ChatGPT

Top 50 General Viva Questions

1. **What is deep learning?**
   Deep learning is a specialized branch of machine learning that uses neural networks with multiple layers (hence "deep") to learn complex features from data [1] [2]. It automatically extracts high-level abstractions from raw data through stacked layers of neurons, enabling tasks like image and speech recognition.

2. **How is deep learning different from traditional machine learning?**
   Deep learning models automatically learn feature representations from raw data, whereas traditional ML often relies on manual feature engineering [3]. Deep nets (3+ layers) can learn hierarchical, non-linear patterns with large datasets [2], while conventional models (e.g. linear regression, decision trees) may need handcrafted inputs and have less representational power.

3. **What is a neural network?**
   A neural network is a computational model inspired by the brain, composed of interconnected layers of nodes (neurons) that transform inputs to outputs through weighted connections and activations [4] [5]. It consists of an input layer, one or more hidden layers, and an output layer; each neuron computes a weighted sum of its inputs plus a bias, then applies a (nonlinear) activation.

4. **What are perceptrons and multilayer perceptrons (MLPs)?**
   A **perceptron** is the simplest neuron model that computes a linear weighted sum of inputs and applies a step (or sigmoid) activation to perform binary classification [6]. A **multilayer perceptron (MLP)** is a feedforward network with one or more hidden layers of perceptrons, allowing it to learn non-linear decision boundaries beyond single-layer (linear) limitations [6].

5. **Why do we use activation functions?**
   Activation functions introduce non-linearity into a neural network, enabling it to learn complex patterns [7]. Without activations, a multi-layer network would collapse to a linear model. For example, Keras explains that activations allow each layer to transform data in a non-linear way [7].

6. **What are common activation functions and their use cases (sigmoid, tanh, ReLU)?**

7. **Sigmoid:** Outputs values in (0,1), often used for binary classification outputs [8].
8. **Tanh:** Outputs in (–1,1), zero-centered, useful in hidden layers to model negative inputs [9].

9. **ReLU (Rectified Linear Unit):** Outputs max(0, x), inducing sparsity and efficient computation [10]. It is widely used in hidden layers because it mitigates vanishing gradients and speeds up training. (ReLU is zero for negative inputs, which can help networks converge faster [10].)

10. **What is a loss function? Give examples.**
    A loss (cost) function measures the discrepancy between the model's predictions and the true targets. For regression tasks, common losses include Mean Squared Error (MSE). For classification, the cross-entropy loss (binary or categorical) is typically used [11]. For example, in

multi-class problems one often uses categorical cross-entropy to penalize incorrect probabilities [11].

11. **What is backpropagation?**
    Backpropagation is the algorithm used to train neural networks. It computes the gradient of the loss function with respect to each weight by the chain rule, propagating the error backward through the network. Modern DL frameworks (e.g. TensorFlow, Theano) perform this automatically via automatic differentiation [12]. The computed gradients are then used by an optimizer (like SGD) to update the weights.

12. **Explain gradient descent and its variants (batch, stochastic, mini-batch).**
    Gradient descent iteratively updates model weights opposite to the loss gradient to minimize error. In **batch GD**, the gradients are computed using the entire training set (precise but slow for large data). **Stochastic GD (SGD)** updates weights using one example at a time (noisy but faster convergence). **Mini-batch GD** uses a small random subset each iteration, balancing convergence stability and efficiency. These variants trade off convergence speed and memory usage.

13. **What are learning rate and momentum?**
    The learning rate is a hyperparameter controlling the step size at each gradient update: too large may overshoot minima, too small slows convergence. Momentum is an extension that accumulates past gradients to smooth updates and accelerate training in relevant directions. Specifically, momentum adds a fraction of the previous update to the current one, helping to damp oscillations and speed up convergence, especially in ravines.

14. **What are epochs, batches, and iterations?**
    An **epoch** is one full pass through the entire training dataset. A **batch** (or mini-batch) is a subset of the data used to compute a single gradient update. An **iteration** often refers to one weight update step; if the data is divided into batches of size *B*, then each epoch contains (number of samples / *B*) iterations.

15. **What are vanishing and exploding gradients?**
    In very deep networks, the gradient of the loss can **vanish** (become extremely small) or **explode** (become very large) as it is backpropagated through many layers. Vanishing gradients make early layers learn very slowly (tiny updates), whereas exploding gradients cause extremely large weight updates and instability. Both issues hinder effective training of deep nets.

16. **How can vanishing/exploding gradients be mitigated?**
    Using non-saturating activations like ReLU, proper weight initialization (e.g. Xavier/He), batch normalization, and architectural techniques like residual (skip) connections can help. For example, batch normalization normalizes each layer's inputs to have zero mean and unit variance, allowing higher learning rates and more stable gradients [13] [14]. This reduces internal covariate shift and mitigates vanishing/exploding gradients. Gradient clipping (capping gradient magnitudes) is also used to control exploding gradients.

17. **What is overfitting and how to prevent it?**
    Overfitting occurs when a model learns training data too well (including noise) and performs poorly on unseen data [15]. The model fits the training set well but generalizes poorly. Prevention techniques include regularization (L1/L2 penalties), dropout (randomly disabling neurons during training) [16], data augmentation, reducing model complexity, and using early stopping when validation loss stops improving.

