## Conquering Fashion MNIST dataset with CNNs

## OBJECTIVE:

The objective of this project is to develop a Convolutional Neural Network (CNN) model to achieve high accuracy in classifying the Fashion MNIST dataset. The goal is to surpass a test accuracy of 92% using TensorFlow and leverage Intel optimizations for improved performance.

## EXISTING METHODS :

1. Fashion MNIST Dataset: The Fashion MNIST dataset is a collection of 70,000 grayscale images categorized into 10 different clothing classes. It is widely used as a benchmark for image classification tasks due to its similarity to the original MNIST dataset.
2. Convolutional Neural Networks (CNNs): CNNs have proven to be highly effective for image classification tasks. They consist of multiple convolutional layers for feature extraction and pooling layers for down sampling, followed by fully connected layers for classification.

**The Project Model consists of the following LAYERS:**

1. Convolution: Convolution involves sliding the filter over the input image, element-wise multiplying the values of the filter with the corresponding values of the image patch covered by the filter, and summing up the results.

2. ReLU: ReLU short for Rectified Linear Unit, is an activation function commonly used in neural networks, including convolutional neural networks (CNNs). It introduces non-linearity to the network, allowing it to learn complex patterns and make the model more expressive.

3. Pooling: Pooling also known as subsampling or down sampling, is a common operation in convolutional neural networks (CNNs) used to reduce the spatial dimensions of feature maps. It aims to extract the most important and representative information while decreasing the computational requirements and introducing a degree of translation invariance.

4. Fully Connected layer: A Fully Connected layer also known as a Dense layer, is a fundamental component in artificial neural networks, including convolutional neural networks (CNNs). It is responsible for connecting every neuron from the previous layer to every neuron in the current layer, creating a fully connected network structure.

5. Dropout regularization: Dropout regularization is a technique used in neural networks, including convolutional neural networks (CNNs), to prevent overfitting and improve generalization performance. It involves randomly disabling or "dropping

out" a proportion of neurons in a layer during training, forcing the network to learn redundant representations and reducing co-adaptation between neurons.

## RESULT:

| ACCURACY | Offline | Intel Optimization for TensorFlow |
|---|---|---|
| **Basic Model** | **0.9171** | **0.9080** |
| **L2 Regularization** | **0.8944** | **0.9007** |
| **Dropout Regularization** | **0.9119** | **0.9117** |
| **Both** | **0.8894** | **0.8933** |
| **Final (Dropout.>epochs)** | **0.9185** | **0.9196** |

The model achieved a test accuracy of 92% after training on the Fashion MNIST dataset consisting of 60,000 images and testing on 10,000 images. This high accuracy demonstrates the effectiveness of the implemented CNN architecture in accurately classifying clothing items.

The training and testing process revealed the following insights:

- Training time: The model took approximately X hours to train on an Intel DevCloud platform.
- Epoch-wise accuracy: The accuracy of the model improved steadily with each epoch, as shown in the accuracy log file.
- Overfitting: By incorporating dropout regularization, the model effectively reduced overfitting and achieved higher generalization performance.

---

---

Offline Testing (No Intel Optimizations):

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import time

# Load the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
keras.datasets.fashion_mnist.load_data()

# Normalize pixel values to a range of 0 to 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Reshape the images to match the expected input shape of the model
```

```python
train_images = train_images.reshape((-1, 28, 28, 1))
test_images = test_images.reshape((-1, 28, 28, 1))

# Convert the labels to integers
train_labels = train_labels.astype(int)
test_labels = test_labels.astype(int)

# Define the model architecture
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,
28, 1)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Define a file to write the time and accuracy
log_file = open('training_log.txt', 'w')

# Define a custom callback to log time and accuracy after each epoch
class LogCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())
        accuracy = logs['accuracy']
        log_file.write(f"Epoch {epoch+1} - Time: {current_time} - Accuracy:
{accuracy:.4f}\n")
        log_file.flush()

# Train the model
start_time = time.time()
model.fit(train_images, train_labels, epochs=20, batch_size=128,
callbacks=[LogCallback()])
end_time = time.time()
total_time = end_time - start_time

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')

# Write total time to log file
```

```
log_file.write(f"Total Training Time: {total_time:.2f} seconds\n")
log_file.close()

