CA2-Data Preprocessing and Algorithms

March 26, 2021

1 Machine Learning Tutorial


```
Author:     Theodora Tataru
            C00231174
Tutor:      Greg Doyle
Course:     Software Development, 4th year
Institute:  Institute of Technology Carlow
Year:       2020
```


1.0.1 This is tutorial focuses on various preprocessing data techniques and algorithms

Artificial Intelligence

1. Artificial Intelligence
2. Tools
3. Techniques
4. ?
5. ?

Pre-processing data methods:

1. Missing data

Machine Learning Algorithms:

1. Neural Network
2. Decision Tree
3. Random Forest
4. Linear Regression
5. Reinforcement

(will be filled more later)

2 Introduction

This tutorial is designed to tackle different aspects of Machine Learning, such as pre-processing data, machine learning algorithms - training, testing and predictions. The tutorial aims to explain some different algorithms used in pre-processing data machine learning models.

3 Requirements

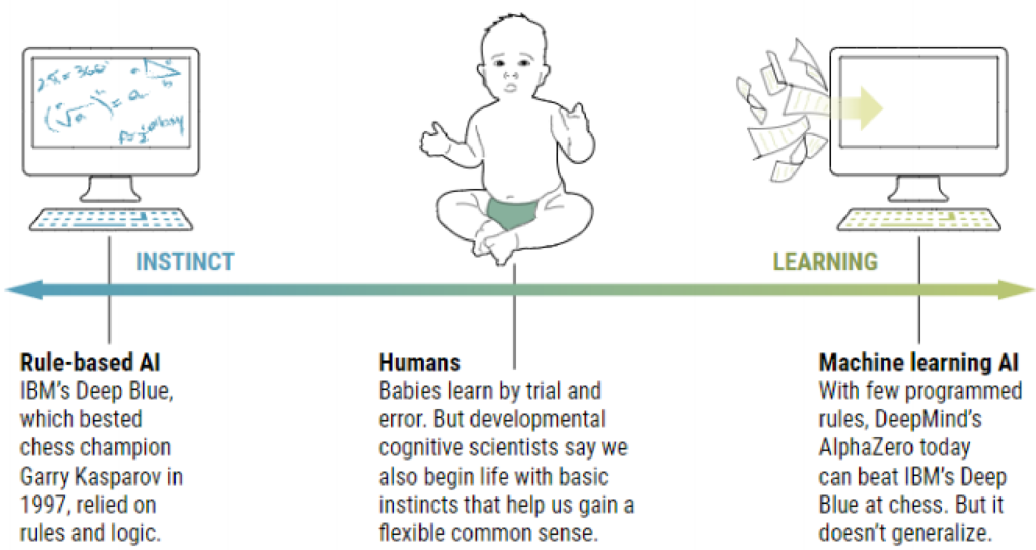
Before starting this tutorial, few requirements need to be satisfied: 1. Python knowledge is required 2. The following need to be installed on the system: #### Python 3 sudo apt-get install python3.8 #### Tensorflow python3 -m pip install tensorflow #### Keras sudo apt-get install keras #### Matplotlib python3 -m pip install mathplotlib #### NumPy python3 -m pip install numpy #### Pands python3 -m pip install pands

Artificial Intelligence

In this section, several aspects of machine learning are described at a high-level. Reading this section is extremely important for a clear understanding of the algorithms explained above.

Artificial Intelligence is the intelligence implemented into machines. This ability can be gained by performing statistical operations on data. To start the learning progress of the AI, a collection of data is be fed to the system as learning data points. The data fed to the AI needs to be accurate and clean – from any unnecessary information. This data is the single source of knowledge of AI and is the base of Artificial Intelligence. There are many ways AI can learn, and the most popular ways are: Machine Learning and Deep Learning. The human brain is programmed by the DNA that defines neural structures. These structures inside change the path of neural activity and organism behavior as a result of our experiences. There are several ways to simulate these learning mechanisms in computers [49]. For learning to happen in both flash and silicon matter, the following are needed: - A way for the system to understand what is expected of it - A way for the system to remember the information needed - A way for the system to input information - A way for the system to output information - A way to load algorithms into the system - Physical matter (hardware for machines) to support all the above bullet points.

Since the beginning of AI, the learning system has shifted from algorithms that rely on logic and rules to machine learning, in which case the algorithms contain fewer rules and absorb training data points to learn, as humans, by trial and error.

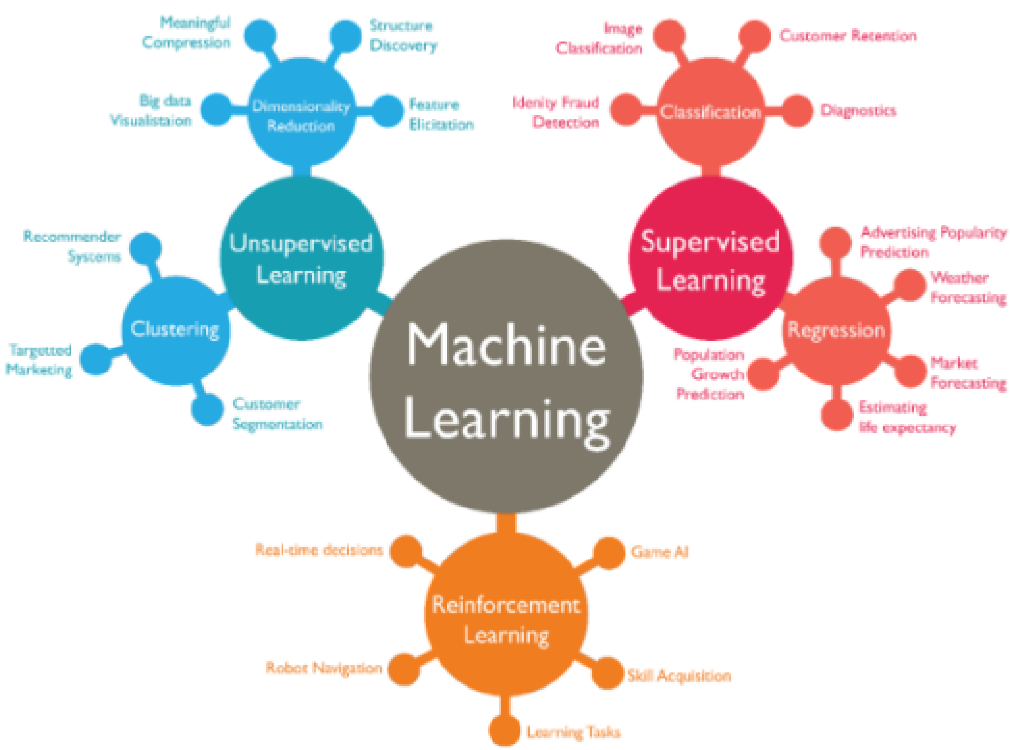


Thanks to powerful computers and big data, Machine Learning has now advanced algorithms called neural networks. These networks are just a collection of computing fundamentals shaped like the neurons in the human brain, that build stronger or weaker links as they assimilate data [50]. This is very important, as humans, do not recognize a dog by definitions like “if (number of legs==4, and tail==true, size>cat, color==” brown or white or black)”. If this was the case, we would not be able to recognize a Chikwawa with 3 legs, as a dog. The goal is to make AI think and acknowledge reality as much as possible as close to humans.

Machine learning is a part of Artificial Intelligence, that contributes with the ability to allow a machine to learn and improve from input data. Machine learning focuses on the development of computer programs that can access data and use it for learning. Machine learning uses two approaches to train a model : - Supervised learning, which ingests a set of input data that describes explicitly what the machine should focus on. These data points fed to the machine are labeled - Unsupervised learning does not feed the AI with labeled data points. In this model, the machine is supposed to organize data on its own, based on the features of the inserted data Usually, the supervised learning is used when the AI model is used to make a prediction, while the unsupervised learning model is used when data needs to be explored. Machine learning gives a system the ability to learn automatically and improve from experience, without being precisely programmed or without any human interaction.

Following, in this section, we have a high-level look at how data is pre-processed, what algorithms are mostly used and how the training and the testing steps are achieved. In later sections, some actions are implemented and explained using Python3.

3.1 There are 3 main Machine Learning Algorithms:



1.
Supervised Learning The supervised machine learning algorithms use labeled data to learn how the mapping function works to map the input variable with the out variable. The supervised learning algorithm consists in several types of supervised learning, but we will focus on two main types: #### Classification: It is used to predict the outcome of input data based on cathegories, that

the model was trained to recognize.

Regression: It is used to predict the outcome of a input data when the output variable is in the form of real values.

2. Unsupervised Learning The unsupervised machine learning algorithms are used when the model is fed with input data and the model itself needs to organize data on its own, based on patterns and features of the inserted data. Again, there are several types of unsupervised machine learning models, but we will focus on the two main types: #### Association It is used to discover relations between input variables. It is highly used in the market analysis as it easily computes the probability of the co-occurrence of items in a collection. #### Clustering is used to group the input data as similarities are found.

3. Reinforcement Learning: Reinforcement learning is a different type of algorithm, that is designed to allow an agent to predict the best next move. The decision is based on the agent's current state and by learning behaviors that maximaze, it's reward.

Tools - python - tensorflow - keras - numpy - matplotlib - pands (This is gonna be filled later)

Techniques (filled later)

3.2 Data pre-processing

This tutorial demonstrates different techniques for pre-processing data. In this part of the tutorial, the following pre-processing methods: - data cleaning - handle missing data - handle noisy data - binning data for data smoothing - data integration and transformation - handle duplicate data - data integration - data reduction - cube aggregation

Data cleaning - Missing data

3.2.1 Packages needed:

```
- Python 3
- Pandas
- NumPy
- Scikit-Learn
```

3.2.2 Overview

Dataset: Diabetes dataset [1]

```
- Diabetes Dataset
- Has missing values: YES
- Source: National Institute of Diabetes and Digestive Kidney Diseases
- Date: 1990
- Number of instances: 768
- Number of attributes: 8+
       1. Number of times pregnant
       2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
       3. Diastolic blood pressure (mm Hg)
       4. Triceps skinfold thickness (mm)
       5. 2-Hour serum insulin (mu U/ml)
```

```
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)
```

Process [1]:

- mark missing values
- remove rows with missing values
- replace missing values
- use algorithms that support missing values

[1]:
```python
## Mark Missing values
from pandas import read_csv ## used to load the dataset
dataset = read_csv('pima-indians-diabetes.csv') ## load the data set from␣
↪harddisk
print(dataset) ## print the summary of the dataset, to see missing values
```

```
      6  148  72  35    0  33.6  0.627  50  1
0     1   85  66  29    0  26.6  0.351  31  0
1     8  183  64   0    0  23.3  0.672  32  1
2     1   89  66  23   94  28.1  0.167  21  0
3     0  137  40  35  168  43.1  2.288  33  1
4     5  116  74   0    0  25.6  0.201  30  0
..   ..  …   ..  ..  …   …     …   ..  ..
762  10  101  76  48  180  32.9  0.171  63  0
763   2  122  70  27    0  36.8  0.340  27  0
764   5  121  72  23  112  26.2  0.245  30  0
765   1  126  60   0    0  30.1  0.349  47  1
766   1   93  70  31    0  30.4  0.315  23  0

[767 rows x 9 columns]
```

Missing data can be represented by out-of-range values. In a numeric field where values should be positive, missing data can be represented by 0 or negative numbers [1].

[2]:
```python
## usingPanda DataFrame, we can print the dataset summary statistics on each␣
↪field
print(dataset.describe())
```

```
                6         148          72          35           0        33.6  \
count  767.000000  767.000000  767.000000  767.000000  767.000000  767.000000
mean     3.842243  120.859192   69.101695   20.517601   79.903520   31.990482
std      3.370877   31.978468   19.368155   15.954059  115.283105    7.889091
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      1.000000   99.000000   62.000000    0.000000    0.000000   27.300000
50%      3.000000  117.000000   72.000000   23.000000   32.000000   32.000000
75%      6.000000  140.000000   80.000000   32.000000  127.500000   36.600000
max     17.000000  199.000000  122.000000   99.000000  846.000000   67.100000
```

```
          0.627         50          1
count  767.000000  767.000000  767.000000
mean     0.471674   33.219035    0.348110
std      0.331497   11.752296    0.476682
min      0.078000   21.000000    0.000000
25%      0.243500   24.000000    0.000000
50%      0.371000   29.000000    0.000000
75%      0.625000   41.000000    1.000000
max      2.420000   81.000000    1.000000
```

As seen in the above line, there are fields that have a minimum value of zero. On some columns, the value of zero does not make sense and indicates that their values are missing [1].

Columns with missing values:

```
1: Plasma glucose concentration
2: Diastolic blood pressure
3: Triceps skinfold thickness
4: 2-Hour serum insulin
5: Body mass index
```

[3]:

```
## We can confirm the missing values, by analyzing the raw data. Therefore, we␣
↪will print the first 10 rows of the dataset
print(dataset.head(10))
```

```
    6  148  72  35    0  33.6  0.627  50  1
0   1   85  66  29    0  26.6  0.351  31  0
1   8  183  64   0    0  23.3  0.672  32  1
2   1   89  66  23   94  28.1  0.167  21  0
3   0  137  40  35  168  43.1  2.288  33  1
4   5  116  74   0    0  25.6  0.201  30  0
5   3   78  50  32   88  31.0  0.248  26  1
6  10  115   0   0    0  35.3  0.134  29  0
7   2  197  70  45  543  30.5  0.158  53  1
8   8  125  96   0    0   0.0  0.232  54  1
9   4  110  92   0    0  37.6  0.191  30  0
```

To simplify things, we can print the count of the number of missing values on each column. For better visualization, we will mark all missing values as “True”, and then, we can count the the “True” values for each column [1].

[4]:

```
dataset = read_csv('pima-indians-diabetes.csv', header=None)
## count the number of missing values from all 5 columns
missing = (dataset[[0,1,2,3,4,5,6,7,8]] == 0).sum()
print(missing)
```

```
0    111
1      5
2     35
```

```
3    227
4    374
5     11
6      0
7      0
8    500
dtype: int64
```

It can be seen that columns 1, 2 and 5 has few missing value, while column 3, 4 and 8 have many missing values. In Python, missing values are usually marked as NaN. This values Nan are ignored when operations are performed [1].

[5]:
```
from numpy import nan
## replacing all zero values by nan in the dataset
dataset[[0,1,2,3,4,5,6,7,8]] = dataset[[0,1,2,3,4,5,6,7,8]].replace(0,nan)
print(dataset.isnull().sum())
```

```
0    111
1      5
2     35
3    227
4    374
5     11
6      0
7      0
8    500
dtype: int64
```

As the sum of counting the zeros in the dayaset, matches the counting when using nan, confirms that we marked and identified the missing values correctly.

[6]:
```
## confirming that the zero values were replaced by NaN
print(dataset.head(10))
```

```
      0      1     2     3      4     5      6   7    8
0   6.0  148.0  72.0  35.0    NaN  33.6  0.627  50  1.0
1   1.0   85.0  66.0  29.0    NaN  26.6  0.351  31  NaN
2   8.0  183.0  64.0   NaN    NaN  23.3  0.672  32  1.0
3   1.0   89.0  66.0  23.0   94.0  28.1  0.167  21  NaN
4   NaN  137.0  40.0  35.0  168.0  43.1  2.288  33  1.0
5   5.0  116.0  74.0   NaN    NaN  25.6  0.201  30  NaN
6   3.0   78.0  50.0  32.0   88.0  31.0  0.248  26  1.0
7  10.0  115.0   NaN   NaN    NaN  35.3  0.134  29  NaN
8   2.0  197.0  70.0  45.0  543.0  30.5  0.158  53  1.0
9   8.0  125.0  96.0   NaN    NaN   NaN  0.232  54  1.0
```

Having missing values in a training dataset can cause errors in the machine learning algorithms and lead to erroneous predictions. It is essential to handle the missing data prior to developing the model and the training process.

Removing the missing values The easiest and simplistic strategy to handle the missing data is to remove all records containing missing data. To achieve this, a new Panda DataFrame can be created with the rows containing the missing values removed. Pandas provide a function dropna(), that can be used to remove columns or rows with missing data. In our example, we will use this function to remove all wors that contain missing data [1].

[7]:
```
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# summarize the shape of the raw data
print(dataset.shape)
# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# drop rows with missing values
dataset.dropna(inplace=True)
# summarize the shape of the data with missing rows removed
print(dataset.shape)
```

```
(768, 9)
(392, 9)
```

The output of the code above: - the first row shows the initial number of rows contained in the dataset - the second row shows the remaining number of rows that do not contain missing data.

Now that the data had been cleaned for missing values, an algorithm sensitive to missing data can be used to determine the accuracy that can be obtained with the remaining data [1]. #### Latent Dirichlet Allocation (LDA) LDA is an unsupervised learning algorithm that views data as words and works on making a key assumption [2].

Running this algorithm, the output might vary, given the nature of the algorithm. The algorithm should be executed a few times in a row and compute the average outcome to determine the average accuracy that the data can provide [2].

[8]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# define the model
model = LinearDiscriminantAnalysis()
# define the model evaluation procedure
cv = KFold(n_splits=3, shuffle=True, random_state=1)
# evaluate the model
result = cross_val_score(model, X, y, cv=cv, scoring='accuracy')
# report the mean performance
print('Accuracy: %.3f' % result.mean())
```

```
Accuracy: 0.781
```

The approach presented above, which implies the deletion of all rows containing missing values, can limit the model's prediction. Therefore, new methods of dealing with missing values will be detailed below [1].

Impute Missing Values This method implies the replacement of missing values, and there are many ways of replacing missing values, such as: Replacing a missing value with: 1. a constant value that has meaning 2. a random value from another record 3. a mean, median or mode value for that column 4. a value estimated by another predictive value Each option presented above will have a different impact on the model, and on the predictions, the model will produce. Pandas, provide a function called fillna(), that replaces missing values with a specific value [1]. ##### Replacing missing values with the mean of the column. This function allows the developer to specify the value that replaces the missing value and the technique used to replace it [1]. The Pipeline is used to define the modeling pipeline, where data is primarily passed through the SimpleImputer to be transformed, and only after fed to the model [1].

[9]:
```
# example of evaluating a model after an imputer transform
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# define the imputer
imputer = SimpleImputer(missing_values=nan, strategy='mean')
# define the model
lda = LinearDiscriminantAnalysis()
# define the modeling pipeline
pipeline = Pipeline(steps=[('imputer', imputer),('model', lda)])
# define the cross validation procedure
kfold = KFold(n_splits=3, shuffle=True, random_state=1)
# evaluate the model
result = cross_val_score(pipeline, X, y, cv=kfold, scoring='accuracy')
# report the mean performance
print('Accuracy: %.3f' % result.mean())
```

```
Accuracy: 0.762
```

Let's compare the accuracy from the LDA algorithm that removes the rows with missing values and the accuracy of the model that replaced the missing values with the column's mean. We can observe that accuracy had decreased. Try replacing the missing values with other values and compare the results again. For a more detailed example of imputing missing values, check this tutorial: https://machinelearningmastery.com/statistical-imputation-for-missing-values-in-machine-learning/

4 Machine Learning Algorithms

4.1 A high level understanding of machine learning algorithms

Convolutional Neural Networks

This tutorial will guide step by step into the training and testing CNN model to classify images. Keras Sequential API is used for this model to create and train the model [3].

Sequential API allows the developers of the model to arrange the layers in sequential order, meaning that the flow of data is processed only in one direction. The disadvantage is that this sequential model does not allow us to build a model with multiple inputs and outputs.

Requirements

[10]:
```
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras import datasets, layers, models
```

[11]:
```
#This method will mark the start and the end of the training and testing the␣
↪models
def date_and_time_now():
    import datetime
    now = datetime.datetime.now()
    return now.strftime("%Y-%m-%d %H:%M:%S")
```

Dataset The cifar-10 image dataset is composed of 60,000 colored images, with a size 32x32 and 10 labels. Each class contains 6,000 images. From the whole set of 60,000 images, 50,000 will be used for training and 10,000 for testing. The classes are unique, and there is no overlap in between them [3].

[12]:
```
# downloading the data and dividing it into the training and testing set
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.
↪load_data()
```

Normalizing the pixel values from 255-0 to 1-0. This step is performed to facilitate the training process's speed, as large values can disrupt or slow down the process learning process. It is good practice to normalize the pixel values so that each pixel value has a value between 0 and 1.

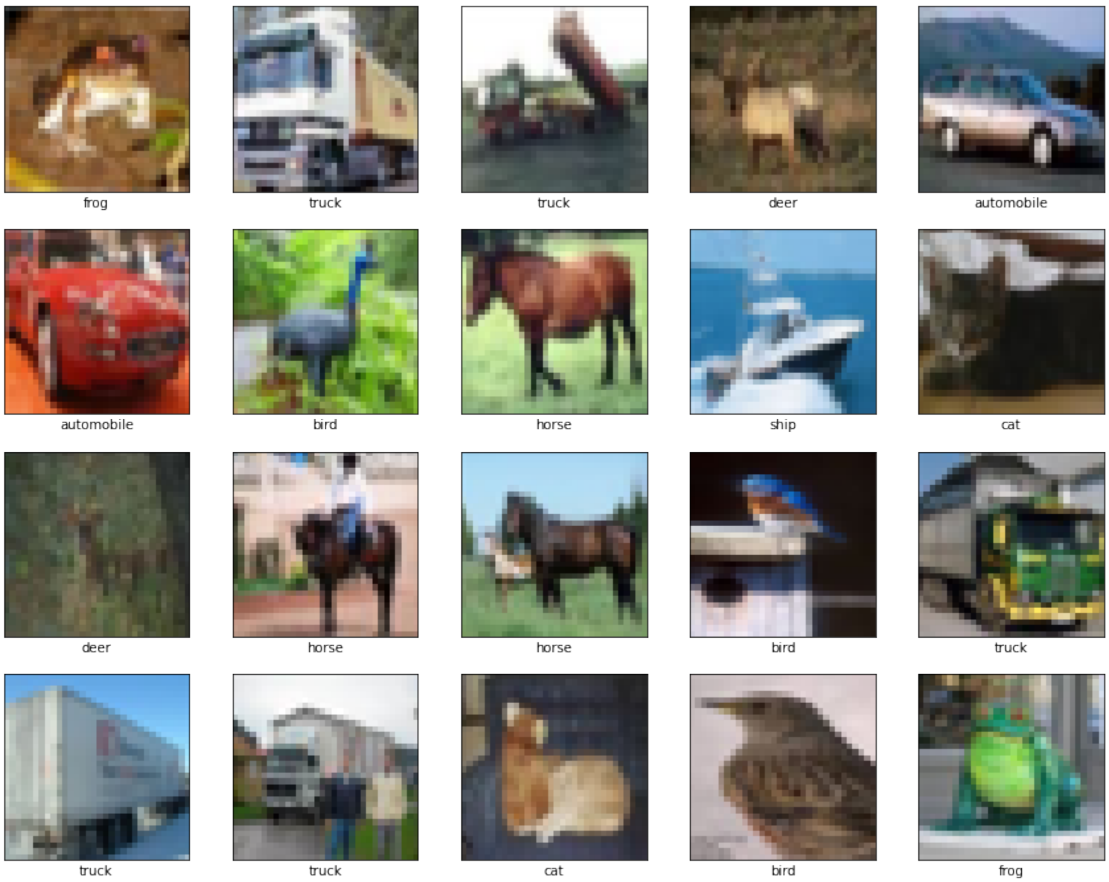
[13]:
```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Verify the data

[14]:
```
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

[15]:

```
plt.figure(figsize=(15,15))
for i in range(20):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(classes[train_labels[i][0]])
plt.show()
```



Creating the sequential model

- Con2D is the input layer that creates a convolutional kernel that feeds its input to the next available layer. The input images are RGB(colored) images with 32x32 dimensions. “relu” is a function that outputs the input directly to the next layer if it is positive, or in case it is negative, outputs 0.
- MaxPooling2D layer reduces the dimension of images by reducing the number of pixels in the output from the previous layer.
- Flatten layer is converting the data from a multi-dimensional array to a 1 dimension array. The reason for using this layer is to create a single long feature vector.

- Dense layer is special, it is fully connected with the previous layer; in other words, this layer's neurons are all connected with every neuron of the previous layer.

[16]:
```
def define_model():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,␣
 ↪3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten(input_shape=(28, 28)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10))
    return model
```

Sumarize the arhitecture of the model:

[17]:
```
model = define_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

As seen above, the images are shrinking as they are going deeper into the model, this is done so that the model can perform computationally more output channels in each Conv2D layer.

