CS 4563 Final Project

Liheng Cao, Aaron Hsieh

April 28, 2021

1 Overview

In this project, we used the sign_mnist dataset.

There are 27,455 examples in the training set and 7,172 examples in the testing set. For machine learning algorithms, each training example is a 1-D array with length 784. For the convolutional neural network, each training example is a single-channel 28x28 image.

The dataset consists of images of American Sign Langauge (ASL) hand gestures for each of the 26 alphabet letters (a-z). However, since the letters *j* and *z* require motions, they were not included in the dataset. Each letter is represented as a label from 0-25, for which we use one-hot encoding for categorical classification.

The goal of the problem was to find the best machine learning and deep learning algorithm to classify the ASL gestures for alphabets (excluding j and z). In this project, we examine the performance of logistic regression, support vector machines, and convolutional neural networks, and the effects of varying various hyperparameters.

2 Data Preprocessing

Two types of scaling from the scikit-learn library [5] were used: MinMax, and Standard.

The Minmax Scaler takes the data and rescales it so that all of the data now fits inside the range $(0, 1)$. This is represented with the following:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

The Standard Scaler takes the data and puts the mean μ to 0 and the variance σ to 1, which is equivalent to setting the standard deviation to 1. This is represented with the following:

```
X_mu = X.mean(axis=0)
X_std = X.std(axis=0)
X = (X–X_mu) / X_std
```

3 Results

3.1 Logistic Regression (LogReg)

For LogReg, the type of regularization, the amount of regularization, and the type of scaling were varied. However, the model using L1 Regularization did not fit and predict in a reasonable amount of time, so it will not be included.

The sci-kit learn (sklearn) library was used to for fitting and prediction. The code was inspired by the HW 5 Programming assignment: I looped through various levels of regularization, stored the accuracies, and then plotted them. Since the libary does not allow regularization to be turned off, a very high value of C was used to simulate no regularization.

3.1.1 Data scaling

One thing that was varied was the type of data scaling or scaler that was used. For each scaler, we also fitted and calcuated the accuracies using different levels of regularization.

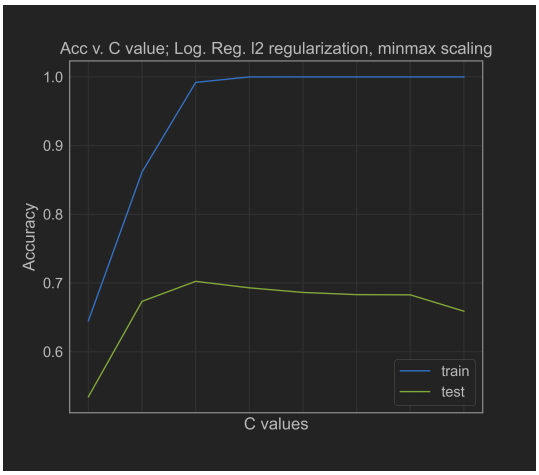

Figure 1: accuracy vs C values

C	train accuracy	test accuracy
0.001	64.51 %	53.44%
0.01	86.16%	67.33%
0.1	**99.2%**	**70.25%**
1.0	99.99%	69.3%
10.0	100.0%	68.63%
100.0	100.0%	68.31%
1000.0	100.0%	68.28%
10000000.0	100.0%	65.9%

Table 1: accuracy vs C

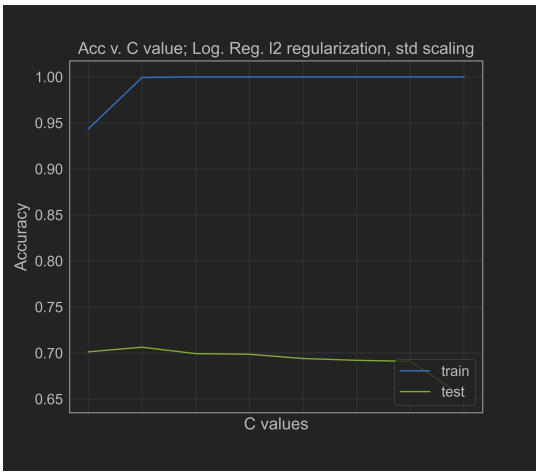

Figure 2: accuracy vs C values

C	train accuracy	test accuracy
0.001	94.36%	70.13%
0.01	**99.93%**	**70.64%**
0.1	100.0%	69.94%
1.0	100.0%	69.87%
10.0	100.0%	69.41%
100.0	100.0%	69.21%
1000.0	100.0%	69.1%
10000000.0	100.0%	65.25%

Table 2: accuracy vs C

3.2 Support Vector Machines (SVM)

For SVM, the the amount of regularization, and the type of kernel (and the degree of the polynomial kernel when it was used) were varied.