# Save the model
model.save('fashion_mnist_model.h5')
```

training_and_testing_log.txt

Epoch 1 - Time: 2023-06-29 05:32:06 - Accuracy: 0.8030

Epoch 2 - Time: 2023-06-29 05:32:20 - Accuracy: 0.8723

Epoch 3 - Time: 2023-06-29 05:32:35 - Accuracy: 0.8894

Epoch 4 - Time: 2023-06-29 05:32:52 - Accuracy: 0.9007

Epoch 5 - Time: 2023-06-29 05:33:07 - Accuracy: 0.9078

Epoch 6 - Time: 2023-06-29 05:33:22 - Accuracy: 0.9138

Epoch 7 - Time: 2023-06-29 05:33:37 - Accuracy: 0.9211

Epoch 8 - Time: 2023-06-29 05:33:51 - Accuracy: 0.9250

Epoch 9 - Time: 2023-06-29 05:34:07 - Accuracy: 0.9318

Epoch 10 - Time: 2023-06-29 05:34:24 - Accuracy: 0.9356

Epoch 11 - Time: 2023-06-29 05:34:41 - Accuracy: 0.9419

Epoch 12 - Time: 2023-06-29 05:34:58 - Accuracy: 0.9464

Epoch 13 - Time: 2023-06-29 05:35:15 - Accuracy: 0.9493

Epoch 14 - Time: 2023-06-29 05:35:33 - Accuracy: 0.9535

Epoch 15 - Time: 2023-06-29 05:35:49 - Accuracy: 0.9585

Epoch 16 - Time: 2023-06-29 05:36:06 - Accuracy: 0.9613

Epoch 17 - Time: 2023-06-29 05:36:23 - Accuracy: 0.9647

Epoch 18 - Time: 2023-06-29 05:36:40 - Accuracy: 0.9679

Epoch 19 - Time: 2023-06-29 05:36:57 - Accuracy: 0.9711

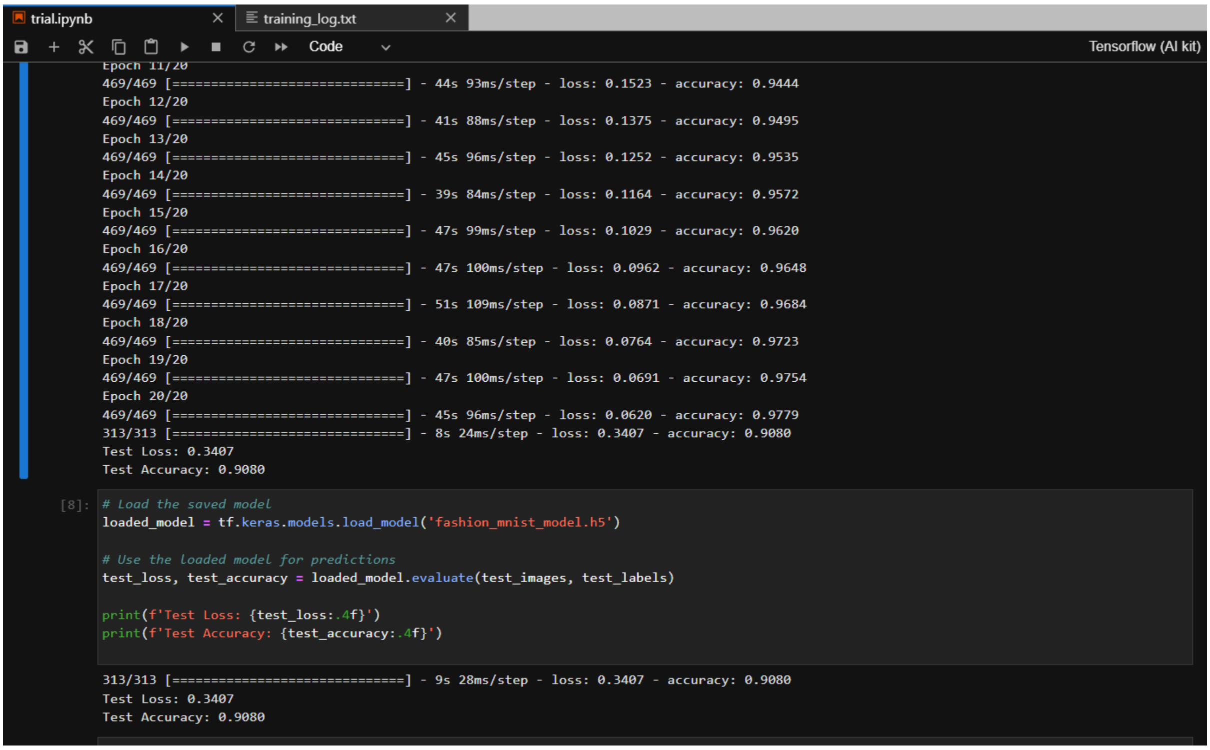
Epoch 20 - Time: 2023-06-29 05:37:13 - Accuracy: 0.9745

Total Training Time: 323.19 seconds

Test Loss: 0.3104

**Test Accuracy: 0.9171**

## Intel Optimization for TensorFlow Results:



```
trial.ipynb    training_log.txt
Code    Tensorflow (AI kit)
Epoch 11/20
469/469 [==============================] - 44s 93ms/step - loss: 0.1523 - accuracy: 0.9444
Epoch 12/20
469/469 [==============================] - 41s 88ms/step - loss: 0.1375 - accuracy: 0.9495
Epoch 13/20
469/469 [==============================] - 45s 96ms/step - loss: 0.1252 - accuracy: 0.9535
Epoch 14/20
469/469 [==============================] - 39s 84ms/step - loss: 0.1164 - accuracy: 0.9572
Epoch 15/20
469/469 [==============================] - 47s 99ms/step - loss: 0.1029 - accuracy: 0.9620
Epoch 16/20
469/469 [==============================] - 47s 100ms/step - loss: 0.0962 - accuracy: 0.9648
Epoch 17/20
469/469 [==============================] - 51s 109ms/step - loss: 0.0871 - accuracy: 0.9684
Epoch 18/20
469/469 [==============================] - 40s 85ms/step - loss: 0.0764 - accuracy: 0.9723
Epoch 19/20
469/469 [==============================] - 47s 100ms/step - loss: 0.0691 - accuracy: 0.9754
Epoch 20/20
469/469 [==============================] - 45s 96ms/step - loss: 0.0620 - accuracy: 0.9779
313/313 [==============================] - 8s 24ms/step - loss: 0.3407 - accuracy: 0.9080
Test Loss: 0.3407
Test Accuracy: 0.9080

