From Pixels to Sharpies:

Evaluating MNIST Models Against Real Handwritten Digits

Parker Christenson, Philip Felizarta, Gabriel Emanuel Colón

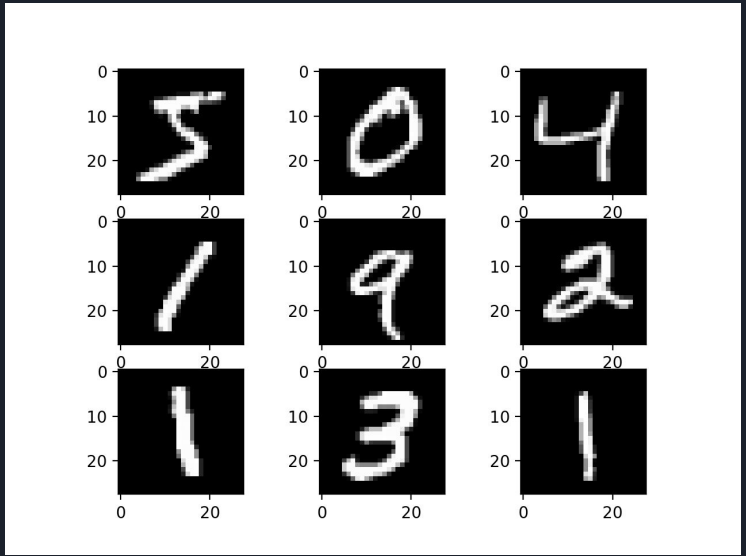
University of San Diego

AAI-501: Introduction to Artificial Intelligence

Prof. A. Van Benschoten

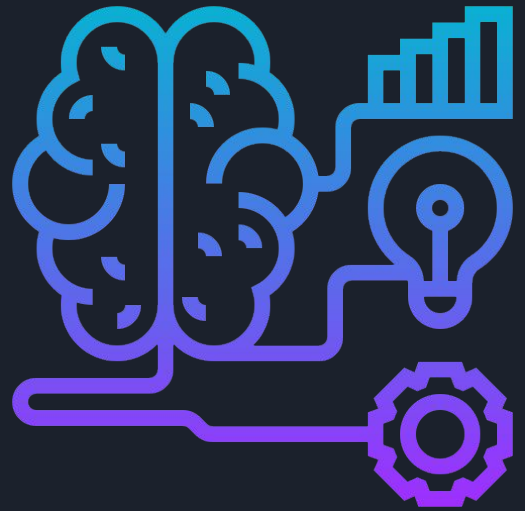
What is the MNIST Data set

- The MNIST data set is a data set comprised of handwritten digits that is used to evaluate machine learning algorithms
- The data set is comprised of 70,000 images total (*NIST Special Database 19 | NIST*, 2019)
  - 60,000 being used for Training
  - 10,000 being used testing or validation data
- This data set is commonly used to test new techniques and get a better understanding of the machine learning concepts



What are We looking to Accomplish?

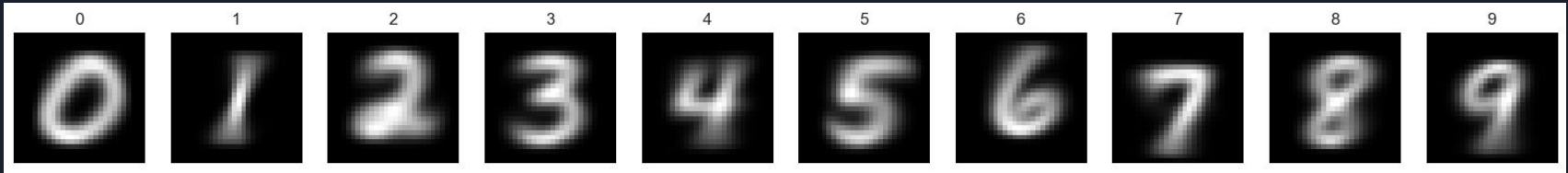
- How accurate can we get our computer vision algorithms?
- Can we apply the models we built to handwritten digits that are not included in the training set?
- What are the best optimized models to perform the MNIST digit recognition?
- Are there some models that are better at identifying numbers than others?



