Annexure 2

Introduction to AI and ML
CSL236

Project Report

Faculty name: Dr. Nidhi Malik

Student name: Sakshi

Roll No.:21csu419

Semester: 5th

Group: DS-B (DS5)

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122001, India

Session 2023-24

Table of Contents

1. Project Description

Our project, "Image Recognition Model using MNIST Fashion Dataset with TensorFlow," signifies a practical exploration of the theoretical concepts we've acquired thus far. This project delves into the vibrant intersection of artificial intelligence and machine learning, particularly within the captivating domain of computer vision. Our ambition is to construct a robust image recognition model capable of accurately categorizing diverse fashion items, a venture that aligns not only with our academic pursuits but also reflects our commitment to staying at the forefront of emerging technologies.

The project's cornerstone is the utilization of the MNIST Fashion Dataset, an extension of the classic MNIST dataset. This dataset, comprising 60,000 training images and 10,000 testing images, spans ten distinct fashion categories. This choice not only introduces us to the complexities of image classification but also places our project in the context of real-world challenges faced by the fashion industry. The selection of TensorFlow as our primary tool underscores our dedication to leveraging state-of-the-art technologies. TensorFlow's open-source framework provides the flexibility required to construct and train intricate neural network architectures, making it an ideal platform for implementing our image recognition model.

Beyond its academic significance, this project holds practical implications for automating tasks within the fashion industry. The ability to automatically classify fashion items from images has the potential to revolutionize inventory management, e-commerce, and trend analysis. By tackling this project, we aim not only to deepen our understanding of machine learning but also to contribute to the ongoing transformation of industries through the practical application of advanced technologies. As we navigate through our academic journey, this project stands as a testament to our commitment to bridging the gap between theory and practical, preparing us for the challenges and opportunities that lie ahead in the rapidly evolving field of computer science and data science.

2. Problem Statement

In the realm of the fashion industry, the surge in visual data presents a challenge: the efficient and accurate classification of diverse fashion items. This manual process is not only time-consuming but also prone to human errors. Our project addresses this by aiming to automate the classification of fashion items using advanced machine learning techniques. The specific goal is to develop an image recognition model capable of accurately identifying various fashion items within the MNIST Fashion Dataset, which includes 60,000 training images and 10,000 testing images across ten different fashion categories. The complexity of fashion items, with their diverse patterns and textures, necessitates the development of a sophisticated model capable of nuanced understanding for accurate classification. The potential benefits extend to streamlining inventory management in retail and enhancing user experience in e-commerce through more accurate search results and recommendations.

3. Analysis

3.1 Hardware Requirement

The hardware requirements for this project are moderate, and a standard laptop or desktop with a decent CPU and GPU would suffice. A machine with a minimum of 8GB RAM is recommended for smooth training and testing processes.

3.2 Software Requirement

The essential software components for this project include:

- Python (3.6 or higher)
- TensorFlow (2.x)
- Jupyter Notebook or any preferred Python IDE
- Matplotlib and NumPy for data visualization and manipulation

4. Design

4.1 Data/Input Output Description

The MNIST Fashion Dataset consists of 60,000 training images and 10,000 testing images of 10 different fashion categories. Each image is a 28x28 pixel grayscale image. The input to the model is the pixel values of these images, and the output is the predicted class label corresponding to the fashion item.

4.2 Algorithmic Approach

The revised algorithmic approach involves configuring the layers of the Classification MLP. The model is structured with multiple dense layers, each equipped with rectified linear unit (ReLU) activation functions to introduce non-linearity. The final layer employs a softmax activation to generate class probabilities. We employ categorical cross-entropy as the loss function, suitable for multi-class classification tasks, and the Adam optimizer for efficient model training.

5. Implementation and Testing

The project unfolds in stages to systematically implement and test the MLP-based image recognition model:

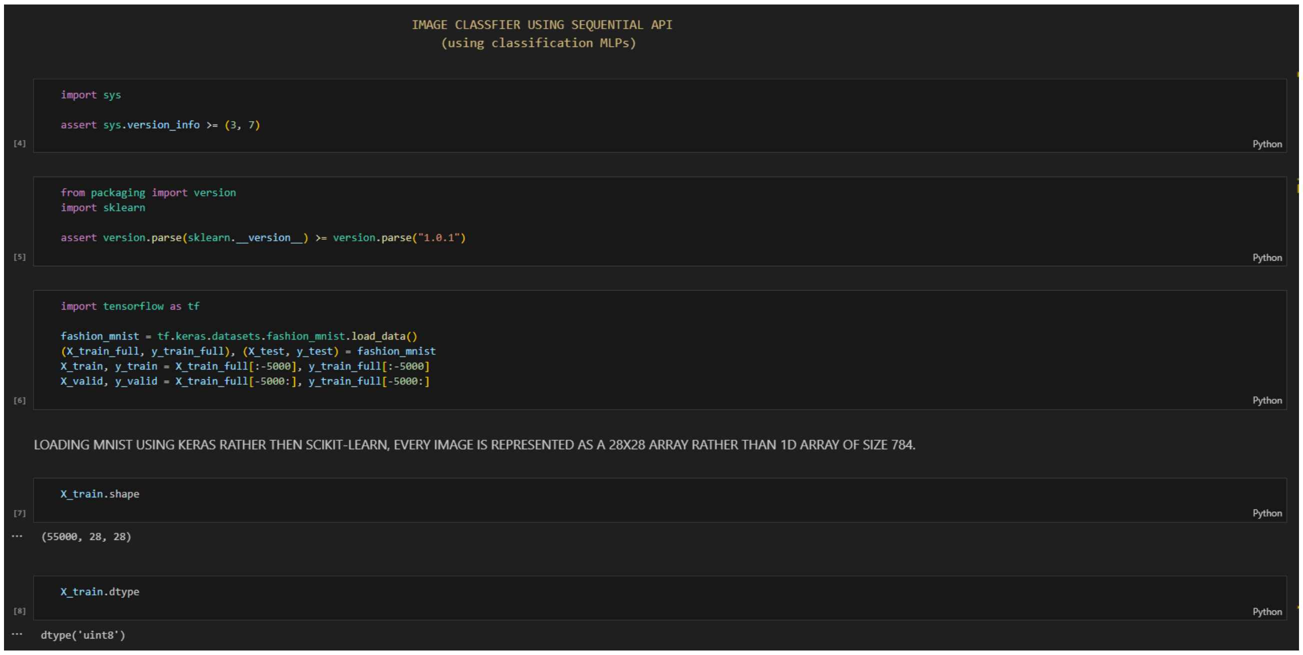
1. **Data Preprocessing:** This stage remains consistent with the initial approach, involving loading and preprocessing the MNIST Fashion Dataset, including normalization of pixel values and one-hot encoding labels.

