CONTENT BEYOND SYLLABUS

PROGRAM

```
import numpy as np import
tensorflow as tf from
tensorflow.keras import
layers, models

# Generate some sample data for binary classification
np.random.seed(42)

X_train = np.random.randn(1000, 10)  # 1000 samples
with 10 features y_train = np.random.randint(2,
size=1000)  # Binary labels (0 or 1)

# Define the architecture of the
neural network model =
models.Sequential([
layers.Dense(64,
activation='relu',
input_shape=(10,)),
                              layers.De
nse(32, activation='relu'),

   layers.Dense(1, activation='sigmoid')  # Output layer with sigmoid activation for binary
classification

])

# Compile the model model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])

# Train the model model.fit(X_train, y_train,
epochs=10, batch_size=32, validation_split=0.2)

# Optionally, evaluate the model on test data
# X_test = np.random.randn(200, 10)  # Example test data
# y_test = np.random.randint(2, size=200)  # Example test
labels # test_loss, test_acc = model.evaluate(X_test,
y_test)

# print('Test accuracy:', test_acc)
```

OUTCOME

Epoch 1/10

25/25 [==============================] - 1s 10ms/step - loss: 0.6963 - accuracy:

0.5400 - val_loss: 0.7098 - val_accuracy:
0.4050 Epoch 2/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6839 -
accuracy: 0.5650 - val_loss: 0.7086 - val_accuracy: 0.4350

Epoch 3/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6755 -
accuracy: 0.5813 - val_loss: 0.7070 - val_accuracy: 0.4650

Epoch 4/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6711 -
accuracy: 0.5875 - val_loss: 0.7085 - val_accuracy: 0.5000

Epoch 5/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6670 -
accuracy: 0.5813 - val_loss: 0.7074 - val_accuracy: 0.5100

Epoch 6/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6636 -
accuracy: 0.6237 - val_loss: 0.7114 - val_accuracy: 0.4800

Epoch 7/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6608 -
accuracy: 0.5950 - val_loss: 0.7083 - val_accuracy: 0.4950

Epoch 8/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6549 -
accuracy: 0.6350 - val_loss: 0.7076 - val_accuracy: 0.4900

Epoch 9/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6512 -
accuracy: 0.6488 - val_loss: 0.7097 - val_accuracy: 0.5000

Epoch 10/10

25/25 [==============================] - 0s 3ms/step - loss: 0.6479 -
accuracy: 0.6425 - val_loss: 0.7100 - val_accuracy: 0.4800

<keras.src.callbacks.History at 0x7f888eadcb80>

ADDITIONAL EXPERIMENTS

MINI PROJECT

LAB EXPERIMENTS:

1. Implement simple vector addition in TensorFlow.
2. Implement a regression model in Keras.
3. Implement a perceptron in TensorFlow/Keras Environment.
4. Implement a Feed-Forward Network in TensorFlow/Keras.
5. Implement an Image Classifier using CNN in TensorFlow/Keras.
6. Improve the Deep learning model by fine tuning hyper parameters.
7. Implement a Transfer Learning concept in Image Classification.
8. Using a pre trained model on Keras for Transfer Learning
9. Perform Sentiment Analysis using RNN
10. Implement an LSTM based Autoencoder in TensorFlow/Keras.
11. Image generation using GAN
12. Train a Deep learning model to classify a given image using pre trained model
13. Recommendation system from sales data using Deep Learning
14. Implement Object Detection using CNN
15. Implement any simple Reinforcement Algorithm for an NLP problem

INSTALLATION & PROCEDURE

The execution of the experiments mentioned can be done using a variety of environments, and the choice depends on your preferences and requirements. Here's a breakdown:

1. **Jupyter Notebooks:**
    - Jupyter Notebooks are interactive documents that allow you to write and execute code in a step-by-step manner.
    - They are widely used for data analysis, machine learning, and experimentation.
    - You can install Jupyter using Anaconda or pip.
2. **Anaconda:**
    - Anaconda is a distribution of Python that comes with pre-installed scientific packages and tools.
    - It includes popular libraries like NumPy, pandas, scikit-learn, and more.

- While Anaconda itself is not necessary, it can simplify the setup of your Python environment.

3. **Google Colab:**
   - Google Colab is a cloud-based Jupyter notebook environment provided by Google.
   - It allows you to run code in the cloud and provides access to GPU/TPU for free.
   - Colab is suitable for resource-intensive tasks like deep learning.
4. **Installation of Libraries:**
   - Most of the experiments listed involve TensorFlow and Keras, which can be installed using **pip** in any Python environment.
   - Virtual environments can be used to manage dependencies for each experiment.
5. **Integrated Development Environments (IDEs):**
   - You can use Python IDEs like PyCharm, Visual Studio Code, or any other IDE of your choice for coding and running scripts.
6. **Platform-Agnostic:**
   - The experiments are platform-agnostic and can be executed on Windows, Linux, or macOS.

Recommendations:

- For quick experimentation and resource-intensive tasks (especially for deep learning experiments), Google Colab is recommended due to its access to free GPU/TPU.
- For a local development environment, using Jupyter Notebooks along with Anaconda or a virtual environment is a common choice.

Note: Ensure that you have the required libraries installed (tensorflow, keras, etc.) in your chosen environment before running the experiments. Use pip install for library installations.

Libraries:

- TensorFlow
- Keras (part of TensorFlow).

