```
pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.17.1-py3-none-any.whl (536 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 536.7/536.7 kB 7.2 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets) (3.13.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.25.2)
Requirement already satisfied: pyarrow>=12.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (14.0.2)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/dist-packages (from datasets) (0.6)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 10.7 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.2)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 12.0 MB/s eta 0:00:00
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2023.6
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.9.3)
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.20.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.19.4->
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (2.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (2024
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2023.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->datasets)
Installing collected packages: dill, multiprocess, datasets
Successfully installed datasets-2.17.1 dill-0.3.8 multiprocess-0.70.16
```

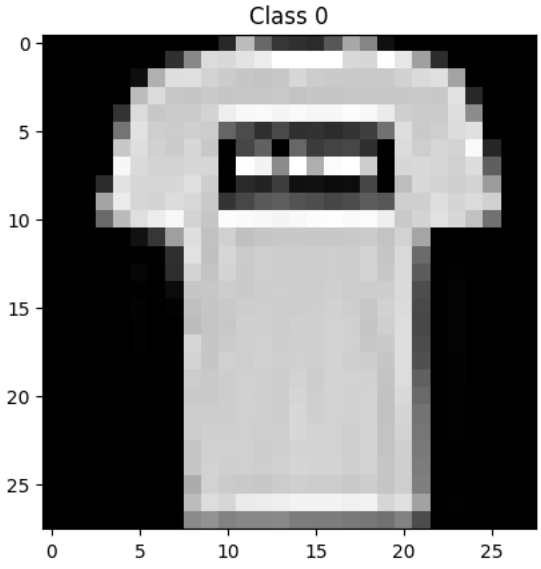
```
from datasets import load_dataset

dataset = load_dataset("fashion_mnist")
```

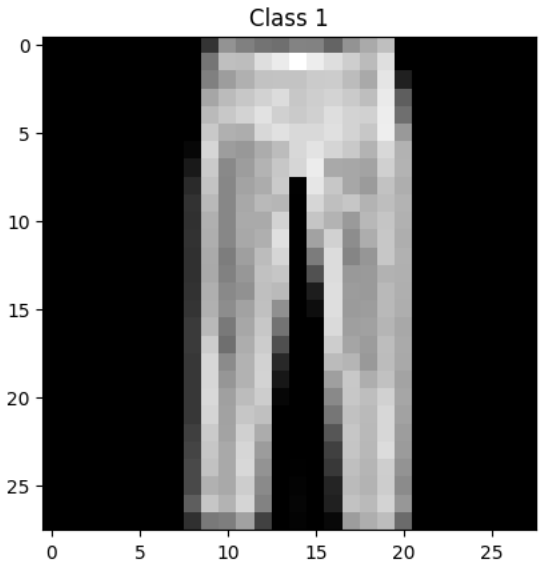
```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarni
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mo
  warnings.warn(
Downloading data: 100%|| 30.9M/30.9M [00:01<00:00, 28.2MB/s]
Downloading data: 100%|| 5.18M/5.18M [00:00<00:00, 5.92MB/s]
Generating train split: 100%|| 60000/60000 [00:00<00:00, 140649.86 examples/s]
Generating test split: 100%|| 10000/10000 [00:00<00:00, 113050.36 examples/s]
```

```
dataset
```

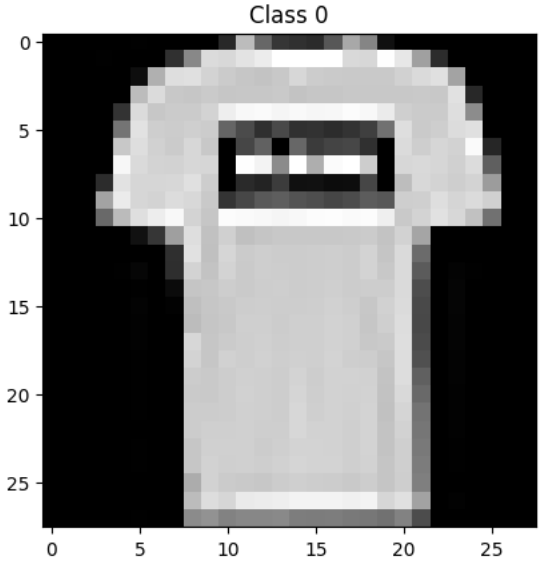
```
DatasetDict({
    train: Dataset({
        features: ['image', 'label'],
        num_rows: 60000
    })
    test: Dataset({
        features: ['image', 'label'],
        num_rows: 10000
    })
})
```

