layers.MaxPooling2D(pool\_size=(2,2)))

modeleasy.add(keras.layers.Dropout(0.25))

modeleasy.add(keras.layers.Flatten())

modeleasy.add(keras.layers.Dense(512, activation='relu'))

modeleasy.add(keras.layers.Dropout(0.5))

modeleasy.add(keras.layers.Dense(10, activation='softmax'))

modeleasy.summary()

### 

* **Convolutional Layers**: Extract features using filters.
* **MaxPooling Layers**: Downsample feature maps to reduce spatial dimensions.
* **Dropout Layers**: Prevent overfitting by randomly dropping neurons during training.
* **Dense Layer**: Fully connected layer with ReLU activation.
* **Output Layer**: Softmax activation for multi-class classification.

**Compiling and Training the Model:**

The model is compiled with categorical cross-entropy loss, Adam optimizer, and accuracy metric. It is trained for 20 epochs with a validation split of 20%.

modeleasy.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

history\_of\_model = modeleasy.fit(x\_train, y\_train\_one\_hot, batch\_size=32, epochs=20, validation\_split=0.2)

**Evaluating the Model:**

The model's performance is evaluated on the test set, and the model is saved for future use.

modeleasy.evaluate(x\_test, y\_test\_one\_hot)

modeleasy.save('my\_easy\_model.keras')



**Visualizing Training History:**

The training and validation accuracy and loss are plotted to visualize the model's learning process.

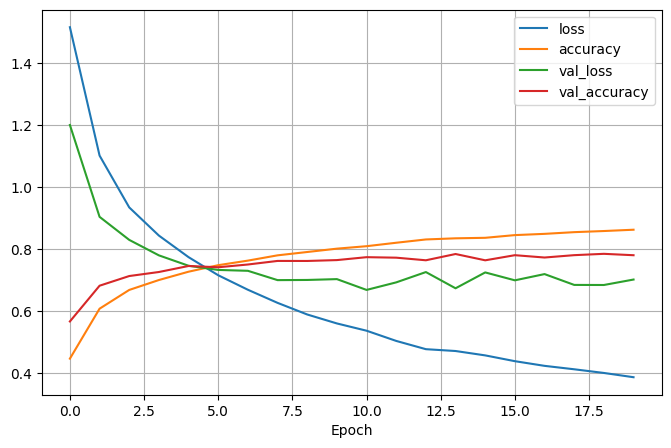
import pandas as pd

pd.DataFrame(history\_of\_model.history).plot(figsize=(8,5))

plt.grid(True)

plt.xlabel('Epoch')

plt.show()

****

**Making Predictions:**

The model makes predictions on a test image, and the class probabilities are printed.

import numpy as np

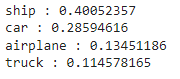
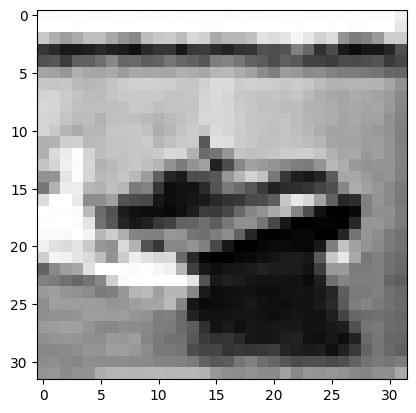
probabilities = modeleasy.predict(np.array([x\_test[150]]))

class\_names = ['airplane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

index = np.argsort(probabilities[0,:])

for i in range(9, 5, -1):

print(class\_names[index[i]], ":", probabilities[0, index[i]])



### VGG-16 Inspired Model

The second model is a deeper CNN inspired by the VGG-16 architecture, which is more complex and designed for larger-scale image recognition tasks.

**Building the Model:**

This model includes more convolutional layers with increasing filter sizes, and several fully connected layers, making it significantly deeper than the first model.