Like LogReg, the code was inspired by the HW 5 Programming assignment.

3.2.1 Linear Kernel

The linear kernel is typically used for data that is linearly separable. Like the other models, various levels are regularization and their accuracies were tested. It is of the form [5]

$$\langle x, x' \rangle$$

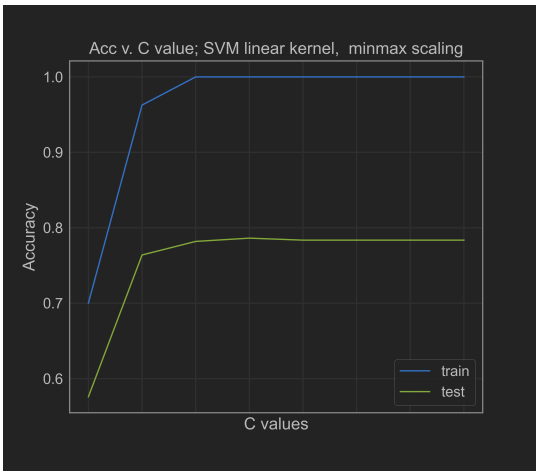

Figure 3: accuracy vs C values

C	train accuracy	test accuracy
0.001	70.02%	57.6%
0.01	96.27%	76.39%
0.1	100.0%	78.19%
1.0	**100.0%**	**78.62%**
10.0	100.0%	78.36%
100.0	100.0%	78.36%
1000.0	100.0%	78.36%
10000000.0	100.0%	78.36%

Table 3: accuracy vs C

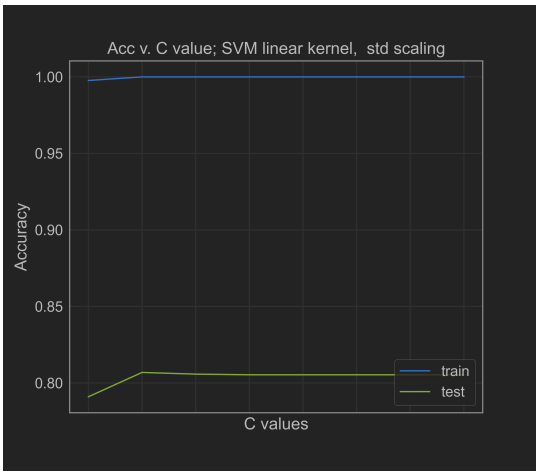

Figure 4: accuracy vs C values

C	train accuracy	test accuracy
0.001	99.77%	79.1%
0.01	**100.0%**	**80.69%**
0.1	100.0%	80.58%
1.0	100.0%	80.53%
10.0	100.0%	80.53%
100.0	100.0%	80.53%
1000.0	100.0%	80.53%
10000000.0	100.0%	80.53%

Table 4: accuracy vs C

3.2.2 Radial Basis Function Kernel

The RBF Kernel is often used for data whose classification is based on proximity. Like the other models, various levels are regularization and their accuracies were tested. It is in the form

$$\exp\left(-\gamma \|x - x'\|\right)$$

where γ is a constant [5].

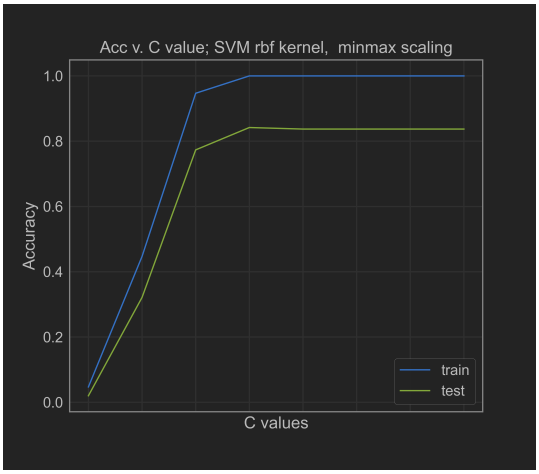

Figure 5: accuracy vs C values

C	train accuracy	test accuracy
0.001	4.71%	2.01%
0.01	44.7%	32.14%
0.1	94.69%	77.34%
1.0	**100.0%**	**84.19%**
10.0	100.0%	83.71%
100.0	100.0%	83.71%
1000.0	100.0%	83.71%
10000000.0	100.0%	83.71%

