Final Semester Exercise (Objectives):

This is a feasibility assessment project that will allow you to demonstrate what you've learned over the semester. **Your results are not as important as how you explain them,** and I'm focused in many ways on letting you see just how much you've picked up along the way.

This project should have 'ready to run' code -- since the objective is to generate results in a short time and provide some interpretation you won't be working from scratch.

The code provided should run out of the box, it won't do everything in one run but it's a good starting point.
If you are unable to complete any section of the work, please provide an in-depth technical break down of your process to debug the code – this will not provide full credit but it may benefit your overall grade if you find some aspects aren't working as anticipated.

A semi-formal white paper and video project report will take the place of your final project / final exam. Guidelines are provided in the file uploaded to Canvas → "Final Project Report Guidelines" –

📣 **Please***: Read the Scenario section of* **"Final Project Report Guidelines"** *and review the requirements presented in the document before getting started.*

Graduates: Implementation of CNN methods are required, please do your best to implement and test the models as completely as you can.

Undergraduates: Implementation of CNN methods is optional but **Highly Recommended** – they can actually *train faster* in most cases than traditional MLP (all code is included – only small modifications should be necessary). If you prefer the MLP methods however, steps and code included for this part as well.

Final Semester Exercise:

All datasets required for this exercise are included in the zip file.
The scripts to import them for use in your training have been included to let you focus on the project.

MNIST and FASHION_MNIST are identical in size, so they are drop in replacements.
EMNIST_LETTERS is much larger and will take longer to load and train, but it fits in memory and is stable on remote desktop.

***NOTE**: *Confusion plots* for EMNIST_LETTERS data are largely **useless** (too visually complex to display) – please report training results with "ROC" curves and "Performance" plots the 'cross entropy' or 'accuracy' performance metrics.

A) MNIST / EMNIST (Extended MNIST) Letters –
Description: https://www.kaggle.com/crawford/emnist
More info on the dataset can be found here:
https://www.nist.gov/itl/products-and-services/emnist-dataset

Graduate: Implement the following methods:

1) Basic MLP ANN Model – train an MLP using the EMNIST_LETTERS data ('emnist_MLP_ANN_CODE.m') – observe and report the performance for at least **3** configurations trained to **usable** accuracy. Try to

pick an initial set of values based on some aspects of the data (number of samples, number of classes, complexity of features, overlapping distributions, etc.). Generate a new set of parameters for each configuration that you think will improve performance based on each previous attempt.

- If your network is reaching validation stops very quickly, look at the output and adjust the parameters and training options until you obtain usable ROC curves and training performance.

2) Basic CNN model – using scripts provided ('emnist_CNN_CODE.m') generate and report the testing performance of an *optimized* letter based EMNIST model. Build a few models (<5) and attempt to select some parameters you think can help with training. Be sure to save the EMNIST `trainedNet!` See more options for training parameters here:
https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html
Provide an explanation for any poor or outstanding performance (low accuracy or very high accuracy).

3) MNIST digits -- implement a CNN solution for the base MNIST dataset ('MNIST_CNN_CODE.m') and compare the stability with different data divisions. Be sure to save MNIST `trainedNet!` Compare and contrast this to the MLP training from the earlier semester exercise.

4) Implement a transfer learning solution by starting from the model you trained in step 3 **-or-** re-loading the saved MNIST `trainedNet` and using the MNIST digits *model* to work with EMNIST_LETTERS data ('emnist_Transfer_Learning_CNN.m').

5) Implement another transfer learning solution, this time starting from the model you trained in step 2 **-or-** re-loading the saved EMNIST `trainedNet` -- using the EMNIST_LETTERS model to work with FASHION_MNIST data ('fashion_Transfer_Learning_CNN.m'). Use the layerGraph function to replace the classification layers and re-train the fully connected, softmax and output layers.