Installation:

pip install tensorflow

PROGRAM

```
#1
import tensorflow as tf
tf.compat.v1.enable_eager_execution()
vector1 = tf.constant([1.0, 2.0, 3.0])
vector2 = tf.constant([4.0, 5.0, 6.0])
result_vector = tf.add(vector1, vector2)
print("Vector 1:", vector1.numpy())
print("Vector 2:", vector2.numpy())
print("Resultant Vector:", result_vector.numpy())
```

OUTPUT

```
Vector 1: [1. 2. 3.]
Vector 2: [4. 5. 6.]
Resultant Vector: [5. 7. 9.]
```

PROGRAM

```
#2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
np.random.seed(42)
X_train = np.random.rand(100, 1)  \
y_train = 2 * X_train + 1 + 0.1 * np.random.randn(100, 1)
model = Sequential()
model.add(Dense(units=1, input_shape=(1,), activation='linear'))
model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=50, batch_size=8)
X_test = np.array([[0.2], [0.5], [0.8]])
predictions=model.predict(X_test)
print("Predictions:")
print(predictions)
```

OUTPUT

```
Epoch 1/50 13/13 [==============================] - 1s 6ms/step - loss: 2.4243
Epoch 2/50 13/13 [==============================] - 0s 5ms/step - loss: 1.2959
Epoch 3/50 13/13 [==============================] - 0s 4ms/step - loss: 0.7089
Epoch 4/50 13/13 [==============================] - 0s 4ms/step - loss: 0.3944
Epoch 5/50 13/13 [==============================] - 0s 4ms/step - loss: 0.2298
 Epoch 6/50 13/13 [==============================] - 0s 4ms/step - loss: 0.1457
Epoch 7/50 13/13 [==============================] - 0s 4ms/step - loss: 0.0984
Epoch 8/50 13/13 [==============================] - 0s 3ms/step - loss: 0.0745
:
:
Epoch 48/50 13/13 [==============================] - 0s 3ms/step - loss: 0.0174
Epoch 49/50 13/13 [==============================] - 0s 2ms/step - loss: 0.0171
```

Epoch 50/50 13/13 [==============================] - 0s 2ms/step - loss: 0.0167 1/1 [==============================] - 0s 90ms/step

Predictions: [[1.5073806] [2.001153] [2.4949255]]

PROGRAM

```
#3
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
np.random.seed(42)
X = np.random.rand(100, 2)
y = (X[:, 0] + X[:, 1] > 1).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')])
model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=50, batch_size=8, verbose=0)
predictions = model.predict(X_test)
binary_predictions = (predictions > 0.5).astype(int)
accuracy = accuracy_score(y_test, binary_predictions)
print("Accuracy on the test set:", accuracy)
```

OUTPUT

```
1/1 [==============================] - 0s 315ms/step
Accuracy on the test set: 0.6
```

PROGRAM

```
#4
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
np.random.seed(42)
X = np.random.rand(100, 2)
y = (X[:, 0] + X[:, 1] > 1).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = tf.keras.Sequential([tf.keras.layers.Input(shape=(2,)), tf.keras.layers.Dense(units=32,
activation='relu'), tf.keras.layers.Dense(units=1, activation='sigmoid')])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=50, batch_size=8, verbose=0)
predictions = model.predict(X_test)
binary_predictions = (predictions > 0.5).astype(int)
accuracy = accuracy_score(y_test, binary_predictions)
print("Accuracy on the test set:", accuracy)
```

OUTPUT

```
1/1 [==============================] - 0s 156ms/step
Accuracy on the test set: 0.9
```

PROGRAM

```
#5
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=5, batch_size=64,
validation_data=(test_images, test_labels))
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

plt.show()

OUTPUT

Downloading data from https://storage.googleapis.com/tensorflow/tfkerasdatasets/mnist.npz 11490434/11490434 [==============================] - 1s 0us/step

Epoch 1/5 938/938 [==============================] - 61s 63ms/step - loss: 0.1887 - accuracy: 0.9410 - val_loss: 0.0621 - val_accuracy: 0.9806

Epoch 2/5 938/938 [==============================] - 56s 60ms/step - loss: 0.0541 - accuracy: 0.9833 - val_loss: 0.0393 - val_accuracy: 0.9878

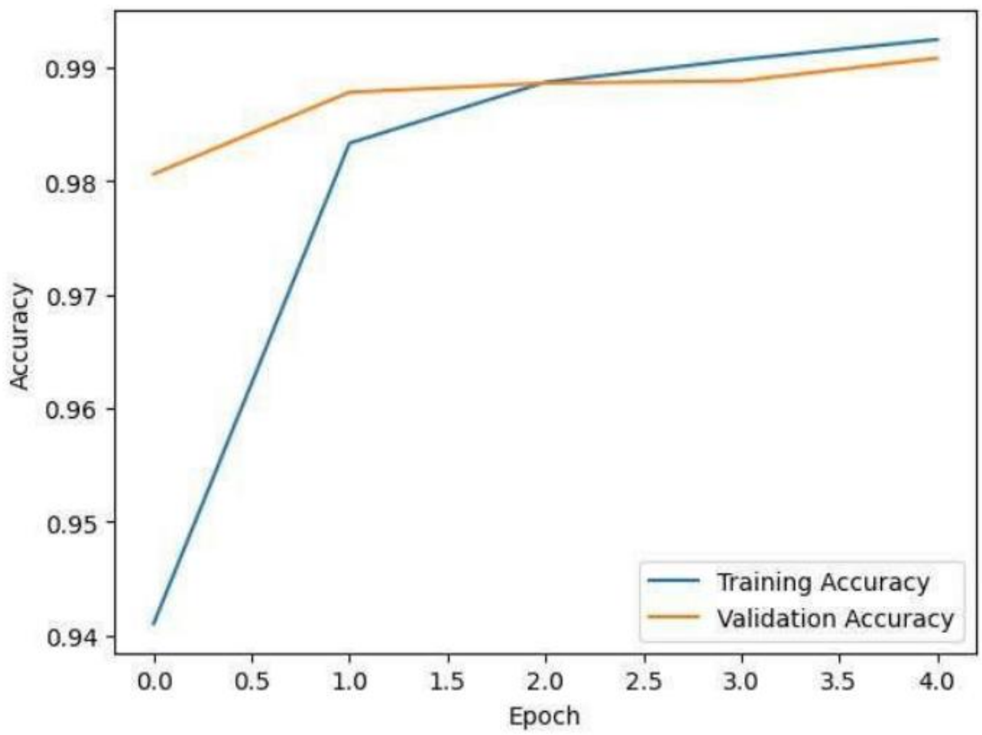
Epoch 3/5 938/938 [==============================] - 54s 57ms/step - loss: 0.0363 - accuracy: 0.9887 - val_loss: 0.0356 - val_accuracy: 0.9886

Epoch 4/5 938/938 [==============================] - 52s 55ms/step - loss: 0.0287 - accuracy: 0.9907 - val_loss: 0.0382 - val_accuracy: 0.9888

Epoch 5/5 938/938 [==============================] - 53s 56ms/step - loss: 0.0239 - accuracy: 0.9924 - val_loss: 0.0288 - val_accuracy: 0.9908

313/313 [==============================] - 3s 9ms/step - loss: 0.0288 - accuracy: 0.9908

Test accuracy: 0.9908000230789185



PROGRAM