(Flaticon, 2020)

Exploratory Data Analysis

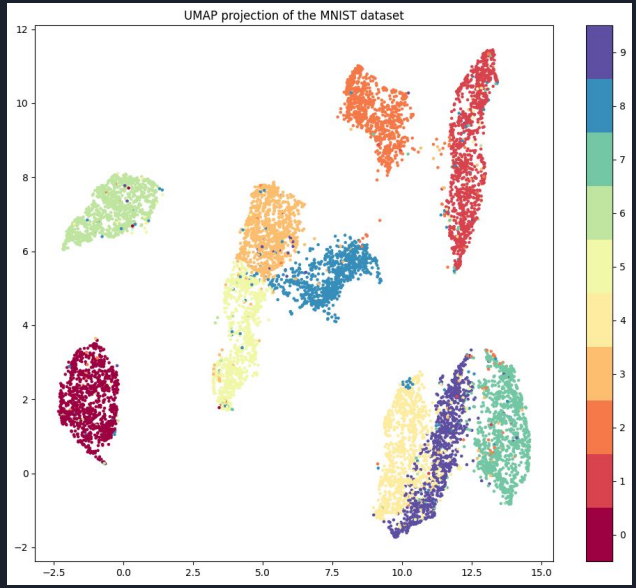
- Below is the example of what all of the numbers in the MNIST Data set look like based off of pixel mean
- We wanted to see what the average ‘shape’ was of every number to get a general idea of dataset



Peeking into The Data Set

- We wanted to be visualize some of the data points in a U-map
    - A U-map is used for dimension Reduction and being able to get a view of all of the categories of data that are being fed into the model

- We can see the color correlates, with that data points, and how they are all clustered together (*Scatterplot With Multiple Semantics — Seaborn 0.13.0 Documentation*, n.d.)

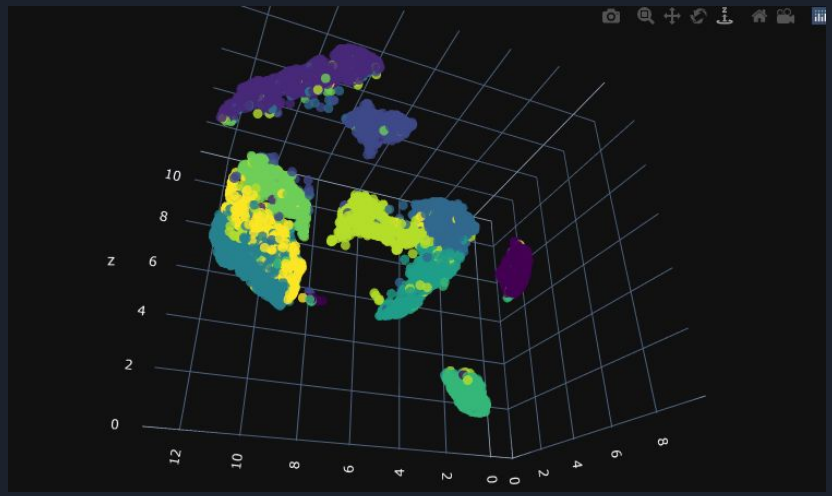


Three Dimensional U-Map

- Advantages of the 3D Umap
  - Better View of Data points
  - Exploring of the odd points
  - Able to get a better grip of the data

- But we can explore our data even further and see unique examples in each of the clusters

- In a JuPyter notebook we could interact with the chart to view the data (*T-SNE*, n.d.)



Why We Picked the Models We did

SIMPLICITY vs COMPLEXITY

- Model Set Up
- Training time
- Accuracy

TRADITIONAL vs NON-TRADITIONAL

- Commonly used ML Models
- Powerful Image Classification Models

KNN

SVM

Neural Networks

K-Neighbors Model Outcome

Test Results

Accuracy: 0.9712857142857143

	precision	recall	f1-score	support
0	0.98	0.99	0.98	1343
1	0.96	0.99	0.98	1600
2	0.97	0.97	0.97	1380
3	0.97	0.96	0.97	1433
4	0.97	0.96	0.97	1295
5	0.98	0.97	0.97	1273
6	0.98	0.99	0.99	1396
7	0.97	0.98	0.97	1503
8	0.99	0.94	0.96	1357
9	0.96	0.95	0.96	1420
accuracy			0.97	14000
macro avg	0.97	0.97	0.97	14000
weighted avg	0.97	0.97	0.97	14000

We were pleased with the results on the MNIST data set.

