Exp:4

14.08.25

BUILD A SIMPLE FEED FORWARD NEURAL NETWORK TO RECOGNIZE HANDWRITTEN CHARACTER.

Aim: To build a simple feed forward neural network to recognize handwritten character.

Pseudo code:

```
1. Import necessary libraries.
   → (numpy, tensorflow / keras, matplotlib)
2. Load dataset
   → (e.g: MNIST / EMNIST of handwritten character)
3. Preprocess data
   a. Normalize pixel values to range [0,1]
   b. Flatten each image into a 1D vector.
   c. Convert labels into on-hot encoding.
4. Define FFNN architecture.
   a. Input layer: size = no. of pixels per image
   b. Hidden layers: fully connecter neurons with
      activation (ReLU)
   c. Output layer: size = no. of character classes,
      activation (Softmax).
5. Compile the model
   a. loss function: Categorical Cross-Entropy.
   b. Optimizers: SGD / Adam.
   c. Metrics: Accuracy, F1 score, recall.
6. Train the model.
   a. Fit training data for N epochs.
   b. Validate on test data.
```

7. Evaluate model performance.

a. Test accuracy

b. Confusion matrix and classification report

8. Display sample predictions w/ true labels.

Justification :

1. Choice of model. (FFNN):

→ Represents the foundational arch. of Deep learning

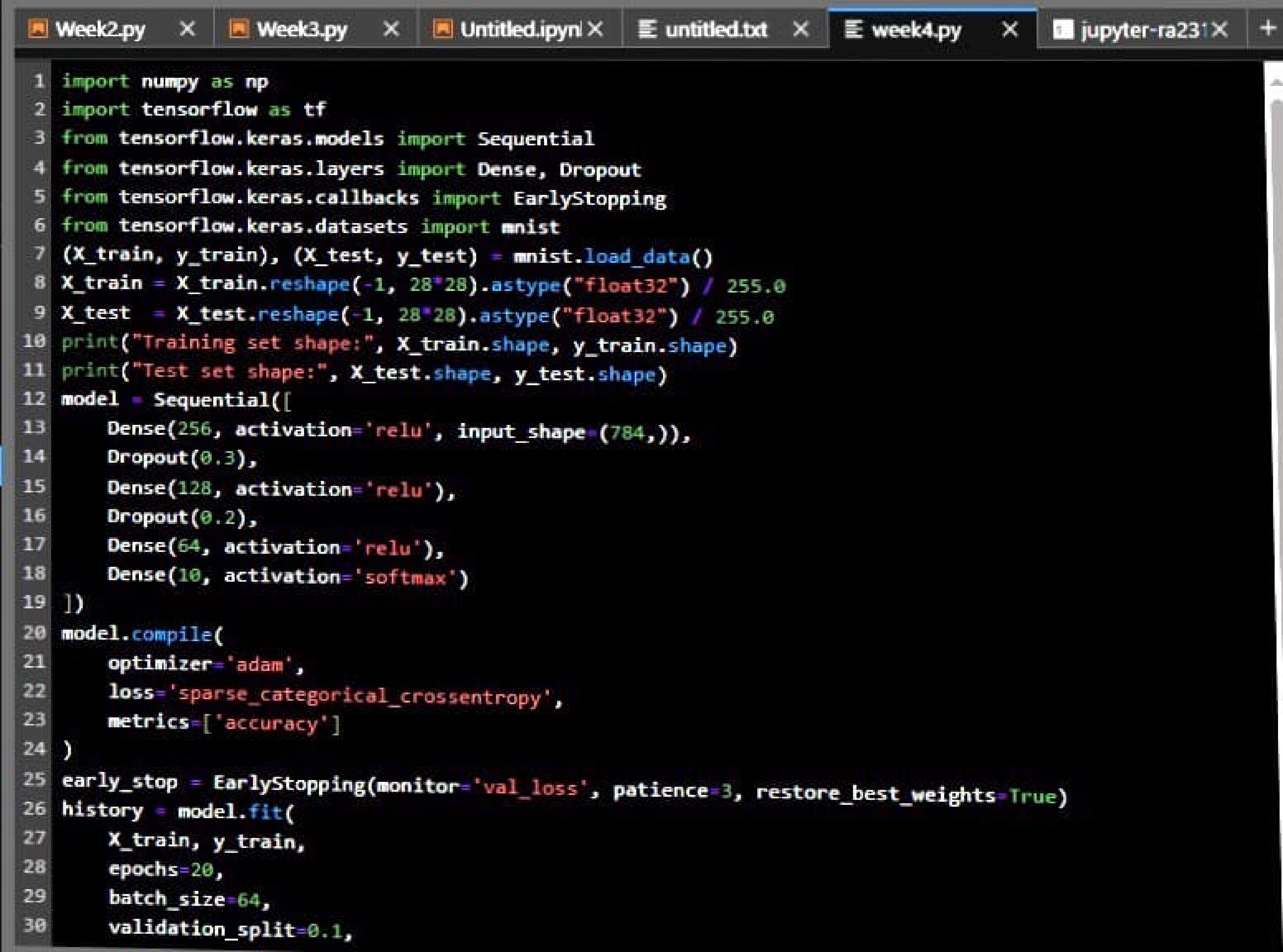
2. Learning outcome:

- Understanding of NN layers & activation func.
- Practical exposure to training, validation & testing in supervised learning

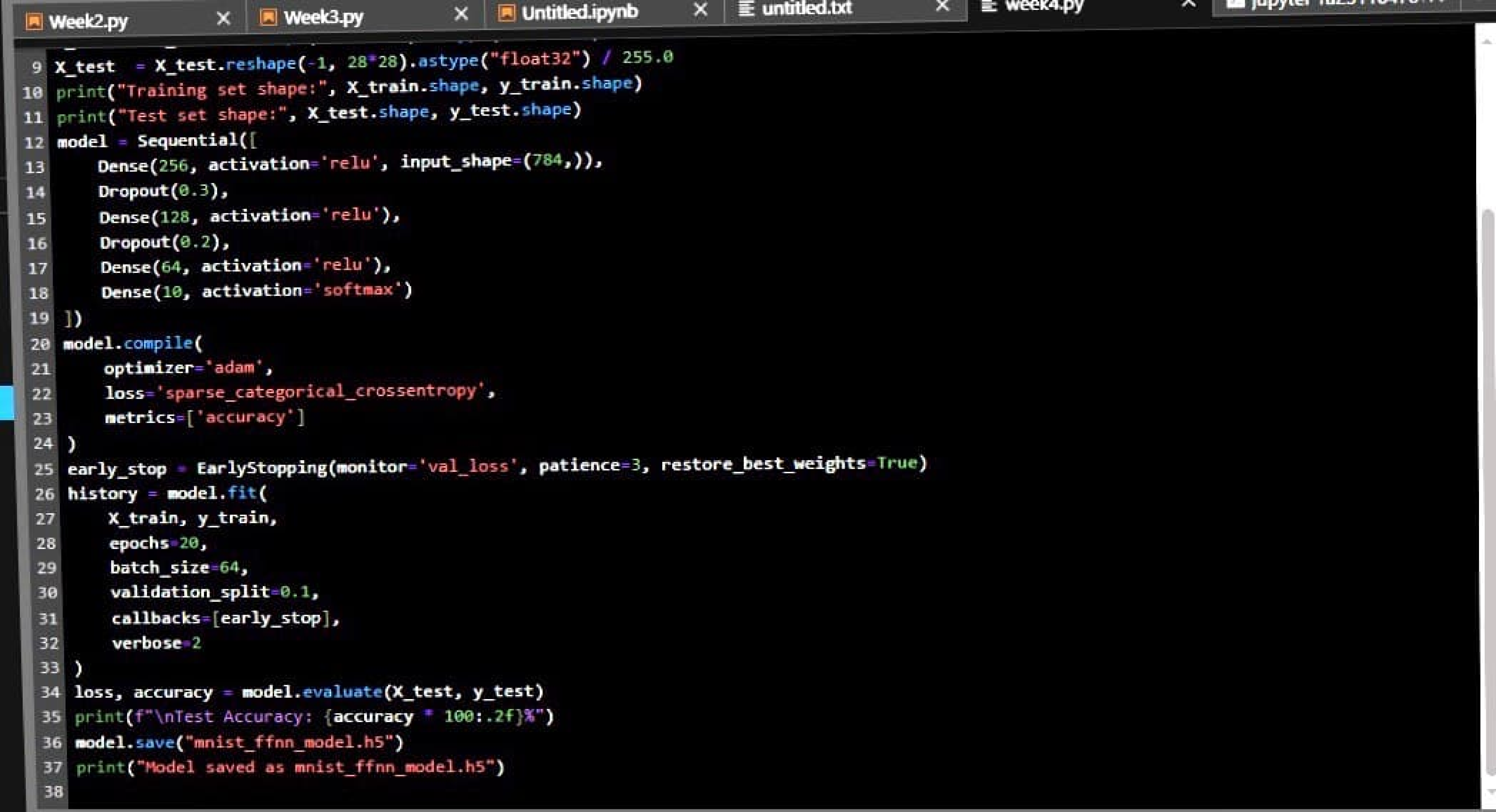
Observation.

