Industrial Interaction Report on

WEAPON CLASSIFICATION USING DEEP LEARNING ALGORITHM -CNN

DEAL-DRDO

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING(Artificial Intelligence and Data Science)

Submitted by:

Ishita Bhatt **2017647**

Department of Computer Science and Engineering
Graphic Era (Deemed to be University)
Dehradun, Uttarakhand
2023-24

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the Industrial Interaction report entitled **"WEAPON CLASSIFICATION USING DEEP LEARNING ALGORITHM -CNN"** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering **(Artificial Intelligence and Data Science)** in the Department of Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun shall be carried out by the undersigned under the supervision of f Mr Mridul Goel, Scientist 'C', DEAL DRDO, Dehradun. **(AEWC&S,** DEAL DRDO**)**.

Ishita Bhatt
2017647
CSE (AI and DS)

Table of Contents

Chapter 1

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Shri Mangal Lal Chand, Director of DEAL DRDO, Dehradun for giving me the permission to carry out the summer training of 6 weeks during the period July-August 2023 in DEAL DRDO, Dehradun.

I am highly thankful to Mr. Mridul Goel, Scientist 'C' for the support and guidance throughout my study, without which this study would not have been successful. I owe thanks to TIC DEAL DRDO for providing standard texts about the subject.

Last but not the least, I express my heartiest thanks with a deep sense of gratitude and respect to all those who provides me immense help, cooperative environment and guidance during my training period.

Chapter 2

ABOUT THE COMPANY

DEAL and DRDO

DRDO was formed in 1958 from the amalgamation of the then already functioning Technical Development Establishment (TDEs) of the Indian Army and the Directorate of Technical Development & Production (DTDP) with the Defence Science Organization (DSO), DRDO was then a small organization with 10 establishments or laboratories. Today, DRDO is a network of 51 laboratories which are deeply engaged in developing defense technologies covering various disciplines, like aeronautics, armaments, electronics, combat vehicles. engineering systems, instrumentation, missiles, advanced computing and simulation, special materials, naval systems, life sciences, training, information systems and agriculture. Presently, over 5000 scientists and about 25,000 other scientific, technical and supporting personnel back the Organization.

Defence Electronics Applications Laboratory (DEAL) came into existence on 23RD February 1965 as the then Himalayan Radio Propagation Unit (HRPU) at Mussoorie. During the last 33 years, DEAL has grown into a major system laboratory of DRDO. Besides radio propagation studies, DEAL has entered into new areas of communications.

Chapter 3

INTRODUCTION AND PROBLEM STATEMENT

Weapons pose a significant threat to public safety and security. The ability to automatically identify and classify weapons in real-time is crucial for law enforcement agencies, security organizations, and public safety initiatives. Traditional methods of weapon detection heavily rely on manual inspection, which can be time-consuming, error-prone, and potentially dangerous. To address these challenges, computer vision and machine learning techniques can be employed to develop an automated weapon classification system.

Gun violence in a lot of areas remains stubbornly high. Therefore, limiting gun violence has been a priority in recent years. In this report, we will be implementing the use of deep learning algorithms to classify any firearms or weapons, to improve response times and reduce potential harm. The proposed system in this report is a weapon classification model that makes use of tensorflow keras. The project will involve the use of various resources for implementation, such as : python, OpenCV ,GoogleColab, Numpy, Pandas, Matplot, Glob, etc.

The successful development of a weapon classification system can significantly enhance public safety and security. It can be deployed in various contexts, including airports, public events, transportation hubs, and high-security facilities, to detect and prevent potential threats, ultimately saving lives and reducing the risk of weapon-related incidents.

Chapter 4

SIGNIFICANCE OF THE PROJECT

An automatic weapon detection system can provide the early detection of potentially violent situations that is of paramount importance for citizens security. One way to prevent these situations is by detecting the presence of dangerous objects such as handguns and knives in surveillance photos.

The role of deep learning in improving task performance in security controls systems is considered indisputable . Deep learning is a sub-field of machine learning . It uses many layers of non-linear processing units for deep learning and feature extraction and conversion . The deep learning structure is based on the learning of more than one feature level of data.

In object recognition applications, the most used deep learning algorithm is the CNN algorithm. This algorithm showed outstanding success in the ImageNet Large-Scale Visual Recognition Competition held in 2012 and has been used in the development of new models in many areas since then.

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

The successful development of a weapon classification system can significantly enhance public safety and security. It can be deployed in various contexts, including airports, public events, transportation hubs, and high-security facilities, to detect and prevent potential threats, ultimately saving lives and reducing the risk of weapon-related incidents.

Chapter 5

WORK RESPONSIBILITY

1. **Literature Review:** Conduct a thorough literature review to understand the latest advancements and best practices in convolutional neural networks.

2. **Data Preparation:** Assist in collecting, cleaning, and preprocessing data for training and testing the CNN model. Ensure data quality and consistency.

3. **Model Development:** Work on implementing and fine-tuning CNN architectures using deep learning frameworks like TensorFlow or PyTorch. Collaborate with team members to design and experiment with different model architectures.

4. **Hyperparameter Tuning:** Explore and optimize hyperparameters to improve the performance of the CNN model. Conduct experiments to find the best combination of hyperparameters.

5. **Training and Evaluation:** Train CNN models on relevant datasets and evaluate their performance using appropriate metrics. Analyse model results and provide insights into strengths and weaknesses.