18. **What is underfitting?**
    Underfitting happens when the model is too simple to capture the underlying pattern, resulting in poor performance on both training and test data (high bias) [17]. It indicates the model cannot learn the training data. The solution is to increase model complexity (more layers/neurons), train longer, or engineer better features.

19. **What is L1 and L2 regularization?**
    L1 (Lasso) regularization adds the sum of absolute values of weights to the loss, encouraging many weights to become exactly zero (sparse model). L2 (Ridge) adds the sum of squares of weights to the loss, penalizing large weights more heavily and encouraging overall smaller weights. Both prevent overfitting by discouraging overly complex models.

20. **What is dropout?**
    Dropout is a regularization technique where, during training, a random subset of neurons is "dropped out" (set to zero) in each layer at each update. This forces the network to learn redundant representations and prevents co-adaptation of neurons [16]. Dropout reduces overfitting by making the presence of any single neuron unreliable, thus improving generalization.

21. **What is batch normalization and why use it?**
    Batch normalization normalizes the inputs of each layer (or mini-batch) to have zero mean and unit variance [13]. This stabilizes and accelerates training by reducing internal covariate shift. It often allows higher learning rates and acts as a regularizer, improving generalization and reducing sensitivity to initialization or choice of hyperparameters [13] [14].

22. **What is a confusion matrix?**
    A confusion matrix is a table that summarizes the performance of a classification model. Rows typically represent actual (true) classes and columns represent predicted classes. The diagonal entries count correct predictions (true positives and true negatives), while off-diagonal entries count misclassifications (false positives and false negatives). It helps visualize where the model is making errors.

23. **Define precision, recall, and F1-score.**
    Precision is the ratio of true positives to all predicted positives (TP/(TP+FP)) [18]. It measures how many predicted positives are correct. **Recall** (sensitivity) is the ratio of true positives to all actual positives (TP/(TP+FN)). It measures how many actual positives were identified. **F1-score** is the harmonic mean of precision and recall, balancing both (2·precision·recall/(precision+recall)).

24. **What is softmax?**
    Softmax is an activation function for multi-class classification. It exponentiates each output and divides by the sum of exponentials, producing a probability distribution over classes (outputs in [0,1] that sum to 1) [19]. It is typically applied to the output layer of a classifier with one neuron per class.

25. **What is cross-entropy loss?**
    Cross-entropy loss measures the distance between two probability distributions (the true labels and the predicted probabilities). For classification, it penalizes confident wrong predictions heavily. In Keras, `categorical_crossentropy` is used for multi-class, and `binary_crossentropy` for binary tasks. (In Assignment 2 we used sparse categorical crossentropy for the multi-class digit problem [11].)

26. **Explain the bias–variance tradeoff.**
    The bias–variance tradeoff describes how a model's complexity affects its errors. High bias (underfitting) means the model is too simple and makes large errors on training and test. High variance (overfitting) means the model is too complex and captures noise, performing well on training but poorly on test. The goal is to find a balance where both bias and variance are low, minimizing total error.

27. **What is an epoch in training?**
    An epoch is one full pass through the entire training dataset. After each epoch, the model has seen all examples once (in batches) and the weights have been updated that many times. Multiple epochs are run until convergence or stopping criteria are met.

28. **What are common optimizers and how do they differ?**
    Common optimizers include **SGD** (Stochastic Gradient Descent), which updates weights using the gradient of the loss; **SGD with momentum**, which adds a fraction of the previous update to accelerate learning; **RMSProp**, which scales learning rates by a moving average of squared gradients; and **Adam**, which combines momentum and adaptive learning rates for each parameter. Adam is often a good default as it adapts learning rates and typically converges faster.

29. **What is Xavier/He initialization?**
    Xavier (Glorot) and He initializations are schemes for setting the initial weights of a network to maintain stable signal variance through layers. Xavier initialization scales weights by the average of input/output layer sizes (good for tanh activations), while He initialization scales by the number of incoming connections (good for ReLU). They help prevent vanishing/exploding activations at start.

30. **Why is GPU/TPU used in deep learning?**
    Deep learning models perform many large matrix multiplications and operations in parallel. GPUs and TPUs are specialized hardware that execute thousands of operations simultaneously, greatly accelerating training and inference. For example, convolutional layers in CNNs are computationally heavy, so GPUs are typically used to handle the high parallel workload [20].

31. **What is transfer learning?**
    Transfer learning is reusing a model trained on one task (often with large data) for a new, related task. In practice, one often takes a pretrained model (e.g. ResNet on ImageNet) and fine-tunes some layers on a new dataset. This leverages learned features and reduces the amount of data/time needed for the new problem.

32. **What is a pre-trained model?**
    A pre-trained model is a network that has already been trained on a large dataset. Its learned parameters (weights) can be reused as a starting point for other tasks. Pre-trained models are especially useful when labeled data for the new task is scarce, as they provide generalized feature extractors out of the box.

33. **What is data augmentation?**
    Data augmentation artificially increases training data diversity by applying random transformations (rotations, flips, crops, color jitter, etc.) to input data. In image tasks, it helps prevent overfitting by exposing the model to varied versions of the same data.

34. **What is cross-validation (e.g. k-fold CV)?**
    Cross-validation is a method to estimate generalization: the data is split into *k* subsets (folds). The model is trained *k* times, each time using a different fold as the validation set and the rest as training. This yields *k* performance estimates, reducing variance in evaluation and helping select hyperparameters.

