Udacity Machine Learning Engineer Nanodegree

Capstone Project

Handwritten Digits Recognition Web App

Abdelrahman Fekry Ali

31 August 2021

Table Of Contents

Definition

Project Overview

in nowadays the need for Document digitization is in demand. due to many benefits such as the ease of accessing the data in the digital form from anywhere and cost reduction and data security through defining accessibility privileges and data storage and recovery.
in computer vision field the process of converting these documents into digital form is called Optical Character Recognition (OCR).
the process of Optical Character Recognition (OCR) includes many steps, in this project we are going to focus on the base step which is Handwritten digit recognition (HDR).

Handwritten digit recognition (HDR) is considered a base step for many applications such as

- Signature Verification,
- Postal-Address Interpretation,
- Bank-Check Processing,
- Writer Recognition.

for this project we used the MNIST (Modified National Institute of Standards and Technology) dataset.
based on the paper[1] The set of images in the MNIST database was created in 1998 as a combination of two of NIST's databases Special Database 1 (digits written by high school students) and Special Database 3 (digits written by employees) of the United States Census Bureau.
There are 70,000 images and each image has 784 features. This is because each image is 28 x 28 pixels, and each feature represents a pixel’s intensity, from 0 to 255.

Sample images from MNIST test dataset

Problem Statement

the main goal of this project is to implement web app that as asks the user to sketch a digit and then send the pixels information to backend where the data is processed then return the predicted value to the frontend.
to solve this problem we used simple HTML and CSS style sheet for the frontend, and we used Flask for the backend where the pre-trained model is loaded.
The machine learning algorithm that used to train the model is CNN as it proven to be the most effective based on paper [2].

Method	Recognition accuracy (%)
CNN	99.18
SVM	93.78
HOG–SVM	97.82
kNN	97.31
Random forest	94.82
RNN	96.95

CNN architectures consists of three types of layers, namely convolutional, pooling, and fully-connected layers.

- **The convolutional layer**
  It's main purpose is to extract features from inputs, it's composed of several learned kernels which are used to compute different feature maps by applying an element-wise nonlinear activation function on the convolved results, each neuron of a feature map is connected to "neuron's receptive field" in the previous layer.
  The kernels in the 1st convolutional layer are designed to detect low-level features such as edges and curves while the kernels in higher layers are learned to encode more abstract features
  The feedforward is calculated by:

$$Z_{i,j,k}^{l} = {w_k^l}^T X_{i,j}^{l} + b_k^l$$

  - $Z_{i,j,k}^{l}$ the feature value at location (i, j) in the k-th feature map of l-th layer.

- **The activation function**
  It's main purpose is to introduces nonlinearities to CNN, which are desirable for multi-layer networks to detect nonlinear features. Typical activation functions are sigmoid, tanh and ReLU.

$$a_{i,j,k}^{l} = a\left(Z_{i,j,k}^{l}\right)$$

  - $a_{i,j,k}^{l}$ The activation value of convolutional feature $Z_{i,j,k}^{l}$
  - $a(.)$ The nonlinear activation function

- **The pooling layer**
  aims to achieve shift-invariance by reducing the resolution of the feature maps. It is usually placed between two convolutional layers. Each feature map of a pooling layer is connected to its corresponding feature map of the preceding convolutional layer. The typical pooling operations are average pooling and max pooling.

$$y_{i,j,k}^{l} = pool\left(a_{m,n,k}^{l}\right), \forall (m.n) \in R_{i,j}$$

  - $pool(.)$ The pooling function
  - $a_{:,:,k}^{l}$ feature map
  - $\forall (m.n) \in R_{i,j}$: for all (m,n) in local neighbourhood around location (i, j)

- **The fully-connected layer**
  which aim to perform high-level reasoning. They take all neurons in the previous layer and connect them to every single neuron of current layer to generate global semantic information.
  fully-connected layer not always necessary as it can be replaced by a 1 × 1 convolution layer.

- **The output layer.**
  It uses softmax activation function for classification tasks, The optimum parameters for a specific task can be obtained by minimizing an appropriate loss function defined on that task, By minimizing the loss function, we can find the best fitting set of parameters. Stochastic gradient descent is a common solution for optimizing CNN network, The loss of CNN can be calculated as follows,

$$loss = \frac{1}{N}\sum_{n=1}^{N} l(\theta, y^n, o^n)$$

  - θ all the parameters of a CNN (e.g., the weight vectors and bias terms)
  - N desired input-output relations
  - x^n is the n-th input data,
  - y^n is the n-th target label
  - o^n is the n-th output of CNN

Metrics

as dealing with a multiclass classification problem that equally cares about Type I Error and Type II Error we are going to select accuracy as the evaluation metric. the accuracy

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

The formula of the Accuracy considers the sum of True Positive and True Negative elements at the numerator and the sum of all the entries of the confusion matrix at the denominator. True Positives and True Negatives are the elements correctly classified by the model and they are on the main diagonal of the confusion matrix, while the denominator also considers all the elements out of the main diagonal that have been incorrectly classified by the model.

Accuracy returns an overall measure of how much the model is correctly predicting on the entire set of data.

In case of imbalanced datasets (when most units are assigned to a single class)
By using Accuracy it is not possible to identify the classes where the algorithm is working worse.
The Accuracy tends to hide strong classification errors for classes with few units, since those classes are less relevant compared to the biggest ones.

