Classification of handwritten indian digits

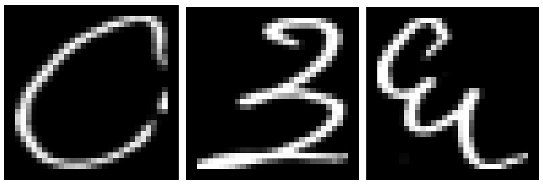
Jakub Podkomórka 253231
Kateryna Bodko 253844

As in the title, the goal that we were trying to achieve in our project was to classify handwritten indian digits from 0 to 9. To do that we were using “Devanagari Handwritten Character Dataset”.

The dataset consists of 46 classes of characters with 2000 examples each. The dataset is already split into training and test sets with the ratio of 85% train examples and 15% test examples. Dataset’s structure is as follows:

- DevanagariHandwrittenCharacterDataset:
    - Test:
        - digit_0:
            - 4574.png
            - 4584.png
            - …
        - digit_1:
            - 4750.png
            - …
        - …
    - Train:
        - digit_0:
            - 4558.png
            - 4559.png
            - …
        - digit_1:
            - 4732.png
            - …
        - …

Handwritten characters pictures are being stored in the separate folders so after discarding characters to leave only digits for classification we have 3000 pictures of test digits and 17000 pictures of train digits. Below the example pictures of 0, 3 and 6 digits can be seen.



The images are in the png format, they are in gray scale resolution is 32x32 although the actual digit is centered within 28 by 28 pixel, padding of 2 pixel is added on all four sides of actual character.

Here is the URL to the dataset:
https://archive.ics.uci.edu/dataset/389/devanagari+handwritten+character+dataset

In our project we approach the potential solutions in two ways. In one approach we treat individual pixels as features and in the other one we pass pictures to the Convolutional Neural Network. Now we will delve into both approaches and show results.

Pixels to features approach

1. **Data preparation**

The script starts by traversing through the dataset folders and reading the images. Each image is converted to grayscale and flattened so that every row in the dataset represents all 1024 pixels of the image (32x32) treated as individual features. After every image is added to the dataset, the corresponding label is also added to the label dataset. This is done for both the test and train datasets. The datasets are then shuffled. We finish that step with 4 lists of following shapes:

X_train: (17000, 1024)
y_train: (17000)
X_test: (3000, 1024)
y_test: (3000)

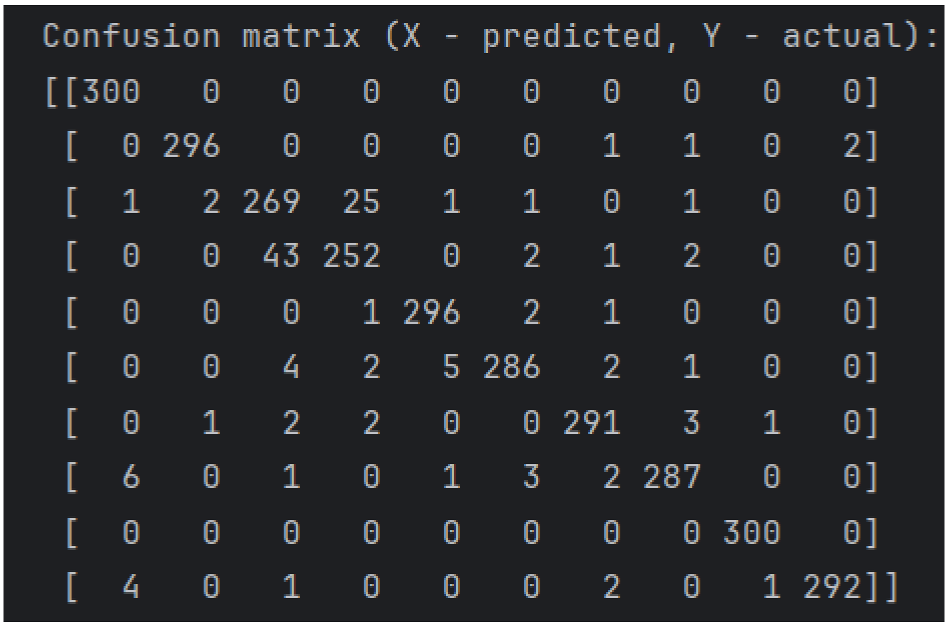
When the datasets are ready, we proceed to the SVM model.

