HappyMonk Technology

About:

Happy Monk Technology is an Artificial Intelligence & Emerging Technology Firm delivering AI-powered software and technical solutions to companies who want to leverage data and machine learning algorithms through Ethical , Federated AI and decentralised systems.

Reasearch & development:

There mainly focused on the reasearch and development of the Artifical inteliigence in to the some extent. By providing countinueos efforts to develop applications.

Tasks Given By the HappyMonk Technology:

For internship:

1. Bank Note Authentication. (Binary Classification Problem)

For Junior Data scientist Position:

1. Bank Note Authentication.
2. MNIST (Digit Recognizer)
3. Iris Dataset.

Contents in Documentation:

1. Bank Note.
2. MNIST
3. Iris dataset

Bank Note

Github Link: https://github.com/praneeth300/HappyMonk-/blob/main/Bank_Note.ipynb

Dataset: https://www.kaggle.com/ritesaluja/bank-note-authentication-uci-data

Problem Statement:

Given a specific activation function

$$g(x) = k0 + k1x \quad (1)$$

and categorical cross-entropy loss, design a Neural Network on Banknote, MNIST or IRIS data where the activation function parameters k0, k1 are learned from the data you choose from one of the above-mentioned data sets. *Your solution must include the learnable parameter values i.e. final k0, k1 values at the end of training, a plot depicting changes in k0, k1 at each epoch, training vs test loss, train vs. test accuracy and a Loss function plot.*

Dataset:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

Shape of the dataset:

Number of columns: 5

Number of observations: 1372

Skewness:

As we can observe that some of the variables in our dataset have some skewness i.e.,curtosis and entrophy show right and left skewnees whereas in variance and skewness display's like it is a gaussina distribution. We need to handle it by using the some of the feature transformation techniques such as "Power transform","Box cox transformation" or "Log normal transformation". Genrally logarthemic transformation perform well on the skewness data, it converts the data in to gaussian distribution or normal distribuiton.

Selecting Parameters:

Given that the dataset is small, a small batch size is probably a good idea, e.g. 16 or 32 rows. Using the Adam version of stochastic gradient descent is a good idea when getting started as it will automatically adapt the learning rate and works well on most datasets.

Model architeture:

We will develop a Multilayer Perceptron (MLP) model for the dataset using TensorFlow.

We cannot know what model architecture of learning hyperparameters would be good or best for this dataset, so we must experiment and discover what works well**.**

After Experimented with most of the activation functions **LeakyRelu (alpha=0.01)** performs well on the training data and validation data , by considering the all parameters of the model.

N_features = 4

feat = x.shape[1]

Initialize the model

l_model = Sequential()

First Hidden layer with 10 Node's and Input layer with the shape of X

l_model.add(Dense(10,activation=LeakyReLU(alpha=0.01),input_shape=(feat,)))

Output Layer with 1 node,because it is an binary classification model

l_model.add(Dense(1,activation='sigmoid'))

Compile the Model with loss is 'Binary Croossentropy"

l_model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

Fit the Model to the training data.

hist_lrel = l_model.fit(x_train,y_train,validation_data=(x_val,y_val),epochs=50,batch_size=32,verbose=1)

Intial Epoch:

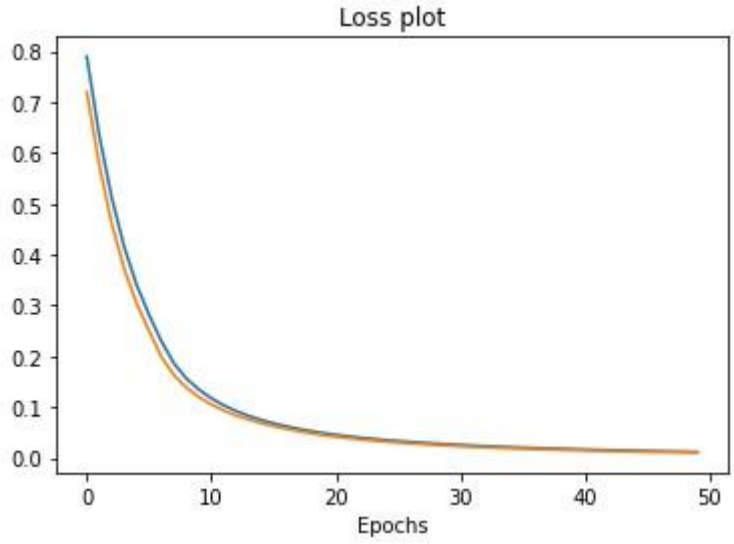
Epoch 1/50 35/35 [==============================] - 1s 6ms/step - loss: 0.7897 - accuracy: 0.6427 - val_loss: 0.7202 - val_accuracy: 0.6691