To complete the model, the last two layers, called Dense, perform the classification. These layers take vectors as input from the Flatten layer.

Compile the model - Training process Compiling parameters: - optimizers [9] - adam is an algorithm, one of the most popular used in CNNs, used in computer vision and natural language processing. The optimizer is computationally efficient and requires little memory to perform. It is appropriate for problems with noisy gradients - adagrad is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing small updates where suitable - adadelta is an extension of adagrad that is less agressive, decreasing the learning rate. - RMSprp is an adaptive learning algorithm that divides the learning rate by exponentially decaying the average of squared gradients - adamax is an algorithm that updates the scaling rule inverse proportionally to the norm of the past gradients and current gradients - nadam can be seen as a combination of Adam and NAG algorithms - More details on https://ruder.io/optimizing-gradient-descent/index.html#momentum - loss is calculate as a difference between predictions and the true labels. The value of loss shows how poorly or well the model behaves during training

[18]:
```
def compile_model():
    models = {}
    optimizers = ["adam", "adamax", "adagrad", "adadelta",  "RMSprop", "nadam"]
    for opt in optimizers:
        ␣
 ↪print("_____________________________________________________________________________________
        print("----->Training the model with:", opt, "oprimizer")
        model.compile(optimizer=opt, loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
        history = model.fit(train_images, train_labels, epochs=10,␣
 ↪validation_data=(test_images, test_labels))

        ␣
 ↪####################################################################################
        # Evaluate the training process
        print("----->Evaluation of the training process for:", opt)
        plt.plot(history.history['accuracy'], label='accuracy')
        plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.title(opt)
        plt.ylim([0.5, 1])
        plt.legend(loc='lower right')
        plt.show()

        ␣
 ↪####################################################################################
        # testing the model
        test_loss, test_acc = model.evaluate(test_images,  test_labels,␣
 ↪verbose=2)
        print('----->Test accuracy for', opt, ':', test_acc)
```

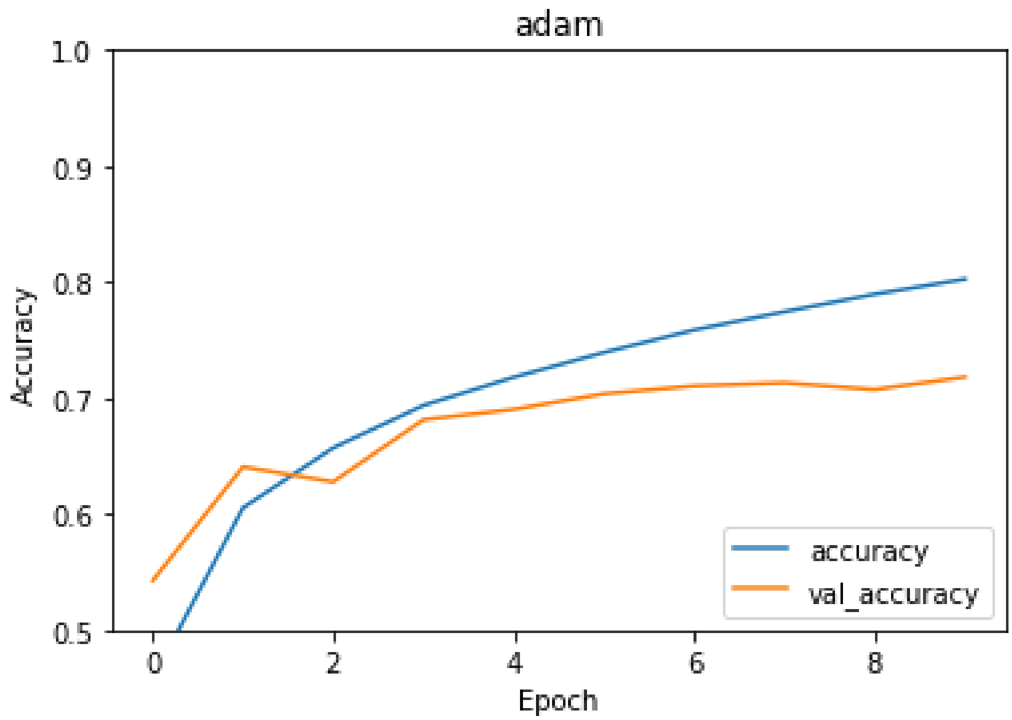
```
    ␣
↪##########################################################################################
        # predictions
        print("----->Prediction for:", opt)
        predictions = model.predict(test_images)
        import numpy as np

        COLOR = 'white'
        plt.rcParams['text.color'] = COLOR
        plt.rcParams['axes.labelcolor'] = COLOR
        def predict(model, image, correct_label):
            classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                       'dog', 'frog', 'horse', 'ship', 'truck']
            prediction = model.predict(np.array([image]))
            predicted_class = classes[np.argmax(prediction)]
            show_image(image, classes[correct_label], predicted_class)
        def show_image(img, label, guess):
            plt.figure()
            plt.imshow(img, cmap=plt.cm.binary)
            print("Expected: " + label)
            print("Guess: " + guess)
            plt.colorbar()
            plt.grid(False)
            plt.show()

        import random
        num = random.randint(1,10000)
        image = test_images[num]
        label = test_labels[num][0]
        predict(model, image, label)

        models[opt] = model
    return models
```

[19]:
```
# marking the start of the process of the simple model
start_simple_model = date_and_time_now()
```

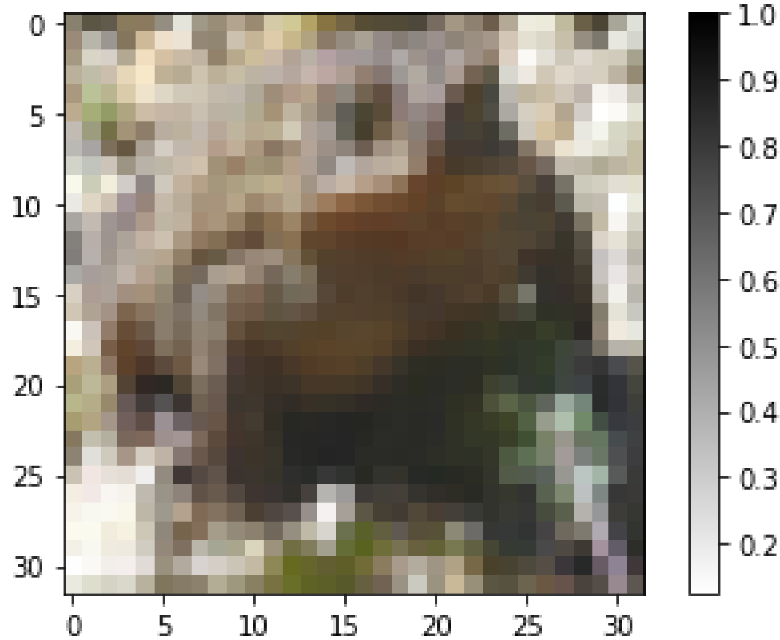
[20]:
```
models = compile_model()
```

```
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: adam oprimizer
Epoch 1/10
1563/1563 [==============================] - 66s 41ms/step - loss: 1.7209 -
accuracy: 0.3637 - val_loss: 1.3099 - val_accuracy: 0.5427
Epoch 2/10
```

```
1563/1563 [==============================] - 63s 41ms/step - loss: 1.1651 -
accuracy: 0.5869 - val_loss: 1.0304 - val_accuracy: 0.6408
Epoch 3/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.9933 -
accuracy: 0.6466 - val_loss: 1.1079 - val_accuracy: 0.6279
Epoch 4/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.8808 -
accuracy: 0.6924 - val_loss: 0.9080 - val_accuracy: 0.6814
Epoch 5/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.8038 -
accuracy: 0.7167 - val_loss: 0.8941 - val_accuracy: 0.6903
Epoch 6/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.7387 -
accuracy: 0.7416 - val_loss: 0.8674 - val_accuracy: 0.7037
Epoch 7/10
1563/1563 [==============================] - 71s 46ms/step - loss: 0.6811 -
accuracy: 0.7613 - val_loss: 0.8451 - val_accuracy: 0.7105
Epoch 8/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.6281 -
accuracy: 0.7784 - val_loss: 0.8678 - val_accuracy: 0.7132
Epoch 9/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.5847 -
accuracy: 0.7934 - val_loss: 0.8969 - val_accuracy: 0.7074
Epoch 10/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.5387 -
accuracy: 0.8103 - val_loss: 0.8744 - val_accuracy: 0.7182
----->Evaluation of the training process for: adam
```

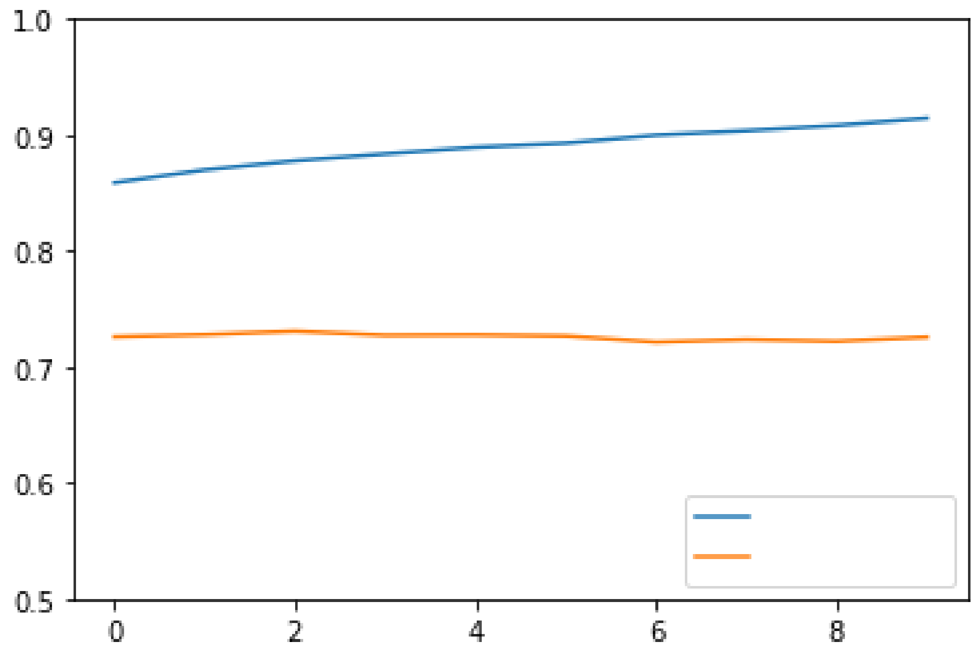



```
313/313 - 3s - loss: 0.8744 - accuracy: 0.7182
----->Test accuracy for adam : 0.7182000279426575
----->Prediction for: adam
Expected: frog
Guess: frog
```

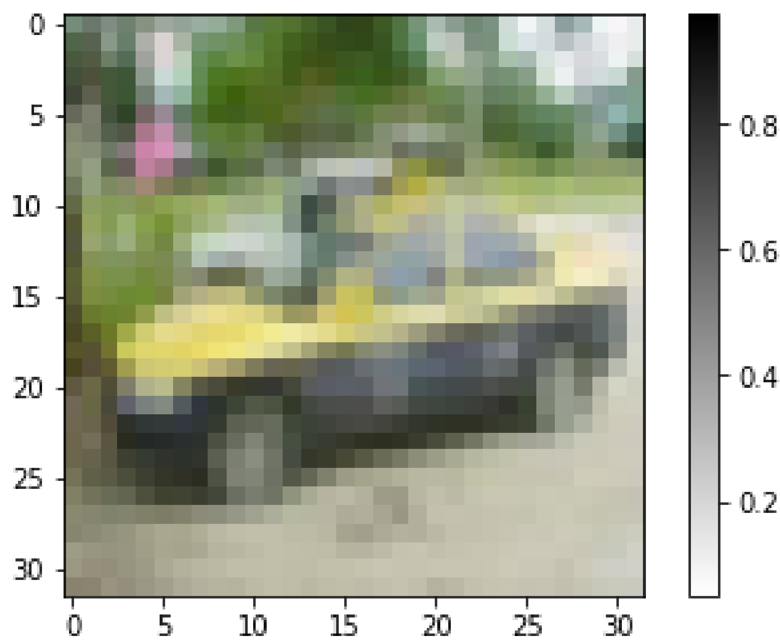



```
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: adamax oprimizer
Epoch 1/10
1563/1563 [==============================] - 64s 40ms/step - loss: 0.4202 -
accuracy: 0.8541 - val_loss: 0.8656 - val_accuracy: 0.7260
Epoch 2/10
1563/1563 [==============================] - 63s 40ms/step - loss: 0.3660 -
accuracy: 0.8736 - val_loss: 0.8907 - val_accuracy: 0.7278
Epoch 3/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.3464 -
accuracy: 0.8823 - val_loss: 0.9031 - val_accuracy: 0.7307
Epoch 4/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.3270 -
accuracy: 0.8898 - val_loss: 0.9348 - val_accuracy: 0.7273
Epoch 5/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.3202 -
accuracy: 0.8923 - val_loss: 0.9341 - val_accuracy: 0.7276
Epoch 6/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.2999 -
accuracy: 0.8994 - val_loss: 0.9512 - val_accuracy: 0.7267
Epoch 7/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.2899 -
accuracy: 0.9021 - val_loss: 0.9925 - val_accuracy: 0.7214
Epoch 8/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.2783 -
```

```
accuracy: 0.9058 - val_loss: 1.0097 - val_accuracy: 0.7236
Epoch 9/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.2647 -
accuracy: 0.9117 - val_loss: 1.0342 - val_accuracy: 0.7221
Epoch 10/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.2466 -
accuracy: 0.9183 - val_loss: 1.0582 - val_accuracy: 0.7257
----->Evaluation of the training process for: adamax
```

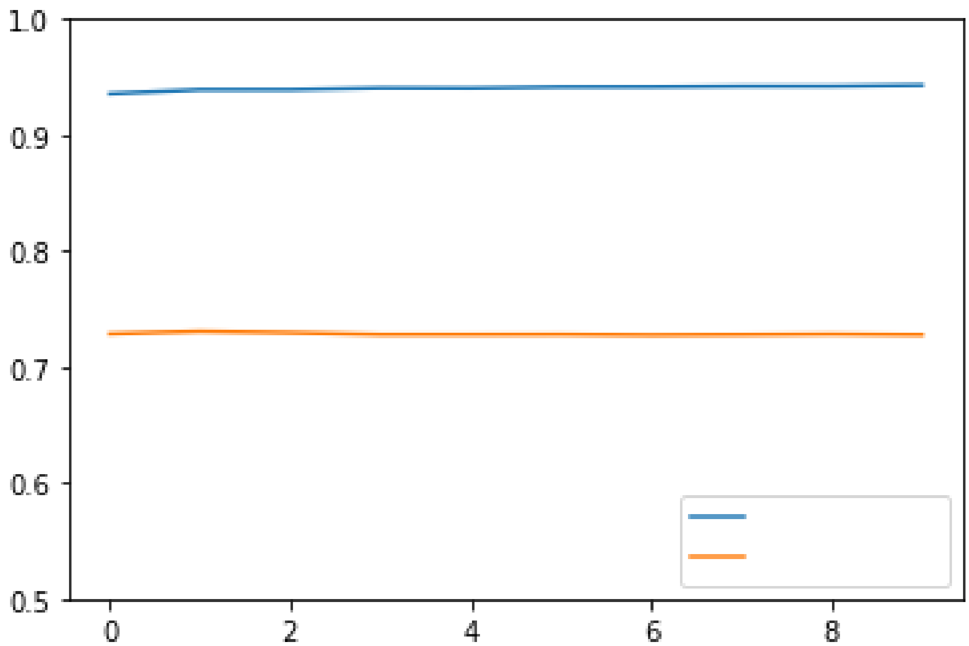



```
313/313 - 3s - loss: 1.0582 - accuracy: 0.7257
----->Test accuracy for adamax : 0.7257000207901001
----->Prediction for: adamax
Expected: automobile
Guess: truck
```

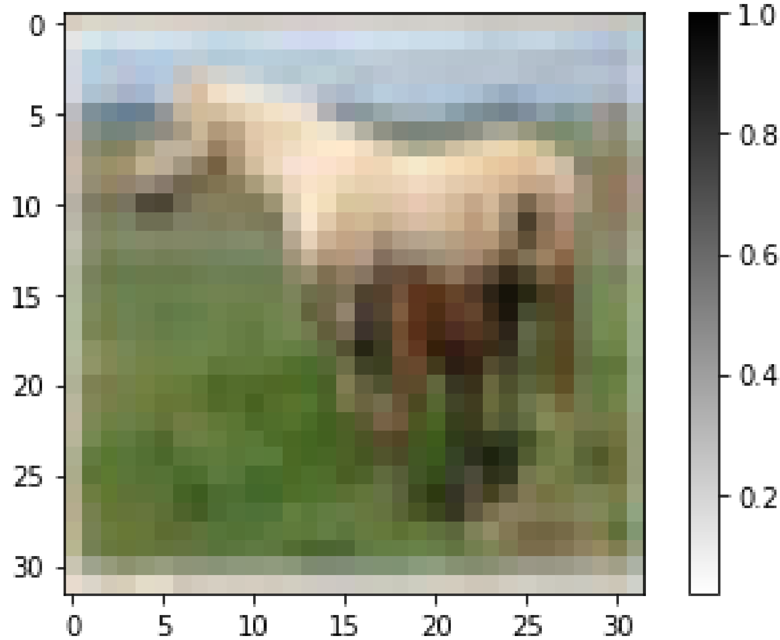



```
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: adagrad oprimizer
Epoch 1/10
1563/1563 [==============================] - 63s 40ms/step - loss: 0.2112 -
accuracy: 0.9346 - val_loss: 1.0613 - val_accuracy: 0.7285
Epoch 2/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.2033 -
accuracy: 0.9399 - val_loss: 1.0705 - val_accuracy: 0.7305
Epoch 3/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.2038 -
accuracy: 0.9394 - val_loss: 1.0761 - val_accuracy: 0.7294
Epoch 4/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1965 -
accuracy: 0.9416 - val_loss: 1.0803 - val_accuracy: 0.7279
Epoch 5/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1972 -
accuracy: 0.9406 - val_loss: 1.0843 - val_accuracy: 0.7279
Epoch 6/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1918 -
accuracy: 0.9430 - val_loss: 1.0883 - val_accuracy: 0.7280
Epoch 7/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1948 -
accuracy: 0.9430 - val_loss: 1.0889 - val_accuracy: 0.7275
Epoch 8/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1940 -
```

```
accuracy: 0.9423 - val_loss: 1.0920 - val_accuracy: 0.7278
Epoch 9/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1951 -
accuracy: 0.9425 - val_loss: 1.0937 - val_accuracy: 0.7282
Epoch 10/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1873 -
accuracy: 0.9448 - val_loss: 1.0948 - val_accuracy: 0.7277
----->Evaluation of the training process for: adagrad
```

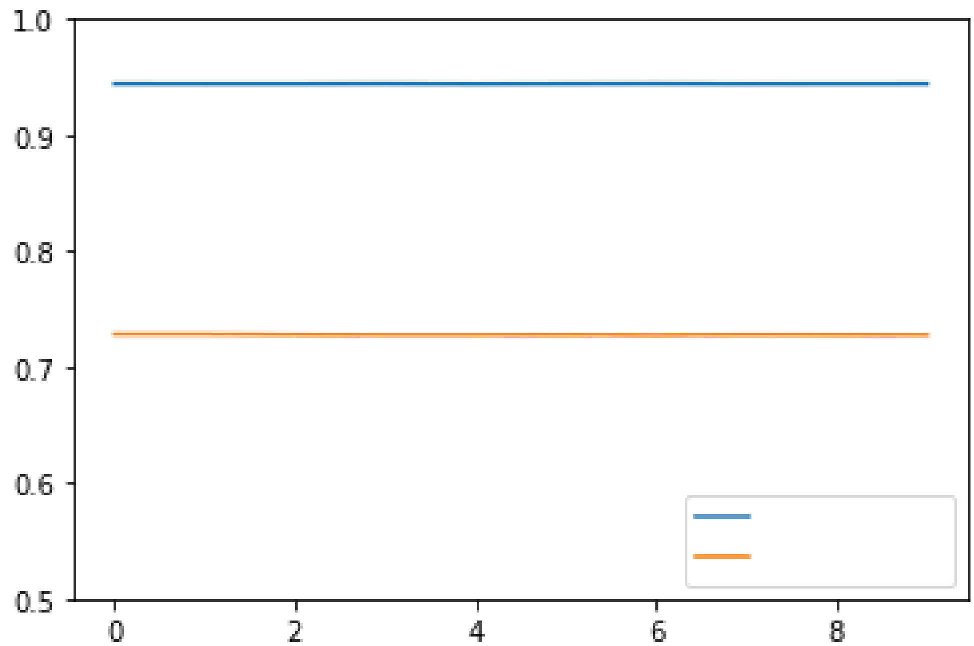



```
313/313 - 3s - loss: 1.0948 - accuracy: 0.7277
----->Test accuracy for adagrad : 0.7276999950408936
----->Prediction for: adagrad
Expected: horse
Guess: horse
```

