RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

CS19P18 - DEEP LEARNING CONCEPTS LABORATORY

LABORATORY RECORD NOTEBOOK

NAME: KIRUTHIGA M

YEAR/SEMESTER: IV/7

BRANCH: COMPUTER SCIENCE AND ENGINNEERING

REGISTER NO:220701132

COLLEGE ROLL NO:2116220701132

ACADEMIC YEAR: 2025 -2026

RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

BONAFIDE CERTIFICATE

NAME: KIRUTHIGA M **BRANCH/SECTION:** COMPUTER SCIENCE AND ENGINEERING **ACADEMIC YEAR:** 2025 -2026 **SEMESTER:** 7

REGISTER NO: 220701132

Certified that this is a Bonafide record of work done by the above student in the **CS19P18 - DEEP LEARNING CONCEPTS** during the year 2025 - 2026

Signature of Faculty In-charge

Submitted for the Practical Examination Held on: ……………………………………………..

Internal Examiner **External Examiner**

INDEX

INSTALLATION AND CONFIGURATION OF TENSORFLOW

Aim:

To install and configure TensorFlow in anaconda environment in Windows 10.

Procedure:

1. Download Anaconda Navigator and install.

2. Open Anaconda prompt

3. Create a new environment dlc with python 3.7 using the following command: conda create -n dlc python=3.7

4. Activate newly created environment dlc using the following command: conda activate dlc

5. In dlc prompt, install tensorflow using the following command: pip install tensorflow

6. Next install Tensorflow-datasets using the following command: pip install tensorflow-datasets

7. Install scikit-learn package using the following command: pip install scikit-learn

8. Install pandas package using the following command: pip install pandas

9. Lastly, install jupyter notebook pip install jupyter notebook

10. Open jupyter notebook bytyping the following in dlc prompt: jupyter notebook

11. Click create new and then choose python 3 (ipykernel)

12. Give the name to the file

13. Type the code and click Run button to execute (eg. Type import tensorflow and then run)

EX NO: 1 CREATE A NEURAL NETWORK TO RECOGNIZE HANDWRITTEN
DATE:14/07/2025 DIGITS USING MNIST DATASET

Aim:

To build a handwritten digit's recognition with MNIST dataset.

Procedure:

1. Download and load the MNIST dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

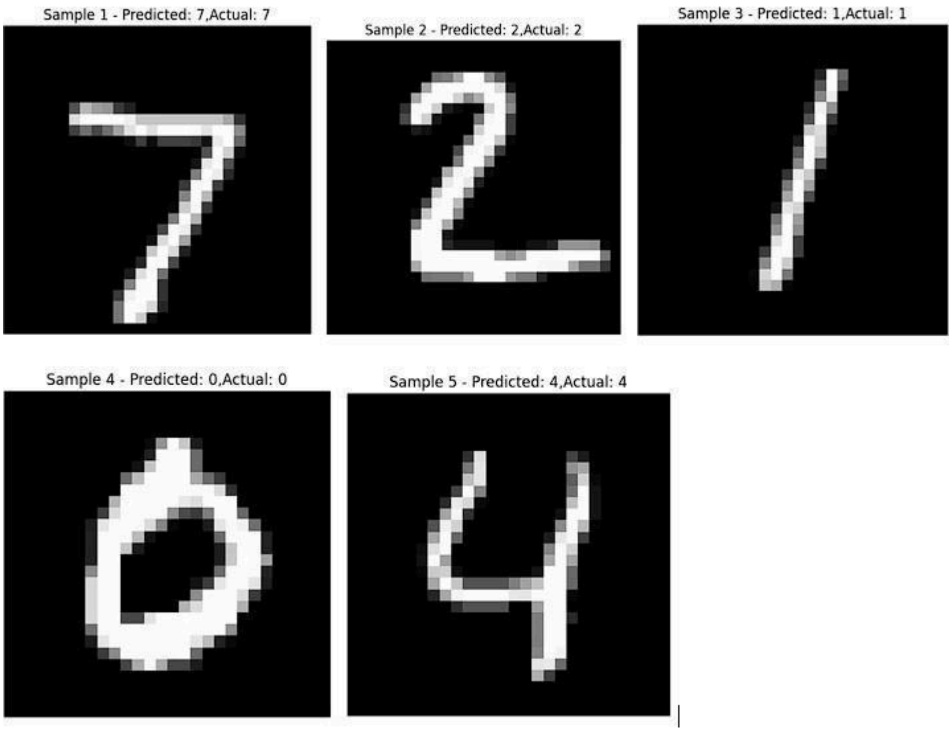
Code:

```
import numpy as np
import tensorflow as
tf
from tensorflow import keras

from sklearn.datasets import make_classification
from sklearn.model_selection import
train_test_split from sklearn.preprocessing import
StandardScaler from sklearn.metrics import
accuracy_score
# Generate a synthetic dataset

X, y = make_classification(n_samples=1000, n_features=20,
random_state=42) # Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Standardize features (optional but often beneficial)
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
```

```
X_test =
scaler.transform(X_test #
Define the model
model = keras.Sequential([
    keras.layers.Input(shape=(X_train.shape[1],)), # Input layer
    keras.layers.Dense(64, activation='relu'),    # Hidden layer with 64 neurons and ReLU
   activation keras.layers.Dense(1, activation='sigmoid') # Output layer with 1 neuron and
   sigmoid activation
)

# Train the model

history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.1) # Evaluate the model on the test set
y_pred = model.predict(X_test)
y_pred_classes = (y_pred >
0.5).astype(int) # Calculate accuracy on
the test set
accuracy = accuracy_score(y_test,
y_pred_classes) # Calculate test loss
test_loss = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy * 100:.2f}%")
print(f"Test loss: {test_loss[0]:.4f}")
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.7900 - loss: 0.7140 - val_accuracy: 0.9495 - val_loss: 0.1838
Epoch 2/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.9548 - loss: 0.1602 - val_accuracy: 0.9618 - val_loss: 0.1274
Epoch 3/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9718 - loss: 0.0971 - val_accuracy: 0.9684 - val_loss: 0.1091
Epoch 4/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.9806 - loss: 0.0686 - val_accuracy: 0.9705 - val_loss: 0.0958
Epoch 5/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.9863 - loss: 0.0487 - val_accuracy: 0.9750 - val_loss: 0.0859
Epoch 6/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.9903 - loss: 0.0376 - val_accuracy: 0.9739 - val_loss: 0.0868
Epoch 7/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.9917 - loss: 0.0293 - val_accuracy: 0.9726 - val_loss: 0.0937
Epoch 8/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9941 - loss: 0.0222 - val_accuracy: 0.9772 - val_loss: 0.0770
Epoch 9/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.9963 - loss: 0.0158 - val_accuracy: 0.9779 - val_loss: 0.0787
Epoch 10/10
192/192 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.9976 - loss: 0.0115 - val_accuracy: 0.9781 - val_loss: 0.0817
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9765 - loss: 0.0787
Test results - Loss: 0.07009122520685196 - Accuracy: 0.9800000190734863
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
```



Result:

Thus, the implementation to build a simple neural network using Keras/TensorFlow has been successfully executed.

EX NO:2 **BUILD A CONVOLUTIONAL NEURAL NETWORK**
DATE:21/07/2025 **USING KERAS/TENSORFLOW**

Aim:
To implement a Convolutional Neural Network (CNN) using Keras/TensorFlow to recognize and classify handwritten digits from the MNIST dataset with high accuracy.