Final Epoch:

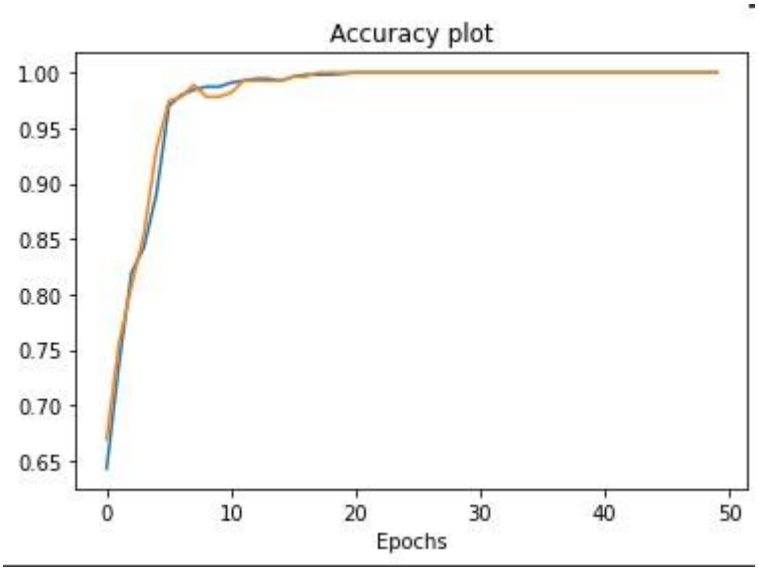
Epoch 50/50 35/35 [==============================] - 0s 3ms/step - loss: 0.0120 - accuracy: 1.0000 - val_loss: 0.0104 - val_accuracy: 1.0000

Plots:

1. **Loss plot:**



2. **Accuracy plot:**



Test data accuracy:

Accuracy: 1.00

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	157
1	1.00	1.00	1.00	118
accuracy			1.00	275
macro avg	1.00	1.00	1.00	275
weighted avg	1.00	1.00	1.00	275

Weights:

```
<tf.Variable 'dense_56/kernel:0' shape=(4, 10) dtype=float32, numpy=
array([[-1.0191457 ,  0.43691105, -0.24920474,  0.91669387,  1.1209365 ,
         0.17461905,  0.30388984,  0.38420722, -1.1818974 , -0.73866546],
       [ 0.12706555,  0.8034793 , -0.09776583,  0.34244478,  0.5177767 ,
        -0.1604144 ,  0.9389532 ,  0.05366395, -0.3934279 , -0.01984872],
       [ 0.64960986, -0.17276475, -0.56822956, -0.01981766,  0.8746438 ,
         0.28390008,  0.78429264, -0.73570484, -0.09717001, -0.9764796 ],
       [ 0.21970525,  0.14473073, -0.7532152 , -0.37142417, -0.3458716 ,
         0.26863584,  0.07149642,  0.34521458,  1.3007131 ,  0.13457708]],
      dtype=float32)>,
<tf.Variable 'dense_56/bias:0' shape=(10,) dtype=float32, numpy=
array([ 0.30863136, -0.35084096,  0.6415316 ,  0.05500666,  0.44968727,
       -0.01826408, -0.04380936,  0.17380986,  0.8931041 ,  0.7091936 ],
      dtype=float32)>,
<tf.Variable 'dense_57/kernel:0' shape=(10, 1) dtype=float32, numpy=
array([[ 0.2933338 ],
       [-0.37113386],
       [ 0.5880925 ],
       [-0.44298902],
       [-0.6951934 ],
       [-0.3590017 ],
       [-0.61465156],
       [ 0.542976  ],
       [ 1.0765252 ],
       [ 1.0486454 ]], dtype=float32)>,
<tf.Variable 'dense_57/bias:0' shape=(1,) dtype=float32, numpy=array([0.24318656], dtype=float32)>]
```

Selected Activation function:

LeakyRelu (alpha = 0.1)

Preseneted By:

P.Praneeth Kumar

Contact: +918309116050

Email: praneethsanthosh555@gmail.com

MNIST

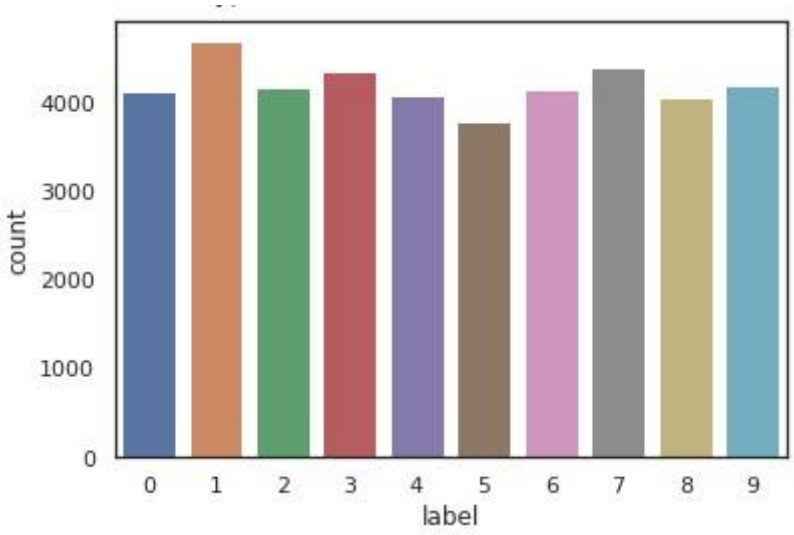
Github: https://github.com/praneeth300/HappyMonk-/blob/main/mnist.ipynb

Dataset: https://www.kaggle.com/c/digit-recognizer

The Document Follows three parts:

- The data preparation
- The CNN modeling and evaluation
- The results prediction and submission

Count of Digits in our dataset:



Normalization:

We perform a grayscale normalization to reduce the effect of illumination's differences.

Moreover the CNN converg faster on [0..1] data than on [0..255].

Reshape:

Np.values.reshape(-1,w,h,1)

Label Encoding:

Y_train = to_categorical(Y_train, num_classes = 10)

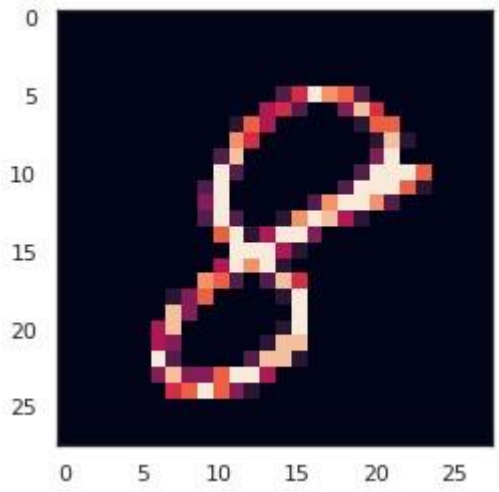
Splitting the data:

X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, random_state=random_seed)

I choosed to split the train set in two parts : a small fraction (10%) became the validation set which the model is evaluated and the rest (90%) is used to train the model.

Since we have 42 000 training images of balanced labels (see 2.1 Load data), a random split of the train set doesn't cause some labels to be over represented in the validation set. Be carefull with some unbalanced dataset a simple random split could cause inaccurate evaluation during the validation.

Sample image:



CNN :

I used the Keras Sequential API, where you have just to add one layer at a time, starting from the input.

The first is the convolutional (Conv2D) layer. It is like a set of learnable filters. I choosed to set 32 filters for the two firsts conv2D layers and 64 filters for the two last ones. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image.

The CNN can isolate features that are useful everywhere from these transformed images (feature maps).

The second important layer in CNN is the pooling (MaxPool2D) layer. This layer simply acts as a downsampling filter. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size (i.e the area size pooled each time) more the pooling dimension is high, more the downsampling is important.

Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

Dropout is a regularization method, where a proportion of nodes in the layer are randomly ignored (setting their wieghts to zero) for each training sample. This drops randomly a propotion of the network and forces the network to learn features in a distributed way. This technique also improves generalization and reduces the overfitting.

'relu' is the rectifier (activation function max(0,x). The rectifier activation function is used to add non linearity to the network.

The Flatten layer is use to convert the final feature maps into a one single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers.

In the end i used the features in two fully-connected (Dense) layers which is just artificial an neural networks (ANN) classifier. In the last layer(Dense(10,activation="softmax")) the net outputs distribution of probability of each class.

Model Architeture:

```
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
```

Data augmentation:

In order to avoid overfitting problem, we need to expand artificially our handwritten digit dataset. We can make your existing dataset even larger. The idea is to alter the training data with small transformations to reproduce the variations occuring when someone is writing a digit.