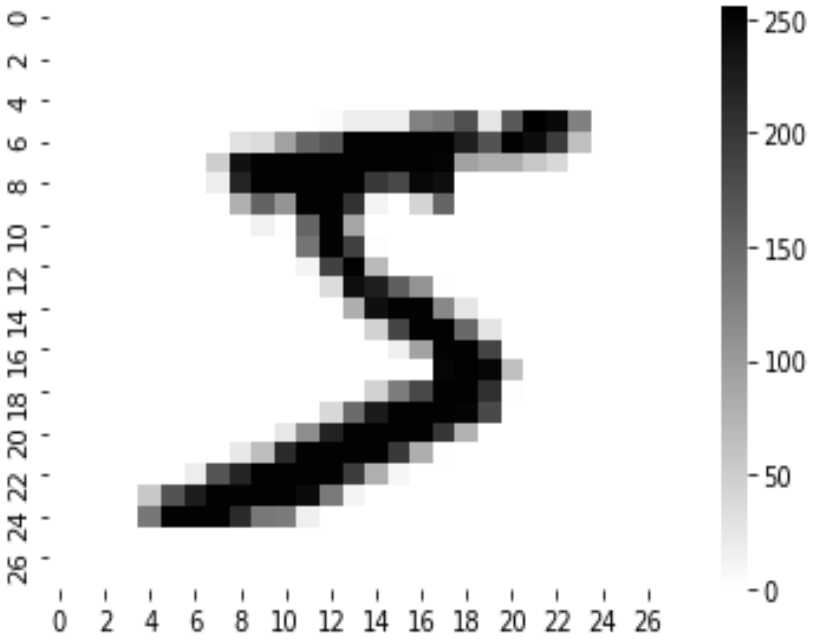
In this case we should calculate Balanced Accuracy for N number of classes

$$\text{Balanced Accuracy } = \frac{\sum_{n=1}^{N}\left(\frac{\text{correctly predicted class}_n\text{total}}{\text{actual class}_n\text{total}}\right)}{N}$$

Analysis

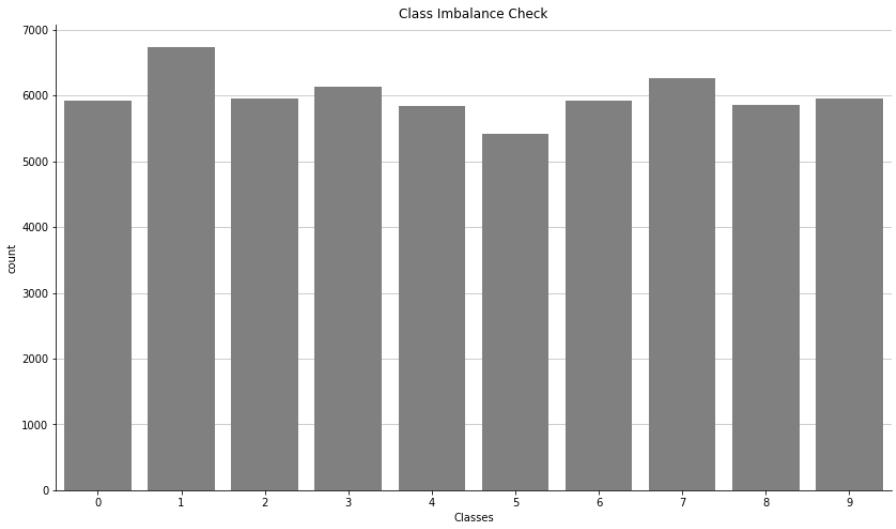
Data Exploration

There are 70,000 images and each image has 784 features. This is because each image is 28 x 28 pixels, and each feature represents a pixel's intensity, from 0 to 255.

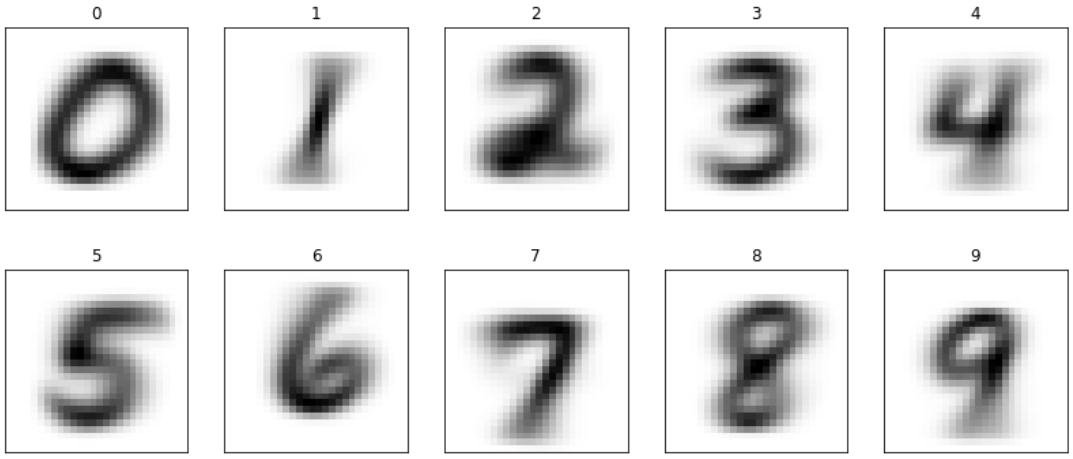


Sample image

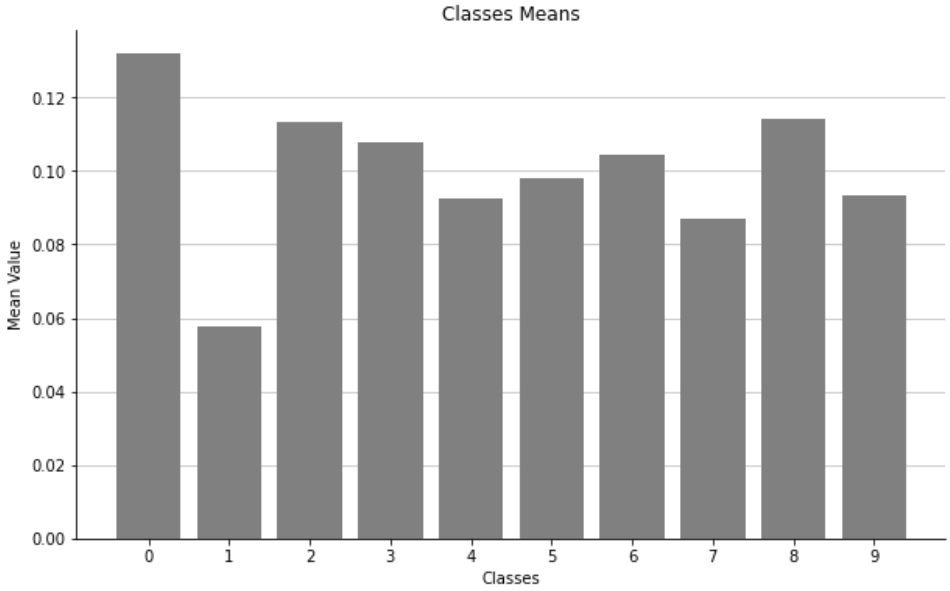
the MNIST dataset contains 10 classes represent images labels from 0 to 9, the classes are represented equally in the dataset in other words it's class balanced



When plotting the means of classes (class wise) it's easy to recognize the digits, that indicates that the data is not noisy and suitable for the CNN model



When plotting the means of classes we can notice that class of digit 1 has the lowest mean due to leak of higher intensity pixels. The means difference can be consider as feature to determine classes or we can standard scale each class at center (0 mean) by subtracting its mean.



