RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

CS19P18 - DEEP LEARNING CONCEPTS LABORATORY

RECORD

NAME: MYTHREIY ANAND

YEAR/SEMESTER: FOUR/SEVEN

BRANCH: COMPUTER SCIENCE AND ENGINNEERING

REGISTER NO: 220701177

COLLEGE ROLL NO: 2116220701177

ACADEMIC YEAR: 2025 -2026

RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)

RAJALAKSHMI NAGAR, THANDALAM- 602 105

BONAFIDE CERTIFICATE

NAME: MYTHREIY ANAND **BRANCH/SECTION:** COMPUTER SCIENCE AND ENGINEERING **ACADEMIC YEAR:** 2025 -2026 **SEMESTER:** SEVEN

REGISTER NO: | **220701177**

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 by typing 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:

```
Epoch 1/10
192/192 [==============================] - 5s 17ms/step - loss: 0.3739 - accuracy: 0.8950 - val_loss: 0.1801 - val_accuracy: 0.
9480
Epoch 2/10
192/192 [==============================] - 3s 14ms/step - loss: 0.1492 - accuracy: 0.9562 - val_loss: 0.1261 - val_accuracy: 0.
9635
Epoch 3/10
192/192 [==============================] - 2s 13ms/step - loss: 0.0980 - accuracy: 0.9714 - val_loss: 0.1129 - val_accuracy: 0.
9676
Epoch 4/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0711 - accuracy: 0.9795 - val_loss: 0.0962 - val_accuracy: 0.
9709
Epoch 5/10
192/192 [==============================] - 2s 10ms/step - loss: 0.0543 - accuracy: 0.9844 - val_loss: 0.0914 - val_accuracy: 0.
9725
Epoch 6/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0402 - accuracy: 0.9888 - val_loss: 0.0866 - val_accuracy: 0.
9737
Epoch 7/10
192/192 [==============================] - 2s 12ms/step - loss: 0.0301 - accuracy: 0.9920 - val_loss: 0.0871 - val_accuracy: 0.
9750
Epoch 8/10
192/192 [==============================] - 2s 12ms/step - loss: 0.0245 - accuracy: 0.9931 - val_loss: 0.0840 - val_accuracy: 0.
9762
Epoch 9/10
192/192 [==============================] - 2s 12ms/step - loss: 0.0180 - accuracy: 0.9956 - val_loss: 0.0878 - val_accuracy: 0.
9760
Epoch 10/10
192/192 [==============================] - 2s 11ms/step - loss: 0.0149 - accuracy: 0.9963 - val_loss: 0.0858 - val_accuracy: 0.
9755
```

File   Edit   View   Insert   Cell   Kernel   Widgets   Help   Trusted | Python 3 (ipykernel) O

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import tensorflow as tf
        from tensorflow.keras.datasets import mnist
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense
        from tensorflow.keras.utils import to_categorical
```

```
In [2]: feature_vector_length =784
        num_classes=10
```

```
In [3]: (X_train, Y_train), (X_test, Y_test) = mnist.load_data()

        Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
        11490434/11490434 [==============================] - 2s 0us/step
```

```
In [4]: input_shape = (feature_vector_length)
        print(f'Feature shape: {input_shape}')

        Feature shape: 784
```

```
In [5]: X_train = X_train.reshape(X_train.shape[0], feature_vector_length)
        X_test = X_test.reshape(X_test.shape[0], feature_vector_length)
```

```
In [6]: X_train = X_train.astype('float32') / 255
        X_test = X_test.astype('float32') / 255
        Y_train = to_categorical(Y_train, num_classes)
        Y_test = to_categorical(Y_test, num_classes)
```

Sample 1 - Predicted: 7,Actual: 7   Sample 2 - Predicted: 2,Actual: 2   Sample 3 - Predicted: 1,Actual: 1

Sample 4 - Predicted: 0,Actual: 0   Sample 5 - Predicted: 4,Actual: 4
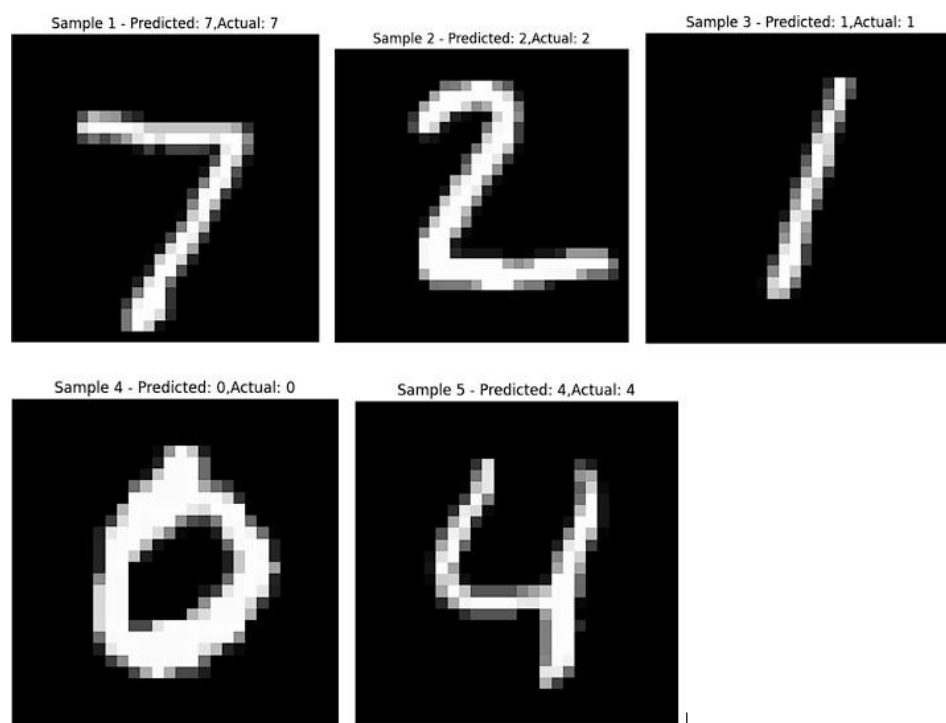
Result:

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')
])
```