2. **SVM models**

We tried two different SVM models. First was the basic SVM model from scikit-learn library with linear kernel. After the training we could see the following results:

Accuracy of the SVM model: 0.9563333333333334

F1 score: 0.9563301346657066

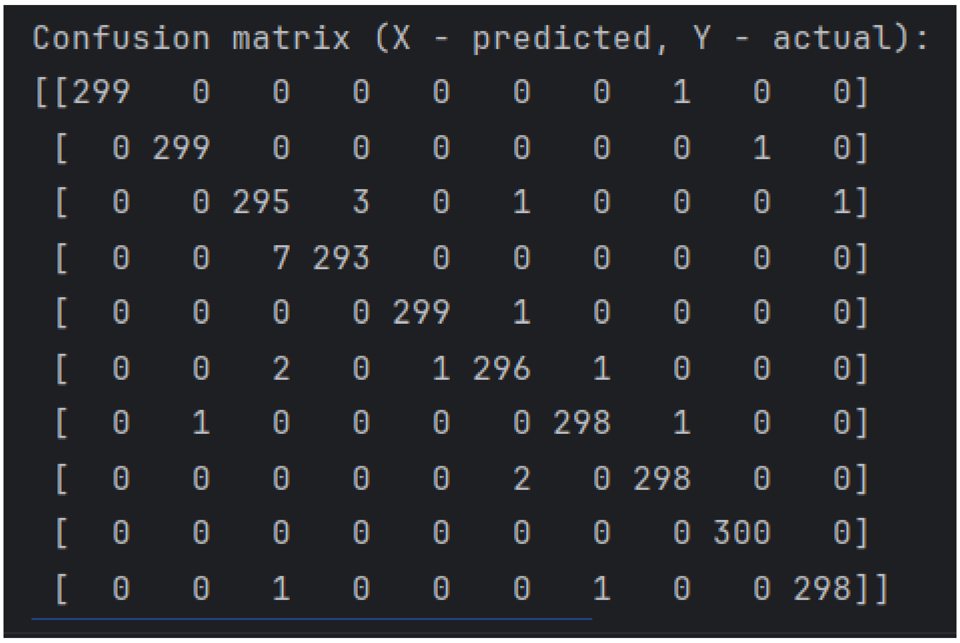


```
Confusion matrix (X - predicted, Y - actual):
[[300   0   0   0   0   0   0   0   0   0]
 [  0 296   0   0   0   0   1   1   0   2]
 [  1   2 269  25   1   1   0   1   0   0]
 [  0   0  43 252   0   2   1   2   0   0]
 [  0   0   0   1 296   2   1   0   0   0]
 [  0   0   4   2   5 286   2   1   0   0]
 [  0   1   2   2   0   0 291   3   1   0]
 [  6   0   1   0   1   3   2 287   0   0]
 [  0   0   0   0   0   0   0   0 300   0]
 [  4   0   1   0   0   0   2   0   1 292]]
```

Then we created a pipeline with StandardScaler and radial basis function kernel. StandardScaler standardizes features by removing the mean and scaling to unit variance. The radial basis function (RBF) kernel, also known as the Gaussian kernel, is the default kernel for Support Vector Machines in scikit-learn. It measures similarity between two data points in infinite dimensions and then approaches classification by majority vote. After training such SVM pipeline the results were as follows:

Accuracy of the SVM model with a pipeline: 0.9916666666666667

F1 score: 0.9916753064908953

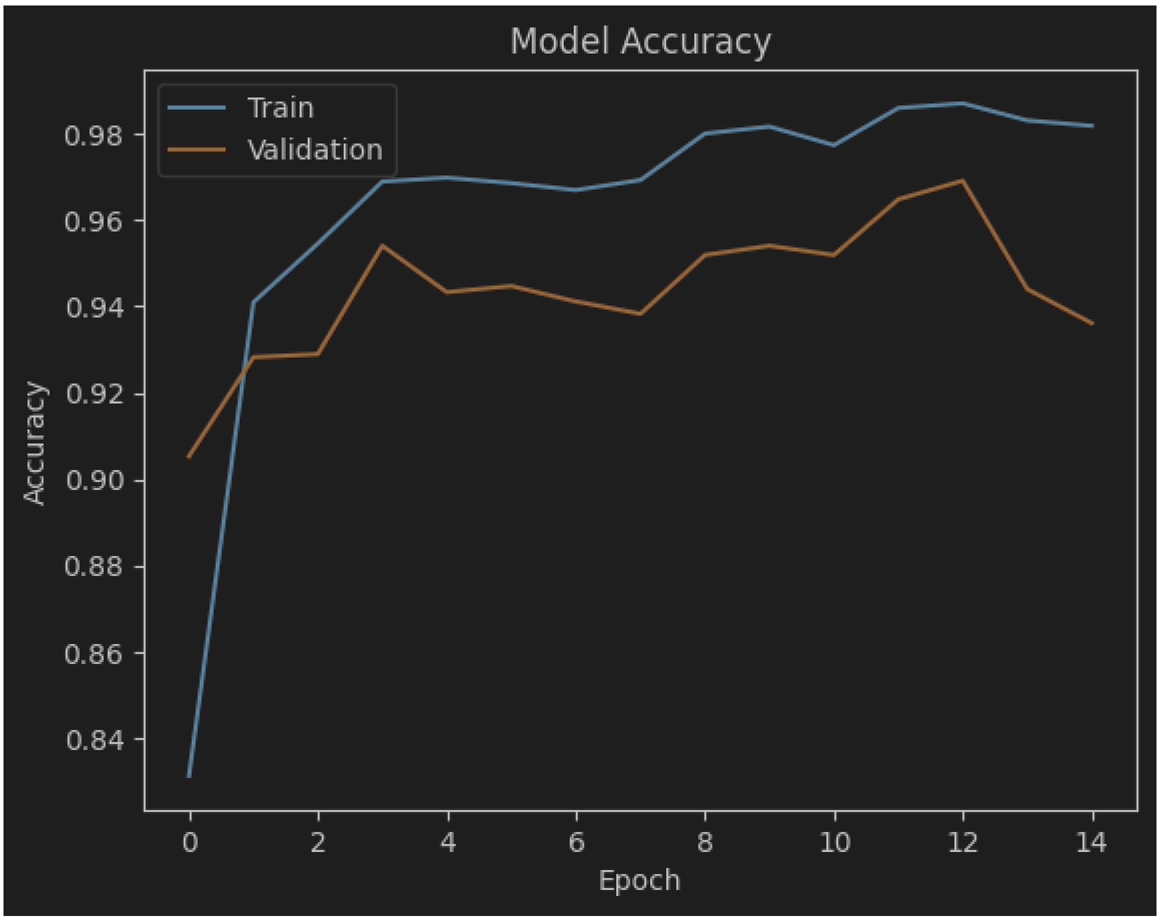


```
Confusion matrix (X - predicted, Y - actual):
[[299   0   0   0   0   0   0   1   0   0]
 [  0 299   0   0   0   0   0   0   1   0]
 [  0   0 295   3   0   1   0   0   0   1]
 [  0   0   7 293   0   0   0   0   0   0]
 [  0   0   0   0 299   1   0   0   0   0]
 [  0   0   2   0   1 296   1   0   0   0]
 [  0   1   0   0   0   0 298   1   0   0]
 [  0   0   0   0   0   2   0 298   0   0]
 [  0   0   0   0   0   0   0   0 300   0]
 [  0   0   1   0   0   0   1   0   0 298]]
```

