Deep Learning Lab — Complete Viva Guide (examiner's / professor's version)

Nice — I've converted your **uploaded lab manual** into a viva-ready, exam-oriented guide covering **all six experiments** (questions at basic / intermediate / advanced levels), with model answers, theory snippets, diagrams described in text, common mistakes, FAQs, expected outcomes, quick revision sheets, and the extra IR/indexing & feature-extraction material you asked for. This is built from your lab manual (LP-IV Deep Learning).

manual_final_LP-IV _pdf (1) (1)

How to use this guide:

- For each experiment you'll find: *key terms → viva Qs (basic → advanced) → short answer → exam-style detailed answer → diagrams (text) → professor FAQs → common mistakes → real apps → expected outputs → short notes & keywords.*
- Use the **short notes / keywords** and **summary sheet** before the viva for fast recall.

---

1) Experiment 1 — Study of deep learning packages (TensorFlow, Keras, Theano, PyTorch)

Key terms

- TensorFlow, Keras, Theano, PyTorch, eager execution, computational graph, autograd, tensors, sessions (Theano legacy), backend, model serialization, ONNX.

Possible viva questions & answers

Basic

1. **Q:** What is TensorFlow?
   A (short): An open-source ML framework by Google for building & deploying ML/DL models; uses tensors and computational graphs.
   Exam-style: TensorFlow represents computations as dataflow graphs of tensors. With TF 2.x, eager execution is default, enabling imperative style. Common APIs: `tf.keras` for high-level modeling, low-level ops, SavedModel for serialization.

2. **Q:** What is Keras and how does it relate to TensorFlow?
   A (short): Keras is a high-level neural-network API; in TF 2.x `tf.keras` is the recommended high-level API.
   Exam-style: Keras provides `Sequential` and functional APIs to compose layers quickly, simplifying model definition, compilation, training (`.fit`), evaluation (`.evaluate`) and saving.

3. **Q:** What is PyTorch?
   A (short): A dynamic DL library by Facebook with pythonic API and strong autograd for research.
   Exam-style: PyTorch uses dynamic computation graphs (defined at run-time) and a powerful autograd engine (`torch.autograd`). Common modules: `torch.nn`, `torch.optim`, `torch.utils.data`.

4. **Q:** What is Theano?
   A (short): An older symbolic numeric library (precursor to many) used to compile mathematical expressions to C; largely superseded.
   Exam-style: Theano offered symbolic differentiation and graph optimization; now mostly historical but foundational.

Intermediate

5. **Q:** When to choose PyTorch vs TensorFlow?
A (short): PyTorch for research/prototyping (dynamic), TF for production and deployment (TF Serving, TF Lite), though both converge.
Exam-style: Consider ecosystem: TF has `tf.keras`, TF Serving/TF Lite/TF.js for deployment; PyTorch has TorchScript, TorchServe, and preferred by many researchers for dynamic debugging.

6. **Q:** What is eager execution? Why is it useful?
   A: Eager execution runs ops immediately (imperative), making debugging easier and code more Pythonic. TF 2.x uses eager mode by default.

7. **Q:** Explain autograd (automatic differentiation).
   A: System that records operations on tensors and computes gradients by reverse-mode

differentiation (backprop). In PyTorch `requires_grad`, in TF `GradientTape`.

Advanced

8. **Q:** How would you convert a PyTorch model to TensorFlow?
A (short): Export via ONNX (PyTorch → ONNX → TF) or reimplement architecture; ensure matching preprocessing & operators.
Exam-style: Use `torch.onnx.export` to get ONNX; then tools like `onnx-tf` to convert. Watch out for ops not supported by target runtime — validate numerically.

9. **Q:** Explain differences in graph optimization approaches of TF and Theano.
A: Both optimize symbolic graphs; Theano compiled to efficient C, TF has extensive graph transforms, XLA (accelerated linear algebra), and optimizations for distributed execution.

Diagram (text)

- **Stack diagram:** Hardware (CPU/GPU/TPU) ← low-level backends (cuDNN, MKL) ← frameworks (TensorFlow/PyTorch/Theano) ← high-level APIs (Keras) ← Model/Code.

