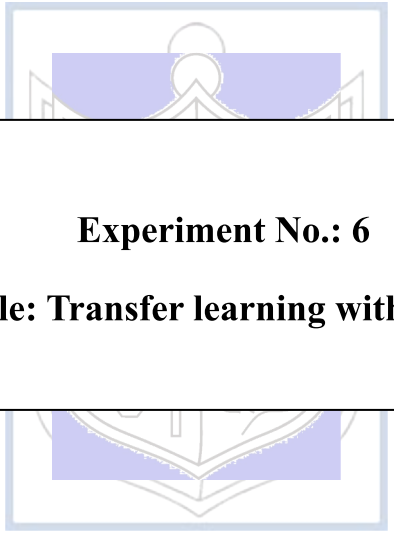


Experiment No.: 6

Title: Transfer learning with CNN

Batch: HO–DL 1 **Roll No.: 16010422234** **Experiment No. 06**

Aim: To implement transfer learning with Convolutional Neural Network.

Theory:

Transfer learning is the reuse of a pre-trained model on a new problem. It's currently very popular in deep learning because it can train deep neural networks with comparatively little data. This is very useful in the data science field since most real-world problems typically do not have millions of labelled data points to train such complex models. In transfer learning, the knowledge of an already trained machine learning model is applied to a different but related problem. For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the knowledge that the model gained during its training to recognize other objects like sunglasses. With transfer learning, we basically try to exploit what has been learned in one task to improve generalization in another. We transfer the weights that a network has learned at "task A" to a new "task B."

When to Use Transfer Learning-

When we don't have enough annotated data to train our model with. When there is a pre-trained model that has been trained on similar data and tasks. If you used TensorFlow to train the original model, you might simply restore it and retrain some layers for your job. Transfer learning, on the other hand, only works if the features learnt in the first task are general, meaning they can be applied to another activity. Furthermore, the model's input must be the same size as it was when it was first trained.

1. TRAINING A MODEL TO REUSE IT

Consider the situation in which you wish to tackle Task A but lack the necessary data to train a deep neural network. Finding a related task B with a lot of data is one method to get around this. Utilize the deep neural network to train on task B and then use the model to solve task A. The problem you're seeking to solve will decide whether you need to employ the entire model or just a few layers.

If the input in both jobs is the same, you might reapply the model and make predictions for your new input. Changing and retraining distinct task-specific layers and the output layer, on the other hand, is an approach to investigate.

2. USING A PRE-TRAINED MODEL

The second option is to employ a model that has already been trained. There are a number of these models out there, so do some research beforehand. The number of layers to reuse and retrain is determined by the task. Keras consists of nine pre-trained models used in transfer learning, prediction, and fine-tuning. These models, as well as some quick lessons on how to utilise them, may be found here. Many research institutions also make trained models accessible. The most popular application of this form of transfer learning is deep learning.

Activity:

1) Import requisite libraries using Tensorflow and Keras.
2) Load the selected dataset.
3) Visualize and display random images belonging to each class.
4) Select the model to be used for transfer learning.
5) Using a pre-trained model. Develop CNN for your dataset.
6) Print Model Summary and display architecture diagram.
7) Compile and fit the model on the train dataset.
8) Calculate training and the cross-validation accuracy.
9) Redefine the model by using appropriate regularization techniques to prevent overfitting.
10) Fit the data on the regularized model.
11) Calculate and plot loss function and accuracy using suitable loss function.
12) Display classification Report for regularized CNN model.
13) Comment on output.

Program:

```
# Importing necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import VGG16
```