Table 5: accuracy vs C

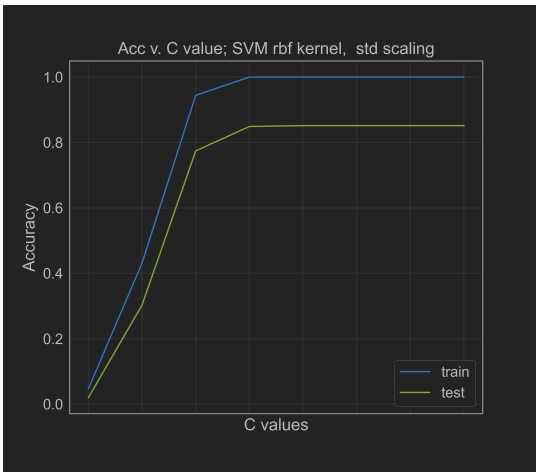

Figure 6: accuracy vs C values

C	train accuracy	test accuracy
0.001	4.71%	2.01%
0.01	43.14%	30.31%
0.1	94.41%	77.43%
1.0	100.0%	84.89%
10.0	**100.0%**	**85.17%**
100.0	100.0%	85.17%
1000.0	100.0%	85.17%
10000000.0	100.0%	85.17%

Table 6: accuracy vs C

3.2.3 Polynomial Kernel

The polynomial kernel has an additional parameter, degree. Like the other models, various levels are regularization and their accuracies were tested. The kernels are feature transformations, which are used to model more complicated decision boundary.

1. The polynomial kernel with degree 1 is very similar to the linear kernel. It is in the form

$$\left(\gamma\langle x, x'\rangle + r\right)^1$$

where γ, r are constants [5].

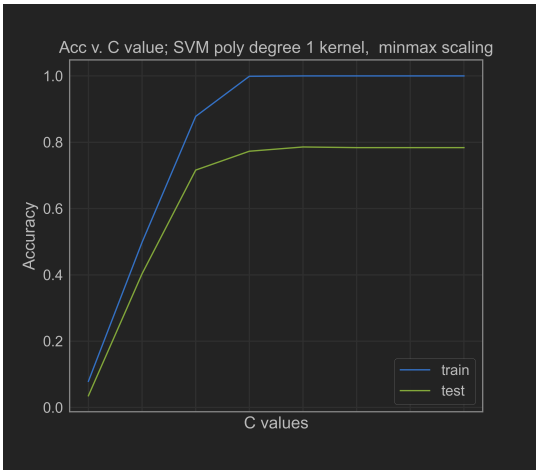

Figure 7: accuracy vs C values

C	train accuracy	test accuracy
0.001	7.9%	3.54%
0.01	49.81%	40.32%
0.1	87.8%	71.6%
1.0	99.89%	77.29%
10.0	**100.0%**	**78.57%**
100.0	100.0%	78.37%
1000.0	100.0%	78.37%
10000000.0	100.0%	78.37%

Table 7: accuracy vs C

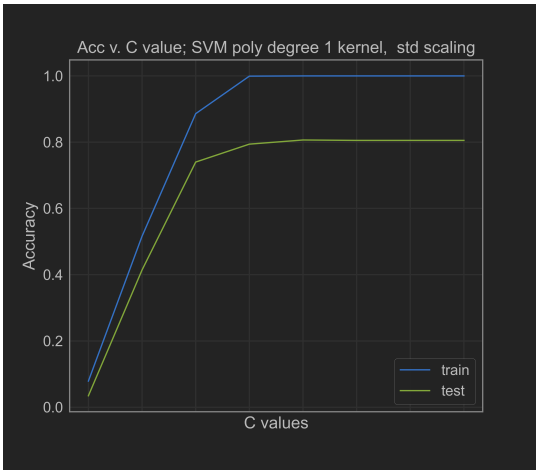

Figure 8: accuracy vs C values

C	train accuracy	test accuracy
0.001	7.89%	3.46%
0.01	51.61%	41.42%
0.1	88.6%	74.0%
1.0	99.93%	79.41%
10.0	**100.0%**	**80.66%**
100.0	100.0%	80.53%
1000.0	100.0%	80.53%
10000000.0	100.0%	80.53%

Table 8: accuracy vs C

2. It is in the form

$$\left(\gamma\langle x, x'\rangle + r\right)^2$$

where γ, r are constants [5].

