Layered Quantum Architecture Search for 3D Point Cloud Classification

Supplementary Material

This supplementary material provides:

- A short introduction to gate-based quantum computing necessary to understand the PQC-based model developed in the main paper, Appendix A.
- A visualisation of the voxelised point clouds for different granularity k on the modelNet10 dataset, Appendix B.
- Details pseudo-codes for the evolutionary and local search protocols, Appendix C.
- Details of the evolutionary and layered search strategies on the full ModelNet10 datasets, including results of the training from scratch of the found models, Appendix D.
- Confusion matrices of different benchmark models on the full test set of the ModelNet10 dataset, Appendix D.3.

A. Gate-based Quantum Computing

We provide a short introduction to gate-based quantum computing necessary to understand our PQC models. We introduce qubits, unitary transformations, and measurements, and give a brief discussion of challenges associated with the training of PQCs.

Qubit. The qubit is the fundamental unit of quantum information. It is represented as a vector in the two-dimensional complex Hilbert space and is commonly written as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{12}$$

where $\alpha, \beta \in \mathbb{C}$ and are subject to the normalisation constraint $|\alpha|^2+|\beta|^2 = 1$. This is because α and β represent the probability amplitudes for physically measuring the qubit in the basis states $|0\rangle$ and $|1\rangle$, respectively.

Due to this normalisation condition, the qubit can be rewritten as

$$|\psi\rangle = e^{i\omega}\left(\cos\frac{\gamma}{2}\,|0\rangle + e^{i\phi}\sin\frac{\gamma}{2}\,|1\rangle\right), \tag{13}$$

where the angles $\gamma \in [0,\pi]$ and $\omega, \phi \in [0, 2\pi]$. This representation, called the *Bloch-sphere* representation of $|\psi\rangle$, translates into a unit vector called the *Bloch vector* $\psi_{\text{bloch}} = (\cos\phi\sin\gamma, \sin\phi\sin\gamma, \cos\gamma)^\top \in \mathbb{R}^3$, which can be visualised as a point on the three-dimensional unit sphere, as shown in Figure 7.

Composed Systems. A system composed of multiple separable qubits $|\psi_1\rangle, \ldots, |\psi_k\rangle$ has a state vector $|\psi\rangle$ that is expressed as the tensor product of the individual qubit state vectors:

$$|\psi\rangle = |\psi_1\rangle \otimes \ldots \otimes |\psi_k\rangle\,. \tag{14}$$

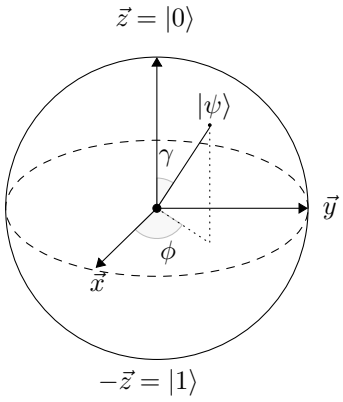


Figure 7. Bloch sphere representation of a qubit. The qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ can be expressed as a unit vector in $\mathbb{R}^3$. Two angles $\gamma \in [0,\pi]$ and $\phi \in [0, 2\pi]$ fully describe the qubit in the basis spanned by the vectors $\vec{x}, \vec{y}$ and $\vec{z}$.

These composed separable qubits span only a subset of the 2^k-dimensional Hilbert space in which the computation takes place. However, a quantum phenomenon known as entanglement allows multiple qubits to become correlated such that the resulting state vector $|\psi\rangle$ can no longer be expressed as a tensor product, thus spanning the complete Hilbert space. A popular example of such a state is the Bell state $|\psi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

Unitary Transformations. Quantum computation consists of transforming the initial state vector of the system into one that encodes the solution of a given problem. In our point cloud classification case, this transformation creates a feature vector useful for classification. Because the norm of the quantum state vector must always be 1, the only valid transformations are unitary transformations, i.e., operators $\mathbf{U}$ satisfying

$$\mathbf{U}\mathbf{U}^\dagger = \mathbf{U}^\dagger\mathbf{U} = \mathbf{I}. \tag{15}$$

Some common single-qubit operators are the Pauli X, Y, Z operators, defined as:

$$\mathrm{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \mathrm{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \mathrm{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{16}$$

In learning tasks, since the exact state vector encoding the solution is unknown, the unitary transformation must be parametrised:

$$\mathbf{U}(\theta,\lambda,\phi) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\theta}{2}\right) \end{pmatrix}. \tag{17}$$

The RX, RY, RZ gates in the main paper are special cases of this general operator.