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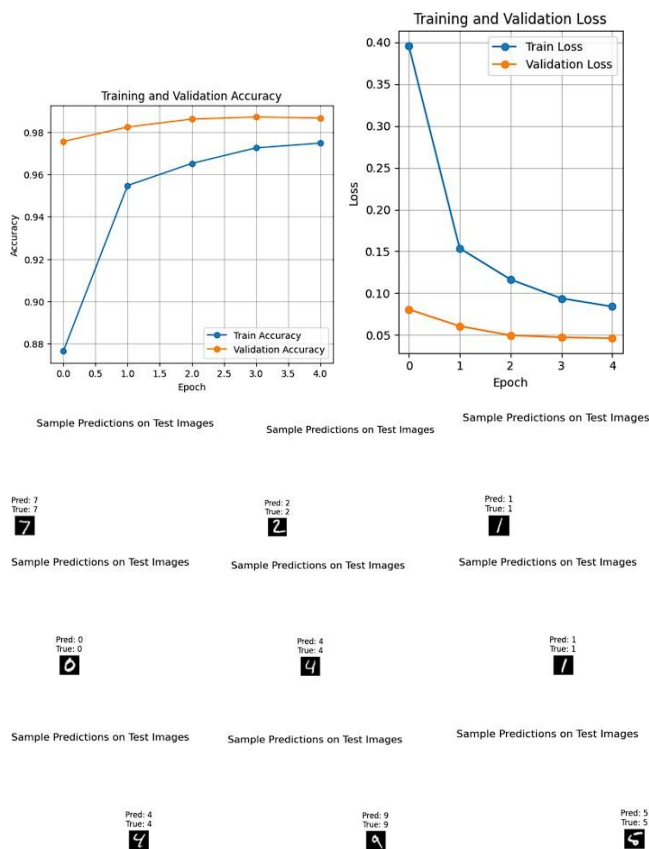
```
plt.axis('off')
plt.suptitle("Sample Predictions on Test Images", fontsize=16)
plt.show()
```

Output:

```
Epoch 1/5
750/750 [==============================] - 30s 39ms/step - loss: 0.3961 - accuracy: 0.8765 - val_loss: 0.0806 - val_accuracy:
0.9756
Epoch 2/5
750/750 [==============================] - 26s 35ms/step - loss: 0.1538 - accuracy: 0.9548 - val_loss: 0.0606 - val_accuracy:
0.9824
Epoch 3/5
750/750 [==============================] - 30s 39ms/step - loss: 0.1163 - accuracy: 0.9652 - val_loss: 0.0495 - val_accuracy:
0.9862
Epoch 4/5
750/750 [==============================] - 27s 36ms/step - loss: 0.0937 - accuracy: 0.9725 - val_loss: 0.0472 - val_accuracy:
0.9872
Epoch 5/5
750/750 [==============================] - 26s 35ms/step - loss: 0.0840 - accuracy: 0.9748 - val_loss: 0.0460 - val_accuracy:
0.9867
313/313 [==============================] - 2s 5ms/step - loss: 0.0390 - accuracy: 0.9880

 Test accuracy: 0.9880
 Test loss: 0.0390
```


Training and Validation Accuracy

Training and Validation Loss

Sample Predictions on Test Images

Sample Predictions on Test Images

Sample Predictions on Test Images

Pred: 7
True: 7

Pred: 2
True: 2

Pred: 1
True: 1

Sample Predictions on Test Images

Sample Predictions on Test Images

Sample Predictions on Test Images

Pred: 0
True: 0

Pred: 4
True: 4

Pred: 1
True: 1

Sample Predictions on Test Images

Sample Predictions on Test Images

Sample Predictions on Test Images

Pred: 4
True: 4

Pred: 9
True: 9

Pred: 5
True: 5

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:

```
Epoch 1/10
625/625 [==============================] - 58s 87ms/step - loss: 1.6801 - accuracy: 0.3846 - val_loss: 1.4341 - val_accuracy:
0.4803
Epoch 2/10
625/625 [==============================] - 37s 60ms/step - loss: 1.3153 - accuracy: 0.5284 - val_loss: 1.3005 - val_accuracy:
0.5388
Epoch 3/10
625/625 [==============================] - 36s 58ms/step - loss: 1.1663 - accuracy: 0.5846 - val_loss: 1.1370 - val_accuracy:
0.6014
Epoch 4/10
625/625 [==============================] - 38s 61ms/step - loss: 1.0629 - accuracy: 0.6249 - val_loss: 1.0984 - val_accuracy:
0.6178
Epoch 5/10
625/625 [==============================] - 41s 65ms/step - loss: 0.9991 - accuracy: 0.6480 - val_loss: 1.0476 - val_accuracy:
0.6379
Epoch 6/10
625/625 [==============================] - 38s 61ms/step - loss: 0.9348 - accuracy: 0.6720 - val_loss: 0.9795 - val_accuracy:
0.6598
Epoch 7/10
625/625 [==============================] - 38s 60ms/step - loss: 0.8764 - accuracy: 0.6970 - val_loss: 1.0013 - val_accuracy:
0.6547
Epoch 8/10
625/625 [==============================] - 38s 61ms/step - loss: 0.8338 - accuracy: 0.7096 - val_loss: 0.9313 - val_accuracy:
0.6770
Epoch 9/10
625/625 [==============================] - 39s 62ms/step - loss: 0.7943 - accuracy: 0.7242 - val_loss: 0.9243 - val_accuracy:
0.6856
Epoch 10/10
625/625 [==============================] - 37s 60ms/step - loss: 0.7588 - accuracy: 0.7362 - val_loss: 0.8994 - val_accuracy:
0.6986
```


Predicted: frog
Actual: frog
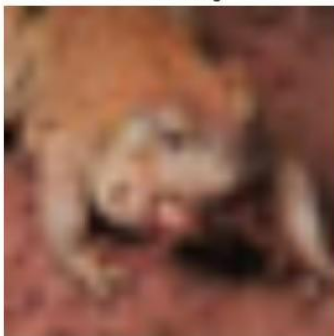
Result

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,
```

```python
                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 [==============================] - 231s 182ms/step - loss: 1.7748 - accuracy: 0.3727 - val_loss: 1.4674 - val_accurac
y: 0.4959
Epoch 2/10
1250/1250 [==============================] - 193s 154ms/step - loss: 1.4665 - accuracy: 0.4920 - val_loss: 1.3556 - val_accurac
y: 0.5322
Epoch 3/10
1250/1250 [==============================] - 187s 150ms/step - loss: 1.3733 - accuracy: 0.5264 - val_loss: 1.2966 - val_accurac
y: 0.5512
Epoch 4/10
1250/1250 [==============================] - 189s 151ms/step - loss: 1.3197 - accuracy: 0.5386 - val_loss: 1.2610 - val_accurac
y: 0.5621
Epoch 5/10
1250/1250 [==============================] - 191s 153ms/step - loss: 1.2777 - accuracy: 0.5551 - val_loss: 1.2352 - val_accurac
y: 0.5739
Epoch 6/10
1250/1250 [==============================] - 190s 152ms/step - loss: 1.2474 - accuracy: 0.5683 - val_loss: 1.2154 - val_accurac
y: 0.5759
Epoch 7/10
1250/1250 [==============================] - 187s 150ms/step - loss: 1.2269 - accuracy: 0.5741 - val_loss: 1.1981 - val_accurac
y: 0.5830
Epoch 8/10
1250/1250 [==============================] - 183s 146ms/step - loss: 1.2039 - accuracy: 0.5834 - val_loss: 1.1839 - val_accurac
y: 0.5864
Epoch 9/10
1250/1250 [==============================] - 177s 142ms/step - loss: 1.1866 - accuracy: 0.5887 - val_loss: 1.1735 - val_accurac
y: 0.5900
Epoch 10/10
1250/1250 [==============================] - 175s 140ms/step - loss: 1.1699 - accuracy: 0.5943 - val_loss: 1.1672 - val_accurac
y: 0.5910
```


Predicted: frog
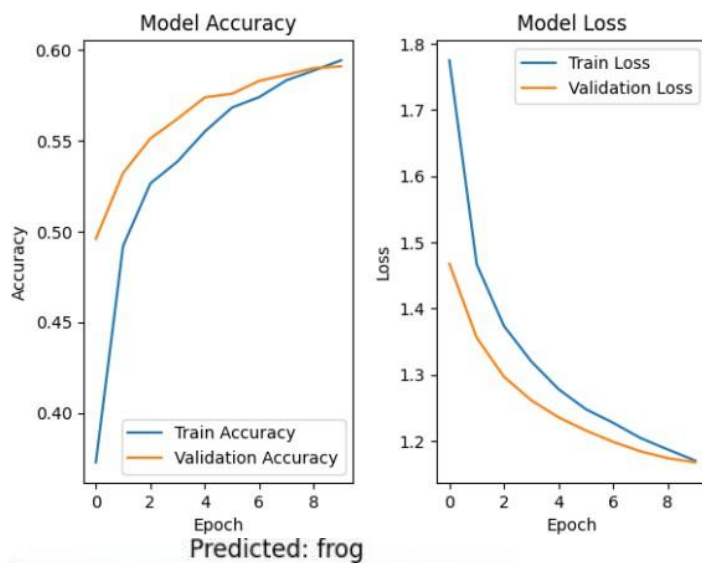
Result

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
DATE:25/08/2025   KERAS/TENSORFLOW

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:
```python
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()}
```