```
#6
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True)
history = model.fit(train_images, train_labels, epochs=20, batch_size=64,
validation_data=(test_images, test_labels), callbacks=[early_stopping, model_checkpoint])
best_model = models.load_model('best_model.h5')
```

```
test_loss, test_acc = best_model.evaluate(test_images, test_labels)

print(f'Test accuracy of the best model: {test_acc}')

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()
```

OUTPUT

Epoch 1/20 938/938 [==============================] - 78s 82ms/step - loss: 0.2188 - accuracy: 0.9339 - val_loss: 0.0786 - val_accuracy: 0.9763

Epoch 2/20 3/938 [..............................] - ETA: 47s - loss: 0.0681 accuracy:0.9792

/usr/local/lib/python3.10/dist- packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`. saving_api.save_model

(938/938 [==============================] - 57s 61ms/step - loss: 0.0694 - accuracy: 0.9784 - val_loss: 0.0539 - val_accuracy: 0.9849

Epoch 3/20 938/938 [==============================] - 62s 66ms/step - loss: 0.0488 - accuracy: 0.9849 - val_loss: 0.0473 - val_accuracy: 0.9856

Epoch 4/20 938/938 [==============================] - 57s 61ms/step - loss: 0.0377 - accuracy: 0.9885 - val_loss: 0.0523 - val_accuracy: 0.9837

Epoch 5/20 938/938 [==============================] - 60s 64ms/step - loss: 0.0301 - accuracy: 0.9904 - val_loss: 0.0537 - val_accuracy: 0.9849

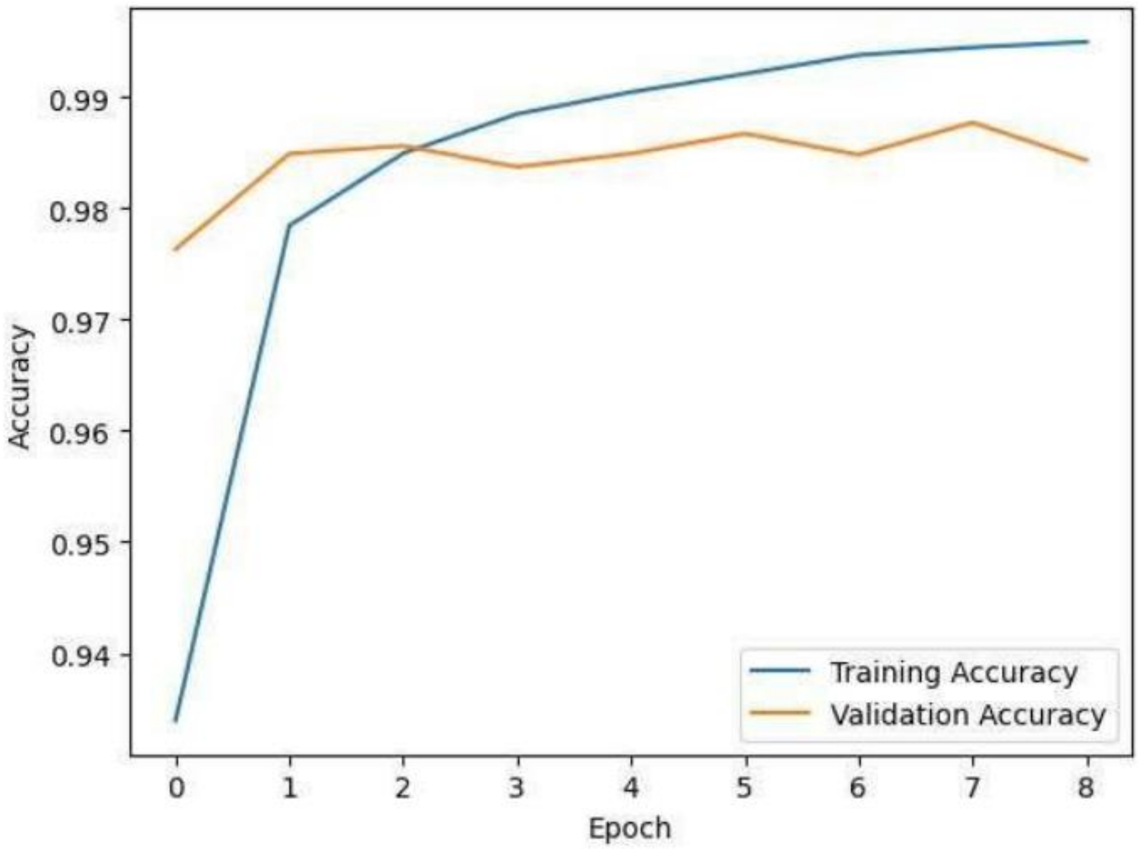
Epoch 6/20 938/938 [==============================] - 58s 62ms/step - loss: 0.0249 - accuracy: 0.9921 - val_loss: 0.0457 - val_accuracy: 0.9867

Epoch 7/20 938/938 [==============================] - 56s 60ms/step - loss: 0.0199 - accuracy: 0.9938 - val_loss: 0.0593 - val_accuracy: 0.9848

Epoch 8/20 938/938 [==============================] - 57s 61ms/step - loss: 0.0167 - accuracy: 0.9944 - val_loss: 0.0480 - val_accuracy: 0.9877

Epoch 9/20 938/938 [==============================] - 56s 60ms/step - loss: 0.0149 - accuracy: 0.9949 - val_loss: 0.0553 - val_accuracy: 0.9843 313/313 [==============================] - 4s 11ms/step - loss: 0.0457 - accuracy: 0.9867

Test accuracy of the best model: 0.9866999983787537



0.99
0.98
0.97
0.96
0.95
0.94
Accuracy
Training Accuracy
Validation Accuracy
0
1
2
3
4
5
6
7
8
Epoch

PROGRAM