Algorithms and Techniques

- **$L_2 - norm$ Regularization**
  Regularization modifies the objective function by adding additional terms that penalize the model complexity it's effective in reducing overfitting.

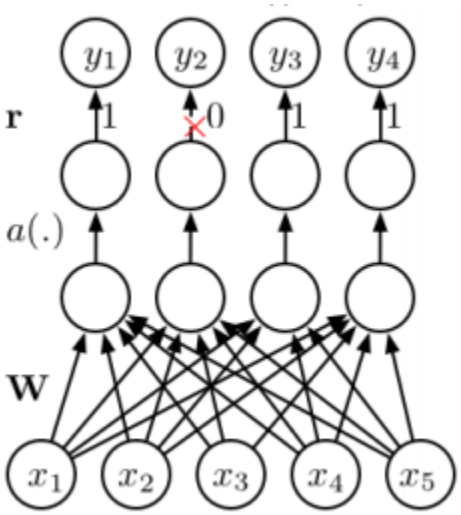
$$loss + \lambda \sum_{i=0}^{n} W_i^2$$

  - W_i^2 is the regularization term
  - λ is the regularization strength

- **Dropout Regularization**
  By Randomly setting a P% outputs of neurons to zero**,** Dropout can prevent the network from becoming too dependent on any one or any small combination of neurons

$$\mathrm{y} = \mathrm{r} * \mathrm{a}(\mathrm{W}^{\mathrm{T}}\mathrm{x})$$

  - x is the input to fully-connected layer
  - W is a weight matrix
  - r is a binary vector whose elements are independently drawn from a Bernoulli distribution with parameter P



- **Data Augmentation**
  Data augmentation consists in transforming the available data into new data without altering their natures**,** popular augmentation methods include simple geometric transformations such as sampling , mirroring , rotating , shifting.

- **Batch Normalization**
  transforms all layer inputs to have zero-mean and unit variance where the estimations of mean and variance are computed after each mini-batch rather than the entire training set

$$z_{norm} = \frac{z - \mu_z}{\sqrt{\sigma_z{}^2 + \epsilon}}$$
$$z = \gamma z_{norm} + \beta$$

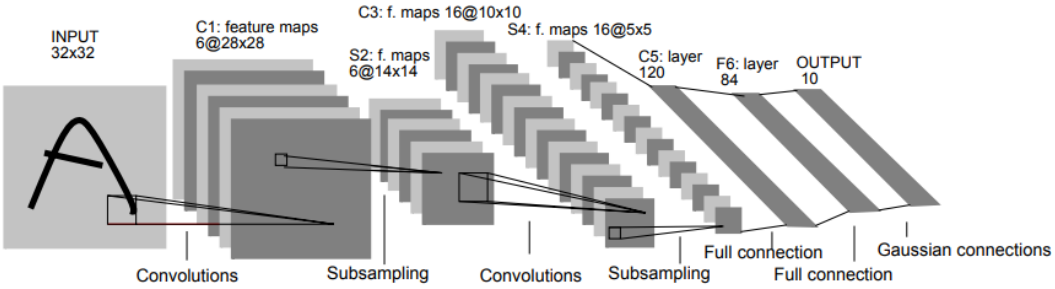
  - μ_z mean of mini-batch
  - σ_z variance of mini-batch
  - γ and β are learned parameters.

  Advantages of Batch Normalization:
  - reduces internal covariant shift.
  - reduces the dependence of gradients on the scale of the parameters or of their initial values, which gives a beneficial effect on the gradient flow through the network. This enables the use of higher learning rate without the risk of divergence.
  - regularizes the model, and thus reduces the need for Dropout
  - makes it possible to use saturating nonlinear activation functions without getting stuck in the saturated model.

Benchmark

Lenet-5 is one of the earliest pre-trained models proposed by Yann LeCun and others in the year 1998, in the research paper [3] They used this architecture for recognizing the handwritten and machine-printed characters. The main reason behind the popularity of this model was its simple and straightforward architecture. It is a multi-layer convolution neural network for image classification.

Architecture of LeNet-5



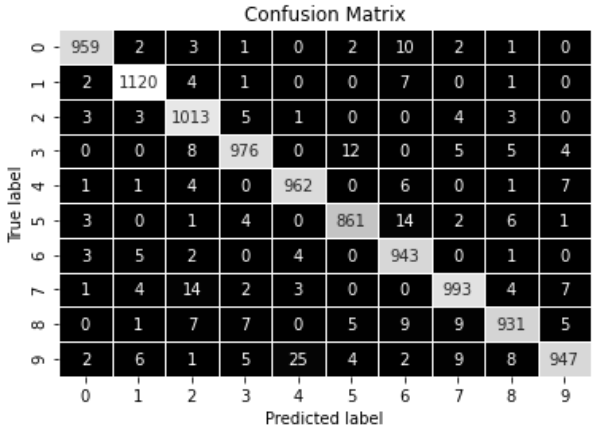
Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

- The first layer is the input layer with feature map size 32X32X1.
- Then we have the first convolution layer with 6 filters of size 5X5 and stride is 1. The activation function used at his layer is tanh. The output feature map is 28X28X6.
- Next, we have an average pooling layer with filter size 2X2 and stride 1. The resulting feature map is 14X14X6. Since the pooling layer doesn't affect the number of channels.
- After this comes the second convolution layer with 16 filters of 5X5 and stride 1. Also, the activation function is tanh. Now the output size is 10X10X16.
- Again comes the other average pooling layer of 2X2 with stride 2. As a result, the size of the feature map reduced to 5X5X16.
- The final pooling layer has 120 filters of 5X5 with stride 1 and activation function tanh. Now the output size is 120.
- The next is a fully connected layer with 84 neurons that result in the output to 84 values and the activation function used here is again tanh.
- The last layer is the output layer with 10 neurons and Softmax function. The Softmax gives the probability that a data point belongs to a particular class. The highest value is then predicted.