```python
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:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ───────────── 0s 0us/step
Training...
Epoch 1/2
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
313/313 ───────────── 21s 59ms/step - accuracy: 0.6479 - loss: 0.6143 - val_accuracy: 0.6644 - val_loss: 0.6085
Epoch 2/2
313/313 ───────────── 17s 53ms/step - accuracy: 0.7939 - loss: 0.4496 - val_accuracy: 0.8186 - val_loss: 0.4121
782/782 ───────────── 10s 13ms/step - accuracy: 0.8237 - loss: 0.4115

Test Accuracy: 0.8230
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ───────────── 0s 0us/step
```

```python
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 ")
```

```
1/1 ───────────── 0s 195ms/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 michael
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(False)
ax.get_yaxis().set_visible(False)
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
```

⮧ Test Loss: 0.09166844934225082
   Test Accuracy: 0.9712756377551021
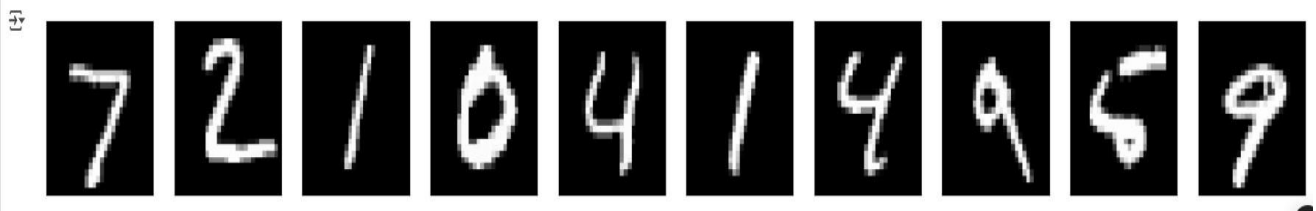

```python
# Display reconstruction with threshold
ax = plt.subplot(2, n, i + 1 + n)
reconstruction = decoded_imgs[i].reshape(28, 28)
plt.imshow(np.where(reconstruction >= threshold, 1.0, 0.0))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
```


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

# Define the paths to the YOLOv3 configuration, weights, and class names files
cfg_file = '/content/yolov3.cfg'
weight_file = '/content/yolov3.weights'
namesfile = '/content/coco.names'

# Load the YOLOv3 model
net = cv2.dnn.readNet(weight_file, cfg_file)

# Load class names
with open(namesfile, 'r') as f:
classes = f.read().strip().split('\n')

# Load an image for object detection
image_path = '/content/hit.jpg'
image = cv2.imread(image_path)

# Get the height and width of the image
height, width = image.shape[:2]

# Create a blob from the image
blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)
net.setInput(blob)

# Get the names of the output layers
layer_names = net.getUnconnectedOutLayersNames()

# Run forward pass
```

```python
outs = net.forward(layer_names)

# Initialize lists to store detected objects' information
class_ids = []
confidences = []
boxes = []

# Define a confidence threshold for object detection
conf_threshold = 0.5

# Loop over the detections
for out in outs:
for detection in out:
scores = detection[5:]
class_id = np.argmax(scores)
confidence = scores[class_id]
if confidence > conf_threshold:
# Object detected
center_x = int(detection[0] * width)
center_y = int(detection[1] * height)
w = int(detection[2] * width)
h = int(detection[3] * height)

# Rectangle coordinates
x = int(center_x - w / 2)
y = int(center_y - h / 2)

class_ids.append(class_id)
confidences.append(float(confidence))
boxes.append([x, y, w, h])

# Apply non-maximum suppression to eliminate overlapping boxes
nms_threshold = 0.4
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

# Draw bounding boxes and labels on the image
for i in indices.flatten(): # flatten for compatibility
x, y, w, h = boxes[i]
label = str(classes[class_ids[i]])
confidence = confidences[i]

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2.putText(image, f'{label} {confidence:.2f}', (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
```

```
# Display the result in Jupyter Notebook
plt.figure(figsize=(10, 8))
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Output:

```
cfg_file=r"C:\Users\agaki\Downloads\yolov3.cfg"
weight_file=r"C:\Users\agaki\Downloads\yolov3.weights"
names_file=r"C:\Users\agaki\Downloads\coco.names"
```

```
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)

            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x,y,w,h])
```
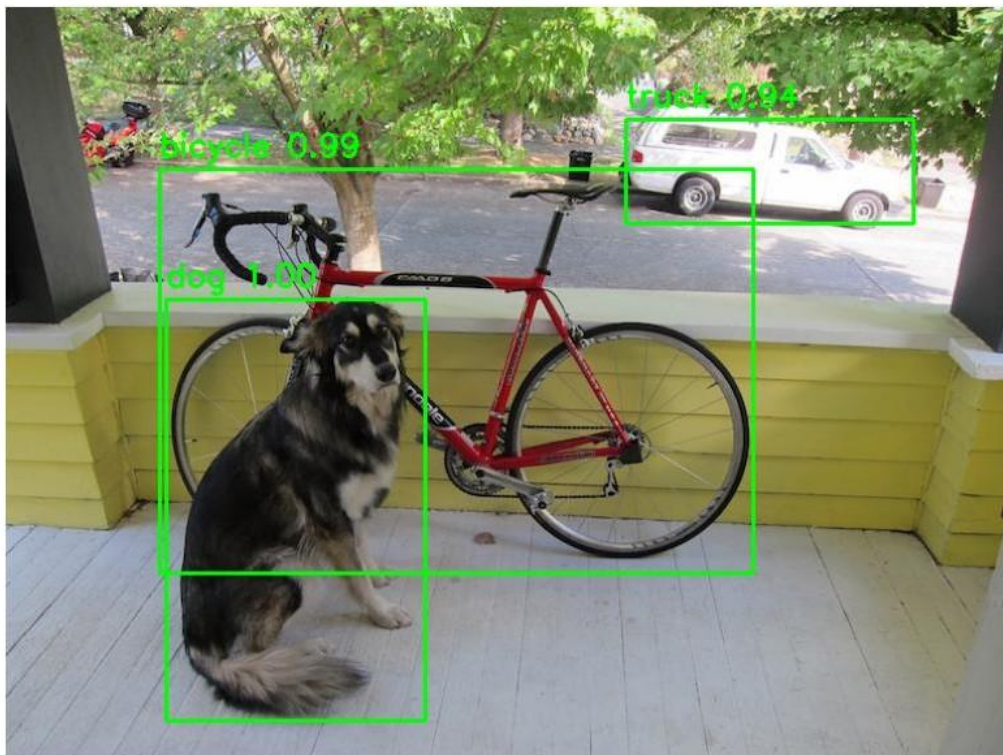

Result

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)
```

```python
    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)
```

```python
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
```

