Assignment 1:

INDUSTRIAL AUTONOMOUS ROBOTIC VEHICLES:

Introduction:

A self-driving vehicle is also denoted as autonomous vehicle, driverless vehicle, and robotic vehicle. This kind of vehicle has a capability to recognize the environment and drive autonomously without input from human. Lidar, Radar, Sonar, GPS, Odometry and some other inertial measurement units are used in this robotic vehicle to sense and take decisions quickly and accurately. The robotic vehicle is supposed to read the signage in the driving path, should recognize the navigation path, and should avoid obstacles automatically. These activities will be monitored and interpreted with the sensory information using highly sophisticated control systems. This autonomous technology is used in different domains such as robot-taxis, connected vehicle platoons, defense based aerial vehicles, marine based gliders and so on. This robotic system requires varies stages in the development. There is no fully functionable autonomous driving systems in the market today. As per the system of SAE (Society of Automotive Engineers), the vehicle autonomy is classified in six different levels. In simple terms, Level 0 - no automation; Level 1 - hands on/shared control; Level 2 - hands off; Level 3 - eyes off; Level 4 - mind off, and Level 5 - steering wheel optional. According to research, the vehicle which lies above the level 3 has a vital role in the market and takes marginal portion as well. For the first time, the company WAYMO has released driverless taxies for the first time. Even though its autonomous still it has a remote human operation. The first legal level 3 vehicle is shown by Honda. consequently, Toyota has introduced level 4 system in Tokyo. Examples of autonomous vehicles, CNH autonomous tractor, Tiger X-1, Slocum G3 Glider, The Marker unmanned ground vehicle (UGV).

Literature review:

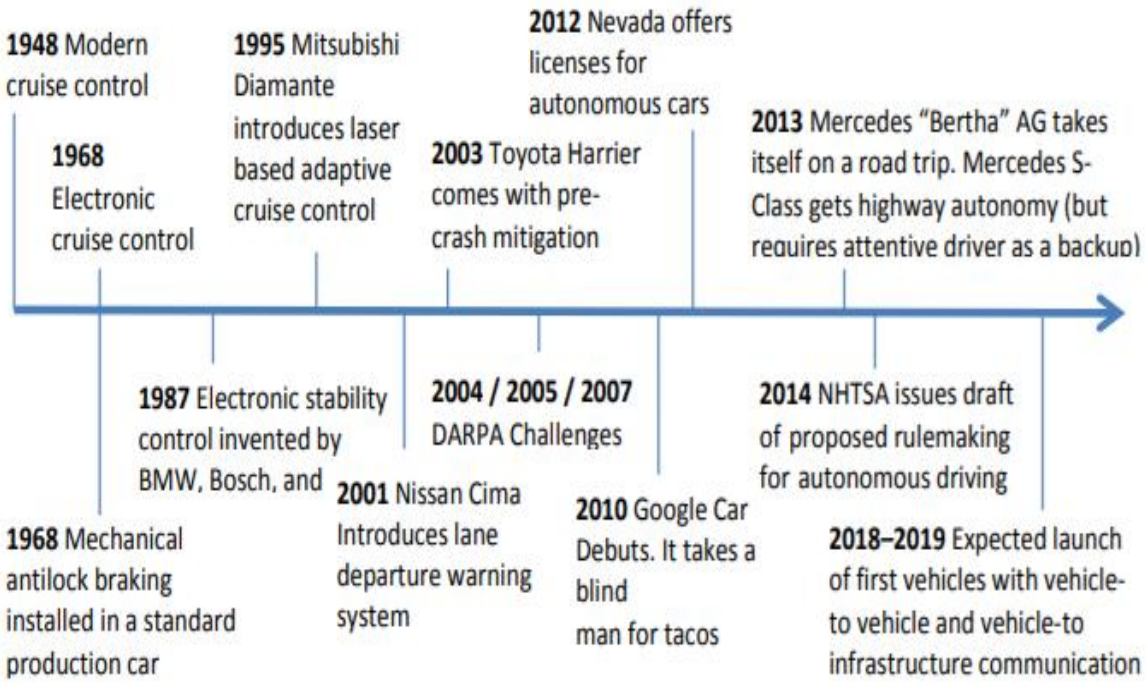


Figure 1: Sixty-five years of automotive baby steps (Ross, 2014, p. 62)

From the figure 2, we can recognize that the development of autonomous vehicles scientific research completely for last decade, which clearly represents the gradual increase of the interest in autonomous technology. In addition to that, vital achievements that speaks the development and research especially in DARPA grand challenge. Between the year 2004 and 2005 or the urban challenge in the end of 2007, that were primary vital competitions in the field of autonomous systems. Moreover, between the year 2013 and 2014 a drastic raise of 60.8 percent can be observed.

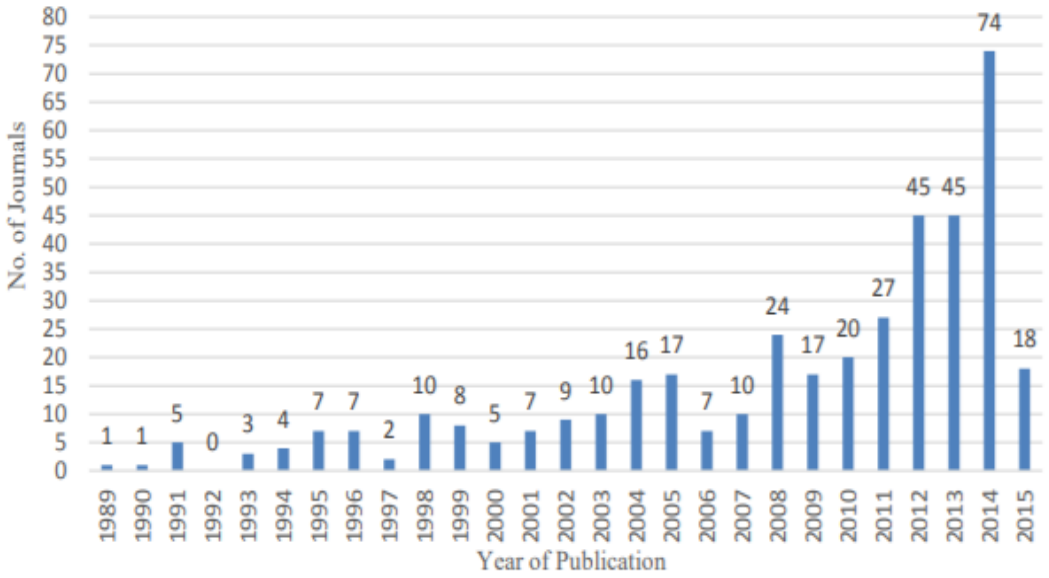


Figure 2: Analysis of Publications over time

Robot vehicle design and operation:

CNH Tractor:

The CNH industrial autonomous tractor has been manufactured with the aim of remote deployment, monitoring and machine controls. The tractor released in two different versions. IH Magnum and New Holland T8 high-horsepower conventional tractors operations is based on satellite signals of GPS for ultra-precise guidance and quick processing of real-time on-field data. To completely remove the operator from cab, IH Magnum concept is used and to maintain the flexibility with traditional human works for roadway transportation, the new HOLLAND T8 NHDrive is used. The driverless technology is capable of using conventional engine, chassis, transmission and so on. The sophisticated headlights, designed bonnet, silhouette, and developed by carbon fiber front fenders. For the sake for status lights, two tone back and red wheel rims and LED are used. The tractor is highly secured where it can record, transfer data, and provide feedback securely. For human interactive experience, an interactive interface has been designed to operate the vehicle. By inputting the boundary map of the field, the tractor operation can be started. Then the user needs to plot the path of the field using the path planning software which is pre-installed in the system. By doing this, high profit with minimal complexity and drastic accuracy can be achieved. The CNH tractor technology can play a vital role in cultivation, planting, spraying, and mowing because this works requires less operator intervention. Path plotting can be done manually if in case requires refueling or when custom paths are essential. Consequently, after finishing the

path plotting, from the pre-programmed toolbar, the user can select the job by simply choosing it where the whole process requires less than 30 seconds. On the other hand, the entire system workings can be spectated or monitored with the use of desktop screen or through tablet interface. This will enhance the efficiency and productivity by taking right decisions by transmitting the real-time data to the user. However, the user will have the entire ownership of their own data.

Fig 1. CNH autonomous tractor

Hyundai Tiger and elevate X-1:

TIGER: Transforming Intelligent Ground Excursion Robot.

X-1: Version of the Robot.

Hyundai has revealed a elevate concept walking car in 2019 at consumer electronics show. Now they went a step further which could help in search and rescue, exploratory mars missions, scientific research and development, defense and so on. its about 20 inches long by 12 inches across by 7 inches tall. The robot can carry load with it. However, it's a complete autonomous vehicle where it can carry humans. Furthermore, the company Hyundai has mentioned that the actual production version will be three to four times bigger than the prototype version which is Tiger X-1. This robot is an electric battery-based vehicle. However, the battery size and range haven't revealed yet. There are few other chances of getting solar panel or internal combustion engine as a range extender. This autonomous vehicle has a ability of manual operation, remote and autonomous functions. On the other hand, not only passengers, but the Tiger robot also has the capacity to deliver medicines and other products to the places where the standard vehicle or a human cannot enter. For an instance, to the places such as forest fire, natural disaster and so on. The Tiger robot can be sent to those horrible zones with the use of unmanned aerial vehicles and can also charge it. The robot can drive like a normal 4 wheeled robot but if in case it detects terrain, the arms which holds tires can articulate to make the vehicle walk. The robot has an option to keep the load with it while in articulation mode. According to Hyundai, the scale of the Tiger X-1 robot will be determined by motor actuators and payload capacity. The legs and chases will be manufactured with 3-D printing technology and the printing material will be carbon fiber. Elevate and TIGER vary in that Elevate can transport humans whereas TIGER is designed to transport a payload.

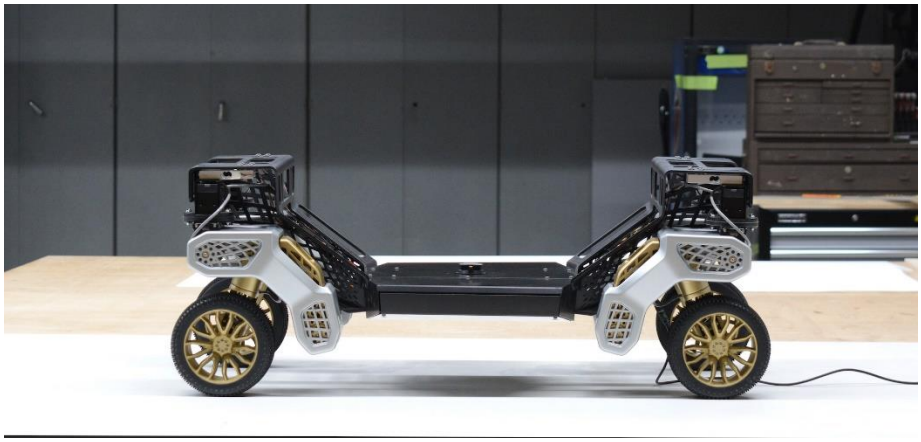


Fig 2. Hyundai Tiger X-1

Slocum G3 Glider:

The Slocum G3 Glider has several sophisticated features such as, Modular payloads, Hybrid thrusters, Real-time remote piloting, Persistent data collection and its highly reliable. This vehicle is buoyancy driven which ensures long range and remote observation for academic, military, and commercial applications. The Slocum glider is remotely operatable where it has the capability of controlling from anywhere in the world via web-based piloting tools. This technology enables minimal personal and infrastructure for operation of the glider. In emergency situations the vehicle can be configured rapidly where it can respond quicker to the environment because of its modularity. The Glider vehicle has nearly 40+ sensors with it which is primarily used to analyze wide variety of oceanic conditions. The key features of Slocum robot are, it operates with Robust software, and its Auto ballast, has speed control, low power consumption modes, ice coping capabilities. The robot is powered with 900cc hydraulic flight drive, pneumatic surfacing drive, wet payload capability, extended energy payload capability, and its hybrid thrusters produce over 1000cc additional drive. The robotic glider doesn't care about state of sea. It is capable of continuous sampling without any risk of personal or costly assets. This robot can operate all over the world even in extreme climatic conditions. The robot can be operated in coastal, offshore, under ice, and extreme conditions. It provides safe, less costly option to traditional ship-based measurements and works. The Slocum robot has given high quality oceanographic data at 1% cost while comparing with survey ship operations. Teledyne Webb Slocum Mission Control (SFMC) which is a software suite used for multiple glider deployment all over the world. The SFMC provides map in the user interface for navigation in the interactive display unit which results positioning details such as past, recent, and planned locations.