Common professor FAQs

- Differences in speed & memory usage for certain models.
- How to use GPU in each package (`torch.cuda`, `tf.device`).
- Serialization formats (SavedModel, `.pt`, ONNX).

Common mistakes

- Not setting device (GPU/CPU) correctly.
- Mixing eager and graph code poorly (TF1.x habits).
- Forgetting `model.eval()` in PyTorch for inference.

Real-world apps

- TF: Production model serving, mobile inference (TF Lite).

- PyTorch: Research prototypes, NLP research.
- Keras: Fast prototyping, education.
- Theano: Legacy research code.

Expected outcomes

- Correct installation, `import` works, simple test script outputs version and sample results.

Short revision bullets

- `tf.keras` = recommended in TF2; `torch.nn` = model building in PyTorch.
- Autograd = gradients; eager = imperative; SavedModel/.pt = serialization.

---

2) Experiment 2 — Implementing feedforward neural networks (MNIST/CIFAR10 with Keras/TensorFlow)

Key terms

- Feedforward NN, input layer, hidden layer, output layer, activation (sigmoid, ReLU, softmax), loss (cross-entropy), optimizer (SGD), epochs, batch size, overfitting, train/validation/test split.

Viva Qs & answers

Basic

1. **Q:** What is a feedforward neural network?
   A: A network where data moves from input → hidden → output without cycles; layered

MLP.

2. **Q:** Why flatten images for a dense feedforward network?
   A: Dense layers expect 1D vectors; flattening converts 2D image to 1D features.

3. **Q:** Which loss is used for multi-class classification?
   A: Categorical cross-entropy (or sparse categorical cross-entropy if labels are integers).

Intermediate

4. **Q:** Why use SGD? Advantages & disadvantages?
A: SGD (stochastic gradient descent) updates weights per batch — efficient memory, faster convergence for large data; disadvantages: noisy updates, may need careful learning-rate scheduling; Adam is adaptive alternative.

5. **Q:** How to evaluate model performance?
   A: Accuracy, confusion matrix, precision, recall, F1, learning curves (train/val loss & accuracy).

6. **Q:** How to prevent overfitting?
   A: Use dropout, L2 regularization, early stopping, data augmentation, reduce capacity.

Advanced

7. **Q:** Explain weight initialization impacts.
A: Poor initialization causes vanishing/exploding gradients. Use Xavier/Glorot or He initializations depending on activation.

8. **Q:** How to implement custom metric in Keras?
   A: Define function taking (y_true, y_pred) using Keras backend ops, and pass to `compile(metrics=[...])`.

Exam-style answer (example question)

Q: Describe steps to train an MNIST classifier with Keras using SGD and how you would report results.
A:

- Load MNIST `.load_data()`, normalize pixel values to [0,1] (`/255.0`), flatten to 784-d vector.

- One-hot encode labels (`to_categorical`).

- Define Sequential model: Dense(512, activation='relu', input_shape=(784,)), Dropout(0.5), Dense(10, activation='softmax').
- Compile with `optimizer=SGD(learning_rate=0.01, momentum=0.9)`, `loss='categorical_crossentropy'`, `metrics=['accuracy']`.
- Fit with `model.fit(x_train, y_train, validation_split=0.1, epochs=20, batch_size=128, callbacks=[EarlyStopping(...)]`.
- Evaluate on test set: `model.evaluate(x_test, y_test)`.
- Plot training/validation loss & accuracy; compute confusion matrix and classification report (precision/recall/F1).

Diagrams (text)

- **NN block:** [Input nodes] → [Fully-connected hidden layer (weights + biases, ReLU)] → [Dropout] → [Output softmax]
- **Plot:** Loss vs epochs (train loss decreasing, val loss decreasing then increasing if overfit).

FAQs profs ask

- Explain difference between batch size = 1 vs large.
- Why shuffle data?
- How to interpret learning curves.

Common mistakes

- Forgetting to normalize inputs.
- Using softmax+MSE for classification (use cross-entropy).
- Not one-hot encoding when required.

Real-world apps

- Digit recognition, basic classification prototypes, baseline models for image tasks.