1. Training & Validation Accuracy :

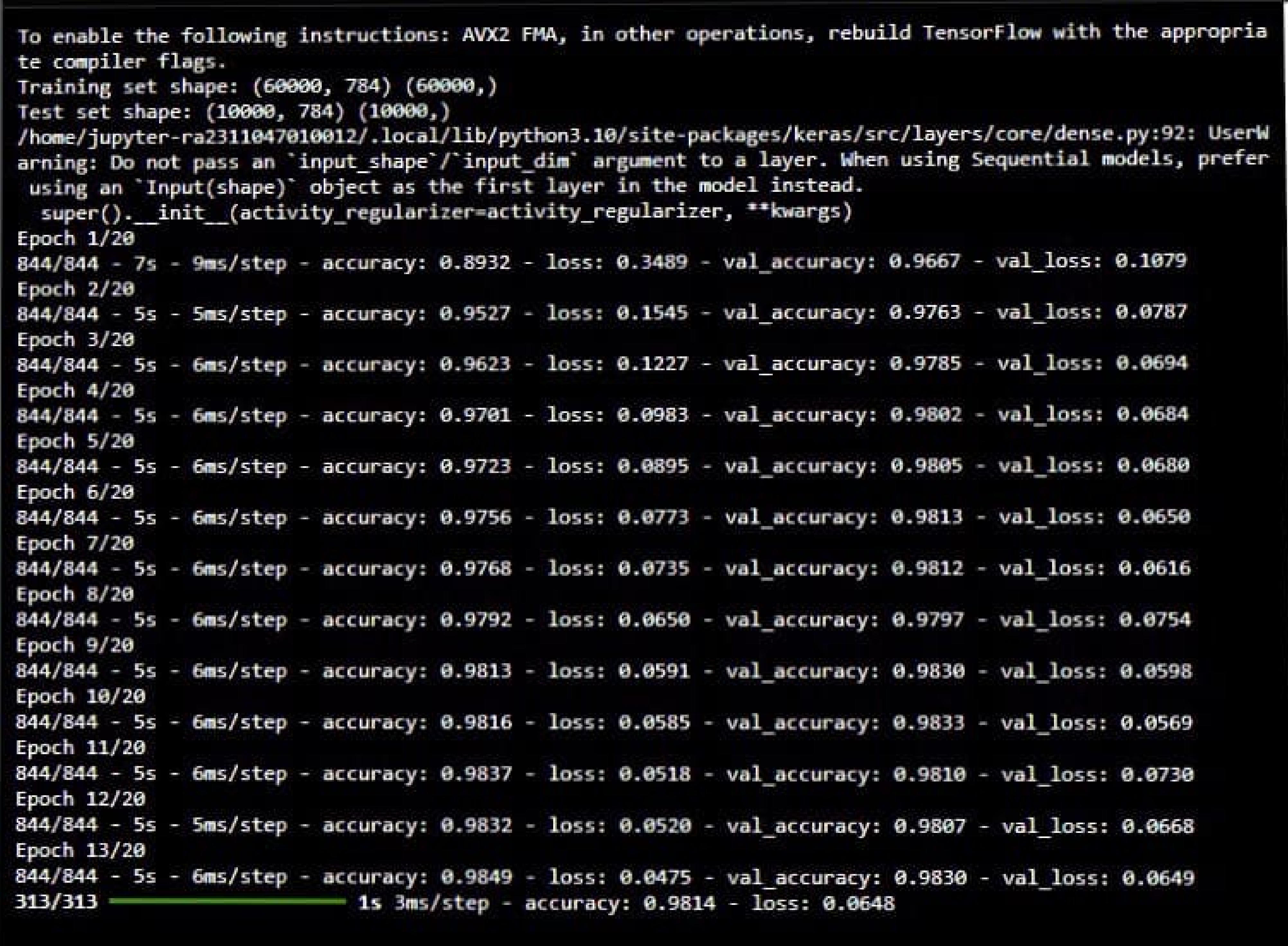
→ Training accuracy ~ 98.85% after 20 epochs.
Validation accuracy ~ 98.47%. indicating model generalizes well to unseen data.



```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28*28).astype("float32") / 255.0
X_test  = X_test.reshape(-1, 28*28).astype("float32") / 255.0
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
model = Sequential([
    Dense(256, activation='relu', input_shape=(784,)),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=64,
    validation_split=0.1,
```



```python
X_test  = X_test.reshape(-1, 28*28).astype("float32") / 255.0
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
model = Sequential([
    Dense(256, activation='relu', input_shape=(784,)),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=64,
    validation_split=0.1,
    callbacks=[early_stop],
    verbose=2
)
loss, accuracy = model.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {accuracy * 100:.2f}%")
model.save("mnist_ffnn_model.h5")
print("Model saved as mnist_ffnn_model.h5")
```



```
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropria
te compiler flags.
Training set shape: (60000, 784) (60000,)
Test set shape: (10000, 784) (10000,)
/home/jupyter-ra2311047010012/.local/lib/python3.10/site-packages/keras/src/layers/core/dense.py:92: UserW
arning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer
 using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