- 97% Accuracy
- High F1 Score
- High Macro Average
- High Weighted Average

Support Vector Model Outcome

Test Results

Accuracy: 0.963

	precision	recall	f1-score	support
0	0.99	0.98	0.98	1343
1	0.98	0.99	0.98	1600
2	0.95	0.96	0.95	1380
3	0.96	0.95	0.96	1433
4	0.96	0.96	0.96	1295
5	0.97	0.96	0.96	1273
6	0.97	0.98	0.97	1396
7	0.92	0.97	0.95	1503
8	0.97	0.95	0.96	1357
9	0.96	0.93	0.95	1420
accuracy			0.96	14000
macro avg	0.96	0.96	0.96	14000
weighted avg	0.96	0.96	0.96	14000

We were pleased with the results of the SVM model on the MNIST data set.

- 96.3% Accuracy
- High F1 Score
- High Macro Average
- High Weighted Average

Although just below the KNN model in accuracy, it did slightly better in classifying specific digits like ‘0’ and ‘1.’

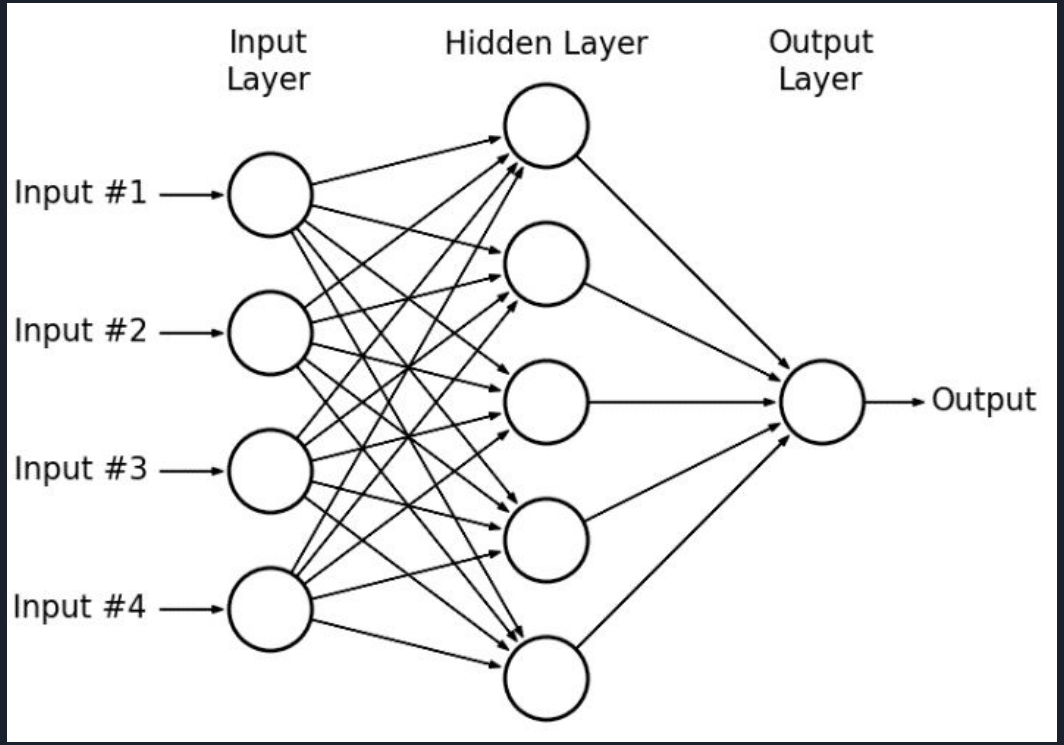
Both of these models came highly optimized out of the box and did not require further optimization.

Multilayer Perceptron (MLP)

Our project employed two MLP models. Both Models included:

- Flattened Vector Input (784)
- A hidden layer with 64 nodes
- Another hidden layer with either 16 or 32 nodes.
- RELU activations for hidden neurons
- Softmax activation for the final layer.
- Trained for 100 epochs with batch size 32.