```
#7
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import VGG16
from tensorflow.keras.utils import to_categorical
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train.astype('float32') / 255.0, x_test.astype('float32') / 255.0
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
base_model.trainable = False
model = models.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test))
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

OUTPUT

```
Model: "sequential"
_________________________________________________________________
Layer (type)              Output Shape            Param #
=================================================================
```

```
vgg16 (Functional)        (None, 1, 1, 512)       14714688

_________________________________________________________________

flatten (Flatten)         (None, 512)             0

_________________________________________________________________

dense (Dense)             (None, 256)             131328

_________________________________________________________________

dropout (Dropout)         (None, 256)             0

_________________________________________________________________

dense_1 (Dense)           (None, 10)              2570

=================================================================

Total params: 14,848,586 (56.64 MB)

Trainable params: 133,898 (523.04 KB)

Non-trainable params: 14,714,688 (56.13 MB)

_________________________________________________________________


Epoch 1/3

782/782 [==============================] - 611s 781ms/step - loss: 1.5134 - accuracy: 0.4695 - val_loss: 1.2851 - val_accuracy: 0.5500

Epoch 2/3

776/782 [============================>.] - ETA: 3s - loss: 1.3041 - accuracy: 0.5435

Epoch 3/3

782/782 [==============================] - 43s 55ms/step - loss: 0.7345 - accuracy: 0.7520 - val_loss: 0.7901 - val_accuracy: 0.7345

Test accuracy: 0.7345
```

PROGRAM

```
#8
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models, optimizers
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
for layer in base_model.layers:
   layer.trainable = False
model = models.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer=optimizers.Adam(lr=1e-4), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.summary()
datagen = ImageDataGenerator(validation_split=0.2)
batch_size = 32
train_generator = datagen.flow(x_train, y_train, batch_size=batch_size, subset='training')
validation_generator = datagen.flow(x_train, y_train, batch_size=batch_size, subset='validation')
history = model.fit(train_generator, steps_per_epoch=len(train_generator), epochs=10,
validation_data=validation_generator, validation_steps=len(validation_generator))
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc * 100:.2f}%')
```

OUTPUT

```
Model: "sequential"

Layer (type)              Output Shape           Param #
=================================================================
vgg16 (Functional)        (None, 1, 1, 512)      14,714,688
_________________________________________________________________
flatten (Flatten)         (None, 512)            0
_________________________________________________________________
dense (Dense)             (None, 256)            131,328
_________________________________________________________________
dropout (Dropout)         (None, 256)            0
_________________________________________________________________
dense_1 (Dense)           (None, 10)             2,570
=================================================================
Total params: 14,846,586
Trainable params: 133,898
Non-trainable params: 14,712,688
Epoch 1/10
1250/1250 [==============================] - 522s 416ms/step - loss: 1.5227 - accuracy: 0.1009 - val_loss: 1.2595 - val_accuracy: 0.1067
……..
Epoch 10/10
1250/1250 [==============================] - 543s 435ms/step - loss: 1.1144 - accuracy: 0.0960 - val_loss: 1.1163 - val_accuracy: 0.1063
313/313 [==============================] - 100s 321ms/step - loss: 1.1382 - accuracy: 0.109
Test accuracy: 64.50%
```

PROGRAM

```
#9
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
data = pd.DataFrame({
    'text': ['I love this product', 'This is the worst experience', 'Amazing quality and service',
             'Not worth the price', 'Highly recommended', 'Terrible customer service'],
    'sentiment': [1, 0, 1, 0, 1, 0]
})
texts = data['text'].values
labels = data['sentiment'].values
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
max_len = 100
padded_sequences = pad_sequences(sequences, maxlen=max_len)
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels, test_size=0.2,
random_state=42)
model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=64, input_length=max_len))
model.add(LSTM(64, return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
  model.compile(optimizer='adam',           loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)
```

```
loss, accuracy = model.evaluate(X_test, y_test)

print(f'Test Accuracy: {accuracy:.4f}')

test_sentence = ["The product quality is excellent"]

test_sequence = tokenizer.texts_to_sequences(test_sentence)

test_padded_sequence = pad_sequences(test_sequence, maxlen=max_len)

prediction = model.predict(test_padded_sequence)

print(f'Sentiment: {"Positive" if prediction[0] > 0.5 else "Negative"}')
```

OUTPUT

Epoch 1/5

1/1 [==============================] - 1s 1s/step - loss: 0.6928 - accuracy: 0.5000 - val_loss: 0.6913 - val_accuracy: 0.5000

Epoch 2/5

1/1 [==============================] - 0s 38ms/step - loss: 0.6909 - accuracy: 0.6667 - val_loss: 0.6894 - val_accuracy: 0.5000

Epoch 3/5

1/1 [==============================] - 0s 37ms/step - loss: 0.6889 - accuracy: 0.6667 - val_loss: 0.6876 - val_accuracy: 0.5000

Epoch 4/5

1/1 [==============================] - 0s 36ms/step - loss: 0.6869 - accuracy: 0.6667 - val_loss: 0.6857 - val_accuracy: 0.5000

Epoch 5/5

1/1 [==============================] - 0s 39ms/step - loss: 0.6849 - accuracy: 0.6667 - val_loss: 0.6839 - val_accuracy: 0.5000

