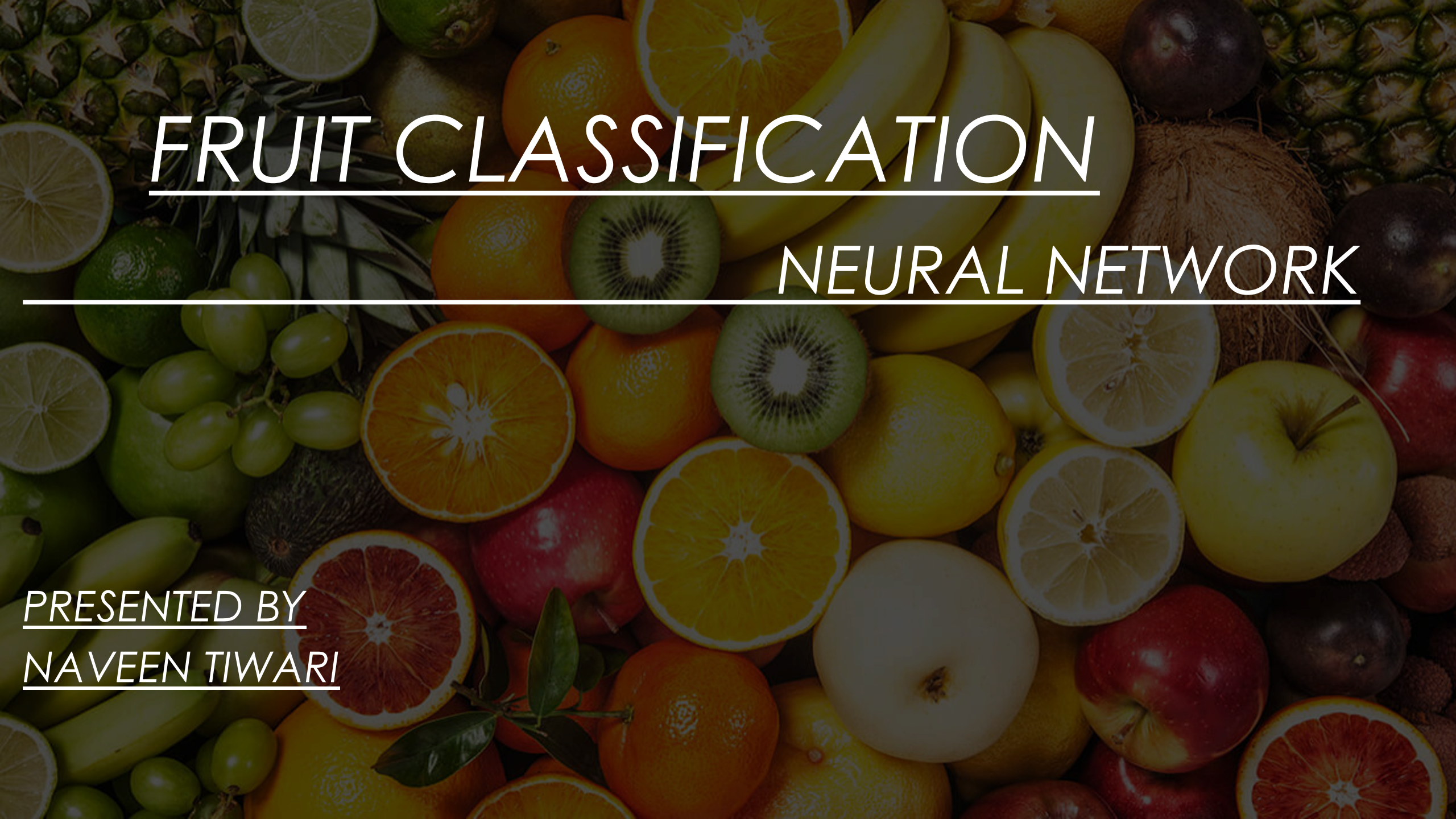


FRUIT CLASSIFICATION

NEURAL NETWORK

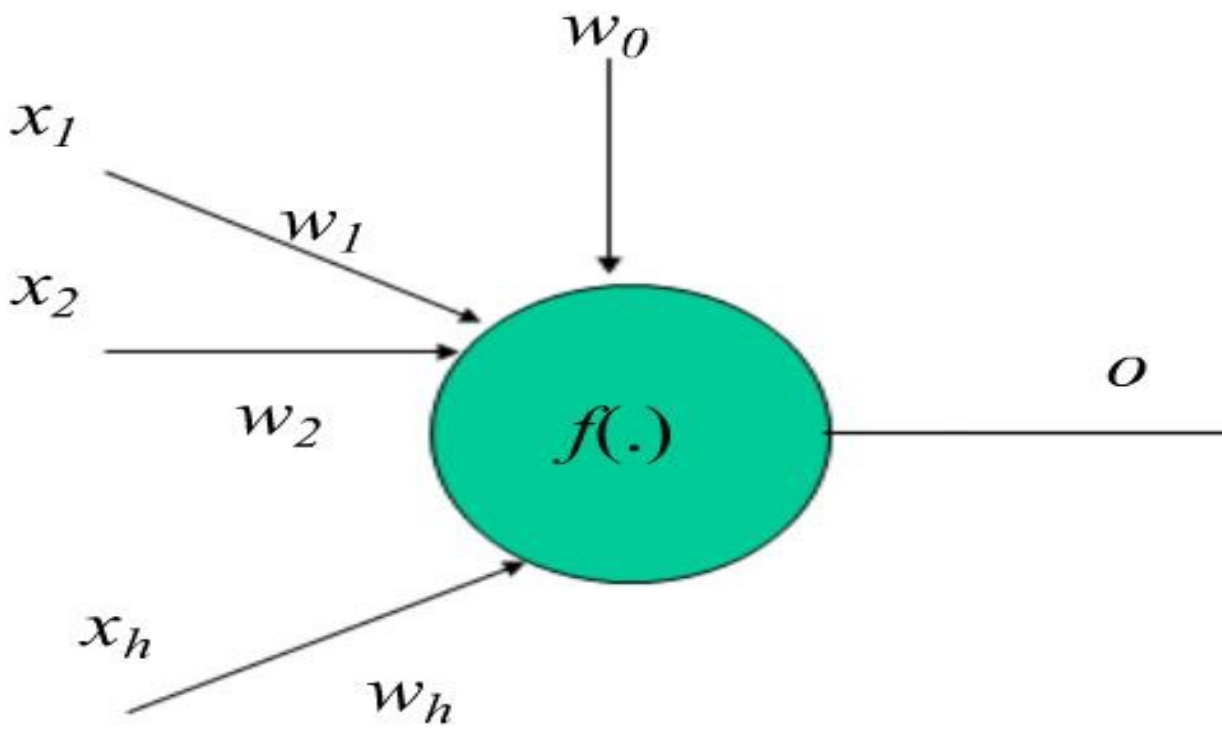
PRESENTED BY
NAVEEN TIWARI

INTRODUCTION

- In this project I ***have used the concept of neural networks to classify different fruits.***
- Neural networks are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.
  Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity.

