EARIN

MINI-PROJECT

Artificial neural networks

“Use FashionMNIST dataset.”

Authors: Das Devansh

Xin Yang

Supervisor: Dr. Daniel Marczak

Catalogue

Introduction

Multilayer perceptron is an architecture of Artificial Neural Network, which is widely used in machine learning and deep learning. It is a feedforward neural network consisting of multiple layers of neurons, each fully connected to the next layer. MLPS can be used in classification and regression problems and are widely used in many application domains, including image recognition, natural language processing, recommender systems, etc.

Multilayer Perceptron (MLP) is a common feedforward neural network architecture, which consists of an input layer, a hidden layer, and an output layer. It is used to solve classification and regression problems and is widely used in many fields. MLPS are trained by a backpropagation algorithm and can be combined with other neural network architectures to handle more complex tasks.

Algorithm:

MLP:

MLP (Multilayer Perceptron) is a popular algorithm in the field of artificial neural networks which is commonly used for classification problems. It consists of an input layer, one or more hidden layers, and an output layer. The input layer takes in the data and the output layer produces the desired output. The hidden layers consist of neurons which perform calculations based on the input and weight values, and pass the result to the next layer. MLP is trained using backpropagation algorithm, which adjusts the weights in the network based on the difference between the predicted and actual output. MLP is a versatile algorithm which can produce accurate results and can be implemented easily using popular open-source frameworks like PyTorch.

Backpropagation:

Backpropagation algorithm is a widely used algorithm for training neural networks, including MLP. It is an iterative technique that adjusts the weights in the network to minimize the difference between the predicted and actual. outputs. The algorithm works by first making a forward pass through the network to obtain the predicted output, then computing the loss function, which measures the difference between the predicted and actual outputs. The gradient of the loss function with respect to the weights is then computed using the chain rule of calculus, which is propagated backwards through the network. Finally, the weights are updated using a learning rate and the calculated gradients. This process is repeated for each input in the training set until convergence is reached. Backpropagation is a powerful algorithm that enables neural networks to learn complex patterns and relationships in data, which makes it widely used in the field of artificial intelligence.

Details:

1. In our code we imports the necessary libraries, including torch, torchvision, and matplotlib.pyplot, for building and training the neural network model and for visualizing the results.
2. sets some hyperparameters such as
   - learning rate (LEARNING_RATE),
   - batch size (BATCH_SIZE),
   - number of hidden layers (HIDDEN_LAYERS),
   - WIDTH of hidden layers (WIDTH),

- activation function (ACT_FUNCTION),
- number of training rounds (EPOCHS).

3. loads the FashionMNIST dataset and splits it into a training set and a validation set. The path of the data set is'. / data, through torchvision. Transforms. ToTensor () converts the image to a tensor format.
4. creates the training and validation dataloaders, which are used to bulk load the data.
5. main part:
   the MLP class is the implementation of a Multi-layer Perceptron neural network using PyTorch. The class takes in the following parameters in its constructor:
   - input_size: The size of the input layer of the MLP, which is 784 for FashionMNIST dataset as each image is 28x28 pixels.
   - output_size: The size of the output layer of the MLP, which is 10 for FashionMNIST dataset as there are 10 different classes of clothing items to predict.
   - num_hidden_layers: The number of hidden layers of the MLP, which is set to HIDDEN_LAYERS in this code (i.e., 2).
   - width: The number of neurons in each hidden layer of the MLP, which is set to WIDTH in this code (i.e., 128).
   - activation_function: The activation function used in the hidden layer neurons of the MLP, which is set to ReLU activation function torch.nn.ReLU() in this code.

   The MLP class inherits from PyTorch's torch.nn.Module class and initializes its layers using a torch.nn.ModuleList(). Depending on the number of hidden layers, the layers are added as follows:

   - For no hidden layers, only one linear (fully-connected) layer is added from input_size to output_size.
   - For num_hidden_layers greater than 0, a linear layer is added from input_size to width, followed by num_hidden_layers-1 linear layers between width and width, and finally a linear layer from width to output_size.

   The forward method performs the feedforward operations of the MLP, including flattening the input tensor, passing it through the hidden layers with ReLU activation function, and passing the output through the final linear layer.