Test Accuracy: 0.7500

Sentiment: Positive

PROGRAM

```
#10

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, LSTM, RepeatVector, TimeDistributed, Dense

from sklearn.model_selection import train_test_split

data = np.random.rand(1000, 10, 8)

X_train, X_test = train_test_split(data, test_size=0.2, random_state=42)

timesteps = X_train.shape[1]

n_features = X_train.shape[2]

inputs = Input(shape=(timesteps, n_features))

encoded = LSTM(128, activation='relu')(inputs)

decoded = RepeatVector(timesteps)(encoded)

decoded = LSTM(128, return_sequences=True)(decoded)

decoded = TimeDistributed(Dense(n_features))(decoded)

autoencoder = Model(inputs, decoded)

autoencoder.compile(optimizer='adam', loss='mse')

history = autoencoder.fit(X_train, X_train, epochs=5, batch_size=32, validation_data=(X_test,
X_test))

encoder = Model(inputs, encoded)

encoded_data = encoder.predict(X_test)

decoded_data = autoencoder.predict(X_test)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

print("Encoded Data Shape:", encoded_data.shape)
```

print("Decoded Data Shape:", decoded_data.shape)

OUTPUT

Epoch 1/5

25/25 [==============================]14s 206ms/step - loss: 0.1893 - val_loss: 0.0966

Epoch 2/5

25/25 [==============================]8s 139ms/step - loss: 0.0934 - val_loss: 0.0843

Epoch 3/5

25/25 [==============================] 3s 110ms/step - loss: 0.0839 - val_loss: 0.0792

Epoch 4/5

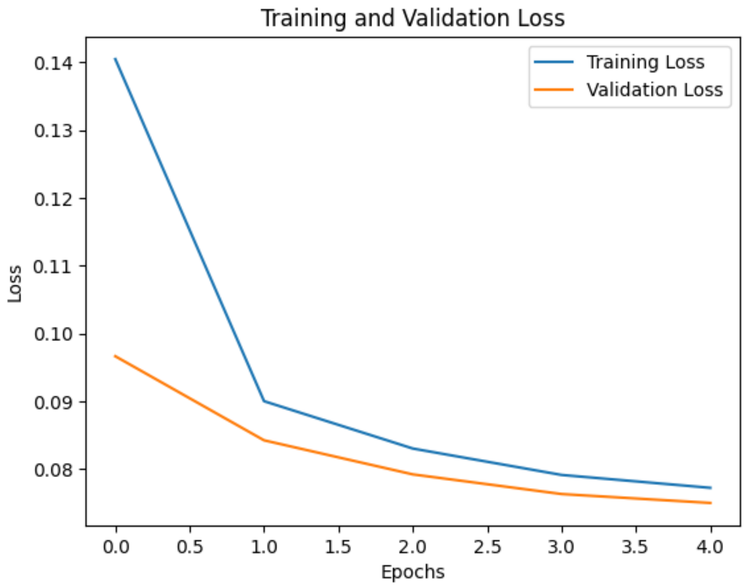
25/25 [==============================]4s 76ms/step - loss: 0.0796 - val_loss: 0.0763

Epoch 5/5

25/25 [==============================]2s 62ms/step - loss: 0.0770 - val_loss: 0.0750

7/7 [==============================]0s 26ms/step

7/7 [==============================]1s 99ms/step



Encoded Data Shape: (200, 128)

Decoded Data Shape: (200, 10, 8)

PROGRAM

```
#11
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
def build_generator(latent_dim):
    model = tf.keras.Sequential()
    model.add(layers.Dense(128, input_dim=latent_dim, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(784, activation='sigmoid'))
    model.add(layers.Reshape((28, 28, 1)))
    return model
def build_discriminator(img_shape):
    model = tf.keras.Sequential()
    model.add(layers.Flatten(input_shape=img_shape))
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model
def build_gan(generator, discriminator):
    discriminator.trainable = False
    model = tf.keras.Sequential()
    model.add(generator)
    model.add(discriminator)
    return model
def load_dataset():
    (X_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
    X_train = X_train / 255.0
    X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
    return X_train
def train_gan(generator, discriminator, gan, X_train, latent_dim, epochs=10000, batch_size=128):
```

```
    for epoch in range(epochs):
        idx = np.random.randint(0, X_train.shape[0], batch_size)
        real_imgs = X_train[idx]
        fake_imgs = generator.predict(np.random.randn(batch_size, latent_dim))
        labels_real = np.ones((batch_size, 1))
        labels_fake = np.zeros((batch_size, 1))
        d_loss_real = discriminator.train_on_batch(real_imgs, labels_real)
        d_loss_fake = discriminator.train_on_batch(fake_imgs, labels_fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
        noise = np.random.randn(batch_size, latent_dim)
        labels_gen = np.ones((batch_size, 1))
        g_loss = gan.train_on_batch(noise, labels_gen)
        if epoch % 100 == 0:
            print(f"Epoch {epoch}, D Loss: {d_loss[0]}, G Loss: {g_loss}")
            save_generated_images(generator, epoch, latent_dim)
def save_generated_images(generator, epoch, latent_dim, examples=10, dim=(1, 10), figsize=(10,
1)):
    noise = np.random.randn(examples, latent_dim)
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(examples, 28, 28)
    plt.figure(figsize=figsize)
    for i in range(generated_images.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        plt.imshow(generated_images[i], interpolation='nearest', cmap='gray_r')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig(f"gan_generated_image_epoch_{epoch}.png")
def main():
    latent_dim = 100
    img_shape = (28, 28, 1)
    generator = build_generator(latent_dim)
```

```
    discriminator = build_discriminator(img_shape)
    gan = build_gan(generator, discriminator)
    X_train = load_dataset()
    discriminator.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    gan.compile(loss='binary_crossentropy', optimizer='adam')
    train_gan(generator, discriminator, gan, X_train, latent_dim)
if __name__ == "__main__":
    main()
```

OUTPUT

```
4/4 [==============================] - 0s 2ms/step
Epoch 0, D Loss: 0.5839875638484955, G Loss: 0.7817459106445312
1/1 [==============================] - 0s 55ms/step
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 2ms/step
:
:
:
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 2ms/step
```

