A. Supplementary Material for LQF

In this Supplementary Material we provide additional empirical results (Appendix A.1), more details about the training procedure and implementation used (Appendix A.2), and we derive the expression in eq. (8) for the change in activations when a training sample is removed (Appendix A.3).

A.1. Additional results

Preconditioning with Adam vs. K-FAC. In Section 4.1 we train LQF and LQF FC using K-FAC for preconditioning, instead of alternatives like Adam. K-FAC provides several advantages: the learning rate choice is easy to interpret in the case of a quadratic problem (Section 3.2), it easy to analyze theoretically (e.g., in eq. 6), and it provides better convergence guarantees that Adam. Moreover, it provides an approximation of the inverse of the Hessian which we need to compute eq. (8). We now test how Adam and K-FAC preconditioning compare to each other in terms of raw test error after hyper-parameter optimization. We train LQF, LQF FC and GaF with Adam on all datasets and report the results in Table 3. For each dataset we try different learning rates $\eta \in \{0.001, 0.0004, 0.0001\}$, weight decay $\lambda \in \{10^{-5}, 10^{-6}\}$ and augmentation schemes (central crop, random crop, random resized crop) and report the best result. We observe that after hyper-parameter optimization LQF obtains similar final accuracies when trained with Adam and K-FAC (Table 3). Similarly, for GaF we observe that Adam achieves errors comparable with SGD (in this case without K-FAC). However, K-FAC performs better than Adam when training only the last layer (LQF FC). This suggests that the more sophisticated pre-conditioning of K-FAC is more important to ensure good convergence when optimizing the lower dimensional, badly conditioned problem of LQF FC.

MSE loss for non-linear fine-tuning. In Figure 8 we train a standard non-linear network using cross-entropy loss and MSE loss with different number of training samples. We train with learning rate $\eta \in \{0.1, 0.05, 0.01, 0.001, 0.0001\}$ and weight decay $\lambda \in \{0.0001, 0.00001\}$ and report the best result. We observe that the MSE loss tends to outperform cross-entropy in the low-data regime, suggesting that some of the benefits of the MSE loss also apply to non-linear fine-tuning.

Ablation study for on-line learning. In addition to Figure 5, in Figure 7 we show the result of using LQF with CE loss instead of MSE, ReLU instead of Leaky ReLU or not using K-FAC. This shows that all components contribute to better on-line performance, but their contribution is relatively minor with respect to the difference between using LQF (solving a convex problem) or NLFT (non-convex).

Tuning weight-decay. We did not observe major differences in accuracy by tuning the weight decay parameter suggesting that the inductive bias of linearization is enough to grant good generalization.

However, if required, we note that LQF provides a way to tune weight decay efficiently. Since there the LQF loss function is strongly convex, it has a unique global minimum. This implies that, after training with a given value of weight decay, we can increase the value of weight decay and fine-tune the previous solution to converge to the new global minimum. This is in contrast with DNNs, which may remain stuck in a suboptimal local minimum if weight decay is changed after convergence [11]. In Figure 9 we show the test accuracy obtained on Caltech-256 when training from

	LQF	LQF FC	GaF
Caltech-256	**14.2**	17.2	16.0
Chest X-Ray	6.6	9.6	**6.1**
Malaria Cells	**4.2**	6.1	4.8
MIT-67	**20.4**	25.4	23.1
Oxford Pets	**6.7**	7.8	7.2
Fine-grained datasets			
Stanford Dogs	12.4	13.3	**11.7**
Oxford Flowers	**6.5**	10.8	13.8
CUB-200	**23.1**	29.3	28.5
Aircrafts	**33.1**	45.9	45.6
Stanford Cars	**24.0**	39.2	37.0

Table 3: **Test errors using Adam.** We report the test errors obtained by training different linear methods with Adam instead of SGD (in the case of LQF, we also train without K-FAC preconditioning). We note that Adam gives comparable results to SGD+K-FAC for LQF and to standalone SGD for GaF (Table 1), but is slighly worse than SGD+K-FAC for LQF FC, where we only optimize the last layer. The only exception where Adam improved results for LQF and GaF is *Chest X-Ray*, possibly due to the more exhaustive hyper-parameter search we used for Adam.