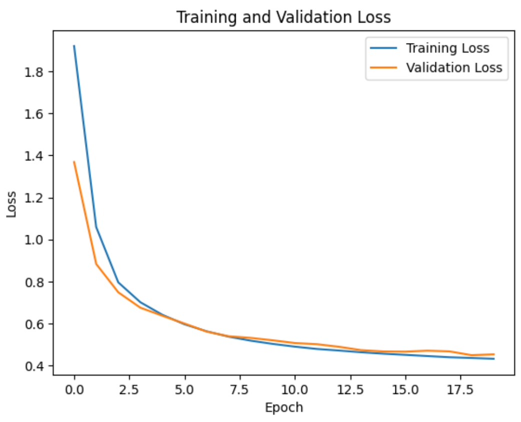
6. creates an instance of the MLP model, defines the loss function (cross-entropy loss) and optimizer (stochastic gradient Descent SGD), which are used to train the model.
7. defines the training and evaluation functions. The training function trains the model using the training dataset and returns the training loss and accuracy. The evaluation function evaluates the model performance using the validation dataset and returns the validation loss and accuracy.

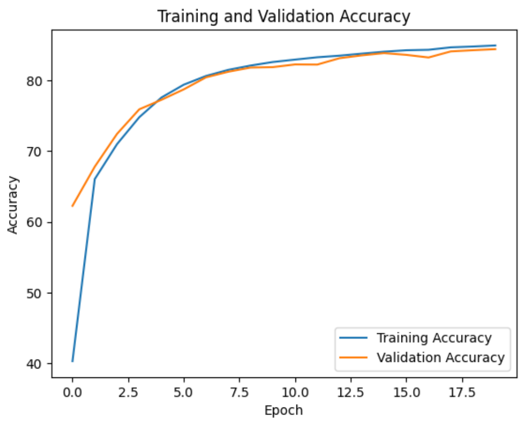
8. uses the train and evaluation functions to train and evaluate the model over a specified number of training rounds (EPOCHS). After each round, the training loss, training accuracy, validation loss, and validation accuracy are printed
9. The last part of the code uses matplotlib.pyplot to plot the training loss and validation loss, as well as the training accuracy and validation accuracy.

Some test case:

Original data:

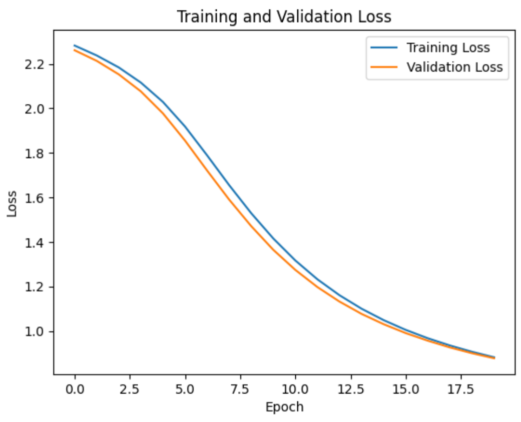
```
LEARNING_RATE = 0.01
BATCH_SIZE = 128
HIDDEN_LAYERS = 2
WIDTH = 128
ACT_FUNCTION = torch.nn.ReLU()
EPOCHS = 20
```

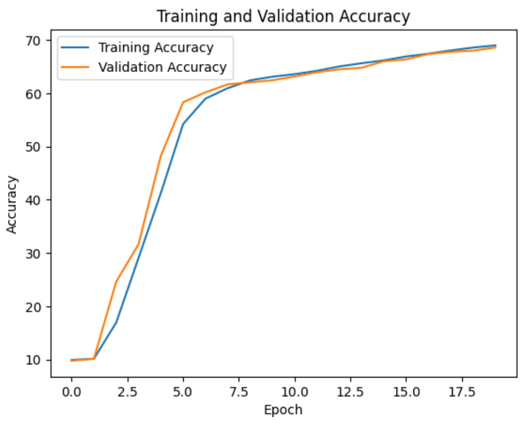




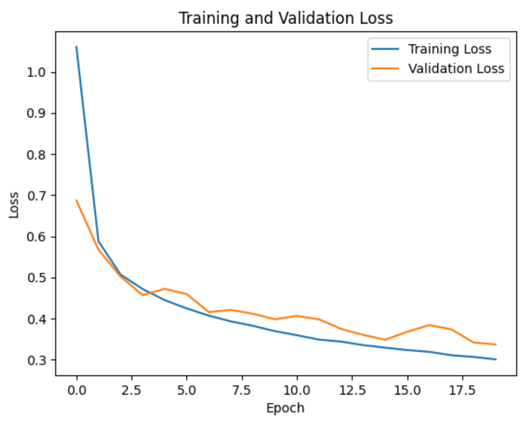
LEARNING_RATE:

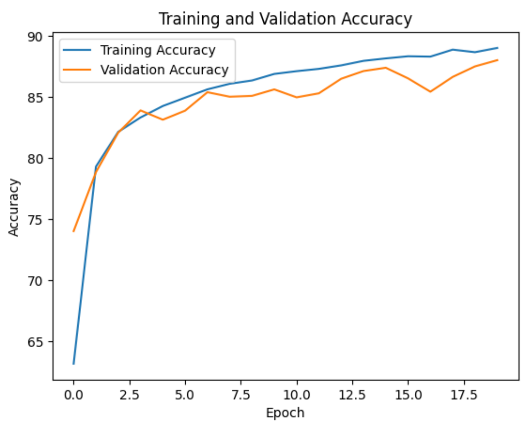
0.001:



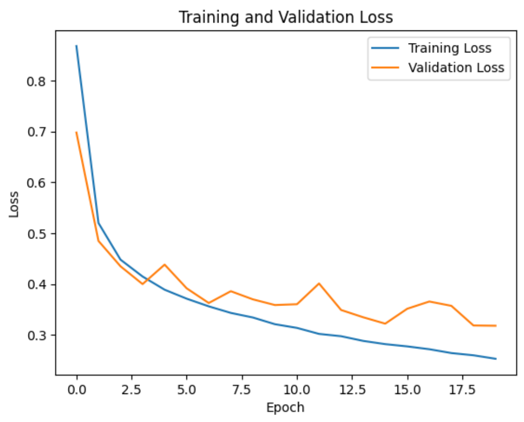


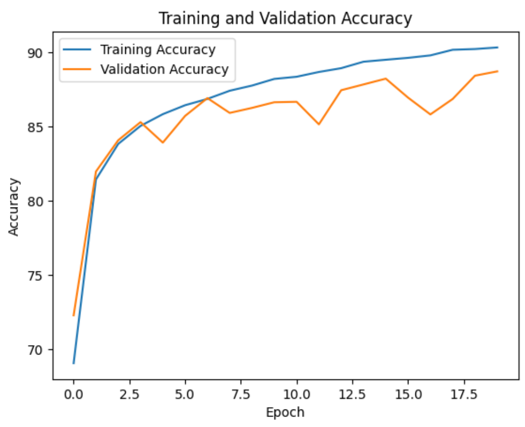
0.05:





0.1:



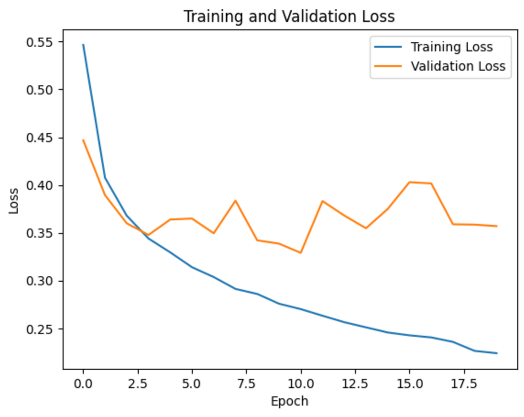


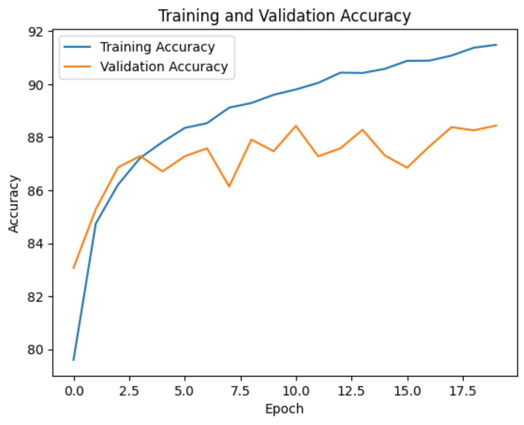
Conclusion:

1. the smaller the learning rate is, the larger the fluctuation of training loss value. With the increase of training times, the loss value gradually becomes the same.
   When the learning rate is too small, the training loss values may no longer conform to the original linear rule
2. the smaller the learning rate, the larger the fluctuation of the training loss value. with the increase of the number of training, the smaller the learning rate, the larger the validation set loss value, but the overall change is not large
3. With the decrease of learning rate, the training accuracy gradually stabilizes in a small interval
4. the overall fluctuation degree of the validation accuracy increases with the increase of the learning rate