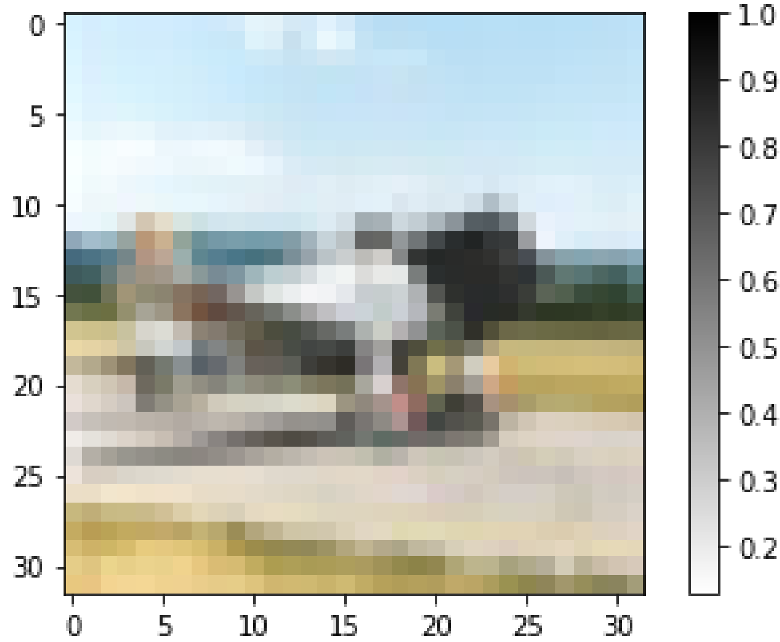



```
---------------------------------------------------------------------------------
---------------------------------
----->Training the model with: adadelta oprimizer
Epoch 1/10
1563/1563 [==============================] - 63s 40ms/step - loss: 0.1921 -
accuracy: 0.9436 - val_loss: 1.0955 - val_accuracy: 0.7282
Epoch 2/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1937 -
accuracy: 0.9437 - val_loss: 1.0963 - val_accuracy: 0.7282
Epoch 3/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1908 -
accuracy: 0.9442 - val_loss: 1.0968 - val_accuracy: 0.7279
Epoch 4/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1943 -
accuracy: 0.9429 - val_loss: 1.0974 - val_accuracy: 0.7276
Epoch 5/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1851 -
accuracy: 0.9460 - val_loss: 1.0979 - val_accuracy: 0.7276
Epoch 6/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1901 -
accuracy: 0.9442 - val_loss: 1.0983 - val_accuracy: 0.7278
Epoch 7/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1874 -
accuracy: 0.9461 - val_loss: 1.0988 - val_accuracy: 0.7274
Epoch 8/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1885 -
```

```
accuracy: 0.9448 - val_loss: 1.0991 - val_accuracy: 0.7279
Epoch 9/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1889 -
accuracy: 0.9447 - val_loss: 1.0996 - val_accuracy: 0.7278
Epoch 10/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.1873 -
accuracy: 0.9437 - val_loss: 1.1000 - val_accuracy: 0.7276
----->Evaluation of the training process for: adadelta
```

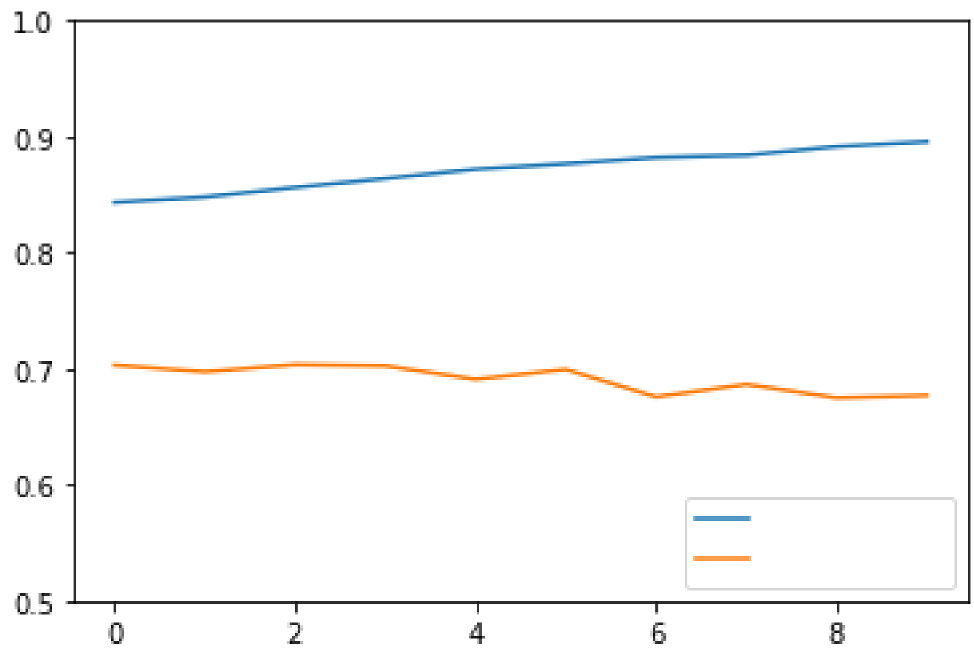



```
313/313 - 3s - loss: 1.1000 - accuracy: 0.7276
----->Test accuracy for adadelta : 0.7275999784469604
----->Prediction for: adadelta
Expected: airplane
Guess: airplane
```

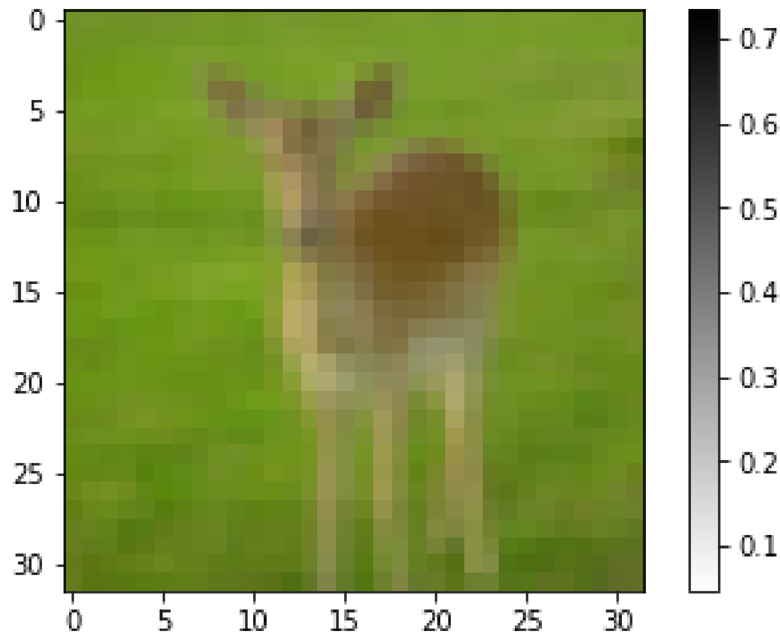



```
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: RMSprop oprimizer
Epoch 1/10
1563/1563 [==============================] - 64s 40ms/step - loss: 0.4134 -
accuracy: 0.8524 - val_loss: 1.0810 - val_accuracy: 0.7031
Epoch 2/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.3994 -
accuracy: 0.8559 - val_loss: 1.1937 - val_accuracy: 0.6974
Epoch 3/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.3792 -
accuracy: 0.8638 - val_loss: 1.1415 - val_accuracy: 0.7035
Epoch 4/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.3592 -
accuracy: 0.8734 - val_loss: 1.1935 - val_accuracy: 0.7025
Epoch 5/10
1563/1563 [==============================] - 64s 41ms/step - loss: 0.3451 -
accuracy: 0.8806 - val_loss: 1.2933 - val_accuracy: 0.6909
Epoch 6/10
1563/1563 [==============================] - 73s 47ms/step - loss: 0.3280 -
accuracy: 0.8841 - val_loss: 1.3098 - val_accuracy: 0.6994
Epoch 7/10
1563/1563 [==============================] - 71s 45ms/step - loss: 0.3181 -
accuracy: 0.8875 - val_loss: 1.4945 - val_accuracy: 0.6757
Epoch 8/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.3028 -
```

```
accuracy: 0.8922 - val_loss: 1.4124 - val_accuracy: 0.6865
Epoch 9/10
1563/1563 [==============================] - 71s 46ms/step - loss: 0.2896 -
accuracy: 0.8996 - val_loss: 1.6892 - val_accuracy: 0.6749
Epoch 10/10
1563/1563 [==============================] - 54s 34ms/step - loss: 0.2823 -
accuracy: 0.9011 - val_loss: 1.6605 - val_accuracy: 0.6768
----->Evaluation of the training process for: RMSprop
```

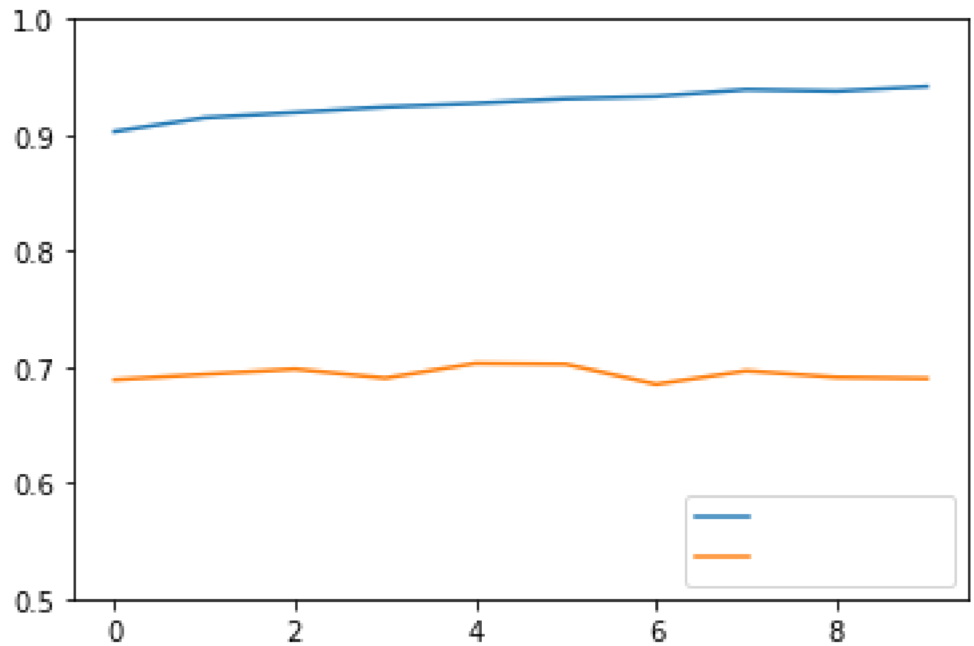



```
313/313 - 3s - loss: 1.6605 - accuracy: 0.6768
----->Test accuracy for RMSprop : 0.676800012588501
----->Prediction for: RMSprop
Expected: deer
Guess: bird
```

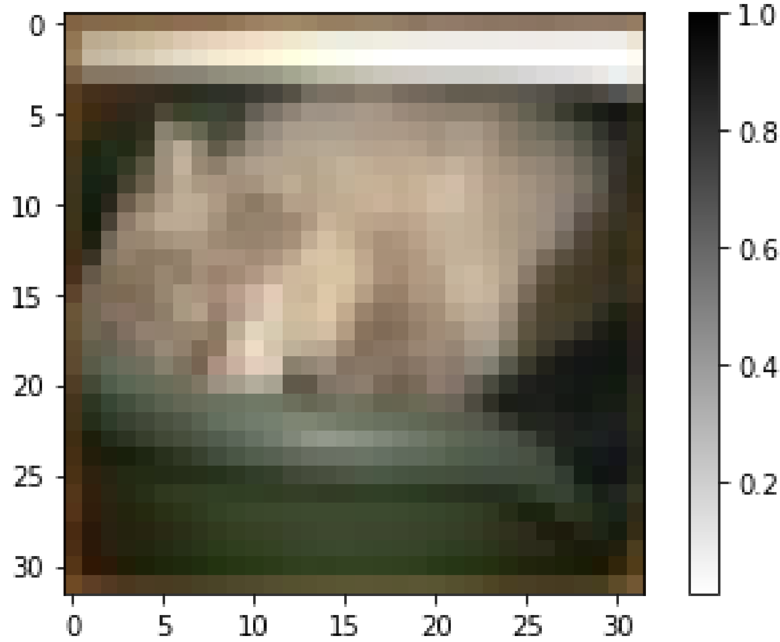



```
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: nadam oprimizer
Epoch 1/10
1563/1563 [==============================] - 57s 35ms/step - loss: 0.2524 -
accuracy: 0.9097 - val_loss: 1.4502 - val_accuracy: 0.6889
Epoch 2/10
1563/1563 [==============================] - 61s 39ms/step - loss: 0.2106 -
accuracy: 0.9255 - val_loss: 1.5467 - val_accuracy: 0.6936
Epoch 3/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.1977 -
accuracy: 0.9292 - val_loss: 1.5052 - val_accuracy: 0.6981
Epoch 4/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.1902 -
accuracy: 0.9322 - val_loss: 1.6411 - val_accuracy: 0.6903
Epoch 5/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.1801 -
accuracy: 0.9345 - val_loss: 1.6441 - val_accuracy: 0.7031
Epoch 6/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.1612 -
accuracy: 0.9427 - val_loss: 1.6911 - val_accuracy: 0.7024
Epoch 7/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.1639 -
accuracy: 0.9410 - val_loss: 1.7797 - val_accuracy: 0.6850
Epoch 8/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.1481 -
```

```
accuracy: 0.9474 - val_loss: 1.7928 - val_accuracy: 0.6966
Epoch 9/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.1493 -
accuracy: 0.9474 - val_loss: 1.8216 - val_accuracy: 0.6910
Epoch 10/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.1420 -
accuracy: 0.9494 - val_loss: 1.9344 - val_accuracy: 0.6900
----->Evaluation of the training process for: nadam
```



```
313/313 - 3s - loss: 1.9344 - accuracy: 0.6900
----->Test accuracy for nadam : 0.6899999976158142
----->Prediction for: nadam
Expected: cat
Guess: cat
```




```
[21]: # marking the end of the process of the simple model
      end_simple_model = date_and_time_now()
```

Testing the same dataset with different optimizers gave us a broad prospect over different learning curves. The optimizers with the highest test accuracy scores over 70% were adamax, adagrad and adadelta but at the same time, the rest of the optimizers did not score less than 69%, which is not bad at all, as all this was achieved with a few lines of code.

Data Augmentation Forward we are going to try to improve the model with Data Augmentation, which involves making copies of the images in the dataset, with small random modifications [10]. In simple terms, data augmentation involves modifying the pictures from the dataset. This action will expand the training dataset and allow the model to learn the same general features differently. There are many data augmentation that could be applied to images, such as [10]: 1. flip the images 2. rotation 3. scale 4. crop 5. translation

Given that our tutorial dataset involves small photos of objects, we need to use data augmentation that does not distort the images too much. Therefore, we will horizontally flip the images, zooming the images, shifting the images' height, and cropping [10].

In the next code section, the data set is populated with the additional "modified" images as specified above. Then, the training process re-starts with the new dataset.

4.1.1 Note: As the dataset is consistently larger, with the new modified images added, the process of training and testing will be consistently longer.

[22]:
```
# marking the start of the process of the data augmentation model
start_DA_model = date_and_time_now()
```

[23]:
```
from keras.preprocessing.image import ImageDataGenerator

for key in models:
   ␣
↪print("_______________________________________________________________________________________________
    print("----->Training the model with:", key, "oprimizer")
    model = models[key]
    datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1,␣
↪horizontal_flip=True)
    # prepare iterator
    it_train = datagen.flow(train_images, train_labels, batch_size=64)
    # fit model
    steps = int(train_images.shape[0] / 64)
    history = model.fit(it_train, steps_per_epoch=steps, epochs=100,␣
↪validation_data=(test_images, test_labels))
    # evaluate model
    x,acc = model.evaluate(test_images, test_labels, verbose=2)
    # learning curves
    # plot loss
    print("----->Evaluation of the training process for:", key)
    plt.plot(history.history['accuracy'], label='accuracy')
    plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
    plt.title(key)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.ylim([0.5, 1])
    plt.legend(loc='lower right')
    plt.show()
    print('----->Test accuracy for', key, ':', acc)
```

```
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: adam oprimizer
Epoch 1/100
781/781 [==============================] - 76s 96ms/step - loss: 1.0702 -
accuracy: 0.6545 - val_loss: 0.9893 - val_accuracy: 0.6754
Epoch 2/100
781/781 [==============================] - 75s 96ms/step - loss: 0.9109 -
accuracy: 0.6876 - val_loss: 0.8768 - val_accuracy: 0.7097
Epoch 3/100
781/781 [==============================] - 75s 96ms/step - loss: 0.8582 -
```

```
accuracy: 0.7059 - val_loss: 0.8707 - val_accuracy: 0.7071
Epoch 4/100
781/781 [==============================] - 75s 96ms/step - loss: 0.8220 -
accuracy: 0.7148 - val_loss: 0.8538 - val_accuracy: 0.7169
Epoch 5/100
781/781 [==============================] - 75s 96ms/step - loss: 0.7992 -
accuracy: 0.7213 - val_loss: 0.8165 - val_accuracy: 0.7301
Epoch 6/100
781/781 [==============================] - 76s 97ms/step - loss: 0.7777 -
accuracy: 0.7290 - val_loss: 0.7979 - val_accuracy: 0.7315
Epoch 7/100
781/781 [==============================] - 79s 101ms/step - loss: 0.7592 -
accuracy: 0.7362 - val_loss: 0.7943 - val_accuracy: 0.7322
Epoch 8/100
781/781 [==============================] - 78s 100ms/step - loss: 0.7457 -
accuracy: 0.7393 - val_loss: 0.7478 - val_accuracy: 0.7449
Epoch 9/100
781/781 [==============================] - 78s 100ms/step - loss: 0.7265 -
accuracy: 0.7457 - val_loss: 0.7604 - val_accuracy: 0.7458
Epoch 10/100
781/781 [==============================] - 78s 100ms/step - loss: 0.7166 -
accuracy: 0.7485 - val_loss: 0.7618 - val_accuracy: 0.7421
Epoch 11/100
781/781 [==============================] - 78s 100ms/step - loss: 0.7013 -
accuracy: 0.7541 - val_loss: 0.7310 - val_accuracy: 0.7531
Epoch 12/100
781/781 [==============================] - 78s 100ms/step - loss: 0.6897 -
accuracy: 0.7587 - val_loss: 0.7525 - val_accuracy: 0.7476
Epoch 13/100
781/781 [==============================] - 78s 100ms/step - loss: 0.6877 -
accuracy: 0.7608 - val_loss: 0.7045 - val_accuracy: 0.7628
Epoch 14/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6748 -
accuracy: 0.7661 - val_loss: 0.7233 - val_accuracy: 0.7541
Epoch 15/100
781/781 [==============================] - 79s 100ms/step - loss: 0.6639 -
accuracy: 0.7666 - val_loss: 0.6959 - val_accuracy: 0.7691
Epoch 16/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6659 -
accuracy: 0.7709 - val_loss: 0.7276 - val_accuracy: 0.7546
Epoch 17/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6520 -
accuracy: 0.7737 - val_loss: 0.7058 - val_accuracy: 0.7636
Epoch 18/100
781/781 [==============================] - 78s 100ms/step - loss: 0.6445 -
accuracy: 0.7746 - val_loss: 0.7019 - val_accuracy: 0.7659
Epoch 19/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6396 -
```

```
accuracy: 0.7769 - val_loss: 0.6652 - val_accuracy: 0.7732
Epoch 20/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6313 -
accuracy: 0.7819 - val_loss: 0.6841 - val_accuracy: 0.7699
Epoch 21/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6274 -
accuracy: 0.7798 - val_loss: 0.6611 - val_accuracy: 0.7799
Epoch 22/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6277 -
accuracy: 0.7804 - val_loss: 0.6680 - val_accuracy: 0.7747
Epoch 23/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6161 -
accuracy: 0.7856 - val_loss: 0.6668 - val_accuracy: 0.7743
Epoch 24/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6106 -
accuracy: 0.7880 - val_loss: 0.6660 - val_accuracy: 0.7781
Epoch 25/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6145 -
accuracy: 0.7835 - val_loss: 0.6776 - val_accuracy: 0.7714
Epoch 26/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6079 -
accuracy: 0.7872 - val_loss: 0.6595 - val_accuracy: 0.7789
Epoch 27/100
781/781 [==============================] - 79s 101ms/step - loss: 0.6011 -
accuracy: 0.7899 - val_loss: 0.6785 - val_accuracy: 0.7806
Epoch 28/100
781/781 [==============================] - 79s 101ms/step - loss: 0.5973 -
accuracy: 0.7902 - val_loss: 0.6486 - val_accuracy: 0.7847
Epoch 29/100
781/781 [==============================] - 79s 101ms/step - loss: 0.5960 -
accuracy: 0.7926 - val_loss: 0.6513 - val_accuracy: 0.7832
Epoch 30/100
781/781 [==============================] - 79s 101ms/step - loss: 0.5921 -
accuracy: 0.7924 - val_loss: 0.6377 - val_accuracy: 0.7871
Epoch 31/100
781/781 [==============================] - 79s 102ms/step - loss: 0.5858 -
accuracy: 0.7946 - val_loss: 0.6465 - val_accuracy: 0.7884
Epoch 32/100
781/781 [==============================] - 79s 101ms/step - loss: 0.5820 -
accuracy: 0.7949 - val_loss: 0.6837 - val_accuracy: 0.7751
Epoch 33/100
781/781 [==============================] - 79s 102ms/step - loss: 0.5788 -
accuracy: 0.7970 - val_loss: 0.6425 - val_accuracy: 0.7886
Epoch 34/100
781/781 [==============================] - 79s 101ms/step - loss: 0.5718 -
accuracy: 0.7991 - val_loss: 0.6451 - val_accuracy: 0.7834
Epoch 35/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5770 -
```