Sensors used:

• Acoustic Doppler Current Profiler (ADCP) • Acoustic Modem • Acoustic Mammal Detection • Beam Attenuation Meter • CTD Pumped or Unpumped • Echosounder • Fish Tag Detection • Hydrophones • Nitrate • Optical Backscatter Options • Optical Attenuation Options • Optical Fluorometry Options • Oxygen Options • PAR • Radiometer • Spectrophotometer for Harmful Algal Blooms (e.g., Red Tide) • Turbulence • Custom Solutions Available

Fig 3. Slocum G3 Glider

GENERAL SPECIFICATIONS

Deployment	Versatile, deployment with 1-2 people. LARS options available.
Power	Alkaline (A) / Rechargeable (Li) / Lithium (L)
Range	350-1200km/ 700-3000km/ 3000-13000km
Deployment Length	15-50 days/ 1-4 months/ 4-18 months
Depth Options	(4 to 150m) or (40 to 1000m) operating depth range*
Navigation	GPS, Pressure Sensor, Altimeter, Dead Reckoning
Communication	RF Modem, Iridium (RUDICS), ARGOS, Acoustic Modem
Horizontal Speed	Buoyancy Engine: 0.35 m/s (0.68 knot) Average, up to 0.5 m/s (1 knots) with full drive. Thruster: Up to 1 m/s (2 knots)
Mass	55 - 70kgs (dependent upon configuration)
Dimensions	Vehicle Length: 1.5 meters; Hull Diameter 22 cm

* Modular buoyancy engine dependent

Note: Endurance and range dependent on sensors and sampling frequency, energy source and communications.

The Marker unmanned ground vehicle:

The autonomous Marker UGV has the hardware specification of five roadwheels, an idler, two roller returns and a drive sprocket. The payload area is featured in the rear domain with the engine whereas the cooling system occupied the front area. The updated version of the Marker robot has a inbuilt unmanned aerial vehicle in it which can perform individual or group tasks through cluster launch module. The autonomous operations takes place via deep neural network algorithms, multispectral vision, and sophisticated data processing systems. Additionally, the vehicle has laser warning systems, laser range finders, early warning system, tracking equipment, thermal sensors, infrared cameras, target detection, and identification.

Fig 4. The Marker unmanned ground vehicle

Human Interaction with the robotic vehicle:

Accidents caused by autonomous vehicles lies under different categories such as Perception error, Decision error, and Action error. The Realtime decision making depends on the perception layer which senses data from different sensors. The development of AVs is demonstrated by complexity, reliability, suitability, maturity of sensing system. Environmental perception sensors include light detection and ranging (LIDAR) sensors, cameras, radars, ultrasonic sensors, contact sensors, and global positioning system (GPS). It should be noted that any errors in the perception of the status, location, and movement of the other road users, traffic signals, and other hazards may raise safety concerns for AVs.

HRI in CNH Tractor:

A sensing and perception package has been included which has radar, LiDAR, and high definition 3-D cameras to avoid obstacles and obstructions in the path of the vehicle. This technology not only ensures safety but also ensures trouble-free production. When an object be detected, the tractors path, visual and audio warnings will be appeared in the interface of the user (Desktop or tablet) which gives the choice of user defining response (waiting for human intervention, driving around the obstacle, driving onwards). For an instance, if a neighboring machine crosses its path and moves continuously, then the system will stop and move off once if the path is clear. In case of loss of critical machine control function, the automatic system stops for safety purpose, while if needs in emergency case, stop button is activated manually for safety reasons to stop the robotic vehicle.

Conclusion:

There are several problems and challenges being faced in autonomous vehicles. The environment perception looks the tremendous challenge for smooth, reliable, and safe and secured driving. There are many social impacts, ethical issues, planning standards, and policy are being addressed and faced. Furthermore, software challenges especially system security and integrity have also represented as a significant issue to be addressed. Subsequently, there is a need of connecting intelligent vehicles together and to the infrastructure which gives rise to the application of Big Data, a topic concerned with the processing and analysis of large datasets. However, the autonomous robotic vehicle technology can enhance traffic flow. Therefore, AVs and SO navigation is without doubt a worthy thread of research and practice for scholars and practitioners alike. The current state of "self-driving" automobiles and drones offers significant problems in terms of safety and acceptance.

Bibliography:

1. *Jun Wang, Li Zhang, Yanjun Huang, and Jian Zhao, "Journal of advanced Transportation", Hindawi, 2020, Volume 2020, Article ID 8867757.* source
2. *Juan Rosenzweig, Michael Bartl, "A Review and Analysis of Literature on Autonomous Driving", THE MAKING-OF INNOVATION, 2015.* source
3. *Saeed Asadi Bagloee, Madjid Tavana, Mohsen Asadi and Tracey Oliver, "Autonomous vehicles: challenges, opportunities, and future implications for transportation policies", 2016.* source
4. *Thomas B. Sheridan, "Human–Robot Interaction: Status and Challenges", Human Factors, sagepub, 2016.* source
5. *https://media.cnhindustrial.com/EUROPE/CNH-INDUSTRIAL-CORPORATE/cnh-industrial-brands-reveal-concept-autonomous-tractor-development--driverless-technology-to-boost-/s/a2259742-061a-412a-8a12-d307dbaedd88.*
6. *https://www.motorauthority.com/news/1131225_hyundai-tiger-x-1-concept-walking-car-could-be-a-lifesaver.*
7. *http://obsplatforms.plocan.eu/media/data_sheets/2017/11/Teledyne_Webb_Research_Brochure_G3_2017_pages1.pdf.*

Assignment – 2:

Description:

In this report, 2 models will be created and compared. From Tensorflow (Keras) framework, the following modules are imported, Cifar10 (dataset), Sequential (to build model), layers - Dense, Flatten, Conv2D, MaxPlling2D, Loss Function – Sparse categorical cross entropy, Optimizer – Adam and RMSprop, and Matplotlib for visualization. Consequently, the model has been configured with specific values for batch size, image shape, loss function, class, epoch, optimizer, and verbosity. However, the configuration for both the models remains same, Except the optimizer. The data will be splited into 4 parts such as, train data, test data, train labels, and test labels. Furthermore, the neural network architecture can only process information between the values 0 and 1 so we will be compressing the values from 0 and 255, to 0 and 1. After finishing the preprocessing stage, using sequential function, layers for the model will be added with the relu activation. There are 2 convolutional layers and 3 maxpooling layers then the data will be flatten with the flatten layers. Hence, the final 3 dense layer will be included with the SoftMax activation function in the final dense layer. The above process is common for both the models but while compilation, the optimizer differs. The first model has been trained with the optimizer of Adam while the second optimizer has been trained with the optimizer RMSprop. To conclude, both the models work with the same configuration and trained for the same epoch.