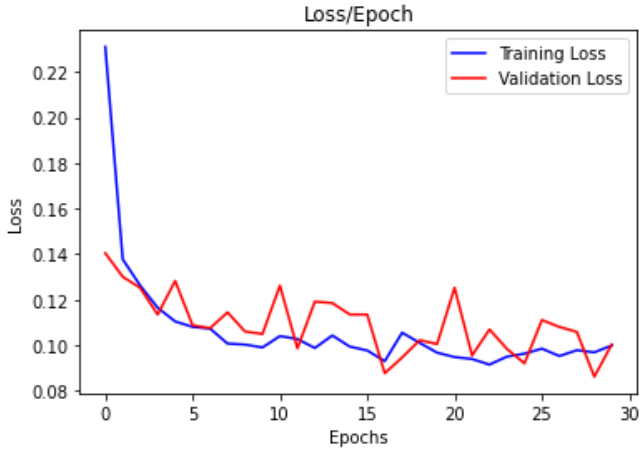
Evaluation

- accuracy 0.9705
- macro precision 0.9705
- macro recall 0.9703

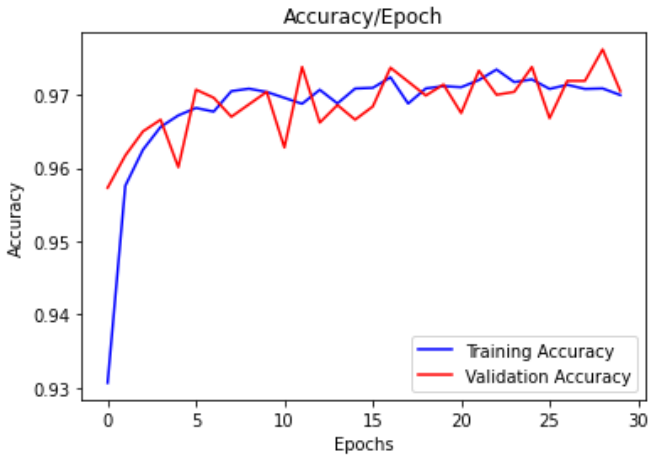
the confusion matrix shows the most miss predicted labels between the digits that have similar features such as (9,4), (7,2),(5,6),(3,5) and (0,6)



We can notice from the graph that validation loss is unstable, it's have a non-converged values through epochs. The unstable performance of the model is not a good practice for our web application



We can notice from the graph that validation accuracy is unstable, it's have a non-converged values through epochs, The unstable performance of the model is not a good practice for our web application



Methodology

Data Preprocessing

1. **Missing Values Check**
   First we checked for missing values

```
np.isnan(x_train).any()
np.isnan(y_train).any()
np.isnan(x_test).any()
np.isnan(y_test).any()
```

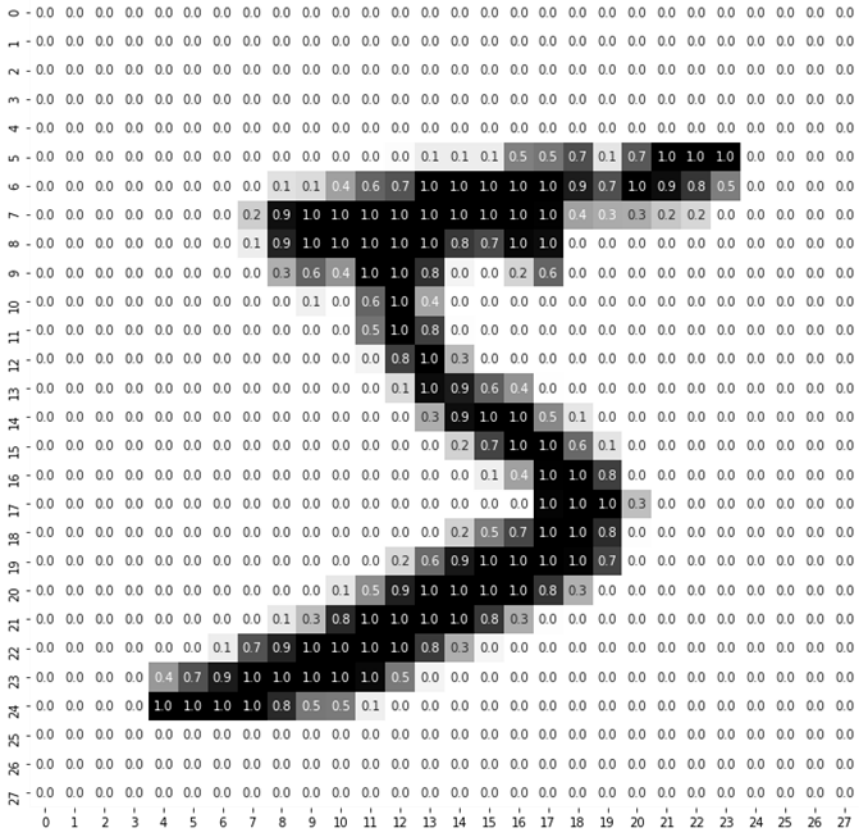
   as expected the returned value was False.

2. **MaxAbs Scaling**
   The maximum pixels intensity is 255 as we checked

```
x_train[0].max()
```

   using scikit `preprocessing.maxabs_scale()` function to rescale pixels intensity range from 0 to 1

```
x_train_scaled = np.array([maxabs_scale(x) for x in x_train])
x_test_scaled = np.array([maxabs_scale(x) for x in x_test])
```