Procedure:
1. Import required libraries (TensorFlow/Keras, NumPy, etc.).
2. Load the MNIST dataset from Keras.
3. Normalize and reshape the image data.
4. Convert labels to one-hot encoded vectors.
5. Build a CNN model with Conv2D, MaxPooling, Flatten, and Dense layers.
6. Compile the model using categorical crossentropy and Adam optimizer.
7. Train the model on training data.
8. Evaluate the model on test data.
9. Display accuracy and predictions.

Code:

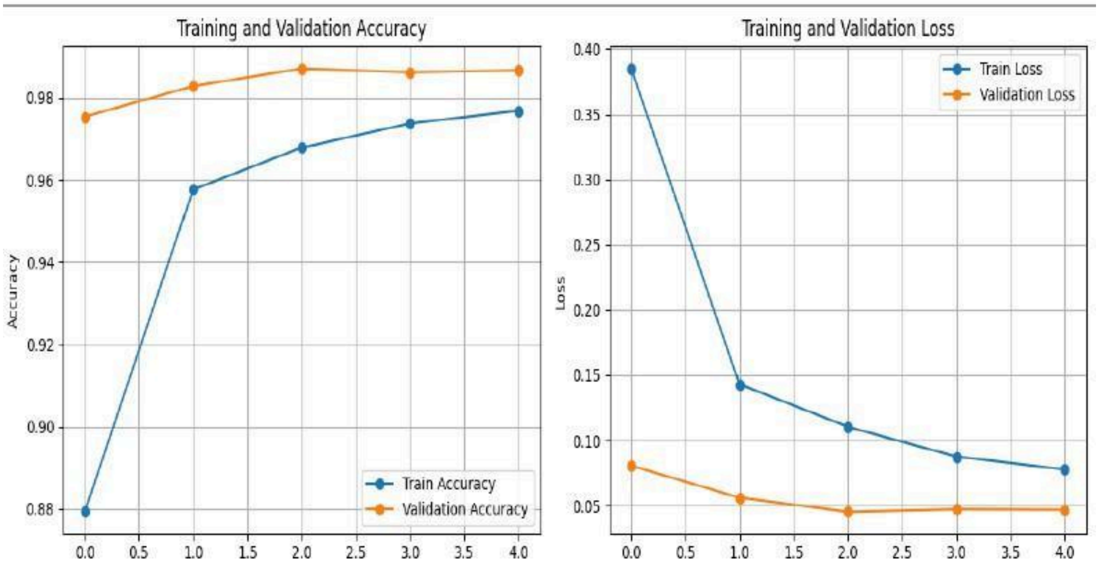
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as
plt import numpy as np

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
train_images = train_images.reshape(-1, 28, 28, 1)
test_images = test_images.reshape(-1, 28, 28, 1)
model = Sequential([
Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
MaxPooling2D((2, 2)),
Conv2D(64, (3, 3), activation='relu'),
MaxPooling2D((2, 2)),
Flatten(),
Dense(64, activation='relu'),
Dropout(0.5),
Dense(10, activation='softmax')
])
```

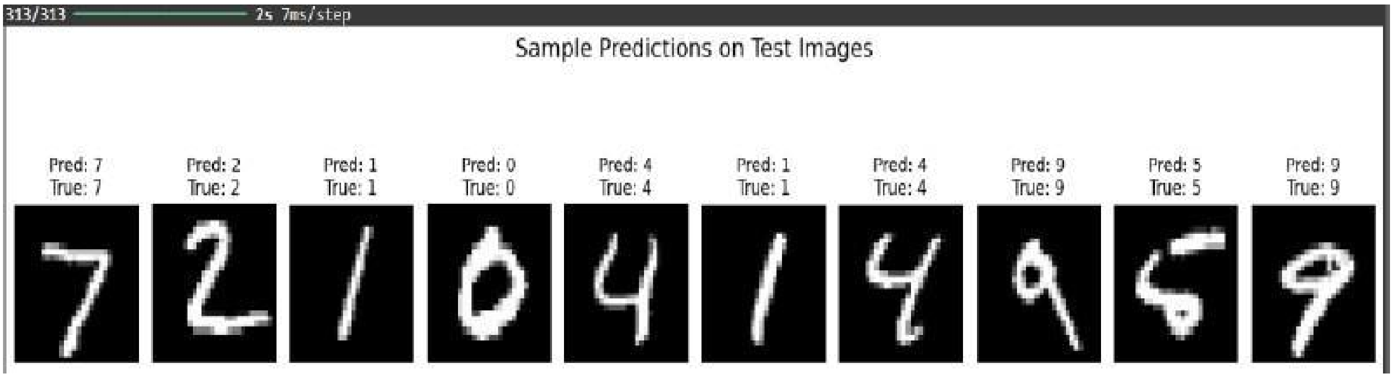
```
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history = model.fit(train_images,
train_labels, epochs=5,
batch_size=64,
validation_split=0.2)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"\n Test accuracy: {test_acc:.4f}")
print(f" Test loss: {test_loss:.4f}")

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy',
marker='o') plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch'
) plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions,
axis=1)

num_samples = 10
plt.figure(figsize=(15, 4))

for i in range(num_samples):
plt.subplot(1, num_samples, i + 1)
plt.imshow(test_images[i].reshape(28, 28), cmap='gray')
plt.title(f"Pred: {predicted_labels[i]}\nTrue: {test_labels[i]}")
```

plt.axis('off')
plt.suptitle("Sample Predictions on Test Images", fontsize=16)
plt.show()

Output:

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a lay
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 38s 48ms/step - accuracy: 0.7641 - loss: 0.7198 - val_accuracy: 0.9754 - val_loss: 0.0806
Epoch 2/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9542 - loss: 0.1575 - val_accuracy: 0.9828 - val_loss: 0.0559
Epoch 3/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9667 - loss: 0.1147 - val_accuracy: 0.9871 - val_loss: 0.0452
Epoch 4/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9737 - loss: 0.0872 - val_accuracy: 0.9862 - val_loss: 0.0472
Epoch 5/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 49ms/step - accuracy: 0.9753 - loss: 0.0801 - val_accuracy: 0.9867 - val_loss: 0.0468
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9845 - loss: 0.0458
```





Result:

Thus, the Convolution Neural Network (CNN) using Keras / Tensorflow to recognize and classify handwritten digits from MNIST dataset has been implemented successfully.

EX NO: 3 IMAGE CLASSIFICATION ON CIFAR-10 DATASET USING CNN

DATE:28/07/2025

Aim:

To build a Convolutional Neural Network (CNN) model for classifying images from the CIFAR-10 dataset into one of the ten categories such as airplanes, cars, birds, cats, etc.

Procedure:

1. Download and load the CIFAR-10 dataset using Keras/TensorFlow.
2. Visualize and analyze sample images from the

dataset. 3, Preprocess the data:

- Normalize the pixel values (divide by 255)
- Convert class labels to one-hot encoded format

4. Build a CNN model using Keras/TensorFlow:
   - Include convolutional, pooling, flatten, and dense layers.
5. Compile the model with suitable loss function and optimizer.
6. Train the model using training data and validate using test data.
7. Evaluate the model using accuracy and loss on test dataset.
8. Perform predictions on new/unseen CIFAR-10 images.