Expected outputs

- Training logs, final test accuracy (MNIST ~ 98% for simple MLP with adequate tuning), plots of loss & accuracy.

Short notes & keywords

- Loss = cross-entropy, optimizer = SGD, metrics = accuracy, regularization = dropout/early stop.

---

3) Experiment 3 — Build an image-classification model (CNN pipeline)

Key terms

- Convolutional Neural Network (CNN), convolution, kernel/filter, stride, padding, pooling, batch normalization, data augmentation, softmax.

Viva Qs & answers

Basic

1. **Q:** Why use CNNs for images?
   A: CNNs capture spatial hierarchies via convolutional filters, sharing weights and reducing parameters.

2. **Q:** What does pooling do?
   A: Reduces spatial dimensions, introduces translation invariance, reduces computation.

Intermediate

3. **Q:** Explain `padding='same'`.
A: Adds zeros around input to keep output spatial dims approx equal to input after convolution.

4. **Q:** What is batch normalization? Why use it?
   A: Normalizes layer inputs per mini-batch to stabilize & accelerate training and can act as regularizer.

Advanced

5. **Q:** How to tune hyperparameters for CNN?
A: Grid/random search on learning rate, batch size, number of filters, kernel size, optimizer, regularization; monitor validation metrics & use callbacks.

6. **Q:** Explain transfer learning vs training from scratch.
   A: Transfer learning uses pre-trained convolutional layers (ImageNet) and fine-tunes top layers to new task — faster, requires less data.

Exam-style answer (example)

Q: Explain full pipeline to build CNN model from custom dataset.
A:

- Data: collect and organize into train/val/test folders; apply transforms (resize, center crop, normalization).
- Preprocess: resize images (224×224), convert to tensors, normalize per ImageNet mean/std if using pre-trained model.
- Model: design Conv→ReLU→Pool blocks, then FC classifier; or load pre-trained backbone like ResNet/VGG and replace classifier.
- Train: set data loaders, optimizer (Adam/SGD), learning rate scheduler, loss (cross-entropy), epochs.
- Evaluate: accuracy, confusion matrix; save best checkpoint.
- Visualize: feature maps, misclassified images.

Diagram (text)

- **Data dir structure**: `/datadir/train/classA/...`, `/datadir/valid/classA/...`, `/datadir/test/classA/...`
- **CNN block**: Input → [Conv(k3) → ReLU → Pool] x N → Flatten → Dense → Softmax

FAQs

- Explain data augmentation choices.
- How to handle class imbalance.
- Why normalization values differ with pre-trained models.

Common mistakes

- Using MSE for classification instead of cross-entropy.
- Not normalizing using the same mean/std as pre-trained model.

Real apps

- Image classification for medical imaging, autonomous vehicles, product tagging.

Expected outputs

- Accuracy depending on dataset; training logs; saved model weights; misclassification examples.

---

4) Experiment 4 — Anomaly detection using Autoencoders

Key terms

- Autoencoder, encoder, decoder, latent representation, reconstruction error, novelty detection, MSLE/MSE loss, thresholding.

Viva Qs & answers

Basic

1. **Q:** What is an autoencoder?
   A: A neural network trained to reconstruct its input; has encoder (compress) and decoder (reconstruct).

2. **Q:** How is it used for anomaly detection?
   A: Train autoencoder on normal data only; anomalies reconstruct poorly → high reconstruction error → flagged.

Intermediate

3. **Q:** How to determine threshold for anomaly?
A: Compute reconstruction errors on validation normal data; set threshold = mean + k·std (commonly mean + std) or use percentile (95th).

4. **Q:** Why scale data before training?
   A: Makes learning stable, accelerates convergence; Autoencoder output matched to input scaling.

Advanced

5. **Q:** Differences between undercomplete autoencoder and variational autoencoder for anomaly detection?
A: Undercomplete compresses to lower dimension; VAE learns probabilistic latent variables and can quantify likelihoods — VAEs can model data distribution more explicitly.

6. **Q:** How to evaluate anomaly detection?
   A: Use precision, recall, F1 on labeled anomalies; ROC-AUC; PR-AUC for imbalanced datasets.