MaxAbs Scaled Sample

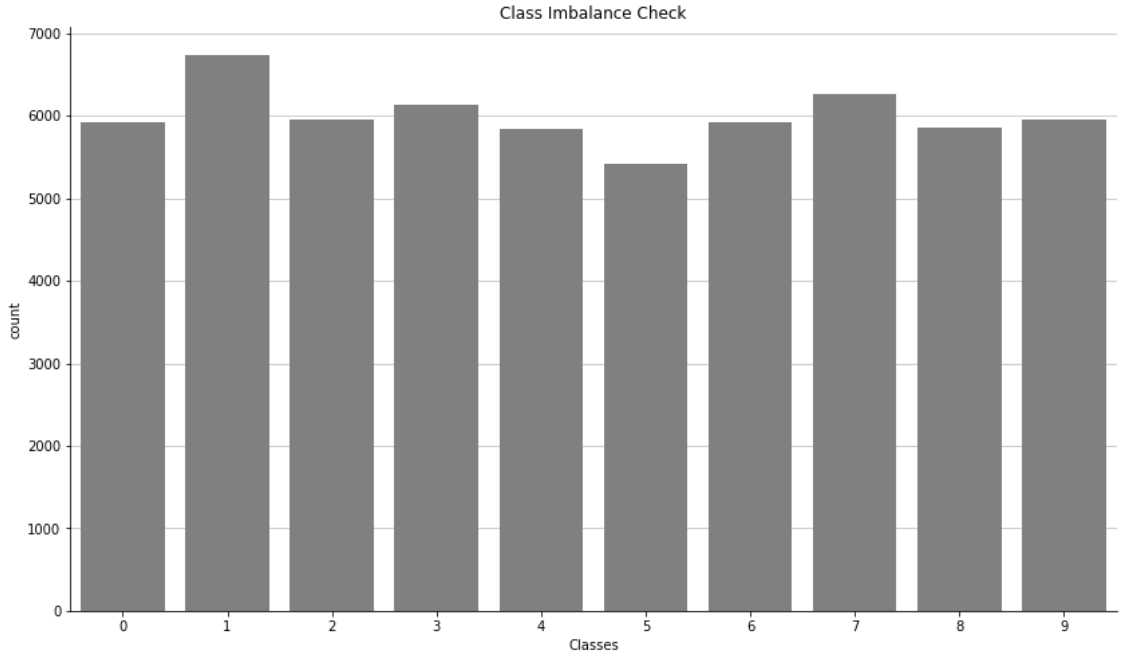
3. **Adjusting Shape And Dimensions**
The input data of the model have the shape (32,32,1) so implemented two steps to approach the targeted shape.
First we expanded the most inner dimension

```
x_train_scaled = np.expand_dims(x_train_scaled,axis=-1)
x_test_scaled = np.expand_dims(x_test_scaled,axis=-1)
```

Second we padded the input data by 2 constant values “0” each side on the axis 1 and 2.

```
x_train_scaled = np.pad(x_train_scaled, ((0,0),(2,2),(2,2),(0,0)), 'constant')
x_test_scaled = np.pad(x_test_scaled, ((0,0),(2,2),(2,2),(0,0)), 'constant')
```

4. **Class Imbalance Check**
the classes are represented equally in the MNIST dataset



5. **OneHot Encoding**
The output data of the model have the shape (10,) which is a softmax value/probability of each digit from 0-9 so we OneHot encoded the data labels to approach the targeted shape.

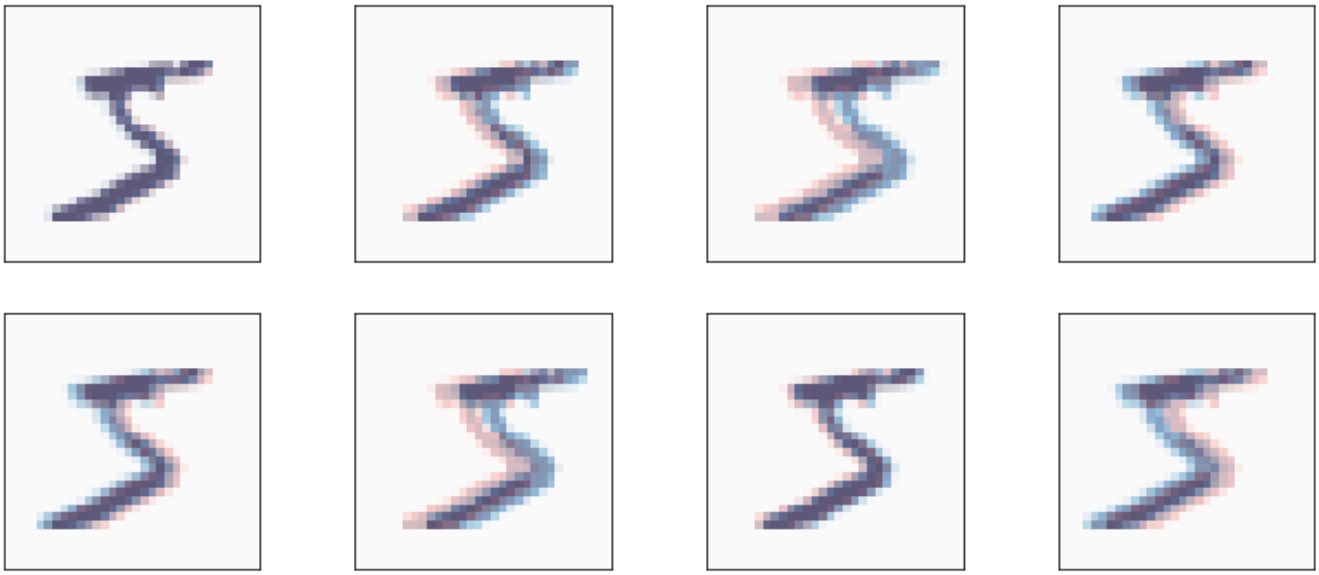
```
y_train_OneHot = to_categorical(y_train, num_classes = 10)
y_test_OneHot = to_categorical(y_test, num_classes = 10)
```

6. **Data augmentation**

Applying different transformations to input data by Shifting and scaling the input data by 10%, and rotation in range -10 to 10 degrees. this reduces the variance and allow model to generalize better to unseen data.

```
datagen = ImageDataGenerator(rotation_range=10,zoom_range = 0.1,width_shift_range=0.1,
height_shift_range=0.1)
datagen.fit(x_train_scaled)
```

we can see in the following figure how the generated output (Blue) is transformed from the original (Red)