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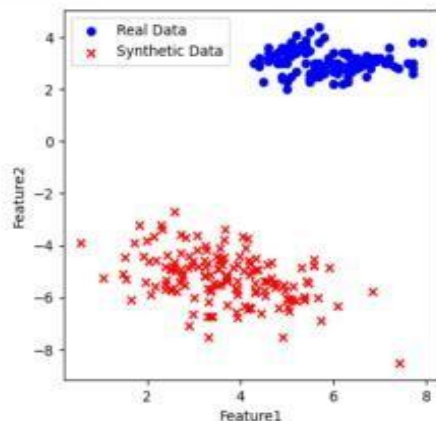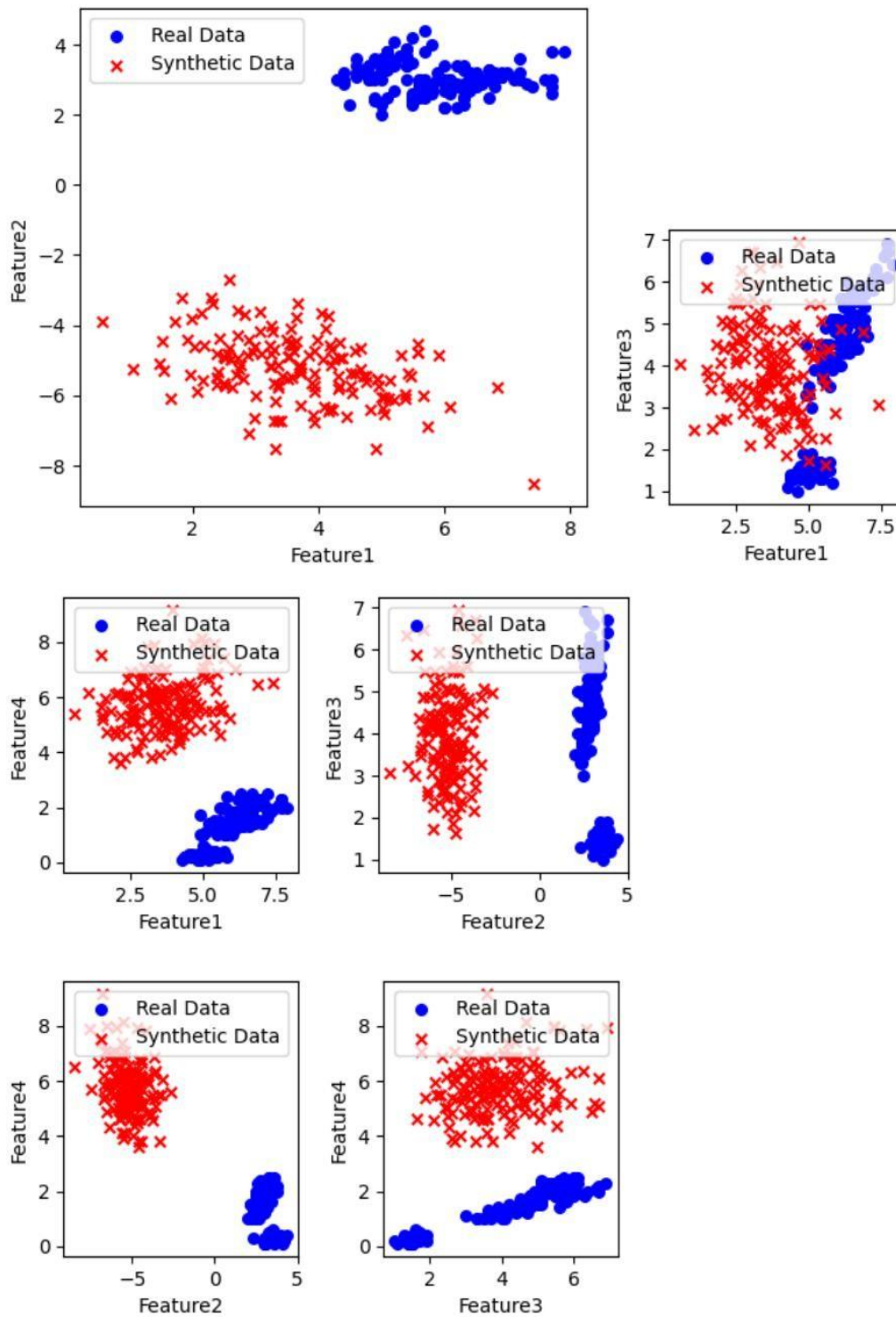
Thus, a generative adversarial neural network using Keras / Tensorflow has been implemented successfully.

MINI PROJECT

Melanoma detection using CNN

Aim:

To develop a deep learning model for accurate melanoma detection and classification from dermoscopic images using a Convolutional Neural Network (CNN) architecture.