2. **Model Architecture:** The focus shifts to configuring the MLP architecture. Layers are defined with appropriate activation functions and output dimensions to facilitate accurate classification.

3. **Training:** The model is trained on the pre-processed training dataset using the Adam optimizer and categorical cross-entropy loss. Monitoring training/validation accuracy and loss remains integral to assessing the model's performance.

4. **Testing:** The MLP model is evaluated on the testing dataset to gauge its effectiveness in classifying fashion items accurately.

6. Output (Screenshots)



```
IMAGE CLASSFIER USING SEQUENTIAL API
(using classification MLPs)
```

```
[4]
import sys

assert sys.version_info >= (3, 7)
```

```
[5]
from packaging import version
import sklearn

assert version.parse(sklearn.__version__) >= version.parse("1.0.1")
```

```
[6]
import tensorflow as tf

fashion_mnist = tf.keras.datasets.fashion_mnist.load_data()
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist
X_train, y_train = X_train_full[:-5000], y_train_full[:-5000]
X_valid, y_valid = X_train_full[-5000:], y_train_full[-5000:]
```

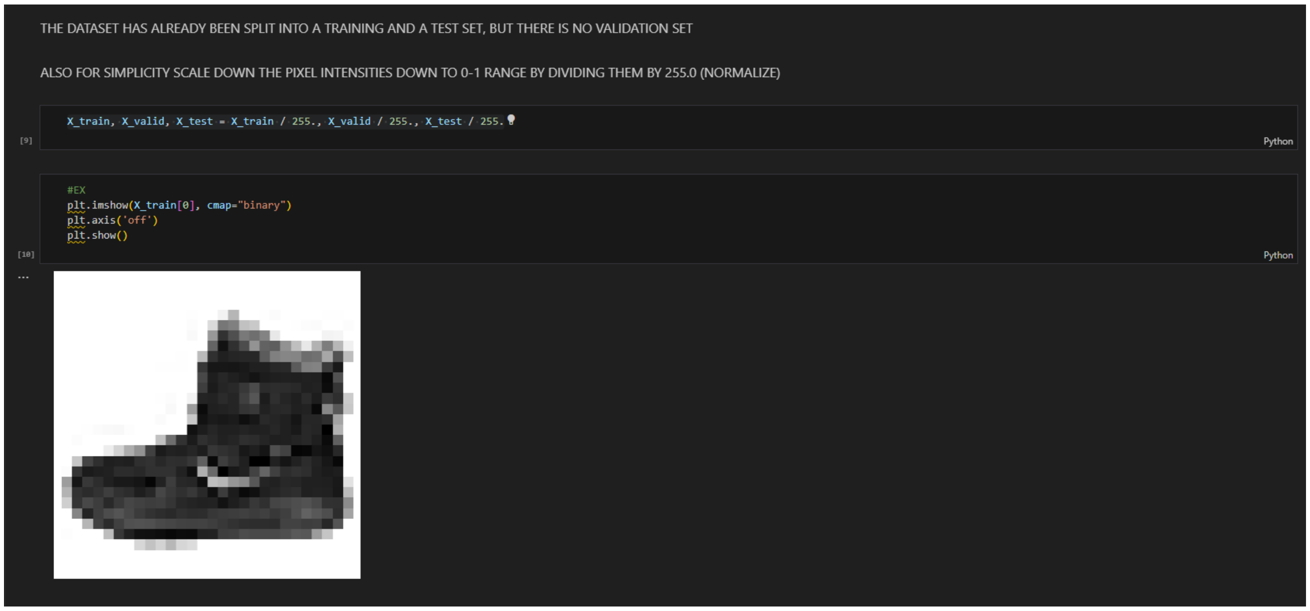
LOADING MNIST USING KERAS RATHER THEN SCIKIT-LEARN, EVERY IMAGE IS REPRESENTED AS A 28X28 ARRAY RATHER THAN 1D ARRAY OF SIZE 784.

```
[7]
X_train.shape
```

```
(55000, 28, 28)
```

```
[8]
X_train.dtype
```

```
dtype('uint8')
```