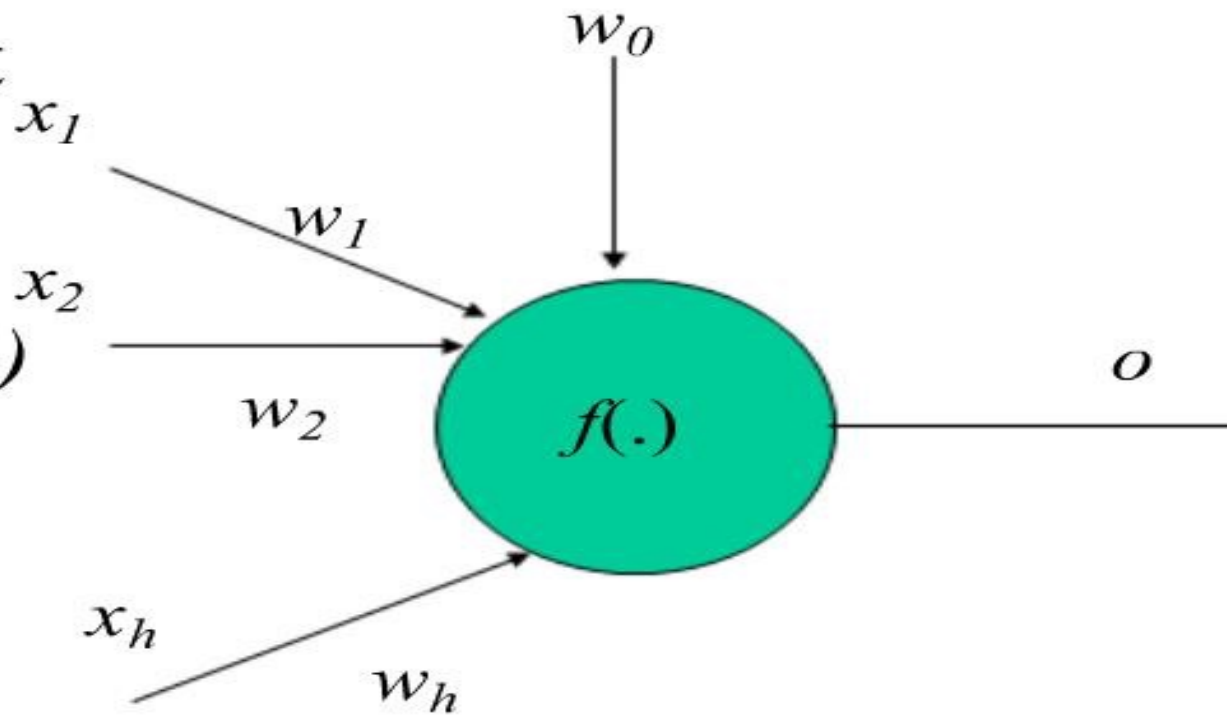
What is Artificial Neuron?

- It is basic processing unit of neural networks
- Artificial Neuron has:
  - Inputs $x_1, x_2, .., x_h$
  - Weights $w_1, w_2, \ldots, w_k$
  - Bias w_0
  - Transfer function f
  - Output o