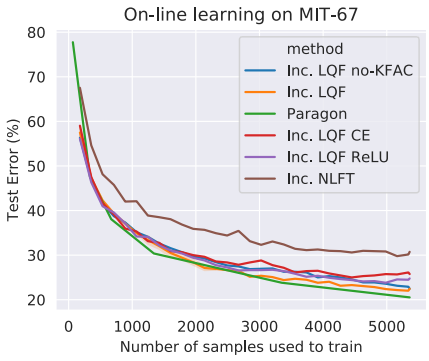

Figure 7: **Ablation study on on-line learning.**

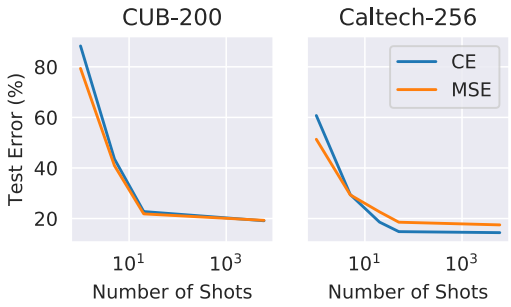

Figure 8: **Using MSE loss with NLFT.** Plot of the test error obtained by training with NLFT using either cross-entropy of MSE loss for different number of training samples per class (shots). While cross-entropy loss is comparably or better than MSE loss when training with many samples, we observe that in the low-shot regime MSE tends to always outperform CE. This suggests that using MSE loss is not beneficial only for lienarized models.

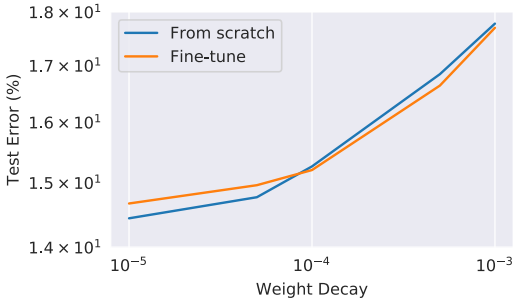

Figure 9: **Exploring different weight decay values via fine-tuning.** We compare the test error obtained by training from scratch with a given weight decay value λ, and the test error obtained by loading the solution found with $\lambda_0 = 5 \cdot 10^{-4}$ and fine-tuning with a different value of λ.

scratch with different values $\lambda \in \{10^{-3}, 5 \cdot 10^{-4}, 10^{-4}, 5 \cdot 10^{-5}, 10^{-5}\}$ of weight decay. We then load the solution obtained with $\lambda_0 = 5 \cdot 10^{-5}$ and fine-tune it with different values λ of weight decay and compare this with the results obtained training from scratch. We observe that the two approaches (training from scratch and fine-tuning) obtain similar errors, suggesting that indeed for LQF we cab use a cheaper hyper-parameter search based on fine-tuning.

We can also go a step further and automatically optimize weight decay using a validation set. Recall from eq. (4) that the weights at convergence as a function of the weight decay λ can be written as:

$$w^*(\lambda) = (J^T J + \lambda I)^{-1} J(Y - f_0(X)). \tag{10}$$

In particular, $w^*(\lambda)$ is a differentiable function of λ, so we expect to be able to optimize λ using gradient descent. In order to do that, consider the validation loss:

$$L_{\text{val}}^{\text{MSE}}(w^*(\lambda)) = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x,y)\in\mathcal{D}_{\text{val}}} \|y - f_0(x) - \nabla_w f_0(x) w^*(\lambda)\|^2 \tag{11}$$

We want to find the value of λ that minimizes L_{val}. The gradient with respect to λ is given by

$$\partial_\lambda L_{\text{val}} = -w^* \cdot \underbrace{(F + \lambda I)^{-1} \nabla_w L_{\text{val}}}_{\text{preconditioned gradient of val. set}} \tag{12}$$

Note that the second term of $\partial_\lambda L_{\text{val}}$ is simply the preconditioned gradient of the validation loss, which can approximated using K-FAC. We leave further exploration of this research direction to future work.

A.2. Experimental details

Choice of datasets. We have compared the algorithms on a several datasets (Table 1). Most of the datasets are standard in the fine-tuning or fine-grained classification literature [29, 30, 32]. We have added additional datasets to this list (*Chest X-ray, Malaria Cells*) so that we could compare on a different domain (medical images instead of natural images). Some of the datasets we use do not have a standard train/test split. In those cases, we use the following splits when reporting the results: For Oxford Flowers we do not merge training and validation data, we only use the training images (1020 samples, instead of the 2040 of train+val). For Caltech-256, we randomly sample 60 images per class to train and test on the remaining images (this scheme is sometimes called Caltech-256-60 in the literature). For Malaria Cell, we randomly select 75% of samples for training, and test of the remaining ones.

Pre-training. In all our experiments we use a ResNet-50 backbone pretrained on ImageNet using SGD (we use the reference PyTorch pretraining scheme: 90 epochs with cross-entropy loss, learning rate 0.1 decayed by a factor 10 every 30 epochs, momentum 0.9). We use Leaky ReLU activations for LQF, while we use standard ReLU activations for all other methods. To ensure that the comparison is as fair as possible, we obtain the weights of the Leaky ReLU network by starting from the pretrained ReLU network, changing the activations, and then fine-tune on ImageNet with SGD for another epoch.[2] We also tried training the backbone from scratch on ImageNet using Leaky ReLU activations, but did not observe any major difference in performance between the two.