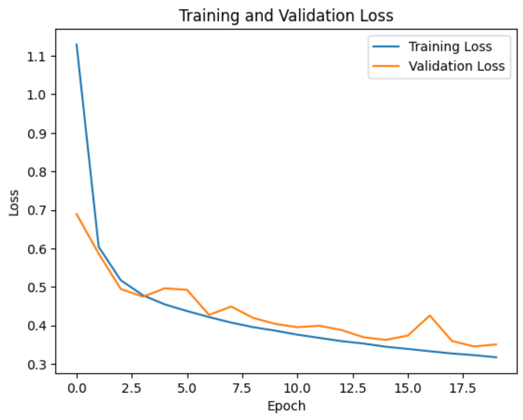
BATCH_SIZE:

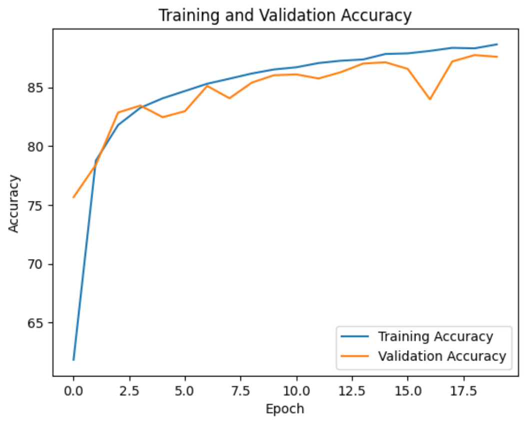
1.



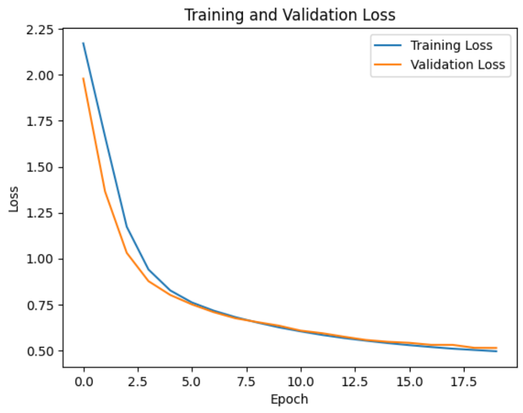


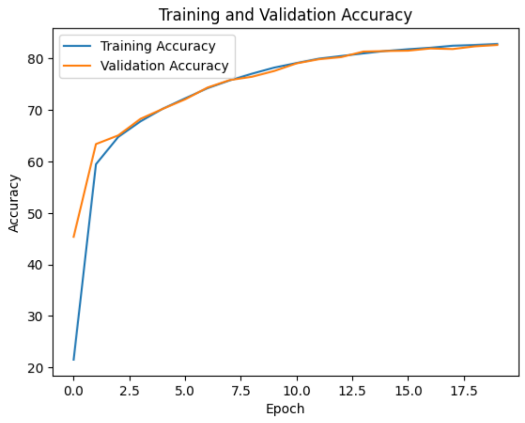
32:





256:



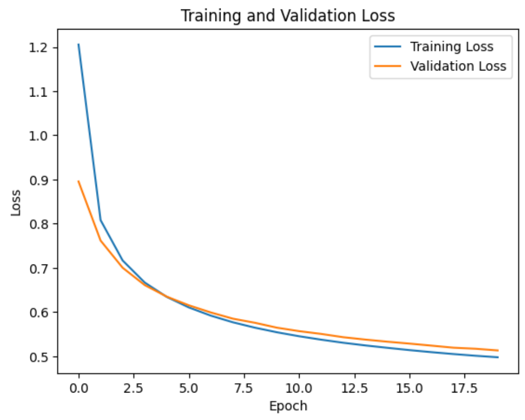


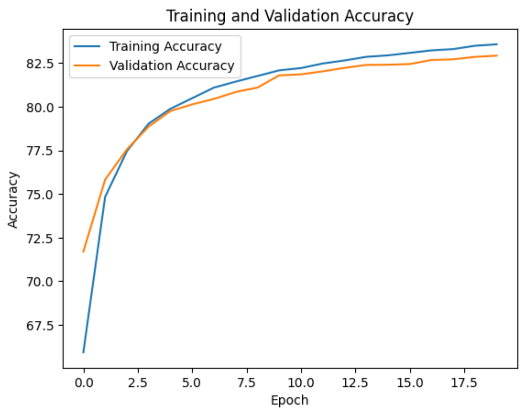
Conclusion:

A smaller batch size may result in less Loss and more accuracy during training, but slower computation; However, a larger batch size may increase the Loss and decrease the accuracy, but the calculation speed is faster.