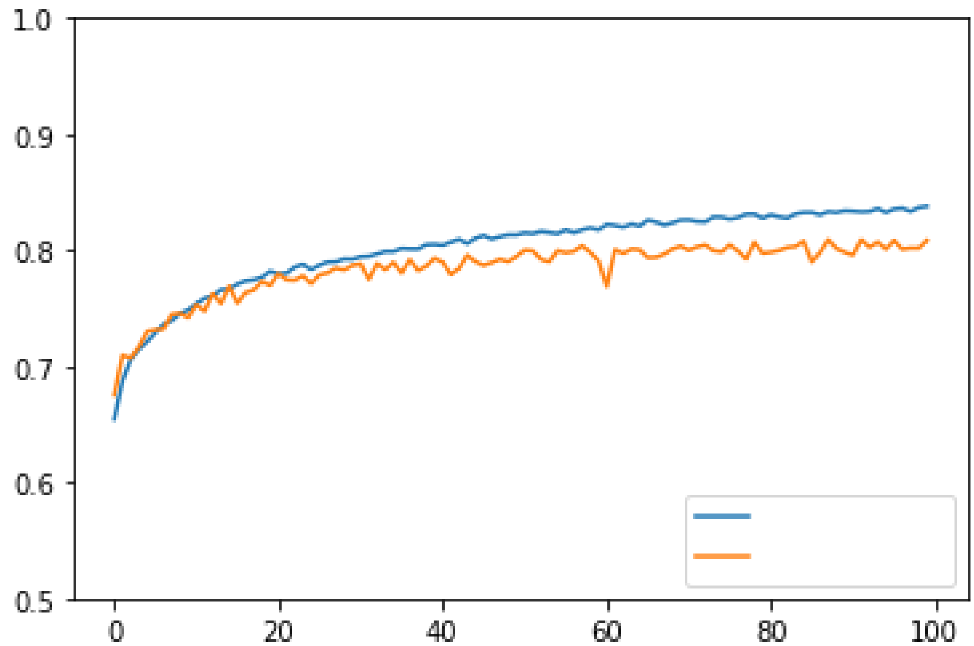
```
accuracy: 0.7990 - val_loss: 0.6351 - val_accuracy: 0.7896
Epoch 36/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5672 -
accuracy: 0.8017 - val_loss: 0.6612 - val_accuracy: 0.7806
Epoch 37/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5667 -
accuracy: 0.8012 - val_loss: 0.6220 - val_accuracy: 0.7920
Epoch 38/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5678 -
accuracy: 0.8012 - val_loss: 0.6487 - val_accuracy: 0.7827
Epoch 39/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5613 -
accuracy: 0.8050 - val_loss: 0.6498 - val_accuracy: 0.7865
Epoch 40/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5567 -
accuracy: 0.8052 - val_loss: 0.6331 - val_accuracy: 0.7932
Epoch 41/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5553 -
accuracy: 0.8044 - val_loss: 0.6409 - val_accuracy: 0.7899
Epoch 42/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5531 -
accuracy: 0.8071 - val_loss: 0.6609 - val_accuracy: 0.7792
Epoch 43/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5495 -
accuracy: 0.8095 - val_loss: 0.6527 - val_accuracy: 0.7845
Epoch 44/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5530 -
accuracy: 0.8059 - val_loss: 0.6178 - val_accuracy: 0.7958
Epoch 45/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5441 -
accuracy: 0.8105 - val_loss: 0.6356 - val_accuracy: 0.7903
Epoch 46/100
781/781 [==============================] - 83s 106ms/step - loss: 0.5417 -
accuracy: 0.8128 - val_loss: 0.6456 - val_accuracy: 0.7871
Epoch 47/100
781/781 [==============================] - 84s 107ms/step - loss: 0.5436 -
accuracy: 0.8098 - val_loss: 0.6448 - val_accuracy: 0.7895
Epoch 48/100
781/781 [==============================] - 83s 106ms/step - loss: 0.5399 -
accuracy: 0.8119 - val_loss: 0.6244 - val_accuracy: 0.7924
Epoch 49/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5368 -
accuracy: 0.8134 - val_loss: 0.6501 - val_accuracy: 0.7903
Epoch 50/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5322 -
accuracy: 0.8133 - val_loss: 0.6224 - val_accuracy: 0.7948
Epoch 51/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5335 -
```

```
accuracy: 0.8150 - val_loss: 0.6028 - val_accuracy: 0.8003
Epoch 52/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5312 -
accuracy: 0.8143 - val_loss: 0.6216 - val_accuracy: 0.7996
Epoch 53/100
781/781 [==============================] - 82s 105ms/step - loss: 0.5285 -
accuracy: 0.8163 - val_loss: 0.6286 - val_accuracy: 0.7923
Epoch 54/100
781/781 [==============================] - 83s 106ms/step - loss: 0.5276 -
accuracy: 0.8153 - val_loss: 0.6457 - val_accuracy: 0.7902
Epoch 55/100
781/781 [==============================] - 85s 108ms/step - loss: 0.5295 -
accuracy: 0.8143 - val_loss: 0.6081 - val_accuracy: 0.7998
Epoch 56/100
781/781 [==============================] - 75s 97ms/step - loss: 0.5228 -
accuracy: 0.8176 - val_loss: 0.6084 - val_accuracy: 0.7981
Epoch 57/100
781/781 [==============================] - 75s 96ms/step - loss: 0.5213 -
accuracy: 0.8152 - val_loss: 0.6133 - val_accuracy: 0.7992
Epoch 58/100
781/781 [==============================] - 75s 97ms/step - loss: 0.5234 -
accuracy: 0.8176 - val_loss: 0.6110 - val_accuracy: 0.8041
Epoch 59/100
781/781 [==============================] - 75s 96ms/step - loss: 0.5189 -
accuracy: 0.8194 - val_loss: 0.6257 - val_accuracy: 0.7983
Epoch 60/100
781/781 [==============================] - 75s 96ms/step - loss: 0.5165 -
accuracy: 0.8176 - val_loss: 0.6322 - val_accuracy: 0.7910
Epoch 61/100
781/781 [==============================] - 75s 96ms/step - loss: 0.5090 -
accuracy: 0.8223 - val_loss: 0.7077 - val_accuracy: 0.7688
Epoch 62/100
781/781 [==============================] - 75s 96ms/step - loss: 0.5142 -
accuracy: 0.8214 - val_loss: 0.6034 - val_accuracy: 0.8007
Epoch 63/100
781/781 [==============================] - 75s 96ms/step - loss: 0.5124 -
accuracy: 0.8198 - val_loss: 0.6258 - val_accuracy: 0.7974
Epoch 64/100
781/781 [==============================] - 78s 100ms/step - loss: 0.5076 -
accuracy: 0.8226 - val_loss: 0.6140 - val_accuracy: 0.8011
Epoch 65/100
781/781 [==============================] - 81s 104ms/step - loss: 0.5100 -
accuracy: 0.8208 - val_loss: 0.6051 - val_accuracy: 0.8006
Epoch 66/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4997 -
accuracy: 0.8260 - val_loss: 0.6240 - val_accuracy: 0.7938
Epoch 67/100
781/781 [==============================] - 81s 103ms/step - loss: 0.5024 -
```

```
accuracy: 0.8243 - val_loss: 0.6215 - val_accuracy: 0.7940
Epoch 68/100
781/781 [==============================] - 81s 104ms/step - loss: 0.5073 -
accuracy: 0.8217 - val_loss: 0.6170 - val_accuracy: 0.7968
Epoch 69/100
781/781 [==============================] - 81s 104ms/step - loss: 0.5055 -
accuracy: 0.8236 - val_loss: 0.6173 - val_accuracy: 0.8012
Epoch 70/100
781/781 [==============================] - 81s 103ms/step - loss: 0.4999 -
accuracy: 0.8261 - val_loss: 0.5952 - val_accuracy: 0.8036
Epoch 71/100
781/781 [==============================] - 81s 104ms/step - loss: 0.4950 -
accuracy: 0.8262 - val_loss: 0.6249 - val_accuracy: 0.8002
Epoch 72/100
781/781 [==============================] - 81s 104ms/step - loss: 0.4976 -
accuracy: 0.8248 - val_loss: 0.6057 - val_accuracy: 0.8030
Epoch 73/100
781/781 [==============================] - 81s 104ms/step - loss: 0.4999 -
accuracy: 0.8245 - val_loss: 0.5911 - val_accuracy: 0.8047
Epoch 74/100
781/781 [==============================] - 82s 104ms/step - loss: 0.4922 -
accuracy: 0.8286 - val_loss: 0.6185 - val_accuracy: 0.7998
Epoch 75/100
781/781 [==============================] - 82s 104ms/step - loss: 0.4919 -
accuracy: 0.8285 - val_loss: 0.6140 - val_accuracy: 0.7988
Epoch 76/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4959 -
accuracy: 0.8268 - val_loss: 0.5913 - val_accuracy: 0.8048
Epoch 77/100
781/781 [==============================] - 81s 104ms/step - loss: 0.4906 -
accuracy: 0.8282 - val_loss: 0.6155 - val_accuracy: 0.7995
Epoch 78/100
781/781 [==============================] - 82s 104ms/step - loss: 0.4829 -
accuracy: 0.8311 - val_loss: 0.6373 - val_accuracy: 0.7926
Epoch 79/100
781/781 [==============================] - 82s 104ms/step - loss: 0.4854 -
accuracy: 0.8311 - val_loss: 0.6121 - val_accuracy: 0.8070
Epoch 80/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4932 -
accuracy: 0.8277 - val_loss: 0.6042 - val_accuracy: 0.7975
Epoch 81/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4855 -
accuracy: 0.8305 - val_loss: 0.6228 - val_accuracy: 0.7985
Epoch 82/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4885 -
accuracy: 0.8288 - val_loss: 0.6250 - val_accuracy: 0.8000
Epoch 83/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4873 -
```

```
accuracy: 0.8277 - val_loss: 0.6007 - val_accuracy: 0.8025
Epoch 84/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4810 -
accuracy: 0.8313 - val_loss: 0.6184 - val_accuracy: 0.8033
Epoch 85/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4831 -
accuracy: 0.8326 - val_loss: 0.5962 - val_accuracy: 0.8079
Epoch 86/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4783 -
accuracy: 0.8326 - val_loss: 0.6579 - val_accuracy: 0.7900
Epoch 87/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4857 -
accuracy: 0.8306 - val_loss: 0.6224 - val_accuracy: 0.7983
Epoch 88/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4766 -
accuracy: 0.8333 - val_loss: 0.5836 - val_accuracy: 0.8093
Epoch 89/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4801 -
accuracy: 0.8325 - val_loss: 0.6118 - val_accuracy: 0.8016
Epoch 90/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4741 -
accuracy: 0.8341 - val_loss: 0.6129 - val_accuracy: 0.7985
Epoch 91/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4762 -
accuracy: 0.8337 - val_loss: 0.6415 - val_accuracy: 0.7959
Epoch 92/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4789 -
accuracy: 0.8331 - val_loss: 0.5818 - val_accuracy: 0.8091
Epoch 93/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4756 -
accuracy: 0.8334 - val_loss: 0.6126 - val_accuracy: 0.8028
Epoch 94/100
781/781 [==============================] - 82s 106ms/step - loss: 0.4671 -
accuracy: 0.8360 - val_loss: 0.5984 - val_accuracy: 0.8071
Epoch 95/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4787 -
accuracy: 0.8326 - val_loss: 0.6045 - val_accuracy: 0.8012
Epoch 96/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4728 -
accuracy: 0.8355 - val_loss: 0.6010 - val_accuracy: 0.8085
Epoch 97/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4659 -
accuracy: 0.8363 - val_loss: 0.6209 - val_accuracy: 0.8010
Epoch 98/100
781/781 [==============================] - 83s 106ms/step - loss: 0.4710 -
accuracy: 0.8335 - val_loss: 0.6199 - val_accuracy: 0.8020
Epoch 99/100
781/781 [==============================] - 82s 105ms/step - loss: 0.4673 -
```

```
accuracy: 0.8368 - val_loss: 0.6083 - val_accuracy: 0.8019
Epoch 100/100
781/781 [==============================] - 82s 106ms/step - loss: 0.4633 -
accuracy: 0.8379 - val_loss: 0.5990 - val_accuracy: 0.8085
313/313 - 3s - loss: 0.5990 - accuracy: 0.8085
----->Evaluation of the training process for: adam
```




```
----->Test accuracy for adam : 0.8084999918937683
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: adamax oprimizer
Epoch 1/100
781/781 [==============================] - 76s 97ms/step - loss: 0.4640 -
accuracy: 0.8385 - val_loss: 0.5984 - val_accuracy: 0.8041
Epoch 2/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4629 -
accuracy: 0.8378 - val_loss: 0.6170 - val_accuracy: 0.8024
Epoch 3/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4678 -
accuracy: 0.8355 - val_loss: 0.6155 - val_accuracy: 0.8013
Epoch 4/100
781/781 [==============================] - 75s 97ms/step - loss: 0.4603 -
accuracy: 0.8385 - val_loss: 0.6179 - val_accuracy: 0.7983
Epoch 5/100
```

```
781/781 [==============================] - 75s 96ms/step - loss: 0.4625 -
accuracy: 0.8372 - val_loss: 0.5997 - val_accuracy: 0.8081
Epoch 6/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4592 -
accuracy: 0.8394 - val_loss: 0.6173 - val_accuracy: 0.8025
Epoch 7/100
781/781 [==============================] - 75s 97ms/step - loss: 0.4630 -
accuracy: 0.8367 - val_loss: 0.6221 - val_accuracy: 0.7992
Epoch 8/100
781/781 [==============================] - 76s 97ms/step - loss: 0.4571 -
accuracy: 0.8388 - val_loss: 0.6003 - val_accuracy: 0.8079
Epoch 9/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4655 -
accuracy: 0.8385 - val_loss: 0.5977 - val_accuracy: 0.8076
Epoch 10/100
781/781 [==============================] - 75s 97ms/step - loss: 0.4617 -
accuracy: 0.8388 - val_loss: 0.6212 - val_accuracy: 0.8027
Epoch 11/100
781/781 [==============================] - 75s 97ms/step - loss: 0.4653 -
accuracy: 0.8382 - val_loss: 0.5843 - val_accuracy: 0.8101
Epoch 12/100
781/781 [==============================] - 75s 97ms/step - loss: 0.4611 -
accuracy: 0.8369 - val_loss: 0.6116 - val_accuracy: 0.8049
Epoch 13/100
781/781 [==============================] - 76s 97ms/step - loss: 0.4575 -
accuracy: 0.8412 - val_loss: 0.6027 - val_accuracy: 0.8091
Epoch 14/100
781/781 [==============================] - 76s 97ms/step - loss: 0.4517 -
accuracy: 0.8409 - val_loss: 0.6390 - val_accuracy: 0.7987
Epoch 15/100
781/781 [==============================] - 76s 97ms/step - loss: 0.4491 -
accuracy: 0.8429 - val_loss: 0.6218 - val_accuracy: 0.8037
Epoch 16/100
781/781 [==============================] - 76s 97ms/step - loss: 0.4529 -
accuracy: 0.8409 - val_loss: 0.6222 - val_accuracy: 0.8107
Epoch 17/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4543 -
accuracy: 0.8402 - val_loss: 0.5918 - val_accuracy: 0.8115
Epoch 18/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4515 -
accuracy: 0.8416 - val_loss: 0.6079 - val_accuracy: 0.8006
Epoch 19/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4570 -
accuracy: 0.8404 - val_loss: 0.6370 - val_accuracy: 0.7983
Epoch 20/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4549 -
accuracy: 0.8423 - val_loss: 0.5775 - val_accuracy: 0.8105
Epoch 21/100
```

```
781/781 [==============================] - 73s 94ms/step - loss: 0.4471 -
accuracy: 0.8429 - val_loss: 0.5991 - val_accuracy: 0.8114
Epoch 22/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4525 -
accuracy: 0.8422 - val_loss: 0.5897 - val_accuracy: 0.8099
Epoch 23/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4516 -
accuracy: 0.8416 - val_loss: 0.5619 - val_accuracy: 0.8170
Epoch 24/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4496 -
accuracy: 0.8430 - val_loss: 0.5893 - val_accuracy: 0.8128
Epoch 25/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4487 -
accuracy: 0.8426 - val_loss: 0.6424 - val_accuracy: 0.8025
Epoch 26/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4519 -
accuracy: 0.8412 - val_loss: 0.5623 - val_accuracy: 0.8165
Epoch 27/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4478 -
accuracy: 0.8431 - val_loss: 0.5913 - val_accuracy: 0.8118
Epoch 28/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4491 -
accuracy: 0.8428 - val_loss: 0.6162 - val_accuracy: 0.8071
Epoch 29/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4428 -
accuracy: 0.8455 - val_loss: 0.5922 - val_accuracy: 0.8101
Epoch 30/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4482 -
accuracy: 0.8437 - val_loss: 0.5954 - val_accuracy: 0.8071
Epoch 31/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4395 -
accuracy: 0.8465 - val_loss: 0.6236 - val_accuracy: 0.8044
Epoch 32/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4467 -
accuracy: 0.8439 - val_loss: 0.6085 - val_accuracy: 0.8070
Epoch 33/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4486 -
accuracy: 0.8435 - val_loss: 0.6096 - val_accuracy: 0.8023
Epoch 34/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4437 -
accuracy: 0.8463 - val_loss: 0.6050 - val_accuracy: 0.8021
Epoch 35/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4424 -
accuracy: 0.8465 - val_loss: 0.5886 - val_accuracy: 0.8152
Epoch 36/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4463 -
accuracy: 0.8420 - val_loss: 0.6069 - val_accuracy: 0.8063
Epoch 37/100
```