PROGRAM

```
#12
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import pathlib
dataset_url =
"https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos.tar', origin=dataset_url, extract=True)
data_dir = pathlib.Path(data_dir).with_suffix('')
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
PIL.Image.open(str(roses[1]))
tulips = list(data_dir.glob('tulips/*'))
PIL.Image.open(str(tulips[0]))
PIL.Image.open(str(tulips[1]))
batch_size = 32
img_height = 180
img_width = 180
train_ds = tf.keras.utils.image_dataset_from_directory(data_dir, validation_split=0.2,
subset="training", seed=123, image_size=(img_height, img_width), batch_size=batch_size)
val_ds = tf.keras.utils.image_dataset_from_directory(data_dir, validation_split=0.2,
subset="validation", seed=123, image_size=(img_height, img_width), batch_size=batch_size)
class_names = train_ds.class_names
print(class_names)
plt.figure(figsize=(10, 10))
```

```
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
normalization_layer = layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))
num_classes = len(class_names)
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

```
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

model.summary()

epochs = 4

history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)

acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

loss = history.history['loss']

val_loss = history.history['val_loss']

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()

data_augmentation = keras.Sequential([

  layers.RandomFlip("horizontal", input_shape=(img_height, img_width, 3)),

  layers.RandomRotation(0.1),

  layers.RandomZoom(0.1),

])

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")
```

OUTPUT

Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz 228813984/228813984 [==============================] - 1s 0us/step 3670 Found 3670 files belonging to 5 classes.

Using 2936 files for training.

Found 3670 files belonging to 5 classes.

Using 734 files for validation. ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']

(32, 180, 180, 3)

(32, 180, 180, 3)

(32, 180, 180, 3)

(32, 180, 180, 3)

:

:

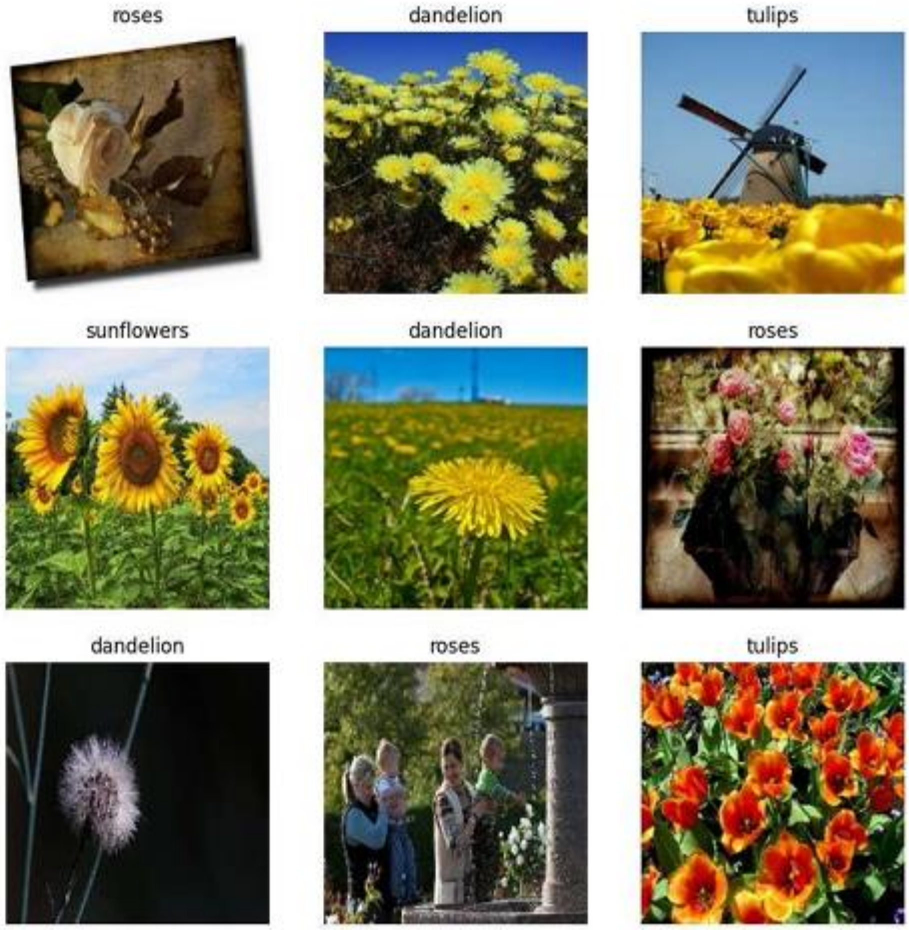
:

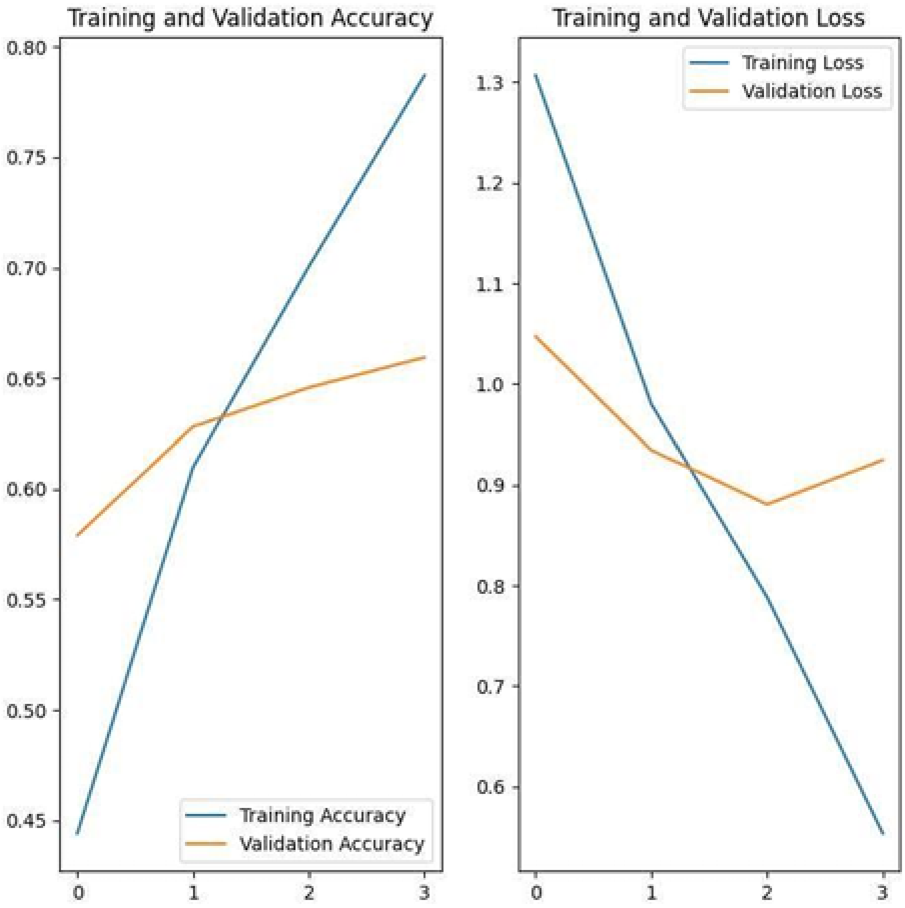
(32, 180, 180, 3)

(32, 180, 180, 3)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                      Output Shape            Param #
=================================================================
 rescaling_1 (Rescaling)           (None, 180, 180, 3)       0
 conv2d (Conv2D)                   (None, 180, 180, 16)      44
 max_pooling2d (MaxPooling2D)      (None, 90, 90, 16)        0
 conv2d_1 (Conv2D)                 (None, 90, 90, 32)        4640
 max_pooling2d_1 (MaxPooling2D)    (None, 45, 45, 32)        0
 conv2d_2 (Conv2D)                 (None, 45, 45, 64)        18496
 max_pooling2d_2 (MaxPooling2D)    (None, 22, 22, 64)        0
```

```
 flatten (Flatten)                    (None, 30976)          0

 dense (Dense)                        (None, 128)            3965056

 dense_1 (Dense)                      (None, 5)              645

=================================================================

Total params: 3,989,285 (15.22 MB)

Trainable params: 3,989,285 (15.22 MB)

Non-trainable params: 0 (0.00 Byte)

_________________________________________________________________

Epoch 1/4

92/92 [==============================] - 104s 1s/step - loss: 1.3069 - accuracy:
0.4441 - val_loss: 1.0472 - val_accuracy: 0.5790

Epoch 2/4

92/92 [==============================] - 100s 1s/step - loss: 0.9803 - accuracy:
0.6097 - val_loss: 0.9344 - val_accuracy: 0.6281

Epoch 3/4

92/92 [==============================] - 116s 1s/step - loss: 0.7882 - accuracy:
0.7006 - val_loss: 0.8806 - val_accuracy: 0.6458

Epoch 4/4

92/92 [==============================] - 103s 1s/step - loss: 0.5535 - accuracy:
0.7871 - val_loss: 0.9246 - val_accuracy: 0.6594
```