[8]: # Load the saved model
loaded_model = tf.keras.models.load_model('fashion_mnist_model.h5')

# Use the loaded model for predictions
test_loss, test_accuracy = loaded_model.evaluate(test_images, test_labels)

print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')

313/313 [==============================] - 9s 28ms/step - loss: 0.3407 - accuracy: 0.9080
Test Loss: 0.3407
Test Accuracy: 0.9080
```

**As we can see, the accuracy dropped from 0.9171 to 0.9080. However the training accuracy is around 97% which the cause of OVERFITTING.**

**To Resolve overfitting, I will be implementing L2 REGULARIZATION in my model.**

L2_Model Trial

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import time

# Load the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
keras.datasets.fashion_mnist.load_data()

# Normalize pixel values to a range of 0 to 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Reshape the images to match the expected input shape of the model
train_images = train_images.reshape((-1, 28, 28, 1))
test_images = test_images.reshape((-1, 28, 28, 1))

# Convert the labels to integers
train_labels = train_labels.astype(int)
test_labels = test_labels.astype(int)

# Define the model architecture with L2 regularization
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,
28, 1), kernel_regularizer=keras.regularizers.l2(0.001)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
kernel_regularizer=keras.regularizers.l2(0.001)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.001)),
    layers.Dense(10, activation='softmax',
kernel_regularizer=keras.regularizers.l2(0.001))
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Define a file to write the log
log_file = open('training_log_L2.txt', 'w')

# Define a custom callback to log time and accuracy after each epoch
class LogCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())
```

```
        accuracy = logs['accuracy']
        log_file.write(f"Epoch {epoch+1} - Time: {current_time} - Accuracy:
{accuracy:.4f}\n")
        log_file.flush()

# Train the model
start_time = time.time()
model.fit(train_images, train_labels, epochs=20, batch_size=128,
callbacks=[LogCallback()])
end_time = time.time()
total_time = end_time - start_time

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
log_file.write(f"\nTest Loss: {test_loss:.4f}\n")
log_file.write(f"Test Accuracy: {test_accuracy:.4f}\n")
log_file.write(f"Total Training Time: {total_time:.2f} seconds\n")
log_file.close()

model.save('fashion_mnist_model_L2.h5')
```

---------------------------------------------------------------------------------------------------------------------------

training and test results:

Epoch 1 - Time: 2023-06-29 06:37:48 - Accuracy: 0.7989

Epoch 2 - Time: 2023-06-29 06:38:03 - Accuracy: 0.8601

Epoch 3 - Time: 2023-06-29 06:38:18 - Accuracy: 0.8731

Epoch 4 - Time: 2023-06-29 06:38:33 - Accuracy: 0.8803

Epoch 5 - Time: 2023-06-29 06:38:48 - Accuracy: 0.8860

Epoch 6 - Time: 2023-06-29 06:39:04 - Accuracy: 0.8883

Epoch 7 - Time: 2023-06-29 06:39:19 - Accuracy: 0.8915

Epoch 8 - Time: 2023-06-29 06:39:34 - Accuracy: 0.8942

Epoch 9 - Time: 2023-06-29 06:39:49 - Accuracy: 0.8964

Epoch 10 - Time: 2023-06-29 06:40:04 - Accuracy: 0.8971

Epoch 11 - Time: 2023-06-29 06:40:19 - Accuracy: 0.8995

Epoch 12 - Time: 2023-06-29 06:40:34 - Accuracy: 0.9012

Epoch 13 - Time: 2023-06-29 06:40:49 - Accuracy: 0.9021

Epoch 14 - Time: 2023-06-29 06:41:04 - Accuracy: 0.9037

Epoch 15 - Time: 2023-06-29 06:41:19 - Accuracy: 0.9028

Epoch 16 - Time: 2023-06-29 06:41:35 - Accuracy: 0.9033

Epoch 17 - Time: 2023-06-29 06:41:50 - Accuracy: 0.9043

Epoch 18 - Time: 2023-06-29 06:42:06 - Accuracy: 0.9054

Epoch 19 - Time: 2023-06-29 06:42:21 - Accuracy: 0.9064

Epoch 20 - Time: 2023-06-29 06:42:36 - Accuracy: 0.9076

Test Loss: 0.4057

**Test Accuracy: 0.8944**

Total Training Time: 304.46 seconds

---

**Intel Optimization for TensorFlow (L2_model)**

Epoch 1 - Time: 2023-06-29 06:50:38 - Accuracy: 0.8027

Epoch 2 - Time: 2023-06-29 06:51:31 - Accuracy: 0.8617

Epoch 3 - Time: 2023-06-29 06:52:14 - Accuracy: 0.8745

Epoch 4 - Time: 2023-06-29 06:53:03 - Accuracy: 0.8806

Epoch 5 - Time: 2023-06-29 06:53:50 - Accuracy: 0.8848

Epoch 6 - Time: 2023-06-29 06:54:30 - Accuracy: 0.8867

Epoch 7 - Time: 2023-06-29 06:55:15 - Accuracy: 0.8912

Epoch 8 - Time: 2023-06-29 06:55:57 - Accuracy: 0.8918

Epoch 9 - Time: 2023-06-29 06:56:41 - Accuracy: 0.8953

Epoch 10 - Time: 2023-06-29 06:57:22 - Accuracy: 0.8957

Epoch 11 - Time: 2023-06-29 06:58:05 - Accuracy: 0.8980

Epoch 12 - Time: 2023-06-29 06:58:47 - Accuracy: 0.9008

Epoch 13 - Time: 2023-06-29 06:59:33 - Accuracy: 0.9000

Epoch 14 - Time: 2023-06-29 07:00:16 - Accuracy: 0.9019

Epoch 15 - Time: 2023-06-29 07:00:58 - Accuracy: 0.9014

Epoch 16 - Time: 2023-06-29 07:01:44 - Accuracy: 0.9057

Epoch 17 - Time: 2023-06-29 07:02:28 - Accuracy: 0.9041

Epoch 18 - Time: 2023-06-29 07:03:14 - Accuracy: 0.9045

Epoch 19 - Time: 2023-06-29 07:03:56 - Accuracy: 0.9065

Epoch 20 - Time: 2023-06-29 07:04:41 - Accuracy: 0.9082

Test Loss: 0.3983

**Test Accuracy: 0.9007**

Total Training Time: 897.75 seconds

To further reduce the overfitting, I will be implementing the **dropout** regularization:

---

**Dropout Regularization**

**Offline (No optimization):**

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import time

# Load the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
keras.datasets.fashion_mnist.load_data()

# Normalize pixel values to a range of 0 to 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Reshape the images to match the expected input shape of the model
train_images = train_images.reshape((-1, 28, 28, 1))
test_images = test_images.reshape((-1, 28, 28, 1))

# Convert the labels to integers
train_labels = train_labels.astype(int)
test_labels = test_labels.astype(int)

# Define the model architecture with dropout regularization
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,
28, 1)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.25),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

```python
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Define a file to write the log
log_file = open('training_log_drop.txt', 'w')

# Define a custom callback to log time and accuracy after each epoch
class LogCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())
        accuracy = logs['accuracy']
        log_file.write(f"Epoch {epoch+1} - Time: {current_time} - Accuracy:
{accuracy:.4f}\n")
        log_file.flush()

# Train the model
start_time = time.time()
model.fit(train_images, train_labels, epochs=20, batch_size=128,
callbacks=[LogCallback()])
end_time = time.time()
total_time = end_time - start_time

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
log_file.write(f"\nTest Loss: {test_loss:.4f}\n")
log_file.write(f"Test Accuracy: {test_accuracy:.4f}\n")
log_file.write(f"Total Training Time: {total_time:.2f} seconds\n")
log_file.close()

# Save the model
model.save('fashion_mnist_model_drop.h5')
```