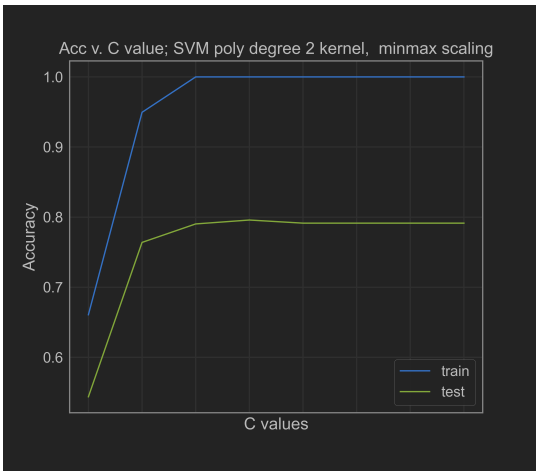

Figure 9: accuracy vs C values

C	train accuracy	test accuracy
0.001	66.08%	54.38%
0.01	94.95%	76.39%
0.1	100.0%	79.03%
1.0	**100.0%**	**79.59%**
10.0	100.0%	79.14%
100.0	100.0%	79.14%
1000.0	100.0%	79.14%
10000000.0	100.0%	79.14%

Table 9: accuracy vs C

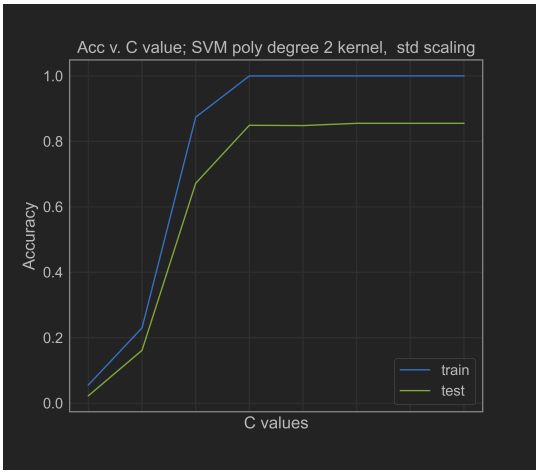

Figure 10: accuracy vs C values

C	train accuracy	test accuracy
0.001	5.58%	2.27%
0.01	23.02%	16.17%
0.1	87.39%	67.19%
1.0	99.98%	84.91%
10.0	100.0%	84.83%
100.0	**100.0%**	**85.51%**
1000.0	100.0%	85.51%
10000000.0	100.0%	85.51%

Table 10: accuracy vs C

3. The degree 3 polynomial kernel adds even more complexity to the model. It is in the form

$$\left(\gamma\langle x, x'\rangle + r\right)^3$$

where γ, r are constants [5].

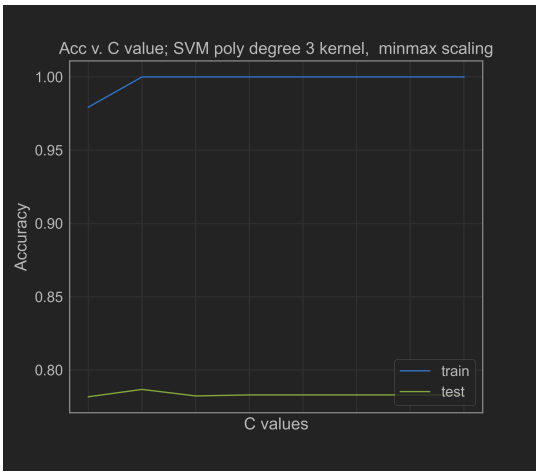

Figure 11: accuracy vs C values

C	train accuracy	test accuracy
0.001	97.94%	78.18%
0.01	**100.0%**	**78.68%**
0.1	100.0%	78.23%
1.0	100.0%	78.31%
10.0	100.0%	78.31%
100.0	100.0%	78.31%
1000.0	100.0%	78.31%
10000000.0	100.0%	78.31%

Table 11: accuracy vs C

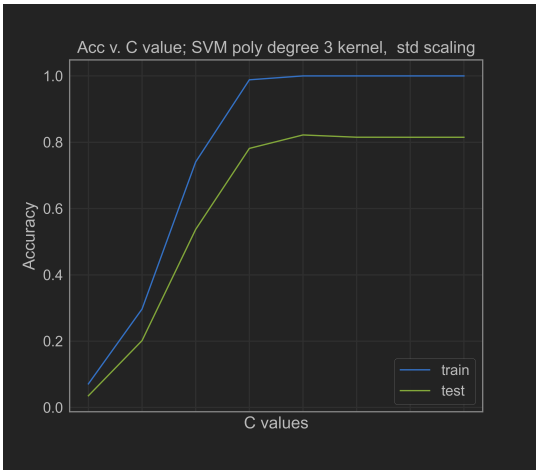

Figure 12: accuracy vs C values

C	train accuracy	test accuracy
0.001	7.13%	3.54%
0.01	29.7%	20.18%
0.1	74.07%	53.72%
1.0	98.82%	78.14%
10.0	**100.0%**	**82.19%**
100.0	100.0%	81.51%
1000.0	100.0%	81.5%
10000000.0	100.0%	81.5%

Table 12: accuracy vs C

4. The degree 4 polynomial kernel builds on top of the complexity that the degree 3 polynomial kernel adds. It is in the form

$$\left(\gamma\langle x, x'\rangle + r\right)^4$$

where γ, r are constants [5].