9 Visualize prediction results with sample images and predicted labels.

Code:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(32, (3,3), activation='relu',
input_shape=(32,32,3))) model.add(tf.keras.layers.MaxPooling2D((2,2)))
model.add(tf.keras.layers.Conv2D(64, (3,3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2,2)))
model.add(tf.keras.layers.Conv2D(64, (3,3), activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) model.fit(x_train, y_train, epochs=10, batch_size=64,
validation_split=0.2) class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
```

```
'dog', 'frog', 'horse', 'ship', 'truck']
index = int(input("Enter an index (0 to 9999) for test image: "))
if index < 0 or index >= len(x_test):
print("Invalid index. Using index 0 by default.")
index = 0
test_image = x_test[index]
true_label = np.argmax(y_test[index])
prediction = model.predict(np.expand_dims(test_image,
axis=0)) predicted_label = np.argmax(prediction)
plt.figure(figsize=(4, 4))
resized_image = tf.image.resize(test_image, [128, 128])
plt.imshow(resized_image)
plt.axis('off')
plt.title(f"Predicted: {class_names[predicted_label]}\nActual: {class_names[true_label]}")
plt.show()
```

Output:

```
loading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
98071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
h 1/10
625 ━━━━━━━━━━━━━━━━━━━━ 51s 79ms/step - accuracy: 0.3104 - loss: 1.8651 - val_accuracy: 0.5242 - val_loss: 1.3433
h 2/10
625 ━━━━━━━━━━━━━━━━━━━━ 51s 81ms/step - accuracy: 0.5359 - loss: 1.3072 - val_accuracy: 0.5898 - val_loss: 1.1679
h 3/10
625 ━━━━━━━━━━━━━━━━━━━━ 88s 91ms/step - accuracy: 0.5960 - loss: 1.1372 - val_accuracy: 0.6009 - val_loss: 1.1314
h 4/10
625 ━━━━━━━━━━━━━━━━━━━━ 50s 81ms/step - accuracy: 0.6345 - loss: 1.0355 - val_accuracy: 0.6460 - val_loss: 1.0102
h 5/10
625 ━━━━━━━━━━━━━━━━━━━━ 84s 83ms/step - accuracy: 0.6662 - loss: 0.9524 - val_accuracy: 0.6433 - val_loss: 1.0243
h 6/10
625 ━━━━━━━━━━━━━━━━━━━━ 51s 82ms/step - accuracy: 0.6868 - loss: 0.8809 - val_accuracy: 0.6774 - val_loss: 0.9246
h 7/10
625 ━━━━━━━━━━━━━━━━━━━━ 49s 78ms/step - accuracy: 0.7089 - loss: 0.8233 - val_accuracy: 0.6843 - val_loss: 0.9061
h 8/10
625 ━━━━━━━━━━━━━━━━━━━━ 49s 78ms/step - accuracy: 0.7337 - loss: 0.7592 - val_accuracy: 0.6893 - val_loss: 0.8919
h 9/10
625 ━━━━━━━━━━━━━━━━━━━━ 83s 80ms/step - accuracy: 0.7495 - loss: 0.7176 - val_accuracy: 0.7007 - val_loss: 0.8667
h 10/10
625 ━━━━━━━━━━━━━━━━━━━━ 83s 82ms/step - accuracy: 0.7608 - loss: 0.6766 - val_accuracy: 0.6884 - val_loss: 0.9299
━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
```

Result

Thus, the Convolution Neural Network (CNN) model for classifying images from CIFAR-10 dataset is implemented successfully.

Ex No: 4 TRANSFER LEARNING WITH CNN AND VISUALIZATION
DATE:04/08/2025

Aim:

To build a convolutional neural network with transfer learning and perform visualization

Procedure:

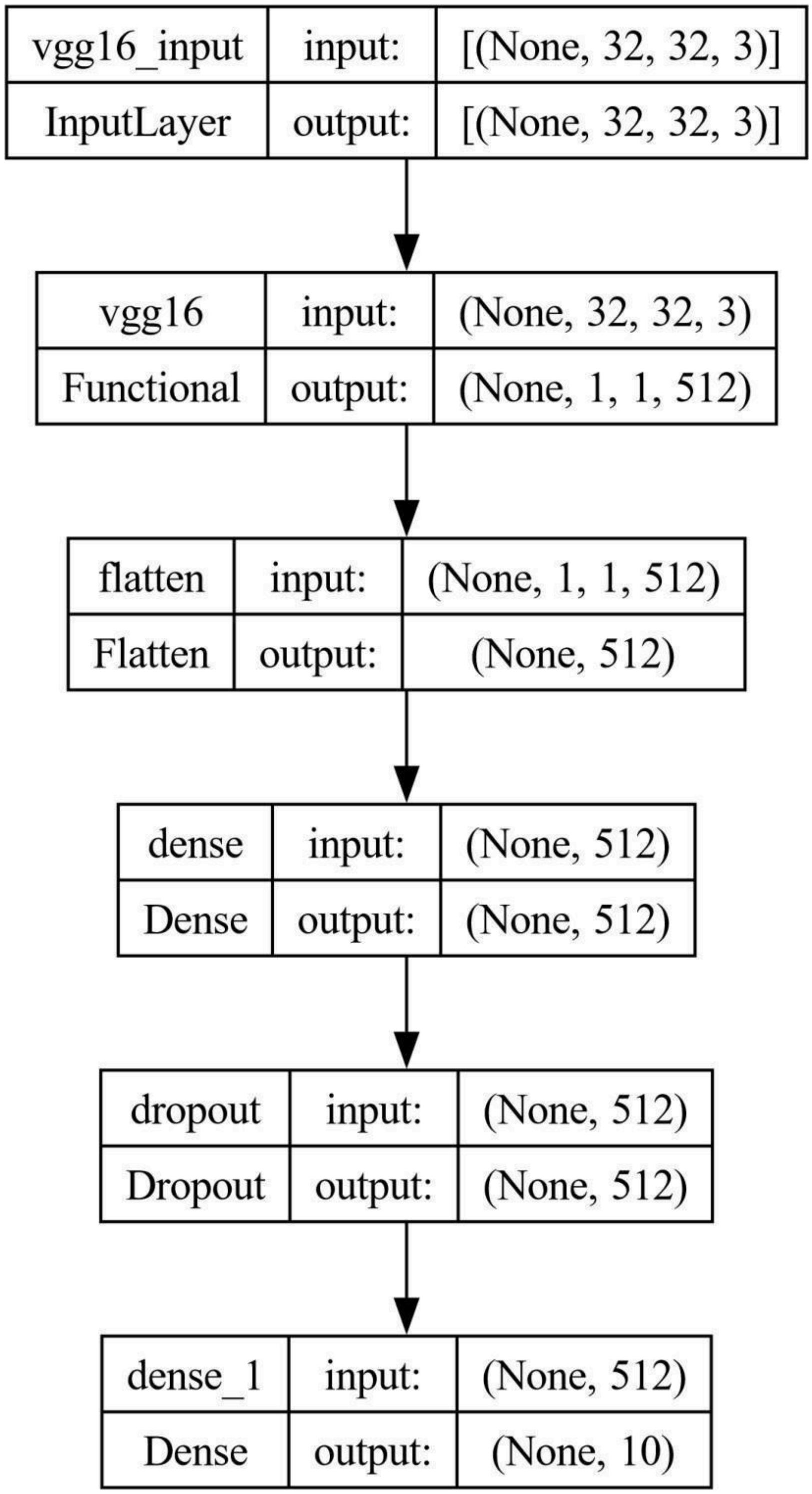
1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Code:

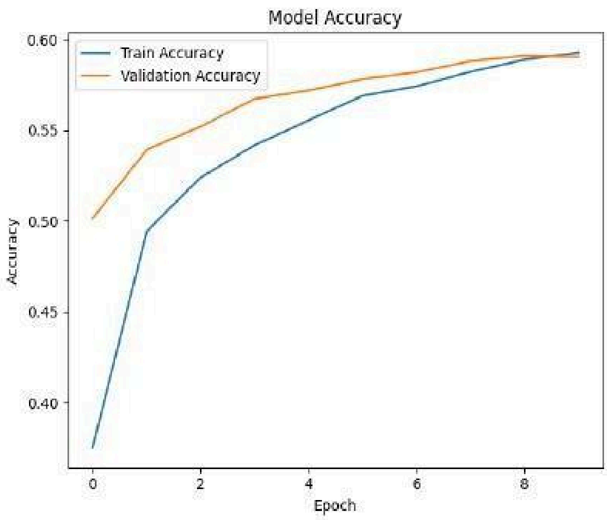
```
conda install -c conda-forge python-graphviz -y
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import
plot_model import matplotlib.pyplot as plt
import numpy as np
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
for layer in vgg_base.layers:
layer.trainable = False
model = Sequential()
model.add(vgg_base)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer=Adam(learning_rate=0.0001),
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
plot_model(model, to_file='cnn.png', show_shapes=True,
```

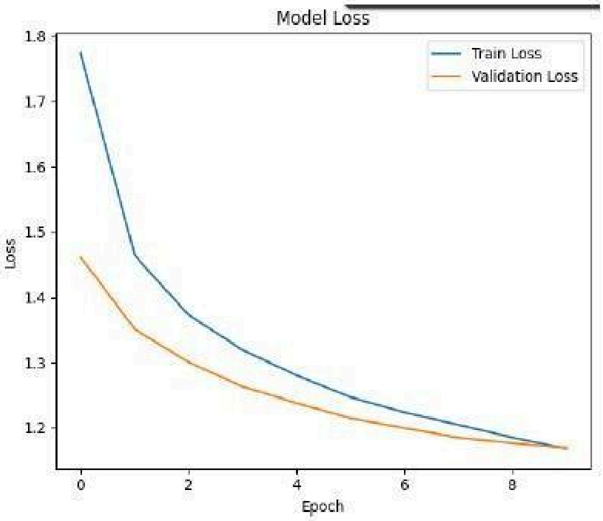
```
show_layer_names=True, dpi=300)
plt.figure(figsize=(20, 20))
img = plt.imread('cnn.png')
plt.imshow(img)
plt.axis('off')
plt.show()
history = model.fit(x_train,
y_train, epochs=10,
batch_size=32,
validation_split=0.2)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_acc * 100:.2f}%')
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
'dog', 'frog', 'horse', 'ship', 'truck']
sample = x_test[0].reshape(1, 32, 32, 3)
prediction = model.predict(sample)
predicted_class = class_names[np.argmax(prediction)]