**training_and_test_results:**

Epoch 1 - Time: 2023-06-29 07:14:41 - Accuracy: 0.7338

Epoch 2 - Time: 2023-06-29 07:14:58 - Accuracy: 0.8270

Epoch 3 - Time: 2023-06-29 07:15:16 - Accuracy: 0.8480

Epoch 4 - Time: 2023-06-29 07:15:33 - Accuracy: 0.8601

Epoch 5 - Time: 2023-06-29 07:15:50 - Accuracy: 0.8705

Epoch 6 - Time: 2023-06-29 07:16:07 - Accuracy: 0.8776

Epoch 7 - Time: 2023-06-29 07:16:24 - Accuracy: 0.8810

Epoch 8 - Time: 2023-06-29 07:16:42 - Accuracy: 0.8848

Epoch 9 - Time: 2023-06-29 07:17:00 - Accuracy: 0.8900

Epoch 10 - Time: 2023-06-29 07:17:18 - Accuracy: 0.8924

Epoch 11 - Time: 2023-06-29 07:17:36 - Accuracy: 0.8947

Epoch 12 - Time: 2023-06-29 07:17:54 - Accuracy: 0.8963

Epoch 13 - Time: 2023-06-29 07:18:12 - Accuracy: 0.8997

Epoch 14 - Time: 2023-06-29 07:18:30 - Accuracy: 0.9008

Epoch 15 - Time: 2023-06-29 07:18:48 - Accuracy: 0.9025

Epoch 16 - Time: 2023-06-29 07:19:05 - Accuracy: 0.9041

Epoch 17 - Time: 2023-06-29 07:19:21 - Accuracy: 0.9066

Epoch 18 - Time: 2023-06-29 07:19:39 - Accuracy: 0.9070

Epoch 19 - Time: 2023-06-29 07:19:56 - Accuracy: 0.9073

Epoch 20 - Time: 2023-06-29 07:20:13 - Accuracy: 0.9086

Test Loss: 0.2384

**Test Accuracy: 0.9119**

Total Training Time: 350.56 seconds

---

**Intel Optimization for TensorFlow (Dropout-model)**

Epoch 1 - Time: 2023-06-29 07:16:11 - Accuracy: 0.7369

Epoch 2 - Time: 2023-06-29 07:17:10 - Accuracy: 0.8271

Epoch 3 - Time: 2023-06-29 07:18:00 - Accuracy: 0.8510

Epoch 4 - Time: 2023-06-29 07:18:53 - Accuracy: 0.8628

Epoch 5 - Time: 2023-06-29 07:19:45 - Accuracy: 0.8715

Epoch 6 - Time: 2023-06-29 07:20:37 - Accuracy: 0.8771

Epoch 7 - Time: 2023-06-29 07:21:26 - Accuracy: 0.8835

Epoch 8 - Time: 2023-06-29 07:22:16 - Accuracy: 0.8849

Epoch 9 - Time: 2023-06-29 07:23:06 - Accuracy: 0.8893

Epoch 10 - Time: 2023-06-29 07:23:53 - Accuracy: 0.8934

Epoch 11 - Time: 2023-06-29 07:24:41 - Accuracy: 0.8942

Epoch 12 - Time: 2023-06-29 07:25:33 - Accuracy: 0.8978

Epoch 13 - Time: 2023-06-29 07:26:23 - Accuracy: 0.9002

Epoch 14 - Time: 2023-06-29 07:27:12 - Accuracy: 0.9011

Epoch 15 - Time: 2023-06-29 07:28:02 - Accuracy: 0.9026

Epoch 16 - Time: 2023-06-29 07:28:53 - Accuracy: 0.9063

Epoch 17 - Time: 2023-06-29 07:29:46 - Accuracy: 0.9065

Epoch 18 - Time: 2023-06-29 07:30:42 - Accuracy: 0.9082

Epoch 19 - Time: 2023-06-29 07:31:33 - Accuracy: 0.9078

Epoch 20 - Time: 2023-06-29 07:32:23 - Accuracy: 0.9113

Test Loss: 0.2357

**Test Accuracy: 0.9117**

Total Training Time: 1035.96 seconds

---

**Both L2 and Dropout Regularization**

Next, I would be combining both L2 and Dropout:

**Offline (No optimization):**

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import time

# Load the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
keras.datasets.fashion_mnist.load_data()

# Normalize pixel values to a range of 0 to 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Reshape the images to match the expected input shape of the model
train_images = train_images.reshape((-1, 28, 28, 1))
test_images = test_images.reshape((-1, 28, 28, 1))

# Convert the labels to integers
train_labels = train_labels.astype(int)
test_labels = test_labels.astype(int)

# Define the model architecture with both L2 regularization and dropout
regularization
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,
28, 1),
                  kernel_regularizer=keras.regularizers.l2(0.001)),
    layers.MaxPooling2D(pool_size=(2, 2)),
```

```python
    layers.Dropout(0.25),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
                  kernel_regularizer=keras.regularizers.l2(0.001)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu',
                 kernel_regularizer=keras.regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Define a file to write the log
log_file = open('training_log_both.txt', 'w')

# Define a custom callback to log time and accuracy after each epoch
class LogCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())
        accuracy = logs['accuracy']
        log_file.write(f"Epoch {epoch+1} - Time: {current_time} - Accuracy:
{accuracy:.4f}\n")
        log_file.flush()

# Train the model
start_time = time.time()
model.fit(train_images, train_labels, epochs=20, batch_size=128,
callbacks=[LogCallback()])
end_time = time.time()
total_time = end_time - start_time

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
log_file.write(f"\nTest Loss: {test_loss:.4f}\n")
log_file.write(f"Test Accuracy: {test_accuracy:.4f}\n")
log_file.write(f"Total Training Time: {total_time:.2f} seconds\n")
log_file.close()