Chapter 6

WORK DETAILS

1. **Dataset collection** : This dataset is a collection of images from 9 different types of weapons. Previously, there have been datasets that has only one class Weapon or Gun. This dataset consists of 9 classes as of now: Automatic Rifle, Bazooka, Handgun, Knife, Grenade Launcher, Shotgun, SMG, Sniper, Sword. This dataset was created with the help of simple_image_download library in Python, which downloads images from internet. 100 images from each class were collected. After inspection invalid images were discarded, leaving us with a total of 714 images for all 9 classes.

   Number of classes: 9

   Label Annotation: YOLO format (.txt)

   Metadata: metadata.csv provides information about the dataset and train-val split

   Weapon Class Map: {'Automatic Rifle': 0, 'Bazooka': 1, 'Grenade Launcher': 2, 'Handgun': 3, 'Knife': 4, 'Shotgun': 5, 'SMG': 6, 'Sniper': 7, 'Sword': 8}

   Difficulty: This is a beginner-friendly dataset on multi-class classification. The splits are given in the dataset folder itself with metadata, so anyone can use this data to run models and produce results.

   https://www.kaggle.com/datasets/snehilsanyal/weapon-detection-test

2. Firstly, we used GoogleColab for the working of model and to determine the accuracy of the model.

   Take input data after mounting the Google drive.

   We have used the following libraries:

   ```
   import matplotlib.pyplot as plt
   import cv2
   from PIL import Image
   import os
   import numpy as np
   from numpy import asarray
   import glob
   ```