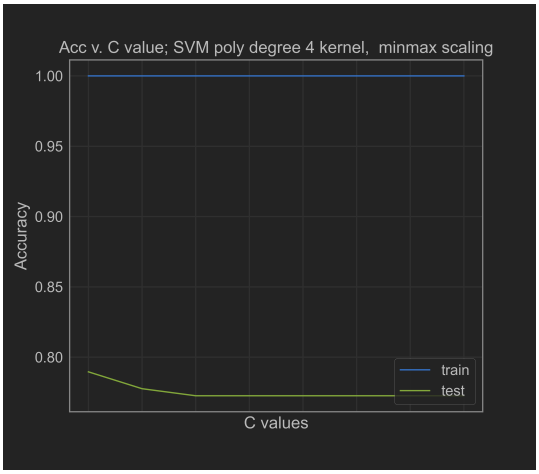

Figure 13: accuracy vs C values

C	train accuracy	test accuracy
0.001	**100.0%**	**78.96%**
0.01	100.0%	77.76%
0.1	100.0%	77.26%
1.0	100.0%	77.26%
10.0	100.0%	77.26%
100.0	100.0%	77.26%
1000.0	100.0%	77.26%
10000000.0	100.0%	77.26%

Table 13: accuracy vs C

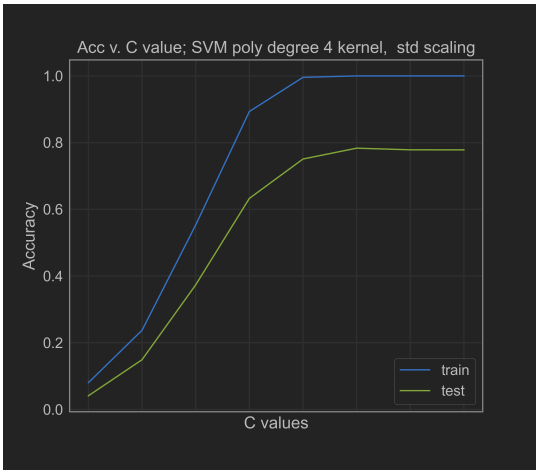

Figure 14: accuracy vs C values

C	train accuracy	test accuracy
0.001	8.03%	4.1%
0.01	23.74%	14.9%
0.1	55.31%	37.35%
1.0	89.33%	63.27%
10.0	99.57%	75.08%
100.0	**100.0%**	**78.33%**
1000.0	100.0%	77.84%
10000000.0	100.0%	77.83%

Table 14: accuracy vs C

3.3 Convolutional Neural Networks (CNN)

For the CNN, we used the following architecture[4]:

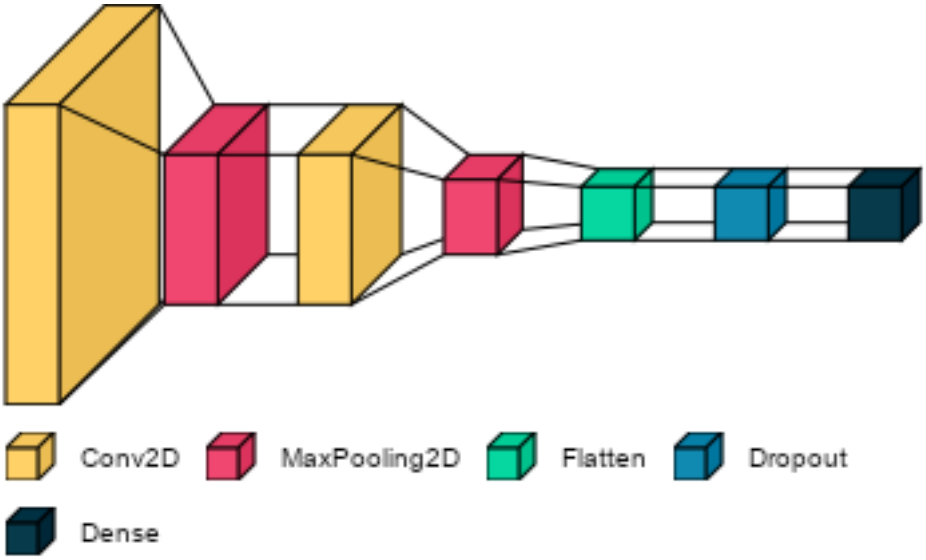

Figure 15: CNN Architecture

The model consists of a Conv2D layer followed by a MaxPooling2D layer, repeated twice, a Flatten layer, a Dropout layer, and then a Dense layer. We used Tensorflow/Keras to create and run the model [3].

