perplexity

Perfect, this is my final code:

a) Import the necessary packages

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
print("Imports success!")
```

b) Load the training and testing data (MNIST)

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

```
with np.load(r"C:\Users\kusha\Desktop\mnist_dataset.npz") as data:
x_train = data["X_train"]
y_train = data["y_train"]
x_test = data["X_test"]
y_test = data["y_test"]

print(
f"x_train:\n"
f" Data type: {x_train.dtype}\n"
f" Shape : {x_train.shape}\n"
f" Pixel value range: {x_train.min()} to {x_train.max()}\n\n"
)

print(
f"y_train:\n"
f" Data type: {y_train.dtype}\n"
f" Shape : {y_train.shape}\n"
f" First 10 labels: {y_train[:10]}\n"
)

print(
f"x_test:\n"
f" Data type: {x_test.dtype}\n"
f" Shape : {x_test.shape}\n"
```

f" Pixel value range: {x_test.min()} to {x_test.max()}\n\n"
)

print(
f"y_test:\n"
f" Data type: {y_test.dtype}\n"
f" Shape : {y_test.shape}\n"
f" First 10 labels: {y_test[:10]}\n"
)

print(f"First image sample of x_train (pixel values):\n{x_train[0]}\n")

c) Define the network architecture using Keras

Preprocess - normalize pixel values to [0, 1]

x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

print(
f"\nAfter normalization:\n"
f"x_train:\n"
f" Data type: {x_train.dtype}\n"
f" Shape : {x_train.shape}\n"
f" Pixel value range: {x_train.min()} to {x_train.max()}\n\n"

```
f"x_test:\n"
f"  Data type: {x_test.dtype}\n"
f"  Shape    : {x_test.shape}\n"
f"  Pixel value range: {x_test.min()} to {x_test.max()}\n\n"

f"  Sample normalized image of x_train (first image pixels):\n{x_train[0]}\n"
```

)

model = keras.Sequential([
layers.Input(shape=(28, 28)),
layers.Flatten(),
layers.Dense(128, activation="relu"),
layers.Dense(10, activation="softmax")
])
model.summary()

d) Train the model using SGD optimizer

```
model.compile(
optimizer='sgd',
loss='sparse_categorical_crossentropy',
metrics=['accuracy']
)
history = model.fit(
x_train, y_train,
validation_data=(x_test, y_test),
epochs=10,
verbose=2
)
```

e) Evaluate the network

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=1)
print(f"\nTest accuracy: {test_acc:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

Show predictions on test set samples

```
count = 11
predictions = model.predict(x_test[:count])
predicted_labels = np.argmax(predictions, axis=1)

plt.figure(figsize=(12, 5))
for i in range(count):
plt.subplot(1, count, i + 1)
plt.imshow(x_test[i], cmap='gray')
plt.title(f"Pred: {predicted_labels[i]}\nTrue: {y_test[i]}")
plt.axis('off')
plt.suptitle("Prediction vs Actual Labels on Test Samples")
plt.show()
```

f) Plot the training loss and accuracy

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss During Training')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy During Training')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()

explain it line by line, not by cell.

give in nice format so I can export your answer later to pdf

Here is the detailed line-by-line explanation of your final code, formatted for easy export to PDF:

a) Import the necessary packages

```
import numpy as np
```

- Imports NumPy library for numerical operations on arrays.

```
import matplotlib.pyplot as plt
```

- Imports Matplotlib's pyplot module for plotting and visualization.

```
import tensorflow as tf
```

- Imports TensorFlow framework for building and training deep learning models.

```
from tensorflow import keras
```

- Imports Keras API from TensorFlow for defining and training neural networks.

```
from tensorflow.keras import layers
```

- Imports the layers module from Keras, for constructing layers like Dense and Flatten.

```
print("Imports success!")
```

- Prints confirmation that imports were successful.

b) Load the training and testing data (MNIST)

```
with np.load(r"C:\Users\kusha\Desktop\mnist_dataset.npz") as data:
    x_train = data["X_train"]
    y_train = data["y_train"]
    x_test  = data["X_test"]
    y_test  = data["y_test"]
```

- Loads MNIST dataset from local `.npz` file using a raw string pathname.
- Extracts training images (`x_train`), training labels (`y_train`), test images (`x_test`), and test labels (`y_test`).

```
print(
    f"x_train:\n"
    f"  Data type: {x_train.dtype}\n"
    f"  Shape    : {x_train.shape}\n"
    f"  Pixel value range: {x_train.min()} to {x_train.max()}\n\n"
)
```

- Prints data type (uint8), shape ((60000, 28, 28)) and pixel value range (0–255) of training images.

```
print(
    f"y_train:\n"
    f"  Data type: {y_train.dtype}\n"
    f"  Shape    : {y_train.shape}\n"
    f"  First 10 labels: {y_train[:10]}\n"
)
```

- Prints data type (uint8), shape ((60000,)), and first 10 digit labels of training labels (e.g., `[5 0 4 1 9 2 1 3 1 4]`).

```
print(
    f"x_test:\n"
    f"  Data type: {x_test.dtype}\n"
    f"  Shape    : {x_test.shape}\n"
    f"  Pixel value range: {x_test.min()} to {x_test.max()}\n\n"
)
```

- Prints data type, shape ((10000, 28, 28)), and pixel value range of test images.

```
print(
    f"y_test:\n"
    f"  Data type: {y_test.dtype}\n"
    f"  Shape    : {y_test.shape}\n"
    f"  First 10 labels: {y_test[:10]}\n"
)
```

- Prints data type, shape ((10000,)), and first 10 digit labels of test labels.