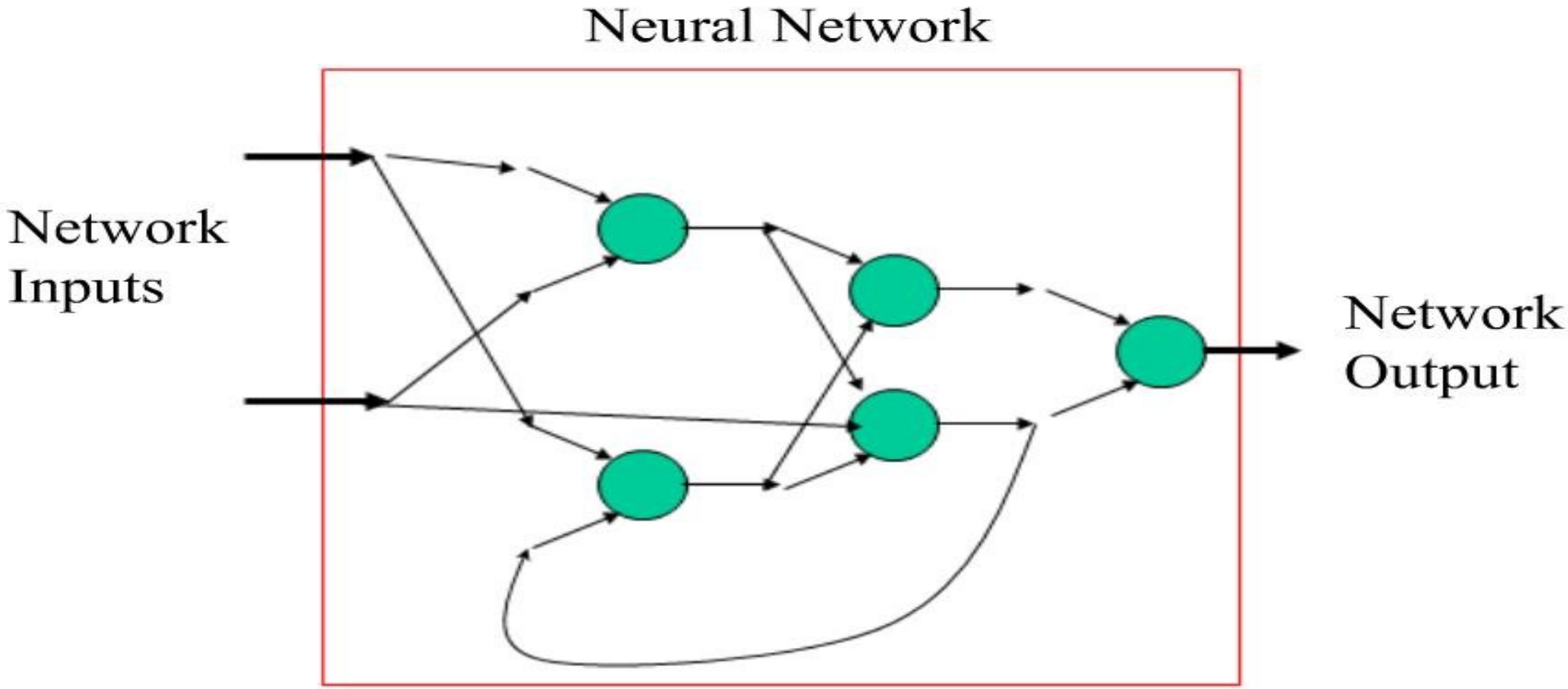


How Artificial Neuron Works?

- First, compute linear combination of inputs weighted by weights and add the bias
  - $y = x_1 w_1 + \ldots + x_h w_h + w_0$
- Then, compute the output
  - Apply transfer function to the linear combination
  - $o = f(x_1 w_1 + \ldots + x_h w_h + w_0)$



Example of Neural Network



Information related to the database:

- I have used the Fruits 360 database(version:2020.05.18.0) from Kaggle as per recommended by my respected professor.
- The databse consists of Apples (different varieties: Crimson Snow, Golden, Golden-Red, Granny Smith, Pink Lady, Red, Red Delicious), Apricot, Avocado, Avocado ripe, Banana (Yellow, Red, Lady Finger), Beetroot Red, Blueberry, Cactus fruit, Cantaloupe (2 varieties), Carambula, Cauliflower, Cherry (different varieties, Rainier), Cherry Wax (Yellow, Red, Black), Chestnut, Clementine, Cocos, Corn (with husk), Cucumber (ripened), Dates, Eggplant, Fig, Ginger Root, Granadilla, Grape (Blue, Pink, White (different varieties)), Grapefruit (Pink, White), Guava, Hazelnut, Huckleberry, Kiwi, Kaki, Kohlrabi, Kumsquats, Lemon (normal, Meyer), Lime, Lychee, Mandarine, Mango (Green, Red), Mangostan, Maracuja, Melon Piel de Sapo, Mulberry, Nectarine (Regular, Flat), Nut (Forest, Pecan), Onion (Red, White), Orange, Papaya, Passion fruit, Peach (different varieties), Pepino, Pear (different varieties, Abate, Forelle, Kaiser, Monster, Red, Stone, Williams), Pepper (Red, Green, Orange, Yellow), Physalis (normal, with Husk), Pineapple (normal, Mini), Pitahaya Red, Plum (different varieties), Pomegranate, Pomelo Sweetie, Potato (Red, Sweet, White), Quince, Rambutan, Raspberry, Redcurrant, Salak, Strawberry (normal, Wedge), Tamarillo, Tangelo, Tomato (different varieties, Maroon, Cherry Red, Yellow, not ripened, Heart), Walnut, Watermelon.
-

Dataset properties:

- The total number of images: 90483.
- Training set size: 67692 images (one fruit or vegetable per image).
- Test set size: 22688 images (one fruit or vegetable per image).
- The number of classes: 131 (fruits and vegetables).
- Image size: 100x100 pixels.
- Filename format: image*index*x100.jpg (e.g. 32*100.jpg) or r*image*index*x*100.jpg (e.g. r32100.jpg) or r2*image*index*100.jpg or r3*image*index*x*100.jpg. "r" stands for rotated fruit. "r2" means that the fruit was rotated around the 3rd axis. "100" comes from image size (100x100 pixels).

