This code trains a Convolutional Neural Network (CNN) using the MNIST dataset of handwritten digits, evaluates its performance, and visualizes the predictions. Here's a detailed breakdown of the code:

**1. Import Libraries**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

* **tensorflow**: The framework used for building and training neural networks.
* **layers and models**: Submodules in Keras for defining the CNN architecture.
* **mnist**: A dataset provided by Keras, containing 70,000 images of handwritten digits (60,000 for training and 10,000 for testing).

**2. Load and Prepare Data**

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

* Loads the MNIST dataset into training and testing splits.
* **train\_images**: Images for training the model.
* **train\_labels**: Corresponding digit labels for training images.
* **test\_images** and **test\_labels**: Reserved for model evaluation.

**2.1 Normalize the Pixel Values**

train\_images = train\_images.astype('float32') / 255.0

test\_images = test\_images.astype('float32') / 255.0

* Scales pixel values from 0-255 to 0-1 to make training faster and improve model performance.

**2.2 Reshape Images**

train\_images = train\_images.reshape((60000, 28, 28, 1))

test\_images = test\_images.reshape((10000, 28, 28, 1))

* Reshapes the data to include a channel dimension (28x28x1).
* CNNs require input with explicit channel dimensions (e.g., grayscale images have 1 channel).

**2.3 One-Hot Encode Labels**

from tensorflow.keras.utils import to\_categorical

train\_labels = to\_categorical(train\_labels)

test\_labels = to\_categorical(test\_labels)

* Converts labels (0–9) into one-hot vectors. Example: 3 becomes [0, 0, 0, 1, 0, 0, 0, 0, 0, 0].
* This format is required for the categorical cross-entropy loss function.

**3. Define the CNN Model**

model = models.Sequential()

* **Sequential**: Used to define a linear stack of layers.

**Add Layers**

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

* **Convolutional layers**:
  + Extract features from the input image using filters (3x3 in size).
  + First layer has 32 filters, subsequent layers have 64 filters.
  + **ReLU** activation function introduces non-linearity.
* **MaxPooling** layers:
  + Down-sample the image dimensions by taking the max value in 2x2 regions, reducing spatial size and computations.

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

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

* **Flatten**: Converts the 2D feature maps into a 1D vector.
* **Dense Layers**:
  + Fully connected layers. First layer has 64 neurons.
  + Final layer has 10 neurons (for 10 digit classes) and uses **softmax** for probabilistic outputs.

**4. Compile the Model**

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

* **Optimizer**: Adam (Adaptive Moment Estimation) adjusts learning rates dynamically.
* **Loss**: Categorical cross-entropy measures the difference between predicted and true distributions.
* **Metrics**: Accuracy evaluates model performance.

**5. Train the Model**

model.fit(train\_images, train\_labels, epochs=10, batch\_size=64, validation\_split=0.2)

* **Training data**: train\_images and train\_labels.
* **Epochs**: Number of passes through the entire training dataset.
* **Batch size**: Number of samples per gradient update.
* **Validation split**: Reserves 20% of training data for validation.

**6. Evaluate the Model**

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print(f'Test accuracy: {test\_acc}')

* Measures the loss and accuracy on the test dataset.

**7. Make Predictions**

predictions = model.predict(test\_images)

predicted\_labels = predictions.argmax(axis=1)

* **predict**: Produces probabilities for each class.
* **argmax**: Selects the class with the highest probability for each prediction.

**8. Visualize Predictions**

**Define Visualization Function**

def plot\_images(i, predictions\_array, true\_label, img):

predictions\_array, true\_label, img = predictions\_array[i], true\_label[i].argmax(), img[i].reshape(28, 28)

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted\_label = predictions\_array.argmax()

color = 'blue' if predicted\_label == true\_label else 'red'

plt.xlabel(f"Predicted: {predicted\_label} ({100 \* tf.reduce\_max(predictions\_array):.2f}%), Actual: {true\_label}", color=color)

* Displays:
  + The image.
  + Predicted label and probability.
  + True label.
* Label color: Blue if correct, red if incorrect.

**Plot Predictions**

num\_rows = 5

num\_cols = 3

num\_images = num\_rows \* num\_cols

plt.figure(figsize=(2 \* 2 \* num\_cols, 2 \* num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 1)

plot\_images(i, predictions, test\_labels, test\_images)

plt.show()

* Creates a grid of images and their predictions.
* Blue labels indicate correct predictions; red labels indicate incorrect predictions.

**Summary**

1. The model is trained on MNIST data to classify handwritten digits.
2. After training, the model is evaluated for accuracy.
3. Predictions are visualized to compare actual and predicted labels.

“This process demonstrates how CNNs can be used for image classification tasks.”