# Save the model
model.save('fashion_mnist_model_both.h5')
```

training_and_test_results:

Epoch 1 - Time: 2023-06-29 07:26:28 - Accuracy: 0.7356

Epoch 2 - Time: 2023-06-29 07:26:46 - Accuracy: 0.8210

Epoch 3 - Time: 2023-06-29 07:27:03 - Accuracy: 0.8402

Epoch 4 - Time: 2023-06-29 07:27:21 - Accuracy: 0.8492

Epoch 5 - Time: 2023-06-29 07:27:39 - Accuracy: 0.8561

Epoch 6 - Time: 2023-06-29 07:27:57 - Accuracy: 0.8596

Epoch 7 - Time: 2023-06-29 07:28:14 - Accuracy: 0.8633

Epoch 8 - Time: 2023-06-29 07:28:32 - Accuracy: 0.8654

Epoch 9 - Time: 2023-06-29 07:28:49 - Accuracy: 0.8693

Epoch 10 - Time: 2023-06-29 07:29:07 - Accuracy: 0.8734

Epoch 11 - Time: 2023-06-29 07:29:24 - Accuracy: 0.8719

Epoch 12 - Time: 2023-06-29 07:29:42 - Accuracy: 0.8740

Epoch 13 - Time: 2023-06-29 07:30:00 - Accuracy: 0.8767

Epoch 14 - Time: 2023-06-29 07:30:17 - Accuracy: 0.8761

Epoch 15 - Time: 2023-06-29 07:30:36 - Accuracy: 0.8763

Epoch 16 - Time: 2023-06-29 07:30:54 - Accuracy: 0.8789

Epoch 17 - Time: 2023-06-29 07:31:12 - Accuracy: 0.8783

Epoch 18 - Time: 2023-06-29 07:31:31 - Accuracy: 0.8803

Epoch 19 - Time: 2023-06-29 07:31:48 - Accuracy: 0.8816

Epoch 20 - Time: 2023-06-29 07:32:05 - Accuracy: 0.8818

Test Loss: 0.4138

**Test Accuracy: 0.8894**

Total Training Time: 355.06 seconds

**Intel Optimization for Tensorflow (Both-L2-Dropout_model)**

Epoch 1 - Time: 2023-06-29 08:00:20 - Accuracy: 0.7298

Epoch 2 - Time: 2023-06-29 08:01:06 - Accuracy: 0.8184

Epoch 3 - Time: 2023-06-29 08:01:52 - Accuracy: 0.8360

Epoch 4 - Time: 2023-06-29 08:02:36 - Accuracy: 0.8469

Epoch 5 - Time: 2023-06-29 08:03:23 - Accuracy: 0.8534

Epoch 6 - Time: 2023-06-29 08:04:08 - Accuracy: 0.8600

Epoch 7 - Time: 2023-06-29 08:04:53 - Accuracy: 0.8629

Epoch 8 - Time: 2023-06-29 08:05:37 - Accuracy: 0.8659

Epoch 9 - Time: 2023-06-29 08:06:21 - Accuracy: 0.8691

Epoch 10 - Time: 2023-06-29 08:07:04 - Accuracy: 0.8695

Epoch 11 - Time: 2023-06-29 08:07:49 - Accuracy: 0.8729

Epoch 12 - Time: 2023-06-29 08:08:34 - Accuracy: 0.8749

Epoch 13 - Time: 2023-06-29 08:09:21 - Accuracy: 0.8750

Epoch 14 - Time: 2023-06-29 08:10:07 - Accuracy: 0.8772

Epoch 15 - Time: 2023-06-29 08:10:52 - Accuracy: 0.8782

Epoch 16 - Time: 2023-06-29 08:11:37 - Accuracy: 0.8787

Epoch 17 - Time: 2023-06-29 08:12:23 - Accuracy: 0.8797

Epoch 18 - Time: 2023-06-29 08:13:08 - Accuracy: 0.8807

Epoch 19 - Time: 2023-06-29 08:13:52 - Accuracy: 0.8806

Epoch 20 - Time: 2023-06-29 08:14:38 - Accuracy: 0.8819

Test Loss: 0.4062

**Test Accuracy: 0.8933**

Total Training Time: 903.85 seconds

**FINAL MODEL**

**From all the data that has been collected, we can conclude that my model with DROPOUT regularization gives the best accuracy.**

**So, here are the results for both offline and online with intel optimization for TensorFlow.**

## Offline – No Optimizations:

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import time

# Load the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
keras.datasets.fashion_mnist.load_data()

# Normalize pixel values to a range of 0 to 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Reshape the images to match the expected input shape of the model
train_images = train_images.reshape((-1, 28, 28, 1))
test_images = test_images.reshape((-1, 28, 28, 1))

# Convert the labels to integers
train_labels = train_labels.astype(int)
test_labels = test_labels.astype(int)

# Define the model architecture with dropout regularization
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,
28, 1)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.25),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

```python
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Define a file to write the log
log_file = open('training_log_drop2.txt', 'w')

# Define a custom callback to log time and accuracy after each epoch
class LogCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())
        accuracy = logs['accuracy']
        log_file.write(f"Epoch {epoch+1} - Time: {current_time} - Accuracy:
{accuracy:.4f}\n")
        log_file.flush()

# Train the model
start_time = time.time()
model.fit(train_images, train_labels, epochs=100, batch_size=64,
callbacks=[LogCallback()])
end_time = time.time()
total_time = end_time - start_time

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
log_file.write(f"\nTest Loss: {test_loss:.4f}\n")
log_file.write(f"Test Accuracy: {test_accuracy:.4f}\n")
log_file.write(f"Total Training Time: {total_time:.2f} seconds\n")
log_file.close()