The improvement is important :

- Without data augmentation i obtained an accuracy of 98.114%
- With data augmentation i achieved 99.67% of accuracy

For the data augmentation, i choosed to :

- Randomly rotate some training images by 10 degrees
- Randomly Zoom by 10% some training images
- Randomly shift images horizontally by 10% of the width
- Randomly shift images vertically by 10% of the height

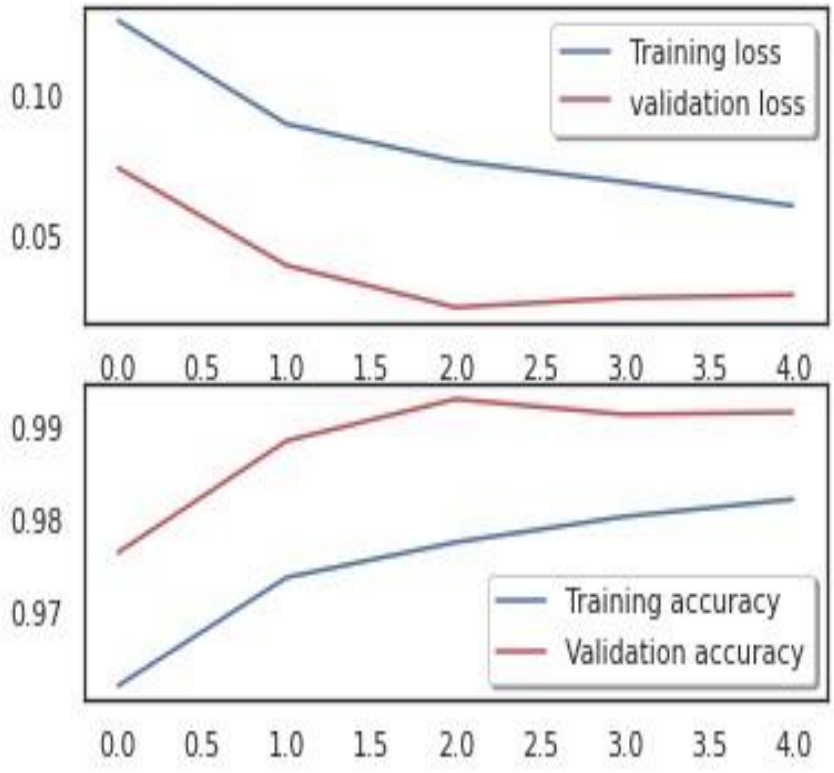
Once our model is ready, we fit the training dataset .

```
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                              epochs = 5, validation_data = (X_val,Y_val),
                              verbose = 2, steps_per_epoch=X_train.shape[0] // batch_size
                              , callbacks=[learning_rate_reduction])
```

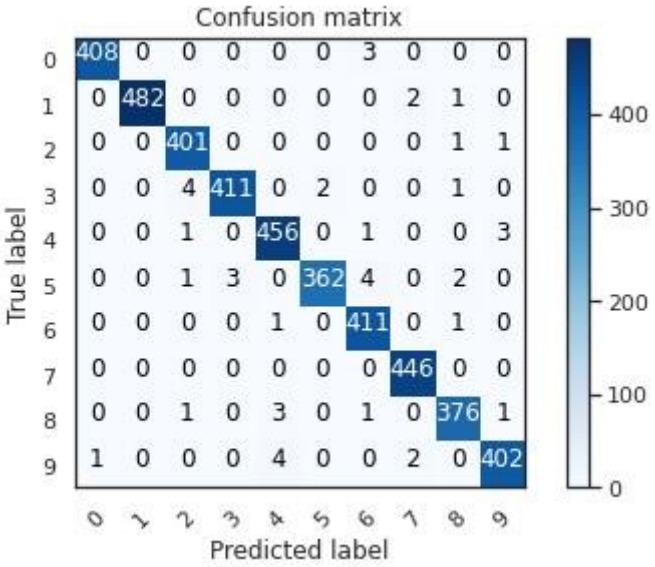
Epochs = 5

```
Epoch 1/5439/439 - 11s - loss: 0.1269 - accuracy: 0.9620 - val_loss: 0.0745 -
val_accuracy: 0.9764

Epoch 2/5439/439 - 11s - loss: 0.0900 - accuracy: 0.9737 - val_loss: 0.0397 -
val_accuracy: 0.9886

Epoch 3/5439/439 - 11s - loss: 0.0769 - accuracy: 0.9775 - val_loss: 0.0248 -
val_accuracy: 0.9931

Epoch 4/5439/439 - 10s - loss: 0.0695 - accuracy: 0.9803 - val_loss: 0.0281 -
val_accuracy: 0.9914

Epoch 5/5439/439 - 11s - loss: 0.0610 - accuracy: 0.9822 - val_loss: 0.0292 -
val_accuracy: 0.9917
```

Loss & accuracy Plots:



Confusion Matrix:



Selected Activation function based on different experiments:

Relu

IRIS Dataset

Dataset:

IRIS dataset is used in this kernel to demonstrate the multiclass classification using `TensorFlow 2.x`. This dataset has 5 features, out of which 4 features are numeric features and 1 is a categorical feature.