35. **What is the universal approximation theorem?**
    This theorem states that a feedforward neural network with at least one hidden layer (and a suitable activation) can approximate any continuous function on compact domains, given enough neurons. It explains why even simple MLPs can, in principle, model complex relationships if sufficiently large.

36. **Why do we need non-linear activation functions?**
    Non-linear activations allow a network to learn non-linear mappings. If all activations were linear, any number of layers would collapse to an equivalent single linear transformation. Non-linearities (like ReLU) enable the network to model complex, non-linear relationships in the data.

37. **What is gradient clipping?**
    Gradient clipping is a technique to prevent exploding gradients by capping their values. When a computed gradient exceeds a threshold, it is scaled back. This keeps weight updates within a reasonable range, improving stability in training deep or recurrent networks.

38. **What is weight sharing in CNNs?**
    In CNNs, weight sharing means the same filter (kernel weights) is used across all spatial locations of the input image. This greatly reduces the number of parameters and enforces that the feature detector learns a pattern that is position-invariant [21].

39. **Why use batch normalization in training?**
    Batch normalization standardizes layer inputs, which reduces internal covariate shift and makes training faster and more stable [13]. It often allows higher learning rates and reduces the need for other forms of regularization, improving overall model performance and convergence [14].

40. **What is dropout's effect on model training?**
    Dropout makes the network more robust by preventing reliance on any single neuron. Each update, it randomly disables neurons, forcing the network to learn redundant, distributed representations. This combats overfitting and helps the model generalize better.

41. **How do you decide the number of layers and neurons?**
    This is typically done by experimentation and considering the complexity of the task: deeper or wider networks can model more complex functions but risk overfitting if data is limited. Practical approaches include starting with a modest architecture and increasing size (or using pre-trained networks) until performance saturates.

42. **What is precision vs recall trade-off?**
    Improving precision (fewer false positives) often lowers recall (more false negatives), and vice versa. The trade-off depends on the application: for medical diagnosis or fraud detection you might prioritize high recall (catch as many positives as possible) even if precision suffers. F1-score or a specific threshold can be used to balance them.

43. **What is fine-tuning a model?**
    Fine-tuning is taking a pre-trained model and continuing to train it (often at a lower learning rate) on new data. Typically, earlier layers (which capture general features) are frozen and later layers are retrained. Fine-tuning adapts the model's high-level features to the new task.

44. **What is the role of a softmax layer?**
    A softmax layer converts raw output scores (logits) into probabilities for each class, ensuring they sum to 1. This is essential for multi-class classification so that the outputs can be interpreted as confidence values for each class [19].

45. **Why normalize input data (feature scaling)?**
    Normalizing input features (e.g. to [0,1] or zero mean/unit variance) ensures that all inputs contribute similarly to the gradients and avoids numerical issues. It speeds up convergence and leads to more stable training, since features on vastly different scales can hinder learning.

46. **What are common pitfalls in DL training?**
    Overfitting, vanishing/exploding gradients, choosing bad hyperparameters (learning rate too high/low), insufficient data, and imbalanced classes are typical issues. Proper validation, regularization, normalization, and algorithmic choices (like appropriate activations) help mitigate them.

47. **What is an epoch vs iteration vs batch?**
    (Repeated from Q11) An epoch is one pass through all training examples. A batch (mini-batch) is a subset of data used to compute an update. An iteration usually refers to one weight update (i.e. processing one batch). E.g., with 1000 samples and batch size 100, each epoch has 10 iterations.

48. **What is the difference between classification and regression outputs?**
    Classification outputs are probabilities or class labels (often via softmax/sigmoid), whereas regression outputs are continuous values (linear output). Losses differ: classification often uses cross-entropy, regression uses MSE/MAE.

49. **How do you prevent overfitting aside from dropout?**
    Other methods include L1/L2 weight regularization, early stopping on a validation set, data augmentation (especially for images), reducing network size, and using more data if possible.

50. **Why do deep networks require large datasets?**
    Deep nets have many parameters and can easily memorize data. Large datasets are needed to provide enough examples to learn robust features and avoid overfitting. In practice, techniques like transfer learning or data augmentation are used when data is scarce.

51. **What is 1D vs 2D vs 3D convolution?**
    1D convolution is applied along one spatial (or temporal) dimension (e.g. time series); 2D convolution is used for images (height × width); 3D convolution extends to three dimensions (e.g. video, or volumetric data) applying filters over depth as well.

52. **What is a recurrent neural network (RNN)?**
    An RNN is a network designed for sequential data, where outputs feed back as inputs to capture temporal dependencies. It maintains a hidden state ("memory") across time steps. (Basic RNNs suffer from vanishing gradients; LSTM/GRU units are common gated variants.)

53. **What is the difference between online learning and batch learning?**
    In batch (offline) learning, the model is trained on the entire dataset at once or in mini-batches repeatedly. In online learning, the model updates incrementally as each new example arrives. Online learning suits streaming data or very large datasets.

Assignment 1: Study of Deep Learning Packages (TensorFlow, Keras, Theano, PyTorch)

1. **What is TensorFlow?**
   TensorFlow is an open-source, end-to-end machine learning platform developed by Google. It provides a comprehensive ecosystem of tools, libraries, and community resources for building and deploying ML models [22]. TensorFlow supports computation on CPUs, GPUs, and TPUs and can run on servers, desktops, mobile devices, and the web.