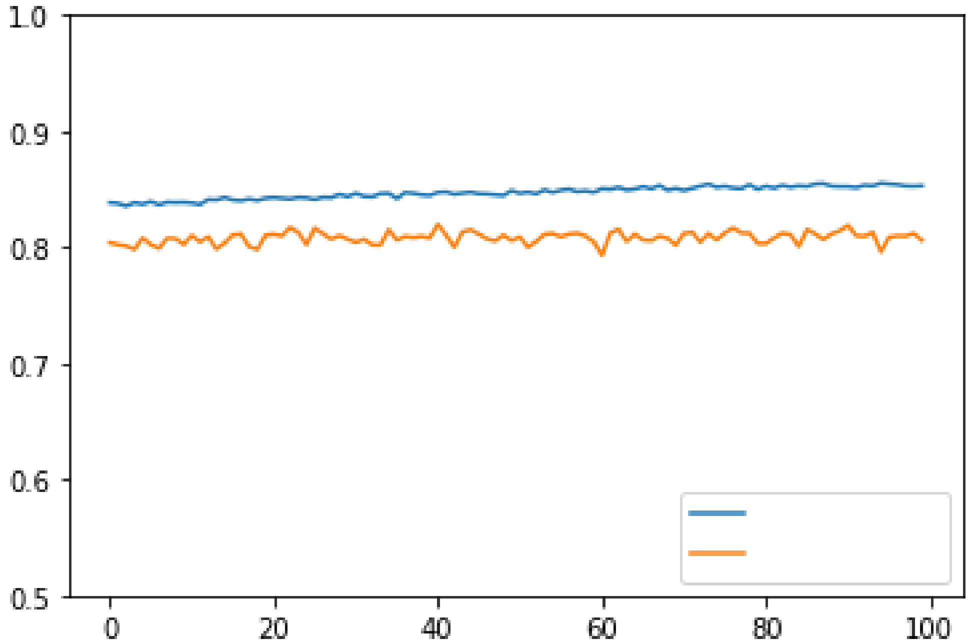
```
781/781 [==============================] - 74s 95ms/step - loss: 0.4406 -
accuracy: 0.8472 - val_loss: 0.5983 - val_accuracy: 0.8095
Epoch 38/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4382 -
accuracy: 0.8466 - val_loss: 0.6038 - val_accuracy: 0.8085
Epoch 39/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4432 -
accuracy: 0.8452 - val_loss: 0.6069 - val_accuracy: 0.8096
Epoch 40/100
781/781 [==============================] - 77s 99ms/step - loss: 0.4452 -
accuracy: 0.8447 - val_loss: 0.6098 - val_accuracy: 0.8081
Epoch 41/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4411 -
accuracy: 0.8473 - val_loss: 0.5881 - val_accuracy: 0.8197
Epoch 42/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4340 -
accuracy: 0.8478 - val_loss: 0.5889 - val_accuracy: 0.8107
Epoch 43/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4410 -
accuracy: 0.8457 - val_loss: 0.6345 - val_accuracy: 0.8002
Epoch 44/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4371 -
accuracy: 0.8467 - val_loss: 0.6087 - val_accuracy: 0.8133
Epoch 45/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4386 -
accuracy: 0.8475 - val_loss: 0.6070 - val_accuracy: 0.8150
Epoch 46/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4356 -
accuracy: 0.8461 - val_loss: 0.6026 - val_accuracy: 0.8115
Epoch 47/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4388 -
accuracy: 0.8459 - val_loss: 0.6035 - val_accuracy: 0.8073
Epoch 48/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4393 -
accuracy: 0.8452 - val_loss: 0.6167 - val_accuracy: 0.8056
Epoch 49/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4382 -
accuracy: 0.8446 - val_loss: 0.5929 - val_accuracy: 0.8105
Epoch 50/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4319 -
accuracy: 0.8488 - val_loss: 0.6243 - val_accuracy: 0.8059
Epoch 51/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4389 -
accuracy: 0.8465 - val_loss: 0.6094 - val_accuracy: 0.8090
Epoch 52/100
781/781 [==============================] - 75s 95ms/step - loss: 0.4349 -
accuracy: 0.8476 - val_loss: 0.6341 - val_accuracy: 0.8002
Epoch 53/100
```

```
781/781 [==============================] - 74s 95ms/step - loss: 0.4371 -
accuracy: 0.8461 - val_loss: 0.6153 - val_accuracy: 0.8052
Epoch 54/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4314 -
accuracy: 0.8494 - val_loss: 0.6065 - val_accuracy: 0.8112
Epoch 55/100
781/781 [==============================] - 75s 95ms/step - loss: 0.4366 -
accuracy: 0.8473 - val_loss: 0.5912 - val_accuracy: 0.8120
Epoch 56/100
781/781 [==============================] - 73s 93ms/step - loss: 0.4318 -
accuracy: 0.8492 - val_loss: 0.6078 - val_accuracy: 0.8092
Epoch 57/100
781/781 [==============================] - 73s 93ms/step - loss: 0.4292 -
accuracy: 0.8501 - val_loss: 0.5965 - val_accuracy: 0.8115
Epoch 58/100
781/781 [==============================] - 73s 93ms/step - loss: 0.4359 -
accuracy: 0.8479 - val_loss: 0.5955 - val_accuracy: 0.8122
Epoch 59/100
781/781 [==============================] - 73s 93ms/step - loss: 0.4290 -
accuracy: 0.8487 - val_loss: 0.5926 - val_accuracy: 0.8099
Epoch 60/100
781/781 [==============================] - 73s 93ms/step - loss: 0.4299 -
accuracy: 0.8473 - val_loss: 0.6232 - val_accuracy: 0.8048
Epoch 61/100
781/781 [==============================] - 73s 93ms/step - loss: 0.4277 -
accuracy: 0.8506 - val_loss: 0.6564 - val_accuracy: 0.7932
Epoch 62/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4283 -
accuracy: 0.8499 - val_loss: 0.6030 - val_accuracy: 0.8123
Epoch 63/100
781/781 [==============================] - 73s 93ms/step - loss: 0.4263 -
accuracy: 0.8517 - val_loss: 0.5751 - val_accuracy: 0.8156
Epoch 64/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4296 -
accuracy: 0.8493 - val_loss: 0.6262 - val_accuracy: 0.8052
Epoch 65/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4285 -
accuracy: 0.8502 - val_loss: 0.6063 - val_accuracy: 0.8114
Epoch 66/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4257 -
accuracy: 0.8523 - val_loss: 0.6287 - val_accuracy: 0.8064
Epoch 67/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4302 -
accuracy: 0.8505 - val_loss: 0.6400 - val_accuracy: 0.8059
Epoch 68/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4201 -
accuracy: 0.8533 - val_loss: 0.6136 - val_accuracy: 0.8097
Epoch 69/100
```

```
781/781 [==============================] - 73s 94ms/step - loss: 0.4324 -
accuracy: 0.8492 - val_loss: 0.6101 - val_accuracy: 0.8079
Epoch 70/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4251 -
accuracy: 0.8507 - val_loss: 0.6246 - val_accuracy: 0.8021
Epoch 71/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4270 -
accuracy: 0.8490 - val_loss: 0.6039 - val_accuracy: 0.8121
Epoch 72/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4236 -
accuracy: 0.8508 - val_loss: 0.6048 - val_accuracy: 0.8131
Epoch 73/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4234 -
accuracy: 0.8528 - val_loss: 0.6241 - val_accuracy: 0.8047
Epoch 74/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4225 -
accuracy: 0.8544 - val_loss: 0.5990 - val_accuracy: 0.8118
Epoch 75/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4288 -
accuracy: 0.8517 - val_loss: 0.6203 - val_accuracy: 0.8065
Epoch 76/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4241 -
accuracy: 0.8526 - val_loss: 0.5861 - val_accuracy: 0.8126
Epoch 77/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4246 -
accuracy: 0.8511 - val_loss: 0.5846 - val_accuracy: 0.8166
Epoch 78/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4226 -
accuracy: 0.8506 - val_loss: 0.6020 - val_accuracy: 0.8125
Epoch 79/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4231 -
accuracy: 0.8540 - val_loss: 0.6119 - val_accuracy: 0.8122
Epoch 80/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4272 -
accuracy: 0.8500 - val_loss: 0.6311 - val_accuracy: 0.8034
Epoch 81/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4223 -
accuracy: 0.8527 - val_loss: 0.6375 - val_accuracy: 0.8033
Epoch 82/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4241 -
accuracy: 0.8510 - val_loss: 0.6162 - val_accuracy: 0.8082
Epoch 83/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4197 -
accuracy: 0.8534 - val_loss: 0.5909 - val_accuracy: 0.8123
Epoch 84/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4239 -
accuracy: 0.8518 - val_loss: 0.5983 - val_accuracy: 0.8110
Epoch 85/100
```

```
781/781 [==============================] - 74s 94ms/step - loss: 0.4202 -
accuracy: 0.8530 - val_loss: 0.6181 - val_accuracy: 0.8008
Epoch 86/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4209 -
accuracy: 0.8524 - val_loss: 0.5909 - val_accuracy: 0.8153
Epoch 87/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4161 -
accuracy: 0.8546 - val_loss: 0.6060 - val_accuracy: 0.8117
Epoch 88/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4166 -
accuracy: 0.8550 - val_loss: 0.6164 - val_accuracy: 0.8068
Epoch 89/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4197 -
accuracy: 0.8527 - val_loss: 0.6075 - val_accuracy: 0.8117
Epoch 90/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4199 -
accuracy: 0.8523 - val_loss: 0.6100 - val_accuracy: 0.8142
Epoch 91/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4206 -
accuracy: 0.8524 - val_loss: 0.5771 - val_accuracy: 0.8189
Epoch 92/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4215 -
accuracy: 0.8515 - val_loss: 0.6014 - val_accuracy: 0.8101
Epoch 93/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4133 -
accuracy: 0.8535 - val_loss: 0.6043 - val_accuracy: 0.8096
Epoch 94/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4188 -
accuracy: 0.8529 - val_loss: 0.6007 - val_accuracy: 0.8129
Epoch 95/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4197 -
accuracy: 0.8553 - val_loss: 0.6389 - val_accuracy: 0.7964
Epoch 96/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4129 -
accuracy: 0.8546 - val_loss: 0.6166 - val_accuracy: 0.8088
Epoch 97/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4200 -
accuracy: 0.8541 - val_loss: 0.5992 - val_accuracy: 0.8099
Epoch 98/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4184 -
accuracy: 0.8528 - val_loss: 0.6276 - val_accuracy: 0.8098
Epoch 99/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4190 -
accuracy: 0.8526 - val_loss: 0.6155 - val_accuracy: 0.8120
Epoch 100/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4160 -
accuracy: 0.8530 - val_loss: 0.6203 - val_accuracy: 0.8062
313/313 - 3s - loss: 0.6203 - accuracy: 0.8062
```

```
----->Evaluation of the training process for: adamax
```




```
----->Test accuracy for adamax : 0.8062000274658203
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: adagrad oprimizer
Epoch 1/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4153 -
accuracy: 0.8536 - val_loss: 0.6354 - val_accuracy: 0.8011
Epoch 2/100
781/781 [==============================] - 73s 93ms/step - loss: 0.4144 -
accuracy: 0.8549 - val_loss: 0.5987 - val_accuracy: 0.8103
Epoch 3/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4155 -
accuracy: 0.8546 - val_loss: 0.6133 - val_accuracy: 0.8094
Epoch 4/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4223 -
accuracy: 0.8524 - val_loss: 0.6088 - val_accuracy: 0.8128
Epoch 5/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4123 -
accuracy: 0.8549 - val_loss: 0.6328 - val_accuracy: 0.8071
Epoch 6/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4132 -
accuracy: 0.8559 - val_loss: 0.6161 - val_accuracy: 0.8130
```

```
Epoch 7/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4142 -
accuracy: 0.8551 - val_loss: 0.5959 - val_accuracy: 0.8135
Epoch 8/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4196 -
accuracy: 0.8537 - val_loss: 0.6081 - val_accuracy: 0.8124
Epoch 9/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4115 -
accuracy: 0.8573 - val_loss: 0.6163 - val_accuracy: 0.8103
Epoch 10/100
781/781 [==============================] - 73s 94ms/step - loss: 0.4162 -
accuracy: 0.8542 - val_loss: 0.6654 - val_accuracy: 0.7981
Epoch 11/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4050 -
accuracy: 0.8564 - val_loss: 0.6086 - val_accuracy: 0.8128
Epoch 12/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4158 -
accuracy: 0.8546 - val_loss: 0.6053 - val_accuracy: 0.8092
Epoch 13/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4138 -
accuracy: 0.8558 - val_loss: 0.5947 - val_accuracy: 0.8139
Epoch 14/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4118 -
accuracy: 0.8546 - val_loss: 0.5924 - val_accuracy: 0.8093
Epoch 15/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4097 -
accuracy: 0.8576 - val_loss: 0.6156 - val_accuracy: 0.8107
Epoch 16/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4119 -
accuracy: 0.8557 - val_loss: 0.6318 - val_accuracy: 0.8088
Epoch 17/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4133 -
accuracy: 0.8543 - val_loss: 0.6089 - val_accuracy: 0.8153
Epoch 18/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4148 -
accuracy: 0.8558 - val_loss: 0.6182 - val_accuracy: 0.8085
Epoch 19/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4069 -
accuracy: 0.8559 - val_loss: 0.6351 - val_accuracy: 0.7996
Epoch 20/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4063 -
accuracy: 0.8561 - val_loss: 0.6014 - val_accuracy: 0.8160
Epoch 21/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4139 -
accuracy: 0.8555 - val_loss: 0.6026 - val_accuracy: 0.8156
Epoch 22/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4088 -
accuracy: 0.8577 - val_loss: 0.6017 - val_accuracy: 0.8162
```

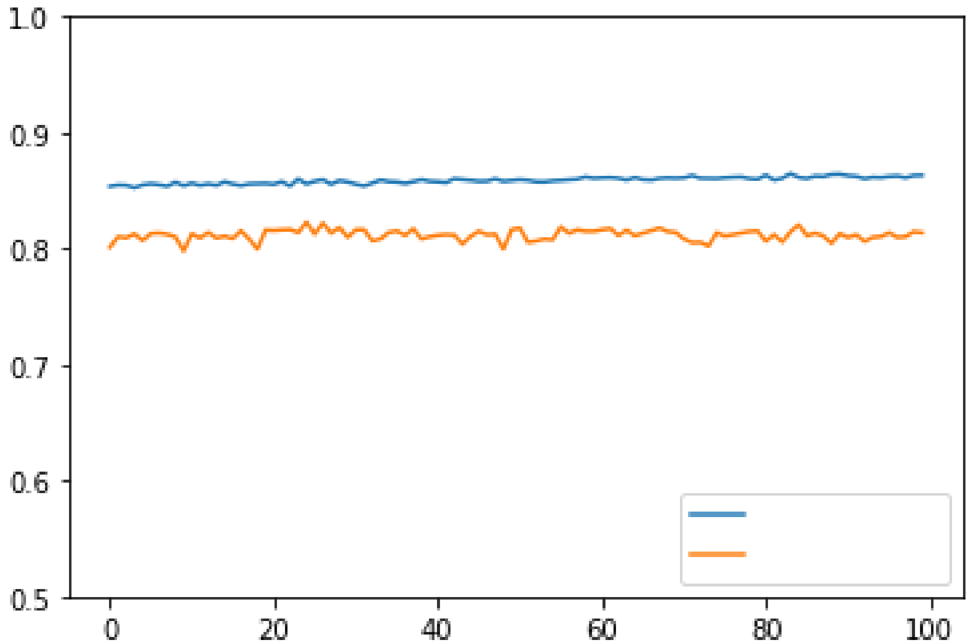
```
Epoch 23/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4146 -
accuracy: 0.8540 - val_loss: 0.5771 - val_accuracy: 0.8164
Epoch 24/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4033 -
accuracy: 0.8601 - val_loss: 0.5996 - val_accuracy: 0.8139
Epoch 25/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4101 -
accuracy: 0.8557 - val_loss: 0.5725 - val_accuracy: 0.8227
Epoch 26/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4079 -
accuracy: 0.8581 - val_loss: 0.6063 - val_accuracy: 0.8126
Epoch 27/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4035 -
accuracy: 0.8595 - val_loss: 0.5832 - val_accuracy: 0.8220
Epoch 28/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4128 -
accuracy: 0.8555 - val_loss: 0.6092 - val_accuracy: 0.8133
Epoch 29/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4071 -
accuracy: 0.8587 - val_loss: 0.5879 - val_accuracy: 0.8181
Epoch 30/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4060 -
accuracy: 0.8575 - val_loss: 0.6128 - val_accuracy: 0.8099
Epoch 31/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4084 -
accuracy: 0.8555 - val_loss: 0.5930 - val_accuracy: 0.8162
Epoch 32/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4125 -
accuracy: 0.8540 - val_loss: 0.5970 - val_accuracy: 0.8163
Epoch 33/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4057 -
accuracy: 0.8564 - val_loss: 0.6282 - val_accuracy: 0.8068
Epoch 34/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4046 -
accuracy: 0.8592 - val_loss: 0.6133 - val_accuracy: 0.8084
Epoch 35/100
781/781 [==============================] - 75s 95ms/step - loss: 0.4077 -
accuracy: 0.8578 - val_loss: 0.5820 - val_accuracy: 0.8139
Epoch 36/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4112 -
accuracy: 0.8575 - val_loss: 0.5928 - val_accuracy: 0.8151
Epoch 37/100
781/781 [==============================] - 79s 101ms/step - loss: 0.4098 -
accuracy: 0.8562 - val_loss: 0.6080 - val_accuracy: 0.8115
Epoch 38/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4081 -
accuracy: 0.8577 - val_loss: 0.5764 - val_accuracy: 0.8170
```

```
Epoch 39/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4034 -
accuracy: 0.8595 - val_loss: 0.6208 - val_accuracy: 0.8085
Epoch 40/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4043 -
accuracy: 0.8581 - val_loss: 0.6130 - val_accuracy: 0.8103
Epoch 41/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4066 -
accuracy: 0.8577 - val_loss: 0.6045 - val_accuracy: 0.8116
Epoch 42/100
781/781 [==============================] - 75s 95ms/step - loss: 0.4072 -
accuracy: 0.8570 - val_loss: 0.5904 - val_accuracy: 0.8122
Epoch 43/100
781/781 [==============================] - 75s 95ms/step - loss: 0.4027 -
accuracy: 0.8603 - val_loss: 0.5971 - val_accuracy: 0.8118
Epoch 44/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4027 -
accuracy: 0.8596 - val_loss: 0.6524 - val_accuracy: 0.8041
Epoch 45/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4043 -
accuracy: 0.8589 - val_loss: 0.5985 - val_accuracy: 0.8107
Epoch 46/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4037 -
accuracy: 0.8581 - val_loss: 0.6223 - val_accuracy: 0.8149
Epoch 47/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4052 -
accuracy: 0.8582 - val_loss: 0.6057 - val_accuracy: 0.8121
Epoch 48/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3974 -
accuracy: 0.8602 - val_loss: 0.6087 - val_accuracy: 0.8125
Epoch 49/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4016 -
accuracy: 0.8581 - val_loss: 0.6671 - val_accuracy: 0.7997
Epoch 50/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4052 -
accuracy: 0.8588 - val_loss: 0.5991 - val_accuracy: 0.8164
Epoch 51/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4039 -
accuracy: 0.8595 - val_loss: 0.5890 - val_accuracy: 0.8177
Epoch 52/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4023 -
accuracy: 0.8589 - val_loss: 0.6234 - val_accuracy: 0.8055
Epoch 53/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4063 -
accuracy: 0.8576 - val_loss: 0.6222 - val_accuracy: 0.8067
Epoch 54/100
781/781 [==============================] - 75s 96ms/step - loss: 0.4064 -
accuracy: 0.8576 - val_loss: 0.6098 - val_accuracy: 0.8083
```

```
Epoch 55/100
781/781 [==============================] - 75s 97ms/step - loss: 0.4019 -
accuracy: 0.8584 - val_loss: 0.6294 - val_accuracy: 0.8077
Epoch 56/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4005 -
accuracy: 0.8589 - val_loss: 0.5960 - val_accuracy: 0.8187
Epoch 57/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4027 -
accuracy: 0.8594 - val_loss: 0.6088 - val_accuracy: 0.8132
Epoch 58/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3969 -
accuracy: 0.8599 - val_loss: 0.6052 - val_accuracy: 0.8162
Epoch 59/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3975 -
accuracy: 0.8619 - val_loss: 0.5929 - val_accuracy: 0.8149
Epoch 60/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3995 -
accuracy: 0.8607 - val_loss: 0.5981 - val_accuracy: 0.8149
Epoch 61/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3992 -
accuracy: 0.8611 - val_loss: 0.6103 - val_accuracy: 0.8163
Epoch 62/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3982 -
accuracy: 0.8614 - val_loss: 0.6053 - val_accuracy: 0.8173
Epoch 63/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3969 -
accuracy: 0.8607 - val_loss: 0.6038 - val_accuracy: 0.8115
Epoch 64/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4031 -
accuracy: 0.8592 - val_loss: 0.6111 - val_accuracy: 0.8158
Epoch 65/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3979 -
accuracy: 0.8613 - val_loss: 0.6300 - val_accuracy: 0.8112
Epoch 66/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4027 -
accuracy: 0.8595 - val_loss: 0.6338 - val_accuracy: 0.8134
Epoch 67/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4039 -
accuracy: 0.8591 - val_loss: 0.6037 - val_accuracy: 0.8154
Epoch 68/100
781/781 [==============================] - 74s 94ms/step - loss: 0.4015 -
accuracy: 0.8606 - val_loss: 0.6081 - val_accuracy: 0.8177
Epoch 69/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3956 -
accuracy: 0.8610 - val_loss: 0.6071 - val_accuracy: 0.8147
Epoch 70/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3963 -
accuracy: 0.8607 - val_loss: 0.5945 - val_accuracy: 0.8139
```

```
Epoch 71/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3999 -
accuracy: 0.8611 - val_loss: 0.6176 - val_accuracy: 0.8084
Epoch 72/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3946 -
accuracy: 0.8630 - val_loss: 0.6336 - val_accuracy: 0.8053
Epoch 73/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3948 -
accuracy: 0.8607 - val_loss: 0.6395 - val_accuracy: 0.8055
Epoch 74/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3999 -
accuracy: 0.8606 - val_loss: 0.6331 - val_accuracy: 0.8024
Epoch 75/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3997 -
accuracy: 0.8605 - val_loss: 0.5979 - val_accuracy: 0.8138
Epoch 76/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3991 -
accuracy: 0.8611 - val_loss: 0.6228 - val_accuracy: 0.8109
Epoch 77/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3943 -
accuracy: 0.8616 - val_loss: 0.6184 - val_accuracy: 0.8122
Epoch 78/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3946 -
accuracy: 0.8618 - val_loss: 0.6296 - val_accuracy: 0.8135
Epoch 79/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4031 -
accuracy: 0.8603 - val_loss: 0.6101 - val_accuracy: 0.8147
Epoch 80/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4004 -
accuracy: 0.8599 - val_loss: 0.6019 - val_accuracy: 0.8150
Epoch 81/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3939 -
accuracy: 0.8634 - val_loss: 0.6456 - val_accuracy: 0.8065
Epoch 82/100
781/781 [==============================] - 74s 95ms/step - loss: 0.4014 -
accuracy: 0.8591 - val_loss: 0.6010 - val_accuracy: 0.8121
Epoch 83/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3990 -
accuracy: 0.8607 - val_loss: 0.6237 - val_accuracy: 0.8058
Epoch 84/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3929 -
accuracy: 0.8647 - val_loss: 0.6058 - val_accuracy: 0.8146
Epoch 85/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3957 -
accuracy: 0.8614 - val_loss: 0.5836 - val_accuracy: 0.8205
Epoch 86/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3997 -
accuracy: 0.8605 - val_loss: 0.6295 - val_accuracy: 0.8112
```

```
Epoch 87/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3925 -
accuracy: 0.8625 - val_loss: 0.6153 - val_accuracy: 0.8134
Epoch 88/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3969 -
accuracy: 0.8621 - val_loss: 0.6093 - val_accuracy: 0.8110
Epoch 89/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3900 -
accuracy: 0.8640 - val_loss: 0.6401 - val_accuracy: 0.8047
Epoch 90/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3924 -
accuracy: 0.8642 - val_loss: 0.6155 - val_accuracy: 0.8130
Epoch 91/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3960 -
accuracy: 0.8629 - val_loss: 0.6356 - val_accuracy: 0.8097
Epoch 92/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3937 -
accuracy: 0.8623 - val_loss: 0.6311 - val_accuracy: 0.8120
Epoch 93/100
781/781 [==============================] - 75s 95ms/step - loss: 0.4018 -
accuracy: 0.8603 - val_loss: 0.6341 - val_accuracy: 0.8063
Epoch 94/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3956 -
accuracy: 0.8616 - val_loss: 0.6125 - val_accuracy: 0.8098
Epoch 95/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3938 -
accuracy: 0.8613 - val_loss: 0.6268 - val_accuracy: 0.8104
Epoch 96/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3956 -
accuracy: 0.8622 - val_loss: 0.6192 - val_accuracy: 0.8137
Epoch 97/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3948 -
accuracy: 0.8626 - val_loss: 0.6309 - val_accuracy: 0.8098
Epoch 98/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3931 -
accuracy: 0.8611 - val_loss: 0.6131 - val_accuracy: 0.8104
Epoch 99/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3917 -
accuracy: 0.8630 - val_loss: 0.5844 - val_accuracy: 0.8145
Epoch 100/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3952 -
accuracy: 0.8631 - val_loss: 0.6144 - val_accuracy: 0.8137
313/313 - 3s - loss: 0.6144 - accuracy: 0.8137
----->Evaluation of the training process for: adagrad
```




```
----->Test accuracy for adagrad : 0.8137000203132629
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: adadelta oprimizer
Epoch 1/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3964 -
accuracy: 0.8614 - val_loss: 0.5913 - val_accuracy: 0.8154
Epoch 2/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3937 -
accuracy: 0.8626 - val_loss: 0.6400 - val_accuracy: 0.8078
Epoch 3/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3923 -
accuracy: 0.8632 - val_loss: 0.6253 - val_accuracy: 0.8116
Epoch 4/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3950 -
accuracy: 0.8607 - val_loss: 0.6169 - val_accuracy: 0.8140
Epoch 5/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3941 -
accuracy: 0.8620 - val_loss: 0.5840 - val_accuracy: 0.8167
Epoch 6/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3918 -
accuracy: 0.8631 - val_loss: 0.6079 - val_accuracy: 0.8148
Epoch 7/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3942 -
```

```
accuracy: 0.8613 - val_loss: 0.6222 - val_accuracy: 0.8073
Epoch 8/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3920 -
accuracy: 0.8644 - val_loss: 0.5817 - val_accuracy: 0.8192
Epoch 9/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3899 -
accuracy: 0.8636 - val_loss: 0.6087 - val_accuracy: 0.8168
Epoch 10/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3938 -
accuracy: 0.8631 - val_loss: 0.5974 - val_accuracy: 0.8113
Epoch 11/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3914 -
accuracy: 0.8635 - val_loss: 0.6321 - val_accuracy: 0.8136
Epoch 12/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3936 -
accuracy: 0.8622 - val_loss: 0.6596 - val_accuracy: 0.8037
Epoch 13/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3885 -
accuracy: 0.8672 - val_loss: 0.6256 - val_accuracy: 0.8086
Epoch 14/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3889 -
accuracy: 0.8644 - val_loss: 0.6144 - val_accuracy: 0.8156
Epoch 15/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3914 -
accuracy: 0.8635 - val_loss: 0.6192 - val_accuracy: 0.8129
Epoch 16/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3875 -
accuracy: 0.8633 - val_loss: 0.6114 - val_accuracy: 0.8141
Epoch 17/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3908 -
accuracy: 0.8648 - val_loss: 0.5983 - val_accuracy: 0.8189
Epoch 18/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3928 -
accuracy: 0.8627 - val_loss: 0.5943 - val_accuracy: 0.8122
Epoch 19/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3902 -
accuracy: 0.8631 - val_loss: 0.6385 - val_accuracy: 0.8079
Epoch 20/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3889 -
accuracy: 0.8633 - val_loss: 0.6048 - val_accuracy: 0.8125
Epoch 21/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3947 -
accuracy: 0.8621 - val_loss: 0.6036 - val_accuracy: 0.8171
Epoch 22/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3859 -
accuracy: 0.8665 - val_loss: 0.6113 - val_accuracy: 0.8097
Epoch 23/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3939 -
```