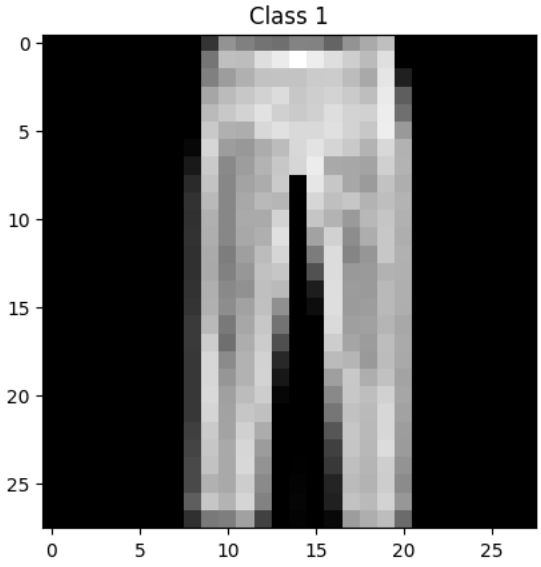
```
from datasets import load_dataset
import matplotlib.pyplot as plt


fashion_mnist = load_dataset('fashion_mnist')

train_data = fashion_mnist['train']

for i in range(10):
    class_images = train_data.filter(lambda x: x['label'] == i)['image']
    plt.imshow(class_images[0], cmap='gray')
    plt.title(f"Class {i}")
    plt.show()
```





```
from datasets import load_dataset
import matplotlib.pyplot as plt

fashion_mnist = load_dataset('fashion_mnist')

train_data = fashion_mnist['train']
for i in range(10):
    class_images = []
    for data in train_data:
        if data['label'] == i:
            class_images.append(data['image'])
            break

    plt.imshow(class_images[0], cmap='gray')
    plt.title(f"Class {i}")
    plt.show()
```





```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train = X_train.reshape((-1, 28, 28, 1)) / 255.0
X_test = X_test.reshape((-1, 28, 28, 1)) / 255.0
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(X_train, y_train, batch_size=64, epochs=10, validation_split=0.1)
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)

# Print probability distribution for the first test sample
sample_index = 0
sample_image = X_test[sample_index:sample_index+1]
predicted_probabilities = model.predict(sample_image)
print("Probability distribution for the first test sample:")
print(predicted_probabilities)
```

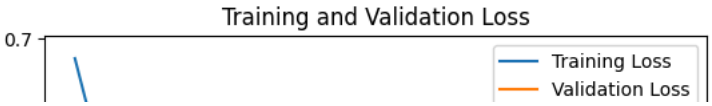
```
Epoch 1/10
844/844 [==============================] - 53s 61ms/step - loss: 0.7238 - accuracy: 0.7424 - val_loss: 0.4486 - val_accuracy: 0.8332
Epoch 2/10
844/844 [==============================] - 51s 61ms/step - loss: 0.4687 - accuracy: 0.8325 - val_loss: 0.3969 - val_accuracy: 0.8543
Epoch 3/10
844/844 [==============================] - 60s 71ms/step - loss: 0.4058 - accuracy: 0.8551 - val_loss: 0.3528 - val_accuracy: 0.8638
Epoch 4/10
844/844 [==============================] - 49s 57ms/step - loss: 0.3630 - accuracy: 0.8693 - val_loss: 0.3255 - val_accuracy: 0.8793
Epoch 5/10
844/844 [==============================] - 49s 59ms/step - loss: 0.3297 - accuracy: 0.8808 - val_loss: 0.3069 - val_accuracy: 0.8858
Epoch 6/10
844/844 [==============================] - 47s 56ms/step - loss: 0.3071 - accuracy: 0.8889 - val_loss: 0.2931 - val_accuracy: 0.8926
Epoch 7/10
844/844 [==============================] - 51s 60ms/step - loss: 0.2850 - accuracy: 0.8978 - val_loss: 0.2992 - val_accuracy: 0.8915
Epoch 8/10
844/844 [==============================] - 46s 54ms/step - loss: 0.2683 - accuracy: 0.9029 - val_loss: 0.2989 - val_accuracy: 0.8893
Epoch 9/10
844/844 [==============================] - 47s 56ms/step - loss: 0.2563 - accuracy: 0.9059 - val_loss: 0.2872 - val_accuracy: 0.8932
Epoch 10/10
844/844 [==============================] - 46s 55ms/step - loss: 0.2397 - accuracy: 0.9122 - val_loss: 0.2904 - val_accuracy: 0.9006
313/313 [==============================] - 3s 10ms/step - loss: 0.3167 - accuracy: 0.8932
Test Accuracy: 0.8931999802589417
1/1 [==============================] - 0s 212ms/step
Probability distribution for the first test sample:
[[2.8046669e-09 6.6003902e-10 1.9737418e-11 3.4668884e-10 2.9956873e-10
  4.1178908e-05 5.1295790e-09 6.9167191e-04 2.7491978e-09 9.9926716e-01]]
```