For multi-qubit systems, the unitary operator can be written as the tensor product of single-qubit operators. Entangled states can only be created through controlled operations, such as the controlled-NOT (CNOT) gate:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{18}$$

One can now parametrise such a controlled operation and let the optimisation decide whether to apply it or not.

Measurement. After the (Parametrised) unitary transformation, the system must be measured to extract useful information. For a one-qubit system, the $|0\rangle$ and $|1\rangle$ basis states are eigenvectors of the Pauli-Z observable, and measurement projects the state onto the Z axis, reading out the z-coordinate:

$$\langle \mathrm{Z} \rangle = \langle \psi | \mathrm{Z} | \psi \rangle \,. \tag{19}$$

Since point cloud classification involves rotations around the x, y, z axes of the Bloch sphere, measuring only in the Z basis is insufficient; hence, all three Pauli bases are measured.

For multi-qubit systems, measurements can be performed globally, by tensoring the single-qubit measurement observables, or locally, by measuring only a subset of qubits. It is important to mention that the measurement process is the only operation on state vectors that is not unitary and cannot be reversed.

Training PQCs. Training PQCs involves optimising circuit parameters to transform the initial state vector into a solution state. Parametrised gates, such as in Equation (17), are differentiable, and measurement, being a vector-matrix-vector multiplication as expressed in Equation (19), is also differentiable. Classically simulated small-scale PQCs in PennyLane use backpropagation for optimisation. For large-scale PQCs, the parameter-shift rule enables computing gradients by evaluating the objective function twice per parameter [36].

Training PQCs faces a major challenge known as the barren plateau, characterised by a flat loss landscape where an exponential number of measurements is needed to approximate gradients accurately [35]. This arises from random states in high-dimensional Hilbert spaces, making measurements vanish on average. The Barren plateau provably has multiple causes ranging from data encoding, PQC architecture, measurement, and noise [30]. Techniques to mitigate this issue include local measurements, proper initialisation, and reducing PQC depth and width [30].

B. Voxelisation

We provide exemplary visualisations of voxelised point clouds for different granularity k in Figure 8 for the data labels `chair`, `sofa`, `table` from the ModelNet10 dataset. We see that the density representation effectively captures finer details about denser and less dense regions of the shape, allowing for further understanding and classifying the shapes. Without density information, the `chair` and the `toilet` at $k = 3$ would more or less have the same shape, up to a rotation. Larger values of k further provide details about the data, but would require more resources and compute time to process the data.

C. Evolutionary Search and Local Search Approaches for PQCs

We provide more details, as well as pseudo-codes for the evolutionary and local search approaches that we use in Section 4.2 of the main text.

C.1. Evolutionary Search for PQC

The evolutionary search approach is similar to other QAS methods [10, 33]. In our setting, we additionally make use of a super-circuit inspired by the classical one-shot architecture search approach in Guo et al. [16]. The super-circuit defines a useful search pool using shared parameters.

Candidate architectures are then sampled randomly from the pool and trained for a few epochs to optimize the super-circuit parameters.

Once the super-circuit is trained, it undergoes an evolutionary search in which a population of architectures evolves to better explore the search space.

The evolutionary search algorithm is described in Algorithm 2. As described above, it consists of two main phases: the training of the super-circuit and the evolutionary search itself.

Search Pool. The super-circuit with shared parameters offers efficient storage management. For our application, the search space consists of several layers of gates, each involving a parametrised gate applied to each qubit. A layer illustration is provided in Figure 9. Candidate gates for each qubit include the identity gate I and Pauli RX, RY, RZ gates as single-qubit operations, as well as controlled CRX rotations for two-qubit interactions. For each qubit, any other qubit can be the target of the CRX. Thus, a layer consists of an application of one gate (single-qubit or controlled-gate) to each qubit.