roses
dandelion
tulips
sunflowers
dandelion
roses
dandelion
roses
tulips



Training and Validation Accuracy
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0
1
2
3
Training Accuracy
Validation Accuracy
Training and Validation Loss
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0
1
2
3
Training Loss
Validation Loss

PROGRAM

```
#13
import pandas as pd
import numpy as np
from faker import Faker
import random
import datetime
fake = Faker()
num_users = 100
users = [fake.name() for _ in range(num_users)]
num_items = 50
items = [fake.word() for _ in range(num_items)]
num_transactions = 500
data = {
    'user': [random.choice(users) for _ in range(num_transactions)],
    'item': [random.choice(items) for _ in range(num_transactions)],
    'purchase': [random.choice([0, 1]) for _ in range(num_transactions)],
    'timestamp': [fake.date_time_between(start_date="-1y", end_date="now") for _ in
range(num_transactions)]
}
sales_data = pd.DataFrame(data)
sales_data.to_csv('sample_sales_data.csv', index=False)
print(sales_data.head())
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Dense, Concatenate
data = pd.read_csv('/content/sample_sales_data.csv')
user_encoder = LabelEncoder()
```

```
item_encoder = LabelEncoder()
data['user_id'] = user_encoder.fit_transform(data['user'])
data['item_id'] = item_encoder.fit_transform(data['item'])
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
def create_model(num_users, num_items, embedding_size=50):
    user_input = Input(shape=(1,), name='user_input')
    item_input = Input(shape=(1,), name='item_input')
    user_embedding = Embedding(input_dim=num_users, output_dim=embedding_size,
input_length=1)(user_input)
    item_embedding = Embedding(input_dim=num_items, output_dim=embedding_size,
input_length=1)(item_input)
    user_flatten = Flatten()(user_embedding)
    item_flatten = Flatten()(item_embedding)
    concat = Concatenate()([user_flatten, item_flatten])
    dense1 = Dense(128, activation='relu')(concat)
    dense2 = Dense(64, activation='relu')(dense1)
    output = Dense(1, activation='sigmoid')(dense2)
    model = Model(inputs=[user_input, item_input], outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
num_users = len(data['user_id'].unique())
num_items = len(data['item_id'].unique())
model = create_model(num_users, num_items)
model.summary()
train_user = train_data['user_id'].values
train_item = train_data['item_id'].values
train_labels = train_data['purchase'].values
model.fit([train_user, train_item], train_labels, epochs=5, batch_size=64, validation_split=0.2)

test_user = test_data['user_id'].values
test_item = test_data['item_id'].values
test_labels = test_data['purchase'].values
```

```python
accuracy = model.evaluate([test_user, test_item], test_labels)

print(f'Test Accuracy: {accuracy[1]*100:.2f}%')
```