plt.imshow(x_test[0])
plt.title(f"Predicted: {predicted_class}")
plt.axis('off')
plt.show()
```

Output:


vgg16_input
input:
[(None, 32, 32, 3)]
InputLayer
output:
[(None, 32, 32, 3)]
vgg16
input:
(None, 32, 32, 3)
Functional
output:
(None, 1, 1, 512)
flatten
input:
(None, 1, 1, 512)
Flatten
output:
(None, 512)
dense
input:
(None, 512)
Dense
output:
(None, 512)
dropout
input:
(None, 512)
Dropout
output:
(None, 512)
dense_1
input:
(None, 512)
Dense
output:
(None, 10)

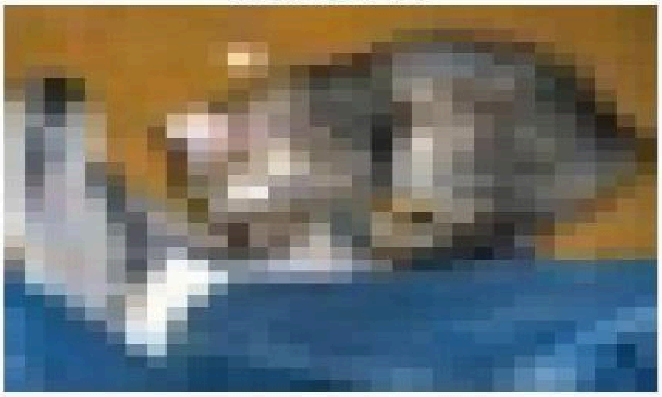
```
Epoch 1/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 580s 463ms/step - accuracy: 0.2851 - loss: 2.0021 - val_accuracy: 0.5015 - val_loss: 1.4614
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 618s 460ms/step - accuracy: 0.4840 - loss: 1.4918 - val_accuracy: 0.5393 - val_loss: 1.3512
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 574s 460ms/step - accuracy: 0.5235 - loss: 1.3748 - val_accuracy: 0.5521 - val_loss: 1.3007
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 624s 461ms/step - accuracy: 0.5373 - loss: 1.3335 - val_accuracy: 0.5674 - val_loss: 1.2630
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 577s 462ms/step - accuracy: 0.5587 - loss: 1.2804 - val_accuracy: 0.5720 - val_loss: 1.2381
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 538s 430ms/step - accuracy: 0.5692 - loss: 1.2482 - val_accuracy: 0.5783 - val_loss: 1.2146
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 600s 461ms/step - accuracy: 0.5717 - loss: 1.2292 - val_accuracy: 0.5821 - val_loss: 1.1994
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 620s 460ms/step - accuracy: 0.5770 - loss: 1.2138 - val_accuracy: 0.5882 - val_loss: 1.1849
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 575s 460ms/step - accuracy: 0.5898 - loss: 1.1833 - val_accuracy: 0.5911 - val_loss: 1.1766
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 624s 461ms/step - accuracy: 0.5937 - loss: 1.1666 - val_accuracy: 0.5905 - val_loss: 1.1690
313/313 ━━━━━━━━━━━━━━━━━━━━ 108s 345ms/step - accuracy: 0.5793 - loss: 1.1799
Test Loss: 1.1821
Test Accuracy: 58.40%
```






```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 251ms/step
```

Predicted: cat



Result

Thus, the Convolution Neural Network (CNN) with transfer learning and perform visualization has been implemented successfully

EX NO: 5 BUILD A RECURRENT NEURAL NETWORK (RNN) USING KERAS/TENSORFLOW

DATE:25/08/2025

Aim:

To build a recurrent neural network with Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Code:

```
import numpy as np
import tensorflow as
tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN,
Dense from sklearn.metrics import r2_score
np.random.seed(0)
seq_length = 10
num_samples = 1000
X = np.random.randn(num_samples, seq_length, 1)
y = X.sum(axis=1) + 0.1 * np.random.randn(num_samples, 1)
split_ratio = 0.8
split_index = int(split_ratio * num_samples)
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]
model = Sequential()
model.add(SimpleRNN(units=50, activation='relu', input_shape=(seq_length, 1)))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()
batch_size = 30
epochs = 50 # Reduced epochs for quick demonstration
history = model.fit(
X_train, y_train,
batch_size=batch_size
, epochs=epochs,
validation_split=0.2
)
test_loss = model.evaluate(X_test, y_test)
```

```
print(f'Test Loss: {test_loss:.4f}')
```

```
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print(f'Test Accuracy (R^2): {r2:.4f}')

new_data = np.random.randn(5, seq_length, 1)
predictions = model.predict(new_data)
print("Predictions for new data:")
print(predictions)
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 50)                2600

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 2,651
Trainable params: 2,651
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/50
22/22 [==============================] - 2s 23ms/step - loss: 8.7454
- val_loss: 6.3263
Epoch 2/50
22/22 [==============================] - 0s 4ms/step - loss: 5.8837
- val_loss: 3.7798
Epoch 3/50
22/22 [==============================] - 0s 5ms/step - loss: 3.7728
- val_loss: 2.3105
Epoch 4/50
22/22 [==============================] - 0s 5ms/step - loss: 1.7141
- val_loss: 0.5373
Epoch 5/50
22/22 [==============================] - 0s 4ms/step - loss: 0.2878
- val_loss: 0.2417
Epoch 6/50
22/22 [==============================] - 0s 4ms/step - loss: 0.1304
- val_loss: 0.1146
Epoch 7/50
```