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.metrics import categorical_crossentropy
from keras.optimizers import SGD


def Alexnet():
  model = Sequential()

  model.add(Conv2D(filters = 96, kernel_size = (11,11), strides= 4, padding = 'valid', activation='relu', input_shape = (227,227,3)))
  model.add(MaxPooling2D(pool_size = (3,3), strides = 2))

  model.add(Conv2D(filters = 256, kernel_size = (5,5), padding = 'same', activation = 'relu'))
  model.add(MaxPooling2D(pool_size = (3,3), strides = 2))

  model.add(Conv2D(filters = 384, kernel_size = (3,3), padding = 'same', activation = 'relu'))
  model.add(Conv2D(filters = 384, kernel_size = (3,3), padding = 'same', activation = 'relu'))
  model.add(Conv2D(filters = 256, kernel_size = (3,3), padding = 'same', activation = 'relu'))
  model.add(MaxPooling2D(pool_size = (3,3), strides = 2))

  model.add(Flatten())
  model.add(Dense(4096, activation = 'relu'))
  model.add(Dropout(0.5))
  model.add(Dense(4096, activation = 'relu'))
  model.add(Dropout(0.5))
  model.add(Dense(1000, activation = 'softmax'))

  # compile the model with a loss funciton, a metric and and optimizer method for estimating the loss function
  opt = SGD(lr = 0.1)
  model.compile(loss = categorical_crossentropy,
                optimizer = opt,
                metrics = ['accuracy'])

  return model


Alexnet_model = Alexnet()


Alexnet_model.summary()
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 55, 55, 96)        34944

 max_pooling2d_4 (MaxPoolin  (None, 27, 27, 96)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 27, 27, 256)       614656

 max_pooling2d_5 (MaxPoolin  (None, 13, 13, 256)       0
 g2D)

 conv2d_6 (Conv2D)           (None, 13, 13, 384)       885120

 conv2d_7 (Conv2D)           (None, 13, 13, 384)       1327488

 conv2d_8 (Conv2D)           (None, 13, 13, 256)       884992

 max_pooling2d_6 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 4096)              37752832

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 1000)              4097000

=================================================================
Total params: 62378344 (237.95 MB)
Trainable params: 62378344 (237.95 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

def create_model(num_hidden_layers, num_neurons):
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    for _ in range(num_hidden_layers):
        model.add(Dense(num_neurons, activation='relu'))
        model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))
    return model