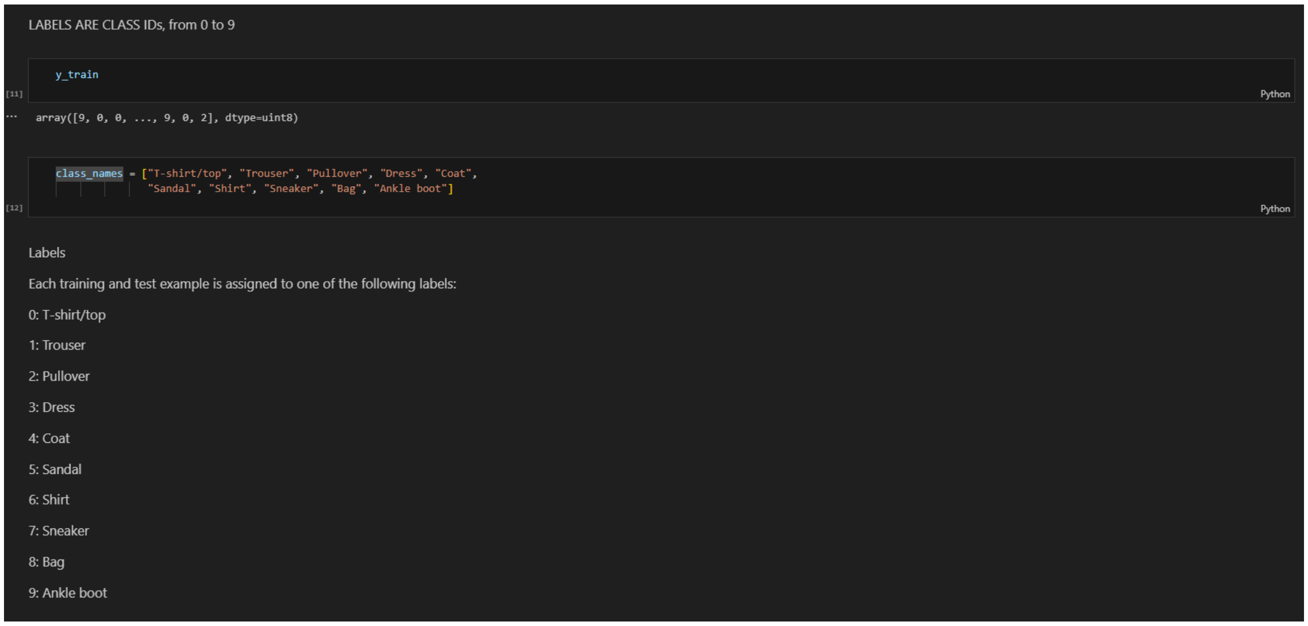


THE DATASET HAS ALREADY BEEN SPLIT INTO A TRAINING AND A TEST SET, BUT THERE IS NO VALIDATION SET

ALSO FOR SIMPLICITY SCALE DOWN THE PIXEL INTENSITIES DOWN TO 0-1 RANGE BY DIVIDING THEM BY 255.0 (NORMALIZE)

```
[9]
X_train, X_valid, X_test = X_train / 255., X_valid / 255., X_test / 255.
```

```
[10]
#EX
plt.imshow(X_train[0], cmap="binary")
plt.axis('off')
plt.show()
```



LABELS ARE CLASS IDs, from 0 to 9

```
y_train
```

```
array([9, 0, 0, ..., 9, 0, 2], dtype=uint8)
```

```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

Labels

Each training and test example is assigned to one of the following labels:

0: T-shirt/top

1: Trouser

2: Pullover

3: Dress

4: Coat

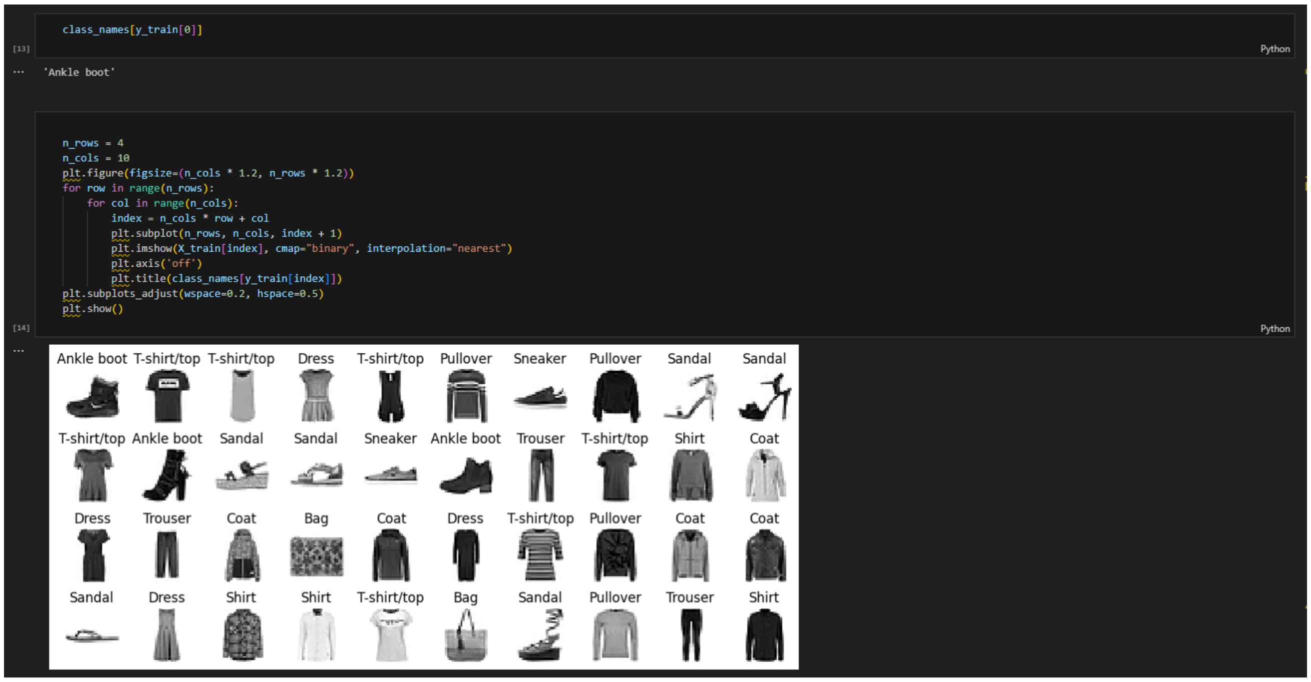
5: Sandal

6: Shirt

7: Sneaker

8: Bag

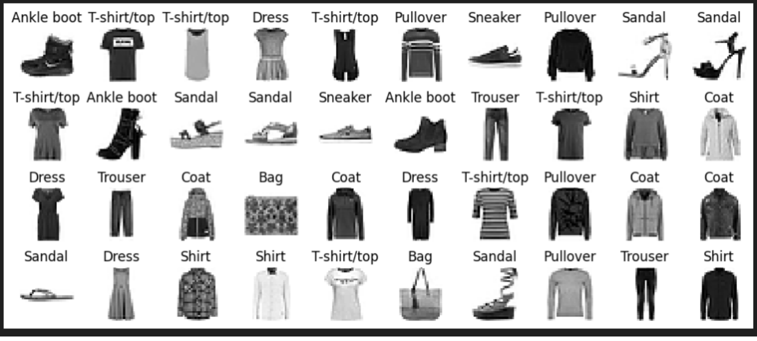
9: Ankle boot

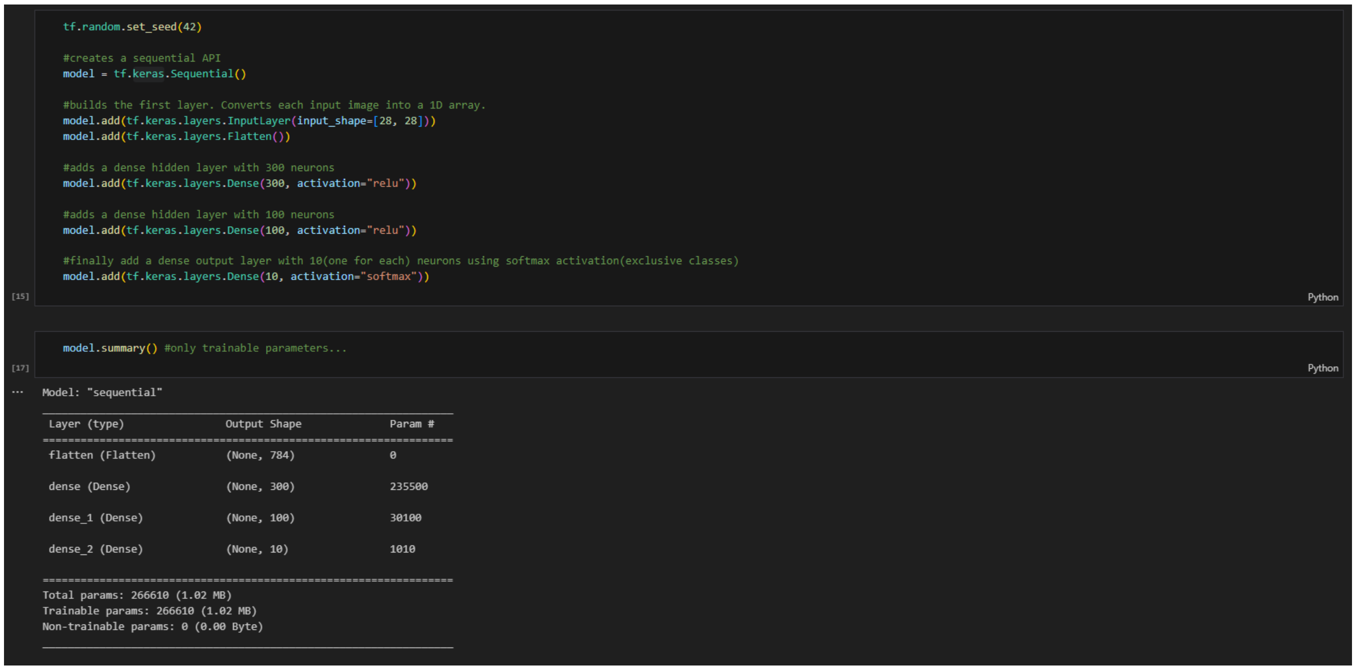


