Training Day-76 Report:

1. Window Padding

Padding is a technique in convolutional neural networks (CNNs) to control the output size of the convolution operation. It adds extra borders (usually filled with zeros) to the input data to ensure certain properties in the output dimensions.

Types of Padding:

1. **Valid Padding (No Padding):**
    - No extra padding is added.
    - Output size decreases with each convolution.
    - Formula for output size:
      Output Size=Input Size−Kernel SizeStride+1\text{Output Size} = \frac{\text{Input Size} - \text{Kernel Size}}{\text{Stride}} + 1
2. **Same Padding:**
    - Padding is added to ensure the output size is the same as the input size.
    - Formula for padding size: P=⌈Kernel Size−12⌉P = \left\lceil \frac{\text{Kernel Size} - 1}{2} \right\rceil

Example:

For a 5×55 \times 5 input and a 3×33 \times 3 kernel with stride 1:

- **Valid Padding**: 3×33 \times 3 output.
- **Same Padding**: 5×55 \times 5 output.

TensorFlow Example:

```
import tensorflow as tf

# Dummy input data
input_data = tf.constant([[[[1], [2], [3]], [[4], [5], [6]], [[7], [8], [9]]]], dtype=tf.float32)

# Convolution layer with 'valid' padding
conv_valid = tf.keras.layers.Conv2D(1, (3, 3), strides=(1, 1), padding='valid')
output_valid = conv_valid(input_data)

# Convolution layer with 'same' padding
conv_same = tf.keras.layers.Conv2D(1, (3, 3), strides=(1, 1), padding='same')
output_same = conv_same(input_data)

print("Valid Padding Output Shape:", output_valid.shape)
```

```
print("Same Padding Output Shape:", output_same.shape)
```

2. Image Classification Using CNN

Image classification involves assigning a label to an image based on its content.

Steps to Implement Image Classification:

1. **Load and Preprocess Data**: Use datasets like CIFAR-10, MNIST, or custom datasets.
2. **Define a CNN Model**: Use convolutional, pooling, and dense layers.
3. **Train the Model**: Use labeled data to optimize the model's performance.
4. **Evaluate and Predict**: Test the model on unseen data.

Example Implementation:

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

# Load and preprocess MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0  # Normalize and reshape
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0

# Define CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # Output layer for 10 classes
])
```

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


# Train the model
model.fit(x_train, y_train, epochs=10, validation_split=0.1)


# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy}")


# Predict on new data
predictions = model.predict(x_test[:5])
print("Predictions:", tf.argmax(predictions, axis=1).numpy())
```

Key Points:

- **Padding** ensures control over output dimensions, especially in deep networks.
- **Image Classification** involves training CNNs to recognize patterns in images and assign labels.
- Use libraries like TensorFlow and datasets like MNIST for practice.