Exam-style answer

Q: Describe steps and metrics to build ECG anomaly detector using autoencoders.
A:

- Load ECG.csv, preprocess (handle missing, scale using `MinMaxScaler`).
- Split: choose normal-only training set for novelty detection.
- Define autoencoder: encoder (dense layers reducing to latent dim), decoder (mirror). Activation: ReLU for hidden, linear for output.
- Compile with `optimizer=Adam`, `loss=msle` (as manual uses) or `mse`.

- Train with validation; plot training/validation loss.
- Compute reconstructions, calculate per sample `reconstruction_error = mse(x, x_rec)`.
- Threshold = mean(reconstruction_errors) + std(reconstruction_errors).
- Predict anomaly if error > threshold and evaluate using accuracy/precision/recall/F1.

Diagram (text)

- **Autoencoder:** Input vector (n) → Dense 128 → Dense 64 → Latent (32) → Dense 64 → Dense 128 → Output (n).
- **Plot:** Reconstruction error histogram (normal vs anomaly) — threshold separation.

FAQs

- Choice of loss (MSLE vs MSE) — MSLE reduces relative error effects.
- How to handle temporal sequences (use LSTM autoencoders).

Common mistakes

- Training autoencoder on mixed (normal + anomaly) data → poor detection.
- Choosing threshold too low/high leading to many false positives/negatives.

Real apps

- ECG anomaly detection, fraud detection, manufacturing defect detection, network intrusion detection.

Expected outputs

- Loss curves, threshold, confusion metrics on test set, list of detected anomalies.

Short notes / keywords

- Train on normal, threshold = mean+std, reconstruction error as anomaly score.

---

5) Experiment 5 — Implement Continuous Bag of Words (CBOW) Model

Key terms

- CBOW, Word2Vec, embedding, context window, one-hot encoding, embedding matrix, negative sampling, skip-gram.

Viva Qs & answers

Basic

1. **Q:** What is CBOW?
   A: CBOW predicts a target word from its surrounding context words (averaging context embeddings).

2. **Q:** What does the embedding layer do?
   A: Maps vocabulary indices to dense continuous vectors (word embeddings).

Intermediate

3. **Q:** How to generate training pairs for CBOW?
A: For each sentence, for each target word, collect `window_size` words on left & right as input (context), target is the center word. Pad/ignore edges.

4. **Q:** Loss function used?
   A: Categorical cross-entropy over vocabulary (or negative sampling/ hierarchical softmax to scale).

Advanced

5. **Q:** Compare CBOW and skip-gram.
A: CBOW predicts target from context (faster, smooths over context), Skip-gram predicts context words from a target (better for rare words).

6. **Q:** How to scale to large corpora?
   A: Use negative sampling or hierarchical softmax; use efficient data pipelines (gensim) and subsampling frequent words.

Exam-style answer

Q: Explain how to build CBOW using Keras and how to extract vectors for use in gensim.
A:

- Preprocess text: tokenize, build word index, create sequences.
- For each target, create `context_word` list of length `2*window_size`. Pad to fixed length and one-hot encode target.
- Model: Input (context indices) → Embedding (vocab_size × dim) → Lambda (mean or sum over context embeddings) → Dense(vocab_size, activation='softmax').
- Compile with `categorical_crossentropy` optimizer `adam`. Train.
- Export embedding weights with `model.get_weights()[0]` and write to vectors.txt in word2vec format for gensim `KeyedVectors.load_word2vec_format`.

Diagrams (text)

- **CBOW flow:** [context indices] → Embedding lookups → Average → Dense(softmax) → target word probability.

FAQs

- Handling OOV words, window size effects, use of negative sampling.

Common mistakes

- Not padding contexts uniformly, wrong index-offset causing embedding misalignment, training small corpus with large embedding size.

Real apps

- Semantic similarity, recommendation systems, warm-starting embeddings for downstream NLP tasks.

Expected outputs

- Trained embedding vectors; similarity queries like `most_similar("king")`.

Short notes

- CBOW = average context → predict target, fast on large corpora, negative sampling for scaling.

---

6) Experiment 6 — Object detection using transfer learning of CNN architectures (PyTorch)

Key terms

- Transfer learning, pre-trained models (VGG/ResNet), freezing layers, fine-tuning, data transforms, optimizers, NLLLoss/log-softmax, early stopping.