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train = X_train.reshape((-1, 28, 28, 1)) / 255.0
X_test = X_test.reshape((-1, 28, 28, 1)) / 255.0
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
num_hidden_layers = 2
num_neurons = 128
model = create_model(num_hidden_layers, num_neurons)
model.compile(optimizer='nadam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=64, epochs=10, validation_split=0.1)
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)
sample_index = 0
sample_image = X_test[sample_index:sample_index+1]
predicted_probabilities = model.predict(sample_image)
print("Probability distribution for the first test sample:")
print(predicted_probabilities)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
844/844 [==============================] - 34s 37ms/step - loss: 0.6804 - accuracy: 0.7571 - val_loss: 0.3591 - val_accuracy: 0.8
Epoch 2/10
844/844 [==============================] - 30s 35ms/step - loss: 0.4273 - accuracy: 0.8551 - val_loss: 0.3200 - val_accuracy: 0.8
Epoch 3/10
844/844 [==============================] - 29s 34ms/step - loss: 0.3686 - accuracy: 0.8724 - val_loss: 0.2941 - val_accuracy: 0.8
Epoch 4/10
844/844 [==============================] - 29s 34ms/step - loss: 0.3385 - accuracy: 0.8832 - val_loss: 0.2819 - val_accuracy: 0.8
Epoch 5/10
844/844 [==============================] - 29s 35ms/step - loss: 0.3151 - accuracy: 0.8909 - val_loss: 0.2692 - val_accuracy: 0.9
Epoch 6/10
844/844 [==============================] - 29s 35ms/step - loss: 0.2971 - accuracy: 0.8956 - val_loss: 0.2656 - val_accuracy: 0.9
Epoch 7/10
844/844 [==============================] - 29s 35ms/step - loss: 0.2840 - accuracy: 0.9008 - val_loss: 0.2547 - val_accuracy: 0.9
Epoch 8/10
844/844 [==============================] - 29s 34ms/step - loss: 0.2700 - accuracy: 0.9047 - val_loss: 0.2489 - val_accuracy: 0.9
Epoch 9/10
844/844 [==============================] - 29s 34ms/step - loss: 0.2573 - accuracy: 0.9084 - val_loss: 0.2433 - val_accuracy: 0.9
Epoch 10/10
844/844 [==============================] - 29s 34ms/step - loss: 0.2459 - accuracy: 0.9118 - val_loss: 0.2461 - val_accuracy: 0.9
313/313 [==============================] - 2s 5ms/step - loss: 0.2572 - accuracy: 0.9101
Test Accuracy: 0.910099983215332
1/1 [==============================] - 0s 105ms/step
Probability distribution for the first test sample:
[[6.27489283e-10 1.24231625e-09 7.52468435e-11 1.06337411e-12
  9.85102139e-11 1.11867244e-04 1.27268696e-09 1.99702522e-03
  9.49850900e-08 9.97891009e-01]]
```



```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam, RMSprop, Nadam
from tensorflow.keras.initializers import RandomNormal, GlorotNormal
from tensorflow.keras.datasets import fashion_mnist
from sklearn.model_selection import train_test_split

# Load and preprocess Fashion-MNIST dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train = X_train.reshape((-1, 28 * 28)) / 255.0
X_test = X_test.reshape((-1, 28 * 28)) / 255.0
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

# Define neural network class
class NeuralNetwork:
    def __init__(self, input_shape, num_classes, optimizer='sgd', learning_rate=0.01, batch_size=32, epochs=10,
                 hidden_layers=3, hidden_units=64, weight_decay=0.0, weight_init='random', activation='relu'):
        self.input_shape = input_shape
        self.num_classes = num_classes
        self.optimizer = optimizer
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.epochs = epochs
        self.hidden_layers = hidden_layers
        self.hidden_units = hidden_units
        self.weight_decay = weight_decay
        self.weight_init = weight_init
        self.activation = activation
        self.model = self.build_model()

    def build_model(self):
        model = Sequential()
        if self.weight_init == 'random':
            initializer = RandomNormal()
        elif self.weight_init == 'xavier':
            initializer = GlorotNormal()
        else:
            raise ValueError("Invalid weight initialization method")

        # Add input layer
        model.add(Dense(self.hidden_units, activation=self.activation, input_shape=self.input_shape,
                        kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2(self.weight_decay)))

        # Add hidden layers
        for _ in range(self.hidden_layers - 1):
            model.add(Dense(self.hidden_units, activation=self.activation, kernel_initializer=initializer,
                            kernel_regularizer=tf.keras.regularizers.l2(self.weight_decay)))