2. **What is Keras?**
   Keras is a high-level neural network API that runs on top of TensorFlow. It offers a user-friendly interface (Sequential and Functional models) for building and training networks. With eager execution (TensorFlow 2.x), Keras allows intuitive model development and easy debugging [23]. (Originally independent, Keras is now tightly integrated into TensorFlow as `tf.keras`.)

3. **What is PyTorch?**
   PyTorch is an open-source deep learning framework known for its flexibility and dynamic computation graphs [24]. It is built to accelerate the path from research prototyping to production deployment [24]. PyTorch integrates seamlessly with Python and supports automatic differentiation on CPUs, GPUs, and custom hardware.

4. **What is Theano?**
   Theano is an open-source Python library for fast numerical computation that supports efficient operations on multi-dimensional arrays [25]. It was one of the first libraries for deep learning research, using symbolic graph optimization and automatic differentiation to implement backpropagation [12]. Theano was widely used (it underpinned early versions of Keras and other libraries) but is now deprecated (development ceased in 2017 [26]).

5. **Compare TensorFlow, PyTorch, Keras, and Theano.**

6. **TensorFlow:** Comprehensive ML platform with static (TF1.x) or eager (TF2.x) graphs, suited for production. It has many deployment tools (TensorBoard, TF Lite, TFX).
7. **PyTorch:** Uses dynamic "define-by-run" graphs [24], making it flexible for research and quick prototyping. It integrates closely with Python and is popular in academia.
8. **Keras:** A high-level API (now part of TF) for easy model building; backend can be TensorFlow or others. It simplifies model definition with concise code (e.g. Sequential) [23].

9. **Theano:** A low-level library with static symbolic graphs. It offered GPU acceleration and auto-differentiation [25] [12], but it is mostly of historical interest now (superseded by TF and PyTorch).

10. **Which frameworks use static vs dynamic computation graphs?**
    TensorFlow 1.x and Theano build static computation graphs that are compiled before execution, while PyTorch and TF 2.x (with eager mode) use dynamic graphs [24] [23]. In TensorFlow 2.x/

Keras, eager execution is on by default (making model definition more intuitive), whereas PyTorch inherently defines the graph at runtime for each input batch.

11. **What tools and libraries are in the TensorFlow ecosystem?**
    TensorFlow's ecosystem includes high-level APIs like `tf.keras`, TensorBoard for visualization, TensorFlow Hub for pretrained models, TensorFlow Lite for mobile, and TensorFlow Extended (TFX) for production pipelines. It also offers specialized libraries like TF Agents (RL), TF Graphics, and supports wide hardware (Cloud TPUs). Many standard models (Inception, ResNet, etc.) and datasets (MNIST, CIFAR) are available through `tf.keras.applications` and `tf.keras.datasets` in the TF ecosystem.

12. **Can you mention any real-world use cases of TensorFlow?**
    TensorFlow is used widely in industry. For example, Google uses it in products like Google Translate and Gmail's smart features [27]. Other companies (e.g. PayPal for fraud detection, Intel for computer vision) also adopt TensorFlow in their AI pipelines. In research, TensorFlow is used for areas like image recognition and NLP. (The referenced lab mentions case studies like PayPal and Intel.)

13. **What is the Keras ecosystem (KerasTuner, KerasCV, KerasNLP, AutoKeras, Model Optimization)?**
    The Keras ecosystem extends core Keras with specialized libraries:

14. **KerasTuner:** Tool for automated hyperparameter tuning (supports Bayesian optimization, Hyperband, etc.) [28].
15. **KerasCV and KerasNLP:** Collections of common components and models for computer vision and NLP tasks (e.g. layers for detection, language modeling).
16. **AutoKeras:** An AutoML library that automatically searches for model architectures and hyperparameters [29].

17. **Keras Model Optimization Toolkit:** Provides techniques like pruning and quantization for model compression. These tools streamline model development and optimization in their respective domains.

18. **How do you build and train a model using Keras Sequential API?**
    Using Keras Sequential, one stacks layers in order. For example, define a model as `model = tf.keras.Sequential([...])` and add layers (e.g. `Flatten`, `Dense`, etc.) [30]. After defining the architecture, compile the model with an optimizer, loss, and metrics (`model.compile(…)`) [30]. Training is done via `model.fit(x_train, y_train, epochs=N, batch_size=M, validation_split=…)`, which internally divides data into batches and runs forward/backward passes over each epoch.

19. **What steps are involved in a Keras training pipeline?**
    Typically: (a) Define the model architecture (Sequential or Functional API); (b) Compile the model with an optimizer (e.g. SGD, Adam), a loss function (e.g. crossentropy), and evaluation metrics; (c) Prepare the data (normalize, one-hot encode labels); (d) Call `model.fit` to train, specifying epochs, batch size, and optionally validation split; (e) After training, evaluate on test data with `model.evaluate`, and use `model.predict` for inference. Keras handles batching and shuffling internally during `fit()`.

20. **What is PyTorch's tensor and Pyro?**
    In PyTorch, a **Tensor** is a multi-dimensional array (like NumPy's ndarray) on which GPU acceleration is possible. PyTorch Tensors support automatic differentiation. **Pyro** is a probabilistic programming library built on PyTorch by Uber AI [31] [32]. Pyro unifies deep learning and Bayesian modeling, allowing users to define complex probabilistic models with neural network components. It uses PyTorch's automatic differentiation and stochastic computation graphs to enable scalable variational inference [32].