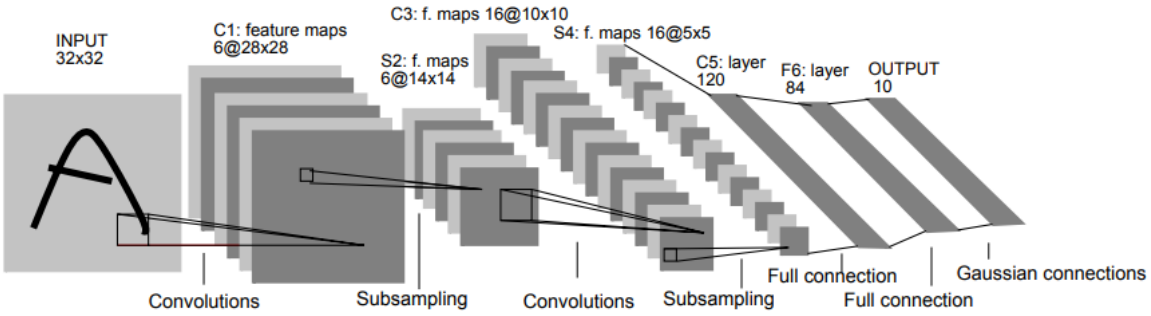
Transformed Training Samples

Implementation

LeNet-5 CNN

based on the paper[3] LeNet-5 CNN is considered as ideal for digit/character recognition so we are going to design our model base on it, with slight modifications.

1. **Structuring**



Architecture of LeNet-5

- **Input Layer**

consists of 32x32 pixel image the reason for padding that the digit can be more centered to be detected by the receptive field. the input pixels are grayscale with a value 0 for a white pixel and 1 for a black pixel

- **Convolution Layer**

It consists of 6 feature maps with Size 28x28 .the learnable kernels size is 5x5. the convolution layer is responsible for feature extraction from the input data.

- **ReLU activation function**

is used at the end of each convolution layer as well as a fully connected layer to enhance the performance of the model.

- **pooling layer**

It reduces the output information from the convolution layer and reduces the number of parameters and computational complexity of the model

- **Flatten layer**

is used after the pooling layer which converts the 2D featured map matrix to a 1D feature vector and allows the output to get handled by the fully connected
layers

- **fully connected layer**

Is another hidden layer also known as the dense layer. It is similar to the hidden layer of Artificial Neural Networks (ANNs) but here it is fully connected and connects every neuron from the previous layer to the next layer

- **output layer**

Consists of ten neurons and determines the digits numbered from 0 to 9. Since the output layer uses an activation function such as softmax, which is used to enhance the performance of the model

We made a slight modifications to the LeNet-5 model summarized on the following

- **Adding dropout regularization layers:**
  by randomly switching off 25% of neurons during training we make more robust CNN. this reduces overfitting and increase the generalization due to not relying too much on any particular neurons to produce an output.

- **Adding Batch Normalization layers:**
  Standard scaling the layers output makes the learning algorithm converge faster.

$$z_{norm} = \frac{z - \bar{z}}{\sqrt{{s_z}^2 + \epsilon}}$$
$$z = \gamma z_{norm} + \beta$$

- **Adding Kernel l2 regularization:**
  Increasing the loss causes the weights to be relatively smaller, which reduce overfitting we can approach that by penalizing the sum of the square of the weights (weight²) to the loss.

$$loss + \lambda \sum_{i=0}^{n} W_i^2$$

- **Adding more Dense layer:**
  Reduces the bias by allowing the model to learn more complex features of the input data.

Final Model Structure

```
model = Sequential()

model.add(layers.Conv2D(filters = 6, kernel_size = (5,5),padding='valid',
input_shape = (32,32,1),kernel_regularizer=l2(0.0005)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size = (2,2), strides = (2,2)))

model.add(layers.Conv2D(filters = 16, kernel_size = (5,5),padding='valid',
kernel_regularizer=l2(0.0005)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size = (2,2), strides = (2,2)))
model.add(layers.Flatten())

model.add(layers.Dense(units = 250))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dropout(0.25))

model.add(layers.Dense(units = 120))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dropout(0.25))

model.add(layers.Dense(units = 84))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))


model.add(layers.Dense(units = 10, activation = 'softmax'))
```

If we didn't pad the data manually earlier we could let the model pad the input with the shape (28,28,1) giving the same result using `padding='Same'` in the first layer

```
Model.add(layers.Conv2D(filters = 6, kernel_size = (5,5),padding = 'Same',
activation ='relu',input_shape = (28,28,1)))
```

2. Training

We used `ReduceLROnPlateau` callback function to reduce learning rate by `factor` when the validation loss metric has stopped improving for a `patience` number of epochs

$$\text{new\_lr} = \text{lr} * \text{factor}$$

dynamically reducing the learning rate helps the algorithm to converge faster and reach closer to the global minima.

```
learning_rate_reduction = callbacks.ReduceLROnPlateau(monitor='val_loss',patience=2,
factor=0.1,min_lr=0.00001)
```

we used adam optimizer as it combine the benefits of RMSProp and AdaGrad optimizers

- RMSProp - adapting learning rates based on the average of the first moment
- AdaGrad - adapting learning rates based on the average of the second moments

beta1 and beta2 control the decay rates of these moving averages.

beta1: The exponential decay rate for the first moment estimates (the mean) as in RMSProp
beta2:The exponential decay rate for the second-moment estimates (the non-centered variance) as in AdaGrad
Epsilon: Is a very small number to prevent any division by zero in the implementation

```
optimizer = optimizers.Adam(learning_rate=0.001,beta_1=0.9,beta_2=0.999,epsilon=1e-7)
```

as we dealing with multi-class classification problem with Softmax activation function output we used Categorical CrossEntropy

$$loss = -\sum_{i=1}^{output\ size} y_i . \log \hat{y}_i$$

- y_i OneHot encoded true labels
- $\hat{y}_i$ Softmax activation function output

```
loss = losses.CategoricalCrossentropy()
```

the targeted metric is accuracy as mentioned earlier

```
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
```

we trained the model for 30 epochs with batch size of 60,000/100 = 600 samples per epoch.

```
history = model.fit(datagen.flow(x_train_scaled,y_train_OneHot,batch_size=100),
                    epochs = 30,
                    validation_data = (x_test_scaled,y_test_OneHot),
                    callbacks=[learning_rate_reduction],
                    verbose = 1)
```

Results

Model Evaluation and Validation

We calculated the predicted labels by taking the argmax of Softmax function output.