```
class_names[y_train[0]]
```

```
'Ankle boot'
```

```
n_rows = 4
n_cols = 10
plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(class_names[y_train[index]])
plt.subplots_adjust(wspace=0.2, hspace=0.5)
plt.show()
```





```
tf.random.set_seed(42)

#creates a sequential API
model = tf.keras.Sequential()

#builds the first layer. Converts each input image into a 1D array.
model.add(tf.keras.layers.InputLayer(input_shape=[28, 28]))
model.add(tf.keras.layers.Flatten())

#adds a dense hidden layer with 300 neurons
model.add(tf.keras.layers.Dense(300, activation="relu"))

#adds a dense hidden layer with 100 neurons
model.add(tf.keras.layers.Dense(100, activation="relu"))

#finally add a dense output layer with 10(one for each) neurons using softmax activation(exclusive classes)
model.add(tf.keras.layers.Dense(10, activation="softmax"))
```

```
model.summary() #only trainable parameters...
```

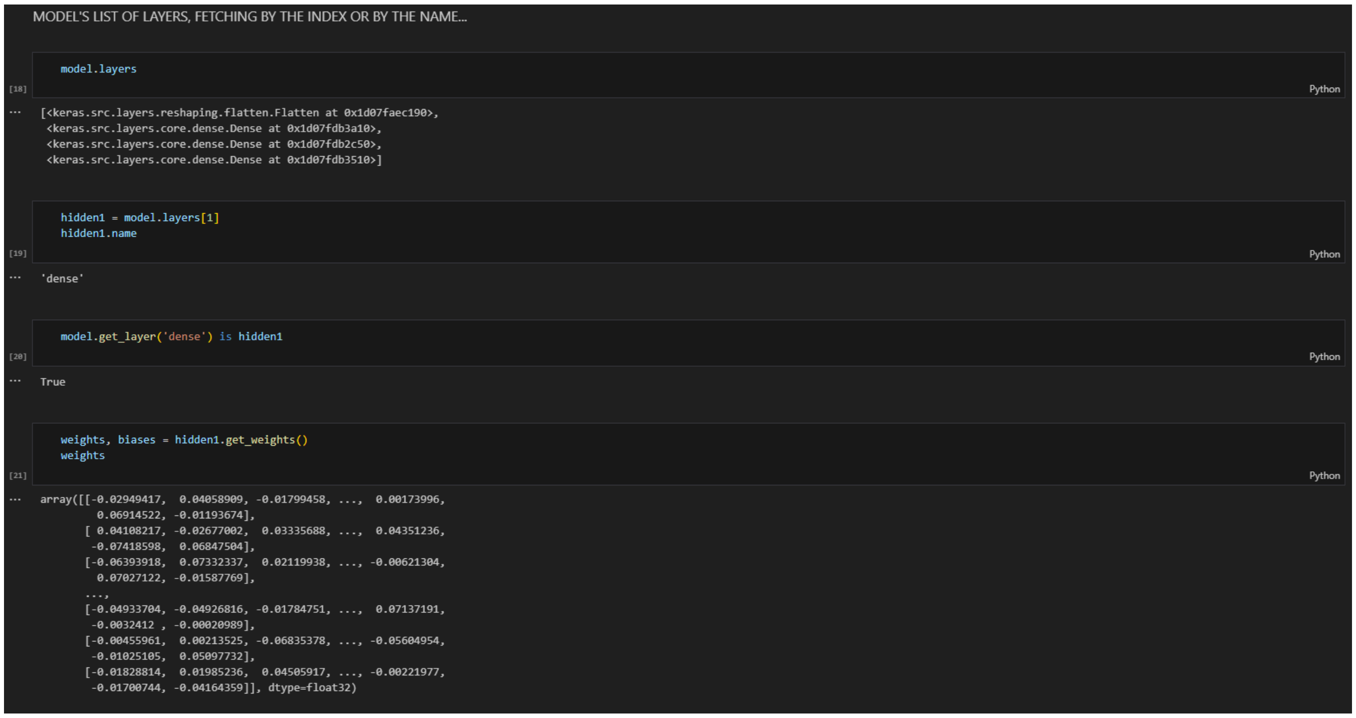
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 300)               235500

 dense_1 (Dense)             (None, 100)               30100

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 266610 (1.02 MB)
Trainable params: 266610 (1.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



MODEL'S LIST OF LAYERS, FETCHING BY THE INDEX OR BY THE NAME...