Code:

```python
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import PIL
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import Adam # - Works
import random
from glob import glob
import seaborn as sns
from tensorflow.keras.losses import SparseCategoricalCrossentropy
import matplotlib.pyplot as plt
import matplotlib.image as img
import warnings
warnings.filterwarnings('ignore')
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
```

```python
gpus = tf.config.experimental.list_physical_devices('GPU')
print(gpus)
try:
    tf.config.experimental.set_memory_growth = True
except Exception as ex:
    print(e)
```

```
[]
```

```python
data_dir_train = "/content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/Train"
data_dir_test = "/content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/Test"
```

```python
import pathlib
import tensorflow as tf
# ... other imports ...

data_dir_train = pathlib.Path("/content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/Train")
data_dir_test = pathlib.Path("/content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/Test")

image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
print(image_count_train)
image_count_test = len(list(data_dir_test.glob('*/*.jpg')))
```

```python
import pathlib
import tensorflow as tf
# ... other imports ...
import random

data_dir_train = pathlib.Path("/content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/Train")
data_dir_test = pathlib.Path("/content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/Test")

image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
print(image_count_train)
image_count_test = len(list(data_dir_test.glob('*/*.jpg')))
print(image_count_test)

batch_size = 32
img_height = 180
img_width = 180
rnd_seed = 123
random.seed(rnd_seed)
```

```
2250
118
```

```python
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir_train,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 2250 files belonging to 9 classes.
Using 1800 files for training.
```

```python
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir_train,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```
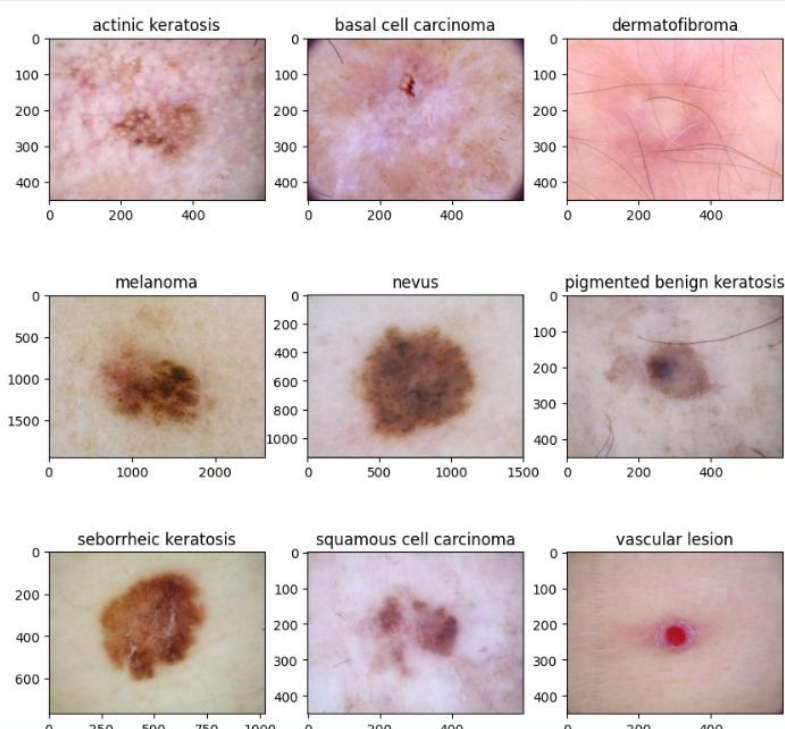
```
Found 2250 files belonging to 9 classes.
Using 450 files for validation.
```

```python
num_classes = len(class_names)
plt.figure(figsize=(10,10))
for i in range(num_classes):
  plt.subplot(3,3,i+1)
  image = img.imread(str(list(data_dir_train.glob(class_names[i]+'/*.jpg'))[1]))
  plt.title(class_names[i])
  plt.imshow(image)
```

```python
for image_batch, labels_batch in train_ds.take(1):
    print(image_batch.shape)
    print(labels_batch.shape)
```

```
(32, 180, 180, 3)
(32,)
```

```python
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense
# Import Rescaling from tensorflow.keras.layers
from tensorflow.keras.layers import Rescaling

num_classes = 9
model = Sequential([Rescaling(1./255, input_shape=(img_height, img_width, 3))])


model.add(Conv2D(32, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(64, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(128, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(256, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(512, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Flatten())
model.add(Dense(1024,activation="relu"))
model.add(Dense(units=num_classes, activation= 'softmax'))
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

```python
from tensorflow.keras.optimizers import Adam # Import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy # Import SparseCategoricalCrossentropy here

# Use learning_rate instead of lr
opt = Adam(learning_rate=0.001)
model.compile(optimizer= opt,
              loss= SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```python
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 32)	896
max_pooling2d (MaxPooling2D)	(None, 90, 90, 32)	0
conv2d_1 (Conv2D)	(None, 90, 90, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 64)	0
conv2d_2 (Conv2D)	(None, 45, 45, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 128)	0
conv2d_3 (Conv2D)	(None, 22, 22, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 11, 11, 256)	0
conv2d_4 (Conv2D)	(None, 11, 11, 512)	1,180,160
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 512)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 1024)	13,108,224
dense_1 (Dense)	(None, 9)	9,225

Total params: 14,686,025 (56.02 MB)
Trainable params: 14,686,025 (56.02 MB)
Non-trainable params: 0 (0.00 B)

```python
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
```

```python
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
model.save('/content/drive/MyDrive/model/model.h5')
```

```
Epoch 1/20
57/57 ──────────── 227s 4s/step - accuracy: 0.7826 - loss: 0.6354 - val_accuracy: 0.5489 - val_loss: 1.9771
Epoch 2/20
57/57 ──────────── 226s 4s/step - accuracy: 0.8241 - loss: 0.4456 - val_accuracy: 0.5867 - val_loss: 1.8838
Epoch 3/20
57/57 ──────────── 260s 4s/step - accuracy: 0.8852 - loss: 0.3100 - val_accuracy: 0.5378 - val_loss: 2.3470
Epoch 4/20
57/57 ──────────── 226s 4s/step - accuracy: 0.8895 - loss: 0.3206 - val_accuracy: 0.5311 - val_loss: 2.0807
Epoch 5/20
57/57 ──────────── 233s 4s/step - accuracy: 0.8829 - loss: 0.3146 - val_accuracy: 0.5578 - val_loss: 2.5108
Epoch 6/20
57/57 ──────────── 264s 4s/step - accuracy: 0.8949 - loss: 0.2499 - val_accuracy: 0.5333 - val_loss: 2.5989
Epoch 7/20
57/57 ──────────── 239s 4s/step - accuracy: 0.9129 - loss: 0.1884 - val_accuracy: 0.5133 - val_loss: 2.5791
Epoch 8/20
57/57 ──────────── 247s 4s/step - accuracy: 0.8685 - loss: 0.3108 - val_accuracy: 0.4756 - val_loss: 2.6437
Epoch 9/20
57/57 ──────────── 241s 4s/step - accuracy: 0.8975 - loss: 0.2596 - val_accuracy: 0.5444 - val_loss: 2.8430
Epoch 10/20
57/57 ──────────── 263s 4s/step - accuracy: 0.9203 - loss: 0.2522 - val_accuracy: 0.5156 - val_loss: 2.9747
Epoch 11/20
57/57 ──────────── 261s 4s/step - accuracy: 0.8893 - loss: 0.2676 - val_accuracy: 0.5311 - val_loss: 2.6957
Epoch 12/20
57/57 ──────────── 263s 4s/step - accuracy: 0.9438 - loss: 0.1267 - val_accuracy: 0.5222 - val_loss: 2.8316
Epoch 13/20
57/57 ──────────── 261s 4s/step - accuracy: 0.9222 - loss: 0.1627 - val_accuracy: 0.5533 - val_loss: 3.1552
Epoch 14/20
57/57 ──────────── 221s 4s/step - accuracy: 0.9449 - loss: 0.1135 - val_accuracy: 0.5578 - val_loss: 3.1667
Epoch 15/20
57/57 ──────────── 233s 4s/step - accuracy: 0.9431 - loss: 0.1112 - val_accuracy: 0.5356 - val_loss: 2.8234
Epoch 16/20
57/57 ──────────── 257s 4s/step - accuracy: 0.9509 - loss: 0.0988 - val_accuracy: 0.5489 - val_loss: 2.9631
Epoch 17/20
57/57 ──────────── 270s 4s/step - accuracy: 0.9399 - loss: 0.1119 - val_accuracy: 0.5444 - val_loss: 2.9215
Epoch 18/20
57/57 ──────────── 254s 4s/step - accuracy: 0.9467 - loss: 0.1068 - val_accuracy: 0.5333 - val_loss: 3.1200
Epoch 19/20
57/57 ──────────── 235s 4s/step - accuracy: 0.9424 - loss: 0.1034 - val_accuracy: 0.5378 - val_loss: 2.9997
Epoch 20/20
57/57 ──────────── 234s 4s/step - accuracy: 0.9420 - loss: 0.1015 - val_accuracy: 0.5422 - val_loss: 2.8432
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format,
```