        # Add output layer
        model.add(Dense(self.num_classes, activation='softmax'))

        # Compile model
        if self.optimizer == 'sgd':
            optimizer = SGD(learning_rate=self.learning_rate)
        elif self.optimizer == 'adam':
            optimizer = Adam(learning_rate=self.learning_rate)
        elif self.optimizer == 'rmsprop':
            optimizer = RMSprop(learning_rate=self.learning_rate)
        elif self.optimizer == 'nadam':
            optimizer = Nadam(learning_rate=self.learning_rate)
        else:
            raise ValueError("Invalid optimizer")

        model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
        return model

    def train(self, X_train, y_train, validation_split=0.1):
        self.model.fit(X_train, y_train, batch_size=self.batch_size, epochs=self.epochs, validation_split=validation_split)

# Ask the user for hyperparameters
optimizer = input("Enter optimizer (sgd, adam, rmsprop, nadam): ")
learning_rate = float(input("Enter learning rate (e.g., 0.01): "))
batch_size = int(input("Enter batch size (e.g., 32): "))
hidden_layers = int(input("Enter number of hidden layers (e.g., 3): "))
hidden_units = int(input("Enter number of hidden units (e.g., 64): "))
weight_decay = float(input("Enter weight decay (e.g., 0.0): "))
weight_init = input("Enter weight initialization method (random, xavier): ")
activation = input("Enter activation function (sigmoid, tanh, relu): ")
```

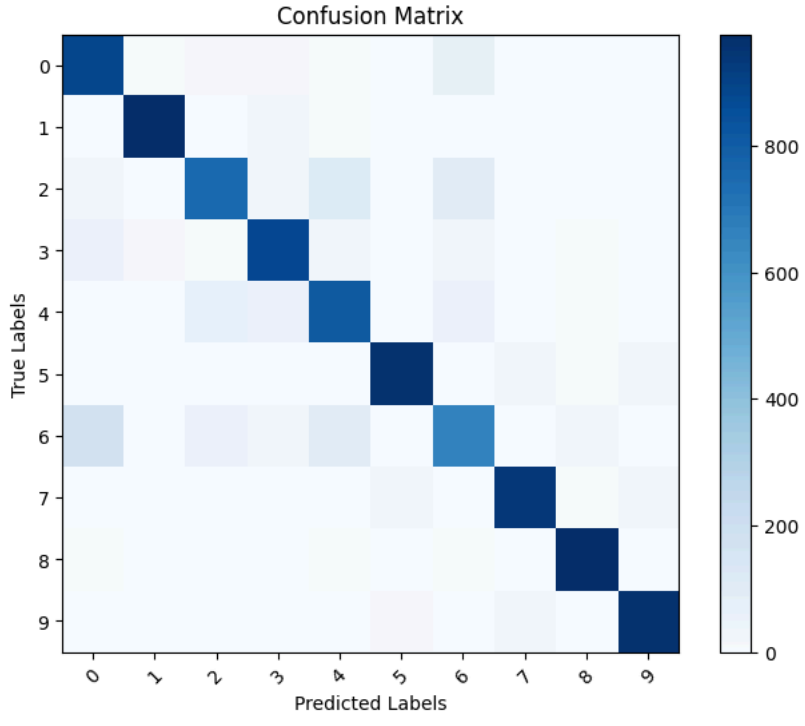
```
# Create and train neural network with user-specified hyperparameters
nn = NeuralNetwork(input_shape=(784,), num_classes=10, optimizer=optimizer,
                   learning_rate=learning_rate, batch_size=batch_size,
                   hidden_layers=hidden_layers, hidden_units=hidden_units,
                   weight_decay=weight_decay, weight_init=weight_init,
                   activation=activation)
nn.train(X_train, y_train)