Both accuracy and f1 score values were higher by 3.5 percentage points in the approach with StandardScaler and non linear kernel and so far that type of Machine Learning approach was the best one. Let’s now cover the neural networks

3. **Neural networks**

We tried 3 different neural networks for this approach. First was a basic neural network model with two 2 hidden layers with 512 and 256 neurons respectively. It was created using TensorFlow. The model was compiled with the Adam optimizer, sparse categorical cross-entropy loss, and sparse categorical accuracy metric. The model was trained on the training data with a batch size of 32 and a validation split of 0.2. The model's accuracy is evaluated on the test data. Here are the results:



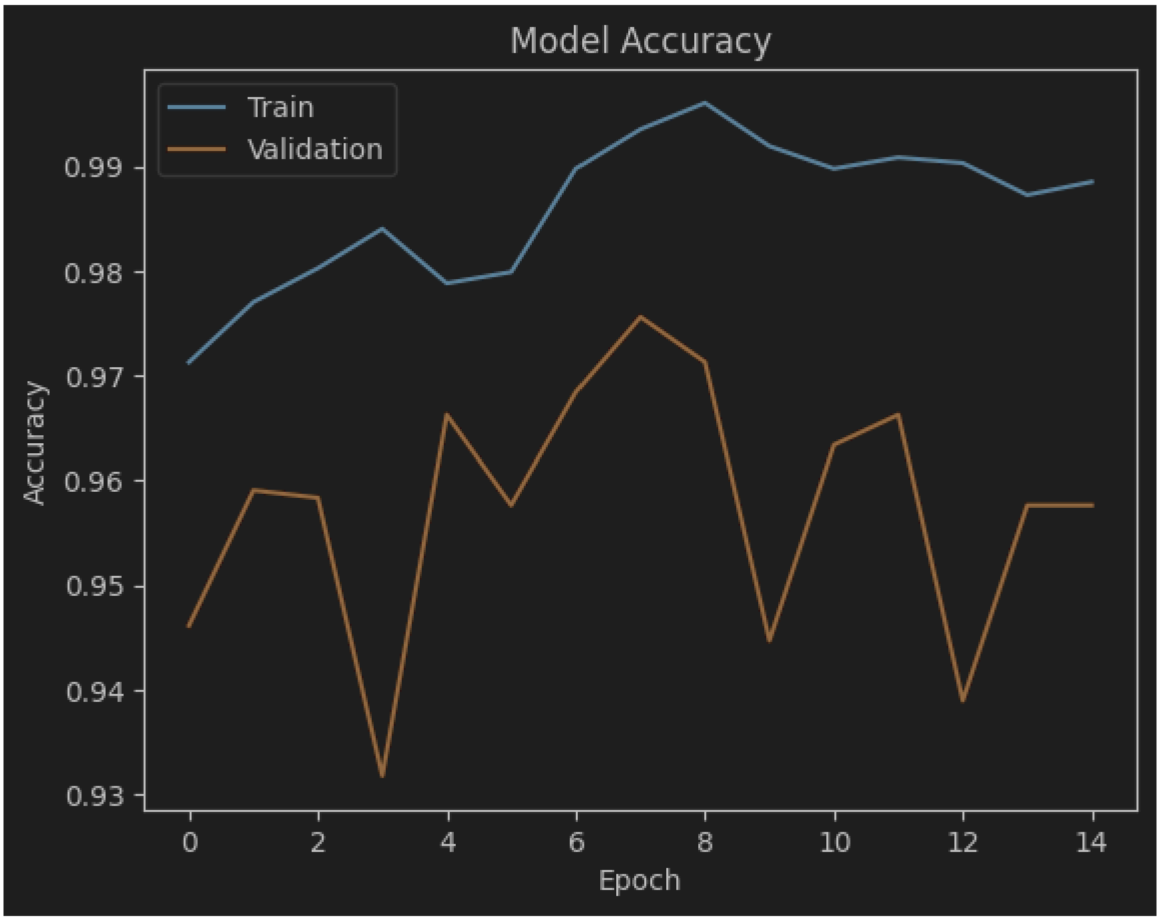
Max train accuracy values throughout history: 0.9870736002922058

