IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

IMAGE CLASSIFICATION PROJECT DOCUMENTATION

PROJECT OVERVIEW:

Objective:

The objective of this project is to develop a Convolutional Neural Network (CNN) Model for image classification using a large labeled dataset. This model is got trained by using the CIFAR-10 dataset and classifies images into one of 10 predefined classes.

Tools and Libraries:

- Tensorflow
- Keras
- NumPy
- Matplotlib

DATASET:

CIFAR-10 Dataset:

The CIFAR-10 dataset consists of 60,000 32x32 colour images in 10 different classes.

The classes are as follows:

1) Airplane
2) Automobile
3) Bird
4) Cat
5) Deer
6) Dog
7) Frog
8) Horse
9) Ship
10) Truck

These 60,000 images were separated into training and testing data while loading the data.

train_images.shape => (50,000 ,32,32,3)

train_images.shape => (10,000 ,32,32,3)

train_labels.shape => (50,000, 1)

test_labels.shape => (10,000,1)

Preprocessing:

Image Normalization:

All images in the dataset were normalized to a range between 0 and 1. This normalization process helps standardize pixel values, making the training process more stable and facilitating faster convergence.

```
train_images = train_images/255
test_images = test_images/255
```

One-Hot Encoding:

To facilitate categorical classification, the labels for each image were one-hot encoded. This process transforms the class labels into a binary matrix, where each class corresponds to a unique column.

```
train_labels = to_categorical (train_labels)
test_labels = to_categorical (test_labels)
```

Data-Augmentation:

Data augmentation techniques were applied using the ImageDataGenerator class from Keras. These techniques include rotation, width and height shifts, and horizontal flipping. Data augmentation increases the diversity of the training set, enabling the model to generalize better to unseen data.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    )
datagen.fit(train_images)
```

MODEL ARCHITECTURE:

Layers:

1) Convolutional layers with ReLU activation, Batch Normalization, and MaxPooling.

   Batch Normalization:

   Batch Normalization normalizes the input of each layer by adjusting and scaling the activations before going into another layer.
   It also acts as a regularizer and stabilizes the training process.
   Improves generalization.

   MaxPooling:

   Reduces spatial dimensions of feature maps and captures important information discarding less relevant details.

2) Flatten layer.

   Before passing input to the Dense layers we use a Flatten layer to convert the multi-dimensional input to a one-dimensional array

3) Dense layers with ReLU activation, Batch Normalization, and Dropout for regularization.
4) Output layer with softmax activation function.
5) Learning rate schedule.
   This model has a learning rate schedule with intial value of 0.001 and multiplies itself with 0.1 for every 30 epochs
   1-30 => 0.001
   31-60 =>0.0001
   61-100 => 0.00001
   this schedule helps the model to converge better.

```
model=models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,
32, 3)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(2, 2))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(2, 2))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(2, 2))

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.20))

model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.20))

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

initial_lr = 0.001

def lr_schedule(epoch):
    lr = initial_lr
    if epoch > 30:
        lr *= 0.1
    if epoch > 60:
        lr *= 0.1
    if epoch > 90:
        lr *= 0.1
    return lr
model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=
['accuracy'])
```

TRAINING:

The model got trained with 100 epochs.