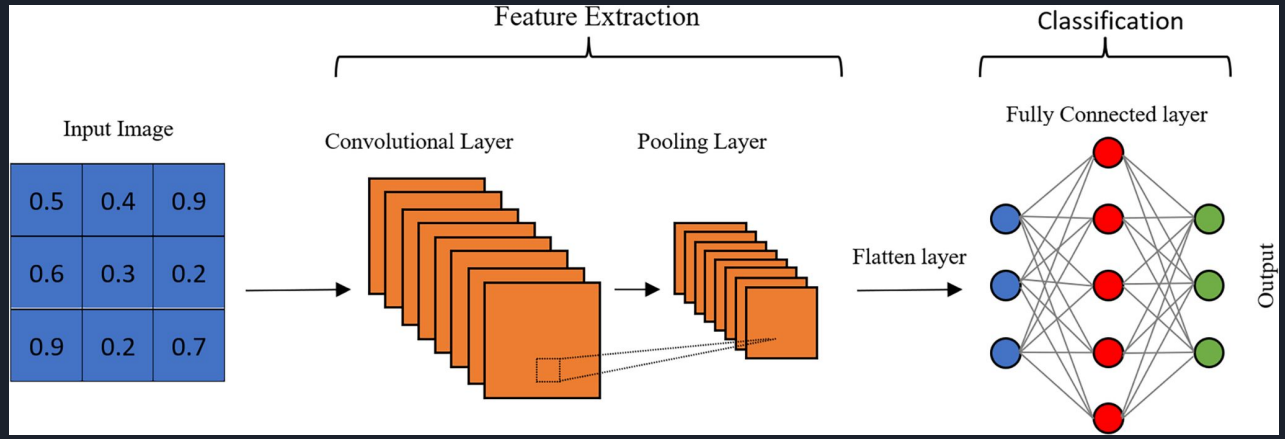
The models where named “64-16” and “64-32” based on their hidden layer node counts.



Convolutional Neural Network (CNN)

Our group created 3 Convolutional Neural Networks each containing:

- A base convolutional layer with 32 filters with a kernel of size 3x3 and a Max Pooling Layer with pool size 2x2.
- Two of the networks extend the convolutional architecture by adding 1 16 filter layer of kernel size 3x3 and 3 16 filter layers of kernel size 3x3. (Their names are 32f, 32-16f, and 32-16x3f)
- All networks flatten the Max Pool output and apply a final logistic regression layer with softmax activation. All networks utilize ReLU activations in the hidden layers.