The epoch with the highest train accuracy: 13

Max validation accuracy values throughout history: 0.969131350517273

The epoch with the highest validation accuracy: 13

Other neural network that we tried learning in this approach was almost the same as the previous but we added the dropout layers. Layers of that kind randomly set input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1 / (1 - rate) such that the sum over all inputs is unchanged. Results of that network are as follows:



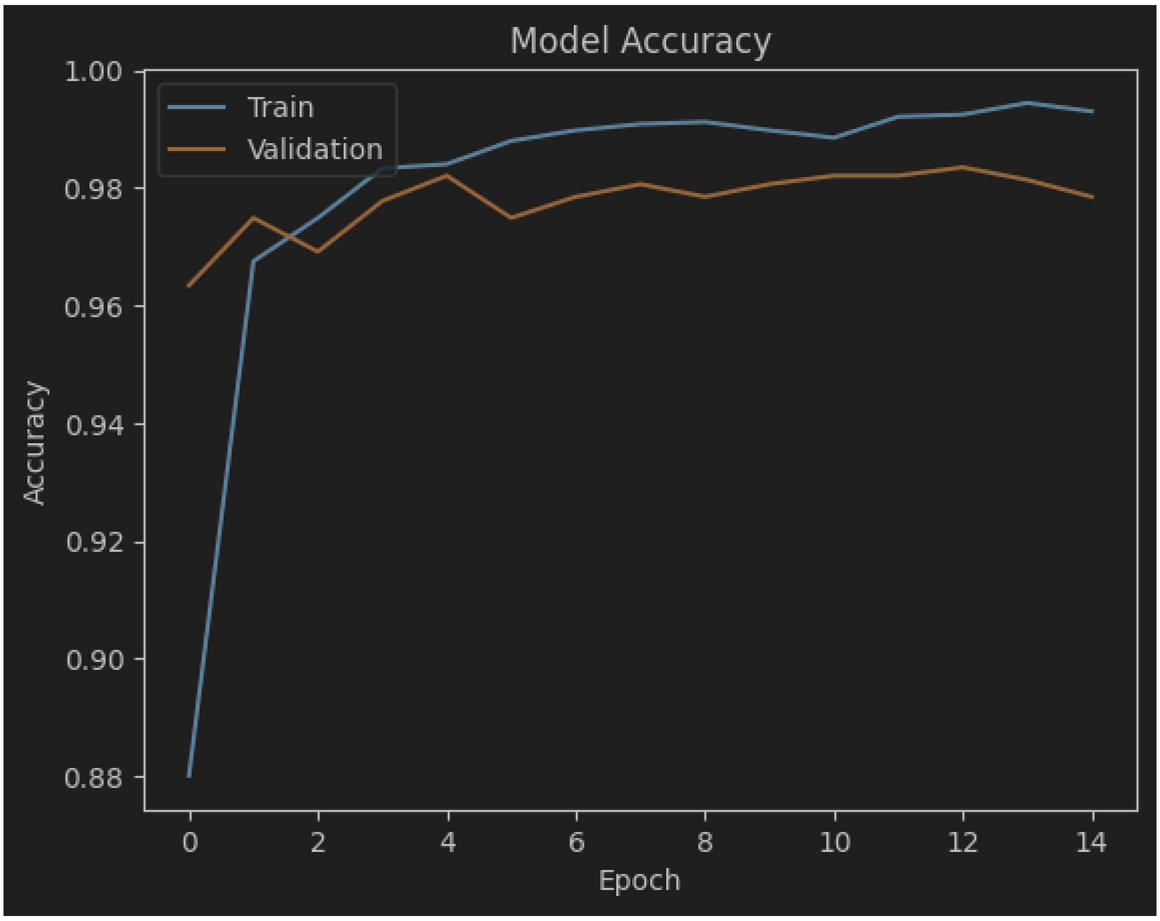
Max train accuracy values throughout history: 0.9960502982139587

The epoch with the highest train accuracy: 9

Max validation accuracy values throughout history: 0.9755922555923462

The epoch with the highest validation accuracy: 8

The last model we tried in this approach was a neural network with an additional normalization of inputs and batch normalization layers after each hidden layer. Normalization of inputs will shift and scale them into a distribution centered around 0 with standard deviation of 1. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. Importantly, batch normalization works differently during training and during inference. During training, the layer normalizes its output using the mean and standard deviation of the current batch of inputs. During inference, the layer normalizes its output using a moving average of the mean and standard deviation of the batches it has seen during training. As such, the layer will only normalize its inputs during inference after having been trained on data that has similar statistics as the inference data. Here are some results:



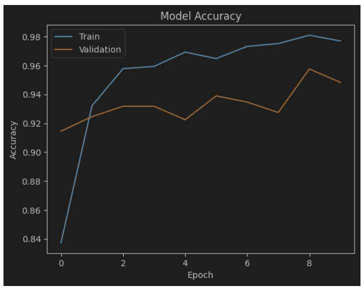
Max train accuracy values throughout history: 0.9944344758987427

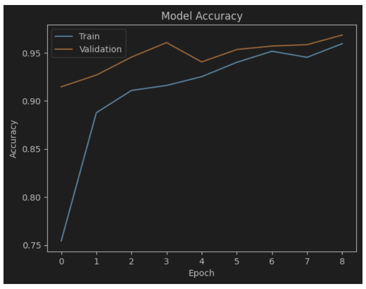
The epoch with the highest train accuracy: 14

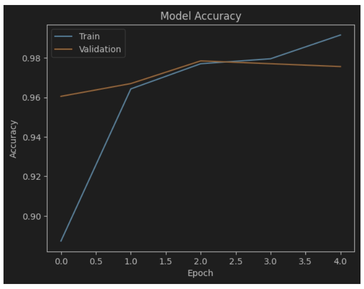
Max validation accuracy values throughout history: 0.9834888577461243

The epoch with the highest validation accuracy: 13

Finally, with such results, we train the networks once again playing around with the number of epochs to get the best results. As the starting point we are choosing the number of epochs based on the previous results where the best validation accuracy occurred besides the normalization one. The very similar accuracy was achieved in the 5th epoch so instead of taking 13th we are taking this one instead. At first we are satisfied by the results of the basic neural network and the one with normalization. They are getting test accuracy around 0.95 and 0.97. But we experiment with the number of epochs for the dropout model as it gives different results and is not consistent. Finally after more experiments we stay at the epochs number of 10 for the basic model, 9 for the dropout model and 5 for the normalization and dropout one. On the plots we can see that we eliminated the overfitting and accuracy fluctuations from the dropout model that were present after 8 and 9 epochs and the learning results are getting mostly better for all epochs across all models.







Neural network model	Test set accuracy
Basic	0.949
With dropout	0.963
Dropout and normalization	0.98

Convolutional approach

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision by their ability to automatically and efficiently learn spatial hierarchies of features from images. For the task of handwritten digit classification, CNNs are particularly well-suited due to several key reasons. CNNs use local receptive fields to detect patterns within small regions of an image. This means that a convolutional layer will look at a small segment of the image at a time and learn to detect features such as edges, corners, and textures. Traditional machine learning approaches often require manual feature extraction, which can be labor-intensive and may not capture the most discriminative features. Unlike fully connected networks, CNNs use a relatively small number of parameters, making them efficient in terms of memory and computational cost.

Data preparation

Before training the Convolutional Neural Network (CNN) for handwritten digit classification, it is crucial to prepare and preprocess the data properly.

Data augmentation is applied to the training dataset to artificially expand its size and diversity. By applying random transformations such as rotations, translations, scaling, shearing, and flipping, the model becomes more robust to variations and less likely to overfit.

Preprocessing involves setting up the `ImageDataGenerator` with rescaling and augmentation parameters. For training data, pixel values are rescaled to the range [0, 1], and the specified transformations are applied. For validation data, only rescaling is performed. Rescaling standardizes the input data, improving training stability and speed.

