Practical-1

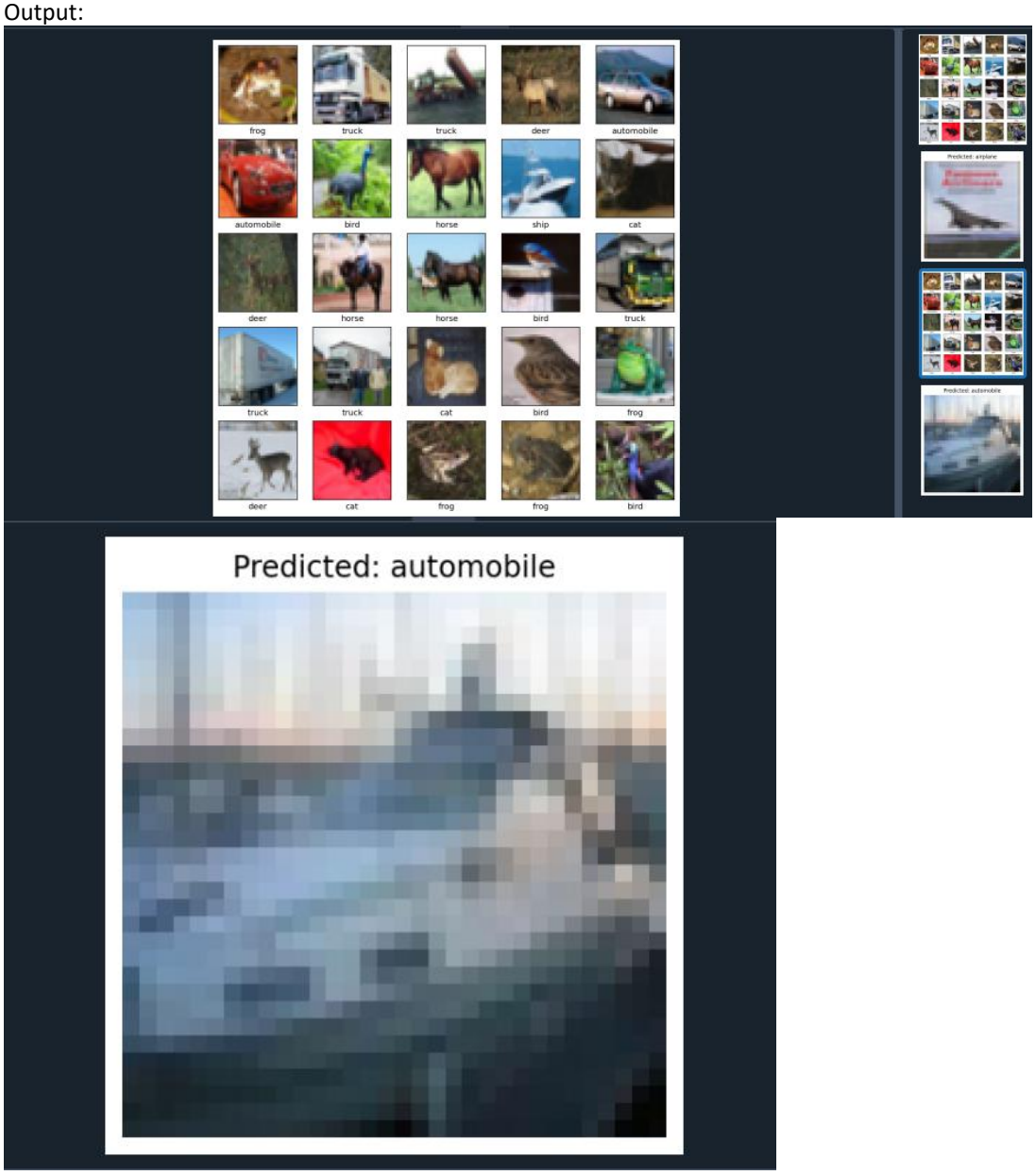
1.A) Aim: Implement advanced deep learning algorithms such as CNN using Tensorflow.