Neural Network Implementation

- Created model_api.py file for training and testing.
- tensorflow.keras and numpy
- LR =3e-3, L2 normalization = 1e-5 with a basic Stochastic Gradient Descent Optimizer.
- Cross Entropy Loss.

```python
# Retrieve Data Set
def get_standard_data():
    (x_train,y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()

    # Normalize Data Set
    x_train = x_train / 255.0
    x_test = x_test / 255.0

    # Use only the training input for normalization.. avoid info leak
    mu = np.mean(x_train)
    std = np.std(x_train)

    x_train = (x_train - mu) / std
    x_test = (x_test - mu) / std


    # Return normalization factors to apply to other datasets!
    return x_train, y_train, x_test, y_test, mu, std
```

```python
def NN_classifier(hidden_neurons, hidden_layers, lr=3e-3, gamma=1e-5, verbose=False):
    '''
        hidden_neurons: the number of neurons per hidden layer (int)
        hidden_layers: the number of hidden layers in the MLP (int)
        lr: the learning rate.. it is the coefficient on the gradient descent algorithm (float)
        gamma: the l2 normalization coefficient, it affects the strength of the l2 penalty on loss. (float)
        verbose: if true, print model summary() (boolean)
    '''
    X = keras.layers.Input(shape=(28,28)) #Take a flat vector 28*28
    flatten = keras.layers.Flatten()(X)

    #Create an initial hidden layer
    #Relu is the standard activation function, we will use it for all but the final layer.

    hidden = keras.layers.Dense(64, activation="relu", kernel_regularizer=keras.regularizers.l2(gamma))(flatten) #Use Default Initialization
    for h in range(hidden_layers - 1):
        hidden = keras.layers.Dense(hidden_neurons, activation="relu", kernel_regularizer=keras.regularizers.l2(gamma))(hidden)

    # Classification Output
    classifier = keras.layers.Dense(10, activation="softmax", kernel_regularizer=keras.regularizers.l2(gamma))(hidden)

    # Create model and optimizer now!
    model = keras.models.Model(inputs=X, outputs=classifier)

    # Lets use the standard Stochastic Gradient Descent method
    optimizer = keras.optimizers.SGD(learning_rate=lr)

    # For our model we will use categorical crossentropy for classification loss
    # sparse in this context is just the format we choose to have our labels in (a single number denoting the category)
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    if verbose:
        model.summary()

    return model
```

```python
def CNN_classifier(filters, hidden_layers, lr=3e-3, gamma=1e-5, verbose=False):
    '''
        filters: the number of 3X3 filters per CNN layer (int)
        hidden_layers: the number of hidden layers in the MLP (int)
        lr: the learning rate.. it is the coefficient on the gradient descent algorithm (float)
        gamma: the l2 normalization coefficient, it affects the strength of the l2 penalty on loss. (float)
        verbose: if true, print model summary() (boolean)
    '''
    X = keras.layers.Input(shape=(28,28, 1)) #Take an image vector (greyscale)

    #Relu is the standard activation function, we will use it for all but the final layer.

    #Create an initial hidden convolutional layer
    conv = keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu', kernel_regularizer=keras.regularizers.l2(gamma))(X)

    #No padding, so conv should be (26, 26, 32)
    for h in range(hidden_layers - 1):
        conv = keras.layers.Conv2D(filters, kernel_size=(3,3), activation='relu', kernel_regularizer=keras.regularizers.l2(gamma))(conv)

    max_pool = keras.layers.MaxPooling2D(pool_size=(2,2))(conv)
    # Boureau, Y.L.; Ponce, J.; LeCun, Y. A theoretical analysis of feature pooling in visual recognition. In Proceedings of the 27th Inter

    latent = keras.layers.Flatten()(max_pool) # Maybe we can study this latent space vector generated only by the Convolutional Layers?

    # Classification Output
    classifier = keras.layers.Dense(10, activation="softmax", kernel_regularizer=keras.regularizers.l2(gamma))(latent)

    # Create model and optimizer now!
    model = keras.models.Model(inputs=X, outputs=classifier)

    # Lets use the standard Stochastic Gradient Descent method
    optimizer = keras.optimizers.SGD(learning_rate=lr)

    # For our model we will use categorical crossentropy for classification loss
    # sparse in this context is just the format we choose to have our labels in (a single number denoting the category)
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    if verbose:
        model.summary()

    return model
```

Neural Network Results on MNIST

Model	Number of Parameters	Test Accuracy	Test Loss
64-16 MLP	51450	97.12%	0.1058853418
64-32 MLP	52650	97.21%	0.1072596163
32f CNN	**54410**	98.37%	**0.05450668931**
32-16f CNN	27994	98.48%	0.06588789076
32-16fx3 CNN	25594	**98.67%**	0.08556618541

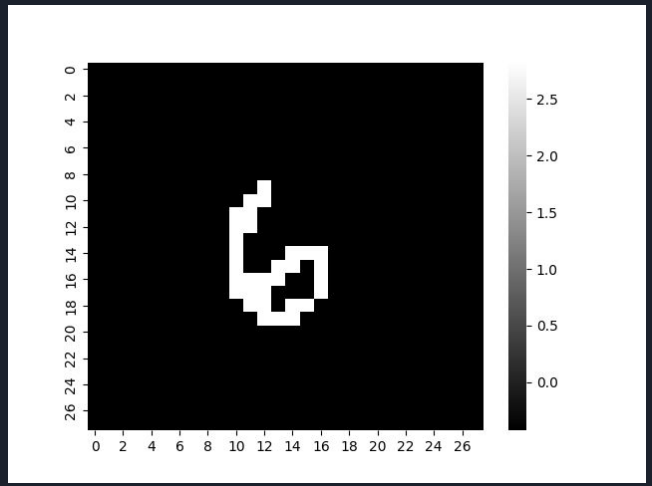
As one would expect, the deepest, most complicated model achieved the highest Test Accuracy. Notice the MLP performances. Despite their larger number of parameters, are lackluster compared to CNNs. This is probably due to the positional information used by the parameterized sliding kernels in CNNs.

