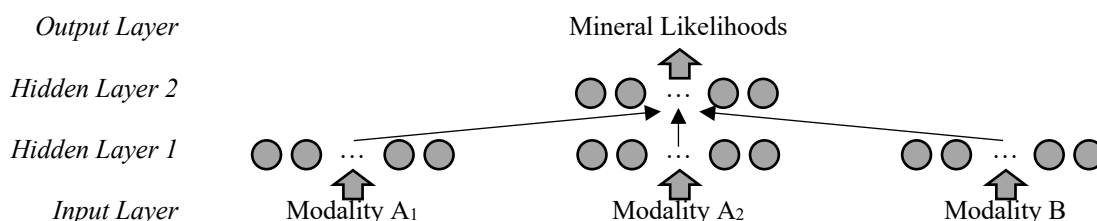


## APPENDIX B: Multimodal Neural Network (MNN) Architecture.

**Figure B1** illustrates the architecture of our MNN. The parallel input layers are used to feed each modality into the MNN. *Hidden layer 1* is used to extract single modality features and is individually trained, by at first directly mapping to mineral likelihoods, considering each modality as independent from the others. *Hidden layer 2* is trained to extract multimodal features by inputting the learned features from *hidden layer 1* and outputting mineral likelihoods. The LR model does not have *hidden layer 1* nor *hidden layer 2*. The MLP model only has *hidden layer 2*. The MNN model has both *hidden layer 1* and *hidden layer 2*.



**Figure B1:** Neural architecture of that used by our Multimodal Neural Network (MNN). Each modality input is a different excitation laser and range of wavenumbers used. Each modality is first trained independently in *hidden layer 1*, and then dependently in *hidden layer 2*.

Each node of a neural network is defined by its activation function, which inputs a linear combination of all nodes in the previous layer. Rectified Linear Units (ReLU)s (Nair and Hinton 2010) is a modern class of activation functions that helped solve a problem in training large neural networks by mitigating unstable error gradients during training, and have been proven to outperform the RBM nodes originally used by Ngiam. Even more recently, Exponential Linear Units (ELU) have been introduced as a variant of ReLU that have been shown by Clevert et al. (2015) to outperform vanilla ReLUs in classification tasks. Sigmoid nodes bind the output between

0 and 1. For all hidden layer nodes in our MNN, we use ELU activation functions, and for all output layer nodes, we use Sigmoid activation functions.

Creating a neural network requires selecting both “trainable” and “hyper” parameters. Each node has a set of trainable parameters that consist of the numerical weights,  $\mathbf{w}$ , used when taking the linear combination of nodes,  $\mathbf{x}$ , in the previous layer and a bias term,  $b$ , used as a starting point (shift) in the activation function,  $f(z)$ , as given by **Equation B1**:

$$f(z) = f(b + \mathbf{w}^T \mathbf{x}) \quad (\text{B1})$$

Hyper parameters define the structure of the neural network and other model parameters used to train it, such as the number of hidden layers and nodes in each layer. Training a neural network consists of optimizing these parameters to minimize a target loss function. The following paragraphs detail the multistage procedure that we use to train an MNN on multimodal spectra. In the following paragraph, we provide ample references for the interested reader, and a more in-depth description is presented by Johnsen et al. (2020).

We first randomly initialize the trainable parameters,  $\Theta$ , near inflection points of each activation function (Glorot and Bengio 2010; He et al. 2015). Then  $\Theta$  is optimized by minimizing the error,  $\varepsilon$ , from our loss function using a partitioned training data set as given by **Equation B2**:

$$\varepsilon = -\frac{1}{nm} \sum_{i=1}^n \sum_{k=1}^m [\mathbf{Y}_{i,k} \log \mathbf{Z}_{i,k} + (1 - \mathbf{Y}_{i,k}) \log(1 - \mathbf{Z}_{i,k})] \quad (\text{B2})$$

Where  $\mathbf{Y}$  is the matrix of ground truth values,  $\mathbf{Z}$  is the matrix of neural network outputs (predictions),  $n$  is the number of spectra in the training set, and  $m$  is the number of mineral classes. **Equation B2** is called a cross-entropy loss function and is commonly used in classification problems to maximize the expected likelihood of model parameters given the training data. We optimize  $\Theta$  using a local search algorithm that utilizes Stochastic Gradient Descent (SGD) with

backpropagation of an error gradient calculated from the loss function (Rumelhart 1986), and a combination of other state-of-the-art processes, including: an Adam optimizer (Kingma and Ba 2014),  $L_1$  regularization,  $L_2$  regularization, dropout (Srivastava et al. 2014), early stopping (Prechelt 1998), and random restarts. The Adam optimizer improves the optimization process by adapting to error gradients based on input data and rate of convergence.  $L_1$  and  $L_2$ , are regularization methods used to “generalize” the model to make more robust inferences on novel data, by mitigating large weights and reducing some to zero (like how a low degree with small coefficients can yield a more robust polynomial fit). Early stopping is used as a termination criterion to determine when to stop the optimization process to mitigate overfitting  $\Theta$  to the training data, by comparing error to a validation set that is not used when calculating the error gradient. Local search algorithm does not guarantee finding the global optimum because it can get stuck in a local optimum that is a function of the initial values of  $\Theta$ . Thus, we employ random restarts by initializing  $\Theta$  with 100 different random seeds that results in 100 different optimized models.

During training, we utilize two methods to capture variance and mitigate error in spectra inputs: dropout (Srivastava et al. 2014) and random perturbations with Gaussian noise. Dropout temporarily drops random nodes (both input and hidden) so that each node learns a latent feature space that is less dependent on its neighbors by reducing the number of co-adaptations. It was shown by Vincent et al. (2008) that randomly dropping features, and further by Bengio et al. (2014) that adding Gaussian noise, can be used to model an expected manifold when using noisy input data. Further, Srivastava et al. (2014) showed that using dropout regularizes the model so that it can make more robust inferences on novel data. By employing such methods, we reduce the need for more intrusive preprocessing methods.

For the MNN, we initially train each stem in *hidden layer 1* on each modality in parallel. After, we freeze the model parameters,  $\Theta$ , in *hidden layer 1* (meaning we do not further change them). Then we train *hidden layer 2*. We later show in our results that if we unfreeze the quantities  $\Theta$  of *hidden layer 1* and train the entire MNN, that we get an erratic learning curve – which shows  $\varepsilon$  as a function of number of training iterations.

We use the Hyperopt library (Bergstra et. al. 2013) to implement an exploration-exploitation Gaussian based process to select the best hyper parameters for each neural network. By using this Gaussian process, we select the following hyper parameters. We use a dropout rate of 0.25,  $L_1$  of and  $L_2$  of  $1e-4$ . The MLP models use 512 nodes in *hidden layer 1*. For the MNN models: *hidden layer 1* uses 64 nodes for modality A<sub>1</sub>, 16 nodes for modality A<sub>2</sub>, and 256 nodes for modality B; *hidden layer 2* uses 32 nodes.