Code 1

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

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

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

y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

model = Sequential([
    Input(shape=(28, 28)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(
    optimizer=SGD(learning_rate=0.01),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    x_train, y_train,
    epochs=20,
    validation_data=(x_test, y_test)
)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Loss : {test_loss}")
print(f"Test Accuracy : {test_acc}")

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 vs Epochs")
plt.xlabel('Epochs')
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 vs Epochs")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Code 2

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, Model, Input, optimizers

data_dir = "mnist-jpg"
img_size = (28, 28)
color_mode = "grayscale"

train_dir = os.path.join(data_dir, "train")
test_dir  = os.path.join(data_dir, "test")

train_ds = keras.preprocessing.image_dataset_from_directory(
    train_dir, labels="inferred", label_mode="int",
    image_size=img_size, color_mode=color_mode,
    batch_size=128, shuffle=True, seed=42
)
test_ds = keras.preprocessing.image_dataset_from_directory(
    test_dir, labels="inferred", label_mode="int",
    image_size=img_size, color_mode=color_mode,
    batch_size=128, shuffle=False
)

def ds_to_numpy(dataset):
    imgs, labels = [], []
    for batch in dataset:
        b_x, b_y = batch
        imgs.append(b_x.numpy())
        labels.append(b_y.numpy())
    return np.vstack(imgs), np.hstack(labels)

x_train, y_train = ds_to_numpy(train_ds)
```

```
x_test, y_test = ds_to_numpy(test_ds)

x_train = x_train[..., 0]
x_test  = x_test[..., 0]
# scale pixel values to [0,1]
x_train = x_train.astype("float32") / 255.0
x_test  = x_test.astype("float32")  / 255.0

inp = Input(shape=x_train.shape[1:], name="input_image")
x = layers.Flatten(name="flatten")(inp)
x = layers.Dense(128, activation="relu", name="dense1")(x)
x = layers.Dense(64, activation="relu", name="dense2")(x)
out = layers.Dense(10, activation="softmax", name="preds")(x)
model = Model(inputs=inp, outputs=out, name="ffnn_mnistjpg")

sgd = optimizers.SGD(learning_rate=0.01, momentum=0.0)
model.compile(optimizer=sgd,
            loss="sparse_categorical_crossentropy",
            metrics=["accuracy"])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.1,
                    verbose=2)


test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\nTest loss: {test_loss:.4f} | Test accuracy: {test_acc:.4f}")

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history["loss"], label="train loss")
plt.plot(history.history["val_loss"], label="val loss")
plt.title("Loss"); plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history["accuracy"], label="train acc")
plt.plot(history.history["val_accuracy"], label="val acc")
plt.title("Accuracy"); plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend()

plt.tight_layout()
plt.show()
```

1. Introduction

A **Feedforward Neural Network (FFNN)** is the simplest type of Artificial Neural Network (ANN).
It consists of **neurons (nodes)** arranged in **layers** — where data flows only in **one direction**,
from **input** → **hidden layers** → **output**, with **no feedback connections**.

⚙️ 2. Structure of FFNN

1. **Input Layer:**
    - Takes the input features (e.g., pixel values, sensor readings, etc.).
    - Passes them to the next layer.
2. **Hidden Layers:**
    - Perform intermediate computations.
    - Each neuron applies a **weighted sum** and **activation function** to introduce non-linearity.
3. **Output Layer:**
    - Produces the final result (e.g., class label or numeric value).

🧮 3. Mathematical Model

For each neuron:

z=∑i=1n(wixi)+bz = \sum_{i=1}^{n} (w_i x_i) + bz=i=1∑n(wixi)+b y=f(z)y = f(z)y=f(z)

Where:

- xix_ixi: input features
- wiw_iwi: weights
- bbb: bias

- f(z)f(z)f(z): activation function (e.g., ReLU, sigmoid)
- yyy: output of the neuron

---

4. Working Process

1. **Forward Propagation:**
    - Input data passes through the network layer by layer.
    - Each neuron computes its activation value.
2. **Loss Calculation:**
    - The difference between the predicted output and actual output is calculated using a **loss function**.
3. **Backward Propagation (Backpropagation):**
    - The loss is propagated backward.
    - The model updates weights using **Gradient Descent** or similar optimization techniques.
4. **Iteration:**
    - Steps 1–3 repeat for several **epochs** until loss is minimized.

10. Advantages of FFNN

- Simple and easy to implement
- Works well for structured data
- Forms the base of advanced networks (CNNs, RNNs, etc.)

⚠️ 11. Limitations

- Cannot handle sequential or spatial data directly
- Prone to overfitting on small datasets
- Requires large data for good accuracy

🧾 12. Conclusion

Feedforward Neural Networks are the **foundation of deep learning**.
They learn complex relationships between input and output using **forward propagation and backpropagation**.
The MNIST experiment demonstrates their ability to classify handwritten digits with high accuracy.