```
model.layers
```

```
[<keras.src.layers.reshaping.flatten.Flatten at 0x1d07faec190>,
 <keras.src.layers.core.dense.Dense at 0x1d07fdb3a10>,
 <keras.src.layers.core.dense.Dense at 0x1d07fdb2c50>,
 <keras.src.layers.core.dense.Dense at 0x1d07fdb3510>]
```

```
hidden1 = model.layers[1]
hidden1.name
```

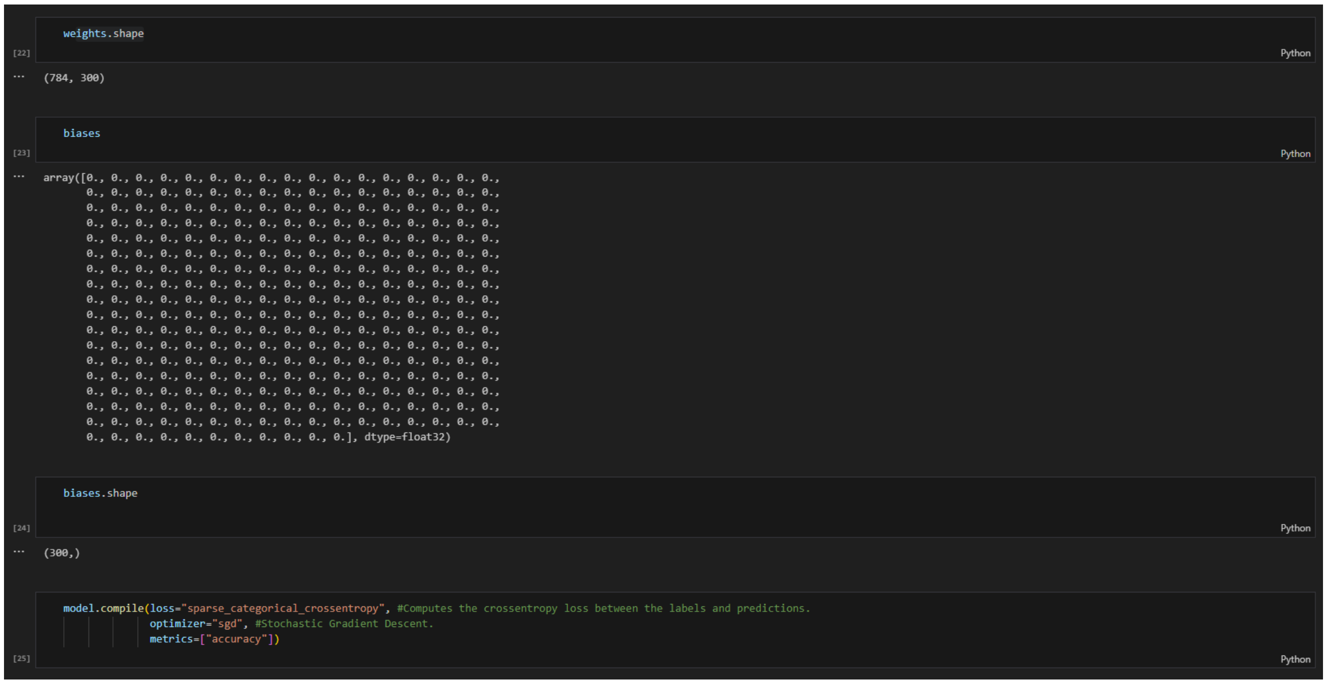
```
'dense'
```

```
model.get_layer('dense') is hidden1
```

```
True
```

```
weights, biases = hidden1.get_weights()
weights
```

```
array([[-0.02949417,  0.04058909, -0.01799458, ...,  0.00173996,
         0.06914522, -0.01193674],
       [ 0.04108217, -0.02677002,  0.03335688, ...,  0.04351236,
        -0.07418598,  0.06847504],
       [-0.06393918,  0.07332337,  0.02119938, ..., -0.00621304,
         0.07027122, -0.01587769],
       ...,
       [-0.04933704, -0.04926816, -0.01784751, ...,  0.07137191,
        -0.0032412 , -0.00020989],
       [-0.00455961,  0.00213525, -0.06835378, ..., -0.05604954,
        -0.01025105,  0.05097732],
       [-0.01828814,  0.01985236,  0.04505917, ..., -0.00221977,
        -0.01700744, -0.04164359]], dtype=float32)
```