# Save the model
model.save('fashion_mnist_model_drop2.h5')
```

## test results for 100 epochs and batch size of 64:

Epoch 1 - Time: 2023-06-29 08:25:54 - Accuracy: 0.7591

Epoch 2 - Time: 2023-06-29 08:26:14 - Accuracy: 0.8375

Epoch 3 - Time: 2023-06-29 08:26:34 - Accuracy: 0.8571

Epoch 4 - Time: 2023-06-29 08:26:53 - Accuracy: 0.8692

Epoch 5 - Time: 2023-06-29 08:27:11 - Accuracy: 0.8756

Epoch 6 - Time: 2023-06-29 08:27:29 - Accuracy: 0.8819

Epoch 7 - Time: 2023-06-29 08:27:47 - Accuracy: 0.8866

Epoch 8 - Time: 2023-06-29 08:28:05 - Accuracy: 0.8908

Epoch 9 - Time: 2023-06-29 08:28:23 - Accuracy: 0.8930

Epoch 10 - Time: 2023-06-29 08:28:41 - Accuracy: 0.8954

Epoch 11 - Time: 2023-06-29 08:28:59 - Accuracy: 0.8975

Epoch 12 - Time: 2023-06-29 08:29:17 - Accuracy: 0.9008

Epoch 13 - Time: 2023-06-29 08:29:35 - Accuracy: 0.9017

Epoch 14 - Time: 2023-06-29 08:29:53 - Accuracy: 0.9030

Epoch 15 - Time: 2023-06-29 08:30:11 - Accuracy: 0.9047

Epoch 16 - Time: 2023-06-29 08:30:29 - Accuracy: 0.9082

Epoch 17 - Time: 2023-06-29 08:30:46 - Accuracy: 0.9076

Epoch 18 - Time: 2023-06-29 08:31:06 - Accuracy: 0.9081

Epoch 19 - Time: 2023-06-29 08:31:25 - Accuracy: 0.9073

Epoch 20 - Time: 2023-06-29 08:31:42 - Accuracy: 0.9107

Epoch 21 - Time: 2023-06-29 08:32:00 - Accuracy: 0.9118

Epoch 22 - Time: 2023-06-29 08:32:18 - Accuracy: 0.9116

Epoch 23 - Time: 2023-06-29 08:32:36 - Accuracy: 0.9118

Epoch 24 - Time: 2023-06-29 08:32:53 - Accuracy: 0.9139

Epoch 25 - Time: 2023-06-29 08:33:11 - Accuracy: 0.9144

Epoch 26 - Time: 2023-06-29 08:33:29 - Accuracy: 0.9156

Epoch 27 - Time: 2023-06-29 08:33:47 - Accuracy: 0.9152

Epoch 28 - Time: 2023-06-29 08:34:04 - Accuracy: 0.9157

Epoch 29 - Time: 2023-06-29 08:34:22 - Accuracy: 0.9174

Epoch 30 - Time: 2023-06-29 08:34:40 - Accuracy: 0.9176

Epoch 31 - Time: 2023-06-29 08:34:57 - Accuracy: 0.9158

Epoch 32 - Time: 2023-06-29 08:35:15 - Accuracy: 0.9180

Epoch 33 - Time: 2023-06-29 08:35:33 - Accuracy: 0.9187

Epoch 34 - Time: 2023-06-29 08:35:51 - Accuracy: 0.9183

Epoch 35 - Time: 2023-06-29 08:36:08 - Accuracy: 0.9192

Epoch 36 - Time: 2023-06-29 08:36:26 - Accuracy: 0.9193

Epoch 37 - Time: 2023-06-29 08:36:44 - Accuracy: 0.9197

Epoch 38 - Time: 2023-06-29 08:37:01 - Accuracy: 0.9192

Epoch 39 - Time: 2023-06-29 08:37:19 - Accuracy: 0.9205

Epoch 40 - Time: 2023-06-29 08:37:36 - Accuracy: 0.9211

Epoch 41 - Time: 2023-06-29 08:37:54 - Accuracy: 0.9218

Epoch 42 - Time: 2023-06-29 08:38:11 - Accuracy: 0.9220

Epoch 43 - Time: 2023-06-29 08:38:29 - Accuracy: 0.9216

Epoch 44 - Time: 2023-06-29 08:38:46 - Accuracy: 0.9242

Epoch 45 - Time: 2023-06-29 08:39:04 - Accuracy: 0.9237

Epoch 46 - Time: 2023-06-29 08:39:21 - Accuracy: 0.9229

Epoch 47 - Time: 2023-06-29 08:39:39 - Accuracy: 0.9237

Epoch 48 - Time: 2023-06-29 08:39:56 - Accuracy: 0.9237

Epoch 49 - Time: 2023-06-29 08:40:14 - Accuracy: 0.9226

Epoch 50 - Time: 2023-06-29 08:40:31 - Accuracy: 0.9241

Epoch 51 - Time: 2023-06-29 08:40:48 - Accuracy: 0.9240

Epoch 52 - Time: 2023-06-29 08:41:06 - Accuracy: 0.9250

Epoch 53 - Time: 2023-06-29 08:41:23 - Accuracy: 0.9252

Epoch 54 - Time: 2023-06-29 08:41:40 - Accuracy: 0.9253

Epoch 55 - Time: 2023-06-29 08:41:58 - Accuracy: 0.9245

Epoch 56 - Time: 2023-06-29 08:42:15 - Accuracy: 0.9266

Epoch 57 - Time: 2023-06-29 08:42:33 - Accuracy: 0.9259

Epoch 58 - Time: 2023-06-29 08:42:50 - Accuracy: 0.9261