[2]Interestingly, we observe that even without fine-tuning the weights learned for ReLU activations still achieve a good test error when used with Leaky ReLU activations. This makes fine-tuning with the new activations particularly easy.

Optimization. In all experiments, we train using SGD with momentum 0.9 and batch size 28. We search for the learning rate in $\eta \in \{0.01, 0.001\}$ and weight decay $\lambda \in \{10^{-4}, 10^{-5}\}$. We report the best result. Instead of applying weight decay directly to the gradient update as often done, we add the ℓ_2 weight penalty to the loss function. This is required when using K-FAC to compute the correct pre-conditioned update, but it does not otherwise change the udpate equation when not using K-FAC.

In Appendix A.1 we present additional results using Adam. In this case we search for the learning rate in $\eta \in \{0.001, 0.0004, 0.0001\}$ and weight decay $\lambda \in \{10^{-5}, 10^{-6}\}$. When training GaF with Adam we use $\beta_1 = 0.5$ as suggested in [36], otherwise we use $\beta_1 = 0.9$.

Data augmentation. Since different tasks may require different types of data augmentation, we train with different augmentation schemes (center crop, random crop, resized random crop) and report the best result. For *center crop*, we resize the image to 256×256 and extract the central crop of size 224×224, while for *random crop* we extract a 224×224 in a random position and also apply a random horizontal flip. In both cases we test with center crops. We also try the *resized random crops* augmentation commonly used for ImageNet pre-training. We found this augmentation to be too strong for some tasks, and it gives significantly better results only in the case of *Chest X-Rays*. For this reason, and to save computation time, we only train random crop with one combination of hyper-parameters (learning rate $\eta = 0.001$ and weight decay $\lambda = 10^{-5}$).

Linearization. We follow the procedure of [36] to linearize the network. The main difference is that we do not fuse the batch norm layer with the previous convolutional layer, but rather we also linearize the batch norm layer. We did not observe major difference in performance by using one or the other solution, but we opted for the latter to keep the structure of the linearized and original network as close as possible when performing the comparison. In order to ensure linearity, for all linear models (LQF, LQF FC, GaF, Standard FC) we put batch normalization in eval mode when training (that is, we use the (frozen) running mean and variance to whithen the features rather than using the current mini-batch statistics).

B-LQF. The only change necessary for B-LQF with respect to stnadard LQF is to replace the global average pooling before the classification layer with a linearized version of bilinear pooling with square root normalization [31]. Let $z \in \mathbb{R}^{HW \times C}$ be the last convolutional layer features, where H and W are the spatial dimensions and C is the number of channels. Their covariance matrix $\Sigma \in \mathbb{R}^{C \times C}$ can be written as:

$$\Sigma = \frac{1}{N} z^T (I - \frac{1}{N} \mathbb{1}) z = z^T A z \tag{13}$$

where I denotes the identity matrix and $\mathbb{1}$ denotes the matrix of all ones and $N = H \cdot W$. Using the same notation as [36], let $h(z(r)) = \sqrt{\Sigma} = \sqrt{z^T(r) A z(r)}$. To compute the linearization forward pass of the layer we need to compute:

$$\partial_r h(z(r)) = \frac{1}{2} \sqrt{\Sigma}^{-1} (\partial_r z^T A z + z^T A \partial_r z). \tag{14}$$

Note that we already have $\sqrt{\Sigma}$ from the forward pass of the base model, so computing the forward pass of the linearized model does not add much to the complexity.

Robustness to optimization hyper-parameters. As mentioned in Section 4.7, to obtain the plot in Figure 1 (right) we train LQF and NLFT with SGD with $\eta \in \{0.05, 0.01, 0.005, 0.001, 0.0001\}$, batch size $b \in \{16, 32, 64\}$ and weight decay $\lambda \in \{10^{-5}, 10^{-4}, 5 \cdot 10^{-4}\}$ and we report the best result. Due to the larger search space, we report the results only for a representative subset of the datasets: *Oxford Flowers, MIT-67,CUB-200, Caltech-256, Stanford Dogs, FGVC Aircrafts*.

A.3. Derivation of interpretability (eq. 8)

Recall that in the case of LQF the hessian of the MSE loss (eq. (3)) is given by $H = F + \lambda I$ where $F = \frac{1}{N} J J^T = \frac{1}{N} \sum_{j=1}^{N} g_j^T g_j$, where $g_j = \nabla_w f_0(x_j)$ are the Jacobian of the i-th sample computed at the linearization point w_0. To keep the notation simple, assume this a binary classification problem, so that the label $y \in \{0, 1\}$ is a scalar and the Jacobian g_i is a row vector (a similar derivation holds for a multi-class problem).