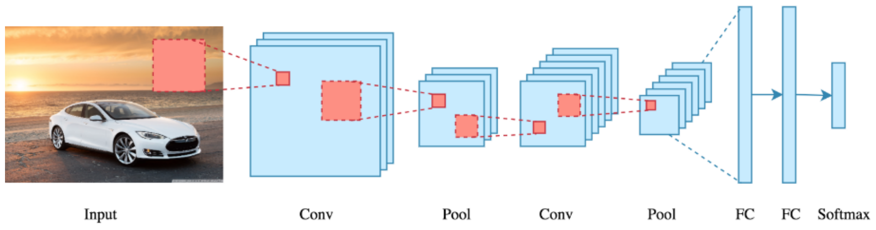
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten
from keras.utils import np_utils
from keras.utils import to_categorical
```

3. Imported data files using the Glob .

- Reshaped all the images using the resize function to (400,400).
- Defining the input shape and put training images to train data after reshaping them.
- Put training images to train data after reshaping them.
- Changed the data type of all images to float.
- Normalized the images by /255.
- One Hot encoding has been done to train and test labels.
- Created the model and applied CNN.

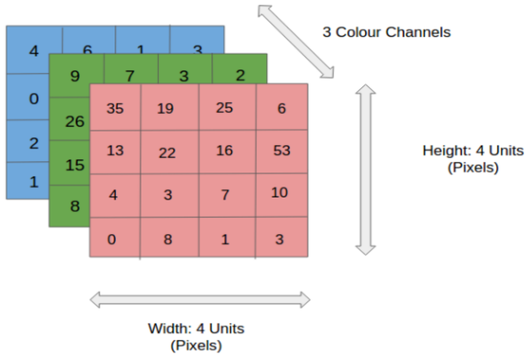
4. A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

   The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes. The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image.



How does it work?

Before we go to the working of CNN's let's cover the basics such as what is an image and how is it represented. An RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane. Take a look at this image to understand more.

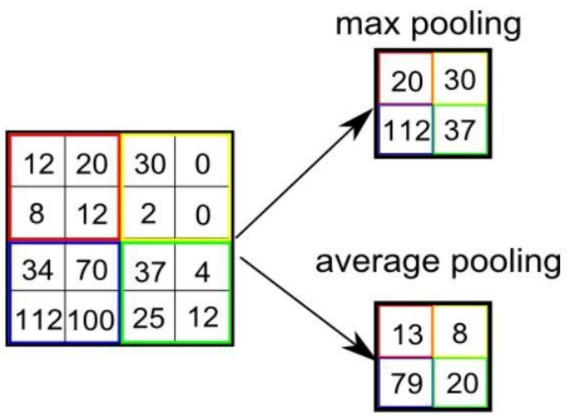


- **What Is a Pooling Layer?**

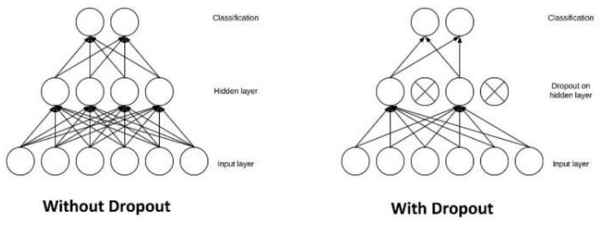
  Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data by reducing the dimensions. There are two types of pooling average pooling and max pooling.

  In this project, we have used max pooling.

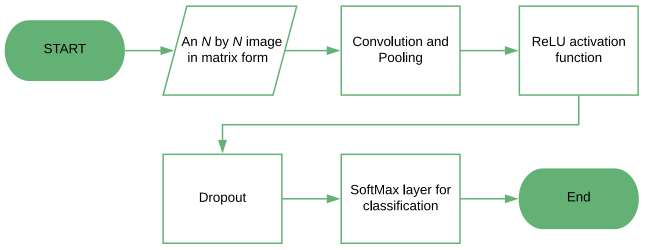
  So what we do in Max Pooling is we find the maximum value of a pixel from a portion of the image covered by the kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.



Dropout layer : The Dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others. We can apply a Dropout layer to the input vector, in which case it nullifies some of its features; but we can also apply it to a hidden layer, in which case it nullifies some hidden neurons.



A CNN With *ReLU* and a Dropout Layer


START
An *N* by *N* image in matrix form
Convolution and Pooling
ReLU activation function
Dropout
SoftMax layer for classification
End

ReLU ACTIVATION FUNCTION:

This function has two major advantages over sigmoidal functions such as \sigma{(x)} or \tanh{(x)}.

1. ReLU is very simple to calculate, as it involves only a comparison between its input and the value 0.

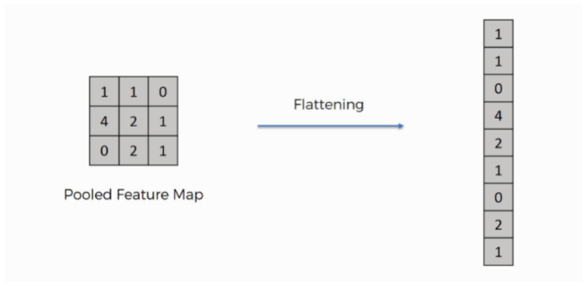
2.It also has a derivative of either 0 or 1, depending on whether its input is respectively negative or not.

The latter, in particular, has important implications for backpropagation during training. It means in fact that calculating the gradient of a neuron is computationally inexpensive.

ReLU is simple to compute and has a predictable gradient for the backpropagation of the error.

Flattening in CNN:

We're supposed to have a pooled feature map by now. As the name of this step implies, we are literally going to flatten our pooled feature map into a column like in the image below.



The reason we do this is that we're going to need to insert this data into an artificial neural network later on.

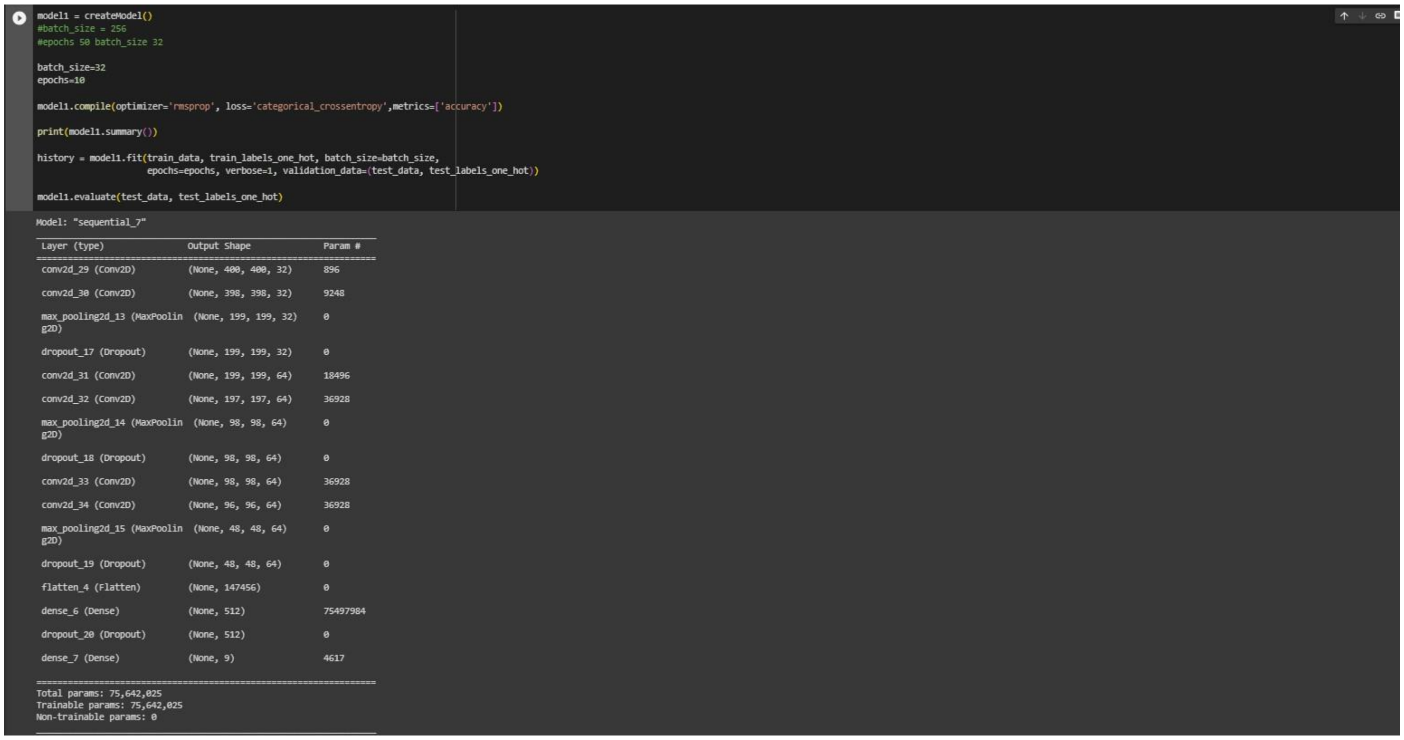
Dense Layer is simple layer of neurons in which each neuron receives input from all the neurons of previous layer, thus called as dense. Dense Layer is used to classify image based on output from convolutional layers.

Softmax : It is an activation function that scales numbers/logits into probabilities. The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.

Chapter 8

RESULTS AND OBSERVATIONS

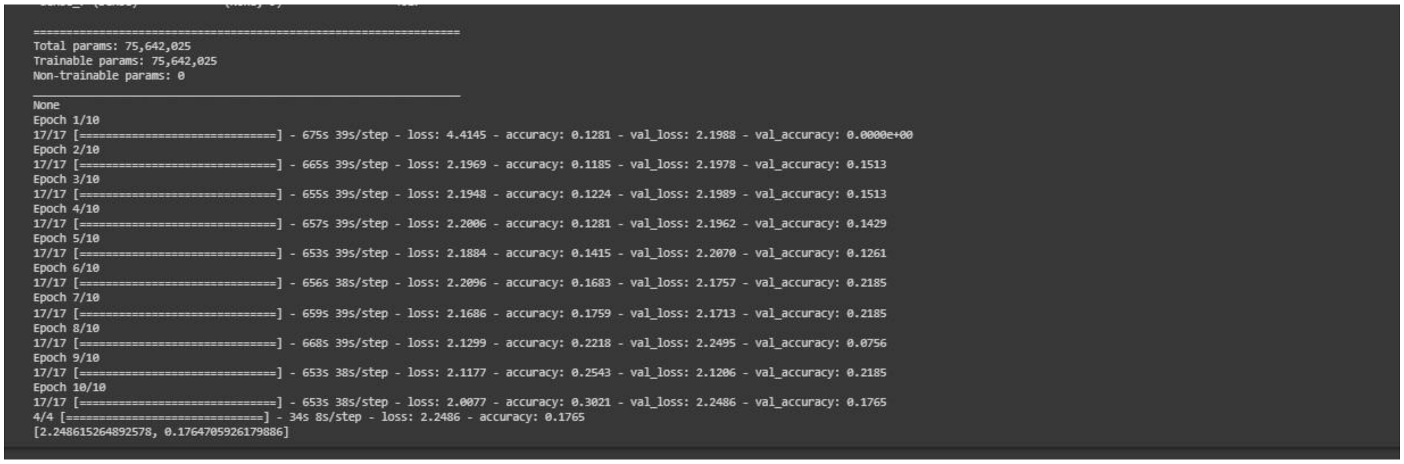
Here, we are creating and fitting the model.