```
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```
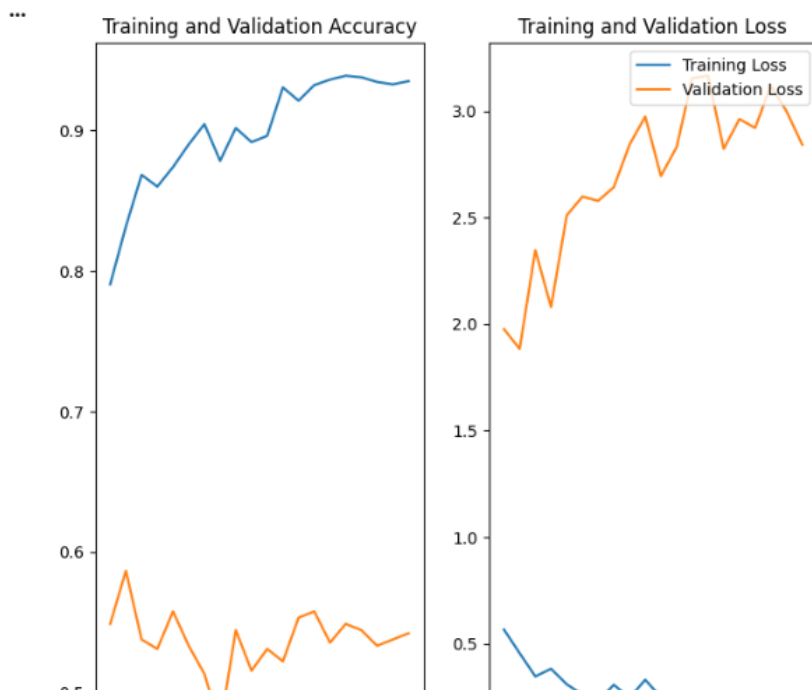
```python
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```
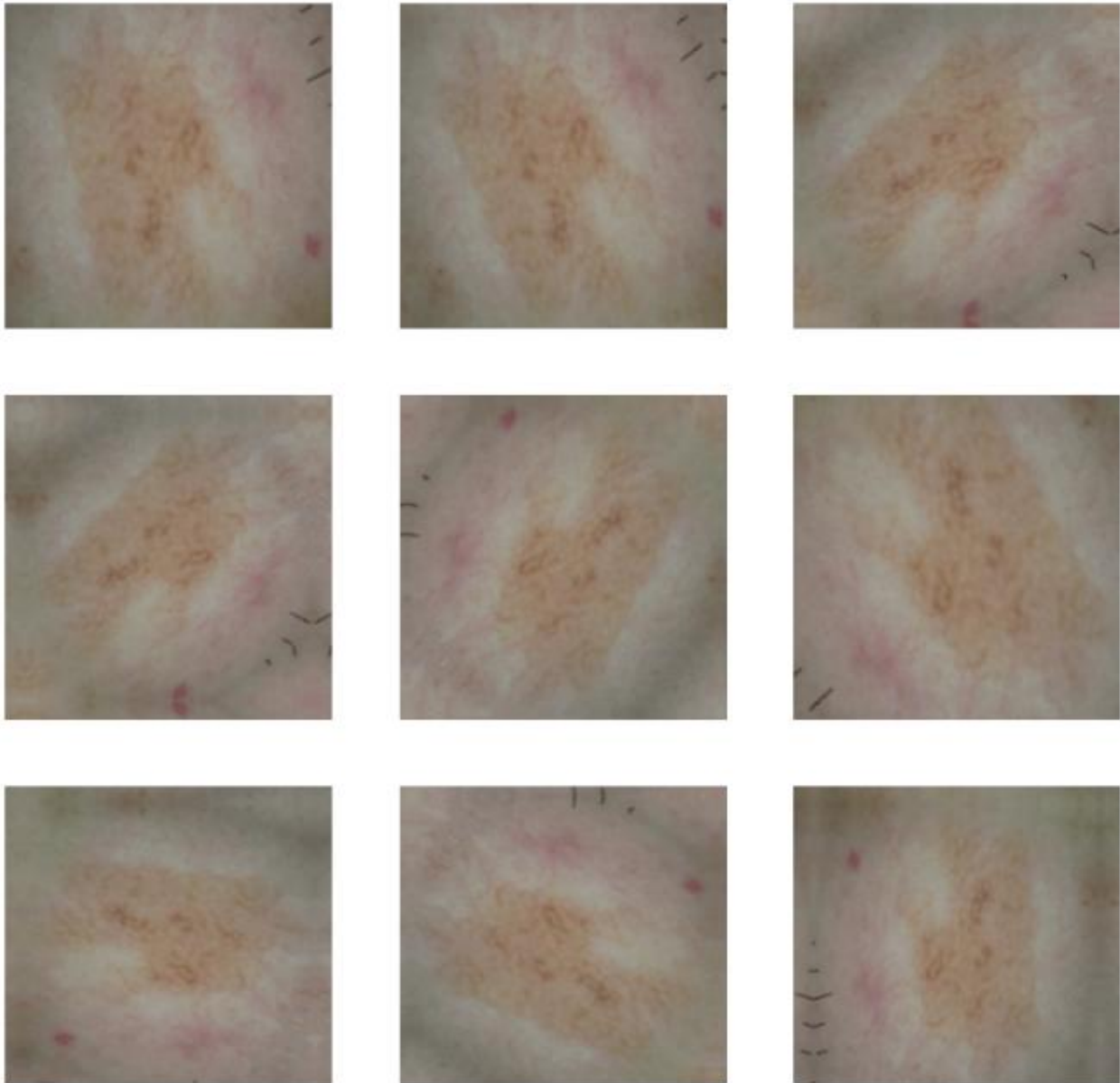
```python
import Augmentor
import os
import tensorflow as tf
import pathlib # Import pathlib here

# Corrected path to your training dataset in Google Colab
path_to_training_dataset = "/content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/Train/"