844/844 - 7s - 9ms/step - accuracy: 0.8932 - loss: 0.3489 - val_accuracy: 0.9667 - val_loss: 0.1079
Epoch 2/20
844/844 - 5s - 5ms/step - accuracy: 0.9527 - loss: 0.1545 - val_accuracy: 0.9763 - val_loss: 0.0787
Epoch 3/20
844/844 - 5s - 6ms/step - accuracy: 0.9623 - loss: 0.1227 - val_accuracy: 0.9785 - val_loss: 0.0694
Epoch 4/20
844/844 - 5s - 6ms/step - accuracy: 0.9701 - loss: 0.0983 - val_accuracy: 0.9802 - val_loss: 0.0684
Epoch 5/20
844/844 - 5s - 6ms/step - accuracy: 0.9723 - loss: 0.0895 - val_accuracy: 0.9805 - val_loss: 0.0680
Epoch 6/20
844/844 - 5s - 6ms/step - accuracy: 0.9756 - loss: 0.0773 - val_accuracy: 0.9813 - val_loss: 0.0650
Epoch 7/20
844/844 - 5s - 6ms/step - accuracy: 0.9768 - loss: 0.0735 - val_accuracy: 0.9812 - val_loss: 0.0616
Epoch 8/20
844/844 - 5s - 6ms/step - accuracy: 0.9792 - loss: 0.0650 - val_accuracy: 0.9797 - val_loss: 0.0754
Epoch 9/20
844/844 - 5s - 6ms/step - accuracy: 0.9813 - loss: 0.0591 - val_accuracy: 0.9830 - val_loss: 0.0598
Epoch 10/20
844/844 - 5s - 6ms/step - accuracy: 0.9816 - loss: 0.0585 - val_accuracy: 0.9833 - val_loss: 0.0569
Epoch 11/20
844/844 - 5s - 6ms/step - accuracy: 0.9837 - loss: 0.0518 - val_accuracy: 0.9810 - val_loss: 0.0730
Epoch 12/20
844/844 - 5s - 5ms/step - accuracy: 0.9832 - loss: 0.0520 - val_accuracy: 0.9807 - val_loss: 0.0668
Epoch 13/20
844/844 - 5s - 6ms/step - accuracy: 0.9849 - loss: 0.0475 - val_accuracy: 0.9830 - val_loss: 0.0649
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9814 - loss: 0.0648
```