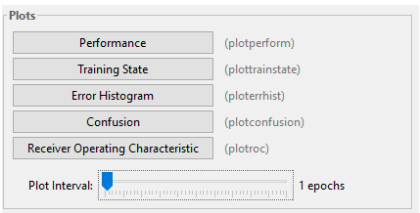
You can reverse the process by ('emnist_Transfer_Learning_CNN.m') and see how well the model recalls the original training [*no need to re-load the model, just use one in the workspace*], or see how well the process prepares the model for the MNIST data ('mnist_Transfer_Learning_CNN.m').

Did the model converge faster than anticipated when working with a pre-trained model?

Undergraduate: Train a network using a pattern net that evaluates the viability of letter classification using an MLP ('emnist_MLP_ANN_CODE.m'). Please look at the training function used in the script to determine the training parameters you can modify, and train **at least** 3 configurations to compare and contrast.

1) Change the nodes, learning rate, or number of training examples in a meaningful way.
- If your network is reaching validation stops very quickly, look at the output and adjust the parameters and training options until you obtain usable ROC curves and training performance.

2) Modify the division ratios: Use as few neurons as you think will work with <40% of your total samples used as training / validation data. Test the model on whatever data (>60% of the set) is left over. {explain how this is more realistic}



3) Try a configuration of the model with only **one** layer. Then generate another with **more** than 2 layers [to add another layer – simply make a new variable 'hiddenLayer3Size' with your selected node size

then add it to the array ↓

```
net = patternnet([hiddenLayer1Size hiddenLayer2Size hiddenLayer3Size],'trainrp');
```

Note: this may limit your training function choice due to the complexity of the back propagation {if you get an **error** related to memory when using 'trainlm' or ' traingdx' try 'trainrp' instead}

`hint: use patternnet so you can see ROC curves` to monitor the performance of your network while it trains, `adjust the plot interval slider to >~10 if you notice any performance downgrade.`

B) Fashion-MNIST

Kaggle Info: https://www.kaggle.com/zalando-research/fashionmnist
Github Link: https://github.com/zalandoresearch/fashion-mnist

Graduate: Implement the following methods:

1) Basic CNN model – using the 3 layer CNN model generate and report the testing performance of your best configuration out of 3. Only 1 training is required for each configuration. Adjust the kernel size, stride, learning rate or training function and record the results.

2) Starting with a CNN model trained from MNIST digits or EMNIST_LETTERS and attempt to implement a transfer learning solution. Compare the rate of adaptation to the generic model.

3) Implement a basic MLP using the provided scripts ('MNIST_MLP_ANN_CODE.m' & 'fashion_MLP_ANN_CODE.m'), adjust the parameters and compare the performance of the models.

Undergraduate: Train an MLP to recognize the subtleties of human clothing.
Give the performance of **at least** 3 networks configurations: selected based on the MLP general rules given in the (Supplemental Notes at the end of this document) for a rough number of nodes needed for some dataset.

1) Using ('fashion_MLP_ANN_CODE.m') Try training with an increased learning rate, and perhaps try training with one of the other training functions such as '*trainlm*' or '*traingdx*' as given in: https://www.mathworks.com/help/deeplearning/ug/choose-a-multilayer-neural-network-training-function.html

2) Compare and contrast MNIST vs EMNIST vs Fashion-MNIST ('MNIST_MLP_ANN_CODE.m' - 'emnist_MLP_ANN_CODE.m' - 'fashion_MLP_ANN_CODE.m') – select some parameters from your past work with both the basic and extended data sets and see how differently the training process unfolds.

3) Try a configuration of the models with **more*** than 2 layers

*[see step 3 in the EMNIST model for implementation info]

Supplemental Notes:
This section is for material that may act as an explanation or give some tips and tricks for those looking for answers or a new approach.

General Rules for MLP Node Complexity:
Described in class was a numerical method for generating a starting point for the number of nodes needed to solve some given problem with a MLP ANN. This formula is given in the review slides for the midterm.
Alternatively you can use the following guidelines for node selection for any arbitrary layer (treat the previous later as input and the next layer in the sequence as the output).

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

🎖 Start off with a single layer when computing these values. If you need to add a layer – put it in-between your hidden layer and the output layer and use the calculation ratio, or the methods above to justify the number of nodes based on the first hidden layer as an input and the classification layer as an output [the new layer will take your first hidden layer's outputs as an input, treating them like features].

