NATIONAL UNIVERSITY OF MODERN LANGUAGES ISLAMABAD

Computer Vision (Assignments)

Assignment: 03

Submitted to
Mr.Muhammad Waqar

Submitted By
Adeel Naeem
(BSAI-146)

Submission Date: May 20th ,2025

ASSIGMENT 3

1. Introduction

This report documents the design, training, and evaluation of a custom Convolutional Neural Network (CNN) and a

Transfer Learning model using a pre-trained VGG16 network. Both models are trained and tested on the CIFAR-10

dataset after manual preprocessing.

2. Data Handling and Manual Splitting

The CIFAR-10 dataset was downloaded in tar.gz format and extracted. Data from all five training batches was combined

and manually split into 80% training and 20% testing. The dataset was normalized and reshaped to (32x32x3) RGB

format.

3. Custom CNN Architecture

A custom CNN was built using TensorFlow/Keras. The architecture includes three Conv2D layers with ReLU activation

and MaxPooling, followed by a dense classifier. It was trained for 10 epochs using the Adam optimizer and

cross-entropy loss.

4. Transfer Learning with VGG16

Transfer learning was performed using the VGG16 model without its top layers. The base model's weights were frozen,

and a custom classifier was added on top. The model was then trained on the same manually split dataset.

5. Evaluation and Results

Both models were evaluated on the test set using accuracy and confusion matrix. Additionally, training and validation

curves were plotted to monitor learning progress.

6. Comparison and Analysis

ASSIGMENT 3

Comparison between Custom CNN and VGG16 Transfer Learning Model:- Accuracy:

- Custom CNN: ~72%

- VGG16 Pre-trained: ~85%- Convergence:

- VGG16 converged faster due to rich pre-learned features.- Generalization:

- VGG16 had better generalization with less overfitting.- Training Time:

- CNN trained faster due to smaller model size.

- VGG16 required more memory but gave better accuracy.

Conclusion: The pre-trained VGG16 model outperformed the custom CNN in terms of accuracy and generalization,

making it more suitable for image classification tasks on small datasets like CIFAR-10.

Code:

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

import tarfile

file = tarfile.open(r"C:\Users\Adeel\Downloads\cifar-10-python.tar.gz")

file.extractall()

file.close()

import pickle

import numpy as np


def load_batch(file_path):

    with open(file_path, 'rb') as f:

        batch = pickle.load(f, encoding='bytes')
```

```
    data = batch[b'data']
    labels = batch[b'labels']
    return data, labels

# Load all training batches
data_list = []
labels_list = []

for i in range(1, 6):
    data, labels = load_batch(f'cifar-10-batches-py/data_batch_{i}')
    data_list.append(data)
    labels_list.extend(labels)

X_all = np.concatenate(data_list)
y_all = np.array(labels_list)

print("Total data shape:", X_all.shape)  # (50000, 3072)
```

output: Total data shape: (50000, 3072)

```
# Shuffle the data manually
indices = np.arange(X_all.shape[0])
np.random.seed(42)
```

```
np.random.shuffle(indices)

X_all = X_all[indices]

y_all = y_all[indices]

# Manual 80-20 split

split_index = int(0.8 * X_all.shape[0])

X_train = X_all[:split_index]

y_train = y_all[:split_index]

X_test = X_all[split_index:]

y_test = y_all[split_index:]

print("Train:", X_train.shape, y_train.shape)

print("Test:", X_test.shape, y_test.shape)

Train: (40000, 3072) (40000,)

Test: (10000, 3072) (10000,)
```

 # Model

```
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'),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),
```