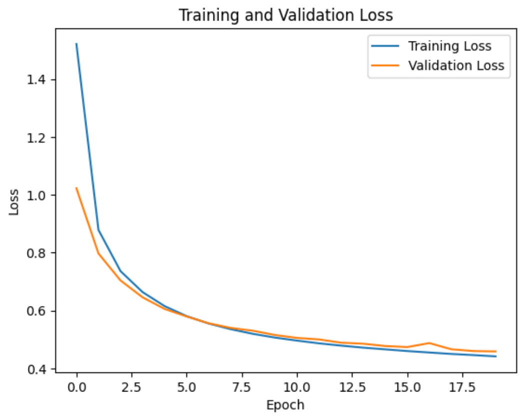
HIDDEN_LAYERS:

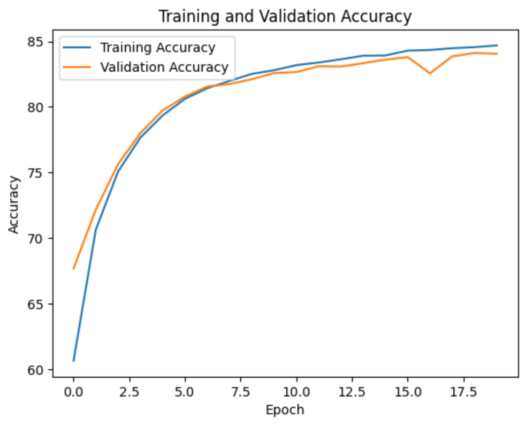
0:



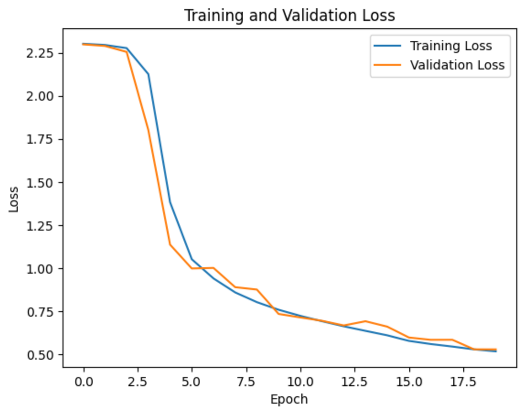


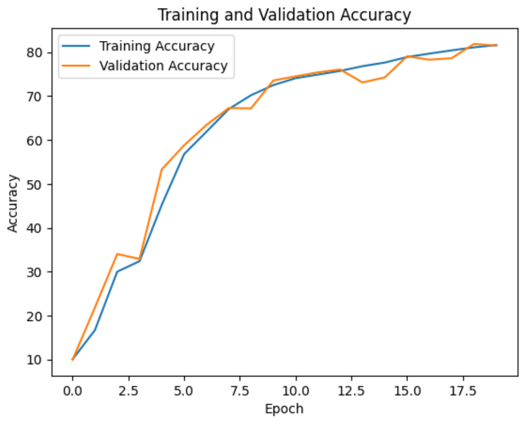
1:





5:





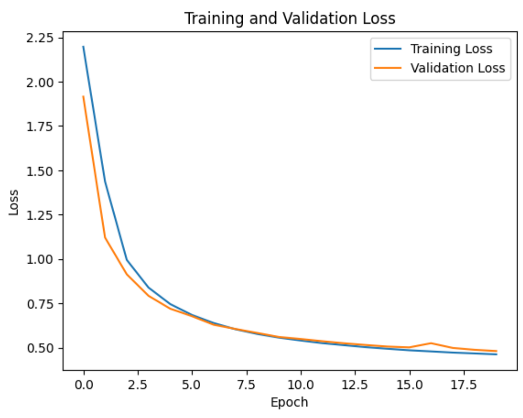
Conclusion:

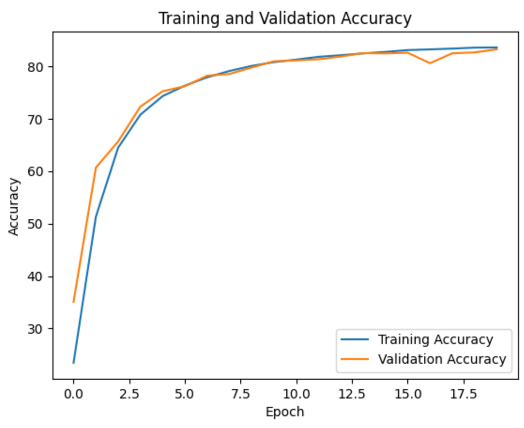
1. When the hidden layer is 0, the fitting ability of the model is weak, which may lead to higher training set loss values and lower training set accuracy.
2. When increasing the number of hidden layers to 1 or 2, the expressive power of the model is improved, and the feature extraction and nonlinear transformation can be better performed. As a result, the loss value of the training set decreases and the accuracy of the training set increases.

3. When the number of hidden layers reaches 5, the model is easy to overfit the training set, which leads to the degradation of the test set performance, which is manifested by higher test set loss values and lower test set accuracy.

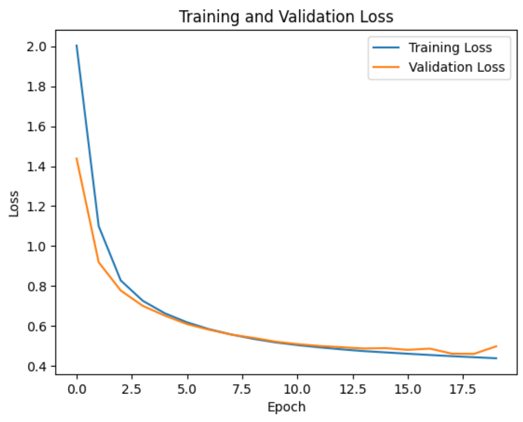
WIDTH:

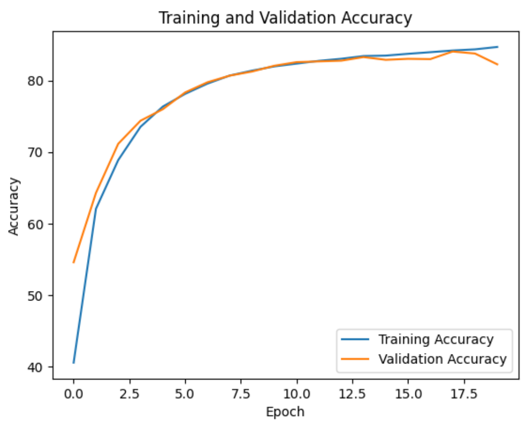
16:



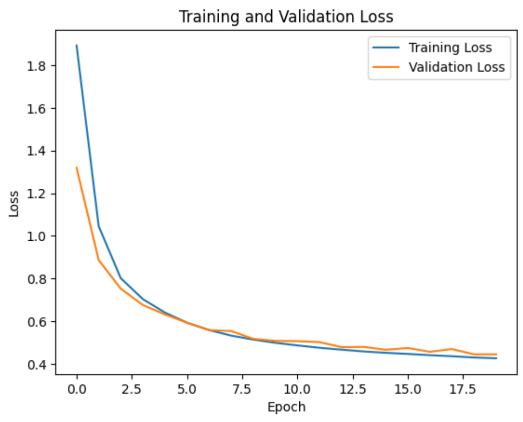


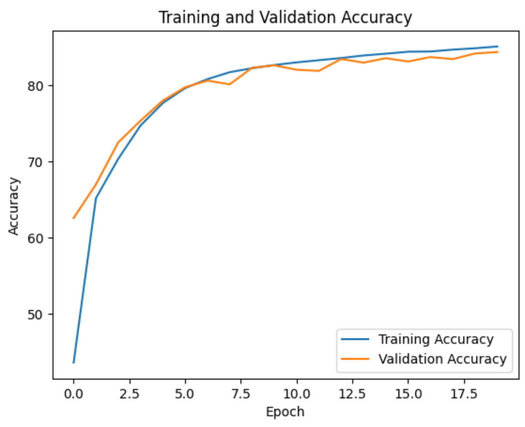
64:





256:





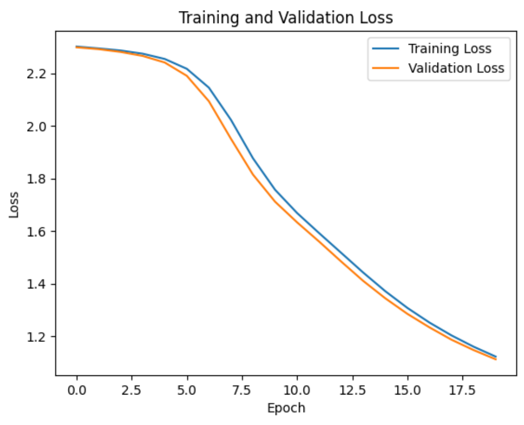
Conclusion:

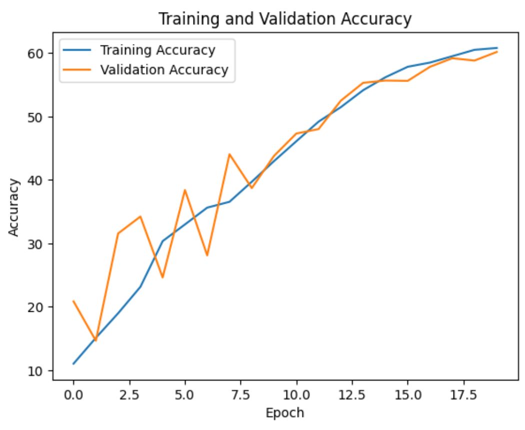
1. When the width is 16, the expressive power of the model is limited, which may lead to higher training set loss values and lower training set accuracy.

2. When the width is 64 or 128, increasing the width improves the expressive power of the model and makes it better fit the training set. This may lead to a decrease in training set loss value and an increase in training set accuracy
3. When the width is 256, a larger width further enhances the expressive power and fitting ability of the model, which may further reduce the training set loss value and improve the training set accuracy. Also increases the risk of overfitting, possibly leading to degraded test set performance

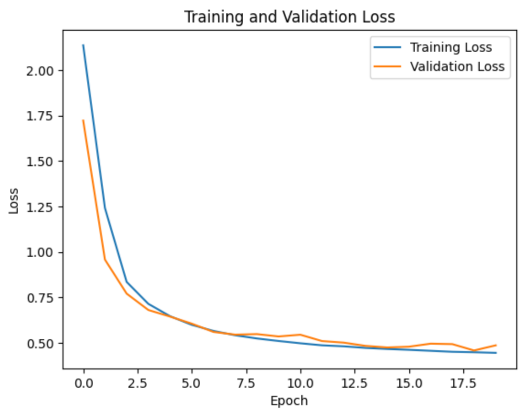
ACT_FUNCTION:

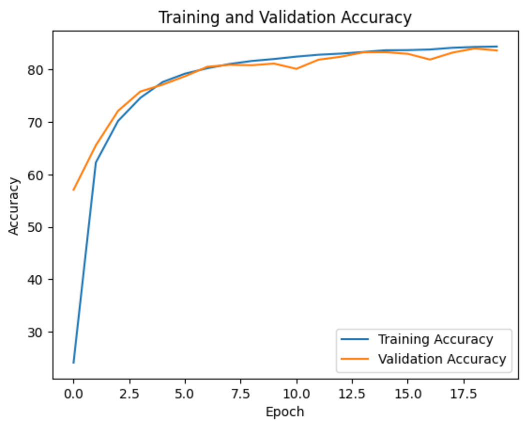
sigmoid:





GELU:





Conclusion:

1. Sigmoid function is suitable for binary classification problems, higher validation set loss values and lower validation set accuracy due to the vanishing gradient problem
2. ReLU function can address the vanishing gradient problem, facilitating faster convergence and higher training set accuracy.
3. GELU function combines the advantages of the sigmoid and ReLU functions, featuring smooth non-linearity and approximate linearity. It contribute to faster convergence and higher training set accuracy.

Reflecting on what went well:

1. well-structured and easy to follow. The hyperparameters are clearly defined at the beginning of the code, making it easy to modify them if needed.
2. produces clear and informative plots of the training and validation losses and accuracies.
3. multiple sets of tests with different data were performed

What could be improved:

1. could define a range of values for the learning rate and batch size and use a grid search or random search to find the optimal values.
2. may move the model to the GPU if it is available