BASIC IMAGE CLASSIFICATION WITH TENSORFLOW

Recognise handwritten digit image.

Dataset → MNIST

TASK 1 - IMPORT LIBRARIES

TASK 2 - THE DATA SET.

We used Keras API with tensorflow as backend

```
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

x_test, y_test → data to validate the performance of trained neural network

All are numpy arrays and are multi dimensional.

```
x_train.shape
y_train.shape
```
→ to view shape of data

```
plt.imshow(x_train[0], cmap='binary')
plt.show()
```

plt → pyplot from matplotlib

show image

cmap → to give colour to image

x_train[0] → first one of training data

'binary' → to display black and white as our data is black & white

y_train[0] → Display first result i.e the value to our image displayed above.

__/__/____

print (set (Y_train)) → all unique values of Y_train set.

TASK-3 One HOT Encoding

Original label	one-hot encoded
5	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

Why One Hot encoding? → Instead of making model predict from 0-9 we now have 6 class to choose from.

Y_train_encoded = to_categorical (Y_train)
Y_test_encoded = to_categorical (Y_test)

↓

from tensorflow.keras.utils import to_categorical

Y_train_encoded.shape
Y_test_encoded.shape] → Shape

Y_train_encoded [0] → first label

__/__/____

TASK 4: NEURAL NETWORKS

Basic understanding. See video.

TASK-5 Preprocessing the Examples.

Reshaping x_train & x_test to 784 instead of 28×28.

Reshaped

```
x_train_reshaped = np.reshape(x_train, (60000, 784))
x_test_reshaped = np.reshape(x_test, (10000, 784))

x_train_reshaped.shape  # Checking the reshaped version

print(set(X_train_reshaped[0])). # all unique values because printing 784 values is too big.
```

DATA NORMALISATION

```
x_mean = np.mean(x_train_reshaped)
x_std = np.std(x_train_reshaped)
epsilon = 1e-10   -> very small value.
x_train_norm = (x_train_reshaped - x_mean)/(x_std + epsilon)
```

usually we divide only by x_std. But in some cases small value of standard deviation leads to unstability in computation and adding another small value usually solves

__/__/____

this.

x_test_norm = (x_test_reshaped - x_mean) / (x_std + epsilon)

x_mean and x_std are not recalculated on test set to keep the preprocessing same.

TASK-6 CREATING A MODEL

Creating a model

model = Sequential([

Input to first layer

Hidden Layer

Dense (128, activation = 'relu', input_shape = (784,)),

Dense (128, activation = 'relu'),

Output Layer

Dense (10, activation = 'softmax')

Softmax gives probability score of each class in the output layer.

The input_shape () automatically defines the input layer

Compiling a model

model.compile (

optimization algo called stochastic gradient descent.

optimizer = 'sgd'

loss = 'categorical_crossentropy'

It is kind of diff. between actual & pred Y. This we need to minimise

metrics = ['accuracy']

On to what feature we look at when model trains

__/__/____

model.summary() → architecture of model.

TASK -> TRAINING THE MODEL

model.fit(X_train_norm, Y_train_encoded, epoch=3)

X_train_norm: training data
Y_train_encoded: labels
epoch=3: iterations

We must make sure that the machine has not just memorised the result/examples but actually understood the underlined function.

To do that we evaluate model.

loss, accuracy = model.evaluate(X_test_norm, Y_test_encoded)

print(accuracy)

In evaluation there is no training step. All it does is it uses the model state as it is and it does forward pass to predict the testing data and then compares with the given actual test labels.

If accuracy is almost same as training accuracy it means the model understood our underline function.

If accuracy is very low, it means during

__/__/____

Leaving the model memorized the result/ examples and that's why its accuracy was high during training and it failed during testing/ evaluating the model.

TASK-8 Predictions.

Make predictions → predictions = model.predict(x_test_norm)

print(predictions.shape)
print(predictions[0])

We cannot go through all 10000 predictions but we will take a look at few.

plt.figure(figsize=(12,12))

start_index = 0

for i in range(25):
    plt.subplot(5,5,i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

Each other parameters

Highest prediction vali class bhata hai

Each prediction → pred = np.argmax(predictions[start_index+i])
Each actual value → actual = y_test[start_index+i]

esult/
uracy
t
model.

col = 'g' → green if O/P
if pred != actual:
col = 'r' → Red if prediction is wrong.

Printing the prediction: plt.xlabel('i={}, pred={}, actual={}'.format(start_index+i, pred, actual))

Image: plt.imshow(x_test[start_index+i], cmap='binary')

plt.show() (outside for loop)

One of the result is incorrect.

We will now check what were the other probabilities for it.

plt.plot(preds[8])

↑ Index of wrong one

plt.show

The probability score of prediction sum upto total of 1. The probability scores for the 10 classes will sum to 1 and class with the higher probability score is used as the final predicted class.