Viva Qs & answers

Basic

1. **Q:** What is transfer learning?
   A: Re-using a model pre-trained on a large dataset (like ImageNet) for a new task; typically freeze early layers and train classifier head.

2. **Q:** Why freeze weights?
   A: To retain learned low-level features and reduce trainable parameters; speeds training and avoids overfitting when data is limited.

Intermediate

3. **Q:** Explain the PyTorch training loop for transfer learning.
A: For each epoch iterate batches: forward pass `out = model(data)`, compute `loss = criterion(out, targets)`, `loss.backward()`, `optimizer.step()`, zero grads; track train/val accuracy.

4. **Q:** Why use transforms.Normalize with ImageNet mean/std?
A: Pretrained models expect inputs normalized with specific mean/std used during their original training.

Advanced

5. **Q:** How to fine-tune further layers safely?
A: Unfreeze a few top convolutional blocks, lower learning rate for pre-trained layers, higher LR for new classifier; use gradual unfreezing.

6. **Q:** How to handle class imbalance?
A: Weighted loss (`pos_weight`), oversampling minority, focal loss, data augmentation.

Exam-style answer

Q: Describe steps to adapt VGG16 for 100-class classification and training in PyTorch.
A:

- Prepare dataset folders (`train/valid/test`) with splits (50/25/25).
- Transforms: training augmentations with `RandomResizedCrop`, `RandomHorizontalFlip`, normalization; validation with resize & center crop.
- Dataloaders: `DataLoader` with `shuffle=True` for train.
- Load pretrained `vgg16(pretrained=True)`, freeze parameters `param.requires_grad = False`.
- Replace classifier last layer: `model.classifier[6] = nn.Sequential(nn.Linear(n_inputs,256), nn.ReLU(), nn.Dropout(0.4), nn.Linear(256, n_classes), nn.LogSoftmax(dim=1))`.
- Define `criterion = nn.NLLLoss()` and `optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=1e-3)`.

- Train loop with epoch-wise validation, early stopping, and save best model.

Diagram (text)

- **Transfer flow:** Pretrained conv blocks (frozen) → custom FC layers (trainable) → softmax output.

FAQs

- Effects of batch size, learning rate schedulers, GPU memory optimizations.

Common mistakes

- Forgetting to `model.train()` / `model.eval()`.
- Not resetting gradients (`optimizer.zero_grad()`).

Real apps

- Object detection/classification in resource-limited datasets (medical imaging, wildlife detection).

Expected outputs

- Training/validation accuracy curves, saved model with better generalization than training from scratch on small data.

Additional requested topics

Indexing, IR performance measures, and feature extraction — concise guide (exam-style)

Although not a DL lab core, these are useful for NLP & retrieval tasks (CBOW, recommender, web-crawl use-cases).

Indexing basics

- **Inverted index:** term → list of documents (postings). Key for search engines.
- **Tokenization, stop-word removal, stemming/lemmatization** applied before indexing.

Feature extraction (for IR/NLP)

- **Bag-of-Words (BoW)**, **TF-IDF**: TF = term frequency, IDF = log(N/df).
    - `tfidf(t,d) = tf(t,d) * log(N / (df(t)+1))`
- **Word embeddings:** Word2Vec (CBOW/Skip-gram), GloVe, contextual embeddings (BERT). Useful for semantic search and recommendations.
- **Image features:** CNN feature vectors (use pre-trained CNN as extractor), SIFT/ORB (classical).

IR performance measures

- **Precision@k:** fraction of relevant items in top-k results.
- **Recall@k:** fraction of all relevant items retrieved in top-k.
- **MAP (Mean Average Precision):** average precision across queries.
- **NDCG (Normalized Discounted Cumulative Gain):** accounts for graded relevance and rank position.
    - `DCG_k = sum_{i=1..k} (2^{rel_i}-1) / log2(i+1)`
    - `NDCG_k = DCG_k / IDCG_k`
- **ROC-AUC / PR-AUC:** for binary relevance scoring tasks.