Initial testing suggested that preprocessing did not affect the test accuracy performance, so the orignal scale (0-255) was used.

In a dropout layer, each neuron in the previous layer is set to 0 with a specified probability. It is a regularization method that approximates training a large number of neural networks with different architectures in parallel [1]. In this case, the dropout rate was 0.5. By setting around half of the neurons to 0, it is almost like temporarily removing half of the neurons from the neural network. Effectively, it forces the model to find more meaningful connections since it has less features to work with and promotes independence between individual neurons.

Additionally, during training, we used Tensorflow Keras's EarlyStopping. EarlyStopping monitors the validation error during training. If the validation error did not decrease for 10 epochs, training was halted. Training a neural network "too long" can risk modeling the noise, so the validation error is used to monitor the model's ability to generalize. It prevents over fitting [2].

For the CNN, we varied the learning rate, batch size, and regularization term (l1 and l2). Regularization was applied to the last dense layer. Each of these hyperparameters was varied individually while others were held constant. When a hyperparameter was held constant, we used the following values:

Learning rate: 0.002

Batch size: 256

Regularization term (λ): 0

3.3.1 Learning Rate (α)

The learning rate α determines how big of a step we want to take during each gradient descent step. α varied on a logarithmic scale from -5.0 to -0.5 ($10^{-5.0}$ to $10^{-0.5}$). The results are shown below:

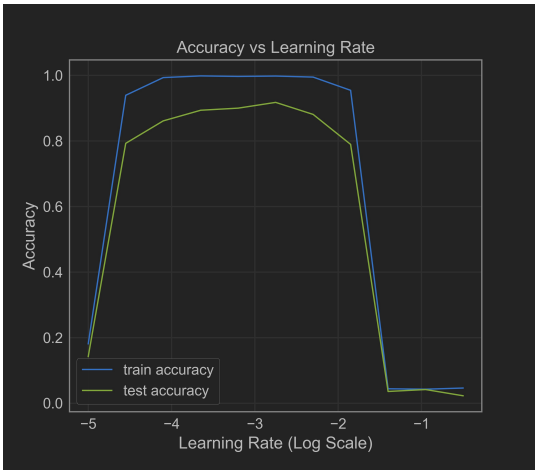

Figure 16: test accuracy vs α

α (log)	train accuracy	test accuracy
-5.0	18.15%	14.26%
-4.55	93.95%	79.29%
-4.1	99.35%	86.14%
-3.65	99.86%	89.39%
-3.2	99.7%	90.03%
-2.75	**99.82%**	**91.79%**
-2.3	99.5%	88.12%
-1.85	95.47%	78.97%
-1.4	4.42%	3.6%
-0.95	4.29%	4.18%
-0.5	4.64%	2.29%

Table 15: test accuracy vs α

3.3.2 Batch Size (t)

The batch size t determines how many examples are viewed before making a gradient descent step. Here, mini-batch gradient descent is used, where the batch size is smaller than the number of training examples. t varied on a logarithmic scale from 2 to 10 (2^2 to 2^{10}). The effect of different batch sizes is shown below:

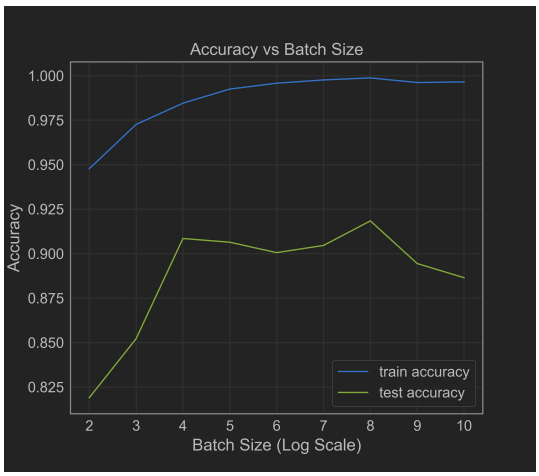

Figure 17: test accuracy vs t

t (log)	train accuracy	test accuracy
2.0	94.77%	81.9%
3.0	97.26%	85.22%
4.0	98.46%	90.85%
5.0	99.25%	90.64%
6.0	99.58%	90.06%
7.0	99.77%	90.46%
8.0	**99.88%**	**91.84%**
9.0	99.62%	89.44%
10.0	99.66%	88.65%