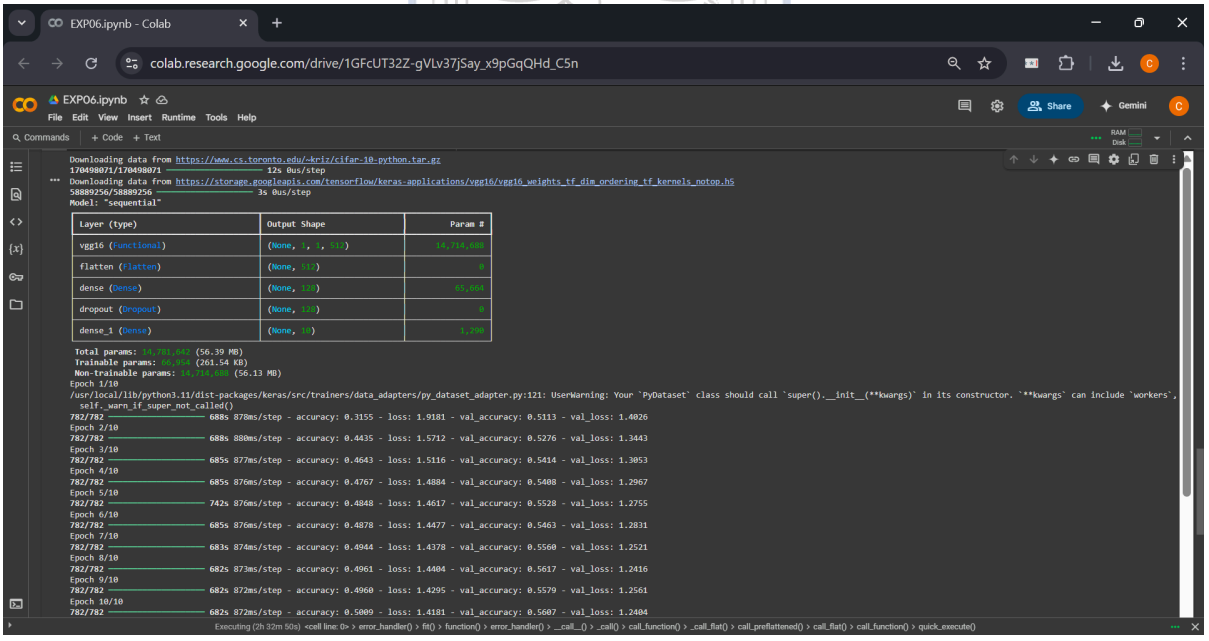
```
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping

# Load the dataset (Example: using CIFAR-10 dataset)
from tensorflow.keras.datasets import cifar10

# Load the dataset and split it into training and testing sets
(train_images, train_labels), (test_images, test_labels) =
cifar10.load_data()

# Preprocess the dataset: normalize the images
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

# Define the model for Transfer Learning using VGG16
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(32, 32, 3))

# Freeze the base model layers
base_model.trainable = False

# Create a new model using the pre-trained VGG16 as base
model = models.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')  # 10 classes in CIFAR-10
])

# Print the model summary
model.summary()

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

# Data augmentation to improve generalization
datagen = ImageDataGenerator(
    rotation_range=20,
```

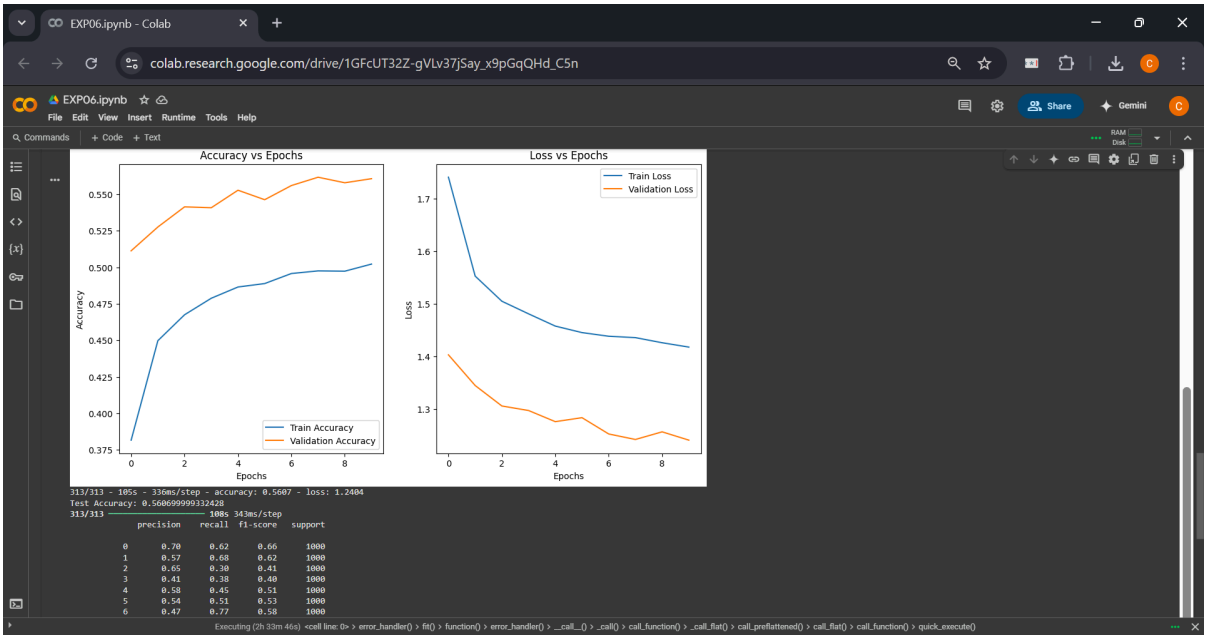
```
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Fit the model with data augmentation
history = model.fit(datagen.flow(train_images, train_labels,
batch_size=64),
                    epochs=10,
                    validation_data=(test_images, test_labels),
                    callbacks=[EarlyStopping(monitor='val_loss',
patience=3)])

# Calculate the accuracy
train_acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]

# Plot the loss and accuracy
plt.figure(figsize=(12, 6))

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

# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Evaluate the model on test data
```