```
model1 = createModel()
#batch_size = 256
#epochs 50 batch_size 32

batch_size=32
epochs=10

model1.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['accuracy'])

print(model1.summary())

history = model1.fit(train_data, train_labels_one_hot, batch_size=batch_size,
                    epochs=epochs, verbose=1, validation_data=(test_data, test_labels_one_hot))

model1.evaluate(test_data, test_labels_one_hot)
```

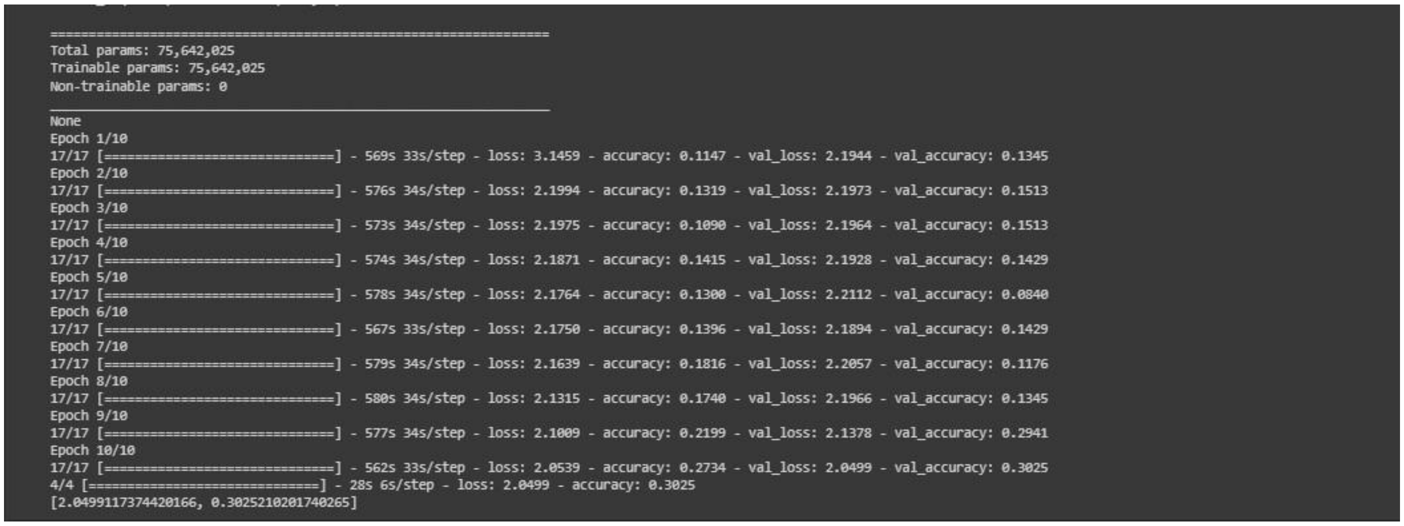
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_29 (Conv2D)          (None, 400, 400, 32)      896

 conv2d_30 (Conv2D)          (None, 398, 398, 32)      9248

 max_pooling2d_13 (MaxPoolin  (None, 199, 199, 32)     0
 g2D)

 dropout_17 (Dropout)        (None, 199, 199, 32)      0

 conv2d_31 (Conv2D)          (None, 199, 199, 64)      18496

 conv2d_32 (Conv2D)          (None, 197, 197, 64)      36928

 max_pooling2d_14 (MaxPoolin  (None, 98, 98, 64)       0
 g2D)

 dropout_18 (Dropout)        (None, 98, 98, 64)        0

 conv2d_33 (Conv2D)          (None, 98, 98, 64)        36928

 conv2d_34 (Conv2D)          (None, 96, 96, 64)        36928

 max_pooling2d_15 (MaxPoolin  (None, 48, 48, 64)       0
 g2D)

 dropout_19 (Dropout)        (None, 48, 48, 64)        0

 flatten_4 (Flatten)         (None, 147456)            0

 dense_6 (Dense)             (None, 512)               75497984

 dropout_20 (Dropout)        (None, 512)               0

 dense_7 (Dense)             (None, 9)                 4617

=================================================================
Total params: 75,642,025
Trainable params: 75,642,025
Non-trainable params: 0
_________________________________________________________________
```

- Firstly, when we start fitting the model we have imported input data without reshaping. Hence, the accuracy after 10 epochs, comes out to be 17% only.