Code:

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

#Load CIFAR-10 dataset
(x_train,y_train), (x_test,y_test) = tf.keras.datasets.cifar10.load_data()

#Normalize pixel values to [0,1]
x_train,x_test = x_train/255.0, x_test/255

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
   plt.subplot(5,5,i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(x_train[i])
   #The CIFAR labels happen to be arrays,which is why you need the extra index
   plt.xlabel(class_names[y_train[i][0]])
plt.show()

# Build the CNN 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,activation='softmax')
])

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

#Train teh model
model.fit(x_train,y_train,epochs=8, batch_size=64,validation_split=0.2)

#Evaluate on test set
test_loss,test_acc = model.evaluate(x_test,y_test)
print(f"Test accuracy: {test_acc:.4f}")

#PREDICTION

#Pick one test image
```

```
img = x_test[2]

# Add batch dimension for prediction
img_batch = np.expand_dims(img,axis=0)

#Predict class probabilities
pred_probs = model.predict(img_batch)

# Get predicted class index
predicted_class = np.argmax(pred_probs)

#CIFAR-10 class names
class_name = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plt.imshow(img)
plt.title(f"Predicted: {class_names[predicted_class]}")
plt.axis('off')
plt.show()
```

Output:



```
In [2]: %runfile 'C:/Users/Kuchira Ittef/.spyder-py3/temp.py' --wdir
Epoch 1/8
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 24ms/step - accuracy: 0.4257 - loss: 1.5854 - val_accuracy: 0.5028 - val_loss: 1.3625
Epoch 2/8
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 26ms/step - accuracy: 0.5539 - loss: 1.2464 - val_accuracy: 0.5952 - val_loss: 1.1568
Epoch 3/8
625/625 ━━━━━━━━━━━━━━━━━━━━ 14s 22ms/step - accuracy: 0.6143 - loss: 1.0952 - val_accuracy: 0.6202 - val_loss: 1.1017
Epoch 4/8
625/625 ━━━━━━━━━━━━━━━━━━━━ 14s 23ms/step - accuracy: 0.6533 - loss: 0.9916 - val_accuracy: 0.6296 - val_loss: 1.0507
Epoch 5/8
625/625 ━━━━━━━━━━━━━━━━━━━━ 14s 22ms/step - accuracy: 0.6751 - loss: 0.9221 - val_accuracy: 0.6649 - val_loss: 0.9774
Epoch 6/8
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.6969 - loss: 0.8639 - val_accuracy: 0.6737 - val_loss: 0.9345
Epoch 7/8
625/625 ━━━━━━━━━━━━━━━━━━━━ 24s 38ms/step - accuracy: 0.7154 - loss: 0.8097 - val_accuracy: 0.6822 - val_loss: 0.9209
Epoch 8/8
625/625 ━━━━━━━━━━━━━━━━━━━━ 23s 37ms/step - accuracy: 0.7343 - loss: 0.7585 - val_accuracy: 0.6981 - val_loss: 0.8758
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.6995 - loss: 0.8736
Test accuracy: 0.6995
```

1 B) Aim: Implement advanced deep learning algorithms such as RNN.

Code:

```
# Simple RNN Sentiment Analysis
# Sample data (tiny dataset for demo)
reviews = [
   "I loved the movie, it was fantastic!",       # positive
   "Absolutely terrible, worst film ever.",      # negative
   "Great acting and wonderful story.",          # positive
   "The movie was boring and too long.",         # negative
   "An excellent and emotional performance.",    # positive (fixed typo)
   "I hated it, very disappointing."             # negative
]
labels = [1, 0, 1, 0, 1, 0]  # 1: positive, 0: negative

import torch
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import DataLoader, TensorDataset
from collections import Counter
import re

# Tokenize and clean
def preprocess(text):
   text = text.lower()
   text = re.sub(r"[^\w\s]", "", text)
   return text.split()

tokenized_reviews = [preprocess(review) for review in reviews]

# Build vocab
all_words = [word for review in tokenized_reviews for word in review]
word_counts = Counter(all_words)
vocab = {word: i+1 for i, (word, _) in enumerate(word_counts.most_common())}  # start indexing from 1
vocab['<PAD>'] = 0  # padding token
vocab['<UNK>'] = len(vocab)  # unknown token

# Encode reviews
encoded_reviews = [[vocab.get(word, vocab['<UNK>']) for word in review] for review in tokenized_reviews]