- For this project I have imported several libraries, shown below:

```
1  import tensorflow as tf
2  from tensorflow.keras import models, layers
3  import matplotlib.pyplot as plt
4  import numpy as np
5
```

- The TensorFlow handles data sets that are arrayed as computational nodes in graph form. The edges that connect the nodes in a graph can represent multidimensional vectors or matrices, creating what are known as tensors. Because TensorFlow programs use a data flow architecture that works with generalized intermediate results of the computations, they are especially open to very large-scale parallel processing applications, with neural networks being a common example.

- Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Keras has the following key features:Allows the same code to run on CPU or on GPU, seamlessly.User-friendly API which makes it easy to quickly prototype deep learning models.Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

- tensorflow.keras.models groups layers into an object with training and inference features.
- There are two ways to instantiate a Model:

1 - With the "Functional API", where you start from Input, you chain layer calls to specify the model's forward pass, and finally you create your model from inputs and outputs:

2 - By subclassing the Model class: in that case, you should define your layers in __init__() and you should implement the model's forward pass in call().

- tensorflow.keras.layers is the class from which all layers inherit.

- matplotlib.pyplot is a state-based interface to matplotlib. It provides an implicit, MATLAB-like, way of plotting. It also opens figures on your screen, and acts as the figure GUI manager.
- pyplot is mainly intended for interactive plots and simple cases of programmatic plot generation:
- NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
  Json for

```python
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import numpy as np

BATCH_SIZE = 32 # it will batch 32 image and read on time

IMAGE_SIZE = 300
CHANNELS=3 #rgb chnl vl
EPOCHS=1 # how much we want to train the model if acc is less inc if gpu slow than dec
#to read the dataset
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/Users/naveen/Downloads/archive/fruits-360-original-size/fruits-360-original-size/Training",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

class_names = dataset.class_names
class_names

for image_batch, labels_batch in dataset.take(1): #prints the image in array form bsd on size  colr
    print(image_batch.shape)
    print(labels_batch.numpy())


plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")


def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds) #size of dataset

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size) #aftr skipng trn siz take data for val
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds



train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset) #take diffrnt data frm datast

len(train_ds)
len(val_ds)
len(test_ds)

#shuffle dataset
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)


resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])

# inverts img hori vert to trn bettr inc acc
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])

 #applyng data argumtn on dtast
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
s = 24

models.Sequential([
ze_and_rescale,
rs.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
rs.MaxPooling2D((2, 2)),
rs.Conv2D(64,  kernel_size = (3,3), activation='relu'),
rs.MaxPooling2D((2, 2)),
rs.Conv2D(64,  kernel_size = (3,3), activation='relu'),
rs.MaxPooling2D((2, 2)),
rs.Conv2D(64, (3, 3), activation='relu'),
rs.MaxPooling2D((2, 2)),
rs.Conv2D(64, (3, 3), activation='relu'),
rs.MaxPooling2D((2, 2)),
rs.Conv2D(64, (3, 3), activation='relu'),
rs.MaxPooling2D((2, 2)),
rs.Flatten(),
rs.Dense(64, activation='relu'),
rs.Dense(n_classes, activation='softmax'),


ild(input_shape=input_shape)

mmary()

ng using adam opt
mpile(
mizer='adam',
=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
ics=['accuracy']

model to pass data for traning
= model.fit(
n_ds,
h_size=BATCH_SIZE,
dation_data=val_ds,
ose=1,
hs=1,


model.evaluate(test_ds)
```

```

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']


 # crtng func to pass some image and test it

for images_batch, labels_batch in test_ds.take(1):
    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:", class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:", class_names[np.argmax(batch_prediction[0])])


    def predict(model, img):
        img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
        img_array = tf.expand_dims(img_array, 0)

        predictions = model.predict(img_array)

        predicted_class = class_names[np.argmax(predictions[0])]
        confidence = round(100 * (np.max(predictions[0])), 2)
        return predicted_class, confidence

plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1): #take any one batch
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]
```

```
    def predict(model, img):
        img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
        img_array = tf.expand_dims(img_array, 0)

        predictions = model.predict(img_array)

        predicted_class = class_names[np.argmax(predictions[0])]
        confidence = round(100 * (np.max(predictions[0])), 2)
        return predicted_class, confidence

plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1): #take any one batch
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")


from fontTools.varLib import plot


def plot_history(history):
    fig, axs = plot.subplots(2)
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")
    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="test error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Error eval")
    plot.show()

model.save("models/fruits.h5")
```