21. **What is Tesla Autopilot (in context of DL)?**
    Tesla Autopilot is Tesla's advanced driver-assistance system that heavily relies on deep neural networks. It uses multiple convolutional neural networks to process camera images and sensor data for tasks like object detection, lane recognition, and decision making. (Not directly cited, but conceptually, Tesla's autopilot involves running complex neural models onboard vehicles.)

22. **What are some key differences between these frameworks?**

    - **User Level:** Keras (TF2) offers the highest-level, user-friendly API for common tasks. PyTorch is intermediate (Pythonic code), TensorFlow low-level (computation graphs) but improving with eager.
    - **Learning Curve:** Keras is easiest, PyTorch is considered intuitive for Python developers, TensorFlow has a steeper learning curve.
    - **Computation Graph:** PyTorch dynamic vs TF static graph (pre-TF2). Static graphs (TF1.x/ Theano) can be faster in production after compilation, but dynamic graphs (PyTorch, TF2) are easier for debugging.
    - **Ecosystem:** TensorFlow has a broader ecosystem (TensorBoard, TF Lite, etc.), PyTorch has strong research support and libraries (torchvision, torchaudio, Hugging Face Transformers). Theano is largely legacy at this point.

23. **How do you install these libraries on Ubuntu?**
    You can install them via `pip` or `conda`. For example:

    - **TensorFlow:** `pip install tensorflow` (or `tensorflow-gpu` for GPU support).
    - **Keras:** Included with TF 2.x (`pip install tensorflow`), or standalone via `pip install keras`.
    - **PyTorch:** Use the official instructions at pytorch.org, e.g., `pip install torch torchvision torchaudio` (specifying CUDA version if needed).
    - **Theano:** `pip install Theano` (though it is deprecated).
      Ensure CUDA and cuDNN are installed for GPU versions, and use virtual environments to manage dependencies.

Assignment 2: Feedforward Neural Network with Keras and TensorFlow

1. **What is a feedforward neural network (FFNN)?**
   A feedforward neural network (also called a multilayer perceptron) is a network where data flows in one direction from input to output with no cycles. It consists of an input layer, one or more hidden layers, and an output layer; each layer is fully connected to the next. In our lab, we used a simple FFNN with one hidden dense layer to classify handwritten digits.

2. **What data is used in this assignment?**
   We used the MNIST dataset of handwritten digits, which has 70,000 28×28 grayscale images of digits 0–9 [33]. MNIST is a standard benchmark for classification and is easy to handle, making it ideal for learning feedforward network implementation.

3. **How is the data preprocessed?**
   The image pixel values (0–255) are normalized to [0,1] by dividing by 255.0 to improve numerical stability [34]. The labels are converted to categorical form (one-hot vectors) using Keras's `to_categorical()` utility [35]. (In code we used `sparse_categorical_crossentropy` loss, which allows integer labels, but one-hot encoding is another common approach.)

4. **What layers are in the model and why?**
   We used a `Flatten` layer to convert each 28×28 image into a 1D vector [36]. Then a hidden `Dense` layer with 64 units and ReLU activation introduces non-linearity. Finally, an output `Dense` layer with 10 units (for 10 classes) and softmax activation produces class probabilities. The flatten layer bridges the image input to fully-connected layers, the hidden layer learns abstract features, and the softmax output provides the classification probabilities.

5. **Why use ReLU and softmax activations?**
   ReLU (`max(0,x)`) is used in hidden layers because it is simple and helps mitigate the vanishing gradient problem, leading to faster convergence [10]. The softmax activation in the output layer normalizes the 10 output scores into a probability distribution (summing to 1) across classes [19], which is necessary for multiclass classification.

6. **What loss function and optimizer are used?**
   We use **sparse categorical crossentropy** as the loss function (suitable for multi-class classification with integer labels) and **Stochastic Gradient Descent (SGD)** as the optimizer. In our example, SGD was configured with a learning rate of 0.01 and momentum 0.9. The compile step in Keras is: `model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])` [37].

7. **What is the purpose of one-hot encoding (to_categorical)?**
   One-hot encoding transforms integer class labels into binary vectors where the index of the class is set to 1. This format is required for categorical crossentropy loss if using `categorical_crossentropy`. (In our code, we used `sparse_categorical_crossentropy`, which can take integer labels directly.) As [77] notes, Keras provides `to_categorical()` to perform this conversion [35].

8. **How do you train the model in Keras?**
   The `model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=…)` function trains the network. It runs for a specified number of epochs, updating weights on each batch of data. Keras automatically handles batching and shuffling. In our code, `model.fit()` was used to train for 10 epochs, and it returned a history object containing loss and accuracy for each epoch [38].

9. **What does `model.predict()` output?**
   `model.predict()` returns the raw output of the network before applying argmax. For our 10-class model, it returns a 10-dimensional probability vector for each input sample. Each element is the model's confidence for one digit. We then take `argmax` of this vector to determine the predicted class [39].

10. **How do you evaluate model performance?**
    We measure accuracy on the test set. In Keras, `model.evaluate(x_test, y_test)` computes the loss and accuracy over the test data [40]. In addition, we can generate predictions on the test set and compare to true labels (e.g. via a confusion matrix or using `accuracy_score` from sklearn). In our example, we also visualized some test predictions to qualitatively inspect performance.