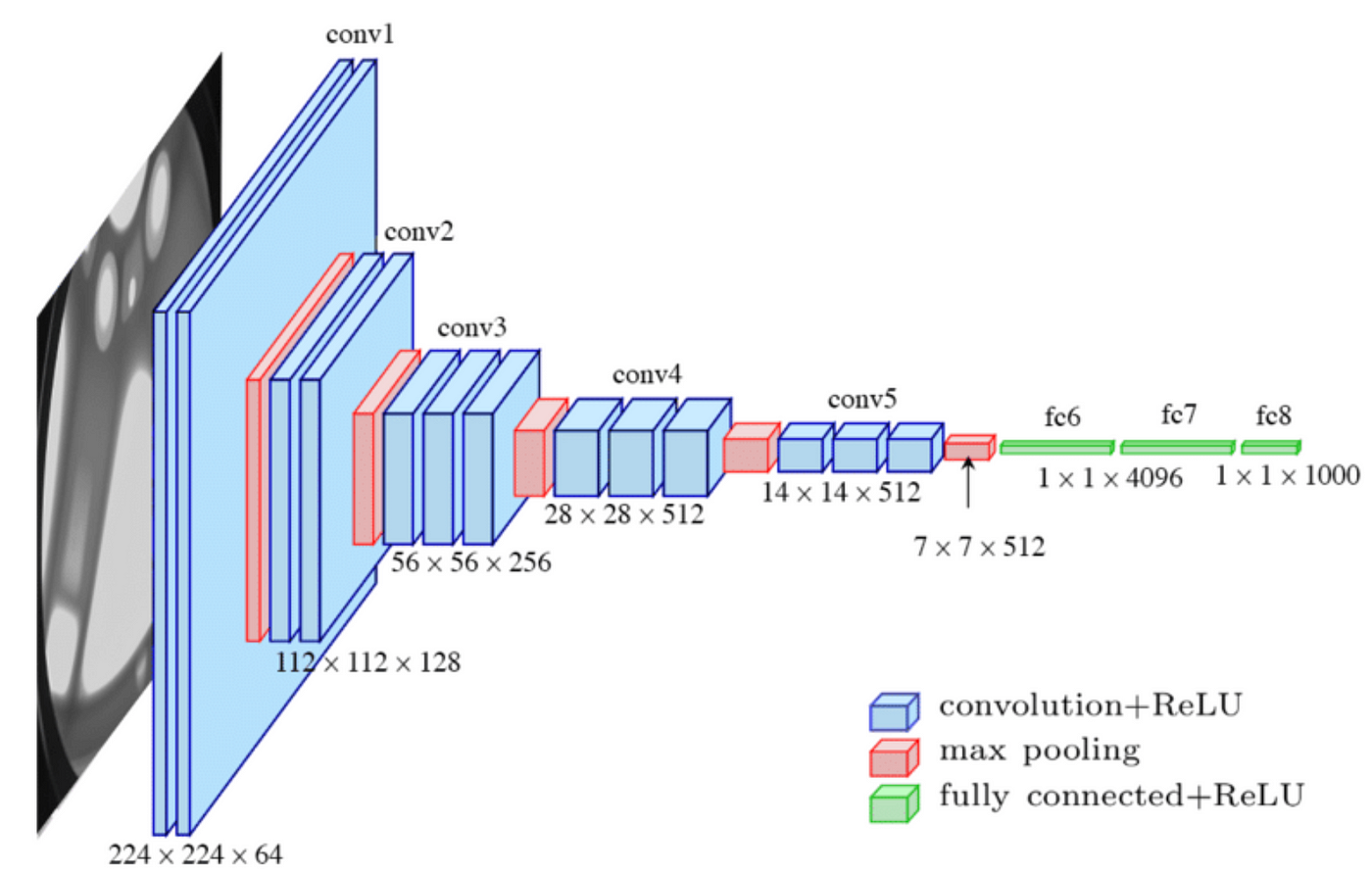
modeldouble = keras.models.Sequential()

modeldouble.add(keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same', input\_shape=(32,32,3)))

# Additional layers...

modeldouble.add(keras.layers.Dense(10, activation='softmax'))

modeldouble.summary()



* **Additional Convolutional Layers**: More layers increase model capacity.
* **Fully Connected Layers**: Three dense layers with ReLU activation enhance the model's ability to learn complex patterns.

**Compiling and Training:**

Similar to the simple model, this model is compiled and trained, but it is designed to handle more complex feature extraction due to its depth.

modeldouble.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

history\_of\_model = modeldouble.fit(x\_train, y\_train\_one\_hot, batch\_size=32, epochs=20, validation\_split=0.2)

**Evaluating and Saving:**

The model is evaluated on the test set and saved for future use.

modeldouble.evaluate(x\_test, y\_test\_one\_hot)

modeldouble.save('my\_double\_model.keras')

### 

**Visualizing Training History:**

The training and validation accuracy and loss are plotted for this model as well.

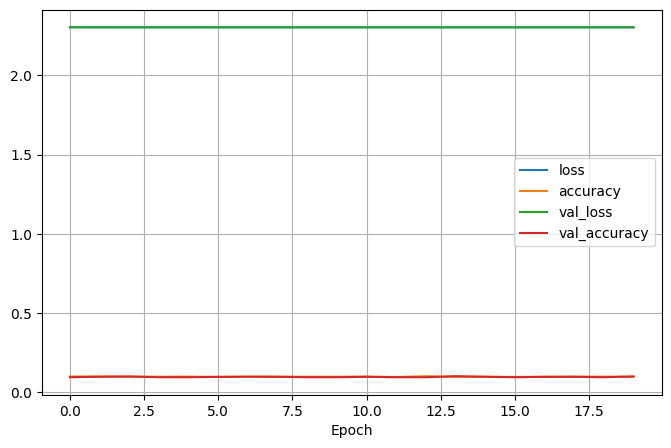
import pandas as pd

pd.DataFrame(history\_of\_model.history).plot(figsize=(8,5))

plt.grid(True)

plt.xlabel('Epoch')

plt.show()