```
print(f"First image sample of x_train (pixel values):\n{x_train[^0]}\n")
```

- Prints pixel intensity matrix of the first training image as a 28×28 array of integers between 0 and 255.

c) Define the network architecture using Keras and normalize inputs

```
x_train = x_train.astype("float32") / 255.0
```

- Converts pixel values from uint8 integers to float32 and normalizes to by dividing by 255.[1]

```
x_test  = x_test.astype("float32")  / 255.0
```

- Normalizes test inputs similarly.

```
print(
    f"\nAfter normalization:\n"
    f"x_train:\n"
    f"  Data type: {x_train.dtype}\n"
    f"  Shape    : {x_train.shape}\n"
    f"  Pixel value range: {x_train.min()} to {x_train.max()}\n\n"

    f"x_test:\n"
    f"  Data type: {x_test.dtype}\n"
    f"  Shape    : {x_test.shape}\n"
    f"  Pixel value range: {x_test.min()} to {x_test.max()}\n\n"

    f"  Sample normalized image of x_train (first image pixels):\n{x_train[^0]}\n"
)
```

- Prints data types (`float32`), shapes, ranges ([0.0, 1.0]) after normalization.
- Shows the first image pixels as floats between 0 and 1.

```
model = keras.Sequential([
    layers.Input(shape=(28, 28)),
    layers.Flatten(),
    layers.Dense(128, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

- Initializes a sequential neural network:
    - `Input`: expects image shape 28×28.
    - `Flatten`: converts image to 784-length vector.
    - `Dense(128, activation="relu")`: Fully connected hidden layer with 128 neurons and ReLU activation ($ ReLU(x) = \max(0,x) $, output range: [0, ∞)).

   - `Dense(10, activation="softmax")`: Output layer with 10 neurons and softmax activation ($\mathrm{softmax}(\mathbf{z})_i = e^{z_i} / \sum_j e^{z_j}$, outputs probability distribution over 10 classes, range, sum=1).[1]

```
model.summary()
```

- Displays network structure: layer names, output shapes, and trainable parameters:
   - `flatten`: converts (28,28) to (784), 0 parameters.
   - `dense`: 128 neurons, 100,480 parameters.
   - `dense_1`: 10 neurons, 1,290 parameters.
- Total parameters: 101,770 (all trainable).

d) Train the model using SGD optimizer

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

- Compiles model for training with:
   - SGD optimizer (Stochastic Gradient Descent).
   - Sparse categorical crossentropy loss (used for integer labels).
   - Track training and validation accuracy.

```
history = model.fit(
    x_train, y_train,
    validation_data=(x_test, y_test),
    epochs=10,
    verbose=2
)
```

- Fits/train model over 10 epochs on training set.
- Validates on test set each epoch.
- `verbose=2`: print one line per epoch showing accuracy, loss.

e) Evaluate the network performance

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=1)
```

- Evaluates model on test dataset; `verbose=1` shows progress bar.

```
print(f"\nTest accuracy: {test_acc:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

- Prints final test accuracy and loss with 4 decimal places.

Show predictions on test set samples

```
count = 11
```

- Number of test images to predict and display.

```
predictions = model.predict(x_test[:count])
```

- Generates predicted probabilities for first 11 test images.

```
predicted_labels = np.argmax(predictions, axis=1)
```

- Converts probabilities to predicted class indices by selecting highest probability per sample.

```
plt.figure(figsize=(12, 5))
for i in range(count):
    plt.subplot(1, count, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f"Pred: {predicted_labels[i]}\nTrue: {y_test[i]}")
    plt.axis('off')
plt.suptitle("Prediction vs Actual Labels on Test Samples")
plt.show()
```

- Displays test images horizontally with predicted and true labels above each.
- Hides axes ticks; adds central figure title; shows the figure.

f) Plot the training loss and accuracy

```
plt.figure(figsize=(12, 5))
```

- Creates a large figure.

```
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss During Training')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

- First subplot: plots training and validation loss vs epochs.
- Adds axes labels, legend, and title.

```
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy During Training')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

- Second subplot: plots training and validation accuracy vs epochs.
- Adds axes labels, legend, and title.

```
plt.show()
```

- Renders both plots.

This completes your requested line-by-line explanation with detailed descriptions of each line's purpose, function arguments, and outputs.

Let me know if you want any part expanded or formulas detailed!

⁂

1. DL_Lab_Manual.pdf
2. https://www.kaggle.com/datasets/valentynsichkar/mnist-preprocessed
3. https://www.geeksforgeeks.org/machine-learning/mnist-dataset/
4. https://stackoverflow.com/questions/27925358/pre-processing-before-digit-recognition-for-nn-cnn-trained-with-mnist-dataset
5. https://www.reddit.com/r/MachineLearning/comments/4u29y8/understanding_the_preprocessing_steps_in_the/
6. https://eitca.org/artificial-intelligence/eitc-ai-gcml-google-cloud-machine-learning/advancing-in-machine-learning/introduction-to-keras/examination-review-introduction-to-keras/what-are-the-steps-involved-in-preprocessing-the-fashion-mnist-dataset-before-training-the-model/
7. https://colab.research.google.com/github/vitay/lecturenotes-neurocomputing/blob/master/neurocomputing/5-exercises/9-MNIST.ipynb
8. https://www.oreilly.com/library/view/machine-learning-for/9781783980284/a487b013-2f0f-4d17-8eb3-05313116090a.xhtml
9. https://www.kaggle.com/code/damienbeneschi/mnist-eda-preprocessing-classifiers