This is the split that I have used:

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds) #size of dataset

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size) #aftr skipng trn siz take data for val
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds



train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset) #take diffrnt data frm datast
```

• Data argumentation:

```
72
73      # inverts img hori vert to trn bettr inc acc
74      data_augmentation = tf.keras.Sequential([
75        layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
76        layers.experimental.preprocessing.RandomRotation(0.2),
77      ])
78
79       #applyng data argumtn on dtast
80      train_ds = train_ds.map(
81          lambda x, y: (data_augmentation(x, training=True), y)
82      ).prefetch(buffer_size=tf.data.AUTOTUNE)
83
84
85      input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
86      n_classes = 24
```

- Data augmentation: a technique to increase the diversity of your training set by applying random (but realistic) transformations, such as image rotation.
- I have used it so that I can train data in a better way in order to get more accuracy.

```
model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)

model.summary()

```

MODEL SUMMARY:

```
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (32, 300, 300, 3)         0

conv2d (Conv2D)              (32, 298, 298, 32)        896

max_pooling2d (MaxPooling2D  (32, 149, 149, 32)        0
)

conv2d_1 (Conv2D)            (32, 147, 147, 64)        18496

max_pooling2d_1 (MaxPooling  (32, 73, 73, 64)          0
2D)

conv2d_2 (Conv2D)            (32, 71, 71, 64)          36928

max_pooling2d_2 (MaxPooling  (32, 35, 35, 64)          0
2D)

conv2d_3 (Conv2D)            (32, 33, 33, 64)          36928

max_pooling2d_3 (MaxPooling  (32, 16, 16, 64)          0
2D)

conv2d_4 (Conv2D)            (32, 14, 14, 64)          36928

max_pooling2d_4 (MaxPooling  (32, 7, 7, 64)            0
2D)

conv2d_5 (Conv2D)            (32, 5, 5, 64)            36928

max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)            0
2D)

flatten (Flatten)            (32, 256)                 0

dense (Dense)                (32, 64)                  16448

dense_1 (Dense)              (32, 24)                  1560

=================================================================
Total params: 185,112
Trainable params: 185,112
Non-trainable params: 0
```

RELU activation function :

- In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.
- The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.In this tutorial, you will discover the rectified linear activation function for deep learning neural networks.After completing this tutorial, you will know:The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem.
- Applies the rectified linear unit activation function. With default values, this returns the standard ReLU activation: max(x, 0) , the element-wise maximum of 0 and the input.
- Use the keyword argument input_shape (tuple of integers, does not include the batch axis) when using this layer as the first layer in a model.tensor.
- ▪ Outputshape
- ▪ Same shape as the input.
- ▪ Arguments
- ▪ max_value: Float >= 0. Maximum activation value. Default to None, which means unlimited.
- negative_slope: Float >= 0. Negative slope coefficient. Default to 0.
- ▪ threshold:Float>=0.Thresholdvalueforthresholdedactivation.Default to 0.

```
#compiling using adam opt
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
# makng model to pass data for traning
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=40,
)

scores = model.evaluate(test_ds)


acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

- **Compiling using ADAM Optimizer :**

- Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.
- Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled "Adam: A Method for Stochastic Optimization". I will quote liberally from their paper in this post, unless stated otherwise.
- The algorithm is che name Adam is derived from adaptive moment estimation.When introducing the algorithm, the authors list the attractive benefits of using Adam on non-convex optimization problems, as follows:alled Adam. It is not an acronym and is not written as "ADAM".
- Straightforward to implement.
  Computationally efficient
  Little memory requirements.
  Invariant to diagonal rescale of the gradients.
  Well suited for problems that are large in terms of data and/or parameters.
  Appropriate for non-stationary objectives.
- Appropriate for problems with very noisy/or sparse gradients.
  Hyper-parameters have intuitive interpretation and typically require little tuning