Model with Adam optimizer:

Epoch:

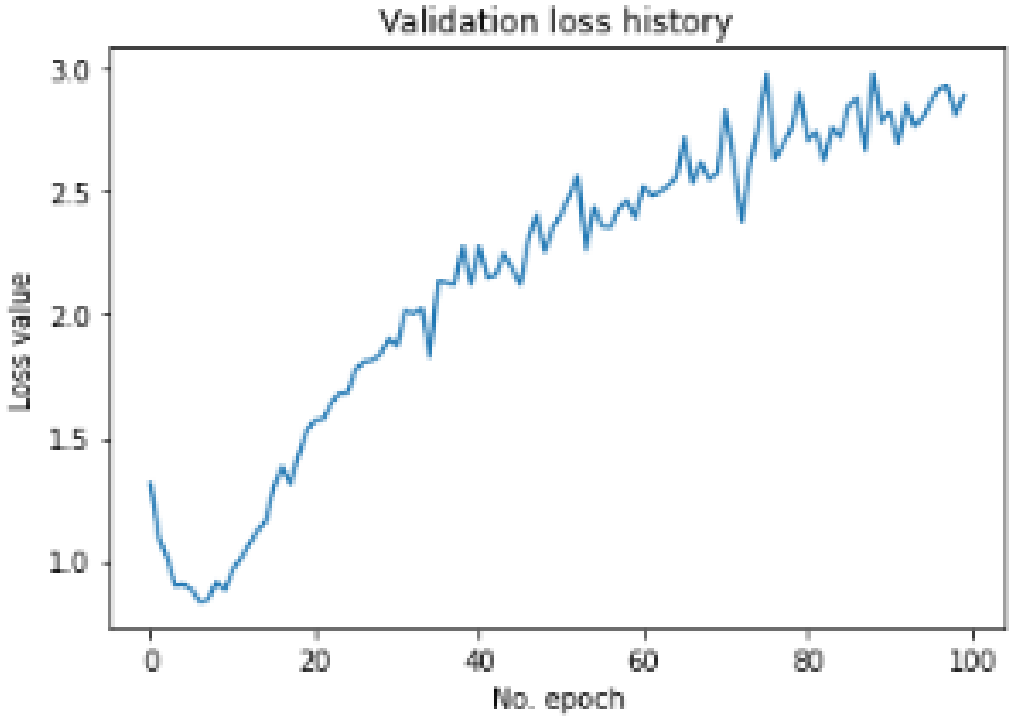
```
Epoch 91/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0481 - accuracy: 0.9856 - val_loss: 2.8266 - val_accuracy: 0.6995
Epoch 92/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0525 - accuracy: 0.9837 - val_loss: 2.6950 - val_accuracy: 0.7082
Epoch 93/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0470 - accuracy: 0.9857 - val_loss: 2.8597 - val_accuracy: 0.7104
Epoch 94/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0520 - accuracy: 0.9847 - val_loss: 2.7678 - val_accuracy: 0.7060
Epoch 95/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0468 - accuracy: 0.9856 - val_loss: 2.8004 - val_accuracy: 0.7066
Epoch 96/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0575 - accuracy: 0.9826 - val_loss: 2.8647 - val_accuracy: 0.7054
Epoch 97/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0451 - accuracy: 0.9865 - val_loss: 2.9208 - val_accuracy: 0.7146
Epoch 98/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0348 - accuracy: 0.9890 - val_loss: 2.9315 - val_accuracy: 0.7106
Epoch 99/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0561 - accuracy: 0.9838 - val_loss: 2.8137 - val_accuracy: 0.7095
Epoch 100/100
800/800 [==============================] - 9s 11ms/step - loss: 0.0443 - accuracy: 0.9876 - val_loss: 2.8900 - val_accuracy: 0.7065
```

Accuracy:

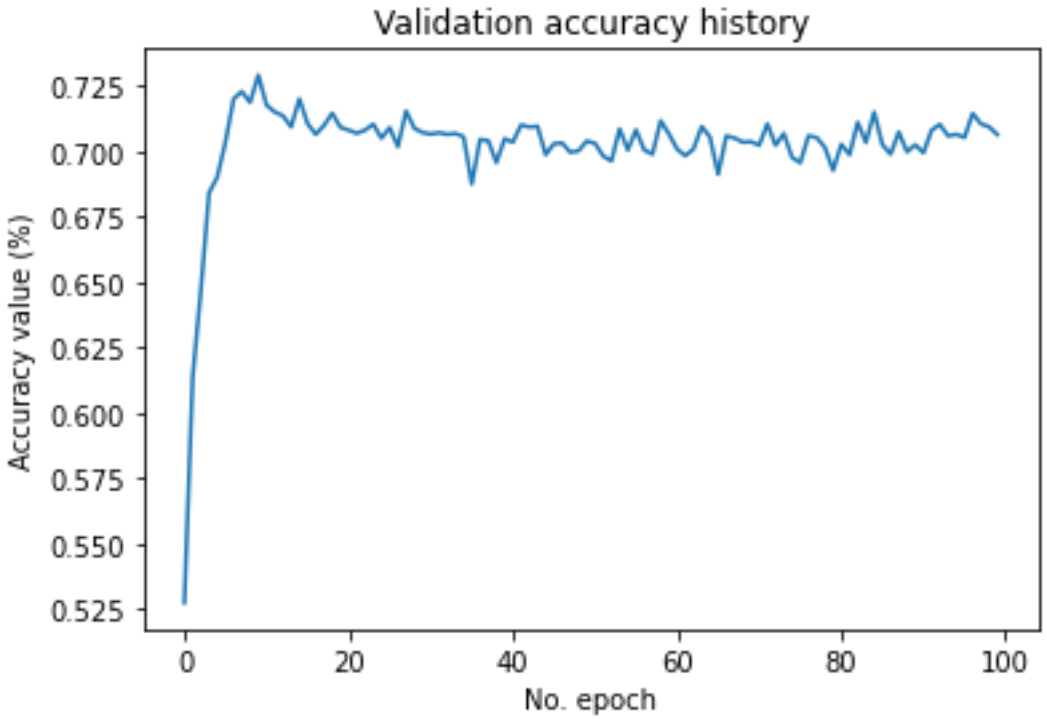
```
# Generate generalization metrics
score = model.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