**Making Predictions:**

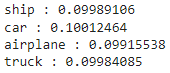
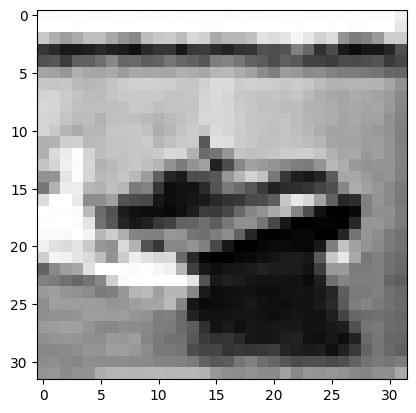
The model makes predictions on a test image, and the class probabilities are printed.

import numpy as np

probabilities = modeldouble.predict(np.array([x\_test[150]]))

for i in range(9, 5, -1):

print(class\_names[index[i]], ":", probabilities[0, index[i]])



## Ultralytics YOLO Model

### **Installing and Setting Up YOLO**

YOLO models are known for their efficiency in real-time object detection and classification.

**Installing Ultralytics:**

!pip install ultralytics

from ultralytics import YOLO

* **YOLO Installation**: Installs the Ultralytics YOLO package for model training and evaluation.

**Loading and Training the YOLO Model:** The YOLOv8n classification model is loaded and trained on the CIFAR-10 dataset.

model = YOLO("yolov8n-cls.pt")

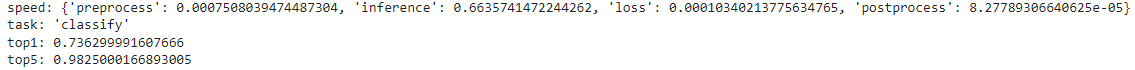
results = model.train(data="cifar10", epochs=5, imgsz=32)

* **Model Loading**: Uses a pre-trained YOLOv8n classification model.
* **Training**: Trains the model on CIFAR-10 for 5 epochs.

**Evaluating the Model:** The trained model is used to make predictions and is evaluated on the test set.

result = model(x\_test)

model.save('my\_cifar10\_model.h5')

model.eval()

# Conclusion

## Summary

In this project, we investigated the performance of three different image classification models on the CIFAR-10 dataset, which consists of low-resolution 32x32 pixel images.

**Simple CNN:**

* **Loss:** 0.7359
* **Accuracy:** 0.7728

The Simple CNN demonstrated a satisfactory performance, achieving a decent accuracy given the low-resolution nature of the dataset. This model's moderate complexity allowed it to generalize well on CIFAR-10 images.

**VGG-16 Inspired Model:**

* **Loss:** 2.3026
* **Accuracy:** 0.1000

The VGG-16 inspired model underperformed significantly, which can be attributed to the low resolution of the CIFAR-10 images. VGG-16 is designed for high-resolution images and its deep architecture could not effectively capture useful features from the 32x32 pixel images, resulting in poor performance.

**YOLO Model:**

* **Top-1 Accuracy:** 0.7363
* **Top-5 Accuracy:** 0.9825

The YOLO model, while typically used for object detection, was repurposed for classification in this project. It achieved comparable Top-1 accuracy to the Simple CNN but excelled with a very high Top-5 accuracy. This indicates that YOLO was able to correctly include the true class in its top five predictions most of the time, showing its robustness in handling the CIFAR-10 dataset.

## Key Takeaways:

* The Simple CNN provided a balanced performance, suitable for low-resolution images.
* The VGG-16 inspired model struggled with CIFAR-10 due to its low-resolution nature, highlighting that VGG-16 is better suited for high-resolution image classification tasks.
* The YOLO model showed strong performance, especially in Top-5 accuracy, suggesting its potential versatility beyond object detection.

Overall, this project highlights the importance of matching model complexity and architecture to the characteristics of the dataset. While deep models like VGG-16 are powerful for high-resolution images, simpler models or versatile models like YOLO can be more effective for lower-resolution tasks such as those presented by CIFAR-10.

# ****References:****

* mag. inf. Kristina Host’s Lectures
* Pandas documentation: <https://pandas.pydata.org/>
* Scikit-learn documentation: <https://scikit-learn.org/stable/>
* Matplotlib documentation: <https://matplotlib.org/>
* Keras documentation: <https://keras.io/>
* Google Colab documentation: <https://colab.research.google.com/notebooks/intro.ipynb>
* Ultralytics(YOLO) documentation: <https://docs.ultralytics.com/>
* Ultralytics(YOLO) image classification tutorial: [https://docs.ultralytics.com/tasks/classify/#tutorial](https://docs.ultralytics.com/tasks/classify/%23tutorial)
* Papers with Code CIFAR-10 Dataset: <https://paperswithcode.com/dataset/cifar-10>
* TensorFlow Documentation: <https://www.tensorflow.org/>
* Original CIFAR-10 Documentation: <https://www.cs.toronto.edu/~kriz/cifar.html>
* StackOverflow for Debugging: <https://stackoverflow.com/search?q=YOLO&s=9cc46da5-03b3-4878-8064-511028e2522f>