Dataset name	# Classes	# Train samples	# Test samples	# Channels	Resolution
fashion	10	60,000	10,000	1	28x28
flowers	102*	5732	2,457	3	512x512
emotions	7	28709	7,178	1	48x48
final test dataset	TBA	TBA	TBA	TBA	TBA

Fashion Dataset : **Type** Simple classification, large dataset

🔷 Chosen:

- **Search Space**: *Cell-based*
- **Search Strategy**: *DARTS* or *Random Search with Weight Sharing*

Why did we choose this search space and not others?
Here we have low resolution images (28x28), and we don't need a large model. Also, since this is not a hard task we can define the search space by Cell-based because using Hierarchical space would be too expressive for such a small input and this is overkilling and just wasting resources.

Why did we choose this search strategy and not others?
As we mentioned in our search space choice, the Fashion-MNIST dataset is low-resolution and relatively easy to learn. Therefore, we adopted a one-shot model approach, which builds a single over-parameterized supernetwork and efficiently explores architectures. Among these, we chose **DARTS** for its ability to perform gradient-based architecture search through continuous relaxation, making it faster and more efficient than discrete methods like **ENAS** or **Random Search**, which train one architecture at a time.

Although DARTS requires storing the full supernetwork in memory, this is manageable due to the small size of the architectures required for this dataset. We did not use Bayesian Optimization (e.g., TPE/SMAC) as it involves modeling the architecture-performance relationship, which adds unnecessary overhead for a simple task. While DARTS has known issues such as sensitivity to poorly optimized architectural weights (α), we initially experimented with **vanilla DARTS** due to its simplicity. If instability or ineffective architectures emerged (e.g., overuse of skip connections), we were prepared to switch to **RobustDARTS**, which stabilizes the search by stopping early based on validation loss curvature. We considered **SmoothDARTS** as well, but opted against it due to its added complexity and hyperparameters, which were not needed for this task.

How does DARTS work?
Imagine we have an image, but we don't know which operations (e.g., 3×3 convolution, 5×5 convolution, max pooling, etc.) to apply. DARTS handles this by using all candidate operations in a predefined search space and assigning each one a learnable architecture weight **α**, initialized randomly at first. It represents the neural network as a computational graph, where nodes correspond to intermediate feature maps (i.e., tensors), and edges represent candidate operations that transform one node's output into another. For each edge between two nodes, DARTS applies all possible operations and combines their outputs using a softmax-weighted sum, with weights

determined by the corresponding **α** values. This results in a blended feature map that reflects all operation choices.

The process continues layer by layer within a structure called a cell. Each node in a cell receives **2** inputs (2 is a design choice that is itself a hyperparameter) from earlier nodes, applies the weighted combination of operations along each incoming edge, and passes the result forward. Eventually, the network produces a final output and computes the loss.

During training, both the model weights **w** (e.g., convolution kernels) and the architecture weights **α** are optimized simultaneously using **gradient descent**. After the search phase is complete, DARTS performs a discretization step: for each edge, it selects the operation with the highest α value (after softmax) and discards the rest. This produces a final, fixed architecture, which can then be retrained from scratch for best performance.

Question:
What is the main difference between DARTS and the other one-shot NAS methods we saw before?
Answer:
DARTS uses continuous weights (α) to decide on the operations we should use. In other one-shot methods, on the other hand, such as standard weight-sharing NAS, we can use **Random Search**, which just selects one architecture, trains it for a couple of epochs, and returns the top-performing architecture for full training. Or we can use **ENAS**, which employs an RNN to select the architecture. After training the selected architecture for a few epochs, it uses the validation loss to give either a reward or punishment to the RNN. In both ENAS and Random Search, only the selected architecture's weights are updated at the end of the batch or epoch. However, in DARTS, we update everything at the same time — both α and w.

Question:
How does DARTS optimize the architectural weights and one-shot weights?
Answer:
DARTS optimizes the architectural weights (α) and the one-shot model weights (w) using a bi-level optimization approach. In this setup, the model first updates the weights w by minimizing the training loss while keeping α fixed. Then, using the updated w, it updates the architecture parameters α by minimizing the validation loss. This process is repeated iteratively: for a given α, the model is trained to find the best w on the training set, and then α is updated to improve performance on the validation set. Through this alternating process, DARTS learns both the optimal network structure and its parameters in a way that encourages generalization.

Question:
What are DARTS' main issues and how can they be fixed?
Answer:
DARTS is too sensitive to the architectural weights α. Finding the best α values for every new task is computationally expensive, and when they are poorly chosen, DARTS may return unreasonable or ineffective architectures (for example, ones that mostly use skip connections and don't learn meaningful features). Two solutions are provided in the slides. **RobustDARTS** monitors the curvature of the validation loss, which refers to how sharply the loss changes with respect to α. If the curvature becomes too large, it suggests that the architecture is overfitting or entering an unstable region, so RobustDARTS stops the search early to prevent this. **SmoothDARTS**, on the other hand, applies random perturbations to the architecture weights α during training and also uses adversarial training on α, which means it adds small worst-case changes to α (in the direction that

increases the validation loss the most) to make the architecture search more robust and avoid getting stuck in overly sharp or sensitive regions of the search space.

Question:
RobustDARTS stops the architecture optimization early, before the curvature of the validation loss becomes high. Why do you think this works?
[Hint: think about the discretization step after the DARTS search.]
Answer:
RobustDARTS stops the architecture optimization early to avoid entering regions of the search space where the validation loss becomes highly sensitive to small changes in the architecture parameters (α). When the curvature of the validation loss is high, it means the model is overfitting to the continuous α values, and small changes in α can cause large swings in performance. However, after the DARTS search, we discretize the architecture by selecting the strongest operation per edge (i.e., converting soft α values into hard decisions). If the α values have overfitted to a sharp region of the loss landscape, the final discretized architecture may perform poorly because it no longer benefits from the soft combination of operations. By stopping early—before the curvature becomes too high—RobustDARTS helps preserve architectures that generalize better after discretization.