Ex No: 5 **Date: 23-08-2024**

TRANSFER LEARNING WITH CNN AND VISUALIZATION

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 transfer learning model with convolutional neural network using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

PROGRAM:

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.applications import VGG16
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import LambdaCallback
import matplotlib.pyplot as plt

# Step 1: Download and load the dataset (Fashion MNIST)
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

```
# Step 2: Perform analysis and preprocessing of the dataset
# Reshape the data to include a single color channel and normalize the images
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0

# Resize images to (32, 32, 1) to meet VGG16's input requirements (minimum size 32x32)
x_train = tf.image.resize(x_train, (32, 32))
x_test = tf.image.resize(x_test, (32, 32))

# Convert grayscale images to 3 channels (RGB) to fit the input requirements of VGG16
x_train = tf.image.grayscale_to_rgb(x_train)
x_test = tf.image.grayscale_to_rgb(x_test)

# Convert labels to one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Step 3: Build a simple neural network model using Keras/TensorFlow with Transfer Learning
# Use VGG16 as the base model with pre-trained weights from ImageNet
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the layers of the base model
base_model.trainable = False

# Build the full model
```

```
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),  # Pooling layer
    Dense(64, activation='relu'),  # Fully connected layer with fewer neurons
    Dense(10, activation='softmax')  # Output layer with 10 neurons for 10 classes
])

# Step 4: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Lists to store accuracy values manually
train_acc = []
val_acc = []

# Define a callback to manually store accuracy values
accuracy_callback = LambdaCallback(
    on_epoch_end=lambda epoch, logs: (
        train_acc.append(logs['accuracy']),
        val_acc.append(logs['val_accuracy'])
    )
)

# Train the model with the callback
model.fit(x_train, y_train, epochs=2, validation_split=0.2, batch_size=64,
callbacks=[accuracy_callback], verbose=2)
```

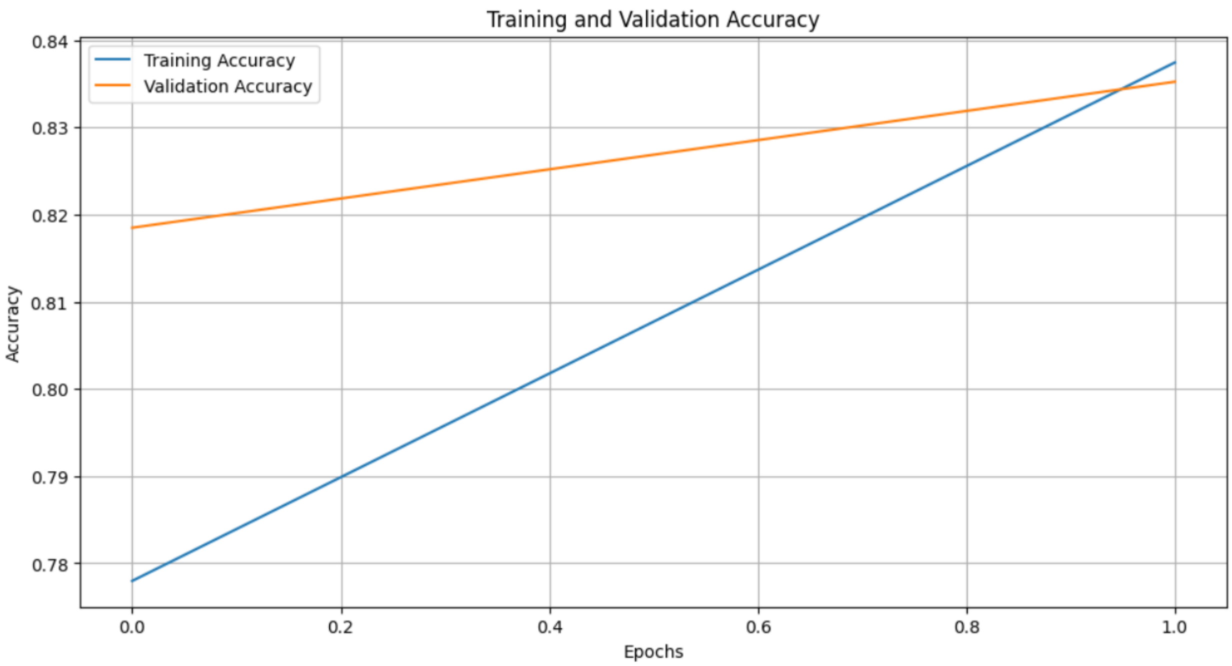
```
# Step 5: Perform prediction with the test dataset
predictions = model.predict(x_test)

# Step 6: Calculate performance metrics
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')

# Visualization: Display a few test images and their predictions
plt.figure(figsize=(10, 5))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(x_test[i].numpy().astype("uint8"))
    plt.title(f'Pred: {predictions[i].argmax()}')
    plt.axis('off')
plt.show()

# Plotting training and validation accuracy without using history
plt.figure(figsize=(12, 6))
plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

plt.grid(True)

plt.show()

OUTPUT:

```
Epoch 1/2
750/750 - 615s - loss: 0.6658 - accuracy: 0.7780 - val_loss: 0.5025 - val_accuracy: 0.8185 - 615s/epoch - 820ms/step
Epoch 2/2
750/750 - 1632s - loss: 0.4505 - accuracy: 0.8375 - val_loss: 0.4472 - val_accuracy: 0.8353 - 1632s/epoch - 2s/step
313/313 [==============================] - 90s 276ms/step
313/313 [==============================] - 82s 260ms/step - loss: 0.4495 - accuracy: 0.8382
Test accuracy: 0.8382
```



RESULT:

Thus, a convolutional neural network with transfer learning was successfully implemented.