Search Step. During the search itself, the architectures are ranked based on their performance on the validation set and optionally other user-supplied metrics. Optionally, a

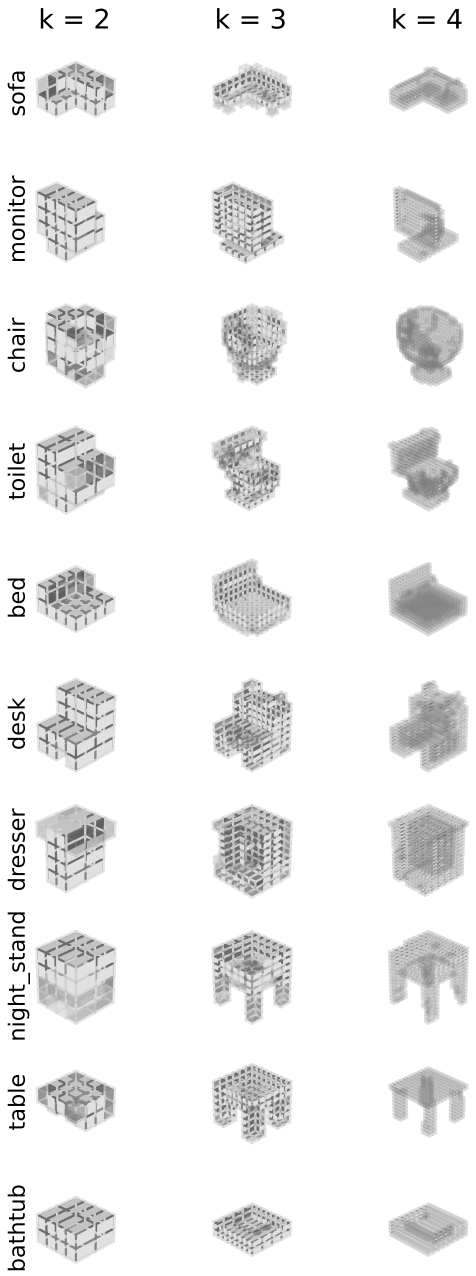


Figure 8. Voxelised point clouds with different granularity k. Brighter voxels are denser voxels, *i.e.* voxels containing most of the points, and darker voxels are less dense voxels. Higher values of k allow for capturing more details on the shape to classify. We use $k = 3$ in our experiments.

fine-tuning of the models can be done before the inference. The best-performing architectures are selected to be parents of the new generation. This new generation of architectures is generated through mutation and crossover operations applied to their parents and becomes the current population:

Algorithm 2 Evolutionary Quantum Architecture Search

Require: SuperCircuit, training and validation sets, population size, ranking metric Best.

for $i = 0, 1, 2, \ldots$ **do**
  Sample $\mathbf{U}_{\text{candidate}}$ from the SuperCircuit pool
  Train $\mathbf{U}_{\text{candidate}}$ for a few epochs
Set $m :=$ PopSise$/2$
Set $n :=$ PopSise$/2$
Initialise population $P_0 :=$ Init(PopSise)
for $i = 0, 1, 2, \ldots$ **do**
  Accuracies $=$ (FineTuning+)Inference(P_i, ValSet)
  Update Topk $=$ Best$(P_i, \text{Accuracies}, k)$ ▷ Ranking
  Set $P_{\text{crossover}} =$ Crossover(Topk, m)
  Set $P_{\text{mutation}} =$ Mutation(Topk, n)
  Update population $P_{i+1} = P_{\text{crossover}} \cup P_{\text{mutation}}$
Return $\mathbf{U} =$ Best$(P_i, \text{Accuracies}, 1)$ ▷ Ranking

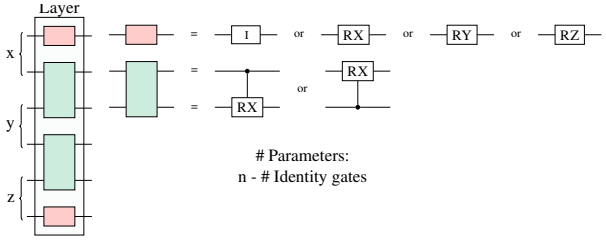


Figure 9. Candidate layers used in the evolutionary search. Note that the gate types and qubits, including controlled and target qubits of controlled gates, are chosen randomly in each layer.