```
[22]
weights.shape

(784, 300)
```

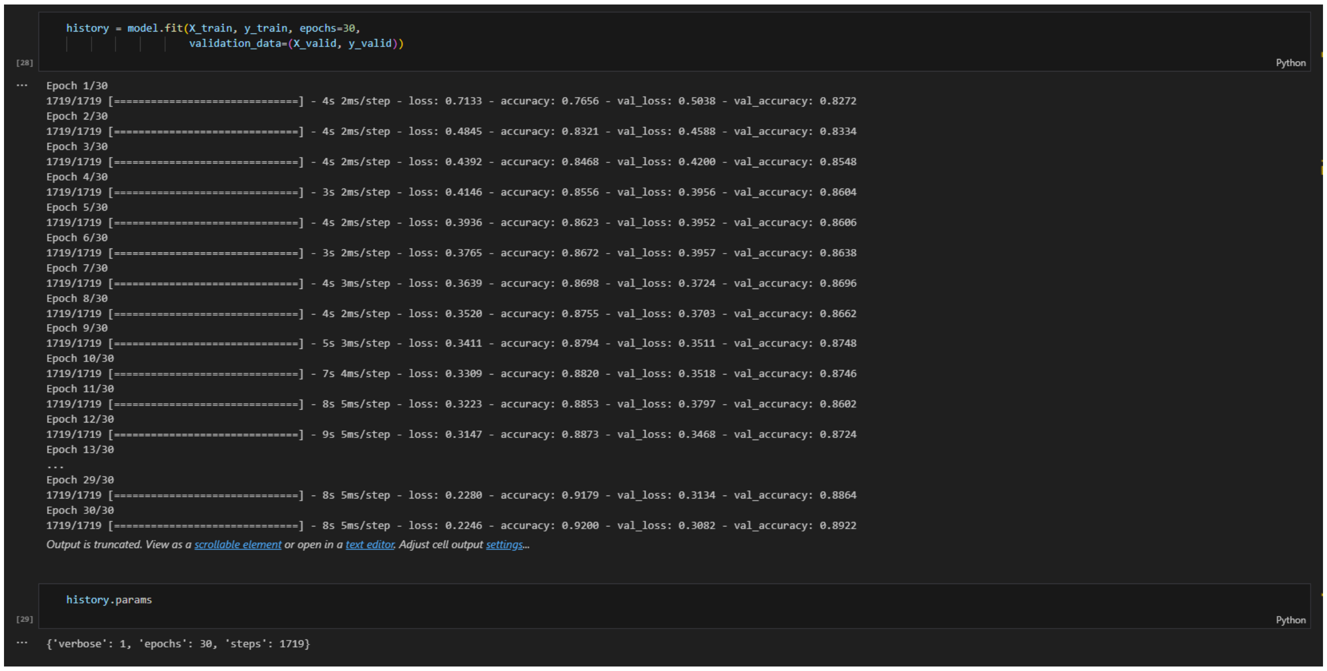
```
[23]
biases

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
[24]
biases.shape

(300,)
```

```
[25]
model.compile(loss="sparse_categorical_crossentropy", #Computes the crossentropy loss between the labels and predictions.
              optimizer="sgd", #Stochastic Gradient Descent.
              metrics=["accuracy"])
```



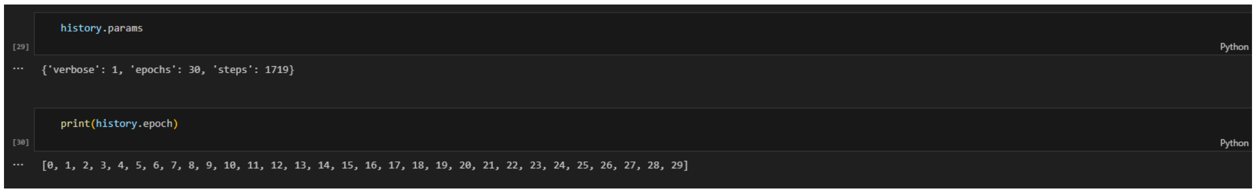
```
[28]
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))

Epoch 1/30
1719/1719 [==============================] - 4s 2ms/step - loss: 0.7133 - accuracy: 0.7656 - val_loss: 0.5038 - val_accuracy: 0.8272
Epoch 2/30
1719/1719 [==============================] - 4s 2ms/step - loss: 0.4845 - accuracy: 0.8321 - val_loss: 0.4588 - val_accuracy: 0.8334
Epoch 3/30
1719/1719 [==============================] - 4s 2ms/step - loss: 0.4392 - accuracy: 0.8468 - val_loss: 0.4200 - val_accuracy: 0.8548
Epoch 4/30
1719/1719 [==============================] - 3s 2ms/step - loss: 0.4146 - accuracy: 0.8556 - val_loss: 0.3956 - val_accuracy: 0.8604
Epoch 5/30
1719/1719 [==============================] - 4s 2ms/step - loss: 0.3936 - accuracy: 0.8623 - val_loss: 0.3952 - val_accuracy: 0.8606
Epoch 6/30
1719/1719 [==============================] - 3s 2ms/step - loss: 0.3765 - accuracy: 0.8672 - val_loss: 0.3957 - val_accuracy: 0.8638
Epoch 7/30
1719/1719 [==============================] - 4s 3ms/step - loss: 0.3639 - accuracy: 0.8698 - val_loss: 0.3724 - val_accuracy: 0.8696
Epoch 8/30
1719/1719 [==============================] - 4s 2ms/step - loss: 0.3520 - accuracy: 0.8755 - val_loss: 0.3703 - val_accuracy: 0.8662
Epoch 9/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3411 - accuracy: 0.8794 - val_loss: 0.3511 - val_accuracy: 0.8748
Epoch 10/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.3309 - accuracy: 0.8820 - val_loss: 0.3518 - val_accuracy: 0.8746
Epoch 11/30
1719/1719 [==============================] - 8s 5ms/step - loss: 0.3223 - accuracy: 0.8853 - val_loss: 0.3797 - val_accuracy: 0.8602
Epoch 12/30
1719/1719 [==============================] - 9s 5ms/step - loss: 0.3147 - accuracy: 0.8873 - val_loss: 0.3468 - val_accuracy: 0.8724
Epoch 13/30
...
Epoch 29/30
1719/1719 [==============================] - 8s 5ms/step - loss: 0.2280 - accuracy: 0.9179 - val_loss: 0.3134 - val_accuracy: 0.8864
Epoch 30/30
1719/1719 [==============================] - 8s 5ms/step - loss: 0.2246 - accuracy: 0.9200 - val_loss: 0.3082 - val_accuracy: 0.8922
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
[29]
history.params