11. **What is an epoch and batch size?**
    An **epoch** is one complete pass through the entire training dataset. **Batch size** is the number of samples processed before the model's internal parameters are updated. For instance, with 60,000 training examples and batch size 32, each epoch consists of 60,000/32 ≈ 1875 iterations. The batch size controls memory usage and gradient estimate stability.

12. **How is the model compiled and trained in code?**
    In code, we first define the network (Sequential model with Flatten and Dense layers) [36], then compile it:

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

    Finally, we train with `model.fit(X_train, y_train, epochs=10, batch_size=32)`. This matches the steps described in the lab material [41] [37].

13. **What is a confusion matrix?**
    A confusion matrix for multi-class classification is a 10×10 table (for 10 digits) where each row represents the true class and each column the predicted class. The entry at (i,j) counts how many samples of true class i were predicted as class j. The diagonal entries are correctly classified counts. This helps identify which digits are commonly confused.

14. **How to improve generalization of this model?**
    Possible improvements include adding regularization (e.g. dropout between layers), increasing the number of neurons or layers, using data augmentation on images, or using a different optimizer (like Adam) with learning rate tuning. Batch normalization could stabilize training. Each of these can help the model better generalize to unseen data.

15. **Why use Keras/TensorFlow for this task?**
    Keras/TensorFlow provides high-level abstractions that make constructing and training neural networks straightforward. Functions like `Sequential`, `Dense`, `compile`, `fit`, `predict` greatly simplify implementation. Also, TensorFlow offers fast GPU execution and a rich library of tools and pretrained models, which is ideal for deep learning tasks.

Assignment 3: Image Classification using CNN

1. **What is a Convolutional Neural Network (CNN)?**
    A CNN is a neural network designed for grid-like data (e.g. images). It uses convolutional layers with learnable filters that scan across the input to detect local features, followed by pooling layers to downsample. CNNs typically consist of alternating convolution and pooling layers,

concluding with fully connected layers [42] [43]. They excel at image tasks by exploiting spatial structure.

2. **Why do we reshape images to (28, 28, 1)?**
   Keras convolution layers expect a 4D tensor of shape (samples, height, width, channels). Our MNIST images are 28×28 pixels with 1 color channel (grayscale). We reshape to add the channel dimension (28,28,1) so that `Conv2D` knows the input has one channel. This allows the conv layer to process the image correctly.

3. **What layers are in the CNN model and why?**
   Our CNN has:

4. **Conv2D(32, (3×3), activation=ReLU):** 32 filters of size 3×3 slide over input to extract low-level features (edges, etc.). Weight sharing in conv layers captures patterns irrespective of position [21].
5. **MaxPooling2D(2×2):** Reduces spatial dimensions by taking the maximum in each 2×2 window, reducing parameters and enforcing translational invariance [44].
6. **Flatten:** Converts the final feature maps into a 1D vector for the dense layer [36].
7. **Dense(100, ReLU):** A fully connected layer to combine features and learn higher-level representations.
8. **Dense(10, Softmax):** Output layer with 10 units for the 10 digit classes, with softmax to produce probabilities.

This architecture leverages convolution/pooling to learn spatial features and dense layers for classification.

1. **Why do we normalize pixel values (divide by 255)?**
   Pixel values originally range from 0 to 255. Dividing by 255 rescales them to [0,1], making the inputs small and consistent. This is a common preprocessing step; it does not "leak" test information because it's a simple linear scaling [34]. Normalization helps the network train more efficiently and prevents large input values from saturating activations.

2. **Why use ReLU activations?**
   ReLU is used in the convolutional and dense hidden layers because it is computationally efficient and helps mitigate the vanishing gradient problem [10]. ReLU outputs 0 for negative inputs and identity for positive, which makes gradients flow when active and speeds up convergence. It also induces sparse representations, often improving performance.

3. **Why use Softmax in the output layer?**
   Softmax converts the 10 output scores into a probability distribution over the 10 digit classes [19]. It exponentiates and normalizes the outputs so they sum to 1. This allows interpreting the output as the model's confidence for each digit, enabling standard multi-class classification loss (cross-entropy).

4. **What optimizer is used and why?**
   We used **Stochastic Gradient Descent (SGD)** with a learning rate of 0.01 and momentum 0.9 (as set in `SGD(learning_rate=0.01, momentum=0.9)` [45]). Momentum helps accelerate convergence by smoothing the updates (using past gradients), often leading to faster training than plain SGD.

5. **What is MaxPooling and why is it used?**
   MaxPooling2D downsamples feature maps by taking the maximum value in each non-overlapping 2×2 window [44]. This reduces the spatial size of the representation, lowering the number of parameters and computations. It also provides translational invariance by keeping only the most prominent feature in each region. (Max pooling is preferred over average pooling when sharp features matter.)

6. **What does `model.summary()` show?**
   `model.summary()` prints a table of the layers, their output shapes, and the number of parameters. It shows each layer's name, output dimensions, and how many weights (trainable parameters) it contains. For example, it indicates how many filters were used in Conv2D and the total trainable parameter count, helping us verify the model architecture and size.

7. **Why do we flatten after convolution/pooling layers?**
   Convolution and pooling layers produce 3D feature maps. A `Flatten` layer collapses these into a 1D vector so it can be fed into the dense (fully connected) layers [36]. Dense layers require flat input, so flattening bridges the convolutional feature extractor and the classifier part of the network.