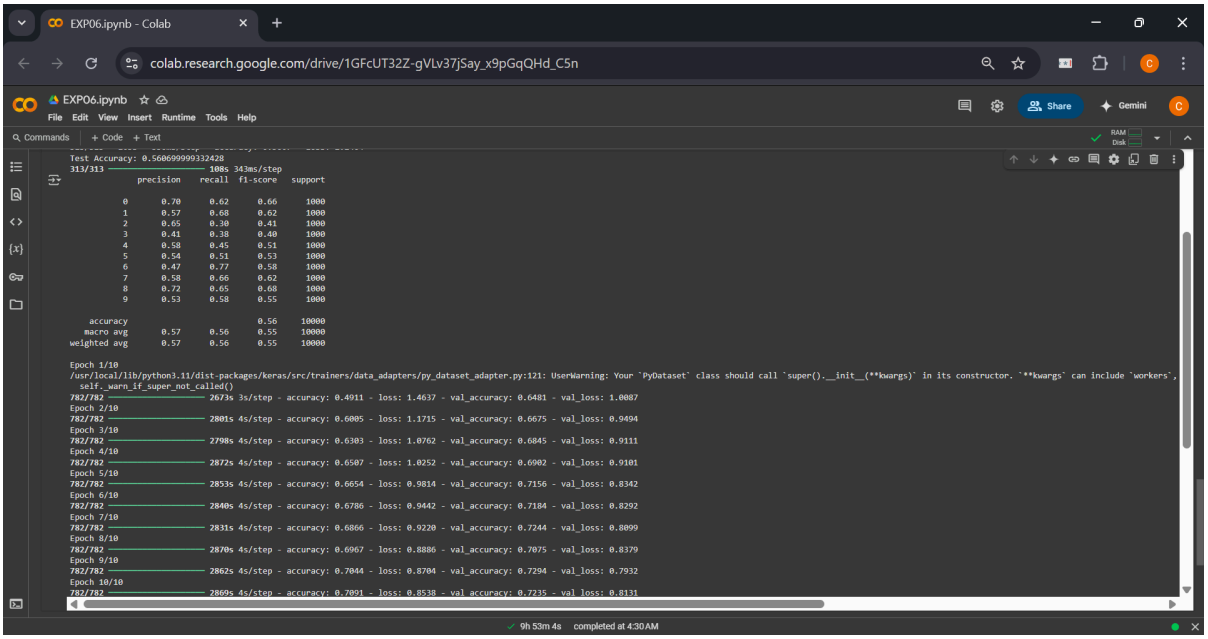
```
test_loss, test_acc = model.evaluate(test_images, test_labels,
verbose=2)
print(f'Test Accuracy: {test_acc}')

# Display classification report
y_pred = model.predict(test_images)
y_pred_classes = np.argmax(y_pred, axis=1)
print(classification_report(test_labels, y_pred_classes))

# Now let's fine-tune the model by unfreezing some layers of the base
model
base_model.trainable = True
for layer in base_model.layers[:15]:
    layer.trainable = False

# Recompile the model after unfreezing some layers
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Fit the model again with fine-tuning
history_fine_tuned = model.fit(datagen.flow(train_images, train_labels,
batch_size=64),
                               epochs=10,
                               validation_data=(test_images,
test_labels),

callbacks=[EarlyStopping(monitor='val_loss', patience=3)])

# Plot the loss and accuracy for fine-tuned model
plt.figure(figsize=(12, 6))

# Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(history_fine_tuned.history['accuracy'], label='Train
Accuracy')
plt.plot(history_fine_tuned.history['val_accuracy'], label='Validation
Accuracy')
plt.title('Fine-tuned Accuracy vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

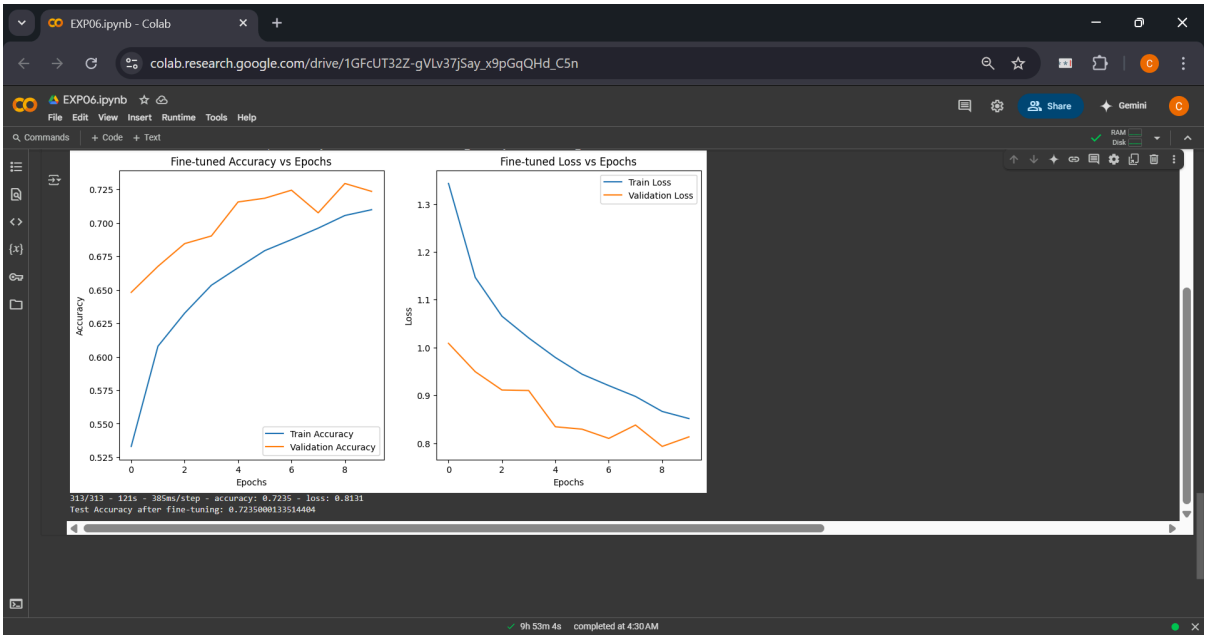
```
# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history_fine_tuned.history['loss'], label='Train Loss')
plt.plot(history_fine_tuned.history['val_loss'], label='Validation
Loss')
plt.title('Fine-tuned Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Evaluate the fine-tuned model
test_loss, test_acc = model.evaluate(test_images, test_labels,
verbose=2)
print(f'Test Accuracy after fine-tuning: {test_acc}')
```

Output:




EXP06.ipynb - Colab
colab.research.google.com/drive/1GFcUT32Z-gVLv37jSay_x9pGqQHd_C5n
EXP06.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Accuracy vs Epochs
Accuracy
Epochs
Train Accuracy
Validation Accuracy
Loss vs Epochs
Loss
Epochs
Train Loss
Validation Loss
313/313 - 105s - 336ms/step - accuracy: 0.5607 - loss: 1.2404
Test Accuracy: 0.560699999332428
313/313 108s 343ms/step
precision recall f1-score support
0 0.70 0.62 0.66 1000
1 0.57 0.68 0.62 1000
2 0.65 0.30 0.41 1000
3 0.41 0.38 0.40 1000
4 0.58 0.45 0.51 1000
5 0.54 0.51 0.53 1000
6 0.47 0.77 0.58 1000


EXP06.ipynb - Colab
colab.research.google.com/drive/1GFcUT32Z-gVLv37jSay_x9pGqQHd_C5n
EXP06.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Test Accuracy: 0.560699999332428
313/313 108s 343ms/step
precision recall f1-score support
0 0.70 0.62 0.66 1000
1 0.57 0.68 0.62 1000
2 0.65 0.30 0.41 1000
3 0.41 0.38 0.40 1000
4 0.58 0.45 0.51 1000
5 0.54 0.51 0.53 1000
6 0.47 0.77 0.58 1000
7 0.58 0.66 0.62 1000
8 0.72 0.65 0.68 1000
9 0.53 0.58 0.55 1000
accuracy 0.56 10000
macro avg 0.57 0.56 0.55 10000
weighted avg 0.57 0.56 0.55 10000
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`,
self._warn_if_super_not_called()
782/782 2673s 3s/step - accuracy: 0.4911 - loss: 1.4637 - val_accuracy: 0.6481 - val_loss: 1.0087
Epoch 2/10
782/782 2801s 4s/step - accuracy: 0.6005 - loss: 1.1715 - val_accuracy: 0.6675 - val_loss: 0.9494
Epoch 3/10
782/782 2798s 4s/step - accuracy: 0.6303 - loss: 1.0762 - val_accuracy: 0.6845 - val_loss: 0.9111
Epoch 4/10
782/782 2872s 4s/step - accuracy: 0.6507 - loss: 1.0252 - val_accuracy: 0.6902 - val_loss: 0.9101
Epoch 5/10
782/782 2853s 4s/step - accuracy: 0.6654 - loss: 0.9814 - val_accuracy: 0.7156 - val_loss: 0.8342
Epoch 6/10