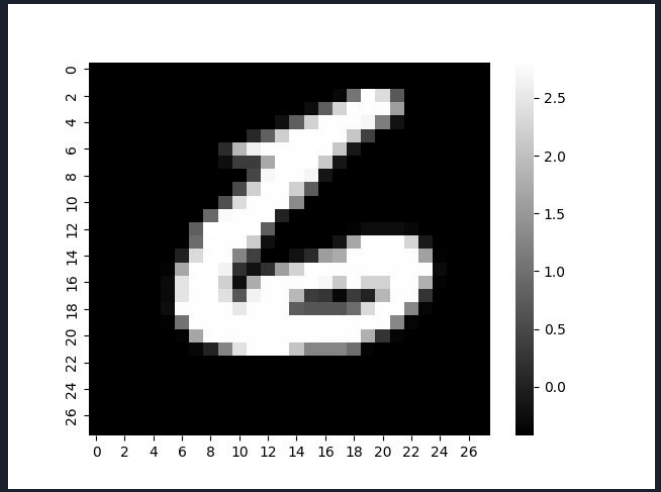
Interestingly, the model with the highest number of parameters (32f CNN has the least amount of kernel parameters but the most logistic regression parameters due to its higher dimensional output after Max Pool) also has the lowest Test Loss. This highlights the subtle difference between loss and accuracy.

These results may suggest that the networks may need to be regularized further. However, these great results, in theory, should indicate some level of performance on custom handwritten digits.

Handwritten Digit Testing

- 20 out of 30 Handwritten Digits used in testing (Gabe's and Philip's Handwriting)
- Parker's Digits had trouble being processed to reasonably clear 28x28 images.
- Each handwritten digit was created using a black sharpie on white paper
- Using the standardization scalers in model_api.py, and Parker's png to numpy array methods we produced a set of 20 digits that could be fed to our trained models. (Gabe's 6 on the left, MNIST 6 on the right)





Handwritten Digit Testing (Continued)

All models had pathetically low accuracy on the handwritten digits provided by Gabe and Philip.

Despite the large amount of preprocessing on the custom data to ensure similar pixel magnitudes, none of the models could generalize to the new data.

Neural networks in general proved to be much better than traditional methods like K-Nearest Neighbors and Support Vector Machine.

The simplest MLP model outperformed all 4 other neural networks on the new task.

Model	Handwritten Digit Accuracy
KNN	0.00%
SVM	25.00%
64-16 MLP	**50.00%**
64-32 MLP	45.00%
32f CNN	45.00%
32-16f CNN	45.00%
32-16fx3 CNN	45.00%