# Evaluate on test data
test_loss, test_accuracy = nn.model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

```
Enter optimizer (sgd, adam, rmsprop, nadam): adam
Enter learning rate (e.g., 0.01): 0.1
Enter batch size (e.g., 32): 16
Enter number of hidden layers (e.g., 3): 3
Enter number of hidden units (e.g., 64): 32
Enter weight decay (e.g., 0.0): 0
Enter weight initialization method (random, xavier): random
Enter activation function (sigmoid, tanh, relu): relu
Epoch 1/10
3375/3375 [==============================] - 9s 2ms/step - loss: 2.3247 - accuracy: 0.1003 - val_loss: 2.3371 - val_accuracy: 0.1027
Epoch 2/10
3375/3375 [==============================] - 8s 3ms/step - loss: 2.3206 - accuracy: 0.0984 - val_loss: 2.3182 - val_accuracy: 0.0973
Epoch 3/10
3375/3375 [==============================] - 9s 3ms/step - loss: 2.3218 - accuracy: 0.0993 - val_loss: 2.3107 - val_accuracy: 0.0985
Epoch 4/10
3375/3375 [==============================] - 7s 2ms/step - loss: 2.3211 - accuracy: 0.0990 - val_loss: 2.3108 - val_accuracy: 0.1003
Epoch 5/10
3375/3375 [==============================] - 9s 3ms/step - loss: 2.3213 - accuracy: 0.0980 - val_loss: 2.3215 - val_accuracy: 0.1008
Epoch 6/10
3375/3375 [==============================] - 8s 2ms/step - loss: 2.3198 - accuracy: 0.1008 - val_loss: 2.3338 - val_accuracy: 0.0985
Epoch 7/10
3375/3375 [==============================] - 9s 3ms/step - loss: 2.3210 - accuracy: 0.0994 - val_loss: 2.3339 - val_accuracy: 0.1027
Epoch 8/10
3375/3375 [==============================] - 8s 2ms/step - loss: 2.3210 - accuracy: 0.1009 - val_loss: 2.3152 - val_accuracy: 0.1027
Epoch 9/10
3375/3375 [==============================] - 7s 2ms/step - loss: 2.3213 - accuracy: 0.1009 - val_loss: 2.3138 - val_accuracy: 0.0985
Epoch 10/10
3375/3375 [==============================] - 9s 3ms/step - loss: 2.3205 - accuracy: 0.0998 - val_loss: 2.3190 - val_accuracy: 0.0973
313/313 [==============================] - 1s 2ms/step - loss: 2.3195 - accuracy: 0.1000
Test Accuracy: 0.10000000149011612
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam, RMSprop, Nadam
from tensorflow.keras.initializers import RandomNormal, GlorotNormal
from tensorflow.keras.datasets import fashion_mnist
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# Load and preprocess Fashion-MNIST dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train = X_train.reshape((-1, 28 * 28)) / 255.0
X_test = X_test.reshape((-1, 28 * 28)) / 255.0
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

# Define neural network class
class Model:
    def __init__(self, input_shape, num_classes, optimizer='sgd', learning_rate=0.01, batch_size=32, epochs=10,
                 hidden_layers=3, hidden_units=64, weight_decay=0.0, weight_init='random', activation='relu'):
        self.input_shape = input_shape
        self.num_classes = num_classes
        self.optimizer = optimizer
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.epochs = epochs
        self.hidden_layers = hidden_layers
        self.hidden_units = hidden_units
        self.weight_decay = weight_decay
        self.weight_init = weight_init
        self.activation = activation
        self.model = self.build_model()

    def build_model(self):
        model = Sequential()
        if self.weight_init == 'random':
            initializer = RandomNormal()
        elif self.weight_init == 'xavier':
            initializer = GlorotNormal()
        else:
            raise ValueError("Invalid weight initialization method")

        # Add input layer
        model.add(Dense(self.hidden_units, activation=self.activation, input_shape=self.input_shape,
                        kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2(self.weight_decay)))

        # Add hidden layers
        for _ in range(self.hidden_layers - 1):
            model.add(Dense(self.hidden_units, activation=self.activation, kernel_initializer=initializer,
                            kernel_regularizer=tf.keras.regularizers.l2(self.weight_decay)))

        # Add output layer
        model.add(Dense(self.num_classes, activation='softmax'))

        # Compile model
        if self.optimizer == 'sgd':
            optimizer = SGD(learning_rate=self.learning_rate)
        elif self.optimizer == 'adam':
            optimizer = Adam(learning_rate=self.learning_rate)
        elif self.optimizer == 'rmsprop':
            optimizer = RMSprop(learning_rate=self.learning_rate)
        elif self.optimizer == 'nadam':
            optimizer = Nadam(learning_rate=self.learning_rate)
        else:
            raise ValueError("Invalid optimizer")

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