Table 16: test accuracy vs t

3.3.3 Regularization Term (λ)

The regularization term is used to determine how much the model should be penalized for having larger weights. λ first started at 0 (effectively $10^{-\infty}$) and then varied on a logarithmic scale from -3.0 to -0.5 ($10^{-3.0}$ to $10^{-0.5}$).

$$Cost_{l1} = J(W, X) + \lambda \sum_{i=1}^{M} |W_i| \tag{3.1}$$

$$Cost_{l2} = J(W, X) + \lambda \sum_{i=1}^{M} (W_i)^2 \tag{3.2}$$

The results for varying λ are shown below:

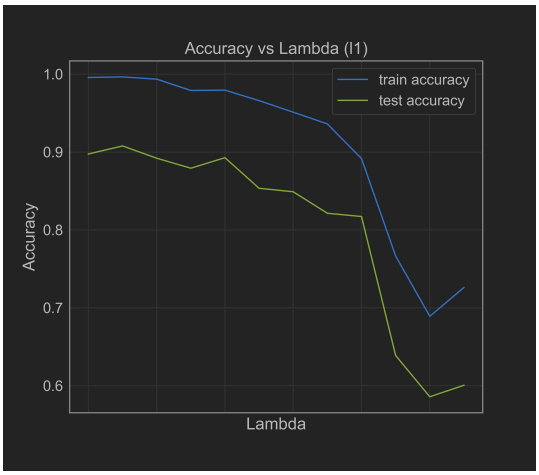

Figure 18: test accuracy vs λ (l1)

λ (l1) (log)	train accuracy	test accuracy
-inf	99.58%	89.75%
-3.0	**99.65%**	**90.78%**
-2.75	99.36%	89.21%
-2.5	97.91%	87.92%
-2.25	97.95%	89.28%
-2.0	96.6%	85.35%
-1.75	95.12%	84.9%
-1.5	93.61%	82.14%
-1.25	89.16%	81.72%
-1.0	76.65%	63.87%
-0.75	68.91%	58.58%
-0.5	72.6%	60.05%

Table 17: test accuracy vs λ (l1)

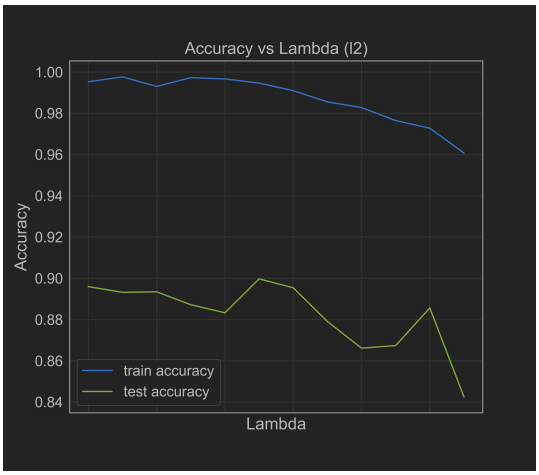

Figure 19: test accuracy vs λ (l2)

λ (l2) (log)	train accuracy	test accuracy
-inf	99.53%	89.6%
-3.0	99.77%	89.32%
-2.75	99.3%	89.35%
-2.5	99.73%	88.72%
-2.25	99.67%	88.33%
-2.0	**99.47%**	**89.98%**
-1.75	99.1%	89.54%
-1.5	98.56%	87.91%
-1.25	98.28%	86.61%
-1.0	97.65%	86.74%
-0.75	97.28%	88.57%
-0.5	96.07%	84.26%

Table 18: test accuracy vs λ (l2)

3.4 Overall

Model	highest test accuracy
LogReg	70.64%
SVM (Linear)	80.69%
SVM (RBF)	85.17%
SVM (poly degree 2)	85.51%
CNN (lr)	91.79%
CNN (batch size)	**91.84%**
CNN (l1 and l2)	90.78%

Table 19: Overall Results.

4 Conclusions

4.1 Logistic Regression

4.1.1 Data scaling

For the model with the minmax scaler, the best test accuracy was when C=0.1. Since sklearn interprets C = $\frac{1}{\lambda}$ [5], a higher value of C means a lower value of λ, which in turn means less regularization. This matches our results: We get better training accuracy with less regularization. Since the testing accuracy decreases afterwards, we can say that overfitting occurs in this model starting at around C = 1. There was underfitting when there was too much regulariziation present, as the train accuracy for low C was very low.

For the model with the standard scaler, there was less underfitting initially, but underfitting was still present: This is shown by the lower relative traning accuracy. Like the model with minmax scaling, overfitting was present, as the testing accuracy dropped when values of C increased.

The best LogReg test accuracy came from the model using the standard scaler and C=0.01, with a test accuracy of 70.64%. It beat the LogReg model using the other scaler by less than 1 percent.