OUTPUT

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape          Param #     Connected to
==================================================================================================
 user_input (InputLayer)        [(None, 1)]           0           []

 item_input (InputLayer)        [(None, 1)]           0           []

 embedding (Embedding)          (None, 1, 50)         4950        ['user_input[0][0]']

 embedding_1 (Embedding)        (None, 1, 50)         2300        ['item_input[0][0]']

 flatten (Flatten)              (None, 50)            0           ['embedding[0][0]']

 flatten_1 (Flatten)            (None, 50)            0           ['embedding_1[0][0]']

 concatenate (Concatenate)      (None, 100)           0           ['flatten[0][0]',
                                                                   'flatten_1[0][0]']

 dense (Dense)                  (None, 128)           12928       ['concatenate[0][0]']

 dense_1 (Dense)                (None, 64)            8256        ['dense[0][0]']

 dense_2 (Dense)                (None, 1)             65          ['dense_1[0][0]']

==================================================================================================
Total params: 28499 (111.32 KB)
Trainable params: 28499 (111.32 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
Epoch 1/5
5/5 [==============================] - 2s 68ms/step - loss: 0.6937 - accuracy: 0.4938 - val_loss: 0.6948 - val_accuracy: 0.4125
Epoch 2/5
5/5 [==============================] - 0s 11ms/step - loss: 0.6864 - accuracy: 0.7281 - val_loss: 0.6955 - val_accuracy: 0.3875
Epoch 3/5
5/5 [==============================] - 0s 13ms/step - loss: 0.6791 - accuracy: 0.8000 - val_loss: 0.6970 - val_accuracy: 0.3625
Epoch 4/5
5/5 [==============================] - 0s 16ms/step - loss: 0.6696 - accuracy: 0.7937 - val_loss: 0.6993 - val_accuracy: 0.3750
Epoch 5/5
5/5 [==============================] - 0s 12ms/step - loss: 0.6555 - accuracy: 0.7875 - val_loss: 0.7039 - val_accuracy: 0.3750
4/4 [==============================] - 0s 6ms/step - loss: 0.6910 - accuracy: 0.4900
Test Accuracy: 49.00%
```

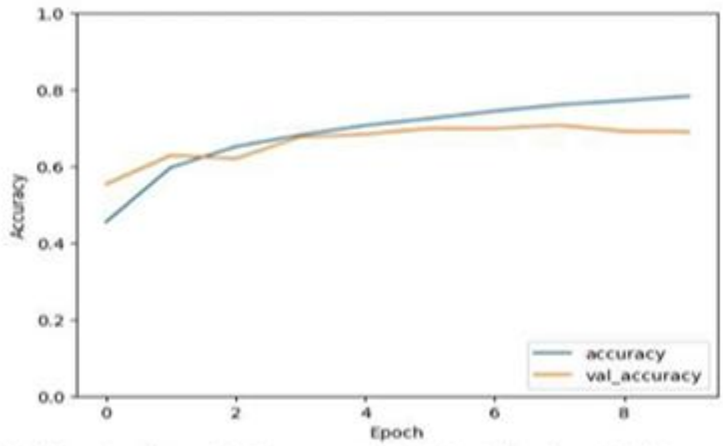
#14

PROGRAM

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

train_images, test_images = train_images / 255.0, test_images / 255.0

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.Conv2D(64, (3, 3), activation='relu')

])

model.add(layers.Flatten()) model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
ue),

metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images,test_labels))

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val_accuracy'], label = 'val_accuracy')

plt.xlabel('Epoch')

 plt.ylabel('Accuracy')

plt.ylim([0, 1])

plt.legend(loc='lower right')

plt.show()

print('\nTest accuracy:', test_acc)
```

OUTPUT

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 11s 0us/step
Epoch 1/10
1563/1563 [==============================] - 82s 51ms/step - loss: 1.5050 - accuracy: 0.4550 - val_loss: 1.2289 - val_accuracy: 0.5537
Epoch 2/10
1563/1563 [==============================] - 67s 43ms/step - loss: 1.1430 - accuracy: 0.5972 - val_loss: 1.0598 - val_accuracy: 0.6288
Epoch 3/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.9956 - accuracy: 0.6515 - val_loss: 1.0731 - val_accuracy: 0.6203
Epoch 4/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.9049 - accuracy: 0.6821 - val_loss: 0.9332 - val_accuracy: 0.6769
Epoch 5/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.8305 - accuracy: 0.7069 - val_loss: 0.9185 - val_accuracy: 0.6838
Epoch 6/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.7771 - accuracy: 0.7252 - val_loss: 0.8801 - val_accuracy: 0.6994
Epoch 7/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.7266 - accuracy: 0.7445 - val_loss: 0.8842 - val_accuracy: 0.6991
Epoch 8/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.6827 - accuracy: 0.7608 - val_loss: 0.8775 - val_accuracy: 0.7075
Epoch 9/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.6483 - accuracy: 0.7719 - val_loss: 0.9336 - val_accuracy: 0.6917
Epoch 10/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.6123 - accuracy: 0.7831 - val_loss: 0.9455 - val_accuracy: 0.6902
```




```
313/313 - 4s - loss: 0.9455 - accuracy: 0.6902 - 4s/epoch - 13ms/step
Test accuracy: 0.6901999711990356
```

#15

PROGRAM

```
!pip install SpeechRecognition pydub

import speech_recognition as sr

recognizer = sr.Recognizer()

audio_file_path = '/content/harvard.wav'  # Replace with your .wav file name

with sr.AudioFile(audio_file_path) as source:
    audio_data = recognizer.record(source)  # Read the entire audio file

try:
    text = recognizer.recognize_google(audio_data)
    print("Recognized text:", text)
except sr.UnknownValueError:
    print("Google Speech Recognition could not understand audio")
except sr.RequestError as e:
    print(f"Could not request results from Google Speech Recognition service; {e}")
```

OUTPUT

Recognized text

the stale smell of old beer lingers it takes heat to bring out the odor a cold dip restores health and zest a salt pickle taste fine with ham tacos al pastor are my favorite a zestful food is the hot cross bun