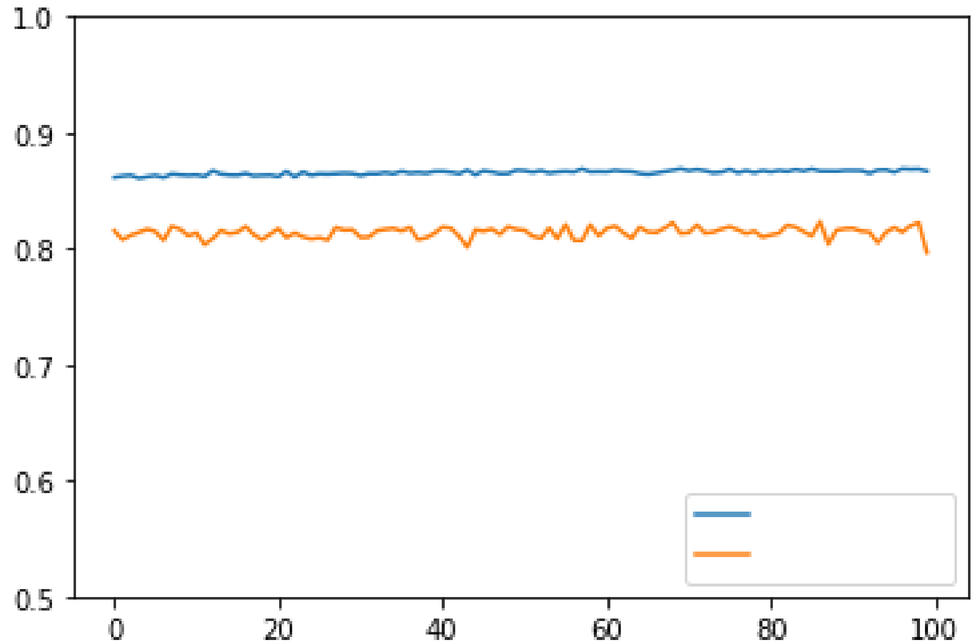
```
accuracy: 0.8617 - val_loss: 0.6087 - val_accuracy: 0.8136
Epoch 24/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3875 -
accuracy: 0.8661 - val_loss: 0.6233 - val_accuracy: 0.8102
Epoch 25/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3892 -
accuracy: 0.8633 - val_loss: 0.6019 - val_accuracy: 0.8084
Epoch 26/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3893 -
accuracy: 0.8645 - val_loss: 0.6262 - val_accuracy: 0.8095
Epoch 27/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3912 -
accuracy: 0.8642 - val_loss: 0.6371 - val_accuracy: 0.8075
Epoch 28/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3917 -
accuracy: 0.8646 - val_loss: 0.5871 - val_accuracy: 0.8182
Epoch 29/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3851 -
accuracy: 0.8651 - val_loss: 0.6067 - val_accuracy: 0.8160
Epoch 30/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3877 -
accuracy: 0.8647 - val_loss: 0.5936 - val_accuracy: 0.8163
Epoch 31/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3906 -
accuracy: 0.8630 - val_loss: 0.6605 - val_accuracy: 0.8098
Epoch 32/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3875 -
accuracy: 0.8646 - val_loss: 0.6177 - val_accuracy: 0.8100
Epoch 33/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3862 -
accuracy: 0.8645 - val_loss: 0.6142 - val_accuracy: 0.8154
Epoch 34/100
781/781 [==============================] - 77s 99ms/step - loss: 0.3847 -
accuracy: 0.8656 - val_loss: 0.6120 - val_accuracy: 0.8166
Epoch 35/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3891 -
accuracy: 0.8645 - val_loss: 0.6139 - val_accuracy: 0.8174
Epoch 36/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3839 -
accuracy: 0.8667 - val_loss: 0.6223 - val_accuracy: 0.8153
Epoch 37/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3875 -
accuracy: 0.8651 - val_loss: 0.5997 - val_accuracy: 0.8183
Epoch 38/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3875 -
accuracy: 0.8656 - val_loss: 0.6168 - val_accuracy: 0.8079
Epoch 39/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3856 -
```

```
accuracy: 0.8650 - val_loss: 0.6396 - val_accuracy: 0.8095
Epoch 40/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3849 -
accuracy: 0.8666 - val_loss: 0.6213 - val_accuracy: 0.8132
Epoch 41/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3837 -
accuracy: 0.8667 - val_loss: 0.5960 - val_accuracy: 0.8186
Epoch 42/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3841 -
accuracy: 0.8659 - val_loss: 0.6164 - val_accuracy: 0.8176
Epoch 43/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3883 -
accuracy: 0.8646 - val_loss: 0.6136 - val_accuracy: 0.8109
Epoch 44/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3830 -
accuracy: 0.8677 - val_loss: 0.6512 - val_accuracy: 0.8014
Epoch 45/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3862 -
accuracy: 0.8637 - val_loss: 0.6054 - val_accuracy: 0.8161
Epoch 46/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3854 -
accuracy: 0.8671 - val_loss: 0.6075 - val_accuracy: 0.8151
Epoch 47/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3847 -
accuracy: 0.8661 - val_loss: 0.5976 - val_accuracy: 0.8169
Epoch 48/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3861 -
accuracy: 0.8646 - val_loss: 0.6383 - val_accuracy: 0.8125
Epoch 49/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3880 -
accuracy: 0.8648 - val_loss: 0.6008 - val_accuracy: 0.8189
Epoch 50/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3775 -
accuracy: 0.8677 - val_loss: 0.5972 - val_accuracy: 0.8166
Epoch 51/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3829 -
accuracy: 0.8673 - val_loss: 0.6044 - val_accuracy: 0.8158
Epoch 52/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3835 -
accuracy: 0.8658 - val_loss: 0.6217 - val_accuracy: 0.8107
Epoch 53/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3818 -
accuracy: 0.8674 - val_loss: 0.6558 - val_accuracy: 0.8091
Epoch 54/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3863 -
accuracy: 0.8652 - val_loss: 0.5993 - val_accuracy: 0.8176
Epoch 55/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3858 -
```

```
accuracy: 0.8662 - val_loss: 0.6244 - val_accuracy: 0.8088
Epoch 56/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3858 -
accuracy: 0.8667 - val_loss: 0.5989 - val_accuracy: 0.8208
Epoch 57/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3791 -
accuracy: 0.8661 - val_loss: 0.6552 - val_accuracy: 0.8072
Epoch 58/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3810 -
accuracy: 0.8689 - val_loss: 0.6316 - val_accuracy: 0.8070
Epoch 59/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3847 -
accuracy: 0.8662 - val_loss: 0.5993 - val_accuracy: 0.8206
Epoch 60/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3807 -
accuracy: 0.8665 - val_loss: 0.6182 - val_accuracy: 0.8112
Epoch 61/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3864 -
accuracy: 0.8662 - val_loss: 0.5996 - val_accuracy: 0.8174
Epoch 62/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3782 -
accuracy: 0.8675 - val_loss: 0.5982 - val_accuracy: 0.8194
Epoch 63/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3831 -
accuracy: 0.8667 - val_loss: 0.6147 - val_accuracy: 0.8138
Epoch 64/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3808 -
accuracy: 0.8665 - val_loss: 0.6160 - val_accuracy: 0.8090
Epoch 65/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3869 -
accuracy: 0.8648 - val_loss: 0.6002 - val_accuracy: 0.8184
Epoch 66/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3891 -
accuracy: 0.8641 - val_loss: 0.6019 - val_accuracy: 0.8144
Epoch 67/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3800 -
accuracy: 0.8651 - val_loss: 0.6190 - val_accuracy: 0.8139
Epoch 68/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3845 -
accuracy: 0.8664 - val_loss: 0.5818 - val_accuracy: 0.8174
Epoch 69/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3794 -
accuracy: 0.8676 - val_loss: 0.5849 - val_accuracy: 0.8224
Epoch 70/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3782 -
accuracy: 0.8689 - val_loss: 0.6152 - val_accuracy: 0.8131
Epoch 71/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3813 -
```

```
accuracy: 0.8673 - val_loss: 0.6303 - val_accuracy: 0.8138
Epoch 72/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3788 -
accuracy: 0.8683 - val_loss: 0.6067 - val_accuracy: 0.8203
Epoch 73/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3859 -
accuracy: 0.8671 - val_loss: 0.6073 - val_accuracy: 0.8133
Epoch 74/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3845 -
accuracy: 0.8653 - val_loss: 0.6083 - val_accuracy: 0.8143
Epoch 75/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3852 -
accuracy: 0.8661 - val_loss: 0.6010 - val_accuracy: 0.8169
Epoch 76/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3786 -
accuracy: 0.8685 - val_loss: 0.6069 - val_accuracy: 0.8189
Epoch 77/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3871 -
accuracy: 0.8654 - val_loss: 0.5951 - val_accuracy: 0.8161
Epoch 78/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3775 -
accuracy: 0.8674 - val_loss: 0.6046 - val_accuracy: 0.8130
Epoch 79/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3808 -
accuracy: 0.8655 - val_loss: 0.6399 - val_accuracy: 0.8150
Epoch 80/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3779 -
accuracy: 0.8674 - val_loss: 0.6227 - val_accuracy: 0.8100
Epoch 81/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3826 -
accuracy: 0.8661 - val_loss: 0.6258 - val_accuracy: 0.8119
Epoch 82/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3833 -
accuracy: 0.8675 - val_loss: 0.6316 - val_accuracy: 0.8134
Epoch 83/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3821 -
accuracy: 0.8665 - val_loss: 0.5841 - val_accuracy: 0.8202
Epoch 84/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3805 -
accuracy: 0.8683 - val_loss: 0.6026 - val_accuracy: 0.8185
Epoch 85/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3808 -
accuracy: 0.8670 - val_loss: 0.6139 - val_accuracy: 0.8146
Epoch 86/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3794 -
accuracy: 0.8687 - val_loss: 0.6178 - val_accuracy: 0.8111
Epoch 87/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3771 -
```

```
accuracy: 0.8669 - val_loss: 0.5924 - val_accuracy: 0.8232
Epoch 88/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3804 -
accuracy: 0.8669 - val_loss: 0.6399 - val_accuracy: 0.8040
Epoch 89/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3803 -
accuracy: 0.8667 - val_loss: 0.6080 - val_accuracy: 0.8159
Epoch 90/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3820 -
accuracy: 0.8674 - val_loss: 0.6117 - val_accuracy: 0.8171
Epoch 91/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3818 -
accuracy: 0.8675 - val_loss: 0.6204 - val_accuracy: 0.8175
Epoch 92/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3764 -
accuracy: 0.8673 - val_loss: 0.6290 - val_accuracy: 0.8150
Epoch 93/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3792 -
accuracy: 0.8650 - val_loss: 0.6260 - val_accuracy: 0.8143
Epoch 94/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3792 -
accuracy: 0.8678 - val_loss: 0.6373 - val_accuracy: 0.8052
Epoch 95/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3788 -
accuracy: 0.8680 - val_loss: 0.6124 - val_accuracy: 0.8140
Epoch 96/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3823 -
accuracy: 0.8659 - val_loss: 0.5953 - val_accuracy: 0.8182
Epoch 97/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3757 -
accuracy: 0.8689 - val_loss: 0.6094 - val_accuracy: 0.8143
Epoch 98/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3799 -
accuracy: 0.8684 - val_loss: 0.6013 - val_accuracy: 0.8196
Epoch 99/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3771 -
accuracy: 0.8686 - val_loss: 0.5986 - val_accuracy: 0.8224
Epoch 100/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3804 -
accuracy: 0.8668 - val_loss: 0.6963 - val_accuracy: 0.7969
313/313 - 3s - loss: 0.6963 - accuracy: 0.7969
----->Evaluation of the training process for: adadelta
```




```
----->Test accuracy for adadelta : 0.7968999743461609
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: RMSprop oprimizer
Epoch 1/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3780 -
accuracy: 0.8673 - val_loss: 0.6001 - val_accuracy: 0.8173
Epoch 2/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3807 -
accuracy: 0.8662 - val_loss: 0.6100 - val_accuracy: 0.8139
Epoch 3/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3823 -
accuracy: 0.8693 - val_loss: 0.5949 - val_accuracy: 0.8199
Epoch 4/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3759 -
accuracy: 0.8689 - val_loss: 0.5904 - val_accuracy: 0.8178
Epoch 5/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3800 -
accuracy: 0.8684 - val_loss: 0.6075 - val_accuracy: 0.8157
Epoch 6/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3804 -
accuracy: 0.8656 - val_loss: 0.6203 - val_accuracy: 0.8138
Epoch 7/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3785 -
```

```
accuracy: 0.8682 - val_loss: 0.6285 - val_accuracy: 0.8082
Epoch 8/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3775 -
accuracy: 0.8677 - val_loss: 0.6725 - val_accuracy: 0.8048
Epoch 9/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3690 -
accuracy: 0.8708 - val_loss: 0.6296 - val_accuracy: 0.8137
Epoch 10/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3802 -
accuracy: 0.8678 - val_loss: 0.6258 - val_accuracy: 0.8099
Epoch 11/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3782 -
accuracy: 0.8688 - val_loss: 0.6465 - val_accuracy: 0.8093
Epoch 12/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3728 -
accuracy: 0.8695 - val_loss: 0.6215 - val_accuracy: 0.8098
Epoch 13/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3710 -
accuracy: 0.8707 - val_loss: 0.6071 - val_accuracy: 0.8170
Epoch 14/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3788 -
accuracy: 0.8690 - val_loss: 0.6092 - val_accuracy: 0.8186
Epoch 15/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3768 -
accuracy: 0.8686 - val_loss: 0.6202 - val_accuracy: 0.8119
Epoch 16/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3783 -
accuracy: 0.8684 - val_loss: 0.5997 - val_accuracy: 0.8204
Epoch 17/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3732 -
accuracy: 0.8688 - val_loss: 0.5925 - val_accuracy: 0.8191
Epoch 18/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3779 -
accuracy: 0.8679 - val_loss: 0.6091 - val_accuracy: 0.8151
Epoch 19/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3725 -
accuracy: 0.8701 - val_loss: 0.6084 - val_accuracy: 0.8145
Epoch 20/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3708 -
accuracy: 0.8695 - val_loss: 0.6125 - val_accuracy: 0.8169
Epoch 21/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3800 -
accuracy: 0.8694 - val_loss: 0.6268 - val_accuracy: 0.8106
Epoch 22/100
781/781 [==============================] - 76s 97ms/step - loss: 0.3768 -
accuracy: 0.8687 - val_loss: 0.6002 - val_accuracy: 0.8179
Epoch 23/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3749 -
```