4.2 Support Vector Machines

4.2.1 Linear Kernel

Like the other models, underfitting occurs with a low C value. However, very little overfitting occured, as the test accuracy did not drop significantly when C was increased. This applies to both models, regardless of which scaler they used.

The linear kernel beats LogReg with with 80.69%. The best results of a certain type of model have come from the model that is using the standard scaler: the best LogReg and the best linear SVM have both comes from the variation that is using the standard scaler. For SVM using the linear kernel, this difference between the scalers' test accuracy is about 2%.

4.2.2 Radial Basis Function Kernel

Both models had problems with underfitting with low values of C, indicating high levels of bias. The problems with bias were so bad that the training accuracy with high regularization was about as good as randomly guessing (26 letters - {j, z} = 24 letters; 1/24 $\approx$ 4%). The testing accuracy with high regularization was even worse, at 2%. This is the worst out of the models presented so far. Fortunately, decreasing the amount of regularization remedied this problem. This model actually ended up giving the best test accuracy so far when the hyperparameters were adjusted, at 85.17%. This beats both the LogReg and Linear SVM models. Like the previous models, the standard scaler has gives better max testing accuracies than their minmax counterparts.

4.2.3 Polynomial Kernel

1. The polynomial with degree 1 gives very similar max testing accuracy to the linear kernel. However, the polynomial kernel of degree 1 has more problems with underfitting. Like

the RBF kernel, when the polynomial kernel with degree 1 has very high regularization, it's accuracies are close to that of randomly guessing.

2. The polynomial model with degree 2 is a more complex model, which likely adds variance and decreases bias. This reduced the underfitting for low values of C for the model using minmax scaling, but the model using the standard scaling still had problems with underfitting. The standard scaler had problems with accuracy when regularization was high. However, the standard scaler eventually had higher performance (test accuracy) when regularization was decreased.
3. Since the max testing accuracy is not as high as that of the degree 2 polynomial kernel model, this is likely too much complexity for our data. Our model is beginning to model the noise, rather than the actual features.
4. Similar to the situation with the degree 3 polynomial kernel model, the model that uses the degree 4 polynomial kernel has even worse performance. This supports the notion that the higher degree polynomial kernels are too complex for our dataset.

Out of the models using a polynomial kernel, the best testing accuracy was from the degree 2 model using the standard scaler. It had an accuracy of 85.51%. It also happens to be the model with the highest testing accuracy out of all the non neural network models.

4.3 Convolutional Neural Network

4.3.1 Learning Rate (α)

The best value for α was $10^{-2.75}$ with test accuracy 91.79%. This makes sense: at the lower and upper values of α, there is high bias. When it was too small, it is possible that the model got stuck in a local minima or simply did not have enough epochs to converge, resulting in underfit. This is because the steps are too small to escape local minima. When it was too big, it is possible that the model was not able to properly converge to a minima, also resulting in underfit. This is because the model takes steps that are too big to converge into a minima. Intermediate values of α helped reduce bias, improving generalization.

4.3.2 Batch Size (t)

The best value for t was $2^8 = 256$ with test accuracy 91.84%. For small batch sizes, the individual gradient descent steps for each batch might have been too sporadic, preventing the model from converging to a minima. This suggests high bias and underfitting. As we increased batch sizes, the training accuracy showed an overall upward trend while test accuracy started to decrease in larger batch sizes. This suggests that the model has high variance and is overfitting to the data. An intermediate value of t helped reduce bias and variance.

4.3.3 Regularizaiton Term (λ)

The best value (including l1 and l2) for λ was 10^{-3} from l1 regularization with test accuracy 90.78%. The results were expected: increasing λ decreased training and test accuracy, which suggests high bias. By selecting the best value for λ, we were able to reduce variance while minimizing the sacrifice of higher bias.

4.4 Final Thoughts

Overall, the CNN yielded the best test accuracy. The features learned from the convolutional filters have better generalization ability. One possible reason is because the filters are looking for a specific feature independent of its location. The position of the hand could shift, and it would still identify the same features. On the other hand, the ML models assume the location of each part of the hand. For example, it might assume that the tips of fingers are always found in the upper right quadrant.

One way to further improve the CNN's performance is to apply image augmentation. For each image, we could shift the image in a random direction or flip it horizontally and vertically. Not only does this increase our dataset size, but it also introduces more variation in the training set, which helps with reducing variance.

References

[1] Jason Brownlee. *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. Aug. 2019. URL: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/.

[2] Jason Brownlee. *A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks*. Aug. 2019. URL: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/.

[3] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[4] paulgavrikov. *visualkeras*. https://github.com/paulgavrikov/visualkeras/. 2021.

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.