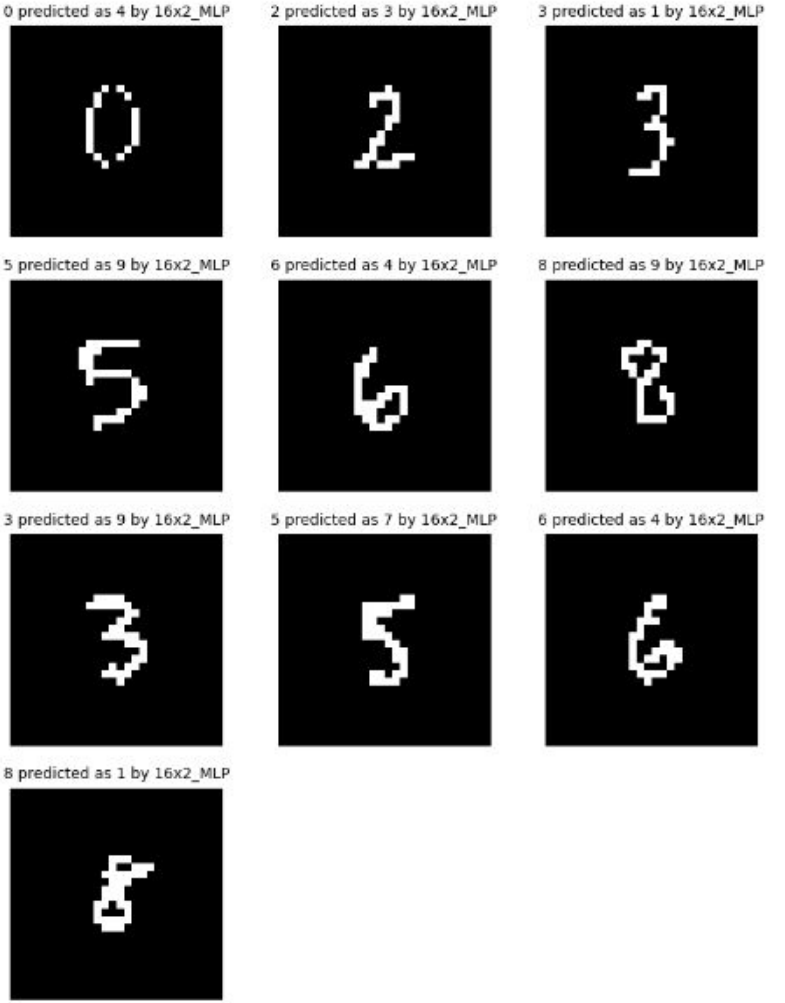
Best Model Analysis

Our group decided to get a better look at how the best model misclassified the custom digits.

The picture on the right depicts how the 64-16 MLP misclassified the custom digits.

The model seems to always classify 1's correctly, though there is one false positive in the last example.

We theorize the models struggled with the custom data due to the differences in digit sizes, digit orientation, and brushes compared to the original MNIST data.



Conclusion

Though every model trained in our study achieved high accuracy on MNIST test data, all models failed to generalize to the custom dataset.

The custom dataset already had many preprocessing measures to ensure the data resembled the profile of MNIST digits. Our team made measures to convert pngs to grayscale and utilize the original scalar normalizations from the training data to give the models a fighting chance!

Though our study could have included a more thorough set of models and regularization techniques to ensure better generalization, it is easy to conclude that MNIST test accuracy does not reflect the generalizability of a model architecture.

Our team concludes that MNIST should be seen as the bare minimum preliminary test for a more serious study of a model's performance. If the model can't learn MNIST, don't bother with more complicated data. But 98% accuracy on MNIST doesn't mean very much either!

References

1.6. Nearest neighbors. (n.d.-a). Scikit-learn. https://scikit-learn.org/stable/modules/neighbors.html#classification

3.3. Metrics and scoring: quantifying the quality of predictions. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/model_evaluation.html#model-evaluation

Ai, L. (2023, January 10). K-Nearest Neighbors (KNN) - Learn AI - Medium. *Medium*. https://medium.com/@divakar1591/k-nearest-neighbors-knn-f1677d989049

Anderson, A. (n.d.). *The difference between computer vision and machine vision*. https://www.clearview-imaging.com/en/blog/the-difference-between-computer-vision-and-machine-vision#:~:text=A%20machine%20vision%20system%20uses,of%20a%20larger%20machine%20system.

Guide. (n.d.). TensorFlow. https://www.tensorflow.org/guide

References

Hamilton, D. (2020). kNN vs. SVM: A comparison of algorithms. US Forest Service Research and Development. https://www.fs.usda.gov/research/treesearch/62328

joblib.load — joblib 1.4.dev0 documentation. (n.d.). https://joblib.readthedocs.io/en/latest/generated/joblib.load.html

LeCun, Y., Bengio, Y., & Hinton, G. (2015, May 28). Review - Department of Computer Science, University of Toronto. Nature Deep Review. doi:10.1038/nature14539

McInnes, L., Healy, J. J., Saul, N., & Großberger, L. (2018). UMAP: Uniform manifold Approximation and Projection. *Journal of Open Source Software*, *3*(29), 861. https://doi.org/10.21105/joss.00861

Practices of Science: False Positives and False Negatives | manoa.hawaii.edu/ExploringOurFluidEarth. (n.d.). https://manoa.hawaii.edu/exploringourfluidearth/chemical/matter/properties-matter/practices-science-false-positives-and-false-negatives

References

Pupale, R. (2023, November 9). Support Vector Machines(SVM) — an overview - towards data science. *Medium*. https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

Statquest with Josh Starmer. (n.d.). Home [Channel]. YouTube. Retrieved October 11, 2020, from https://www.youtube.com/channel/UCtYLUTtgS3k1Fg4y5tAhLbw

Team, K. (n.d.). *Keras: Deep Learning for humans*. https://keras.io/

Towards Data Science. (2001, December 9). *Towards data science*. https://towardsdatascience.com/

T-SNE. (n.d.). https://plotly.com/python/t-sne-and-umap-projections/

Boureau, Y.-L., Ponce, J., & LeCun, Y. (2010, June 25). A Theoretical Analysis of Feature Pooling in Visual Recognition - ENS. NYU Scholars. https://www.di.ens.fr/willow/pdfs/icml2010b.pdf