```
history = model.fit(datagen.flow(train_images, train_labels,
batch_size=32),steps_per_epoch=len(train_images) / 32,
epochs=100,validation_data=(test_images, test_labels),
callbacks=[LearningRateScheduler(lr_schedule)])
```

EVALUATION:

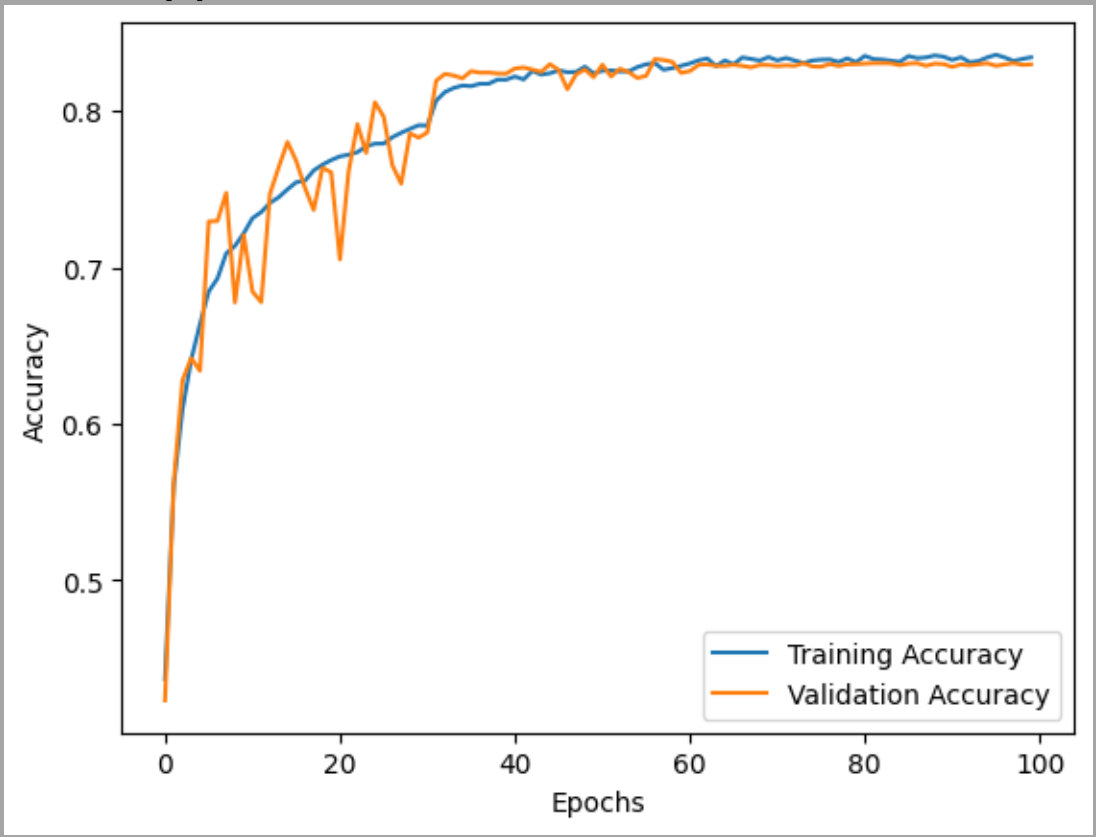
For evaluation I used F1score,Recall score.

```
from sklearn.metrics import f1_score, recall_score


test_accuracy, test_loss = model.evaluate(test_images,test_labels)


test_predictions = model.predict(test_images)
test_predictions_classes = np.argmax(test_predictions, axis=1)
test_true_classes = np.argmax(test_labels, axis=1)

test_accuracy = model.evaluate(test_images, test_labels)[1]
test_f1 = f1_score(test_true_classes, test_predictions_classes,
average='weighted')
test_recall = recall_score(test_true_classes, test_predictions_classes,
average='weighted')