782/782 2840s 4s/step - accuracy: 0.6786 - loss: 0.9442 - val_accuracy: 0.7184 - val_loss: 0.8292
Epoch 7/10
782/782 2831s 4s/step - accuracy: 0.6866 - loss: 0.9220 - val_accuracy: 0.7244 - val_loss: 0.8099
Epoch 8/10
782/782 2870s 4s/step - accuracy: 0.6967 - loss: 0.8886 - val_accuracy: 0.7075 - val_loss: 0.8379
Epoch 9/10
782/782 2862s 4s/step - accuracy: 0.7044 - loss: 0.8704 - val_accuracy: 0.7294 - val_loss: 0.7932
Epoch 10/10
782/782 2869s 4s/step - accuracy: 0.7091 - loss: 0.8538 - val_accuracy: 0.7235 - val_loss: 0.8131
9h 53m 4s completed at 4:30 AM



CO – 3: Assimilate fundamentals of Convolutional Neural Network.

Conclusion:

In this experiment, we explored the concept of transfer learning with Convolutional Neural Networks (CNNs). By utilizing pre-trained models, we were able to efficiently tackle problems with limited labeled data. The use of a pre-trained model allowed us to transfer learned features from one task to another, enhancing the generalization ability of the model on new, unseen data. The regularization techniques applied during the training process helped prevent overfitting, ensuring that the model performed well both on training data and during cross-validation. Overall, transfer learning demonstrated its effectiveness in improving model accuracy, especially when data is scarce, making it a valuable tool in the field of deep learning.

Grade: AA / AB / BB / BC / CC / CD / DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Josh Patterson and Adam Gibson, “Deep Learning A Practitioner’s Approach”, O’Reilly Media, 2017

2. Nikhil Buduma, “Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence Algorithms”, O’Reilly Media 2017

3. Ian Goodfellow Yoshua Bengio Aaron Courville. “Deep Learning”, MIT Press 2017