```
1/1 [==============================] - 0s 20ms/step
Predictions for new data:
[[ 1.5437698]
 [ 0.4290885]
 [-2.1180325]
 [-0.5443404]
 [-3.8416493]]
```

Result:

Thus, the Recurrent Neural Network (RNN) has been implemented using Tensorflow.

EX NO: 6 **SENTIMENT CLASSIFICATION OF TEXT USING RNN**
DATE:15/09/2025

Aim:

To implement a Recurrent Neural Network (RNN) using Keras/TensorFlow for classifying the sentiment of text data (e.g., movie reviews) as positive or negative.

Procedure:

1. Import necessary libraries.
2. Load and preprocess the text dataset (e.g., IMDb).
3. Pad sequences and prepare labels.
4. Build an RNN model with Embedding and SimpleRNN layers.
5. Compile the model with loss and optimizer.
6. Train the model on training data.
7. Evaluate the model on test data.
8. Predict sentiment for new inputs

Code:

```
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import
pad_sequences from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
max_words = 5000
max_len = 200
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_words)
X_train = pad_sequences(x_train, maxlen=max_len)
X_test = pad_sequences(x_test,
maxlen=max_len) model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=32, input_length=max_len))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print("Training...")
model.fit(X_train, y_train, epochs=2, batch_size=64, validation_split=0.2)
loss, acc = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {acc:.4f}")
word_index = imdb.get_word_index()
reverse_word_index = {v: k for (k, v) in word_index.items()}
```

```
def decode_review(review):
return " ".join([reverse_word_index.get(i - 3, "?") for i in review])
sample_review = X_test[0]
prediction = model.predict(sample_review.reshape(1, -1))[0][0]
print("\nReview text:", decode_review(x_test[0]))
print("Predicted Sentiment:", "Positive " if prediction > 0.5 else "Negative ")
```

Output:

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - accuracy: 0.6261 - loss: 0.6246 - val_accuracy: 0.8182 - val_loss: 0.4225
Epoch 2/2
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - accuracy: 0.8167 - loss: 0.4321 - val_accuracy: 0.8284 - val_loss: 0.4171
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8329 - loss: 0.4071

Test Accuracy: 0.8304
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step

Review text: ? please give this one a miss br br ? ? and the rest of the cast ? terrible performances the show is flat flat flat br br i don't know how m
ichael ? could have allowed this one on his ? he almost seemed to know this wasn't going to work out and his performance was quite ? so all you ? fans gi
ve this a miss
Predicted Sentiment: Negative
```

Result

Thus, the Recurrent Neural Network (RNN) using Keras has been implemented for classifying sentiment of text successfully.

Ex No: 7 **BUILD AUTOENCODERS WITH KERAS/TENSORFLOW**

DATE:22/09/2025

Aim:

To build autoencoders with Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from keras.layers import Input,
Dense from keras.models import
Model from keras.datasets import
mnist
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train),
np.prod(x_train.shape[1:]))) x_test = x_test.reshape((len(x_test),
np.prod(x_test.shape[1:]))) input_img = Input(shape=(784,))
encoded = Dense(32, activation='relu')(input_img)
decoded = Dense(784,
activation='sigmoid')(encoded) autoencoder =
Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train,
epochs=50,
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test))
test_loss = autoencoder.evaluate(x_test, x_test)
decoded_imgs = autoencoder.predict(x_test)
threshold = 0.5
correct_predictions = np.sum(
np.where(x_test >= threshold, 1, 0) ==
np.where(decoded_imgs >= threshold, 1, 0))
total_pixels = x_test.shape[0] * x_test.shape[1]
test_accuracy = correct_predictions / total_pixels
```

```
print("Test Loss:", test_loss)
```

print("Test Accuracy:", test_accuracy)
n = 10
plt.figure(figsize=(20,
4)) for i in range(n):
# Display original
ax = plt.subplot(2, n, i + 1)
plt.imshow(x_test[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
# Display reconstruction with threshold
ax = plt.subplot(2, n, i + 1 + n)
reconstruction = decoded_imgs[i].reshape(28, 28)
plt.imshow(np.where(reconstruction >= threshold, 1.0, 0.0))
plt.gray()
ax.get_xaxis().set_visible(Fals
e)
ax.get_yaxis().set_visible(Fals
e) plt.show()

Output:

```
Epoch 1/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - loss: 0.3805 - val_loss: 0.1906
Epoch 2/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.1808 - val_loss: 0.1547
Epoch 3/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.1501 - val_loss: 0.1342
Epoch 4/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1321 - val_loss: 0.1221
Epoch 5/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1210 - val_loss: 0.1138
Epoch 6/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.1134 - val_loss: 0.1081
Epoch 7/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - loss: 0.1079 - val_loss: 0.1039
Epoch 8/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.1042 - val_loss: 0.1006
Epoch 9/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - loss: 0.1011 - val_loss: 0.0981
Epoch 10/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0989 - val_loss: 0.0963
Epoch 11/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0972 - val_loss: 0.0951
Epoch 12/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0964 - val_loss: 0.0943
Epoch 13/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0954 - val_loss: 0.0938
Epoch 14/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0950 - val_loss: 0.0934
Epoch 15/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0944 - val_loss: 0.0932
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
Test Loss: 0.09166844934225082
Test Accuracy: 0.9712756377551021
```

Result

Thus, an Autoencoder has been implemented using Keras / Tensorflow.

Ex No:8 **OBJECT DETECTION WITH YOLO3**

DATE:29/09/2025

Aim:

To build an object detection model with YOLO3 using Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Code:

```
import cv2
import matplotlib.pyplot as
plt import numpy as np
import os

# --- 1. DEFINE FILE PATHS ---
# This is the only version you should use.
# Your test proved all these files are in the correct place.
cfg_file = 'yolov3.cfg'
weight_file =
'yolov3.weights' names_file =
'coco.names'
image_path = 'my_image.jpg' # <-- THIS IS THE CORRECT PATH

# --- 2. LOAD YOLO MODEL AND CLASS NAMES ---
try:
   net = cv2.dnn.readNet(weight_file, cfg_file)
except cv2.error as e:
   print(f"Error loading model: {e}")
   exit()

with open(names_file, 'r') as f:
   classes =
   f.read().strip().split('\n')

# --- 3. LOAD AND PROCESS THE IMAGE ---
image = cv2.imread(image_path)

if image is None:
   print(f"Error: Could not read the image file at
   '{image_path}'") # Add a printout of files to help debug if it
```

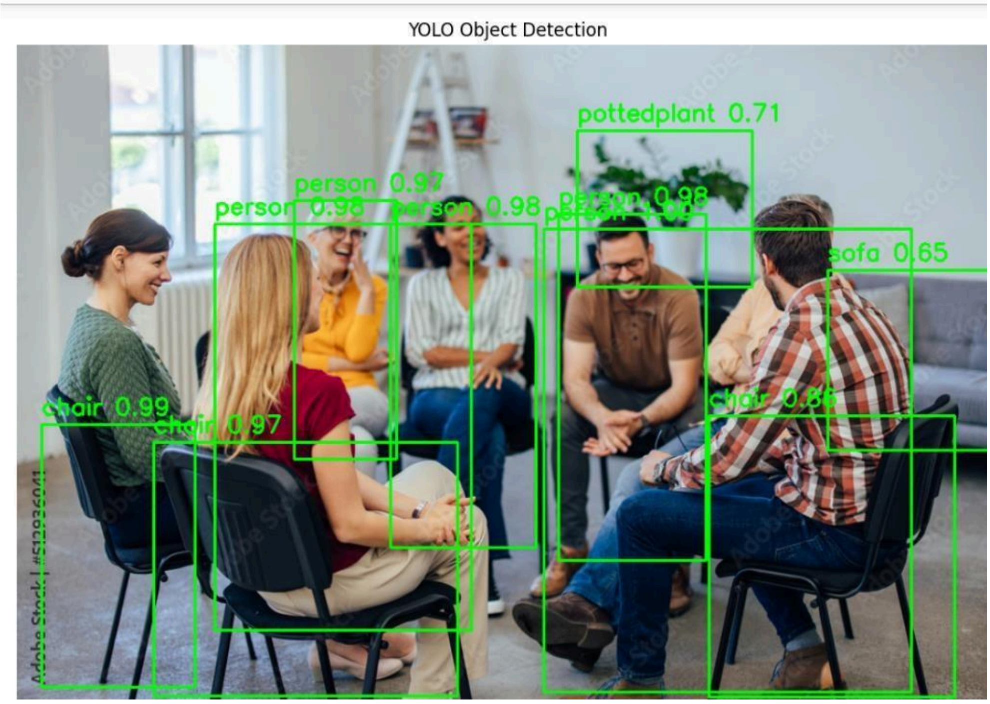
fails again print(f"Files in the current directory are:
{os.listdir()}")
else:

```
height, width = image.shape[:2]
blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)
net.setInput(blob)

# --- 4. RUN FORWARD PASS AND GET DETECTIONS ---
layer_names = net.getUnconnectedOutLayersNames()
outs = net.forward(layer_names)

# --- 5. PROCESS DETECTIONS AND APPLY NMS ---
class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4

for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence >
        conf_threshold:
            center_x = int(detection[0] *
            width) center_y = int(detection[1]
            * height) w = int(detection[2] *
            width)
            h = int(detection[3] *
            height) x = int(center_x - w
            / 2)
            y = int(center_y - h / 2)
            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold,

nms_threshold) # --- 6. DRAW BOUNDING BOXES AND DISPLAY THE

FINAL IMAGE ---

if len(indices) > 0:
    for i in indices.flatten():
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = confidences[i]
        color = (0, 255, 0)
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
```

```
cv2.putText(image, f'{label} {confidence:.2f}', (x, y - 10),
          cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)
```

```
plt.figure(figsize=(12, 10))
plt.imshow(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)) plt.title("YOLO Object
Detection")
plt.axis('off')
plt.show()
```

Output:



Result

Thus, object detection using YOLOV5 has been implemented successfully.

Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK
DATE:29/09/2025

Aim:

To build a generative adversarial neural network using Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import
Sequential from tensorflow.keras.optimizers
import Adam from sklearn.datasets import
load_iris
import matplotlib.pyplot as plt

# Load and Preprocess the Iris
Dataset iris = load_iris()
x_train = iris.data

# Build the GAN model
def build_generator():
model = Sequential()
model.add(Dense(128, input_shape=(100,), activation='relu'))
model.add(Dense(4, activation='linear')) # Output 4 features
return model

def build_discriminator():
model = Sequential()
model.add(Dense(128, input_shape=(4,), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
return model

def build_gan(generator,
discriminator): discriminator.trainable
= False
model = Sequential()
```

```
model.add(generator)
model.add(discriminator)
```

```
return model

generator = build_generator()
discriminator = build_discriminator()
gan = build_gan(generator, discriminator)

# Compile the Models
generator.compile(loss='mean_squared_error', optimizer=Adam(0.0002, 0.5))
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002,
0.5), metrics=['accuracy'])
gan.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))

# Training Loop
epochs = 200
batch_size = 16

for epoch in range(epochs):
# Train discriminator
idx = np.random.randint(0, x_train.shape[0], batch_size)
real_samples = x_train[idx]
fake_samples = generator.predict(np.random.normal(0, 1, (batch_size, 100)), verbose=0)

real_labels = np.ones((batch_size, 1))
fake_labels = np.zeros((batch_size,
1))

d_loss_real = discriminator.train_on_batch(real_samples, real_labels)
d_loss_fake = discriminator.train_on_batch(fake_samples,
fake_labels)

# Train generator
noise = np.random.normal(0, 1, (batch_size, 100))
g_loss = gan.train_on_batch(noise, real_labels)

# Print progress
print(f"Epoch {epoch}/{epochs} | Discriminator Loss: {0.5 * (d_loss_real[0] + d_loss_fake[0])} |
Generator Loss: {g_loss}")

# Generating Synthetic Data
synthetic_data = generator.predict(np.random.normal(0, 1, (150, 100)), verbose=0)

# Create scatter plots for feature pairs
plt.figure(figsize=(12, 8))
plot_idx = 1

for i in range(4):
```

```
for j in range(i + 1, 4):
plt.subplot(2, 3,
plot_idx)
```

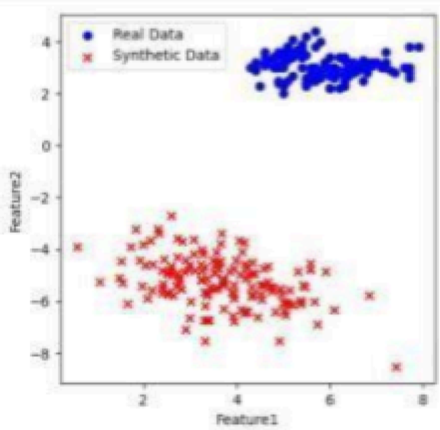
plt.scatter(x_train[:, i], x_train[:, j], label='Real Data', c='blue', marker='o', s=30)
plt.scatter(synthetic_data[:, i], synthetic_data[:, j], label='Synthetic Data', c='red', marker='x', s=30)
plt.xlabel(f'Feature {i + 1}') plt.ylabel(f'Feature {j + 1}') plt.legend()
plot_idx += 1