Let w^* be the optimum of the loss function computed using N training samples. Since in a quadratic problem a Newton-update converges to the optimum in a single step, we can write the new optimal weights obtained after removing the i-th training sample – that is, training with only $N - 1$ samples – as:

$$w^*_{-i} = w^* - H^{-1}_{-i} \nabla_w L_{-i}(w^*) \tag{15}$$

where L_{-i} and H_{-i} are respectively the training loss computed without the sample i and H_{-i} is its hessian. Note that we can write $L_{-i}(w)$ as

$$L_{-i}(w) = L(w) - \frac{1}{N} \|y_i - f_0(x_i) - g_i w\|^2.$$

Since w^* is the optimum of the original problem, we have $\nabla L(w^*) = 0$. Using this and the previous equation we get:

$$\nabla_w L_{-i}(w^*) = \nabla_w L(w^*) - g_i^T e_i = -g_i^T e_i$$

where $e_i \equiv \frac{1}{N}(y - g_i w^*)\cdot$ is the (weighted) prediction error on the sample x_i. Plugging this in eq. (15) we have

$$w^*_{-i} = w^* + H^{-1}_{-i} g_i^T e_i.$$

We now derive an expression for H_{-i}^{-1} as a function of F. Note that

$$H_{-i} = F_{-i} + \lambda I = \frac{N}{N-1}F + \lambda I - \frac{1}{N-1} g_i^T g_i.$$

Let $A = \frac{N}{N-1}F + \lambda I$. Note that H_{-i} is a rank-1 update of A. Using the Sherman–Morrison formula for the inverse of a rank-1 update (or, more generally, the Woodbury identity in the case of a multi-class problem), we have

$$H_{-i}^{-1} = (A - \frac{1}{N-1} g_i^T g_i)^{-1} = A^{-1} - \frac{A^{-1} g_i^T g_i A^{-1}}{N - 1 + g_i A^{-1} g_i^T}$$

The activation change $\Delta f(x_{\text{test}}) = f_{w^*}(x_{\text{test}}) - f_{w^*_{-1}}(x_{\text{test}})$ after removing a training sample is then given by:

$$\begin{aligned}
\Delta f(x_{\text{test}}) &= g_{\text{test}}(w^* - w^*_{-i}) \\
&= g_{\text{test}} H_{-i}^{-1} g_i^T e_i \\
&= e_i g_{\text{test}} A^{-1} g_i - \frac{e_i g_{\text{test}} A^{-1} g_i^T g_i A^{-1} g_i}{N - 1 + g_i A^{-1} g_i^T}
\end{aligned}$$

Reorganizing the terms, we obtain the following expression for eq. (8) in the main paper:

$$\boxed{\Delta f(x_{\text{test}}) = (1 - \frac{\alpha_i}{N - 1 + \alpha_i}) e_i g_{\text{test}} A^{-1} g_i^T}$$

where we defined $\alpha_i = g_i A^{-1} g_i^T$. In particular, note that for large datasets ($N \gg 1$) the change in activations is simply given by $e_i g_{\text{test}} A^{-1} g_i^T$ which measures the similarity of the Jacobian of the sample x_{test} and the jacobian of x_i under the metric induced by the kernel A^{-1}. Finally, note that A^{-1} is the inverse of the Fisher Information Matrix F plus a multiple of the identity, which we can easily estimate using K-FAC [35]. This is particularly convenient since we are already computing the K-FAC approximation to precondition the gradients.

In the case of a multiclass problem, the Jacobian g_i is a $C \times D$ matrix, where C is the number of classes and D is the number of parameters. In this case, we get a similar expression:

$$H_{-i}^{-1} = A^{-1} - A^{-1} g_i^T ((N-1) I_c + g_i A^{-1} g_i^T)^{-1} g_i A^{-1}$$

The activation change $\Delta f(x_{\text{test}}) = f_{w^*}(x_{\text{test}}) - f_{w^*_{-1}}(x_{\text{test}})$ after removing a training sample is then given by:

$$\begin{aligned}
\Delta f(x_{\text{test}}) =& g_{\text{test}}(w^* - w^*_{-i}) \\
=& g_{\text{test}} H_{-i}^{-1} g_i^T e_i \\
=& g_{\text{test}} A^{-1} g_i^T e_i \\
& - g_{\text{test}} A^{-1} g_i^T ((N-1) I_c + g_i A^{-1} g_i^T)^{-1} g_i A^{-1} g_i^T e_i
\end{aligned}$$

Reorganizing the terms, we have:

$$\Delta f(x_{\text{test}}) = g_{\text{test}} A^{-1} g_i^T (I_C - M^{-1} \alpha_i) e_i$$

where $M = (N-1) I_c + \alpha_i$, $\alpha_i = g_i A^{-1} g_i^T$ is now a $C \times C$ matrix and I_C denotes the $C \times C$ identity.