8. **What does `model.fit()` do and what are epochs/batch size?**
   `model.fit(X_train, y_train, epochs=10, batch_size=32)` trains the CNN on the training data. `epochs=10` means the model sees the entire training set 10 times. `batch_size=32` means the gradient is computed and weights updated after every 32 samples. Training progress (loss and accuracy per epoch) is returned in a history object (see [77†L278-L284] for an example of plotting training history).

9. **What does `model.predict()` do here?**
   After training, `model.predict()` takes input data and outputs the model's predictions. For example, `model.predict(image.reshape(1,28,28,1))` outputs a 10-dimensional vector of probabilities for that single image (one-hot style confidences for each digit) [39]. We used `argmax` on this output to pick the most likely digit.

10. **How is test accuracy measured?**
    We use `model.evaluate(X_test, y_test)` to compute the loss and accuracy on the test set [40]. This function returns, e.g., `[loss, accuracy]`. Alternatively, we can predict on `X_test`, compare to true labels using `sklearn.metrics.accuracy_score(y_test, predictions)`. In our code, we computed predictions with `np.argmax(model.predict(X_test), axis=-1)` and used `accuracy_score`.

11. **What accuracy did the model achieve?**
    In this run, the test accuracy was around 98% (as shown in the output). This means the CNN correctly classified about 98% of the MNIST test images. (Typical CNNs can achieve around 99% accuracy on MNIST with more epochs or deeper models.)

12. **How could you improve this CNN?**
    Possible improvements include: increasing the number of filters or adding more convolutional layers, adding dropout between layers to reduce overfitting, using a more sophisticated optimizer (e.g. Adam), or using data augmentation (slight rotations/shifts of images). Batch normalization layers could also stabilize training. Each change should be validated on a held-out set.

Assignment 4: Anomaly Detection using Autoencoder

1. **What is an autoencoder?**
   An autoencoder is a neural network that learns to reconstruct its input. It consists of an **encoder** that maps input data to a lower-dimensional code (latent representation) and a **decoder** that reconstructs the original input from this code [46]. By training to minimize reconstruction error, the autoencoder learns a compressed representation of the "normal" data.

2. **How are autoencoders used for anomaly detection?**
   Train the autoencoder on normal (non-fraud) data so it learns the typical patterns. When fed anomalous data (fraudulent), its reconstruction error will be higher because it hasn't learned those patterns. Thus, we set a threshold on reconstruction error: inputs with error above the threshold are flagged as anomalies. As noted in the fraud-detection handbook, "normal" data tends to have low reconstruction error while outliers have higher error [47].

3. **What dataset is used and how is it prepared?**
   The assignment uses the Credit Card Fraud dataset, which contains many normal transactions and very few fraudulent ones (imbalanced). We split it into training and test sets and then further separated "normal" and "fraud" cases. Only the normal transactions are used to train the autoencoder (unsupervised), as anomalies are not included in the training of the encoder/decoder.

4. **Why do we scale the data?**
   We apply `StandardScaler` to features like `Time` and `Amount` (zero mean, unit variance), and then scale the transaction data between 0 and 1 (min-max) [48] [49]. Scaling ensures that all input features are on a comparable scale, which helps the network train more effectively. Large variances in input magnitudes can make optimization difficult.

5. **Describe the architecture of the autoencoder.**
   The input layer has size 30 (the number of features) [50]. The encoder compresses this to 14 units (tanh activation, L2-regularized) [51], then to 7 units (ReLU) [52], and finally to 4 units (leaky ReLU) [53]. The decoder is symmetric: it takes the 4-unit code and expands to 7 (ReLU) [54], then 14, and finally back to 30 units (tanh) to reconstruct the input. Dropout layers (20%) are applied after the first encoder layer and the first decoder layer for regularization [55] [54].

6. **Why use different activations (tanh, ReLU, leaky ReLU)?**
   Different activations can help capture various patterns. We use **tanh** on the first encoding layer to bound outputs in [-1,1], **ReLU** on the next layers for sparse activations, and **leaky ReLU** to allow a small gradient even when inputs are negative (avoiding "dead" neurons) [56]. The final output uses tanh to match the scaled input range.

7. **Why include dropout and L2 regularization?**
   Dropout (20%) randomly disables neurons during training [57] [54], which prevents the model from simply memorizing the training data and encourages robustness. L2 regularization on the first dense layer's weights penalizes large weights, also preventing overfitting. Both techniques ensure the autoencoder generalizes well on normal data and doesn't overfit to noise.

8. **What are `ModelCheckpoint` and `EarlyStopping` used for?**
   These are Keras callbacks to improve training. `ModelCheckpoint` saves the model weights

whenever the validation loss improves (here it saves the best model to "autoencoder_fraud.h5") [58]. `EarlyStopping` monitors validation loss and stops training if it doesn't improve after a certain number of epochs (patience=10) [59]. EarlyStopping also restores the best weights so far. Together they prevent over-training and keep the best model.

9. **How is the autoencoder compiled and trained?**
   We compile the autoencoder with **mean squared error (MSE)** as the loss (since we want to minimize reconstruction error) and the **Adam** optimizer [60]. We also include accuracy as a metric (though for reconstruction tasks accuracy is less meaningful). We then train with `autoencoder.fit(normal_train_data, normal_train_data, epochs=50, batch_size=64, validation_data=(test_data, test_data), callbacks=[cp, early_stop])` [61], meaning we train the model to reproduce its input for 50 epochs, using only normal data as both input and target.