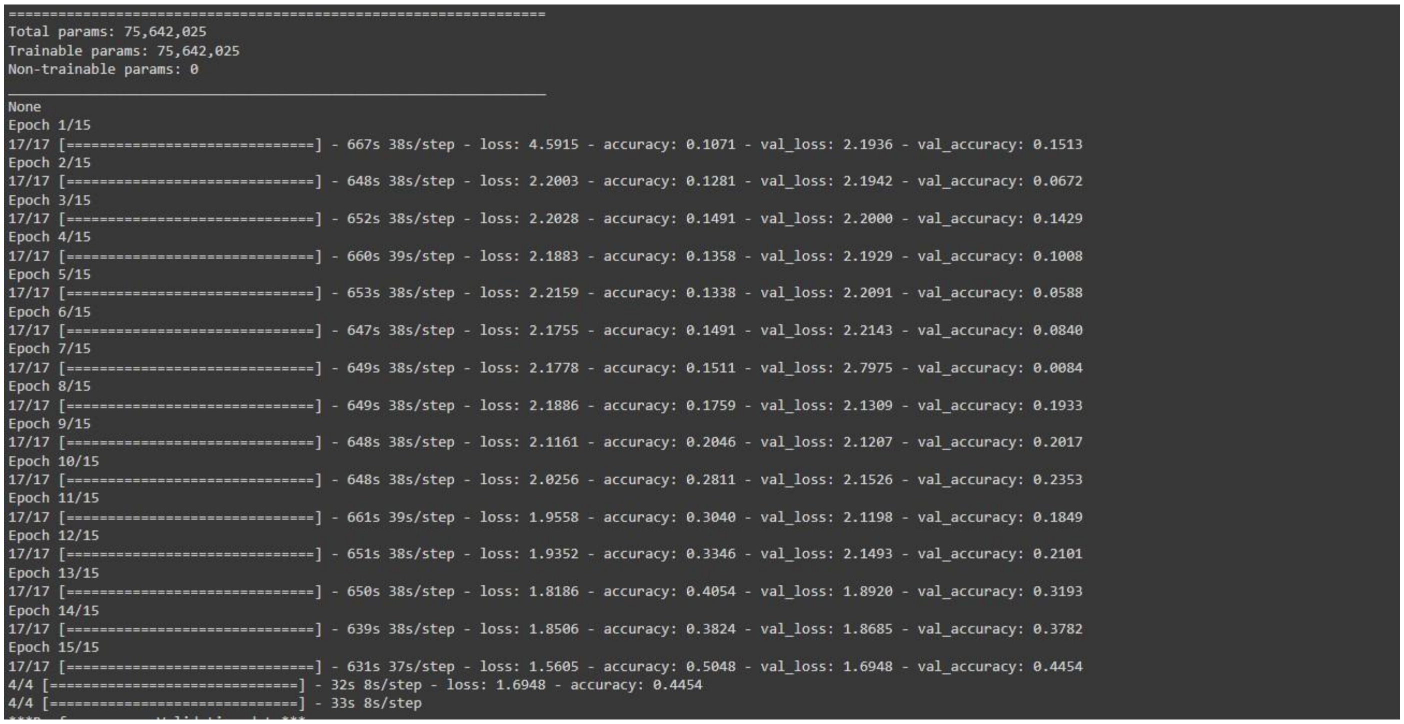
```
=================================================================
Total params: 75,642,025
Trainable params: 75,642,025
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
17/17 [==============================] - 675s 39s/step - loss: 4.4145 - accuracy: 0.1281 - val_loss: 2.1988 - val_accuracy: 0.0000e+00
Epoch 2/10
17/17 [==============================] - 665s 39s/step - loss: 2.1969 - accuracy: 0.1185 - val_loss: 2.1978 - val_accuracy: 0.1513
Epoch 3/10
17/17 [==============================] - 655s 39s/step - loss: 2.1948 - accuracy: 0.1224 - val_loss: 2.1989 - val_accuracy: 0.1513
Epoch 4/10
17/17 [==============================] - 657s 39s/step - loss: 2.2006 - accuracy: 0.1281 - val_loss: 2.1962 - val_accuracy: 0.1429
Epoch 5/10
17/17 [==============================] - 653s 39s/step - loss: 2.1884 - accuracy: 0.1415 - val_loss: 2.2070 - val_accuracy: 0.1261
Epoch 6/10
17/17 [==============================] - 656s 38s/step - loss: 2.2096 - accuracy: 0.1683 - val_loss: 2.1757 - val_accuracy: 0.2185
Epoch 7/10
17/17 [==============================] - 659s 39s/step - loss: 2.1686 - accuracy: 0.1759 - val_loss: 2.1713 - val_accuracy: 0.2185
Epoch 8/10
17/17 [==============================] - 668s 39s/step - loss: 2.1299 - accuracy: 0.2218 - val_loss: 2.2495 - val_accuracy: 0.0756
Epoch 9/10
17/17 [==============================] - 653s 38s/step - loss: 2.1177 - accuracy: 0.2543 - val_loss: 2.1206 - val_accuracy: 0.2185
Epoch 10/10
17/17 [==============================] - 653s 38s/step - loss: 2.0077 - accuracy: 0.3021 - val_loss: 2.2486 - val_accuracy: 0.1765
4/4 [==============================] - 34s 8s/step - loss: 2.2486 - accuracy: 0.1765
[2.248615264892578, 0.1764705926179886]
```

- Now when we reshaped the data, the accuracy increased a bit and becomes 30% approximately.