- **Loss Sparse categorical crossentropy :**

- Computes the cross-entropy loss between true labels and predicted labels.Use this cross-entropy loss for binary (0 or 1) classification applications. The loss function requires the following inputs:
- y_true (true label): This is either 0 or 1.
- y_pred (predicted value): This is the model's prediction, i.e, a single floating- point value which either represents a logit, (i.e, value in [-inf, inf] when from_logits=True) or a probability (i.e, value in [0., 1.] when from_logits=False

- **Metrics accuracy:**

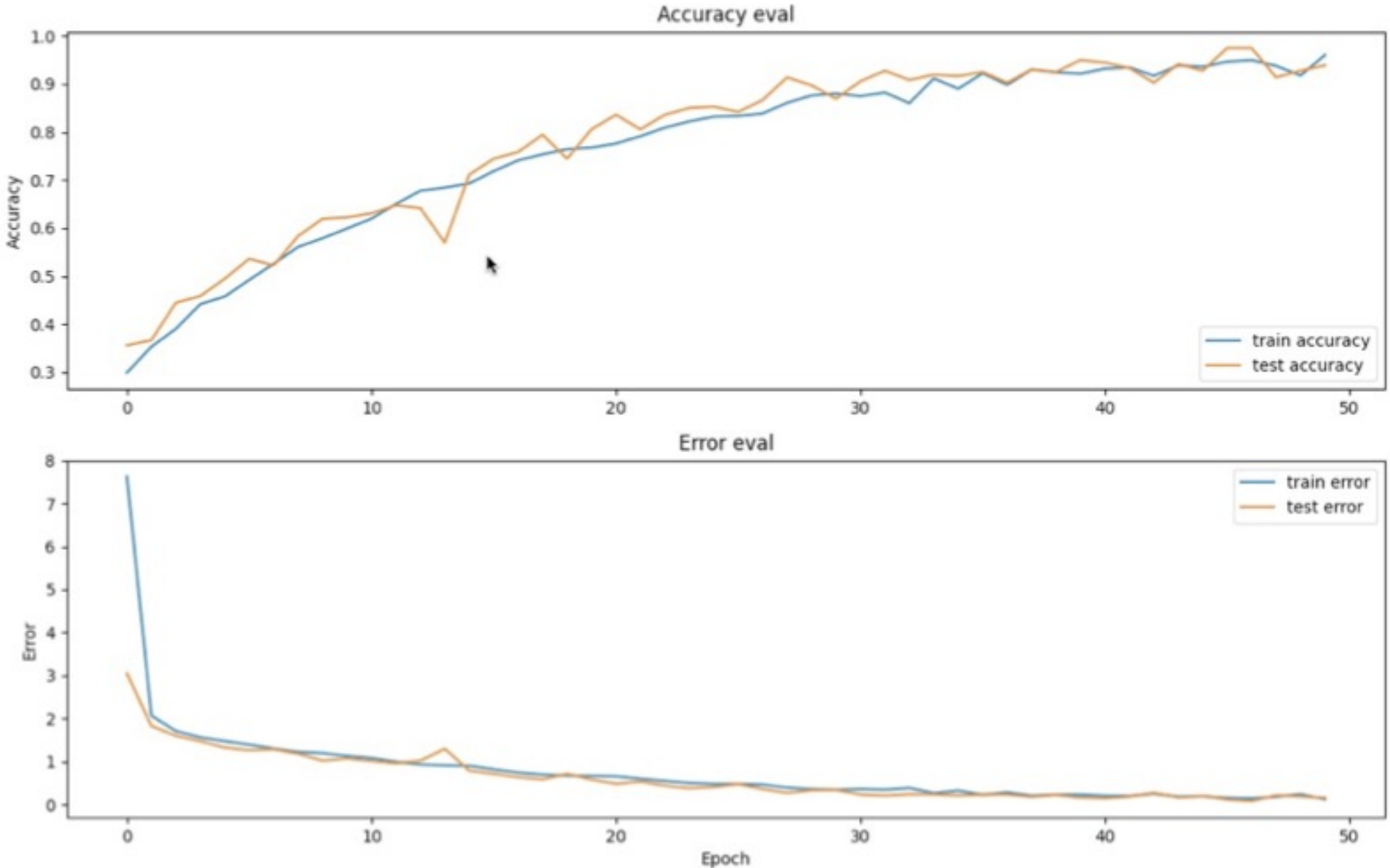
- The metrics that you choose to evaluate your machine learning algorithms are very important.
- Choice of metrics influences how the performance of machine learning algorithms is measured and compared. They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose.
- In this post, you will discover how to select and use different machine learning performance metrics in Python with scikit-learn.