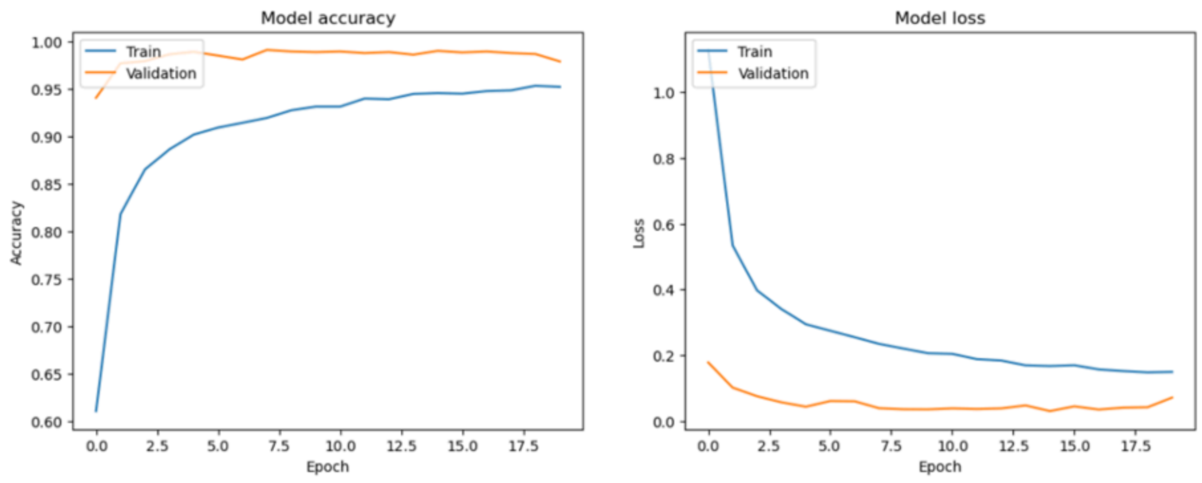
```
# Data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

Training

When tackling the problem of handwritten digit classification, it's essential to experiment with various Convolutional Neural Network (CNN) architectures to determine the most effective model. Different combinations of layers can significantly influence the model's ability to learn and generalize from the data. Several CNN architectures with varying complexity to find the optimal solution were explored:

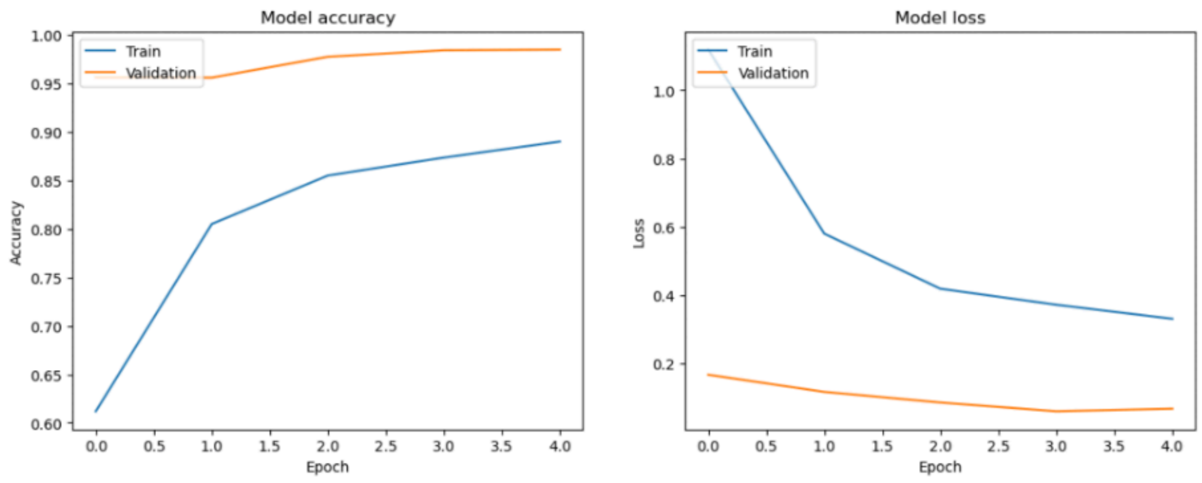
1. **Baseline Model.** It consists of a simple architecture with three convolutional layers followed by max-pooling layers and a dense layer before the output layer. Model

captures essential features with three layers of convolutions and pooling, followed by a dense layer for classification. The dropout layer helps prevent overfitting.



```
Test Accuracy: 97.95%, after 20 epochs.
```

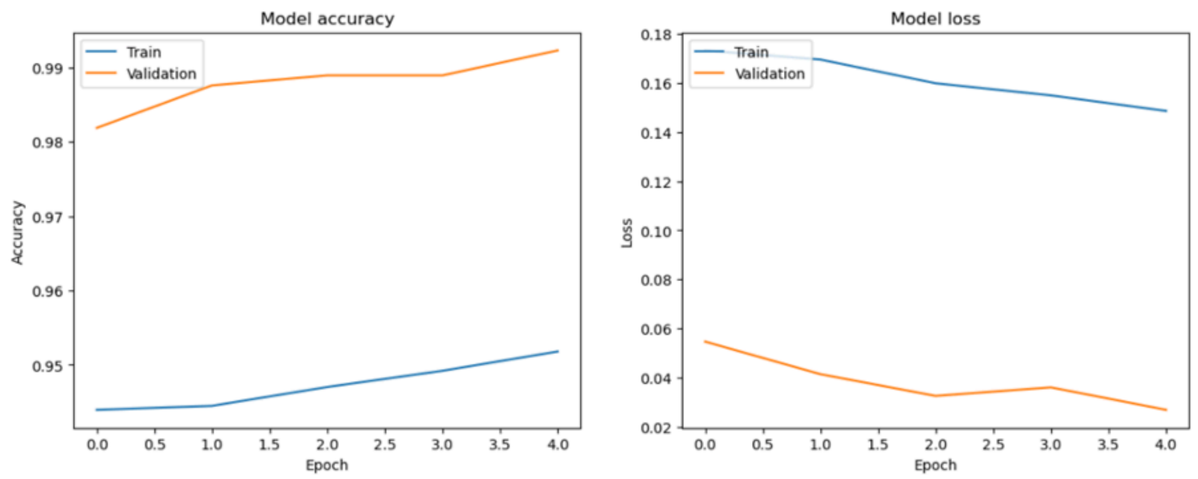
The training and validation accuracy plots show that the model achieves a high accuracy of 97.95% after 20 epochs, with the validation accuracy stabilizing early around 98%. The training loss decreases significantly, while the validation loss remains low and stable, indicating good generalization without overfitting. The decision to initially try 5 epochs was to quickly gauge the model's performance and ensure that it was learning correctly before committing to a longer training period.