```
Test loss: 3.0315611362457275 / Test accuracy: 0.6952999830245972
```

Validation Loss as Visualization:



Validation Accuracy Visualization:



Model with RMSprop:

Epoch:

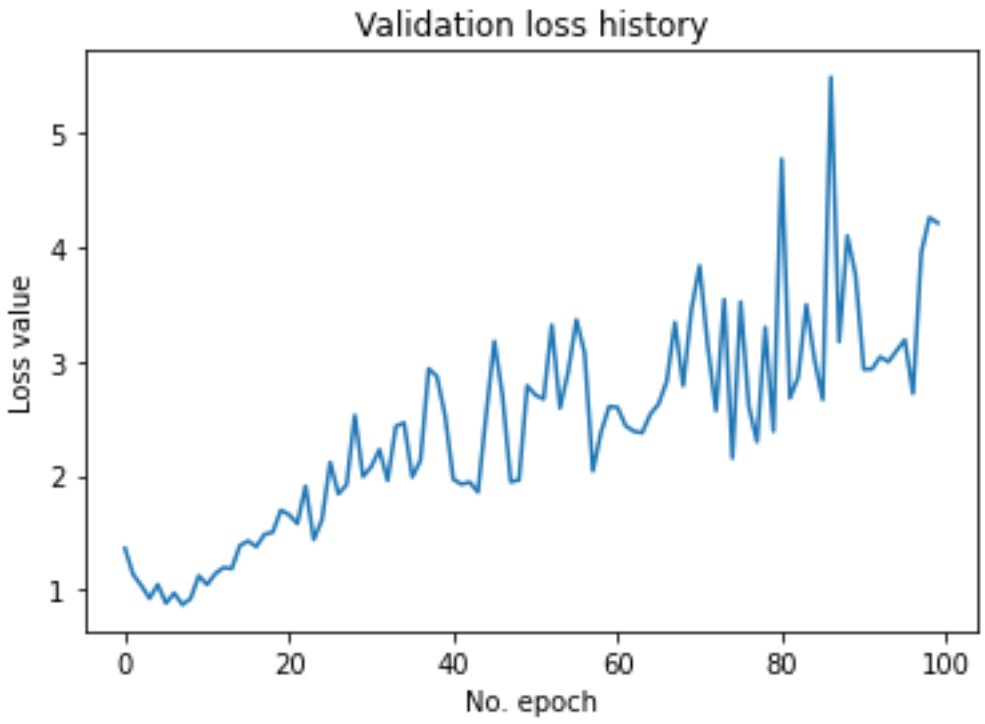
```
Epoch 91/100
800/800 [==============================] - 11s 14ms/step - loss: 0.1829 - accuracy: 0.9535 - val_loss: 2.9321 - val_accuracy: 0.7129
Epoch 92/100
800/800 [==============================] - 11s 14ms/step - loss: 0.1867 - accuracy: 0.9532 - val_loss: 2.9338 - val_accuracy: 0.7098
Epoch 93/100
800/800 [==============================] - 11s 14ms/step - loss: 0.1819 - accuracy: 0.9553 - val_loss: 3.0408 - val_accuracy: 0.7058
Epoch 94/100
800/800 [==============================] - 11s 14ms/step - loss: 0.1879 - accuracy: 0.9553 - val_loss: 2.9982 - val_accuracy: 0.6782
Epoch 95/100
800/800 [==============================] - 12s 14ms/step - loss: 0.1687 - accuracy: 0.9587 - val_loss: 3.0957 - val_accuracy: 0.6988
Epoch 96/100
800/800 [==============================] - 12s 14ms/step - loss: 0.1756 - accuracy: 0.9583 - val_loss: 3.1910 - val_accuracy: 0.6770
Epoch 97/100
800/800 [==============================] - 12s 15ms/step - loss: 0.1887 - accuracy: 0.9554 - val_loss: 2.7203 - val_accuracy: 0.6955
Epoch 98/100
800/800 [==============================] - 12s 15ms/step - loss: 0.1800 - accuracy: 0.9559 - val_loss: 3.9517 - val_accuracy: 0.7113
Epoch 99/100
800/800 [==============================] - 11s 14ms/step - loss: 0.1803 - accuracy: 0.9578 - val_loss: 4.2613 - val_accuracy: 0.6642
Epoch 100/100
800/800 [==============================] - 11s 14ms/step - loss: 0.1797 - accuracy: 0.9588 - val_loss: 4.2126 - val_accuracy: 0.7167
```

Accuracy:

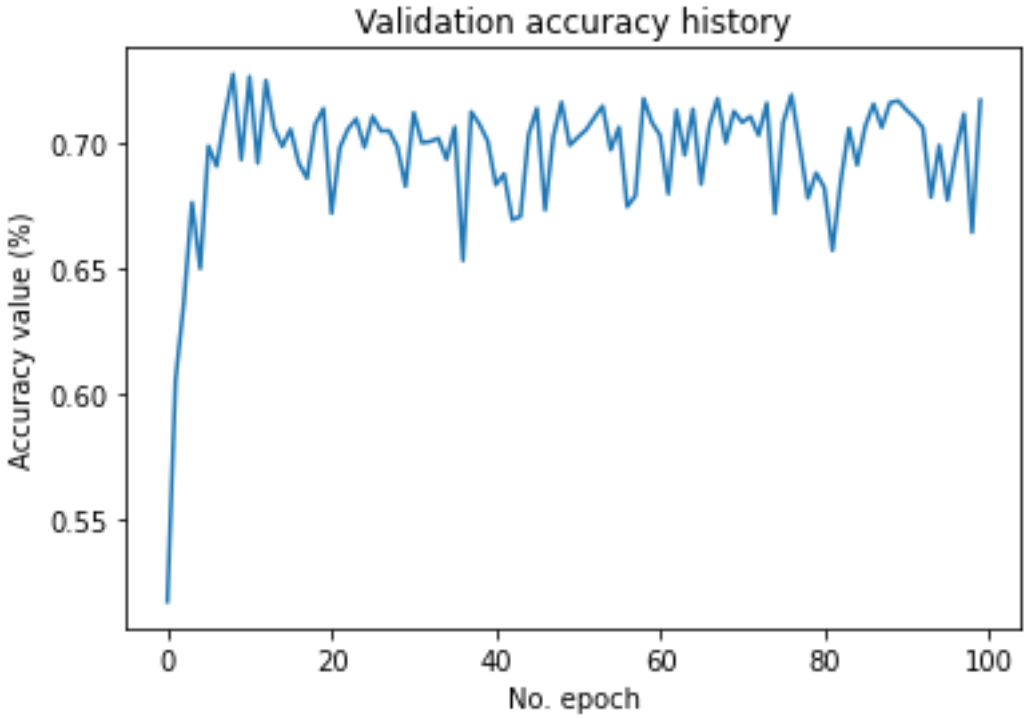
```
1 # Generate generalization metrics
2 score = model.evaluate(input_test, target_test, verbose=0)
3 print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')# Visualize history
4
```