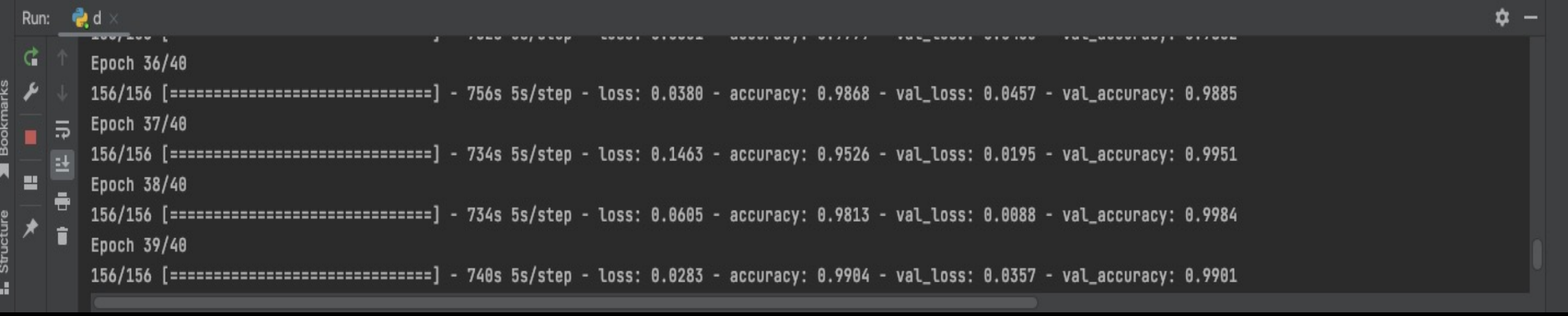
I have also created a predict function which randomly takes images from one batch and test it whether the predicted results match from the original one.

```
148
149
150     def predict(model, img):
151         img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
152         img_array = tf.expand_dims(img_array, 0)
153
154         predictions = model.predict(img_array)
155
156         predicted_class = class_names[np.argmax(predictions[0])]
157         confidence = round(100 * (np.max(predictions[0])), 2)
158         return predicted_class, confidence
159
160 plt.figure(figsize=(15, 15))
161 for images, labels in test_ds.take(1): #take any one batch
162     for i in range(9):
163         ax = plt.subplot(3, 3, i + 1)
164         plt.imshow(images[i].numpy().astype("uint8"))
165
166         predicted_class, confidence = predict(model, images[i].numpy())
167         actual_class = class_names[labels[i]]
168
169         plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")
170
171         plt.axis("off")
172
173
```

PLOTS:

```
173
174  from fontTools.varLib import plot
175
176
177  def plot_history(history):
178      fig, axs = plot.subplots(2)
179      axs[0].plot(history.history["accuracy"], label="train accuracy")
180      axs[0].plot(history.history["val_accuracy"], label="test accuracy")
181      axs[0].set_ylabel("Accuracy")
182      axs[0].legend(loc="lower right")
183      axs[0].set_title("Accuracy eval")
184      axs[1].plot(history.history["loss"], label="train error")
185      axs[1].plot(history.history["val_loss"], label="test error")
186      axs[1].set_ylabel("Error")
187      axs[1].set_xlabel("Epoch")
188      axs[1].legend(loc="upper right")
189      axs[1].set_title("Error eval")
190      plot.show()
191
192  model.save("models/fruits.h5")
193
```


Accuracy eval
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50
train accuracy
test accuracy
Error eval
Error
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
Epoch
train error
test error



As shown above I was able to achieve the accuracy upto 99.84%

END