# Create the output directory if it doesn't exist
output_dir = "//content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/augment"
os.makedirs(output_dir, exist_ok=True)


data_dir_train = pathlib.Path(path_to_training_dataset)
class_names = [item.name for item in data_dir_train.iterdir() if item.is_dir()]

for i in class_names:
    # Construct the correct path to the class directory
    class_dir = os.path.join(path_to_training_dataset, i)

    # Construct the output directory for this class
    output_class_dir = os.path.join(output_dir, i, "output")
    os.makedirs(output_class_dir, exist_ok=True)  # Create output directory for the class

    # Create the Augmentor Pipeline
    p = Augmentor.Pipeline(class_dir, output_directory=output_class_dir)
    p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
    p.sample(1000)
```

```
Collecting Augmentor
  Downloading Augmentor-0.2.12-py2.py3-none-any.whl.metadata (1.3 kB)
Requirement already satisfied: Pillow>=5.2.0 in /usr/local/lib/python3.11/dist-packages (from Augmentor) (11.2.1)
Requirement already satisfied: tqdm>=4.9.0 in /usr/local/lib/python3.11/dist-packages (from Augmentor) (4.67.1)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from Augmentor) (2.0.2)
Downloading Augmentor-0.2.12-py2.py3-none-any.whl (38 kB)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.12
Initialised with 139 image(s) found.
Output directory set to //content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/augment/vascular lesion/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7CE99A0A9CD0>: 100%|██████████| 1000/1000 [00:30<00:00, 32.
Initialised with 114 image(s) found.
Output directory set to //content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/augment/actinic keratosis/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7CE99AC628D0>: 100%|██████████| 1000/1000 [00:32<00:00, 3
Initialised with 357 image(s) found.
Output directory set to //content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/augment/nevus/output.Processing <PIL.Image.Image image mode=RGB size=3072x2304 at 0x7CE9949468D0>: 100%|██████████| 1000/1000 [01:46<00:00,  9.38 Sampl
Initialised with 377 image(s) found.
Output directory set to //content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/augment/basal cell carcinoma/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7CE9A42D0550>: 100%|██████████| 1000/1000 [00:28<00:00
Initialised with 448 image(s) found.
Output directory set to //content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/augment/melanoma/output.Processing <PIL.Image.Image image mode=RGB size=1024x768 at 0x7CE9A5CEA590>: 100%|██████████| 1000/1000 [02:01<00:00,  8.22 Sam
Initialised with 95 image(s) found.
Output directory set to //content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/augment/dermatofibroma/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7CE9988F7D10>: 100%|██████████| 1000/1000 [00:28<00:00, 35.6
Initialised with 181 image(s) found.
Output directory set to //content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/augment/squamous cell carcinoma/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7CE9988DB710>: 100%|██████████| 1000/1000 [00:35<00
```

```python
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd # Import pandas
import os
import PIL
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import Adam # - Works
import random
from glob import glob
import seaborn as sns
from tensorflow.keras.losses import SparseCategoricalCrossentropy
import matplotlib.pyplot as plt
import matplotlib.image as img
import warnings
warnings.filterwarnings('ignore')
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

# ... (Rest of the code)
num_classes = len(class_names)
total = 0
all_count = []
class_name = []

# First loop to calculate total
for i in range(num_classes):
  count = len(list(output_dir.glob(class_names[i]+'/output/*.jpg')))
  total += count

print("total training image count = {} \n".format(total))
print("-------------------------------------")

# Second loop to print class details and proportions
for i in range(num_classes):
  count = len(list(output_dir.glob(class_names[i]+'/output/*.jpg'))) # Recalculate count for each class
  print("Class name = ",class_names[i])
  print("count      = ",count)
  # Check if total is not zero before dividing
  if total > 0:
      print("proportion = ",count/total)
  else:
      print("proportion = 0 (Total image count is zero)")
  print("-------------------------------------")
  all_count.append(count)
  class_name.append(class_names[i])
```

```python
for i in range(num_classes):
    count = len(list(output_dir.glob(class_names[i]+'/output/*.jpg'))) # Recalculate count for each class
    print("Class name = ",class_names[i])
    print("count       = ",count)
    # Check if total is not zero before dividing
    if total > 0:
        print("proportion = ",count/total)
    else:
        print("proportion = 0 (Total image count is zero)")
    print("------------------------------------")
    all_count.append(count)
    class_name.append(class_names[i])


temp_df = pd.DataFrame(list(zip(all_count, class_name)), columns = ['count', 'class_name'])
sns.barplot(data=temp_df, y="count", x="class_name")
plt.xticks(rotation=90)
plt.show()
```

```
total training image count = 45000

------------------------------------
Class name =  vascular lesion
count       =  5000
proportion =  0.1111111111111111
------------------------------------
Class name =  actinic keratosis
count       =  5000
proportion =  0.1111111111111111
------------------------------------
Class name =  nevus
count       =  5000
proportion =  0.1111111111111111
------------------------------------
Class name =  basal cell carcinoma
count       =  5000
proportion =  0.1111111111111111
------------------------------------
Class name =  melanoma
count       =  5000
proportion =  0.1111111111111111
------------------------------------
Class name =  dermatofibroma
count       =  5000
proportion =  0.1111111111111111
------------------------------------
Class name =  squamous cell carcinoma
count       =  5000
proportion =  0.1111111111111111
------------------------------------
Class name =  pigmented benign keratosis
count       =  5000
proportion =  0.1111111111111111
------------------------------------
Class name =  seborrheic keratosis
count       =  5000
proportion =  0.1111111111111111
------------------------------------
```

```
proportion =  0.111111111111111
-------------------------------------
```
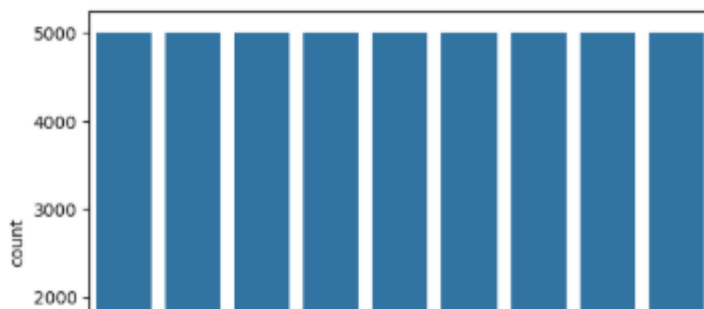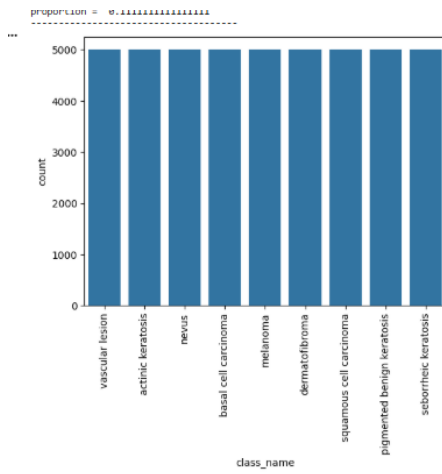