```
Test loss: 3.9217870235443115 / Test accuracy: 0.7084000110626221
```

Validation Loss as Visualization:



Validation Accuracy Visualization:



Comparisons:

Optimizer	Loss	Accuracy
Adam	3.0315611362457275	0.6952999830245972
RMSprop	3.9217870235443115	0.7084000110626221

Model trained for 10 epochs each:

Optimizer: Adam

Epoch:

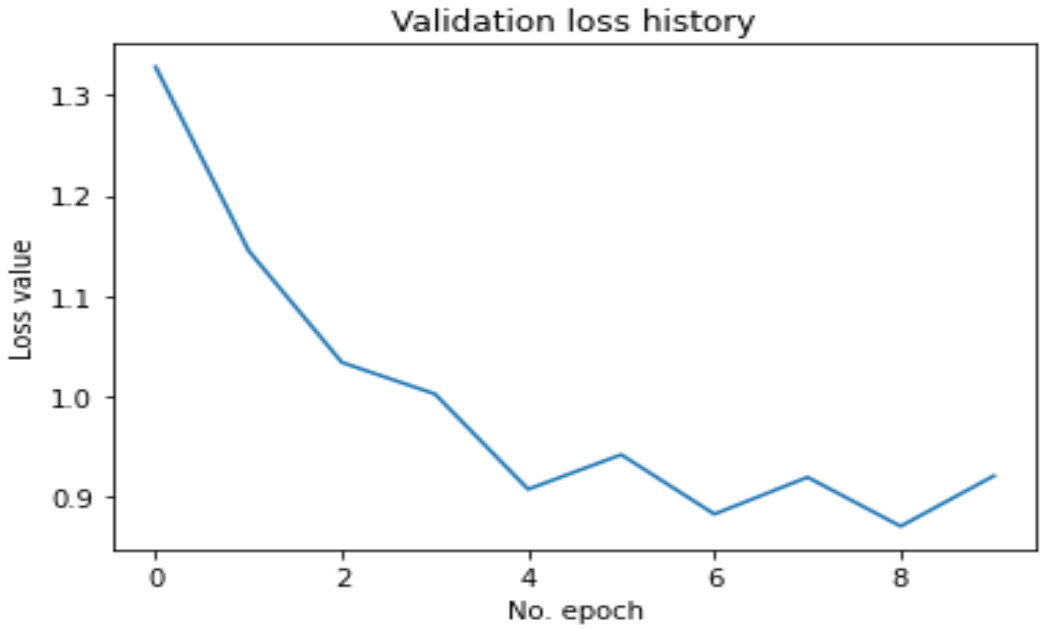
```
Epoch 1/10
800/800 [==============================] - 39s 12ms/step - loss: 1.6633 - accuracy: 0.3823 - val_loss: 1.3278 - val_accuracy: 0.5276
Epoch 2/10
800/800 [==============================] - 10s 12ms/step - loss: 1.2429 - accuracy: 0.5513 - val_loss: 1.1450 - val_accuracy: 0.5984
Epoch 3/10
800/800 [==============================] - 9s 11ms/step - loss: 1.0661 - accuracy: 0.6209 - val_loss: 1.0341 - val_accuracy: 0.6361
Epoch 4/10
800/800 [==============================] - 9s 11ms/step - loss: 0.9437 - accuracy: 0.6654 - val_loss: 1.0027 - val_accuracy: 0.6540
Epoch 5/10
800/800 [==============================] - 10s 12ms/step - loss: 0.8572 - accuracy: 0.6953 - val_loss: 0.9081 - val_accuracy: 0.6896
Epoch 6/10
800/800 [==============================] - 9s 11ms/step - loss: 0.7831 - accuracy: 0.7236 - val_loss: 0.9424 - val_accuracy: 0.6784
Epoch 7/10
800/800 [==============================] - 9s 11ms/step - loss: 0.7182 - accuracy: 0.7481 - val_loss: 0.8833 - val_accuracy: 0.6963
Epoch 8/10
800/800 [==============================] - 9s 11ms/step - loss: 0.6585 - accuracy: 0.7653 - val_loss: 0.9200 - val_accuracy: 0.6944
Epoch 9/10
800/800 [==============================] - 9s 11ms/step - loss: 0.6007 - accuracy: 0.7893 - val_loss: 0.8713 - val_accuracy: 0.7045
Epoch 10/10
800/800 [==============================] - 9s 11ms/step - loss: 0.5572 - accuracy: 0.8042 - val_loss: 0.9214 - val_accuracy: 0.6984
```

Accuracy:

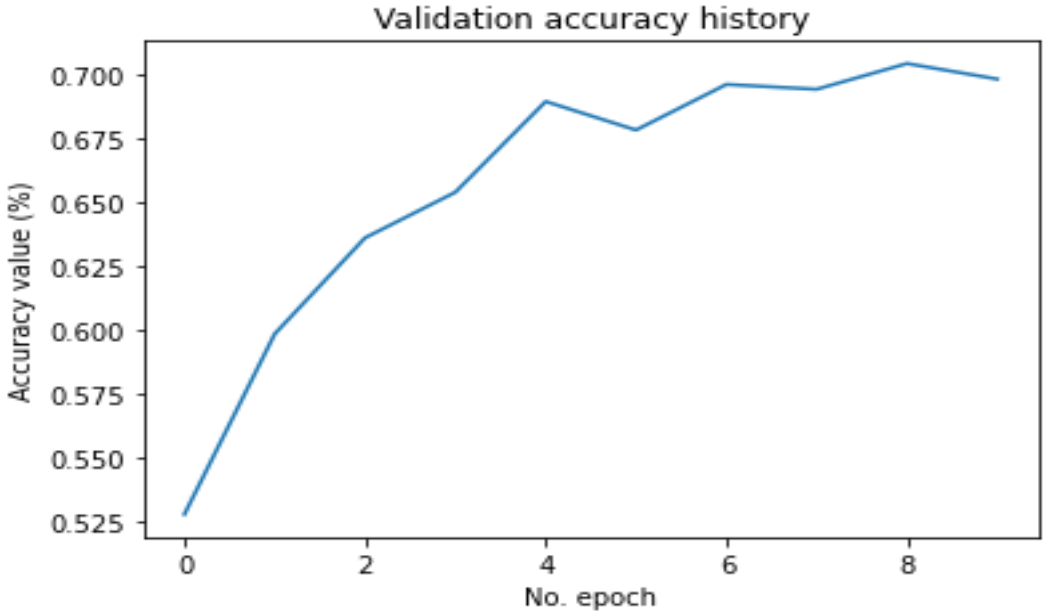
```
# Generate generalization metrics
score = model.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

```
Test loss: 0.9410311579704285 / Test accuracy: 0.6916000247001648
```