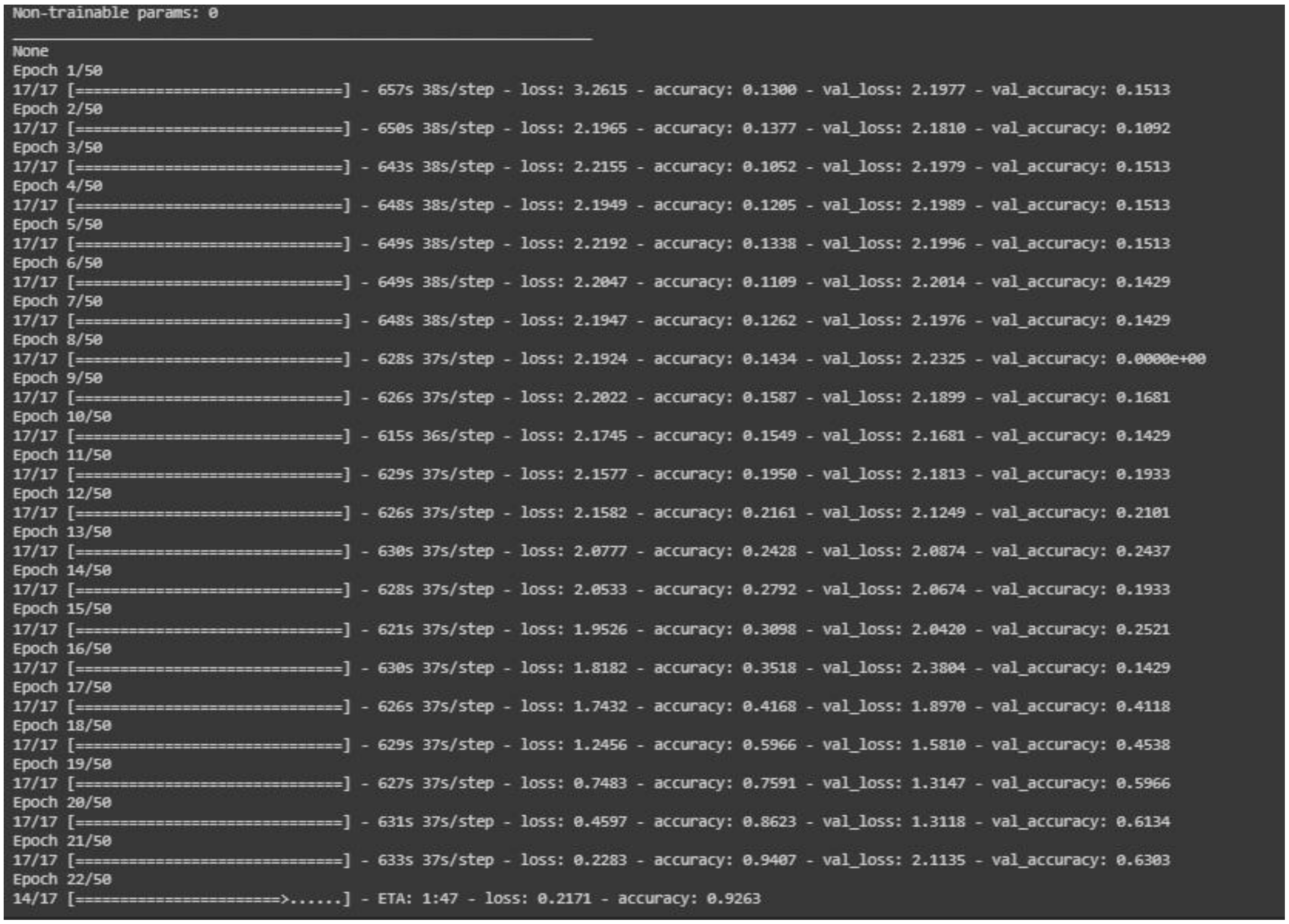
```
=================================================================
Total params: 75,642,025
Trainable params: 75,642,025
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
17/17 [==============================] - 569s 33s/step - loss: 3.1459 - accuracy: 0.1147 - val_loss: 2.1944 - val_accuracy: 0.1345
Epoch 2/10
17/17 [==============================] - 576s 34s/step - loss: 2.1994 - accuracy: 0.1319 - val_loss: 2.1973 - val_accuracy: 0.1513
Epoch 3/10
17/17 [==============================] - 573s 34s/step - loss: 2.1975 - accuracy: 0.1090 - val_loss: 2.1964 - val_accuracy: 0.1513
Epoch 4/10
17/17 [==============================] - 574s 34s/step - loss: 2.1871 - accuracy: 0.1415 - val_loss: 2.1928 - val_accuracy: 0.1429
Epoch 5/10
17/17 [==============================] - 578s 34s/step - loss: 2.1764 - accuracy: 0.1300 - val_loss: 2.2112 - val_accuracy: 0.0840
Epoch 6/10
17/17 [==============================] - 567s 33s/step - loss: 2.1750 - accuracy: 0.1396 - val_loss: 2.1894 - val_accuracy: 0.1429
Epoch 7/10
17/17 [==============================] - 579s 34s/step - loss: 2.1639 - accuracy: 0.1816 - val_loss: 2.2057 - val_accuracy: 0.1176
Epoch 8/10
17/17 [==============================] - 580s 34s/step - loss: 2.1315 - accuracy: 0.1740 - val_loss: 2.1966 - val_accuracy: 0.1345
Epoch 9/10
17/17 [==============================] - 577s 34s/step - loss: 2.1009 - accuracy: 0.2199 - val_loss: 2.1378 - val_accuracy: 0.2941
Epoch 10/10
17/17 [==============================] - 562s 33s/step - loss: 2.0539 - accuracy: 0.2734 - val_loss: 2.0499 - val_accuracy: 0.3025
4/4 [==============================] - 28s 6s/step - loss: 2.0499 - accuracy: 0.3025
[2.0499117374420166, 0.3025210201740265]
```