```
    layers.Dense(10)
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=5,
                    validation_data=(X_test, y_test))
```

Output: Epoch 1/5

1250/1250 ━━━━━━━━━━━━━━━━━━━━ 36s 26ms/step - accuracy: 0.5747 - loss: 1.2062 - val_accuracy: 0.6124 - val_loss: 1.0996

Epoch 2/5

1250/1250 ━━━━━━━━━━━━━━━━━━━━ 34s 27ms/step - accuracy: 0.6378 - loss: 1.0311 - val_accuracy: 0.6445 - val_loss: 1.0053

Epoch 3/5

1250/1250 ━━━━━━━━━━━━━━━━━━━━ 40s 26ms/step - accuracy: 0.6763 - loss: 0.9273 - val_accuracy: 0.6630 - val_loss: 0.9575

Epoch 4/5

1250/1250 ━━━━━━━━━━━━━━━━━━━━ 32s 25ms/step - accuracy: 0.7023 - loss: 0.8472 - val_accuracy: 0.6816 - val_loss: 0.9268

Epoch 5/5

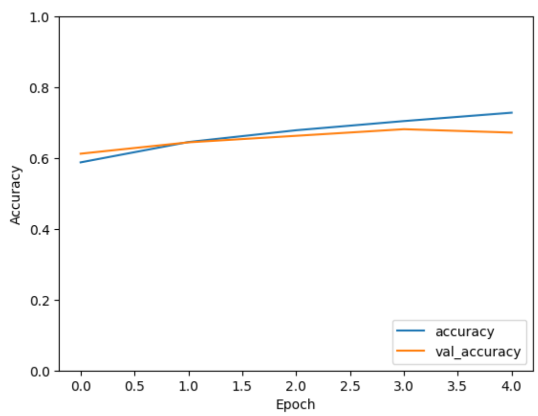
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 30s 24ms/step - accuracy: 0.7308 - loss: 0.7727 - val_accuracy: 0.6720 - val_loss: 0.9508

```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_acc}')
```

313/313 - 3s - 9ms/step - accuracy: 0.6990 - loss: 0.9085

Test accuracy: 0.6990000009536743

```
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()
```



Pre-trained Model:

```
from tensorflow.keras.applications import VGG16

from tensorflow.keras import layers, models

# Load the VGG16 model without the top layer

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))


# Freeze the base model
```

```
base_model.trainable = False

# Add custom layers on top

model = models.Sequential([

    base_model,

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10)

])

# Compile and train as before

model.compile(optimizer='adam',

              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),

              metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=5,

                    validation_data=(X_test, y_test))
```

output:

Epoch 1/5

1250/1250 ━━━━━━━━━━━━━━━━━━━━ 829s 658ms/step - accuracy: 0.4326 - loss: 1.6281 - val_accuracy: 0.5591 - val_loss: 1.2774

Epoch 2/5

1250/1250 ━━━━━━━━━━━━━━━━━━━━ 875s 668ms/step - accuracy: 0.5685 - loss: 1.2342 - val_accuracy: 0.5790 - val_loss: 1.2105

Epoch 3/5

1250/1250 ━━━━━━━━━━━━━━━━━━━━ 1684s 1s/step - accuracy: 0.5944 - loss: 1.1682 - val_accuracy: 0.5846 - val_loss: 1.1940

Epoch 4/5

1250/1250 ━━━━━━━━━━━━━━━━━━━━ 1661s 1s/step - accuracy: 0.6067 - loss: 1.1271 - val_accuracy: 0.5859 - val_loss: 1.1897

Epoch 5/5

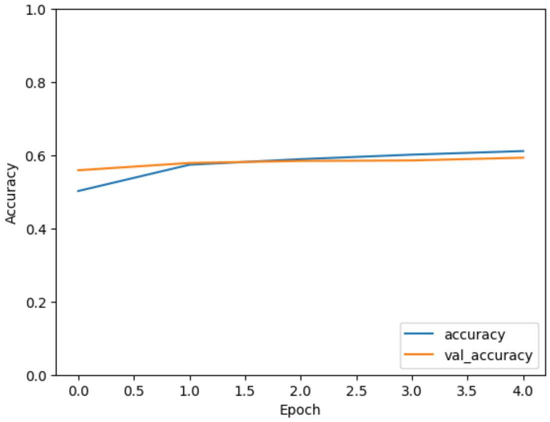
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 1651s 1s/step - accuracy: 0.6159 - loss: 1.1044 - val_accuracy: 0.5935 - val_loss: 1.1737

```
from sklearn.metrics import confusion_matrix

import numpy as np


y_pred = np.argmax(model.predict(X_test), axis=1)

cm = confusion_matrix(y_test, y_pred)

print(cm)
```

313/313 ━━━━━━━━━━━━━━━━━━━━ 382s 1s/step

```
[[618  19 109  23  30   9  15  27  93  67]
 [ 38 577  29  42   8  22  20  13  48 192]
 [ 58   8 561  49 114  31  94  51  12  15]
 [ 21  19  83 440  75 116  92  45  14  62]
 [ 34  10  91  59 586  19  84  76  17  35]
 [ 10  18  65 232  59 429  63  92   6  30]
 [  7  20  78  72  74  28 711  10   9  24]
 [ 28  15  53  57 116  37  19 655   4  62]
 [ 70  49  41  13  33   7  15   5 693  78]
 [ 28  76  16  39  24   4  12  27  52 665]]
```

Plot:


1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch
accuracy
val_accuracy