Dataset Link: https://www.kaggle.com/uciml/iris

Github: https://github.com/praneeth300/HappyMonk-/blob/main/iris-data-tensorflow-neural-network.ipynb

Sample data:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Analyze the each feature and plot the distributions of the numerical data and then compare it with feature in the dataset we have make hypothesis testing. Do some univarient and Bivarient analysis , make a decisions from the data, Compare it with the target variable of how variables are related to the target variable.

Prepare dependent and independent features:

```
X = iris_data.loc[:, iris_data.columns != 'species']
y = iris_data.loc[:, ['species']]
```

Convert the target feature in to categorical using label encoding:

```
y_enc = LabelEncoder().fit_transform(y)
y_label = tf.keras.utils.to_categorical(y_enc)
```

Split the data set in to training and validation set:

70% for training set and 30% validation set

```
X_train, X_test, y_train, y_test = train_test_split(X, y_label, test_size=0.3)
```

Model architeture:

After applying so many activation function's we can consider Relu perform better compared to the other activation functions.

A Artificial neural network with input shape of 4 Layers and 258 HL1,512HL2,100HL3 and 3 output nodes.

```
def get_model(activations):
        _model = Sequential([
        keras.layers.Input(shape=X_train.shape[1:]),
        keras.layers.Dense(258, activation=activations),
        keras.layers.Dense(512, activation=activations,),
        keras.layers.Dense(100, activation=activations),
        keras.layers.Dropout(0.2),
        keras.layers.Dense(3, activation='softmax')])
        return model


model = get_model(activations = 'relu')
model.compile(optimizer='adam',
         loss=keras.losses.CategoricalCrossentropy(),
        metrics=['accuracy'])
```

Epochs = 50

Batch size = 32

Fit the model:

history = model.fit(X_train, y_train, epochs=50,batch_size = 32, validation_data=(X_test, y_test), verbose=1)

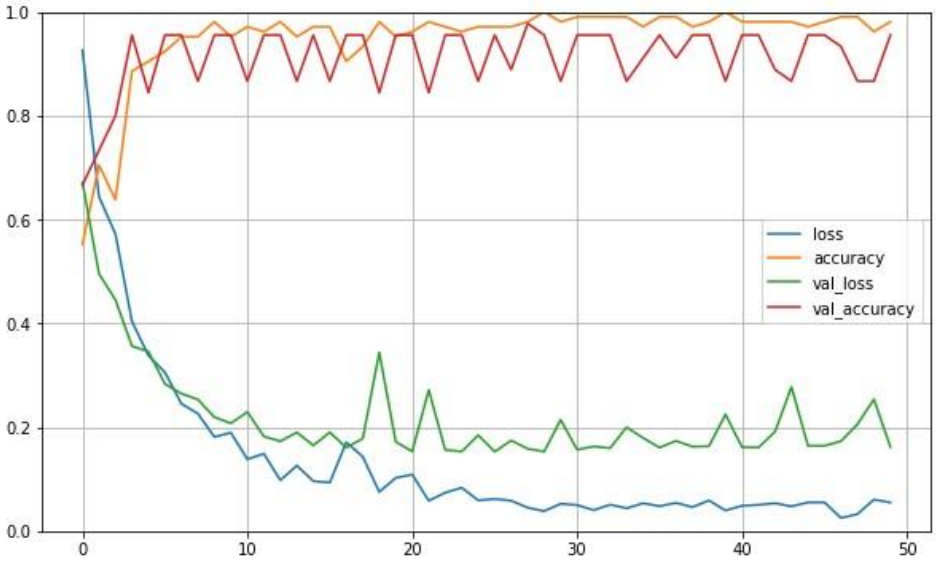
Intial Epoch:

Epoch 1/504/4 [==============================] - 0s 34ms/step - loss: 0.9872 - accuracy: 0.4762 - val_loss: 0.7859 - val_accuracy: 0.7333

Final Epoch:

Epoch 50/504/4 [==============================] - 0s 10ms/step - loss: 0.0576 - accuracy: 0.9714 - val_loss: 0.1975 - val_accuracy: 0.9556

Loss and Accuarcy plots:



Selected activation function:

Relu

Thank you...!