# DL4CV – Assignment 1

Group 5

June 2025

## 1. What is the difference between an iteration and an epoch in this training scenario?

An **epoch** is one complete pass through the entire training dataset.
An **iteration** refers to a single update of the model's parameters using a batch of data.

In this training setup, the training set size is 4200 images and the batch size is 32. Therefore, the number of iterations per epoch is:

$$\text{Iterations per epoch} = \frac{4200}{32} \approx 132$$

Each epoch consists of approximately 132 iterations (i.e., parameter updates).

## 2. What do the values in the last (linear) layer tell you? How are they called?

The values in the last linear layer are called **logits**. They are raw, unnormalized prediction scores output by the model for each class.

In our case, the final linear layer produces a tensor of shape `[32, 2]`, where each row contains two logits corresponding to the two classes: `car` and `not_car`.

Thus, logits are tensors that encode the model's confidence in each class before normalization. A higher logit value for a class indicates greater model confidence in that class.

## 3. Which function do you need to apply in order to receive an actual classification result?

To obtain an actual classification result from the output of the last layer, we need to apply the `argmax` function to the **logits**.

The logits are raw scores for each class. While applying the softmax function:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

converts them into probabilities, classification only requires the index of the highest score, which corresponds to the predicted class:

$$\hat{y} = \arg\max_i \text{ logits}_i$$

This selects the class with the highest logit value for each sample in the batch.

## 4. Why is Cross Entropy an adequate loss function for classification? Could you use an L2-Loss? Why/why not?

**Cross Entropy** is an appropriate loss function for classification tasks because it directly measures the difference between the predicted probability distribution (after applying softmax) and the true class labels. It penalizes incorrect predictions more heavily when the model is confident but wrong, helping to improve class separation and convergence.

**L2 Loss (Mean Squared Error)** is generally not suitable for classification tasks because:

- It treats class predictions as continuous values rather than probabilities.
- It does not promote learning sharp decision boundaries.
- It often leads to slower convergence and inferior performance for categorical outputs.

Therefore, cross entropy is designed for classification and works well with probabilistic outputs, while L2 loss is more suited to regression problems.

## 5. In PyTorch, a Softmax function is integrated into the Cross Entropy loss. What is a Softmax function and what is its effect on the output?

The **Softmax** function is used to convert a vector of raw scores (logits) into a probability distribution over classes.

Given a vector of logits $\mathbf{x} = [x_1, x_2, \ldots, x_K]$, the softmax function computes:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad \text{for } i = 1, \ldots, K$$

This ensures that:

- All output values are in the range $(0, 1)$
- The outputs sum to 1, forming a valid probability distribution

**Effect on the output:**

- It amplifies the difference between high and low scores.
- The class with the highest logit receives the highest probability.

In PyTorch, the `CrossEntropyLoss` function includes the softmax step internally, so logits can be passed directly without applying softmax manually.

## 6. Explain briefly and precisely what leads to model overfitting.

**Overfitting** occurs when a model learns the training data too well, including its noise and irrelevant patterns, instead of generalizing to unseen data. This results in high training accuracy but poor validation or test performance.

Overfitting is typically caused by:

- A model that is too **complex** (e.g., too many layers or parameters)

- **Insufficient training data**
- **Lack of regularization** (e.g., no dropout, no weight decay)
- **Training for too many epochs**

## 7. How would an overfitting of the classifier affect the classification metrics?

When a classifier is overfitted, it performs very well on the training data but poorly on unseen validation or test data. This negatively impacts classification metrics such as:

- **Accuracy:** May remain high on training data but drop on test data.
- **Precision and Recall:** Can become unbalanced, especially for underrepresented classes.
- **F1-Score:** Decreases as both precision and recall decline on unseen data.
- **Confusion Matrix:** Shows more misclassifications on test data compared to training data.

Overfitting leads to misleading performance on training sets and poor generalization.

## 8. Which model performed better in your tests? Why did it perform better? Explain in detail.

In our experiments, the **custom CNN model (CarClassifier)** outperformed the **pretrained ResNet18** in terms of test accuracy and classification metrics.

### Test Accuracy Comparison

- **Custom CNN:** 95.77%
- **ResNet18:** 94.13%

### Metric Comparison (on Test Set)

| **Metric** | **Custom CNN** | **ResNet18** |
|---|---|---|
| Precision (car) | 0.95 | 0.92 |
| Recall (car) | 0.90 | 0.88 |
| F1-score (car) | 0.93 | 0.90 |
| Precision (not_car) | 0.96 | 0.95 |
| Recall (not_car) | 0.98 | 0.97 |
| F1-score (not_car) | 0.97 | 0.96 |

### Why the Custom CNN Performed Better

- **Input Size Match:** The custom CNN was optimized for $64 \times 64$ inputs, whereas ResNet18 was pretrained on $224 \times 224$. Resizing may have caused information loss.
- **Training Strategy:** In ResNet18, only the final layer was fine-tuned. The rest was frozen, limiting adaptation. The custom CNN was trained end-to-end.
- **Overfitting Control:** The custom CNN used dropout for regularization. ResNet18, being more complex, may have overfit slightly.
- **Task Simplicity:** The classification task (car vs. not_car) was relatively simple. A lightweight, task-specific network performed more efficiently.

## Conclusion

Although ResNet18 is deeper and pretrained on a large dataset, the custom CNN achieved better generalization for this specific task. This demonstrates that a lightweight, well-designed network can outperform a complex pretrained model when tuned appropriately.