- **Crossover** combines parts of two parent architectures to create a new architecture. The new architecture inherits gate sequences from either one parent or the other, consistently across all layers for each qubit.
- **Mutation** randomly alters some parts of the architecture. To do this, we randomly select a qubit and layer and change the corresponding gate.

The process is repeated until a user-defined number of generations is reached.

C.2. Local Search for PQC

We also adapt the classical local search [49] to the setting of PQCs by making use of the trained super-circuit from the evolutionary search approach. As this super circuit was already trained on a search pool for the evolutionary search, we leverage the trained parameters of the super-circuit to optionally just fine-tune the candidate models for one epoch before inference on the validation set, instead of training all candidate architectures in the neighbourhood of the current model from scratch as described in White et al. [49],

We start from a random architecture, randomly select a qubit and layer, change the corresponding gate, and accept the change if it improves the QPC performance on the vali-

Algorithm 3 Local Quantum Architecture Search

Require: SuperCircuit, training and validation sets, Neighbourhood function, ranking metric Best.

Sample $\mathbf{U}$ from the SuperCircuit pool
for $i = 0, 1, 2, \ldots$ **do**
  ArchList := [ ] ▷ Empty list
  for $k = 0, 1, 2, \ldots,$ NumCandidates **do**
    Pool_k = SuperCircuit ∩ Neighborhoud($\mathbf{U}$)
    Sample $\mathbf{U}_{\text{candidate}}$ from Pool_k
    Optionally fine-tune $\mathbf{U}_{\text{candidate}}$ for one epoch
    Append $\mathbf{U}_{\text{candidate}}$ to ArchList
  Update $\mathbf{U}$ = Best(ArchList) ▷ Ranking
Return $\mathbf{U}$

dation set after one fine-tuning epoch.

We provide the detailed algorithm used for the local search in Algorithm 3.

D. Search on the Full ModelNet10 Dataset

We describe the behaviours of the layered and evolutionary models when trained on the full, large ModelNet10 dataset.

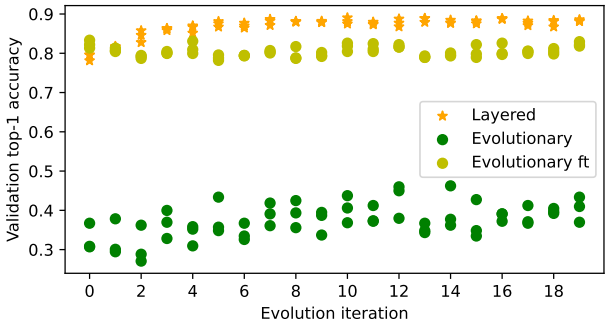
D.1. Analysis of the Search Behaviours

As mentioned in Section 4, we observed that the search algorithms, principally the evolutionary search, struggle to identify better models when using the full datasets, in addition to increasing the training time. The principal reason is that 5 epochs of training candidate models on the full, large dataset is sufficient for the model parameters to converge. As a consequence, the weight-sharing mechanism of the evolutionary search leads to conflictual parameter updates.

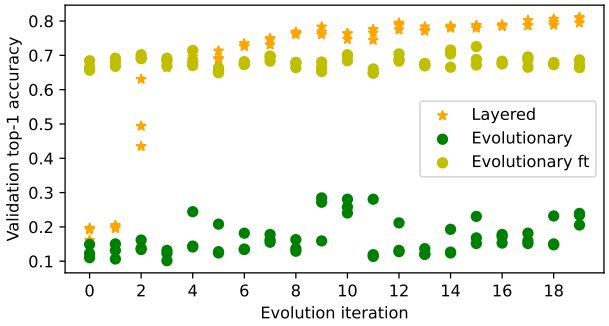
Figure 10 presents the convergence behaviour of the search algorithms on the full ModelNet10 dataset, with the same training configuration as in the main paper. We see that both the evolutionary search variants, with and without fine-tuning, face difficulties in identifying good models over generations. In contrast, the layered search continues to improve prediction accuracy, with competition between candidate models becoming tighter as generations progress. The interpretation is that sufficient training data underscores the strengths of the models, leading to improved overall performance.