10. **Why train only on normal data?**
    The autoencoder is meant to capture the distribution of normal transactions. By training on only non-fraudulent data, it learns to reconstruct those patterns accurately. Fraudulent transactions are "out-of-distribution" for the model, so when they are input, the reconstruction error will be higher. This allows us to detect anomalies without having trained on them.

11. **What do the training and validation loss curves show?**
    We plot the MSE loss for training (on normal data) and validation (on mixed or all test data) over epochs [62]. Typically, we expect training loss to decrease. If validation loss stops improving or diverges, it may indicate overfitting. In our result, both losses decreased and leveled off, showing the autoencoder was learning and then converging.

12. **How do we compute reconstruction error and threshold?**
    After training, we predict on the test data to get reconstructed outputs. We compute the MSE per example: `mse = np.mean((test_data - test_x_predictions)**2, axis=1)` [63]. This `Reconstruction_error` is high for anomalies. We then analyze the error distribution for normal vs fraud to choose a threshold. For example, we set a threshold of 52 and label examples with error >52 as fraud.

13. **What does the reconstruction error plot show?**
    We plot each data point's reconstruction error, distinguishing true normal vs fraud [64]. Most normal points have low error (blue points), while fraud points often exceed the red threshold line. This visualization confirms that fraud cases generally have larger errors than normal cases.

14. **How is the confusion matrix used, and what were the results?**
    Using the chosen threshold, we predict each test example as fraud (1) or normal (0). The confusion matrix compares these predictions to the true labels [65]. In our case (threshold=52), the confusion matrix indicated most frauds were detected (high recall) but with some false positives. The exact counts can be read off the heatmap; this informs us of true/false positives/negatives.

15. **What are accuracy, precision, and recall in this context?**

    - **Accuracy:** (TP+TN)/(all samples). In our output, accuracy was ~0.99. This is high because the data is imbalanced (many more normal than fraud).

- **Recall (sensitivity):** TP/(TP+FN) – the fraction of actual frauds correctly identified. Here recall was 1.0, meaning all frauds were caught at the chosen threshold.
- **Precision:** TP/(TP+FP) – the fraction of predicted frauds that were true fraud. Precision was ~0.33, meaning many predicted frauds were actually normal. The formulas are standard (precision = TP/(TP+FP)) [18] . A trade-off exists: raising the threshold would increase precision but lower recall.

Each answer above is aligned with the practical steps and outcomes of the assignments, focusing on concepts and implementations rather than theory alone.

Sources: Authoritative references on deep learning concepts and the provided lab materials [1] [15] [13] [7] [42] [22] [24] [25] [31] [33] [34] [46] [47] [18] [37] [36] [60] provide the factual basis for these answers.

---

[1] [2] [3] [4] What is Deep Learning? A Tutorial for Beginners | DataCamp
https://www.datacamp.com/tutorial/tutorial-deep-learning-tutorial

[5] [20] [21] [42] [43] [44] What are Convolutional Neural Networks? | IBM
https://www.ibm.com/think/topics/convolutional-neural-networks

[6] Perceptron - Wikipedia
https://en.wikipedia.org/wiki/Perceptron

[7] [8] [9] [10] Activation functions in Neural Networks - GeeksforGeeks
https://www.geeksforgeeks.org/machine-learning/activation-functions-neural-networks/

[11] [19] [30] [35] [36] [37] [38] [39] [40] [41] [56] Assignment_2.pdf
file://file_000000001ae072069327b2ef81ea7f2f

[12] [25] [26] Theano Definition | DeepAI
https://deepai.org/machine-learning-glossary-and-terms/theano

[13] [14] Batch normalization - Wikipedia
https://en.wikipedia.org/wiki/Batch_normalization

[15] [16] [17] Day 49: Overfitting and Underfitting in DL — Regularization Techniques | by Adithya Prasad Pandelu | Medium
https://medium.com/@bhatadithya54764118/day-49-overfitting-and-underfitting-in-dl-regularization-techniques-8ded20baa3d6

[18] precision_score — scikit-learn 1.7.2 documentation
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

[22] [23] [27] TensorFlow | Google Open Source Projects
https://opensource.google/projects/tensorflow

[24] PyTorch – PyTorch
https://pytorch.org/projects/pytorch/

[28] [29] The Keras ecosystem
https://keras.io/getting_started/ecosystem/

[31] [32] Uber Open Sources Pyro, a Deep Probabilistic Programming Language | Uber Blog
https://www.uber.com/en-IN/blog/pyro/

[33] 3.3. The MNIST Dataset — conx 3.7.9 documentation
https://conx.readthedocs.io/en/latest/MNIST.html

34 keras - Does normalizing images by dividing by 255 leak information between train and test set? - Stack Overflow
https://stackoverflow.com/questions/55859716/does-normalizing-images-by-dividing-by-255-leak-information-between-train-and-te

45 Assignment No.3.pdf
file://file_0000000072b0720982ba7c979c5faa14

46 47 3. Autoencoders and anomaly detection — Reproducible Machine Learning for Credit Card Fraud detection - Practical handbook
https://fraud-detection-handbook.github.io/fraud-detection-handbook/Chapter_7_DeepLearning/Autoencoders.html

48 49 50 51 52 53 54 55 57 58 59 60 61 62 63 64 65 Assignment-4 Code With Output.pdf
file://file_0000000066b4720694071f390eb635fb