Practical tips for DL-backed retrieval

- Use embeddings + approximate nearest neighbors (FAISS) for semantic search.
- Combine lexical (TF-IDF) + semantic (embedding) scores for robust ranking.

Common viva FAQs across experiments (professor-style)

- **Explain bias-variance tradeoff and how DL models manifest it.**
    - High-capacity models can overfit (low bias, high variance); regularization, dropout, augment data, reduce capacity to manage.
- **Why monitor validation loss, not just training loss?**
    - Validation indicates generalization; training loss can decrease while val loss increases (overfitting).
- **How to choose optimizer?**
    - Adam for fast convergence; SGD (with momentum) often generalizes better; experiment and tune LR.
- **Explain difference between precision and recall with example.**
    - Precision = TP/(TP+FP), recall = TP/(TP+FN). E.g., in anomaly detection, recall measures how many real anomalies were found.

Common mistakes (cross-experiment checklist)

- Not fixing random seeds when comparing experiments.

- Not normalizing inputs to match pre-trained models.
- Forgetting to set model to eval mode for inference (`model.eval()` in PyTorch).
- Using wrong loss for task type.
- Training / evaluating on same dataset (data leakage).

Real-world application mapping (quick)

- **Feedforward / MLP:** tabular data classification (credit scoring).
- **CNN / Image classification / Transfer learning:** medical imaging, autonomous driving, object recognition.
- **Autoencoder anomaly detection:** fraud detection, predictive maintenance (sensor data), ECG monitoring.
- **CBOW/Word2Vec:** recommender systems (item embedding), semantic search, chatbots.
- **Transfer learning (PyTorch):** small-data fine-tuning for production.

Expected lab outcomes (what examiners check)

- Correct code runs and produces outputs (plots, metrics).
- Proper dataset preprocessing and splitting.
- Correct use of loss & optimizer.
- Interpretation of plots and metrics (explain why train & val behave as they do).

- Clear documentation and printouts attached.

Quick formulae & references (for viva recall)

- **Cross-entropy (multi-class):** `L = - sum_i y_i * log(p_i)`
- **Accuracy:** `(TP+TN) / (TP+TN+FP+FN)`
- **Precision:** `TP / (TP + FP)`
- **Recall:** `TP / (TP + FN)`
- **F1-score:** `2 * (precision * recall) / (precision + recall)`
- **TF-IDF:** `tf * log(N/df)`
- **NDCG:** see DCG formula above.

Short notes & keywords per experiment (one-line flashcards)

1. **DL Packages:** TF2 eager, `tf.keras`, PyTorch autograd, Theano legacy.
2. **Feedforward NN:** Flatten → Dense → Softmax → cross-entropy → SGD.
3. **Image classification:** Conv → ReLU → Pool → FC → Softmax; augmentation & normalization.
4. **Autoencoder:** Train on normal, reconstruction error → threshold → anomaly.

5. **CBOW:** Context → average embeddings → predict target; negative sampling for scale.
6. **Transfer learning:** Freeze conv layers, replace classifier, fine-tune top layers.

Last-minute summary sheet (for oral use)

- **Training loop essentials:** forward → loss → backward → `optimizer.step()` → zero grads.
- **Evaluation:** always `model.eval()` (disable dropout/batchnorm update) + `torch.no_grad()` / `tf.keras` inference mode.
- **Metrics to report:** accuracy, confusion matrix, precision, recall, F1, ROC-AUC (if applicable).
- **Debugging tips:** check shapes, check data normalization, run on a very small subset to verify convergence, visualize wrong predictions.

Annotated checklist to present during viva (what to carry)

- Dataset description & split details.
- Model architecture diagram (text or drawing).
- Hyperparameters (batch size, epochs, learning rate).
- Training & validation curves (loss & accuracy).
- Final metrics & brief interpretation.
- One or two failure cases & what you would try next (regularization, more data, different optimizer).

If you want, I can now:

- Produce a **one-page printable cheat-sheet** from the above (concise) — or
- Generate **detailed Q&A flashcards** (CSV) for quick revision — or
- Create a **slide deck** summarizing each experiment with diagrams & code snippets.

Which of these would you like next? (I'll produce it right away.)