{'verbose': 1, 'epochs': 30, 'steps': 1719}
```

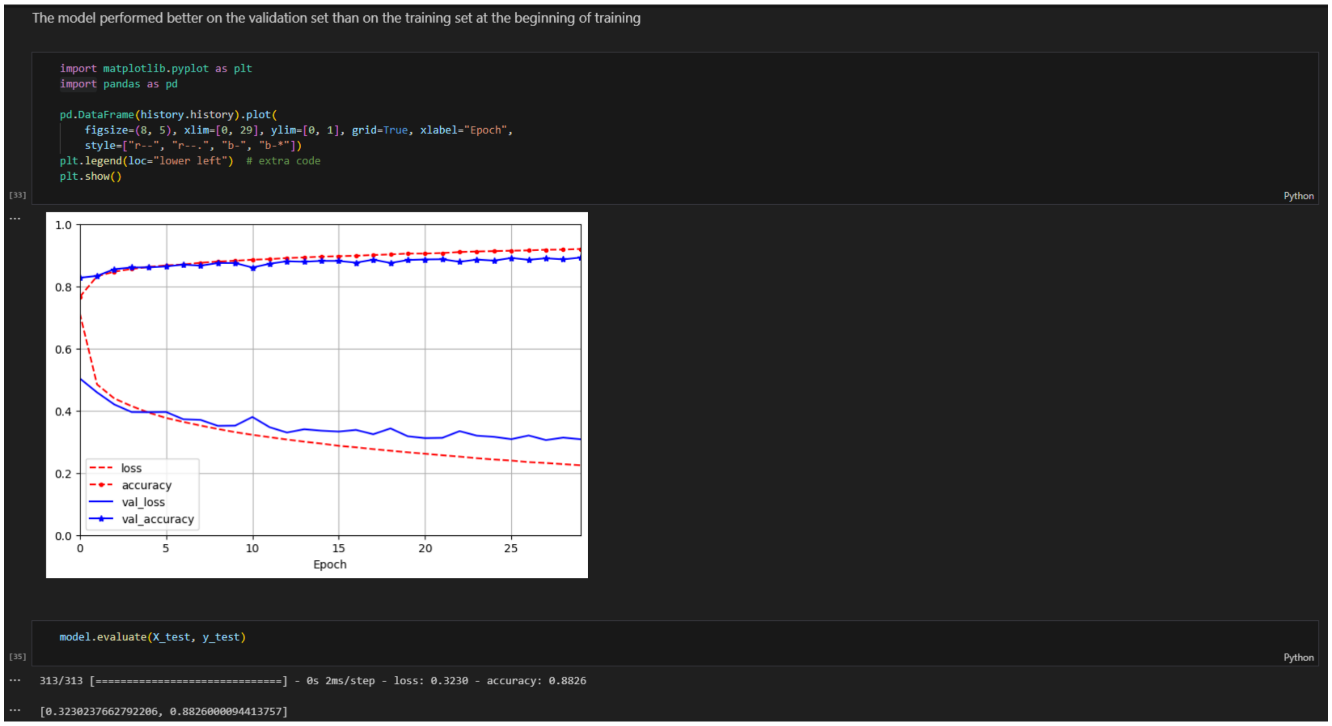


```
[29]
history.params

{'verbose': 1, 'epochs': 30, 'steps': 1719}
```

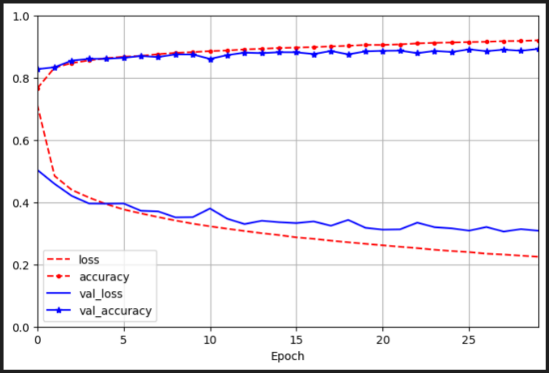
```
[30]
print(history.epoch)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
```


The model performed better on the validation set than on the training set at the beginning of training

```python
import matplotlib.pyplot as plt
import pandas as pd

pd.DataFrame(history.history).plot(
    figsize=(8, 5), xlim=[0, 29], ylim=[0, 1], grid=True, xlabel="Epoch",
    style=["r--", "r--.", "b-", "b-*"])
plt.legend(loc="lower left")  # extra code
plt.show()
```

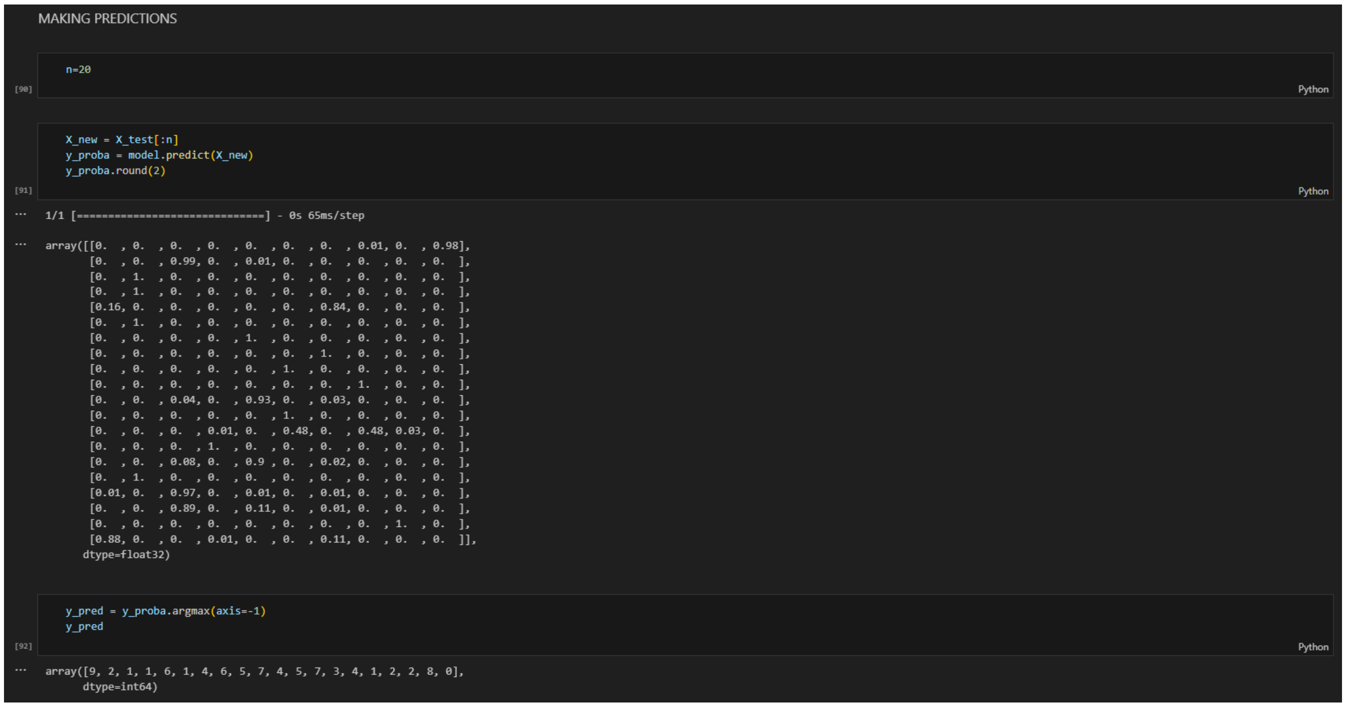


```python
model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 0s 2ms/step - loss: 0.3230 - accuracy: 0.8826

