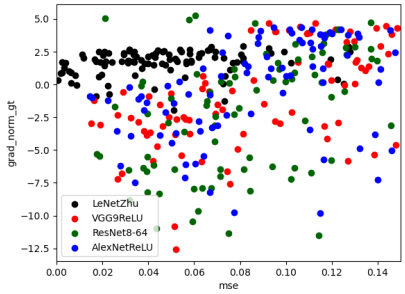
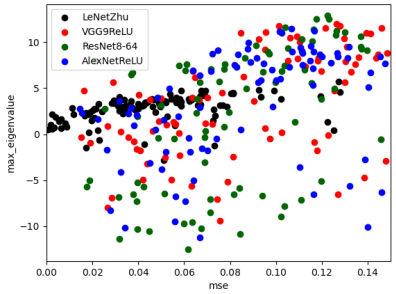
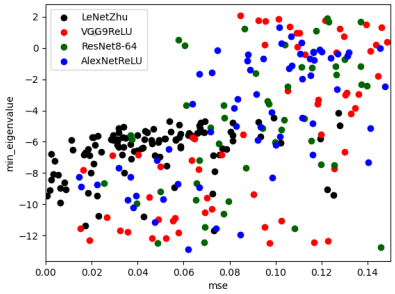


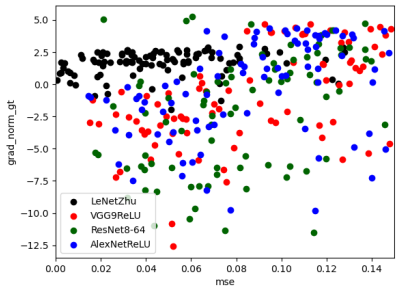
(a) grad_norm vs MSE (L2, $\sigma_S = 0.35$)



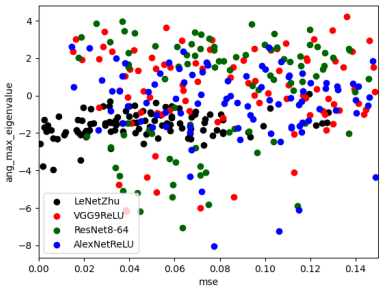
(b) max vs MSE (L2, $\sigma_S = 0.66$)



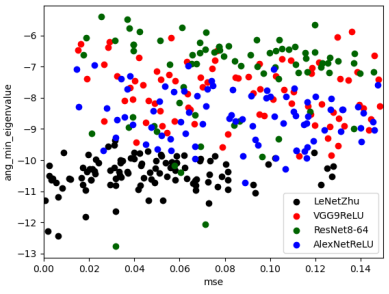
(c) min vs MSE (L2, $\sigma_S = 0.68$)



(d) grad_norm vs MSE (CS, $\sigma_S = 0.00$)



(e) ang_max vs MSE (CS, $\sigma_S = 0.72$)



(f) ang_min vs MSE (CS, $\sigma_S = 0.75$))

Figure 6: **Comparison of gradient norm, maximum and minimum eigenvalues of Hessian in terms of the correlation with MSE of reconstructed samples over several architectures on ImageWoof test samples.**

Algorithm 1: Pseudocode for computing maximum eigenvalue of Hessian, PyTorch-like

```
def max_eigenvalue(x, label, model, g_gt):
    # x: ground-truth image
    # label : x's class
    # model : FL model
    # g_gt : model gradient from ground truth

    m_e = 0 # max eigenvalue candidate

    for _ in range(N): # N is large enough
        v = torch.randn_like(x) # initialize vector
        v = v/torch.norm(v) # normalize vector

        #initialie gradients to zero
        x_tmp = x.copy()
        x_tmp.grad *= 0.0
        model.zero_grad()

        loss1 = loss_fn(model(x_tmp - e*v), label) # compute loss at neighborhood of x, e is small

        gradient1 = torch.autograd.grad(loss1, model.parameters(), create_graph=True) #compute gradient

        g1 = torch.cat([gradient1.view(-1, 1).detach() for g in gradient1], dim=0).squeeze() # flatten gradient into 1-D

        model.zero_grad()

        loss2 = loss_fn(model(x_tmp + e*v), label) # compute loss at neighborhood of x at the opposite side, e is small

        gradient2 = torch.autograd.grad(loss2, model.parameters(), create_graph=True) #compute gradient

        g2 = torch.cat([gradient2.view(-1, 1).detach() for g in gradient2], dim=0).squeeze() # flatten gradient into 1-D


        g_diff = (g2 - g)/(2*e) # compute difference, which is approximately a Jacobian vector product

        g_diff = g_diff.detach()

        # For cosine similiarity loss, compute intermeidate terms with g_gt

        if loss_type == 'cosine':
           g_diff = g_diff - (g_diff*g_gt).sum(0)*g_gt

        model.zero_grad()
        loss = loss_fn(model(x_tmp), label) # compute loss at gt
        gradient = torch.autograd.grad(loss, model.parameters(), create_graph=True) # compute gradient
        g = gorch.cat([gradient.view(-1, 1).detach() for g in gradient], dim=0).squeeze()
        #flatten gradient into 1-D
        ig = torch.autograd.grad(outputs=g, inputs=x, grad_outputs=g_diff) #Hessian vector product

        nrm = (ig[0]*v).sum() #maximum eigenvalue candidate

        m_e = max(m_e, nrm) #update maximum eigenvalue

        v = ig[0]
return m_e
```