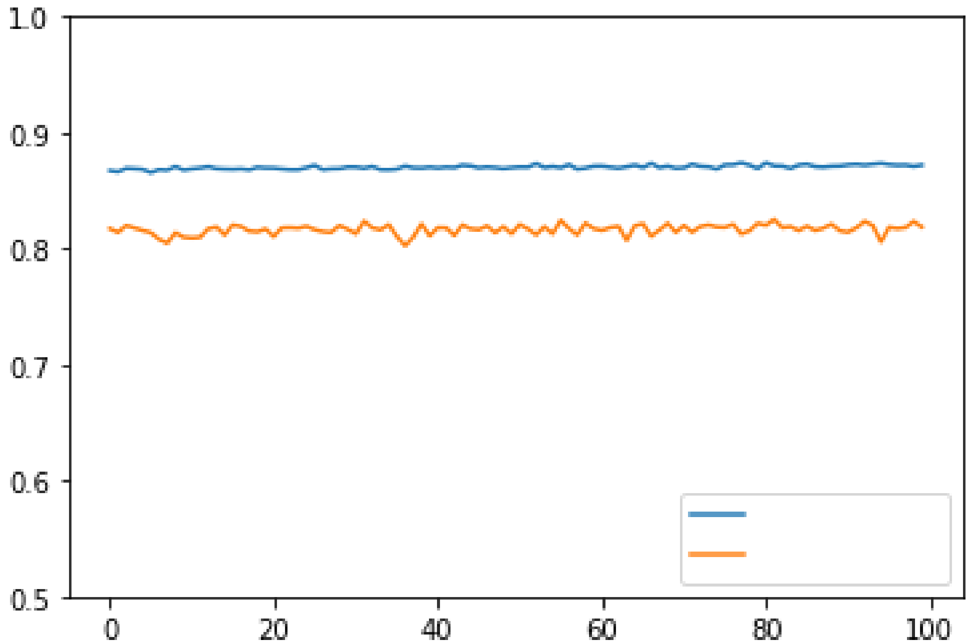
```
accuracy: 0.8681 - val_loss: 0.6118 - val_accuracy: 0.8181
Epoch 24/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3790 -
accuracy: 0.8681 - val_loss: 0.6007 - val_accuracy: 0.8176
Epoch 25/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3764 -
accuracy: 0.8698 - val_loss: 0.5957 - val_accuracy: 0.8192
Epoch 26/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3684 -
accuracy: 0.8718 - val_loss: 0.5955 - val_accuracy: 0.8163
Epoch 27/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3762 -
accuracy: 0.8680 - val_loss: 0.6086 - val_accuracy: 0.8146
Epoch 28/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3706 -
accuracy: 0.8688 - val_loss: 0.6120 - val_accuracy: 0.8140
Epoch 29/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3756 -
accuracy: 0.8690 - val_loss: 0.5935 - val_accuracy: 0.8198
Epoch 30/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3749 -
accuracy: 0.8703 - val_loss: 0.6065 - val_accuracy: 0.8167
Epoch 31/100
781/781 [==============================] - 75s 97ms/step - loss: 0.3734 -
accuracy: 0.8703 - val_loss: 0.6264 - val_accuracy: 0.8128
Epoch 32/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3723 -
accuracy: 0.8693 - val_loss: 0.5837 - val_accuracy: 0.8236
Epoch 33/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3710 -
accuracy: 0.8708 - val_loss: 0.5981 - val_accuracy: 0.8179
Epoch 34/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3746 -
accuracy: 0.8681 - val_loss: 0.6070 - val_accuracy: 0.8162
Epoch 35/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3730 -
accuracy: 0.8680 - val_loss: 0.6134 - val_accuracy: 0.8207
Epoch 36/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3772 -
accuracy: 0.8685 - val_loss: 0.6250 - val_accuracy: 0.8106
Epoch 37/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3697 -
accuracy: 0.8712 - val_loss: 0.6224 - val_accuracy: 0.8026
Epoch 38/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3739 -
accuracy: 0.8696 - val_loss: 0.6135 - val_accuracy: 0.8100
Epoch 39/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3715 -
```

```
accuracy: 0.8695 - val_loss: 0.5807 - val_accuracy: 0.8212
Epoch 40/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3722 -
accuracy: 0.8702 - val_loss: 0.6050 - val_accuracy: 0.8112
Epoch 41/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3751 -
accuracy: 0.8695 - val_loss: 0.5969 - val_accuracy: 0.8180
Epoch 42/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3695 -
accuracy: 0.8702 - val_loss: 0.6048 - val_accuracy: 0.8179
Epoch 43/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3696 -
accuracy: 0.8698 - val_loss: 0.6390 - val_accuracy: 0.8115
Epoch 44/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3685 -
accuracy: 0.8720 - val_loss: 0.6009 - val_accuracy: 0.8204
Epoch 45/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3685 -
accuracy: 0.8714 - val_loss: 0.6027 - val_accuracy: 0.8170
Epoch 46/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3702 -
accuracy: 0.8696 - val_loss: 0.6144 - val_accuracy: 0.8164
Epoch 47/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3744 -
accuracy: 0.8703 - val_loss: 0.6098 - val_accuracy: 0.8195
Epoch 48/100
781/781 [==============================] - 75s 95ms/step - loss: 0.3724 -
accuracy: 0.8698 - val_loss: 0.6371 - val_accuracy: 0.8137
Epoch 49/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3734 -
accuracy: 0.8689 - val_loss: 0.6121 - val_accuracy: 0.8189
Epoch 50/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3694 -
accuracy: 0.8698 - val_loss: 0.6311 - val_accuracy: 0.8129
Epoch 51/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3730 -
accuracy: 0.8703 - val_loss: 0.5968 - val_accuracy: 0.8208
Epoch 52/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3725 -
accuracy: 0.8702 - val_loss: 0.6073 - val_accuracy: 0.8169
Epoch 53/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3680 -
accuracy: 0.8734 - val_loss: 0.6201 - val_accuracy: 0.8120
Epoch 54/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3715 -
accuracy: 0.8699 - val_loss: 0.6038 - val_accuracy: 0.8189
Epoch 55/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3755 -
```

```
accuracy: 0.8709 - val_loss: 0.6017 - val_accuracy: 0.8128
Epoch 56/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3719 -
accuracy: 0.8697 - val_loss: 0.5876 - val_accuracy: 0.8244
Epoch 57/100
781/781 [==============================] - 73s 93ms/step - loss: 0.3691 -
accuracy: 0.8725 - val_loss: 0.6115 - val_accuracy: 0.8172
Epoch 58/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3739 -
accuracy: 0.8685 - val_loss: 0.6212 - val_accuracy: 0.8119
Epoch 59/100
781/781 [==============================] - 73s 93ms/step - loss: 0.3747 -
accuracy: 0.8694 - val_loss: 0.5803 - val_accuracy: 0.8220
Epoch 60/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3671 -
accuracy: 0.8711 - val_loss: 0.6316 - val_accuracy: 0.8172
Epoch 61/100
781/781 [==============================] - 73s 93ms/step - loss: 0.3699 -
accuracy: 0.8712 - val_loss: 0.6269 - val_accuracy: 0.8157
Epoch 62/100
781/781 [==============================] - 73s 93ms/step - loss: 0.3694 -
accuracy: 0.8702 - val_loss: 0.6097 - val_accuracy: 0.8181
Epoch 63/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3731 -
accuracy: 0.8693 - val_loss: 0.6235 - val_accuracy: 0.8192
Epoch 64/100
781/781 [==============================] - 73s 93ms/step - loss: 0.3704 -
accuracy: 0.8706 - val_loss: 0.6526 - val_accuracy: 0.8072
Epoch 65/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3686 -
accuracy: 0.8721 - val_loss: 0.6132 - val_accuracy: 0.8197
Epoch 66/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3754 -
accuracy: 0.8700 - val_loss: 0.5965 - val_accuracy: 0.8214
Epoch 67/100
781/781 [==============================] - 73s 93ms/step - loss: 0.3668 -
accuracy: 0.8736 - val_loss: 0.6342 - val_accuracy: 0.8109
Epoch 68/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3708 -
accuracy: 0.8698 - val_loss: 0.6039 - val_accuracy: 0.8159
Epoch 69/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3696 -
accuracy: 0.8712 - val_loss: 0.6031 - val_accuracy: 0.8217
Epoch 70/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3736 -
accuracy: 0.8693 - val_loss: 0.6263 - val_accuracy: 0.8141
Epoch 71/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3746 -
```

```
accuracy: 0.8697 - val_loss: 0.6107 - val_accuracy: 0.8198
Epoch 72/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3650 -
accuracy: 0.8728 - val_loss: 0.6237 - val_accuracy: 0.8142
Epoch 73/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3693 -
accuracy: 0.8710 - val_loss: 0.5988 - val_accuracy: 0.8190
Epoch 74/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3712 -
accuracy: 0.8705 - val_loss: 0.5871 - val_accuracy: 0.8204
Epoch 75/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3723 -
accuracy: 0.8686 - val_loss: 0.6114 - val_accuracy: 0.8186
Epoch 76/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3678 -
accuracy: 0.8722 - val_loss: 0.6022 - val_accuracy: 0.8183
Epoch 77/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3681 -
accuracy: 0.8724 - val_loss: 0.6132 - val_accuracy: 0.8209
Epoch 78/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3671 -
accuracy: 0.8741 - val_loss: 0.6265 - val_accuracy: 0.8127
Epoch 79/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3705 -
accuracy: 0.8720 - val_loss: 0.6058 - val_accuracy: 0.8154
Epoch 80/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3752 -
accuracy: 0.8691 - val_loss: 0.5982 - val_accuracy: 0.8218
Epoch 81/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3624 -
accuracy: 0.8741 - val_loss: 0.5998 - val_accuracy: 0.8200
Epoch 82/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3701 -
accuracy: 0.8714 - val_loss: 0.5804 - val_accuracy: 0.8252
Epoch 83/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3653 -
accuracy: 0.8712 - val_loss: 0.6071 - val_accuracy: 0.8180
Epoch 84/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3695 -
accuracy: 0.8692 - val_loss: 0.6001 - val_accuracy: 0.8190
Epoch 85/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3670 -
accuracy: 0.8721 - val_loss: 0.6080 - val_accuracy: 0.8155
Epoch 86/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3650 -
accuracy: 0.8728 - val_loss: 0.6166 - val_accuracy: 0.8194
Epoch 87/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3702 -
```

```
accuracy: 0.8707 - val_loss: 0.6042 - val_accuracy: 0.8156
Epoch 88/100
781/781 [==============================] - 74s 94ms/step - loss: 0.3672 -
accuracy: 0.8703 - val_loss: 0.6064 - val_accuracy: 0.8176
Epoch 89/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3685 -
accuracy: 0.8711 - val_loss: 0.6071 - val_accuracy: 0.8211
Epoch 90/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3656 -
accuracy: 0.8713 - val_loss: 0.6224 - val_accuracy: 0.8154
Epoch 91/100
781/781 [==============================] - 74s 95ms/step - loss: 0.3717 -
accuracy: 0.8723 - val_loss: 0.6336 - val_accuracy: 0.8141
Epoch 92/100
781/781 [==============================] - 83s 106ms/step - loss: 0.3652 -
accuracy: 0.8727 - val_loss: 0.6109 - val_accuracy: 0.8179
Epoch 93/100
781/781 [==============================] - 105s 135ms/step - loss: 0.3656 -
accuracy: 0.8720 - val_loss: 0.5924 - val_accuracy: 0.8234
Epoch 94/100
781/781 [==============================] - 86s 110ms/step - loss: 0.3679 -
accuracy: 0.8728 - val_loss: 0.5921 - val_accuracy: 0.8200
Epoch 95/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3645 -
accuracy: 0.8736 - val_loss: 0.6761 - val_accuracy: 0.8059
Epoch 96/100
781/781 [==============================] - 86s 110ms/step - loss: 0.3660 -
accuracy: 0.8725 - val_loss: 0.6366 - val_accuracy: 0.8184
Epoch 97/100
781/781 [==============================] - 86s 110ms/step - loss: 0.3654 -
accuracy: 0.8719 - val_loss: 0.6157 - val_accuracy: 0.8173
Epoch 98/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3711 -
accuracy: 0.8721 - val_loss: 0.6139 - val_accuracy: 0.8181
Epoch 99/100
781/781 [==============================] - 86s 111ms/step - loss: 0.3666 -
accuracy: 0.8710 - val_loss: 0.5846 - val_accuracy: 0.8233
Epoch 100/100
781/781 [==============================] - 86s 111ms/step - loss: 0.3688 -
accuracy: 0.8721 - val_loss: 0.6040 - val_accuracy: 0.8186
313/313 - 3s - loss: 0.6040 - accuracy: 0.8186
----->Evaluation of the training process for: RMSprop
```




```
----->Test accuracy for RMSprop : 0.8185999989509583
--------------------------------------------------------------------------------
---------------------------------
----->Training the model with: nadam oprimizer
Epoch 1/100
781/781 [==============================] - 112s 136ms/step - loss: 0.3676 -
accuracy: 0.8716 - val_loss: 0.5956 - val_accuracy: 0.8232
Epoch 2/100
781/781 [==============================] - 84s 107ms/step - loss: 0.3600 -
accuracy: 0.8741 - val_loss: 0.5703 - val_accuracy: 0.8307
Epoch 3/100
781/781 [==============================] - 85s 108ms/step - loss: 0.3702 -
accuracy: 0.8700 - val_loss: 0.6080 - val_accuracy: 0.8202
Epoch 4/100
781/781 [==============================] - 95s 121ms/step - loss: 0.3673 -
accuracy: 0.8705 - val_loss: 0.6022 - val_accuracy: 0.8180
Epoch 5/100
781/781 [==============================] - 89s 113ms/step - loss: 0.3595 -
accuracy: 0.8733 - val_loss: 0.6075 - val_accuracy: 0.8165
Epoch 6/100
781/781 [==============================] - 86s 111ms/step - loss: 0.3649 -
accuracy: 0.8714 - val_loss: 0.6271 - val_accuracy: 0.8161
Epoch 7/100
781/781 [==============================] - 89s 113ms/step - loss: 0.3687 -
```

```
accuracy: 0.8718 - val_loss: 0.5909 - val_accuracy: 0.8206
Epoch 8/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3713 -
accuracy: 0.8699 - val_loss: 0.6054 - val_accuracy: 0.8216
Epoch 9/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3657 -
accuracy: 0.8732 - val_loss: 0.6131 - val_accuracy: 0.8205
Epoch 10/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3698 -
accuracy: 0.8733 - val_loss: 0.6098 - val_accuracy: 0.8186
Epoch 11/100
781/781 [==============================] - 111s 142ms/step - loss: 0.3619 -
accuracy: 0.8743 - val_loss: 0.5880 - val_accuracy: 0.8262
Epoch 12/100
781/781 [==============================] - 102s 131ms/step - loss: 0.3684 -
accuracy: 0.8727 - val_loss: 0.6134 - val_accuracy: 0.8136
Epoch 13/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3694 -
accuracy: 0.8728 - val_loss: 0.6023 - val_accuracy: 0.8212
Epoch 14/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3662 -
accuracy: 0.8729 - val_loss: 0.6060 - val_accuracy: 0.8203
Epoch 15/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3662 -
accuracy: 0.8727 - val_loss: 0.5972 - val_accuracy: 0.8220
Epoch 16/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3695 -
accuracy: 0.8717 - val_loss: 0.6219 - val_accuracy: 0.8181
Epoch 17/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3672 -
accuracy: 0.8714 - val_loss: 0.6256 - val_accuracy: 0.8076
Epoch 18/100
781/781 [==============================] - 86s 111ms/step - loss: 0.3703 -
accuracy: 0.8708 - val_loss: 0.6102 - val_accuracy: 0.8174
Epoch 19/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3600 -
accuracy: 0.8738 - val_loss: 0.5907 - val_accuracy: 0.8197
Epoch 20/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3646 -
accuracy: 0.8716 - val_loss: 0.6202 - val_accuracy: 0.8116
Epoch 21/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3649 -
accuracy: 0.8738 - val_loss: 0.6222 - val_accuracy: 0.8179
Epoch 22/100
781/781 [==============================] - 90s 115ms/step - loss: 0.3658 -
accuracy: 0.8727 - val_loss: 0.6112 - val_accuracy: 0.8184
Epoch 23/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3641 -
```

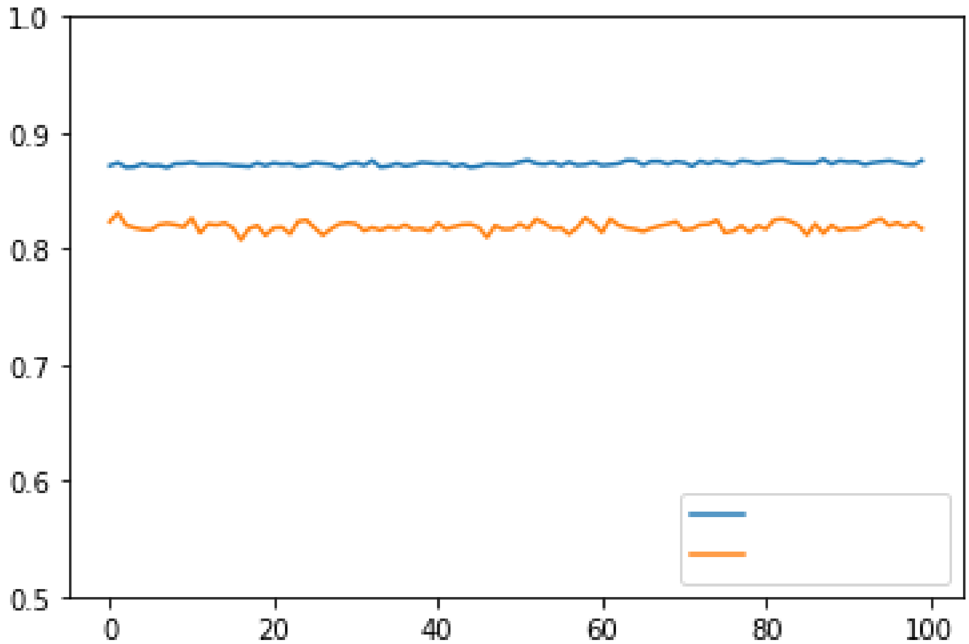
```
accuracy: 0.8733 - val_loss: 0.6331 - val_accuracy: 0.8128
Epoch 24/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3686 -
accuracy: 0.8710 - val_loss: 0.5920 - val_accuracy: 0.8235
Epoch 25/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3646 -
accuracy: 0.8717 - val_loss: 0.5937 - val_accuracy: 0.8245
Epoch 26/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3622 -
accuracy: 0.8742 - val_loss: 0.5941 - val_accuracy: 0.8178
Epoch 27/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3672 -
accuracy: 0.8732 - val_loss: 0.6328 - val_accuracy: 0.8117
Epoch 28/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3679 -
accuracy: 0.8725 - val_loss: 0.6170 - val_accuracy: 0.8170
Epoch 29/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3709 -
accuracy: 0.8699 - val_loss: 0.6022 - val_accuracy: 0.8213
Epoch 30/100
781/781 [==============================] - 89s 114ms/step - loss: 0.3684 -
accuracy: 0.8727 - val_loss: 0.5793 - val_accuracy: 0.8221
Epoch 31/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3615 -
accuracy: 0.8740 - val_loss: 0.6011 - val_accuracy: 0.8212
Epoch 32/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3656 -
accuracy: 0.8718 - val_loss: 0.6161 - val_accuracy: 0.8154
Epoch 33/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3618 -
accuracy: 0.8758 - val_loss: 0.6048 - val_accuracy: 0.8182
Epoch 34/100
781/781 [==============================] - 89s 113ms/step - loss: 0.3691 -
accuracy: 0.8703 - val_loss: 0.6327 - val_accuracy: 0.8159
Epoch 35/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3661 -
accuracy: 0.8713 - val_loss: 0.5859 - val_accuracy: 0.8189
Epoch 36/100
781/781 [==============================] - 95s 122ms/step - loss: 0.3667 -
accuracy: 0.8734 - val_loss: 0.6136 - val_accuracy: 0.8172
Epoch 37/100
781/781 [==============================] - 94s 121ms/step - loss: 0.3682 -
accuracy: 0.8713 - val_loss: 0.6089 - val_accuracy: 0.8205
Epoch 38/100
781/781 [==============================] - 98s 125ms/step - loss: 0.3674 -
accuracy: 0.8724 - val_loss: 0.6035 - val_accuracy: 0.8166
Epoch 39/100
781/781 [==============================] - 98s 125ms/step - loss: 0.3618 -
```

```
accuracy: 0.8743 - val_loss: 0.6078 - val_accuracy: 0.8172
Epoch 40/100
781/781 [==============================] - 85s 109ms/step - loss: 0.3645 -
accuracy: 0.8738 - val_loss: 0.6184 - val_accuracy: 0.8152
Epoch 41/100
781/781 [==============================] - 83s 106ms/step - loss: 0.3634 -
accuracy: 0.8729 - val_loss: 0.6040 - val_accuracy: 0.8219
Epoch 42/100
781/781 [==============================] - 88s 113ms/step - loss: 0.3615 -
accuracy: 0.8736 - val_loss: 0.6158 - val_accuracy: 0.8169
Epoch 43/100
781/781 [==============================] - 89s 114ms/step - loss: 0.3643 -
accuracy: 0.8710 - val_loss: 0.5961 - val_accuracy: 0.8189
Epoch 44/100
781/781 [==============================] - 102s 131ms/step - loss: 0.3617 -
accuracy: 0.8727 - val_loss: 0.6080 - val_accuracy: 0.8207
Epoch 45/100
781/781 [==============================] - 88s 113ms/step - loss: 0.3685 -
accuracy: 0.8701 - val_loss: 0.5897 - val_accuracy: 0.8210
Epoch 46/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3657 -
accuracy: 0.8712 - val_loss: 0.6094 - val_accuracy: 0.8182
Epoch 47/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3638 -
accuracy: 0.8731 - val_loss: 0.6439 - val_accuracy: 0.8099
Epoch 48/100
781/781 [==============================] - 88s 113ms/step - loss: 0.3634 -
accuracy: 0.8728 - val_loss: 0.6187 - val_accuracy: 0.8198
Epoch 49/100
781/781 [==============================] - 89s 113ms/step - loss: 0.3641 -
accuracy: 0.8722 - val_loss: 0.6348 - val_accuracy: 0.8167
Epoch 50/100
781/781 [==============================] - 88s 113ms/step - loss: 0.3649 -
accuracy: 0.8725 - val_loss: 0.6134 - val_accuracy: 0.8169
Epoch 51/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3596 -
accuracy: 0.8747 - val_loss: 0.5975 - val_accuracy: 0.8213
Epoch 52/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3613 -
accuracy: 0.8763 - val_loss: 0.6097 - val_accuracy: 0.8173
Epoch 53/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3627 -
accuracy: 0.8734 - val_loss: 0.5784 - val_accuracy: 0.8253
Epoch 54/100
781/781 [==============================] - 88s 113ms/step - loss: 0.3678 -
accuracy: 0.8727 - val_loss: 0.6029 - val_accuracy: 0.8219
Epoch 55/100
781/781 [==============================] - 80s 103ms/step - loss: 0.3629 -
```

```
accuracy: 0.8744 - val_loss: 0.6218 - val_accuracy: 0.8172
Epoch 56/100
781/781 [==============================] - 80s 103ms/step - loss: 0.3697 -
accuracy: 0.8717 - val_loss: 0.6302 - val_accuracy: 0.8180
Epoch 57/100
781/781 [==============================] - 81s 104ms/step - loss: 0.3575 -
accuracy: 0.8746 - val_loss: 0.6252 - val_accuracy: 0.8126
Epoch 58/100
781/781 [==============================] - 85s 109ms/step - loss: 0.3630 -
accuracy: 0.8719 - val_loss: 0.6012 - val_accuracy: 0.8183
Epoch 59/100
781/781 [==============================] - 84s 108ms/step - loss: 0.3665 -
accuracy: 0.8725 - val_loss: 0.5886 - val_accuracy: 0.8266
Epoch 60/100
781/781 [==============================] - 84s 108ms/step - loss: 0.3612 -
accuracy: 0.8743 - val_loss: 0.6073 - val_accuracy: 0.8209
Epoch 61/100
781/781 [==============================] - 84s 108ms/step - loss: 0.3660 -
accuracy: 0.8719 - val_loss: 0.6253 - val_accuracy: 0.8141
Epoch 62/100
781/781 [==============================] - 84s 108ms/step - loss: 0.3649 -
accuracy: 0.8724 - val_loss: 0.5962 - val_accuracy: 0.8254
Epoch 63/100
781/781 [==============================] - 85s 108ms/step - loss: 0.3626 -
accuracy: 0.8733 - val_loss: 0.6114 - val_accuracy: 0.8203
Epoch 64/100
781/781 [==============================] - 85s 108ms/step - loss: 0.3585 -
accuracy: 0.8761 - val_loss: 0.6194 - val_accuracy: 0.8178
Epoch 65/100
781/781 [==============================] - 85s 109ms/step - loss: 0.3577 -
accuracy: 0.8760 - val_loss: 0.6157 - val_accuracy: 0.8168
Epoch 66/100
781/781 [==============================] - 84s 108ms/step - loss: 0.3699 -
accuracy: 0.8722 - val_loss: 0.6316 - val_accuracy: 0.8148
Epoch 67/100
781/781 [==============================] - 86s 110ms/step - loss: 0.3592 -
accuracy: 0.8752 - val_loss: 0.6118 - val_accuracy: 0.8175
Epoch 68/100
781/781 [==============================] - 86s 111ms/step - loss: 0.3593 -
accuracy: 0.8752 - val_loss: 0.6108 - val_accuracy: 0.8195
Epoch 69/100
781/781 [==============================] - 86s 111ms/step - loss: 0.3662 -
accuracy: 0.8729 - val_loss: 0.6069 - val_accuracy: 0.8211
Epoch 70/100
781/781 [==============================] - 83s 106ms/step - loss: 0.3614 -
accuracy: 0.8751 - val_loss: 0.6058 - val_accuracy: 0.8231
Epoch 71/100
781/781 [==============================] - 73s 94ms/step - loss: 0.3602 -
```

```
accuracy: 0.8742 - val_loss: 0.6119 - val_accuracy: 0.8162
Epoch 72/100
781/781 [==============================] - 71s 91ms/step - loss: 0.3663 -
accuracy: 0.8712 - val_loss: 0.6131 - val_accuracy: 0.8171
Epoch 73/100
781/781 [==============================] - 71s 91ms/step - loss: 0.3604 -
accuracy: 0.8755 - val_loss: 0.6185 - val_accuracy: 0.8206
Epoch 74/100
781/781 [==============================] - 78s 100ms/step - loss: 0.3625 -
accuracy: 0.8733 - val_loss: 0.6121 - val_accuracy: 0.8211
Epoch 75/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3595 -
accuracy: 0.8756 - val_loss: 0.6101 - val_accuracy: 0.8245
Epoch 76/100
781/781 [==============================] - 88s 112ms/step - loss: 0.3618 -
accuracy: 0.8739 - val_loss: 0.6269 - val_accuracy: 0.8137
Epoch 77/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3646 -
accuracy: 0.8727 - val_loss: 0.5960 - val_accuracy: 0.8152
Epoch 78/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3588 -
accuracy: 0.8758 - val_loss: 0.6066 - val_accuracy: 0.8197
Epoch 79/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3588 -
accuracy: 0.8750 - val_loss: 0.6199 - val_accuracy: 0.8141
Epoch 80/100
781/781 [==============================] - 87s 111ms/step - loss: 0.3623 -
accuracy: 0.8733 - val_loss: 0.6079 - val_accuracy: 0.8198
Epoch 81/100
781/781 [==============================] - 86s 111ms/step - loss: 0.3616 -
accuracy: 0.8745 - val_loss: 0.6113 - val_accuracy: 0.8169
Epoch 82/100
781/781 [==============================] - 88s 113ms/step - loss: 0.3602 -
accuracy: 0.8757 - val_loss: 0.5763 - val_accuracy: 0.8246
Epoch 83/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3572 -
accuracy: 0.8759 - val_loss: 0.6051 - val_accuracy: 0.8255
Epoch 84/100
781/781 [==============================] - 88s 113ms/step - loss: 0.3616 -
accuracy: 0.8740 - val_loss: 0.6024 - val_accuracy: 0.8234
Epoch 85/100
781/781 [==============================] - 89s 113ms/step - loss: 0.3627 -
accuracy: 0.8737 - val_loss: 0.6037 - val_accuracy: 0.8198
Epoch 86/100
781/781 [==============================] - 86s 110ms/step - loss: 0.3653 -
accuracy: 0.8737 - val_loss: 0.6220 - val_accuracy: 0.8119
Epoch 87/100
781/781 [==============================] - 85s 109ms/step - loss: 0.3615 -
```

```
accuracy: 0.8736 - val_loss: 0.5850 - val_accuracy: 0.8213
Epoch 88/100
781/781 [==============================] - 102s 131ms/step - loss: 0.3548 -
accuracy: 0.8771 - val_loss: 0.6373 - val_accuracy: 0.8131
Epoch 89/100
781/781 [==============================] - 96s 123ms/step - loss: 0.3630 -
accuracy: 0.8730 - val_loss: 0.6066 - val_accuracy: 0.8202
Epoch 90/100
781/781 [==============================] - 87s 112ms/step - loss: 0.3574 -
accuracy: 0.8758 - val_loss: 0.6212 - val_accuracy: 0.8156
Epoch 91/100
781/781 [==============================] - 76s 97ms/step - loss: 0.3583 -
accuracy: 0.8746 - val_loss: 0.6235 - val_accuracy: 0.8176
Epoch 92/100
781/781 [==============================] - 80s 103ms/step - loss: 0.3636 -
accuracy: 0.8750 - val_loss: 0.6088 - val_accuracy: 0.8171
Epoch 93/100
781/781 [==============================] - 90s 114ms/step - loss: 0.3667 -
accuracy: 0.8725 - val_loss: 0.6178 - val_accuracy: 0.8190
Epoch 94/100
781/781 [==============================] - 85s 108ms/step - loss: 0.3594 -
accuracy: 0.8741 - val_loss: 0.5993 - val_accuracy: 0.8232
Epoch 95/100
781/781 [==============================] - 91s 117ms/step - loss: 0.3623 -
accuracy: 0.8749 - val_loss: 0.5934 - val_accuracy: 0.8256
Epoch 96/100
781/781 [==============================] - 79s 101ms/step - loss: 0.3583 -
accuracy: 0.8758 - val_loss: 0.6210 - val_accuracy: 0.8199
Epoch 97/100
781/781 [==============================] - 82s 105ms/step - loss: 0.3583 -
accuracy: 0.8744 - val_loss: 0.5917 - val_accuracy: 0.8221
Epoch 98/100
781/781 [==============================] - 75s 96ms/step - loss: 0.3645 -
accuracy: 0.8731 - val_loss: 0.5929 - val_accuracy: 0.8189
Epoch 99/100
781/781 [==============================] - 113s 144ms/step - loss: 0.3651 -
accuracy: 0.8723 - val_loss: 0.6043 - val_accuracy: 0.8221
Epoch 100/100
781/781 [==============================] - 152s 195ms/step - loss: 0.3553 -
accuracy: 0.8760 - val_loss: 0.6099 - val_accuracy: 0.8171
313/313 - 6s - loss: 0.6099 - accuracy: 0.8171
----->Evaluation of the training process for: nadam
```




```
----->Test accuracy for nadam : 0.8170999884605408
```

[24]:
```
# marking the end of the process of the data augmentation model
end_DA_model = date_and_time_now()
```

[25]:
```
# start and end of the process
print("\n\nSTART AND END -> SIMPLE MODEL:")
print("\tStart: \t" , start_simple_model)
print("\tEnd: \t" , end_simple_model)

print("\n\nSTART AND END -> DATA AUGMENTATION MODEL:")
print("\tStart: \t" , start_DA_model)
print("\tEnd: \t" , end_DA_model)
```

```
START AND END -> SIMPLE MODEL:
        Start:   2021-03-25 21:42:44
        End:     2021-03-25 22:47:52


START AND END -> DATA AUGMENTATION MODEL:
        Start:   2021-03-25 22:47:53
        End:     2021-03-26 11:45:38
```

As seen above, adding the augmented images to the original dataset increased the training and the testing time exponentially but also increased the model's accuracy by approximately 10%.