[0.3230237662792206, 0.8826000094413757]
```

ACCURACY: 0.8826000094413757


MAKING PREDICTIONS

```python
n=20
```

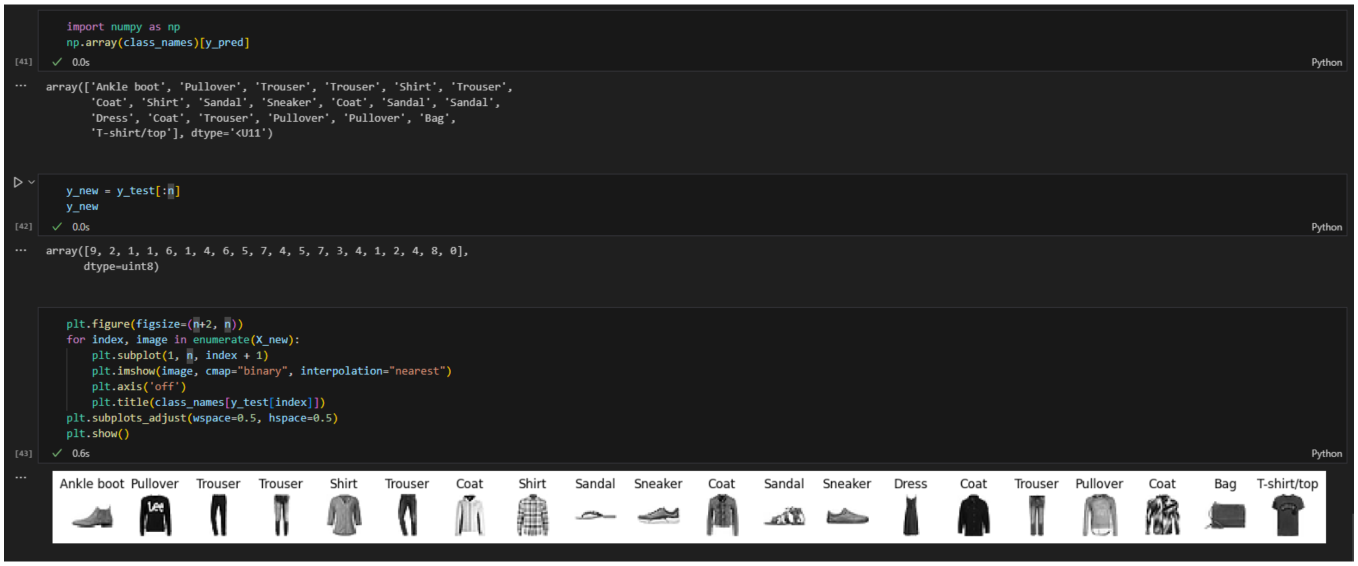
```python
X_new = X_test[:n]
y_proba = model.predict(X_new)
y_proba.round(2)
```

```
1/1 [==============================] - 0s 65ms/step

array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.98],
       [0.  , 0.  , 0.99, 0.  , 0.01, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.16, 0.  , 0.  , 0.  , 0.  , 0.  , 0.84, 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ],
       [0.  , 0.  , 0.04, 0.  , 0.93, 0.  , 0.03, 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , 0.01, 0.  , 0.48, 0.  , 0.48, 0.03, 0.  ],
       [0.  , 0.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.08, 0.  , 0.9 , 0.  , 0.02, 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.01, 0.  , 0.97, 0.  , 0.01, 0.  , 0.01, 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.89, 0.  , 0.11, 0.  , 0.01, 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  , 0.  ],
       [0.88, 0.  , 0.  , 0.01, 0.  , 0.  , 0.11, 0.  , 0.  , 0.  ]],
      dtype=float32)
```

```python
y_pred = y_proba.argmax(axis=-1)
y_pred
```

```
array([9, 2, 1, 1, 6, 1, 4, 6, 5, 7, 4, 5, 7, 3, 4, 1, 2, 2, 8, 0],
      dtype=int64)
```



```
import numpy as np
np.array(class_names)[y_pred]
```

[41] ✓ 0.0s Python

```
array(['Ankle boot', 'Pullover', 'Trouser', 'Trouser', 'Shirt', 'Trouser',
       'Coat', 'Shirt', 'Sandal', 'Sneaker', 'Coat', 'Sandal', 'Sandal',
       'Dress', 'Coat', 'Trouser', 'Pullover', 'Pullover', 'Bag',
       'T-shirt/top'], dtype='<U11')
```

```
y_new = y_test[:n]
y_new
```

[42] ✓ 0.0s Python

```
array([9, 2, 1, 1, 6, 1, 4, 6, 5, 7, 4, 5, 7, 3, 4, 1, 2, 4, 8, 0],
      dtype=uint8)
```

```
plt.figure(figsize=(n+2, n))
for index, image in enumerate(X_new):
    plt.subplot(1, n, index + 1)
    plt.imshow(image, cmap="binary", interpolation="nearest")
    plt.axis('off')
    plt.title(class_names[y_test[index]])
plt.subplots_adjust(wspace=0.5, hspace=0.5)
plt.show()
```

[43] ✓ 0.6s Python

7. Conclusion and Future Scope

In adapting our approach to a Classification Multilayer Perceptron (MLP) for image recognition in the MNIST Fashion Dataset, we have embraced the dynamic nature of machine learning projects. This shift not only showcases our ability to pivot based on project requirements but also reflects the iterative and exploratory nature of the field. As we navigate through the implementation and testing phases, we anticipate unraveling the unique characteristics and capabilities of MLPs in the context of image classification.

Looking forward, the project opens doors to intriguing possibilities. The future scope encompasses a nuanced exploration of hyperparameter tuning to optimize the MLP's performance. Additionally, delving into ensemble methods could further enhance the model's robustness and accuracy. An exciting avenue for future research involves investigating transfer learning techniques, potentially leveraging pre-trained models to extract features that are transferable to our fashion image classification task.

Beyond the technical aspects, the user interface and deployment of the model remain key considerations for the project's future evolution. Developing an intuitive interface and deploying the model as a user-friendly application would bridge the gap between the sophisticated machine learning backend and practical, real-world usability. This convergence of technical refinement and user-centric deployment marks the holistic progression of our project, embodying the spirit of innovation and adaptability that defines the dynamic landscape of machine learning. As we move forward, we remain committed to not only mastering the intricacies of classification MLPs but also contributing to the practical advancements in the ever-evolving field of artificial intelligence and data science.