Validation Loss as Visualization:



Validation Accuracy Visualization:



Optimizer: RMSprop

Epoch:

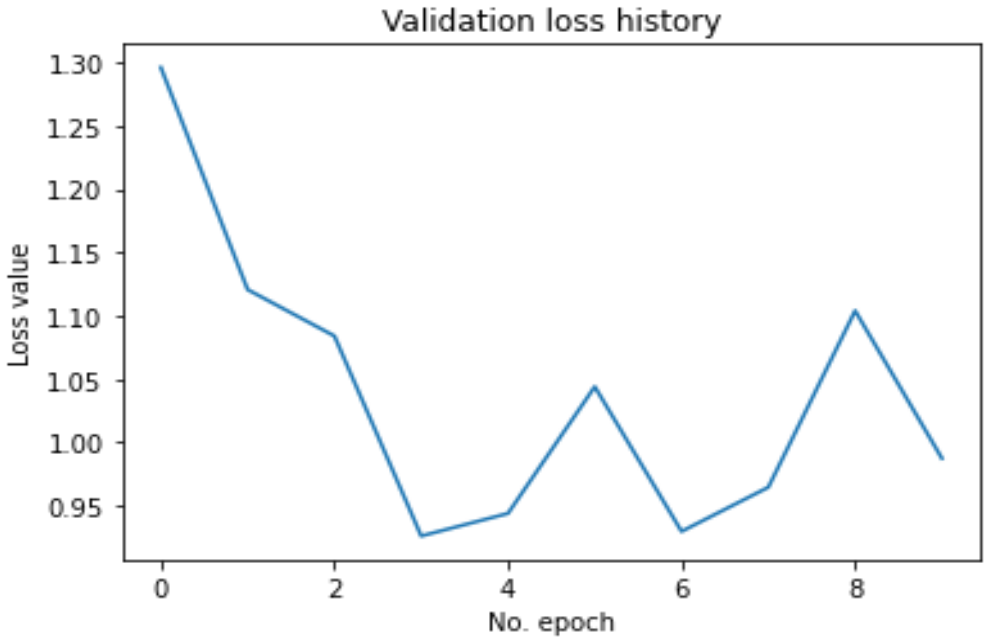
```
Epoch 1/10
800/800 [==============================] - 12s 14ms/step - loss: 1.6843 - accuracy: 0.3873 - val_loss: 1.2961 - val_accuracy: 0.5396
Epoch 2/10
800/800 [==============================] - 11s 13ms/step - loss: 1.2361 - accuracy: 0.5615 - val_loss: 1.1206 - val_accuracy: 0.6095
Epoch 3/10
800/800 [==============================] - 11s 14ms/step - loss: 1.0373 - accuracy: 0.6381 - val_loss: 1.0840 - val_accuracy: 0.6188
Epoch 4/10
800/800 [==============================] - 12s 15ms/step - loss: 0.8967 - accuracy: 0.6862 - val_loss: 0.9259 - val_accuracy: 0.6769
Epoch 5/10
800/800 [==============================] - 11s 14ms/step - loss: 0.7878 - accuracy: 0.7270 - val_loss: 0.9438 - val_accuracy: 0.6790
Epoch 6/10
800/800 [==============================] - 11s 14ms/step - loss: 0.6934 - accuracy: 0.7587 - val_loss: 1.0439 - val_accuracy: 0.6743
Epoch 7/10
800/800 [==============================] - 11s 14ms/step - loss: 0.6166 - accuracy: 0.7856 - val_loss: 0.9295 - val_accuracy: 0.6948
Epoch 8/10
800/800 [==============================] - 11s 13ms/step - loss: 0.5491 - accuracy: 0.8098 - val_loss: 0.9645 - val_accuracy: 0.6966
Epoch 9/10
800/800 [==============================] - 11s 14ms/step - loss: 0.4880 - accuracy: 0.8297 - val_loss: 1.1039 - val_accuracy: 0.7003
Epoch 10/10
800/800 [==============================] - 11s 13ms/step - loss: 0.4387 - accuracy: 0.8472 - val_loss: 0.9873 - val_accuracy: 0.7050
```

Accuracy:

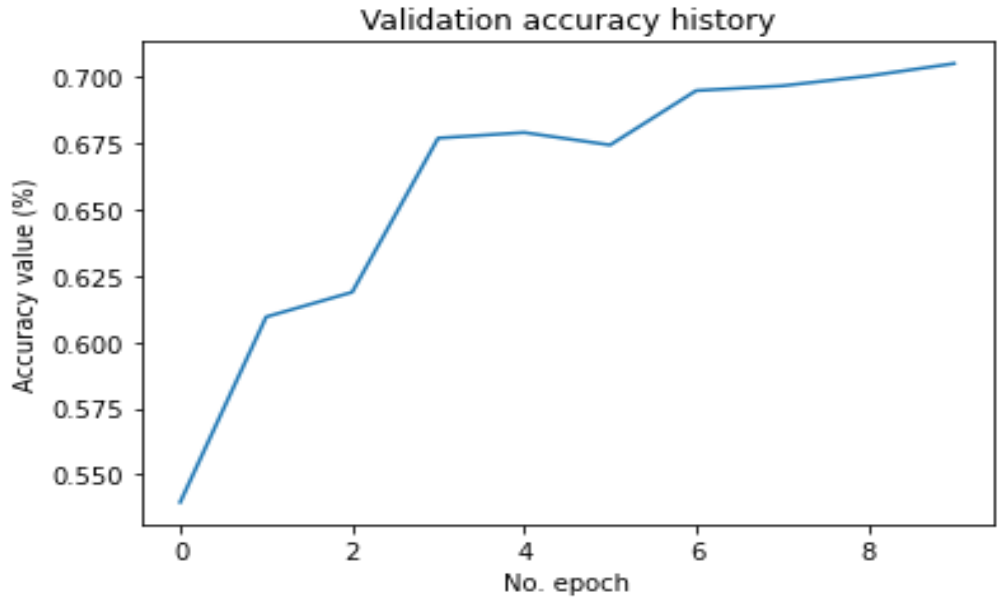
```
1 # Generate generalization metrics
2 score = model.evaluate(input_test, target_test, verbose=0)
3 print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')# Visualize history
4
```

```
Test loss: 1.0046465396881104 / Test accuracy: 0.7039999961853027
```

Validation Loss as Visualization:



Validation Accuracy Visualization:



Comparisons for 10 epochs:

Optimizer	Loss	Accuracy
Adam	0.9410311579704285	0.6916000247001648
RMSprop	1.004646539688104	0.7039999961853027

Conclusion:

As stated in the above statement, Both the models trained for 100 epochs with same configuration with different optimizers. While looking at the test accuracy, the model with RMSprop optimizer has performed well and has its peak accuracy of 70%. However, the loss is comparatively high which holds 3.92. but the model with Adam has minimal loss off 3.03 which has lower accuracy of 69%.

