Training Day-77 Report:

1. Convolutional Neural Networks (CNNs)

CNNs are primarily used for processing grid-like data such as images. They excel at feature extraction, learning hierarchical patterns like edges, textures, and shapes, making them ideal for tasks like image classification, object detection, and more.

Key Components of CNNs:

1. **Convolutional Layer**:
    - Applies filters (kernels) to the input data.
    - Outputs feature maps highlighting specific features.
    - Formula for output size: O=I−K+2PS+1O = \frac{I - K + 2P}{S} + 1 Where:
        - OO: Output size
        - II: Input size
        - KK: Kernel size
        - PP: Padding
        - SS: Stride
2. **Activation Function**:
    - Introduces non-linearity, e.g., ReLU (max⁡(0,x)\max(0, x)).
3. **Pooling Layer**:
    - Reduces spatial dimensions using techniques like MaxPooling.
4. **Fully Connected Layer**:
    - Connects all neurons from the previous layer for classification or regression.

Example CNN in TensorFlow:

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

# Define a CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
```

```
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # Output for 10 classes
])

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

# Summary
model.summary()
```

2. Recurrent Neural Networks (RNNs)

RNNs are designed for sequential data like time series, text, and audio. Unlike feedforward networks, RNNs retain a "memory" of past inputs, making them effective for handling temporal dependencies.

How RNNs Work:

1. **Input Sequence**: Data is processed one time step at a time.
2. **Hidden State**: The hidden state captures information about previous time steps, allowing the network to maintain a memory of past events.
3. **Output**: Each time step generates an output based on the current input and the hidden state.

Challenges:

- **Vanishing Gradient Problem**: Difficulty in learning long-term dependencies.
- **Solutions**: Use variants like Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU).

Example RNN in TensorFlow:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import SimpleRNN, Dense

# Dummy sequential data
X = tf.random.normal([100, 10, 8])  # 100 samples, 10 timesteps, 8 features
y = tf.random.uniform([100], maxval=2, dtype=tf.int32)  # Binary labels

# Define an RNN model
model = Sequential([
    SimpleRNN(32, activation='tanh', input_shape=(10, 8)),
    Dense(1, activation='sigmoid')
])

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

# Train the model
model.fit(X, y, epochs=5, batch_size=16)

# Evaluate the model
loss, accuracy = model.evaluate(X, y)
print(f"Accuracy: {accuracy}")
```

Key Differences: CNNs vs. RNNs

Feature	CNN	RNN
Input Type	Spatial data (e.g., images)	Sequential data (e.g., text, time series)
Architecture	Filters and pooling layers	Recurrent connections with memory
Use Case	Image classification, object detection	Language modeling, time series prediction

---