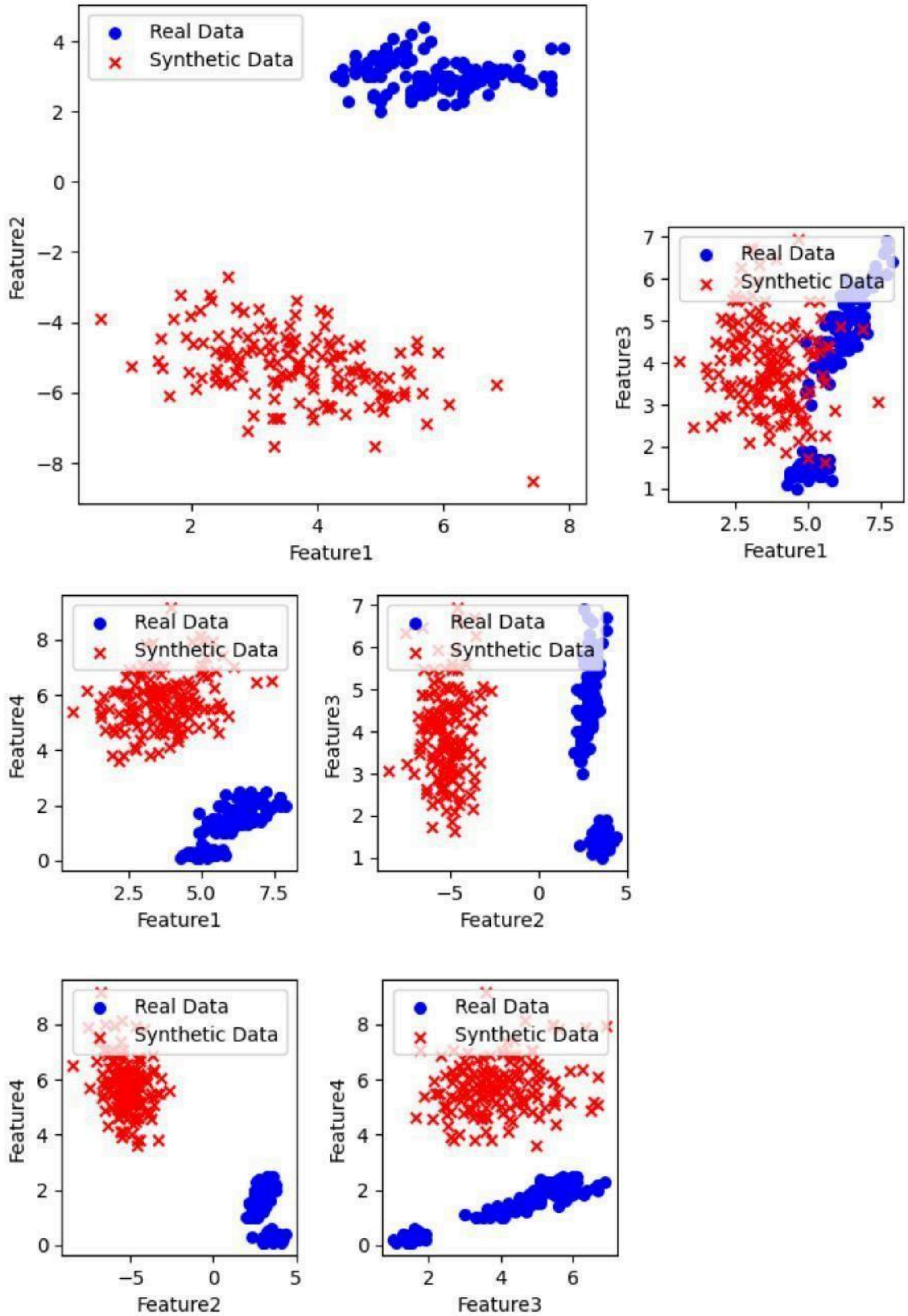
plt.tight_layout()
plt.show()

Output:

```
Epoch 0/200 | Discriminator Loss: 0.8773080408573151 |Generator Loss: 0.764731228351593
Epoch 1/200 | Discriminator Loss: 0.9332943856716156 |Generator Loss: 0.7988691329956055
Epoch 2/200 | Discriminator Loss: 0.9277275502681732 |Generator Loss: 0.8127573728561401
Epoch 3/200 | Discriminator Loss: 0.8921994566917419 |Generator Loss: 0.7757299542427063
Epoch 4/200 | Discriminator Loss: 0.913447916507721 |Generator Loss: 0.7737997174263
Epoch 5/200 | Discriminator Loss: 0.8916181325912476 |Generator Loss: 0.8003895282745361
Epoch 6/200 | Discriminator Loss: 0.9026078879833221 |Generator Loss: 0.814433217048645
Epoch 7/200 | Discriminator Loss: 0.9135120809078217 |Generator Loss: 0.8237183690071106
Epoch 8/200 | Discriminator Loss: 0.879832923412323 |Generator Loss: 0.7563657760620117
Epoch 9/200 | Discriminator Loss: 0.9439513385295868 |Generator Loss: 0.7623365521430969
Epoch 10/200 | Discriminator Loss: 0.9355685114860535 |Generator Loss: 0.7924684286117554
Epoch 11/200 | Discriminator Loss: 0.9386743903160095 |Generator Loss: 0.7614541053771973
Epoch 12/200 | Discriminator Loss: 0.960555225610733 |Generator Loss: 0.7792538404464722
Epoch 13/200 | Discriminator Loss: 0.9134297668933868 |Generator Loss: 0.792992115020752
Epoch 14/200 | Discriminator Loss: 0.8851655125617981 |Generator Loss: 0.7628173232078552
Epoch 15/200 | Discriminator Loss: 0.9505723416805267 |Generator Loss: 0.7851851582527161
Epoch 16/200 | Discriminator Loss: 0.92226842045784 |Generator Loss: 0.769191563129425
Epoch 17/200 | Discriminator Loss: 0.8982412815093994 |Generator Loss: 0.7685977220535278
Epoch 18/200 | Discriminator Loss: 0.9125983119010925 |Generator Loss: 0.7730982899665833
Epoch 19/200 | Discriminator Loss: 0.9367325305938721 |Generator Loss: 0.7837406396865845
Epoch 20/200 | Discriminator Loss: 0.9531015455722809 |Generator Loss: 0.7827053070068359
Epoch 21/200 | Discriminator Loss: 0.9306998252868652 |Generator Loss: 0.7667914032936096
Epoch 22/200 | Discriminator Loss: 0.8887360095977783 |Generator Loss: 0.7845874428749084
Epoch 23/200 | Discriminator Loss: 0.9426513016223907 |Generator Loss: 0.746765673160553
Epoch 24/200 | Discriminator Loss: 0.9331325888633728 |Generator Loss: 0.761589765548706
Epoch 25/200 | Discriminator Loss: 0.9080778360366821 |Generator Loss: 0.7709233164787292
Epoch 26/200 | Discriminator Loss: 0.9232879281044006 |Generator Loss: 0.7773635387420654
Epoch 27/200 | Discriminator Loss: 0.9102294743061066 |Generator Loss: 0.7809370756149292
Epoch 28/200 | Discriminator Loss: 0.9312145709991455 |Generator Loss: 0.7647197246551514
Epoch 29/200 | Discriminator Loss: 0.9415165781974792 |Generator Loss: 0.7561923861503601
Epoch 30/200 | Discriminator Loss: 0.930676281452179 |Generator Loss: 0.7709008455276489
Epoch 31/200 | Discriminator Loss: 0.9495892226696014 |Generator Loss: 0.7595088481903076
```

```
In [33]: synthetic_data = generator.predict(np.random.normal(0,1,(150,100)),verbose=0)
plt.figure(figsize=(12,8))
plot_idx=1

for i in range(4):
    for j in range(i+1,4):
        plt.subplot(2,3,plot_idx)
        plt.scatter(x_train[:,i],x_train[:,j],label='Real Data',c='blue',marker='o',s=30)
        plt.scatter(synthetic_data[:,i],synthetic_data[:,j],label='Synthetic Data',c='red',marker='x',s=30)
        plt.xlabel(f'Feature{i+1}')
        plt.ylabel(f'Feature{j+1}')
        plt.legend()
        plot_idx+=1

        plt.tight_layout()
        plt.show()
```





Result

Thus, a generative adversarial neural network using Keras / Tensorflow has been implemented successfully.

Ex No: 10 MINI PROECT – COLOR DETECTION SYSTEM

Aim:
To develop a **Color Detection System using Deep Learning (CNN)** that can classify an image based on its dominant color (like Red, Green, Blue, Yellow, etc.).
Code:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split

# Define color labels and their RGB values
color_labels = {
    "red": [255, 0, 0],
    "green": [0, 255, 0],
    "blue": [0, 0, 255],
    "yellow": [255, 255, 0],
    "cyan": [0, 255, 255],
    "magenta": [255, 0, 255],
    "orange": [255, 165, 0],
    "purple": [128, 0, 128]
}

# Generate 200 images per color
images = []
labels = []

for i, (color, rgb) in enumerate(color_labels.items()):
    for _ in range(200):
        img = np.ones((32, 32, 3), dtype=np.uint8) * np.array(rgb, dtype=np.uint8)
        img = img + np.random.randint(-10, 10, img.shape, dtype=np.int16) # Add noise
        img = np.clip(img, 0, 255)
        images.append(img)
        labels.append(i)

images = np.array(images) / 255.0
labels = np.array(labels)