        return model

    def train(self, X_train, y_train):
        history = self.model.fit(X_train, y_train, batch_size=self.batch_size,
                                 epochs=self.epochs)

# Instantiate the Model class with hyperparameters and train the model
my_model = Model(input_shape=(784,), num_classes=10, optimizer='adam', learning_rate=0.001,
                 batch_size=32, hidden_layers=3, hidden_units=64,
                 weight_decay=0.0, weight_init='random', activation='relu')
my_model.train(X_train, y_train)
```

```
# Evaluate on test data and print test accuracy
test_loss, test_accuracy = my_model.model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)

# Make predictions on the test set
y_pred = my_model.model.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)

# Calculate the confusion matrix
cm = confusion_matrix(np.argmax(y_test, axis=1), y_pred_labels)

# Display the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(np.unique(np.argmax(y_test, axis=1))))
plt.xticks(tick_marks, np.unique(np.argmax(y_test, axis=1)), rotation=45)
plt.yticks(tick_marks, np.unique(np.argmax(y_test, axis=1)))
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/initializers/initializers.py:120: UserWarning: The initializer RandomNormal is
  warnings.warn(
Epoch 1/10
1875/1875 [==============================] - 8s 3ms/step - loss: 0.5426 - accuracy: 0.8030
Epoch 2/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3849 - accuracy: 0.8591
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3477 - accuracy: 0.8719
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3242 - accuracy: 0.8805
Epoch 5/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.3076 - accuracy: 0.8860
Epoch 6/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2921 - accuracy: 0.8920
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2811 - accuracy: 0.8948
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2726 - accuracy: 0.8974
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2624 - accuracy: 0.9018
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2556 - accuracy: 0.9033
313/313 [==============================] - 1s 1ms/step - loss: 0.3353 - accuracy: 0.8803
Test Accuracy: 0.880299985408783
313/313 [==============================] - 1s 1ms/step
```



```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam, RMSprop, Nadam
from tensorflow.keras.initializers import RandomNormal, GlorotNormal
from tensorflow.keras.datasets import fashion_mnist
from sklearn.metrics import accuracy_score, mean_squared_error

# Load and preprocess Fashion-MNIST dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train = X_train.reshape((-1, 28 * 28)) / 255.0
X_test = X_test.reshape((-1, 28 * 28)) / 255.0
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

# Define neural network class
class NeuralNetwork:
    def __init__(self, input_shape, num_classes, optimizer='sgd', learning_rate=0.01, batch_size=32, epochs=10,
                 hidden_layers=3, hidden_units=64, weight_decay=0.0, weight_init='random', activation='relu', loss='categorical_crossent
        self.input_shape = input_shape
        self.num_classes = num_classes
        self.optimizer = optimizer
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.epochs = epochs
        self.hidden_layers = hidden_layers
        self.hidden_units = hidden_units
        self.weight_decay = weight_decay
        self.weight_init = weight_init
        self.activation = activation
        self.loss = loss
        self.model = self.build_model()

    def build_model(self):
        model = Sequential()
        if self.weight_init == 'random':
            initializer = RandomNormal()
        elif self.weight_init == 'xavier':
            initializer = GlorotNormal()
        else:
            raise ValueError("Invalid weight initialization method")

        # Add input layer
        model.add(Dense(self.hidden_units, activation=self.activation, input_shape=self.input_shape,
                        kernel_initializer=initializer, kernel_regularizer=tf.keras.regularizers.l2(self.weight_decay)))

        # Add hidden layers
        for _ in range(self.hidden_layers - 1):
            model.add(Dense(self.hidden_units, activation=self.activation, kernel_initializer=initializer,
                            kernel_regularizer=tf.keras.regularizers.l2(self.weight_decay)))

        # Add output layer
        model.add(Dense(self.num_classes, activation='softmax'))

        # Compile model
        if self.optimizer == 'sgd':
            optimizer = SGD(learning_rate=self.learning_rate)
        elif self.optimizer == 'adam':
            optimizer = Adam(learning_rate=self.learning_rate)