print(f'Test Accuracy: {test_accuracy:.4f}')
print(f'Test F1 Score: {test_f1:.4f}')
print(f'Test Recall: {test_recall:.4f}')
```

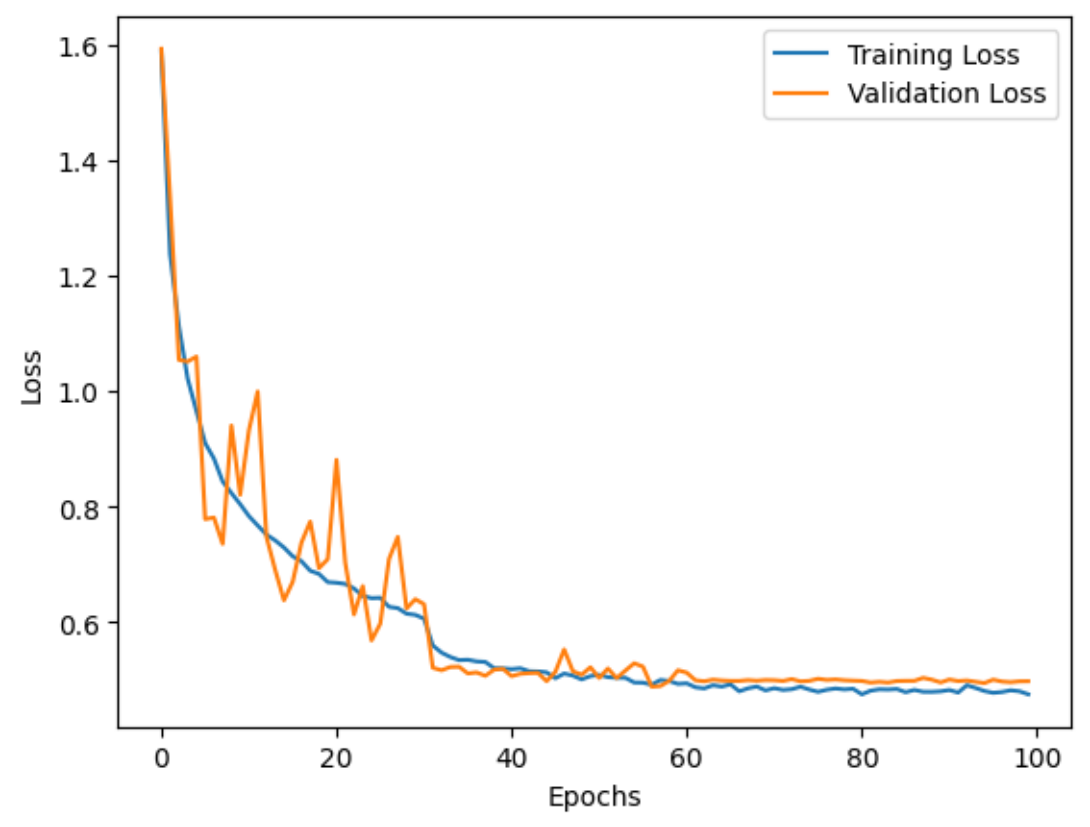
and the results are like this

Test accuracy: 0.8297
Test F1 score : 0.8269
Test Recall : 0.8297

Accuracy plot :



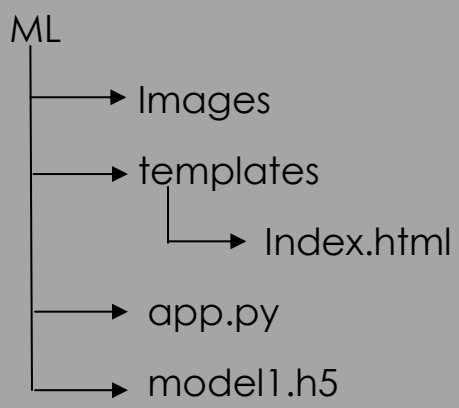
Loss:



DEPLOYMENT:

I have used Flask app for deploying the trained model and predict ne images. By using this app we can predict images on localhost:5678.

I have created a folder named ML and in it



Instructions to be followed for deploying:

1. Images which we want to test should be kept in the "Images" folder in .jpeg/.jpg/.png formats.
2. Run the app and ensure that path of the terminal is pointed to ML folder.
3. After running the app open locahost:5678 in the browser. http://localhost:5678
4. Click on choose file and select an image from the images folder and then click on predict.
5. If at any point the website is showing that it don't have access to the images folder please re-run the app.

We can also use this model in Colab,

Go to the Colab notebook and import model1.h5 into folders there than run the last code block in the terminal.

Instructions to use in Colab:

1. Upload model1.h5 to the folders.
2. Place the path of the model1.h5 after uploading into the "model_path" place.
3. Upload the image you want to predict into the "image_path" place.
4. Otherwise we can also use the test images for it.
5. Import data and run.

Colab Notebook for deploying:

https://colab.research.google.com/drive/1qPw-B9vqwWT31dix9qF3pt5ids3inC7P?usp=sharing

Colab Notebook of full project:

https://colab.research.google.com/drive/120a66U0jPjdAODfX-RlbQQcUAsvo14lc?usp=sharing