# Pad sequences
padded_reviews = pad_sequence([torch.tensor(seq) for seq in encoded_reviews], batch_first=True)

labels_tensor = torch.tensor(labels)
dataset = TensorDataset(padded_reviews, labels_tensor)
dataloader = DataLoader(dataset, batch_size=2, shuffle=True)

# Model
import torch.nn as nn
class ReviewRNN(nn.Module):
   def __init__(self, vocab_size, embed_size, hidden_size, num_classes):
      super(ReviewRNN, self).__init__()
      self.embedding = nn.Embedding(vocab_size, embed_size, padding_idx=0)
      self.rnn = nn.RNN(embed_size, hidden_size, batch_first=True)
      self.fc = nn.Linear(hidden_size, num_classes)

   def forward(self, x):
      x = self.embedding(x)
      output, _ = self.rnn(x)
```

```
        out = self.fc(output[:, -1, :])  # last timestep hidden state
        return out

# Training setup
vocab_size = len(vocab)
embed_size = 32
hidden_size = 64
num_classes = 2

model = ReviewRNN(vocab_size, embed_size, hidden_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# Train loop
num_epochs = 8
for epoch in range(num_epochs):
    for batch_x, batch_y in dataloader:
        outputs = model(batch_x)
        loss = criterion(outputs, batch_y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")

# Prediction
def predict_sentiment(text):
    model.eval()
    with torch.no_grad():
        tokens = preprocess(text)
        encoded = [vocab.get(word, vocab['<UNK>']) for word in tokens]
        tensor = torch.tensor(encoded).unsqueeze(0)
        tensor = pad_sequence([tensor.squeeze()], batch_first=True)  # ensure padding shape
        output = model(tensor)
        pred = torch.argmax(output, dim=1).item()
        return "Positive" if pred == 1 else "Negative"

# Test prediction
print(predict_sentiment("I really enjoyed the movie"))
print(predict_sentiment("It was the worst movie ever"))
print(predict_sentiment("An excellent and emotional performance."))
print(predict_sentiment("Amazing movie!."))
```

output:

```
In [3]: %runfile 'C:/Users/Ruchira Itte/.spyder-py3/untitled0.py' --wdir
Epoch [1/8], Loss: 0.8382
Epoch [2/8], Loss: 0.1023
Epoch [3/8], Loss: 0.0062
Epoch [4/8], Loss: 0.0010
Epoch [5/8], Loss: 0.0005
Epoch [6/8], Loss: 0.0002
Epoch [7/8], Loss: 0.0002
Epoch [8/8], Loss: 0.0003
Negative
Positive
Negative
Negative
```

1 C) Implement advanced deep learning algorithms such as CNN using PyTorch

Code:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

# 1. Transform + Load CIFAR-10
transform = transforms.ToTensor()
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)

classes = ['plane','car','bird','cat','deer','dog','frog','horse','ship','truck']

# 2. Simple CNN
class SimpleCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(3, 16, 3, padding=1)  # fewer filters
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(16*16*16, 10)  # 16 filters × 16×16 image

    def forward(self, x):
        x = self.pool(torch.relu(self.conv(x)))
        x = x.view(-1, 16*16*16)
        return self.fc(x)

# 3. Training setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 4. Train
for epoch in range(3):  # fewer epochs
    for images, labels in trainloader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        loss = criterion(model(images), labels)
        loss.backward()
        optimizer.step()
    print(f"Epoch {epoch+1} done")

# 5. Test accuracy
correct, total = 0, 0
with torch.no_grad():
    for images, labels in testloader:
        outputs = model(images.to(device))
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted.cpu() == labels).sum().item()
print(f"Accuracy: {100*correct/total:.2f}%")
```

```
# 6. Show one prediction
images, labels = next(iter(testloader))
output = model(images[0].unsqueeze(0).to(device))
pred = torch.argmax(output, 1).item()

plt.imshow(images[0].permute(1, 2, 0))
plt.title(f"Predicted: {classes[pred]}, Actual: {classes[labels[0]]}")
plt.axis("off")
plt.show()
```

Output:

```
In [4]: %runfile 'C:/Users/Ruchira Itte/.spyder-py3/untitled1.py' --wdir
Reloaded modules: torch.ops, torch.classes
Epoch 1 done
Epoch 2 done
Epoch 3 done
Accuracy: 58.98%
```