        elif self.optimizer == 'rmsprop':
            optimizer = RMSprop(learning_rate=self.learning_rate)
        elif self.optimizer == 'nadam':
            optimizer = Nadam(learning_rate=self.learning_rate)
        else:
            raise ValueError("Invalid optimizer")

        model.compile(optimizer=optimizer, loss=self.loss, metrics=['accuracy'])
        return model

    def train(self, X_train, y_train, validation_data):
        self.model.fit(X_train, y_train, batch_size=self.batch_size, epochs=self.epochs, validation_data=validation_data)

# Train neural network with cross-entropy loss
nn_cross_entropy = NeuralNetwork(input_shape=(784,), num_classes=10, loss='categorical_crossentropy')
nn_cross_entropy.train(X_train, y_train, validation_data=(X_test, y_test))

# Train neural network with squared error loss
nn_squared_error = NeuralNetwork(input_shape=(784,), num_classes=10, loss='mean_squared_error')
nn_squared_error.train(X_train, y_train, validation_data=(X_test, y_test))

# Evaluate models on test data
```

```
test_loss_ce, test_accuracy_ce = nn_cross_entropy.model.evaluate(X_test, y_test)
test_loss_se, test_accuracy_se = nn_squared_error.model.evaluate(X_test, y_test)

print("Cross-Entropy Loss - Test Accuracy:", test_accuracy_ce)
print("Squared Error Loss - Test Accuracy:", test_accuracy_se)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/initializers/initializers.py:120: UserWarning: The initializer RandomNormal is uns
  warnings.warn(
Epoch 1/10
1875/1875 [==============================] - 19s 9ms/step - loss: 1.1813 - accuracy: 0.5836 - val_loss: 0.7184 - val_accuracy: 0.734
Epoch 2/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.6056 - accuracy: 0.7839 - val_loss: 0.6213 - val_accuracy: 0.7733
Epoch 3/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.5158 - accuracy: 0.8173 - val_loss: 0.5421 - val_accuracy: 0.8034
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.4689 - accuracy: 0.8335 - val_loss: 0.5060 - val_accuracy: 0.8204
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4391 - accuracy: 0.8446 - val_loss: 0.4654 - val_accuracy: 0.8381
Epoch 6/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.4157 - accuracy: 0.8515 - val_loss: 0.4395 - val_accuracy: 0.8433
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3974 - accuracy: 0.8583 - val_loss: 0.4331 - val_accuracy: 0.8459
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.3834 - accuracy: 0.8629 - val_loss: 0.4119 - val_accuracy: 0.8556
Epoch 9/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.3721 - accuracy: 0.8670 - val_loss: 0.4030 - val_accuracy: 0.8571
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3614 - accuracy: 0.8699 - val_loss: 0.3910 - val_accuracy: 0.8644
Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.0899 - accuracy: 0.1152 - val_loss: 0.0898 - val_accuracy: 0.1207
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0896 - accuracy: 0.1292 - val_loss: 0.0895 - val_accuracy: 0.1479
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0893 - accuracy: 0.1760 - val_loss: 0.0892 - val_accuracy: 0.2137
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0889 - accuracy: 0.2524 - val_loss: 0.0887 - val_accuracy: 0.2664
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0883 - accuracy: 0.2927 - val_loss: 0.0880 - val_accuracy: 0.2914
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0875 - accuracy: 0.3027 - val_loss: 0.0870 - val_accuracy: 0.2921
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0863 - accuracy: 0.2948 - val_loss: 0.0855 - val_accuracy: 0.2842
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0843 - accuracy: 0.2817 - val_loss: 0.0831 - val_accuracy: 0.2777
Epoch 9/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0813 - accuracy: 0.2862 - val_loss: 0.0795 - val_accuracy: 0.3206
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0768 - accuracy: 0.4369 - val_loss: 0.0741 - val_accuracy: 0.4760
313/313 [==============================] - 1s 2ms/step - loss: 0.3910 - accuracy: 0.8644
313/313 [==============================] - 1s 2ms/step - loss: 0.0741 - accuracy: 0.4760
Cross-Entropy Loss - Test Accuracy: 0.8644000291824341
Squared Error Loss - Test Accuracy: 0.47600001096725464
```

```
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from keras.models import Model

input_shape = (224, 224, 3)

# Define the model
input_layer = Input(shape=input_shape)

# 1st Convolutional Layer
x = Conv2D(filters=96, kernel_size=(11, 11), strides=(4, 4), padding='valid', activation='relu')(input_layer)
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='valid')(x)

# 2nd Convolutional Layer
```