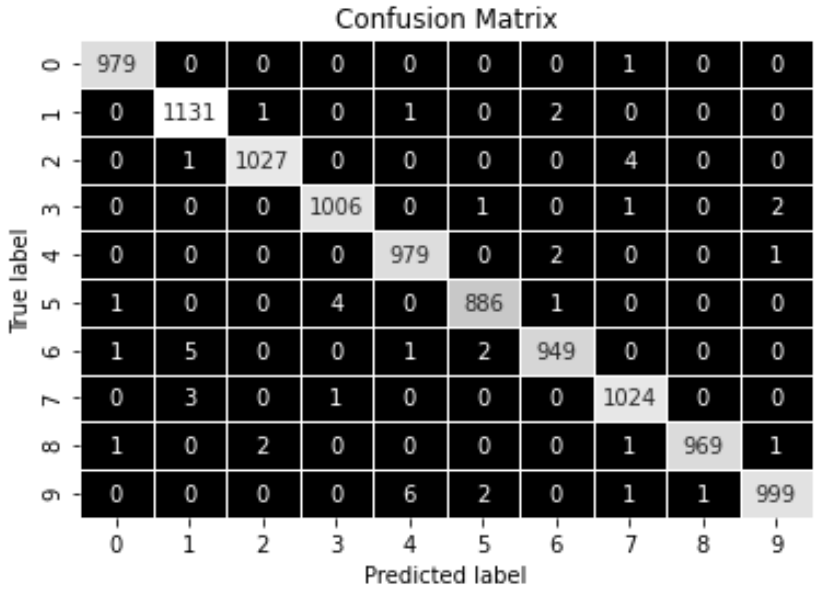
```
y_pred = np.argmax(model.predict(x_test_scaled), axis=-1)
```

the accuracy of the model is excellent as well as macro precision and macro recall

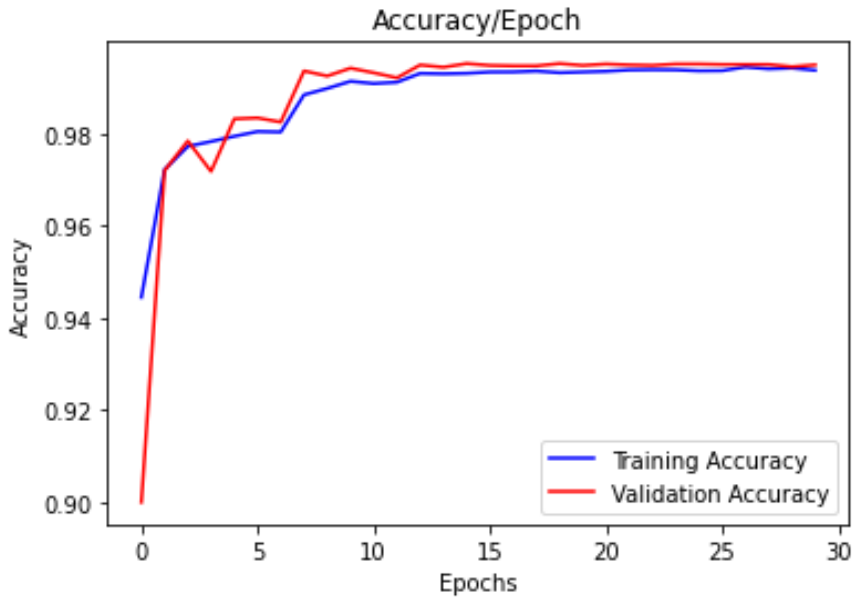
```
print(f'{"accuracy":10} {(y_pred == y_test).mean():.4f}')
print(f'{"precision":10} {precision_score(y_test,y_pred,average="macro"):.4f}')
print(f'{"recall":10} {recall_score(y_test,y_pred,average="macro"):.4f}')
```

- accuracy 0.9949
- macro precision 0.9949
- macro recall 0.9949

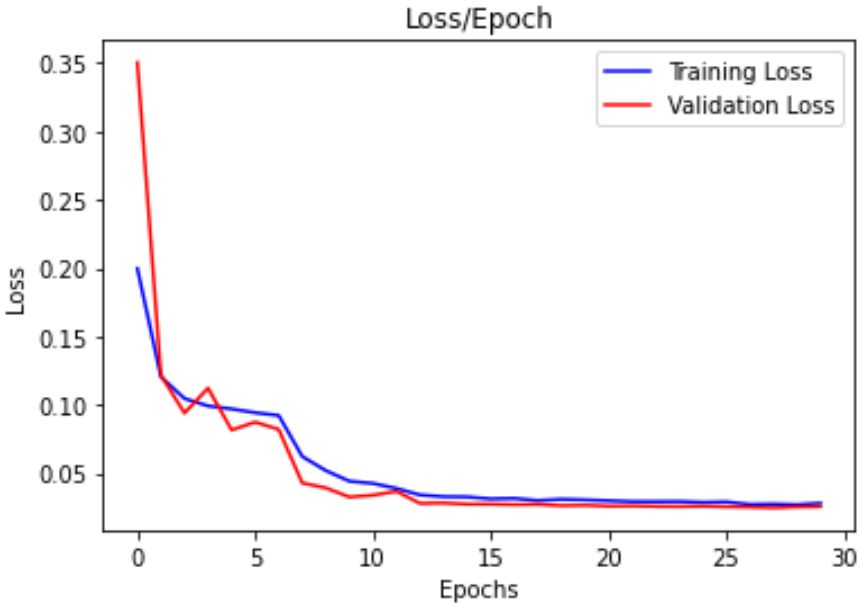
from the following confusion matrix the most miss predicted digit is 9 labeled as 4 due to the similarity between the two digits



The accuracy of the validation set was unstable in the first epochs the is recovered in the following epochs giving a very stable performance.