Optimizor	Simple Model	Data Augmenation Model
ADAM	70%	80%
ADAMAX	71%	81%
ADAGRAD	72%	80%
ADADELTA	72%	81%
RMSprop	68%	82%
NADAM	68%	80%

Comparison: approximate accuracies (might change from one exeuction to another) As seen in the table above, the "Simple Model" has a lower accuracy compared with the "Data Augmentation Model". Almost all optimizer's accuracy was increased by 10% if the dataset is augmented before the training process.

Your results may vary depending on the environment and evaluation procedure.

Decision tree

In machine learning, decision trees are a form of supervised learning in which data is constantly split according to parameters. Trees contain two different types of entities: nodes (also called branch) and leaves, where leaves are the final outcome, and nodes is where the data splits [13].

If things seem too confusing, look at the image below and answer the questions (questions are nodes or branches), while the final answers are called leaves. [11]

However, things do get a little more complicated than this, as there are two types of decision trees: 1. Classification Trees 2. Regression Trees

1. Classification trees This type of tree matches the example in the picture where the result is binary: true or false. The result of this type of tree is called categorial [13].

2. Regression trees Regression trees are used to predict a value. To simplify the idea, think about how prices are formed. The price of a house will depend on different factors, such as area, number of rooms, schools around, square footage, and others [13]. [12]

Now that the notion of "Decision Tree" is understood, we will go into detail about how they work and when they should be used. Recursive partitioning is used to build a decision tree. Starting from the root node, the node at the very top of the tree, each node represents a decision that will lead to one of the node children. In our example, the root node is "Work to do?" and depending on the answer (yes or no) the data is redirected to one of the node's children: "Stay in" or "Outlook?". The "Stay in" node is a leaf, meaning that it represents a final outcome, while the "Outlook?" node splits further to other children [11].

When real data is inserted into a decision tree, starting from the root, the data split into one child or another depending on its value. In order to create the nodes that split the data, an objective function is required. This function maximizes the information gain at each node that splits the data [11].

All this sounds very fancy and difficult, so further, we will implement the popular IRIS set to get a better understanding of the Decision Trees.

Iris Flowers and Biology Biology? Well, to understand how the Iris Flowers dataset is categorized, we will go over a little bit of biology. Iris is a beautiful flower, and if you had never seen one, look at the image below. [13]

The irises present in the dataset are Iris setosa, Iris virginica, and Iris versicolor. And the record present in the data are the lengths and widths of petals and sepals of these 3 particular kinds of irises (see image below). [14]

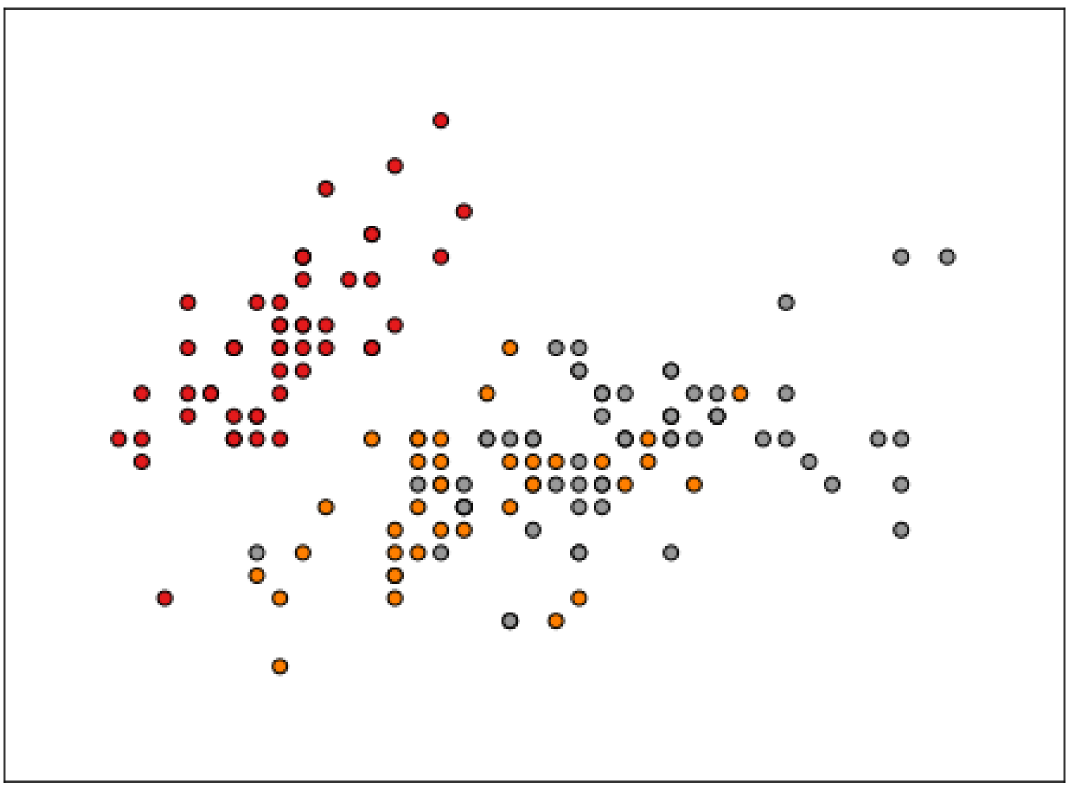
Observing the image above, we can clearly see there are similarities between the different types of flowers, but the objective function calculates the difference between these similarities. If you are interested, you can observe the function in the image below [14],[15]: [14]

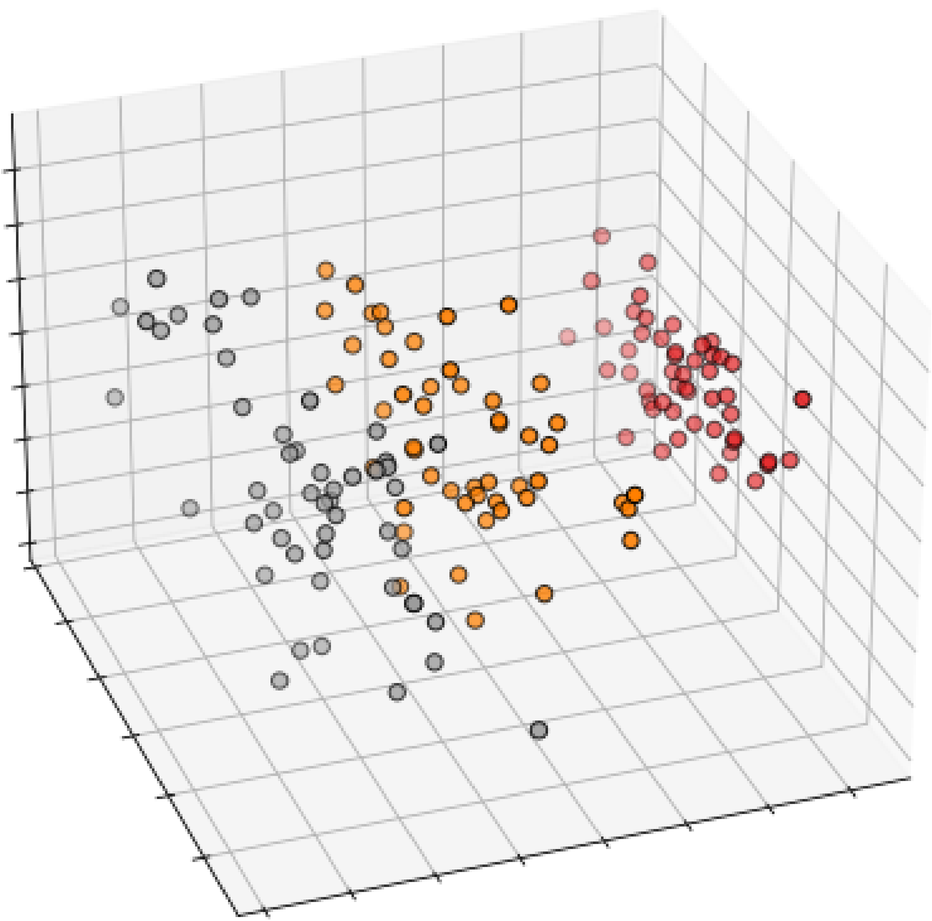
The model we will implement will focus on classification trees, and our goal is to predict each flower from the dataset to which category it belongs. The sepal lengths and width in centimeters are stored in columns. These dimensions are called features, and they describe the iris flower.

The code section below, will help you visuzalize

[26]:
```
# Code source: Gaël Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.clf()

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
```

```python
plt.xticks(())
plt.yticks(())

# To getter a better understanding of interaction of the dimensions
# plot the first three PCA dimensions
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])

plt.show()
```





Another way to visualise the data, is to see it in a table.

Now that you had visualized the data, is time to implement the model, train it and test it.

We will use a Decision Tree, as it was mentioned above. The dataset contains three different classes where data needs to be categorised: 1. setosa 2. versicolor 3. virginica

Each flower will have the following features: 1. sepal length 2. sepal width 3. petal length 4. petal width

And again, our goal is to predict the iris flower class, depending on its features.

The following tutorial is based on: [16], [17]

The first step is to import all the necessary libraries needed to achieve our goal.

[82]:
```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
```

The sklearn.dataset contains multiple datasets, and the one needed for the model is the iris dataset.

From this dataset, we can extract the classes and the features (data). * to extract classes: data.target_names * to extract the features: data.data

[83]:
```
#Loading the iris data
iris = datasets.load_iris()
print('Classes to predict: ', iris.target_names)
```

```
Classes to predict:  ['setosa' 'versicolor' 'virginica']
```

[84]:
```
# extracting data attributes
X = iris.data
# extracting target/ class labels
y = iris.target
print('Number of examples in the data:', X.shape[0])
```

```
Number of examples in the data: 150
```

Now we can visualize all features for each iris.

[85]:
```
ir = pd.DataFrame(X)
ir.columns = iris.feature_names
ir['CLASS'] = iris.target
ir.head(7)
```

[85]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
5                5.4               3.9                1.7               0.4
6                4.6               3.4                1.4               0.3

   CLASS
0      0
1      0
2      0
3      0
```

```
4       0
5       0
6       0
```

Until now we had separated the classes and the features. Now it is time to split the data into two different sets: 1. Training set 2. Testing set

[86]:
```python
# using the train_test_split to create train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 47,␣
↪test_size = 0.25)
```

We will use the DecisionTreeClassifier function from the sklearn library because we are dealing with a classification problem. We will change the criterion parameter to entropy, which sets the measurement for splitting to information gain.

[87]:
```python
tree = DecisionTreeClassifier(criterion = 'entropy')
```

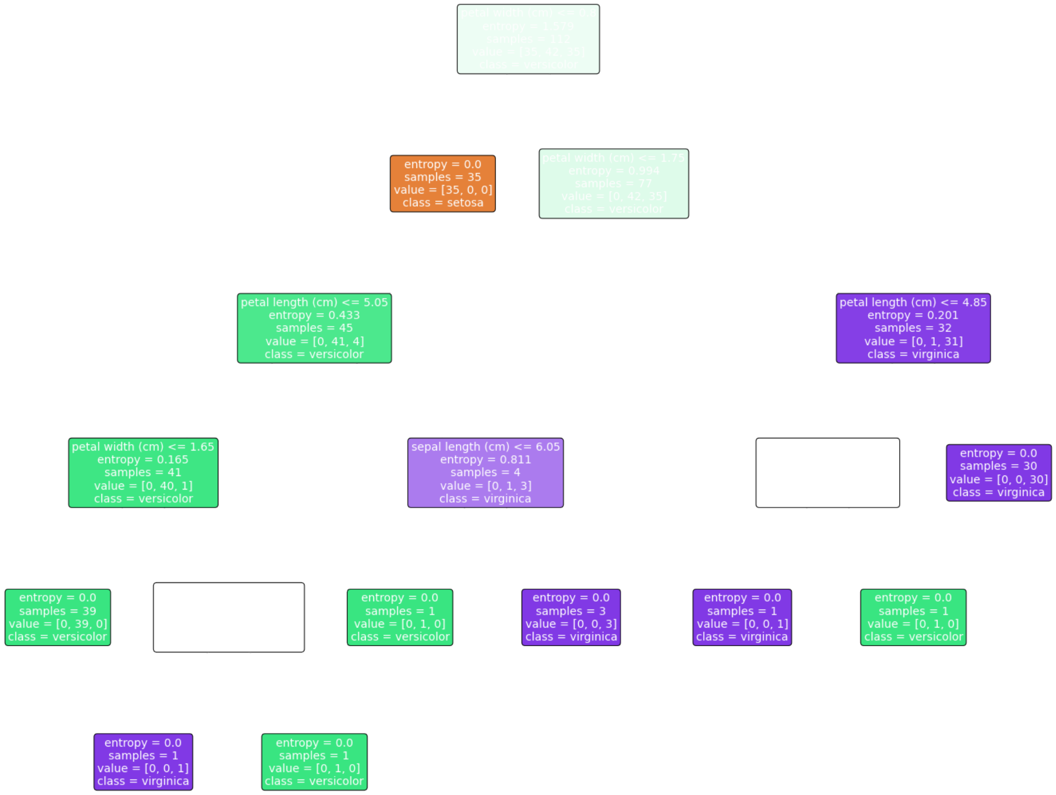
Not is an exciting moment when we train the model with the training dataset. The model will be trained to predict the class of the irises based on their features.

[88]:
```python
# training the decision tree classifier.
tree.fit(X_train, y_train)
```

[88]:
```
DecisionTreeClassifier(criterion='entropy')
```

After the model had been trained, we can plot a tree so that we can visualize it. This is very useful to understand better how the decisions are made inside the tree.

[89]:
```python
fig = plt.figure(figsize=(25,20))
a = plot_tree(tree,
      feature_names=iris.feature_names,
      class_names=iris.target_names,
      filled=True,
      rounded=True,
      fontsize=14)
```



The model is trained; therefore, it is time to test it. We reached the moment when we test the model and see if we achieved our goal.

[90]:
```
# predicting labels on the test set.
y_pred =  tree.predict(X_test)
print('Accuracy Score on train data: ', accuracy_score(y_true=y_train,␣
 ↪y_pred=tree.predict(X_train)))
print('Accuracy Score on test data: ', accuracy_score(y_true=y_test,␣
 ↪y_pred=y_pred))
```

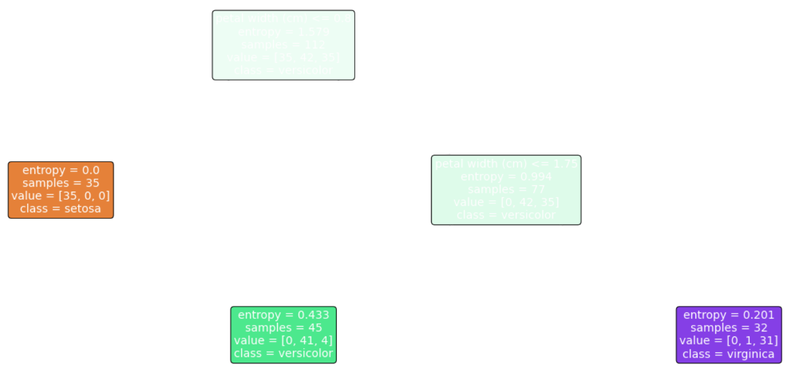
```
Accuracy Score on train data:  1.0
Accuracy Score on test data:  0.9736842105263158
```

This step is a bonus, as we change the tree's parameters, trying to improve its precision. Even though, 94% is a pretty good accuracy score. One of those parameters is "min_samples_split" which specifies the minimum number of samples needed to split an internal node. As seen below, the minimum samle split will be 50.

[91]:
```
tree = DecisionTreeClassifier(criterion='entropy', min_samples_split=50)
tree.fit(X_train, y_train)
```

[91]: DecisionTreeClassifier(criterion='entropy', min_samples_split=50)

[92]:
```
fig = plt.figure(figsize=(25,10))

a = plot_tree(tree,
      feature_names=iris.feature_names,
      class_names=iris.target_names,
      filled=True,
      rounded=True,
      fontsize=14)
```



[93]:
```
print('Accuracy Score on train data: ', accuracy_score(y_true=y_train,␣
 ↪y_pred=tree.predict(X_train)))
print('Accuracy Score on the test data: ', accuracy_score(y_true=y_test,␣
 ↪y_pred=tree.predict(X_test)))
```

```
Accuracy Score on train data:  0.9553571428571429
Accuracy Score on the test data:  0.9736842105263158
```

You can see in the plot above that the size of the tree decreased consistently. Therefore, the training score decreased as well, but the accuracy score increased by 3%, which is correlated with the fact that 'min_samples_split' helps balance the decision boundary and prevents overfitting.

Random Forest

(same dataset - irises)

5 Conclusion

(needs to be filled)

[ ]:
```

```

6 References

1. Jason Brownlee (2017), “How to Handle Missing Data with Python”, Available at: https://machinelearningmastery.com/handle-missing-data-python/, (Accessed 10 December 2020)

2. Tyler Doll, 2018, “LDA Topic Modeling: An Explanation”, Available at: https://towardsdatascience.com/lda-topic-modeling-an-explanation-e184c90aadcd, (Accessed 01.01.2021)

3. Tenserflow (2020), “Convolutional Neural Network (CNN)”, Available at: https://www.tensorflow.org/tutorials/images/cnn, (Accessed 05.01.2020)

4. Theodora Tataru and Liliana O’Sullivan (2020), “4th Year Data Science CA 1- Research Document” Available at: https://instituteoftechnol663-my.sharepoint.com/:b:/g/personal/c00231174_itcarlow_ie/EaPwa2jNdhxEs0FtRfYoPlABahguc5fn0Rd8gf9 (Accessed 05.01.2020)

5. Douglas Daseeco (2017), “How does AI learn?”, Available at: https://ai.stackexchange.com/questions/3640/how-does-an-ai-learn, (Accessed 05.01.2021)

6. Matthew Hutson (2018), “How researchers are teaching AI to learn like a child”, Available at: https://www.sciencemag.org/news/2018/05/howresearchers-are-teaching-ai-learn-child, (Accessed 05.01.2021)

7. Daksh Trehan (2020), “Convolutional Neural Networks”, Available at: https://medium.com/towards-artificial-intelligence/convolutional-neural-networks-for-dummies-afd7166cd9e, (Accessed 22.01.2021)

8. Ben Dickson (2020), “What are convolutional neural networks (CNN)?”, Available at: https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/, (Accessed 22.01.2021)

9. Sebastian Ruder (2016), “An overview of gradient descent optimization algorithms”, Available at: https://ruder.io/optimizing-gradient-descent/index.html#momentum, (Accessed 27.02.2021)

10. Arun Gandhi (2021), “Data Augmentation | How to use Deep Learning when you have Limited Data—Part 2”, Available at: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/, (Accessed 19.03.2021)

11. Loreinne Li (2019), “Classification and Regression Analysis with Decision Trees”, Available at https://towardsdatascience.com/https-medium-com-lorrli-classification-and-regression-analysis-with-decision-trees-c43cdbc58054, (Accessed 23.03.2021)

12. Samet Girgin (2019), “Decision Tree Regression in 6 Steps with Python”, Avaialble at https://medium.com/pursuitnotes/decision-tree-regression-in-6-steps-with-python-1a1c5aa2ee16, (Accessed 23.03.2021)

13. Oriant (2017), “Decision Trees for Classification: A Machine Learning Algorithm”, Avaialble at https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html#:~:text=%7C-,Blog,namely%20decision%20nodes%20and%20leaves, (Accessed 24.03.2021)

14. Susan Candelario (2021), “iris flower collage”, Available at https://www.flickr.com/photos/susancandelario/50016355531, (Accessed 24.03.2021)

15. Yong Cui (2020), “The Iris Dataset — A Little Bit of History and Biology”, Avaialble at https://towardsdatascience.com/the-iris-dataset-a-little-bit-of-history-and-biology-fb4812f5a7b5, (Accessed 24.03.2021)

16. Hackerearth (2021), “Decision Tree”, Avialable at https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/ml-decision-tree/tutorial/, (Accessed 24.03.2021)

17. Piotr Płoński (2020), “Visualize a Decision Tree in 4 Ways with Scikit-Learn and Python”, Available at https://mljar.com/blog/visualize-decision-tree/, (Accessed 24.03.2021)

[ ]:

```

```