- Now when the epochs were increased from 10 to 15, the accuracy increased and becomes 44% (approx.)



```
=================================================================
Total params: 75,642,025
Trainable params: 75,642,025
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/15
17/17 [==============================] - 667s 38s/step - loss: 4.5915 - accuracy: 0.1071 - val_loss: 2.1936 - val_accuracy: 0.1513
Epoch 2/15
17/17 [==============================] - 648s 38s/step - loss: 2.2003 - accuracy: 0.1281 - val_loss: 2.1942 - val_accuracy: 0.0672
Epoch 3/15
17/17 [==============================] - 652s 38s/step - loss: 2.2028 - accuracy: 0.1491 - val_loss: 2.2000 - val_accuracy: 0.1429
Epoch 4/15
17/17 [==============================] - 660s 39s/step - loss: 2.1883 - accuracy: 0.1358 - val_loss: 2.1929 - val_accuracy: 0.1008
Epoch 5/15
17/17 [==============================] - 653s 38s/step - loss: 2.2159 - accuracy: 0.1338 - val_loss: 2.2091 - val_accuracy: 0.0588
Epoch 6/15
17/17 [==============================] - 647s 38s/step - loss: 2.1755 - accuracy: 0.1491 - val_loss: 2.2143 - val_accuracy: 0.0840
Epoch 7/15
17/17 [==============================] - 649s 38s/step - loss: 2.1778 - accuracy: 0.1511 - val_loss: 2.7975 - val_accuracy: 0.0084
Epoch 8/15
17/17 [==============================] - 649s 38s/step - loss: 2.1886 - accuracy: 0.1759 - val_loss: 2.1309 - val_accuracy: 0.1933
Epoch 9/15
17/17 [==============================] - 648s 38s/step - loss: 2.1161 - accuracy: 0.2046 - val_loss: 2.1207 - val_accuracy: 0.2017
Epoch 10/15
17/17 [==============================] - 648s 38s/step - loss: 2.0256 - accuracy: 0.2811 - val_loss: 2.1526 - val_accuracy: 0.2353
Epoch 11/15
17/17 [==============================] - 661s 39s/step - loss: 1.9558 - accuracy: 0.3040 - val_loss: 2.1198 - val_accuracy: 0.1849
Epoch 12/15
17/17 [==============================] - 651s 38s/step - loss: 1.9352 - accuracy: 0.3346 - val_loss: 2.1493 - val_accuracy: 0.2101
Epoch 13/15
17/17 [==============================] - 650s 38s/step - loss: 1.8186 - accuracy: 0.4054 - val_loss: 1.8920 - val_accuracy: 0.3193
Epoch 14/15
17/17 [==============================] - 639s 38s/step - loss: 1.8506 - accuracy: 0.3824 - val_loss: 1.8685 - val_accuracy: 0.3782
Epoch 15/15
17/17 [==============================] - 631s 37s/step - loss: 1.5605 - accuracy: 0.5048 - val_loss: 1.6948 - val_accuracy: 0.4454
4/4 [==============================] - 32s 8s/step - loss: 1.6948 - accuracy: 0.4454
4/4 [==============================] - 33s 8s/step
```