D.2. Performance Evaluation

We next report performance results of the best models found on the full ModelNet10 dataset. The best model of each search strategy is the model with the highest validation top-1 accuracy after search. At the same time, we investigate whether or not the warm start of the candidate models is beneficial in optimising the found architectures. To this end,



(a) Learnable linear layer



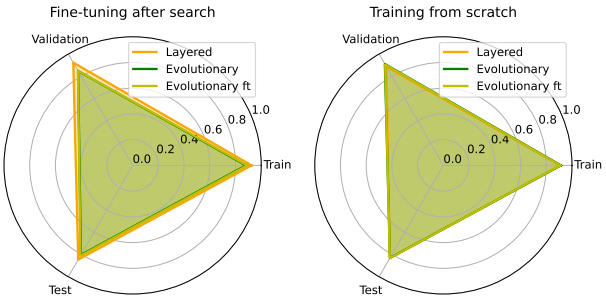
(b) Frozen linear layer

Figure 10. Convergence plot of the search algorithms on the full ModelNet10 dataset. Shown are the accuracies of the three best candidate models for each search strategy. The Layered search improves the prediction accuracy by enhancing the PQC expressivity. The evolutionary search, even with fine-tuning, shows only slight improvement over the generations.

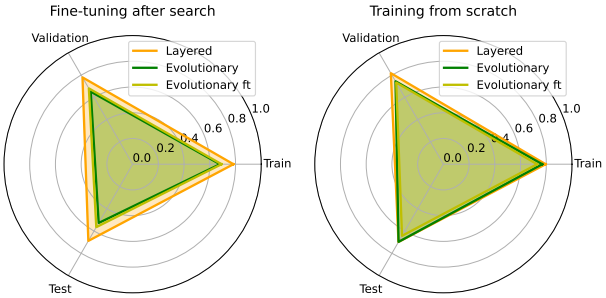
we: (i) Fine-tune the found models over 10 more epochs with the reduced learning rate (we call this fine-tuning after search); and (ii) Train the found models from scratch with randomly initialised parameters for 20 epochs, followed by fine-tuning for 10 more epochs (we call this training from scratch). The parameter range for random initialisation is $[-10^{-2}, 10^{-2}]$.

Figure 11 showcases the performance of the top models identified by the layered and evolutionary search methods on the full ModelNet10 dataset. The layered model significantly outperforms the evolutionary models, with the fine-tuned evolutionary model offering only a slight improvement over the non-fine-tuned version. There is a substantial discrepancy between the accuracies of the evolutionary models after fine-tuning and those trained from scratch, with the latter performing better. In contrast, the layered model trained from scratch achieves a similar accuracy to that obtained after fine-tuning. This indicates that the validation accuracies relied upon by the evolutionary search are highly questionable.

Indeed, search methods using super-circuits or supernetworks, as phrased in common machine learning approaches,