# Split dataset
```

```
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

print("Training samples:", X_train.shape[0])
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(64, (3,3),
    activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(color_labels), activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2)
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"✅ Test Accuracy: {test_acc*100:.2f}%")
idx = np.random.randint(0, len(X_test))
sample = X_test[idx].reshape(1, 32, 32, 3)
pred = np.argmax(model.predict(sample))
predicted_color = list(color_labels.keys())[pred]

plt.imshow(X_test[idx])
plt.title(f"Predicted: {predicted_color}")
plt.axis('off')
plt.show()

#TO UPLOAD OWN IMAGE

from google.colab import files
from PIL import Image

uploaded = files.upload()
for filename in uploaded.keys():
```

```
img = Image.open(filename).resize((32, 32))
```

```
img_array = np.array(img) / 255.0
pred = np.argmax(model.predict(img_array.reshape(1, 32, 32, 3)))
predicted_color = list(color_labels.keys())[pred]
plt.imshow(img)
plt.title(f"Predicted: {predicted_color}")
plt.axis('off')
plt.show()
```

Output:

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
```

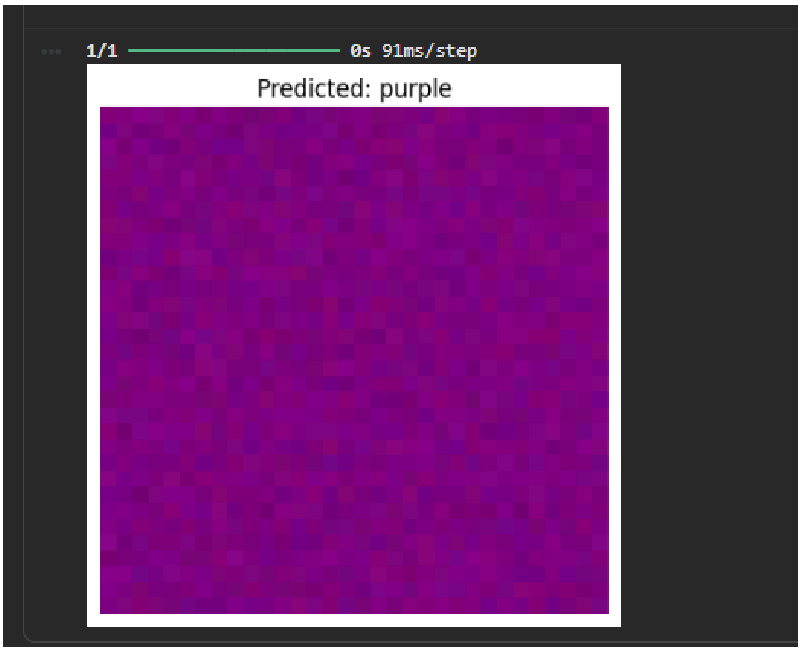
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense_3 (Dense)	(None, 128)	295,040
dense_4 (Dense)	(None, 8)	1,032

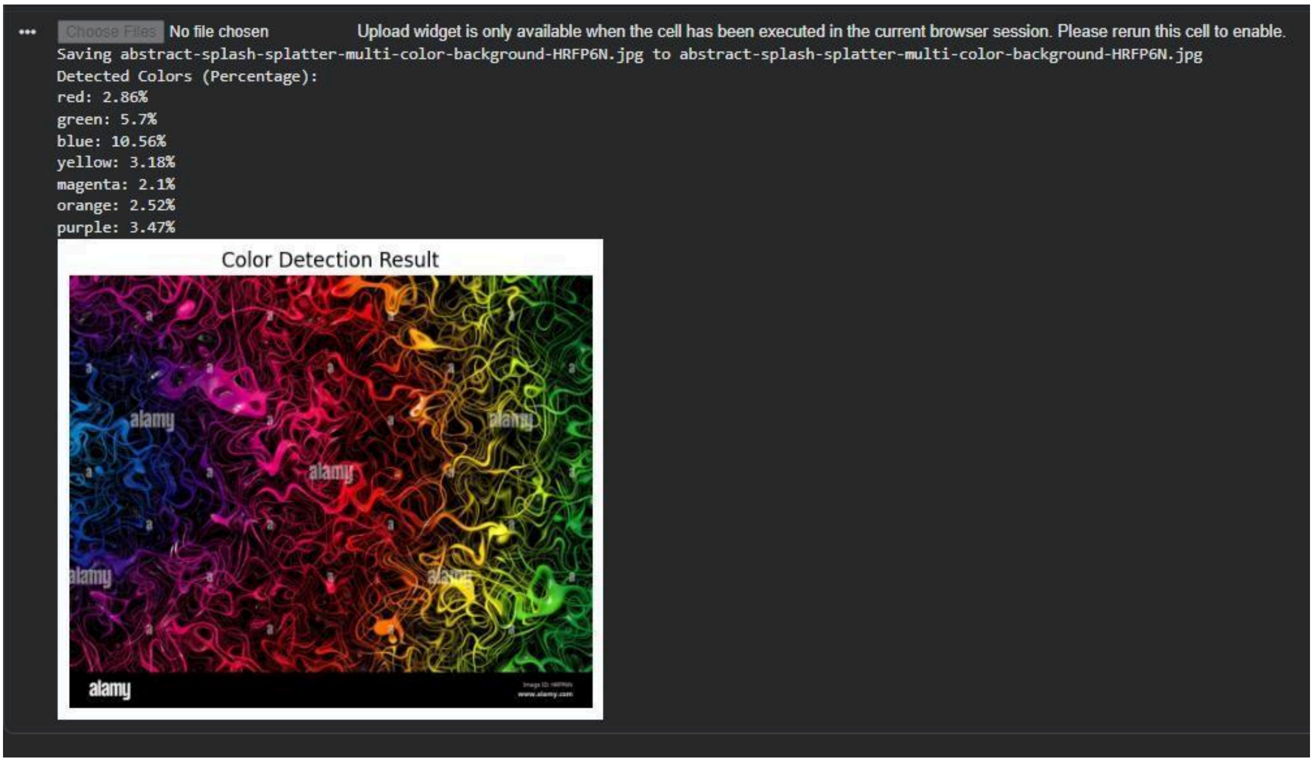
```
 Total params: 315,464 (1.20 MB)
 Trainable params: 315,464 (1.20 MB)
 Non-trainable params: 0 (0.00 B)
```

```
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 1.0000 - loss: 3.1360e-04 - val_accuracy: 1.0000 - val_loss: 2.7132e-04
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 1.0000 - loss: 2.3697e-04 - val_accuracy: 1.0000 - val_loss: 2.2418e-04
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 1.0000 - loss: 1.8263e-04 - val_accuracy: 1.0000 - val_loss: 1.8816e-04
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 1.0000 - loss: 1.5586e-04 - val_accuracy: 1.0000 - val_loss: 1.6224e-04
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 1.0000 - loss: 1.4002e-04 - val_accuracy: 1.0000 - val_loss: 1.4151e-04
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 1.0000 - loss: 1.2547e-04 - val_accuracy: 1.0000 - val_loss: 1.2546e-04
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 1.0000 - loss: 1.0262e-04 - val_accuracy: 1.0000 - val_loss: 1.1006e-04
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 1.0000 - loss: 9.2942e-05 - val_accuracy: 1.0000 - val_loss: 9.8230e-05
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 1.0000 - loss: 9.7027e-05 - val_accuracy: 1.0000 - val_loss: 8.8017e-05
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 1.0000 - loss: 7.4408e-05 - val_accuracy: 1.0000 - val_loss: 7.9657e-05
```

```
test_loss, test_acc = model.evaluate(X_test, y_test)
```





Result:

Thus this project can detects and quantifies multiple colors in any image