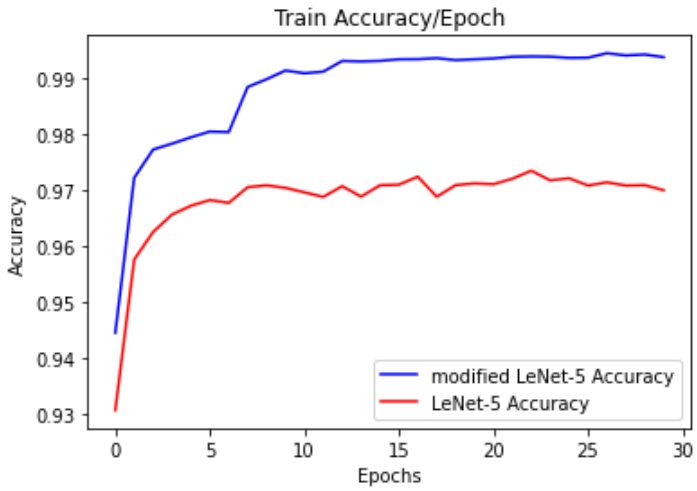


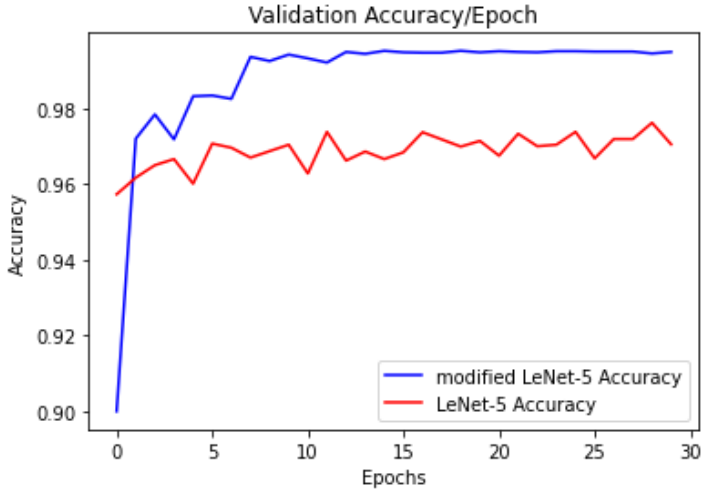
The loss of the validation set was unstable in the early epochs then it recovered in the following epochs giving very stable performance.

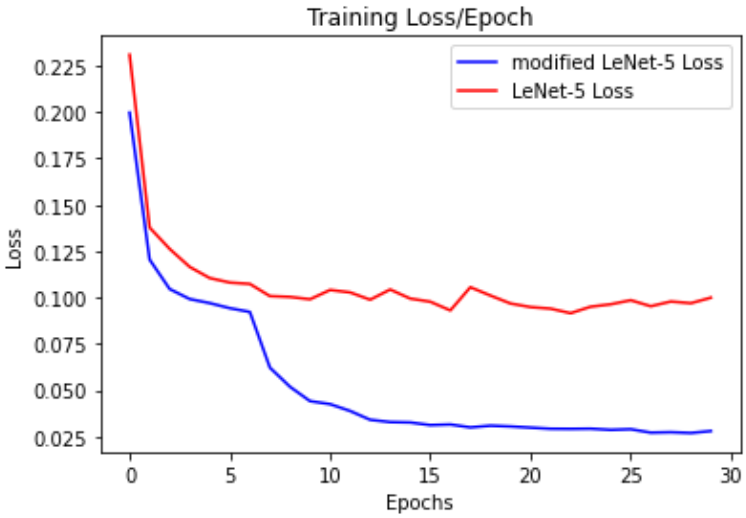


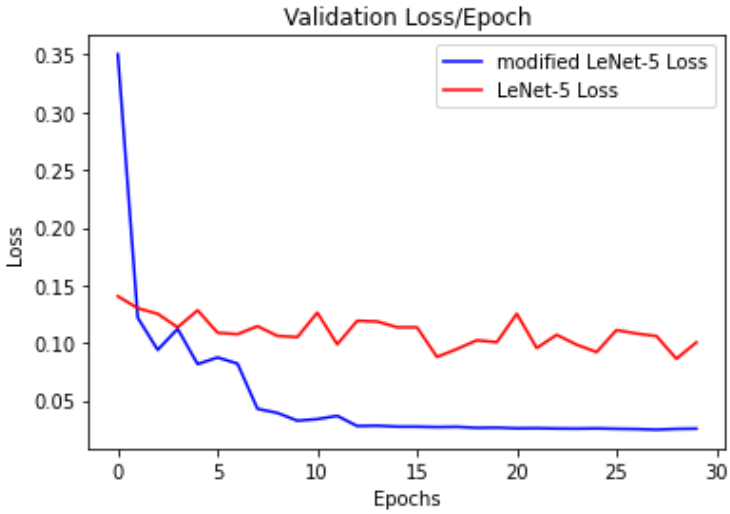
Justification

When comparing the performance of the Classic LeNet-5 by our modified model we can notice that our modified model gives higher accuracy 99.49% which is the desired metric and it's more stable in the later epochs we can predict that the stable performance will continue as tested on new unseen data. We should diffidently use our modified model for the web application.









Pre-trained Model Saving

```
data_dir = 'trained_models'
if not os.path.exists(data_dir):
    os.makedirs(data_dir)
model_1.save(f'{data_dir}/LeNet_5.h5')
```

References

- [1] "THE MNIST DATABASE of handwritten digits" Yann LeCun, Corinna Cortes, Christopher J.C. Burges http://yann.lecun.com/exdb/mnist/
- [2] "ARDIS: a Swedish historical handwritten digit dataset" Huseyin Kusetogullari, Amir Yavariabdi, Abbas Cheddad March 2019 https://link.springer.com/article/10.1007/s00521-019-04163-3
- [3] "GradientBased Learning Applied to Document Recognition" Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner ,November 1998 http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- [4] "Handwritten digit recognition with a back-propagation network" LeCun, Y.,Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; Jacker, L. D. (June 1990). http://yann.lecun.com/exdb/publis/pdf/lecun-90c.pdf
- [5] "Generalization and network design strategies" Lecun, Yann (June 1989). http://yann.lecun.com/exdb/publis/pdf/lecun-89.pdf
- [6] "Recent Advances in Convolutional Neural Networks" Jiuxiang Gua, Zhenhua Wangb , Jason Kuenb , Lianyang Mab , Amir Shahroudyb , Bing Shuaib , Ting Liub , Xingxing Wangb , Li Wangb , Gang Wangb , Jianfei Caic , Tsuhan Chenc https://arxiv.org/pdf/1512.07108.pdf
- [7] "Metrics for Multi-Class Classification: an Overview" Margherita Grandini, Enrico Bagli, Giorgio Visani https://arxiv.org/pdf/2008.05756.pdf