(a) Learnable linear layer



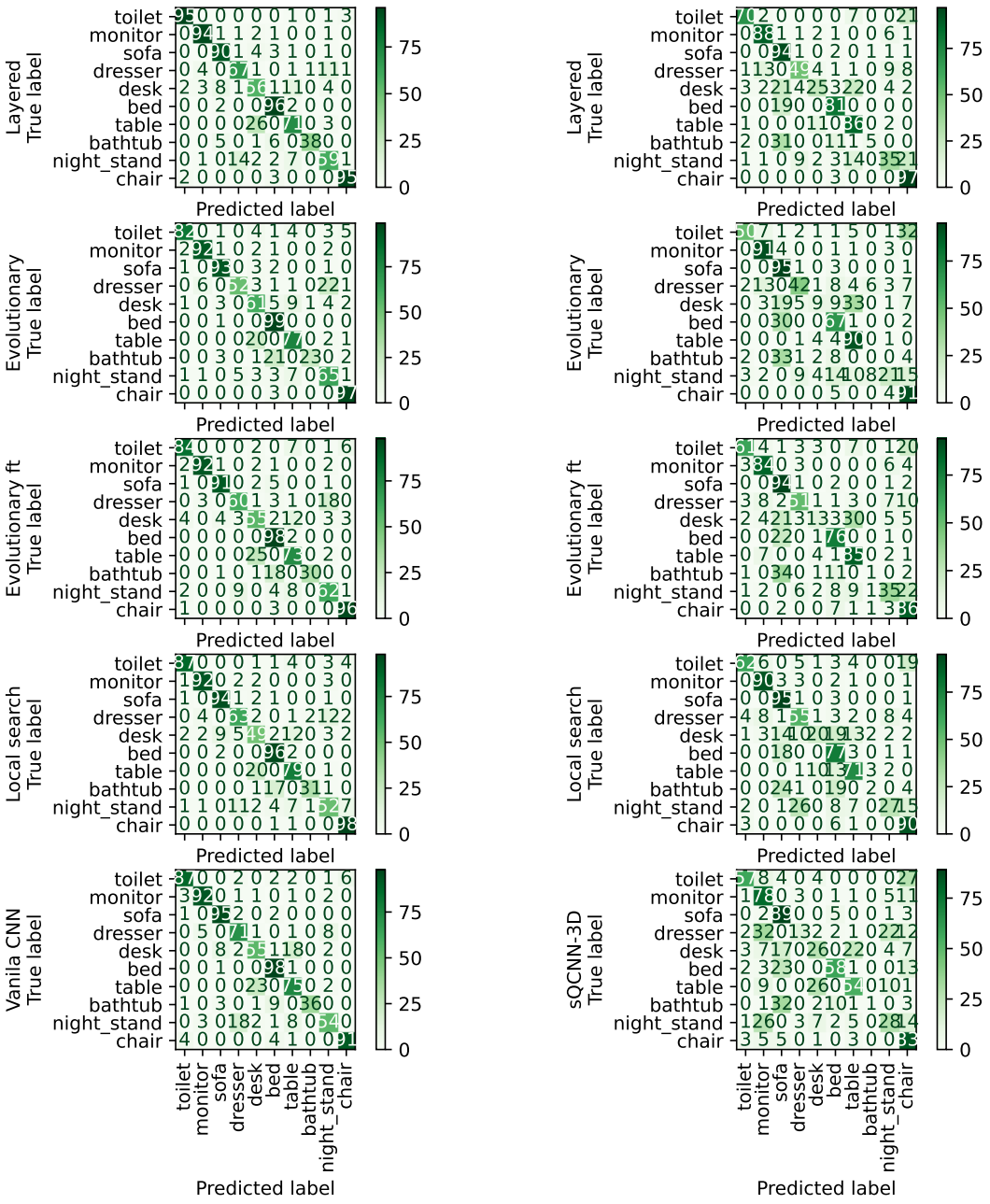
(b) Frozen linear layer

Figure 11. Benchmark performance of found models. Reported are top-1 accuracies on the respective sets.

are based on the assumption that the performance of models using the weight-sharing weights is correlated with the performance of the models being trained from scratch. However, several works discuss the validity of this assumption, arguing that the correlation is heavily dependent on the defined search space and the training of the supernetworks [54, 58]. Thus, the results of the models using weight-sharing weights and training them from scratch can lead to different performances.

D.3. Confusion Matrices on ModelNet10

We now take a closer look at the classification challenges on ModelNet10. Confusion matrices are provided in Figure 12 for all the benchmark models. We observe that some class pairs are more challenging to classify than others. For instance, all the models show difficulties in distinguishing `desk` & `table` or `toilet` & `chair`, since those shapes visually also look very similar. The latter pair is particularly difficult for models with frozen linear layers. The misclassification may be attributed to the coarse voxelisation granularity and the limited expressivity of the models. From the confusion matrices, however, we can see again that the layered model, both with learnable and frozen linear layers, can better classify challenging pairs than other models.



(a) Learnable linear layer

(b) Frozen linear layer

Figure 12. Confusion matrices on the test set of the ModelNet10 dataset. Models with learnable linear layers are more accurate than those with frozen linear layers. All models consistently misclassify similar shapes like `desk`&`table`, `bathtub`&`bed`, or `toilet`&`chair`. The latter pair is particularly difficult for models with frozen linear layers. Note that learnable and frozen linear layers do not apply to the vanilla CNN and sQCNN-3D models on the last row. "ft." stands for fine tuning.