```python
# Define these variables before calling image_dataset_from_directory
batch_size = 32
img_height = 180
img_width = 180
rnd_seed = 123 # Although rnd_seed is not directly used in this specific call, it's good practice to keep related variables together

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    output_dir,
    seed=123,
    validation_split = 0.2,
    subset = 'training',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 45000 files belonging to 9 classes.
Using 36000 files for training.
```

```python
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    output_dir,
    seed=123,
    validation_split = 0.2,
    subset = 'validation',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 45000 files belonging to 9 classes.
Using 9000 files for validation.
```

```python
print(train_ds.class_names)
```

```
['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanoma', 'nevus', 'pigmented benign keratosis', 'seborrheic keratosis', 'squamous cell carcinoma', 'vascular lesion']
```

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential


from tensorflow.keras.layers import Rescaling

from tensorflow.keras.layers import Conv2D, MaxPool2D

num_classes = 9

model = Sequential([Rescaling(1./255, input_shape=(img_height, img_width, 3))])


model.add(Conv2D(32, 3,padding="same",activation='relu'))
model.add(MaxPool2D())
```

Double-click (or enter) to edit

```python
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt,
              loss=SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential


from tensorflow.keras.layers import Rescaling, Flatten

from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense

num_classes = 9

model = Sequential([Rescaling(1./255, input_shape=(img_height, img_width, 3))])


model.add(Conv2D(32, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(64, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(128, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(256, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(512, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Flatten())
model.add(Dense(1024,activation="relu"))
model.add(Dense(units=num_classes, activation= 'softmax'))
```

```python
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt,
              loss=SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```python
    return img_array

def predict_skin_cancer(image_path, model, class_names):
    processed_img = preprocess_image(image_path)
    predictions = model.predict(processed_img)
    predicted_class = np.argmax(predictions[0])
    confidence = np.max(predictions[0])

    plt.imshow(plt.imread(image_path))
    plt.axis('off')
    plt.title(f"Predicted: {class_names[predicted_class]}\nConfidence: {confidence:.2f}")
    plt.show()


num_classes = 9

model = Sequential([Rescaling(1./255, input_shape=(img_height, img_width, 3))])

model.add(Conv2D(32, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(64, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(128, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(256, 3,padding="same",activation='relu'))
model.add(MaxPool2D())

model.add(Conv2D(512, 3,padding="same",activation='relu'))
model.add(MaxPool2D())


model.add(Flatten())

model.add(Dense(1024,activation="relu"))
model.add(Dense(units=num_classes, activation= 'softmax')) # This layer should output probabilities for each class


model_path = '/content/drive/MyDrive/model/model.h5'
model.load_weights(model_path)

model.compile(optimizer=Adam(learning_rate=0.001),
              loss=SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```python
def predict_skin_cancer(image_path, model, class_names):
    import numpy as np
    processed_img = preprocess_image(image_path)
    predictions = model.predict(processed_img)
    predicted_class = np.argmax(predictions[0])
    confidence = np.max(predictions[0])

    plt.imshow(plt.imread(image_path))
    plt.axis('off')
    plt.title(f"Predicted: {class_names[predicted_class]}\nConfidence: {confidence:.2f}")
    plt.show()

    return class_names[predicted_class], confidence


image_path = '/content/drive/MyDrive/archive (1)/Skin cancer ISIC The International Skin Imaging Collaboration/Test/nevus/ISIC_0000000.jpg'
predicted_class, confidence = predict_skin_cancer(image_path, model, class_names)
print(f"Predicted: {predicted_class} with confidence {confidence:.2f}")
```
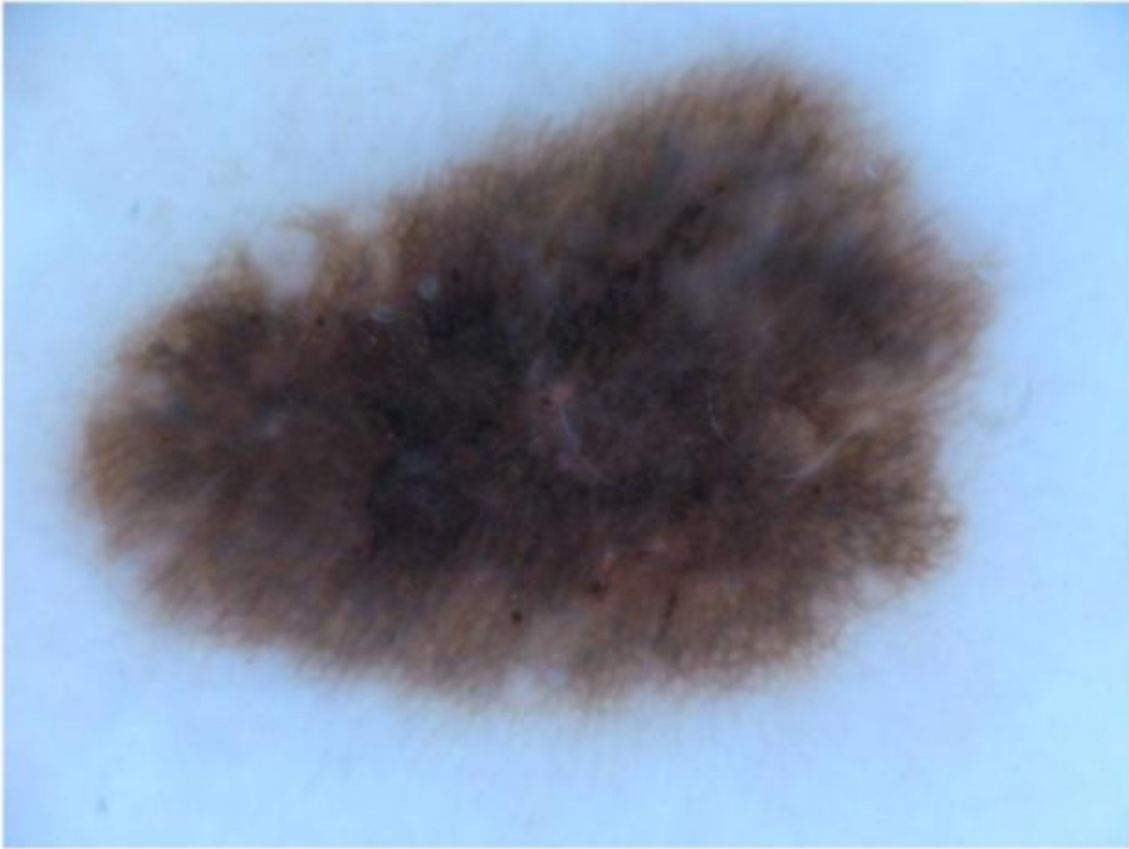
Output:

```
...  1/1 ───────────────  0s 278ms/step
```


Predicted: basal cell carcinoma
Confidence: 0.97

Predicted: basal cell carcinoma with confidence 0.97

Result:

The system successfully classified dermoscopic images into melanoma and non-melanoma categories, demonstrating the capability of the CNN model to accurately detect and differentiate skin cancer types based on visual features.