These are great questions related to Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Here's a breakdown of the answers:

**CNN Questions**

**1. Advantages of CNN over Fully Connected DNN for Image Classification:**

* **Fewer Parameters**: CNNs share weights using filters, reducing the number of parameters compared to fully connected networks.
* **Translation Invariance**: Convolutional layers detect features regardless of their position in the image.
* **Better Generalization**: The weight-sharing mechanism reduces overfitting.
* **Spatial Hierarchy**: Lower layers capture edges and textures, while deeper layers learn high-level representations like object parts.

**2. Parameters and Memory Calculation for the Given CNN**

**Given:**

* 3 convolutional layers, each using **3×3 kernels**, stride = 2, and "same" padding.
* Number of filters: **100 → 200 → 400**.
* Input size: **200 × 300 × 3 (RGB)**.

Each convolutional layer has parameters calculated as:

(Filter Width×Filter Height×Channels+1)×Number of Filters(\text{Filter Width} \times \text{Filter Height} \times \text{Channels} + 1) \times \text{Number of Filters}

where +1 accounts for the bias term.

**Layer 1: (3×3×3 + 1) × 100 = 2800**  
**Layer 2: (3×3×100 + 1) × 200 = 180200**  
**Layer 3: (3×3×200 + 1) × 400 = 721200**  
**Total Parameters = 2800 + 180200 + 721200 = 904200**

**Memory Requirement:**  
Each parameter is stored as a **32-bit float (4 bytes)**:

904200×4=3.62 MB904200 \times 4 = 3.62 \text{ MB}

For a mini-batch of 50 images, we must account for activations and gradients, requiring **significantly more RAM** depending on batch size, optimizer, and layer activations.

**3. If GPU Runs Out of Memory, Solutions:**

1. Reduce batch size.
2. Use mixed precision (16-bit floats instead of 32-bit).
3. Apply gradient checkpointing.
4. Reduce model size (fewer filters, layers, or use depthwise separable convolutions).
5. Use a memory-efficient framework like TensorFlow’s tf.data pipeline.

**4. Why Use Max Pooling Instead of a Convolutional Layer with the Same Stride?**

* Reduces computational cost.
* Helps retain the most prominent features while reducing noise.
* Introduces translation invariance.

**5. When to Use Local Response Normalization (LRN)?**

* Used in older architectures like **AlexNet** to normalize feature maps across channels.
* Helps improve generalization but is largely replaced by **Batch Normalization**.

**6. Innovations in CNN Architectures:**

* **AlexNet**: ReLU activation, dropout, overlapping max-pooling, LRN.
* **GoogLeNet**: Inception modules, global average pooling.
* **ResNet**: Skip (residual) connections.
* **SENet**: Squeeze-and-excitation blocks for attention.
* **Xception**: Depthwise separable convolutions.

**7. Fully Convolutional Networks (FCNs) & Dense to Convolutional Layer Conversion**

* FCNs replace fully connected layers with 1×1 convolutions.
* A dense layer with **n** units can be converted to a **1×1 convolution** with **n** filters.

**8. Main Technical Challenge in Semantic Segmentation**

* **Preserving spatial resolution** while maintaining computational efficiency.
* Solutions: Use **U-Net, Fully Convolutional Networks (FCNs), or Transformers**.

**9. Build a CNN from Scratch for MNIST**

* Use a simple architecture: **Conv → ReLU → Pool → Dense**.
* Use **data augmentation** and **batch normalization** to improve accuracy.

**10. Transfer Learning for Large Image Classification**

**Steps:**

1. Gather a dataset with **100+ images per class**.
2. Split into training, validation, and test sets.
3. Use tf.data pipeline with **data augmentation** (flipping, rotation, color jittering).
4. Fine-tune a pretrained model (ResNet, EfficientNet) using **transfer learning**.

**RNN Questions**

**1. Applications of Different RNN Architectures:**

* **Sequence-to-sequence RNN**: Machine translation, chatbots, text summarization.
* **Sequence-to-vector RNN**: Sentiment analysis, text classification.
* **Vector-to-sequence RNN**: Image captioning, text generation.

**2. RNN Input & Output Dimensions**

* Inputs: (batch\_size, time\_steps, features), representing **batch size, sequence length, and input features per time step**.
* Outputs depend on whether return\_sequences=True (returns outputs at each time step) or False (only the final output).

**3. Setting return\_sequences=True in Deep RNNs**

* **Sequence-to-sequence RNN**: All RNN layers except the last should have return\_sequences=True.
* **Sequence-to-vector RNN**: Only the final layer should return the last output (return\_sequences=False).

**4. RNN Architecture for Forecasting Next 7 Days in a Time Series**

* **Many-to-many RNN** (sequence-to-sequence model).
* **LSTM/GRU encoder-decoder architecture**.

**5. Main Difficulties in Training RNNs & Solutions**

* **Exploding gradients** → Use gradient clipping.
* **Vanishing gradients** → Use LSTMs/GRUs.
* **Long training time** → Use parallel processing and optimize batch sizes.
* **Memory constraints** → Use truncated backpropagation through time (TBPTT).

**6. Sketch of an LSTM Cell:**

* Contains **forget, input, and output gates** that regulate the flow of information.
* Uses **sigmoid and tanh** activations.

**7. Why Use 1D Convolutions in RNNs?**

* Extracts **local patterns in sequences** before feeding them into an RNN.
* Reduces sequential dependencies, speeding up training.

**8. Neural Network for Video Classification**

* **3D CNN** (spatiotemporal feature extraction).
* **CNN + RNN (LSTM)**: Extracts spatial features with CNN, temporal dependencies with LSTM.
* **Transformers (ViViT, TimeSformer)** for advanced modeling.

**9. Train a Classification Model for SketchRNN Dataset**

* Use **CNN or RNN-based model** with preprocessing.
* Convert stroke-based drawings into **image embeddings or sequential representations**.
* Train with **Adam optimizer and data augmentation**.