```
Test Accuracy: 98.86%, after 20 epochs.
```

The model achieved a test accuracy of 98.86% after just 5 epochs, as shown in the provided plots. The training accuracy increases steadily, while the validation accuracy starts high and improves slightly, stabilizing around 98%. The training loss decreases significantly, and the validation loss remains low and stable, indicating excellent generalization. The high test accuracy suggests that the model is effectively learning the features of the handwritten digits and generalizing well to unseen data. **20 epochs may be excessive** for this specific model and dataset, as the model reaches high accuracy quickly and additional epochs may not provide significant benefits.

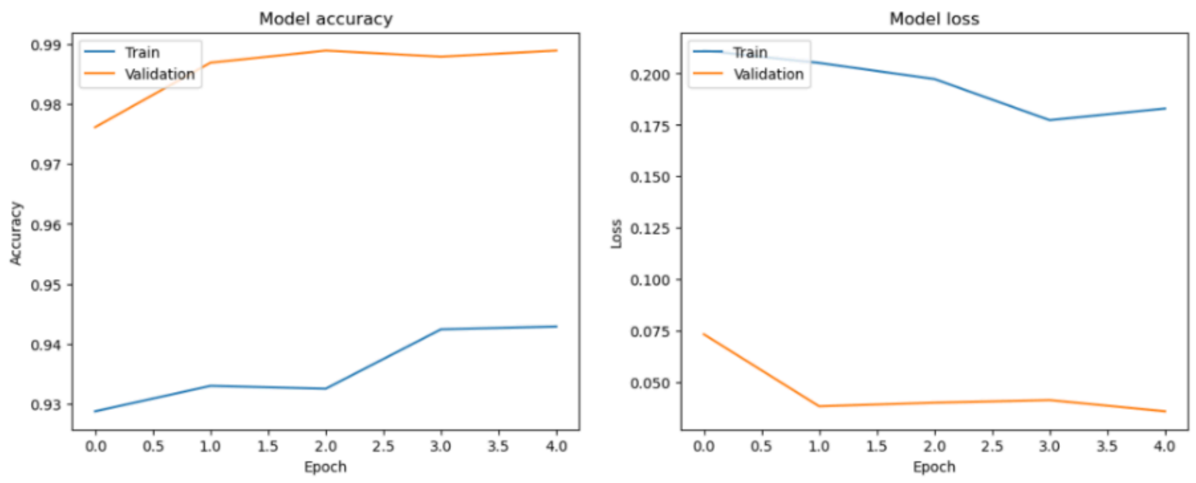
2. **Deeper Model.** A deeper model includes more convolutional layers, allowing the network to learn more complex features. By adding an extra convolutional layer, this model can capture finer details and more abstract features, potentially leading to better performance on complex datasets.



```
Test Accuracy: 99.23% after 5 epochs.
```

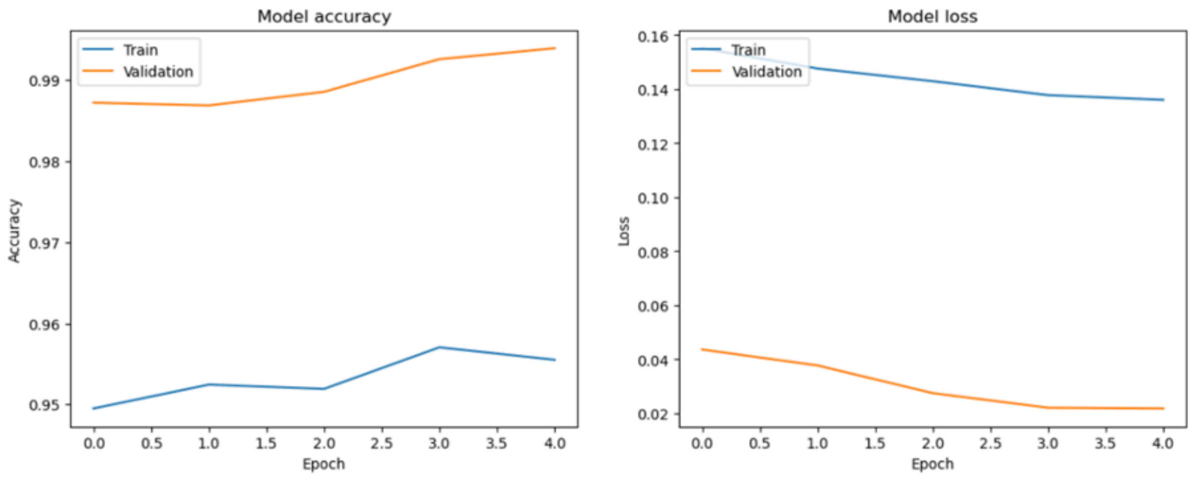
The deeper model's higher test accuracy of 99.23% suggests that the additional layers and complexity have allowed the model to learn more effectively from the data. This improvement indicates that, for this particular task and dataset, a deeper architecture is beneficial.

3. **Wider Model.** In this architecture, we increase the number of filters in each convolutional layer, making the network wider. Increasing the number of filters allows the model to learn more feature maps, which can be beneficial when dealing with diverse and intricate datasets.