- Lastly, when the epochs ran to about 25/50 , the accuracy of the model increases significantly to 92%.



```
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
17/17 [==============================] - 657s 38s/step - loss: 3.2615 - accuracy: 0.1300 - val_loss: 2.1977 - val_accuracy: 0.1513
Epoch 2/50
17/17 [==============================] - 650s 38s/step - loss: 2.1965 - accuracy: 0.1377 - val_loss: 2.1810 - val_accuracy: 0.1092
Epoch 3/50
17/17 [==============================] - 643s 38s/step - loss: 2.2155 - accuracy: 0.1052 - val_loss: 2.1979 - val_accuracy: 0.1513
Epoch 4/50
17/17 [==============================] - 648s 38s/step - loss: 2.1949 - accuracy: 0.1205 - val_loss: 2.1989 - val_accuracy: 0.1513
Epoch 5/50
17/17 [==============================] - 649s 38s/step - loss: 2.2192 - accuracy: 0.1338 - val_loss: 2.1996 - val_accuracy: 0.1513
Epoch 6/50
17/17 [==============================] - 649s 38s/step - loss: 2.2047 - accuracy: 0.1109 - val_loss: 2.2014 - val_accuracy: 0.1429
Epoch 7/50
17/17 [==============================] - 648s 38s/step - loss: 2.1947 - accuracy: 0.1262 - val_loss: 2.1976 - val_accuracy: 0.1429
Epoch 8/50
17/17 [==============================] - 628s 37s/step - loss: 2.1924 - accuracy: 0.1434 - val_loss: 2.2325 - val_accuracy: 0.0000e+00
Epoch 9/50
17/17 [==============================] - 626s 37s/step - loss: 2.2022 - accuracy: 0.1587 - val_loss: 2.1899 - val_accuracy: 0.1681
Epoch 10/50
17/17 [==============================] - 615s 36s/step - loss: 2.1745 - accuracy: 0.1549 - val_loss: 2.1681 - val_accuracy: 0.1429
Epoch 11/50
17/17 [==============================] - 629s 37s/step - loss: 2.1577 - accuracy: 0.1950 - val_loss: 2.1813 - val_accuracy: 0.1933
Epoch 12/50
17/17 [==============================] - 626s 37s/step - loss: 2.1582 - accuracy: 0.2161 - val_loss: 2.1249 - val_accuracy: 0.2101
Epoch 13/50
17/17 [==============================] - 630s 37s/step - loss: 2.0777 - accuracy: 0.2428 - val_loss: 2.0874 - val_accuracy: 0.2437
Epoch 14/50
17/17 [==============================] - 628s 37s/step - loss: 2.0533 - accuracy: 0.2792 - val_loss: 2.0674 - val_accuracy: 0.1933
Epoch 15/50
17/17 [==============================] - 621s 37s/step - loss: 1.9526 - accuracy: 0.3098 - val_loss: 2.0420 - val_accuracy: 0.2521
Epoch 16/50
17/17 [==============================] - 630s 37s/step - loss: 1.8182 - accuracy: 0.3518 - val_loss: 2.3804 - val_accuracy: 0.1429
Epoch 17/50
17/17 [==============================] - 626s 37s/step - loss: 1.7432 - accuracy: 0.4168 - val_loss: 1.8970 - val_accuracy: 0.4118
Epoch 18/50
17/17 [==============================] - 629s 37s/step - loss: 1.2456 - accuracy: 0.5966 - val_loss: 1.5810 - val_accuracy: 0.4538
Epoch 19/50
17/17 [==============================] - 627s 37s/step - loss: 0.7483 - accuracy: 0.7591 - val_loss: 1.3147 - val_accuracy: 0.5966
Epoch 20/50
17/17 [==============================] - 631s 37s/step - loss: 0.4597 - accuracy: 0.8623 - val_loss: 1.3118 - val_accuracy: 0.6134
Epoch 21/50
17/17 [==============================] - 633s 37s/step - loss: 0.2283 - accuracy: 0.9407 - val_loss: 2.1135 - val_accuracy: 0.6303
Epoch 22/50
14/17 [=======================>......] - ETA: 1:47 - loss: 0.2171 - accuracy: 0.9263
```

- So, after observing the results, we saw that increasing the epochs, increased the accuracy of the model. If we follow this observation, we can say that the accuracy after 50 or say 100 epochs can easily be at 98 or 99%.
- Even though we can see an increase in accuracy in more epochs, we can still not consider the accuracy as 92%/98%.
- This is because there is a sudden increase in accuracy from 44% to 59%. This can mean that the model might have been over trained and the actual accuracy is about 44%.
- The accuracy of about 44% is good as the dataset we had did not have uniformity in the images and the size also varied a lot.
- We can improve accuracy with cross validation in the future to better train our model and increase accuracy.

Chapter 9

SKILLS LEARNED

1. **Deep Learning Fundamentals:** Understanding the basics of neural networks, including feedforward and backpropagation. Grasping the concepts of activation functions, loss functions, and optimization algorithms.

2. **Convolutional Neural Networks (CNNs):** Knowledge of the architecture and components of CNNs, including convolutional layers, pooling layers, and fully connected layers. Understanding the importance of feature maps and filters in image recognition.

3. **TensorFlow or PyTorch:** Practical experience with popular deep learning frameworks like TensorFlow or PyTorch. Ability to build, train, and evaluate CNN models using these frameworks.

4. **Data Preprocessing:** Preparing and preprocessing image data for CNNs, including tasks like resizing, normalization, and data augmentation.

5. **Model Architecture and Design:** Designing CNN architectures tailored to specific tasks and datasets. Fine-tuning existing architectures or using pre-trained models for transfer learning.

6. **Hyperparameter Tuning:** Tuning hyperparameters to optimize the performance of CNN models, including learning rates, batch sizes, and regularization parameters.

7. **Validation and Evaluation:** Understanding metrics such as accuracy.

8. **Communication Skills:** Ability to communicate technical concepts effectively, both in writing and verbally.

9. **Continuous Learning:** Demonstrating a commitment to staying updated on the latest developments in deep learning and computer vision.

Chapter 10

CONCLUSION

In conclusion, the utilization of Convolutional Neural Networks (CNNs) for weapon classification represents a significant step forward in enhancing security and safety measures. CNNs have demonstrated their prowess in accurately identifying and categorizing weapons from images and videos, making them a valuable tool in various domains, including law enforcement, border security, and public safety.

Through the training of CNN models on diverse datasets encompassing various types of weapons, these systems have exhibited impressive accuracy and robustness. Their ability to automatically learn distinctive features and patterns from weapon images, even in complex and challenging environments, underscores their practicality.

However, the success of weapon classification using CNN models is contingent upon the availability of high-quality, annotated datasets, and continuous model refinement to adapt to evolving threats and new weapon designs. Ethical considerations and privacy concerns also warrant careful attention when implementing such technology.

REFERENCES

- DEEP LEARNING (IAN GOODFELLOW, YOSHUA BENGIO, AARON CONRVILLE)
- MACHINE LEARNING (TOM M. MITCHELL)
- MACHINE LEARNING ALGORITHMS AND APPLICATIONS (MOHSSEN MOHAMMED, MUHAMMED BADRUDDIN KHAN, EIHAB BASHIER MOHAMMAD BASHIER)