A) EMNIST (Extended MNIST)
    a. Saved models can be reused. They've already learned quite a bit in most cases, so if the data has similar patterns it will just refine what the model can do.
    b. If you re-**train** an MPL network with new data, this is typically called 'adaptation'. (I'm providing the term here in case you want to look up some additional information) Using only -Fully Connected- layers in the deep learning toolbox 'trainNetwork' function you can implement an MLP with weights that can be used to classify data from a new data set with a different number of classes or with a different set of label, this can be done the same way as a CNN transfer approach -- by replacing the last few elements in the *layerGraph* and training the model again.
    c. In general, if you train on small amounts of data, and your model doesn't *generalize very well* when you test it on new samples from an extended dataset (this is what we called "under fitting") you may not have a big enough model [not enough nodes or not enough feature layers] to fully represent the distributions.
    d. There's nothing special you need to do to ***adapt*** a model, the **train** function will do that for you – when you **train** a model that has ***non-zero weights***, the function will start off with the weights already in the model rather than generating new ones.
        i. I suggest you copy the weights from the previous training rather than just adapting the model itself – it allows you to attempt the adaptation while retaining the original.
            1. The elements below show how to save, load, and copy a model:
```
save net; //save a copy of the model
load net; //load the values of an existing model
newnet = net; // copy the existing model to adapt
```

e. If you adapt your model and you just can't seem to get good test set results from the new data set -- you'll likely need to retrain *the whole network* with more parameters. More parameters means more nodes/layers which means more degrees of freedom to better **fit** the functions underlying the distributions.

B) Deep Learning Model – EMNIST CNN

Training should not take more than 20 minutes per model to reach viable results (greater than 90% accuracy).

CNN's will typically train MUCH faster than MLPs for MNIST data -- the configurations given should give 'good' results – but you may improve them by fine tuning.

alternative_options – given in the *transfer learning* scripts contain things that can radically alter the performance of a network. If you want more freedom to change the training, copy those sections to the CNN training scripts (be sure to change the name to simply 'options' –or- update the trainNetwork parameters to use the alternative_options object).

The training process for small image data is now VERY fast on the lab or remote computers using the optimized network tools (my models were training from scratch in 5-7 mins). If you need more info on how the models were built – the link below will show you how:
https://www.mathworks.com/help/deeplearning/examples/create-simple-deep-learning-network-for-classification.html

C) *'Transfer learning'* typically applies to the idea of bringing knowledge generated from a model trained on another set of similar data. Remove classification layers and retrain the model with new ones to change the labels or number of classes the model can generate output for. If you freeze the layers leading up the final layers, this is direct transfer, but if you retrain the whole model with the new layers in the loop - this process is typically called 'fine tuning'.
https://www.mathworks.com/help/deeplearning/gs/get-started-with-transfer-learning.html
Not an exact 'drop in' for our code, but its the primary source for the items I've supplied if you need more info.

*Networks are sensitive to batch size, gradient decay, and changes in learning rate (for longer run times). If you want to extend the run time past 2-4 epochs, scale the learning rate and rate update formula accordingly in the transfer learning scripts.

D) Fashion-MNIST

For those of you interested in implementing and training a CNN based solution, the following paper may provide some interesting insights. {In the paper they use the term 'MCNN' – this means the same thing that "CNN" does from our discussions}
Focus on the parameters used (page 69320 shows an 8-layer topology) in their work, and see if you can get similar results.
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8721236

This data is harder to model and will never get as good of results as we did with the standard MNIST model. You might need to play with the parameters a bit and train for a few minutes to see if you're getting results before letting the model train for very long. Since the representations are more complex, it might require more layers than an EMNIST model as well.

Some additional ideas about MLP generalization are included here:
https://www.mathworks.com/help/deeplearning/ug/improve-neural-network-generalization-and-avoid-overfitting.html
You can use this as a resource to discuss potential 'further steps' you might take if the project were approved as given by the scenario!