Report on **Neural Network** Implementation and Training

Data Preparation
The MNIST dataset, which consists of a large number of handwritten digits, is used in this project. The data is loaded using the mnist command and then normalized by dividing by 255. This scales the pixel values from the range of 0-255 to 0-1, aiding in the training process. Finally, the data is split into training and test sets.

Model Construction
With the training and test sets prepared, we proceed to implement the neural network model using the Sequential method, which allows us to build a linear stack of layers. The layers used in order are as follows:

1. Flatten: Converts the input 28x28 pixel images into a 784-pixel one-dimensional array.
2. Dense: A fully connected layer. The first hidden layer uses 128 neurons with the ReLU activation function. The second hidden layer consists of 64 neurons, and the output layer has 10 neurons corresponding to the 10 possible digit classes.

In many deep neural networks, a layer called dropout is used to improve training accuracy. Dropout randomly sets input units to zero with a frequency of about 10% at each step during training, which helps prevent overfitting. However, implementing this layer in hardware, especially in FPGAs or ASICs, presents challenges due to the randomness involved. Therefore, we chose to use two fully connected layers instead.

Loss Function: The loss function measures the difference between the model's predictions and the actual values. In this classification problem, we used the Sparse Categorical Crossentropy loss function to train our model effectively.

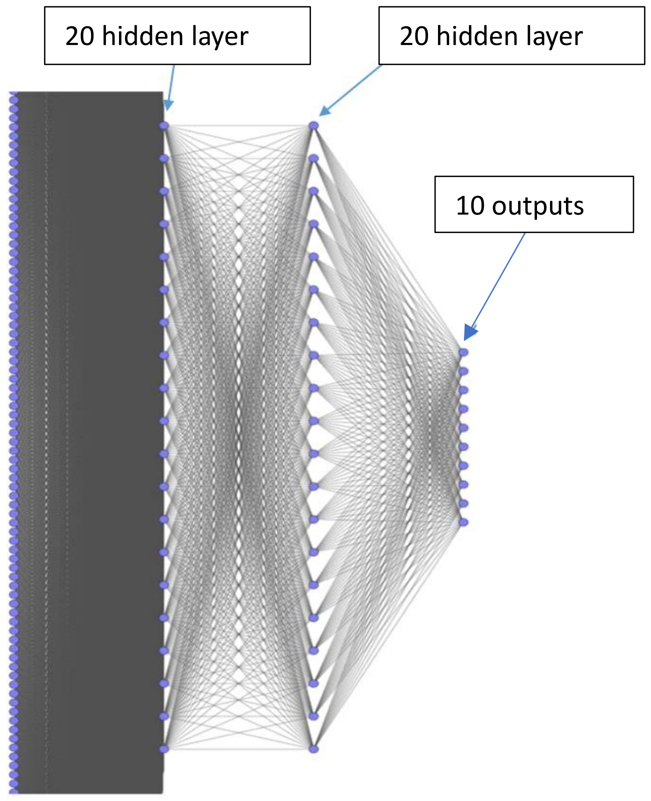
Metrics: Metrics are used to monitor the training process and do not affect the training itself. In this implementation, we use accuracy as the evaluation metric, which indicates the percentage of correctly classified samples.

Optimizer - SGD with Nesterov Momentum: SGD (Stochastic Gradient Descent) is an optimization method used to find the parameter values that minimize the cost function. Instead of calculating the exact gradient of the cost function based on all data, SGD estimates the gradient based on a random sample of the data, which speeds up the training process.

Nesterov momentum is an improvement to SGD aimed at reducing oscillations and accelerating convergence. It predicts the next parameter position using the current momentum and then calculates the gradient at that predicted point. This approach helps in improving the accuracy of update steps. SGD Parameters:

- learning_rate = 0.01: Determines the size of the update steps during the search for the minimum of the cost function.
- momentum = 0.9: Helps in accumulating the gradient of past steps to accelerate convergence and smooth out oscillations.

Compiling the Model
The model is compiled using TensorFlow's compile method, which sets up the loss function, optimizer, and metrics for the training process.



Overview of fully connected ANN with 784 input