```
Test Accuracy: 98.86%
```

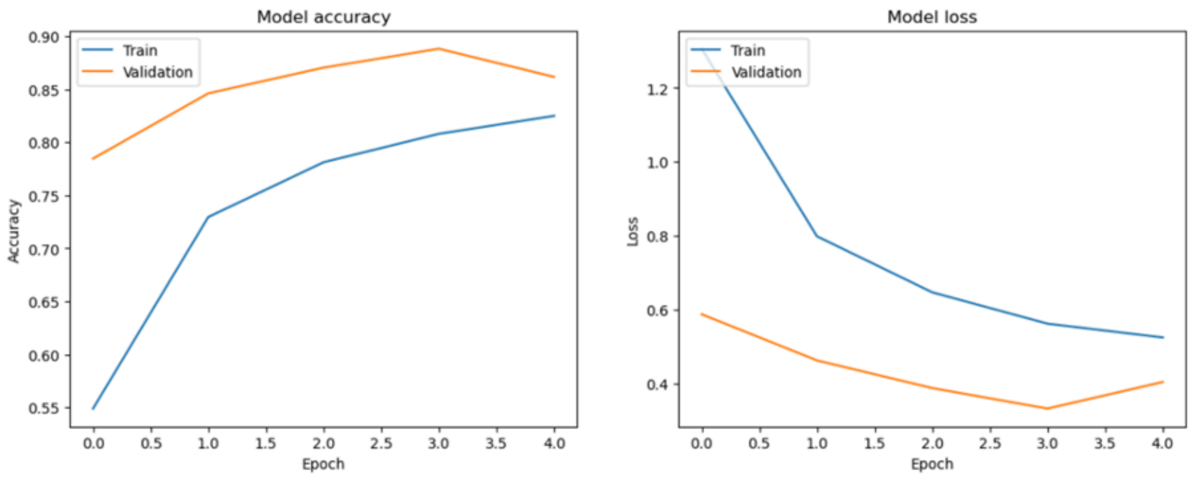
4. **Model with Batch Normalization.** Adding batch normalization helps stabilize and accelerate the training process by normalizing the inputs to each layer. Batch normalization can lead to faster training times and better performance by normalizing the activations in the network, reducing the risk of vanishing or exploding gradients.



```
Test Accuracy: 99.43%
```

The high test accuracy of 99.43% achieved with batch normalization demonstrates the effectiveness of this technique in improving model performance. By stabilizing the learning process and acting as a regularizer, batch normalization helps the model learn better and generalize well to new data.

5. **Simplest model.** The simplest CNN model with minimal layers while still leveraging the power of convolutional operations. A very basic CNN architecture that includes only one convolutional layer followed by a max-pooling layer, a flattening layer, and a dense layer for classification was also tested.



```
Test Accuracy: 86.02%
```

Conclusions

1. **Pixels to features approach:**

After testing 2 SVM models and 3 neural networks we can see that the SVM model with input normalization and radial basis function kernel is performing the best. It achieved both accuracy on the test data and f1 score of approximately 99.17% and consistently outperformed other models that were tested in this approach. The results may seem even better when we add the fact that most misclassifications were done if it comes to digit 2 (10 from 25), every other number is dependent for an average number of 6,7% of the total misclassifications compared to the 40% of misclassifications in case of digit 2. So we can be more sure of classifications of these other digits.

The worst two models from this approach we can pick are the linear SVM and basic neural network with 2 hidden layers, the test set accuracies for them were respectively 95.63% and 94.93%. Number of misclassifications of digit 2 for linear SVM is consistent with the amount of misclassifications of that number for non linear SVM as in that case it is 51 from 131 so 38.93%.

The model that can be better than previously mentioned two is the neural network with dropout layers. In the end it managed to achieve an accuracy of 96.33%.
Final model to mention is the neural network with dropout layers, normalization of inputs and batch normalization of outputs. That model was achieving the best accuracies from all neural networks tested in this approach and managed to get the accuracy of 97.97% although it is still worse than the results of non linear SVM with normalization.

So finally if it comes to using pixels as individual features for now we believe that the non linear SVM with normalized inputs is the best option. It not only gives the best results but is consistent in these results compared to the neural network approaches. We believe that further experimenting with the neural networks, maybe changing number of neurons or layers, could bring us even closer to the results of the non linear SVM but misclassifying only 25 from 3000 test examples is satisfactory for us to stay where we stopped.

2. **Convolutional approach:**

After experimenting with various CNN architectures, the model with batch normalization achieved the highest test accuracy of 99.43%. This model consistently outperformed the simpler, deeper, and wider models, which achieved test accuracies of 98.86%, 99.23%, and 98.86% respectively. The batch normalization technique improved the stability and speed of the learning process, resulting in better performance and generalization. Early stopping was also crucial in preventing overfitting, allowing the model to achieve high accuracy without excessive training. Therefore, the model with batch normalization is the best among the tested architectures for handwritten digit classification.