On the other hand, while training for 10 epochs with same configuration, the RMSprop performs slightly better for test accuracy than Adam but, the loss is high in RMSprop which is 1.004 but the Adam is 0.9410.

Adam takes longer to change direction and even longer to return to the minimum. RMSprop with momentum, on the other hand, travels much further before changing direction.

SOURCE CODE FOR ADAM OPTIMIZER WITH 100 EPOCHS:

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.optimizers import Adam, RMSprop
import matplotlib.pyplot as plt

# Model configuration
batch_size = 50
img_width, img_height, img_num_channels = 32, 32, 3
loss_function = sparse_categorical_crossentropy
no_classes = 10
no_epochs = 100
optimizer = Adam()
validation_split = 0.2
verbosity = 1

# Load CIFAR-10 data
(input_train, target_train), (input_test, target_test) = cifar10.load_data()

# Determine shape of the data
input_shape = (img_width, img_height, img_num_channels)

# Parse numbers as floats
input_train = input_train.astype('float32')
input_test = input_test.astype('float32')

# Normalize data
input_train = input_train / 255
input_test = input_test / 255
```

```
# Create the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(no_classes, activation='softmax'))

# Compile the model
model.compile(loss=loss_function,
              optimizer=optimizer,
              metrics=['accuracy'])

# Fit data to model
history = model.fit(input_train, target_train,
            batch_size=batch_size,
            epochs=no_epochs,
            verbose=verbosity,
            validation_split=validation_split)

# Generate generalization metrics
score = model.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt.title('Validation loss history')
```

```
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.show()

# Plot history: Accuracy
plt.plot(history.history['val_accuracy'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```

SOURCE CODE FOR 100 EPOCHS WITH RMSPROP:

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.optimizers import Adam, RMSprop
import matplotlib.pyplot as plt

# Model configuration
batch_size = 50
img_width, img_height, img_num_channels = 32, 32, 3
loss_function = sparse_categorical_crossentropy
no_classes = 10
no_epochs = 100
optimizer = RMSprop()
validation_split = 0.2
verbosity = 1

# Load CIFAR-10 data
(input_train, target_train), (input_test, target_test) = cifar10.load_data()

# Determine shape of the data
input_shape = (img_width, img_height, img_num_channels)

# Parse numbers as floats
input_train = input_train.astype('float32')
input_test = input_test.astype('float32')

# Normalize data
input_train = input_train / 255
input_test = input_test / 255
```

```
# Create the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(no_classes, activation='softmax'))

# Compile the model
model.compile(loss=loss_function,
              optimizer=optimizer,
              metrics=['accuracy'])

# Fit data to model
history = model.fit(input_train, target_train,
            batch_size=batch_size,
            epochs=no_epochs,
            verbose=verbosity,
            validation_split=validation_split)

# Generate generalization metrics
score = model.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt.title('Validation loss history')
```

```
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.show()

# Plot history: Accuracy
plt.plot(history.history['val_accuracy'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```

SOURCE CODE FOR 10 EPOCHS WITH ADAM OPTIMIZER:

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.optimizers import Adam, RMSprop
import matplotlib.pyplot as plt

# Model configuration
batch_size = 50
img_width, img_height, img_num_channels = 32, 32, 3
loss_function = sparse_categorical_crossentropy
no_classes = 10
no_epochs = 10
optimizer = Adam()
validation_split = 0.2
verbosity = 1

# Load CIFAR-10 data
(input_train, target_train), (input_test, target_test) = cifar10.load_data()

# Determine shape of the data
input_shape = (img_width, img_height, img_num_channels)

# Parse numbers as floats
input_train = input_train.astype('float32')
input_test = input_test.astype('float32')

# Normalize data
input_train = input_train / 255
input_test = input_test / 255
```

```
# Create the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(no_classes, activation='softmax'))

# Compile the model
model.compile(loss=loss_function,
              optimizer=optimizer,
              metrics=['accuracy'])

# Fit data to model
history = model.fit(input_train, target_train,
            batch_size=batch_size,
            epochs=no_epochs,
            verbose=verbosity,
            validation_split=validation_split)

# Generate generalization metrics
score = model.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt.title('Validation loss history')
```

```
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.show()

# Plot history: Accuracy
plt.plot(history.history['val_accuracy'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```

SOURCE CODE FOR 10 EPOCHS WITH RMSPROP:

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.optimizers import Adam, RMSprop
import matplotlib.pyplot as plt

# Model configuration
batch_size = 50
img_width, img_height, img_num_channels = 32, 32, 3
loss_function = sparse_categorical_crossentropy
no_classes = 10
no_epochs = 10
optimizer = RMSprop()
validation_split = 0.2
verbosity = 1

# Load CIFAR-10 data
(input_train, target_train), (input_test, target_test) = cifar10.load_data()

# Determine shape of the data
input_shape = (img_width, img_height, img_num_channels)

# Parse numbers as floats
input_train = input_train.astype('float32')
input_test = input_test.astype('float32')

# Normalize data
input_train = input_train / 255
input_test = input_test / 255
```

```
# Create the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(no_classes, activation='softmax'))

# Compile the model
model.compile(loss=loss_function,
              optimizer=optimizer,
              metrics=['accuracy'])

# Fit data to model
history = model.fit(input_train, target_train,
            batch_size=batch_size,
            epochs=no_epochs,
            verbose=verbosity,
            validation_split=validation_split)

# Generate generalization metrics
score = model.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'])
plt.title('Validation loss history')
```

```
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.show()

# Plot history: Accuracy
plt.plot(history.history['val_accuracy'])
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.show()
```