ML-WEEK-11

Understand CNN implementation Using scikit-learn/keras/tensorflow/PyTorch on mentioned datasets:

1. Understand the working of CNN for image classification. (Refer to material shared).
2. Illustrate a simple architecture for a basic CNN implementation
3. **Extract and visualize feature maps** from CNN for a single image (color image, grayscale image) without training a model.
4. Implement **CNN** for **image classification** on **subset classes** of CIFAR-10/MNIST/similar datasets/dummy generated dataset
5. Demonstrate and note your observations on improving the performance of classification through learning curves, classifier metrics etc.,

****

Code snippet with Keras/Tensorflow (Extract and visualize feature maps for a single color RGB image of size 150x150)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model

# Step 1: Define a simple CNN model
# We define a simple model to apply convolution operations.
# Without training, the filters will have random initial weights,
# resulting in arbitrary feature maps.
def create_simple_cnn():
    # Input layer expects images of a specific shape (e.g., 150x150x3 for
RGB)
    input_layer = tf.keras.layers.Input(shape=(150, 150, 3),
name='input_image')

    # First Convolutional Layer
    conv1 = tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
activation='relu', name='conv_1')(input_layer)
    # First Max Pooling Layer
    pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2),
name='pool_1')(conv1)

    # Second Convolutional Layer
    conv2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3),
activation='relu', name='conv_2')(pool1)
    # Second Max Pooling Layer
    pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2),
name='pool_2')(conv2)
```

```
    # Create the model
    # We define the model with the input and the specific layer output we
want to visualize
    feature_extractor_model = Model(inputs=input_layer, outputs=[conv1,
conv2])

    return feature_extractor_model

# Step 2: Load and Preprocess a single image
def preprocess_image(image_path):
    # Load the image and resize to the target size
    img = load_img(image_path, target_size=(150, 150))
    # Convert to numpy array
    img_array = img_to_array(img)
    # Expand dimensions to match the input shape required by the model (add
batch dimension)
    img_array = np.expand_dims(img_array, axis=0)
    # Normalize pixel values
    img_array /= 255.0
    return img_array

# Step 3: Extract the feature maps
image_path = 'your_image.jpg' # Replace with your image file path
input_image = preprocess_image(image_path)
feature_extractor = create_simple_cnn()

# Get the feature maps for the input image
# The output will be a list of arrays, one for each specified output layer
conv1_features, conv2_features = feature_extractor.predict(input_image)

# Step 4: Visualize the feature maps
def visualize_feature_maps(feature_maps, layer_name):
    # The feature map shape is (1, height, width, channels)
    # Squeeze the batch dimension
    features = np.squeeze(feature_maps, axis=0)
    num_channels = features.shape[-1]

    print(f"Visualizing feature maps for layer: {layer_name}")
    print(f"Feature map shape: {features.shape}")

    # Plot the first few channels (e.g., first 16)
    plt.figure(figsize=(10, 10))
    for i in range(min(16, num_channels)):
        plt.subplot(4, 4, i + 1)
        # Display the channel (must be normalized for visualization if not
already)
        plt.imshow(features[:, :, i], cmap='viridis') # 'viridis' or 'gray'
colormap can be used
        plt.axis('off')
    plt.show()

visualize_feature_maps(conv1_features, 'conv_1')
visualize_feature_maps(conv2_features, 'conv_2')
```
****

Code snippet with <u>Keras/Tensorflow</u> (CNN image classification on a subset of Cifar-10 dataset)

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# 1. Load data and select subset (e.g., classes 1 and 5)
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
desired_classes = [1, 5]

# 2. Filter data
train_filter = np.isin(y_train, desired_classes).flatten()
x_train_subset, y_train_subset = x_train[train_filter], y_train[train_filter]

test_filter = np.isin(y_test, desired_classes).flatten()
x_test_subset, y_test_subset = x_test[test_filter], y_test[test_filter]

# 3. Preprocess: Normalize
x_train_subset = x_train_subset.astype('float32') / 255.0
x_test_subset = x_test_subset.astype('float32') / 255.0

# 4. Build and Train Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(len(desired_classes), activation='softmax')
])

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

model.fit(x_train_subset, y_train_subset, epochs=5,
          validation_data=(x_test_subset, y_test_subset))
```
****

Code snippet with <u>PyTorch</u> (CNN image classification on a subset of Cifar-10 dataset)

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Subset
import numpy as np

# 1. Load the CIFAR-10 dataset
transform = transforms.ToTensor()
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform)
```

```
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)

# 2. Select a subset of data (e.g., classes 0: airplane and 1: automobile)
desired_classes = [0, 1] # Airplane, Automobile
train_indices = [i for i, label in enumerate(trainset.targets) if label in
desired_classes]
test_indices = [i for i, label in enumerate(testset.targets) if label in
desired_classes]

train_subset = Subset(trainset, train_indices)
test_subset = Subset(testset, test_indices)

# 3. Create DataLoaders for the subsets
batch_size = 64
train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_subset, batch_size=batch_size, shuffle=False)

# 4. Define a simple CNN model (example, a more complex one can be used)
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.fc1 = nn.Linear(32 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, len(desired_classes)) # Output layer
tailored to subset size

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 8 * 8) # Flatten the tensor
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = SimpleCNN()

# 5. Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# 6. Train the model
num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data

        # Adjust labels to be 0 or 1 for the two classes selected
        labels = torch.tensor([desired_classes.index(label) for label in
labels], dtype=torch.long)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print(f'Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}')

print('Finished Training')
```
****

Code snippet with Keras/Tensorflow (CNN image classification on randomly generated dummy image dataset)

```
x_train = np.random.random((100, 32, 32, 3))
y_train = np.random.randint(10, size=(100, 1)) # 10 classes
x_test = np.random.random((20, 32, 32, 3))
y_test = np.random.randint(10, size=(20, 1))
```
****

Code snippet with Keras/Tensorflow (JPEG color to grayscale image conversion)

```
import tensorflow as tf
import matplotlib.pyplot as plt

# Load an image (replace 'path/to/your/image.jpg' with your image path)
# The image should be a 3-channel RGB image (height, width, channels)
image_path = 'path/to/your/image.jpg'
image = tf.io.read_file(image_path)
image = tf.image.decode_jpeg(image, channels=3) # Ensure 3 channels

# Convert to grayscale
grayscale_image = tf.image.rgb_to_grayscale(image)

# Display the original and grayscale images (optional)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Color Image')
plt.axis('off')

plt.subplot(1, 2, 2)
# Use tf.squeeze to remove the last dimension (channel) for correct display
with plt.imshow
plt.imshow(tf.squeeze(grayscale_image, axis=-1), cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')

plt.show()

# The resulting grayscale_image will have a shape of (height, width, 1)
print(grayscale_image.shape)
****************************************************************************
```