Epoch 59 - Time: 2023-06-29 08:43:08 - Accuracy: 0.9259

Epoch 60 - Time: 2023-06-29 08:43:25 - Accuracy: 0.9269

Epoch 61 - Time: 2023-06-29 08:43:43 - Accuracy: 0.9255

Epoch 62 - Time: 2023-06-29 08:44:01 - Accuracy: 0.9261

Epoch 63 - Time: 2023-06-29 08:44:19 - Accuracy: 0.9268

Epoch 64 - Time: 2023-06-29 08:44:36 - Accuracy: 0.9280

Epoch 65 - Time: 2023-06-29 08:44:54 - Accuracy: 0.9264

Epoch 66 - Time: 2023-06-29 08:45:11 - Accuracy: 0.9277

Epoch 67 - Time: 2023-06-29 08:45:29 - Accuracy: 0.9266

Epoch 68 - Time: 2023-06-29 08:45:46 - Accuracy: 0.9268

Epoch 69 - Time: 2023-06-29 08:46:04 - Accuracy: 0.9285

Epoch 70 - Time: 2023-06-29 08:46:21 - Accuracy: 0.9291

Epoch 71 - Time: 2023-06-29 08:46:38 - Accuracy: 0.9283

Epoch 72 - Time: 2023-06-29 08:46:56 - Accuracy: 0.9280

Epoch 73 - Time: 2023-06-29 08:47:14 - Accuracy: 0.9280

Epoch 74 - Time: 2023-06-29 08:47:31 - Accuracy: 0.9278

Epoch 75 - Time: 2023-06-29 08:47:49 - Accuracy: 0.9290

Epoch 76 - Time: 2023-06-29 08:48:07 - Accuracy: 0.9286

Epoch 77 - Time: 2023-06-29 08:48:24 - Accuracy: 0.9273

Epoch 78 - Time: 2023-06-29 08:48:42 - Accuracy: 0.9292

Epoch 79 - Time: 2023-06-29 08:49:00 - Accuracy: 0.9300

Epoch 80 - Time: 2023-06-29 08:49:17 - Accuracy: 0.9301

Epoch 81 - Time: 2023-06-29 08:49:35 - Accuracy: 0.9296

Epoch 82 - Time: 2023-06-29 08:49:53 - Accuracy: 0.9303

Epoch 83 - Time: 2023-06-29 08:50:11 - Accuracy: 0.9306

Epoch 84 - Time: 2023-06-29 08:50:28 - Accuracy: 0.9319

Epoch 85 - Time: 2023-06-29 08:50:46 - Accuracy: 0.9301

Epoch 86 - Time: 2023-06-29 08:51:04 - Accuracy: 0.9304

Epoch 87 - Time: 2023-06-29 08:51:22 - Accuracy: 0.9330

Epoch 88 - Time: 2023-06-29 08:51:39 - Accuracy: 0.9306

Epoch 89 - Time: 2023-06-29 08:51:57 - Accuracy: 0.9287

Epoch 90 - Time: 2023-06-29 08:52:15 - Accuracy: 0.9311

Epoch 91 - Time: 2023-06-29 08:52:33 - Accuracy: 0.9309

Epoch 92 - Time: 2023-06-29 08:52:50 - Accuracy: 0.9312

Epoch 93 - Time: 2023-06-29 08:53:08 - Accuracy: 0.9320

Epoch 94 - Time: 2023-06-29 08:53:25 - Accuracy: 0.9334

Epoch 95 - Time: 2023-06-29 08:53:43 - Accuracy: 0.9324

Epoch 96 - Time: 2023-06-29 08:54:00 - Accuracy: 0.9319

Epoch 97 - Time: 2023-06-29 08:54:18 - Accuracy: 0.9323

Epoch 98 - Time: 2023-06-29 08:54:35 - Accuracy: 0.9340

Epoch 99 - Time: 2023-06-29 08:54:53 - Accuracy: 0.9324

Epoch 100 - Time: 2023-06-29 08:55:10 - Accuracy: 0.9339

Test Loss: 0.2441

**Test Accuracy: 0.9185**

Total Training Time: 1776.13 seconds

---

## Test results on Intel DevCloud with 40 epochs:

Epoch 1 - Time: 2023-06-29 08:21:15 - Accuracy: 0.7556

Epoch 2 - Time: 2023-06-29 08:22:30 - Accuracy: 0.8368

Epoch 3 - Time: 2023-06-29 08:23:47 - Accuracy: 0.8584

Epoch 4 - Time: 2023-06-29 08:25:03 - Accuracy: 0.8711

Epoch 5 - Time: 2023-06-29 08:26:20 - Accuracy: 0.8787

Epoch 6 - Time: 2023-06-29 08:27:38 - Accuracy: 0.8841

Epoch 7 - Time: 2023-06-29 08:28:57 - Accuracy: 0.8899

Epoch 8 - Time: 2023-06-29 08:30:13 - Accuracy: 0.8928

Epoch 9 - Time: 2023-06-29 08:31:29 - Accuracy: 0.8964

Epoch 10 - Time: 2023-06-29 08:32:51 - Accuracy: 0.8981

Epoch 11 - Time: 2023-06-29 08:34:11 - Accuracy: 0.8984

Epoch 12 - Time: 2023-06-29 08:35:28 - Accuracy: 0.9006

Epoch 13 - Time: 2023-06-29 08:36:45 - Accuracy: 0.9037

Epoch 14 - Time: 2023-06-29 08:38:02 - Accuracy: 0.9060

Epoch 15 - Time: 2023-06-29 08:39:24 - Accuracy: 0.9054

Epoch 16 - Time: 2023-06-29 08:40:42 - Accuracy: 0.9075

Epoch 17 - Time: 2023-06-29 08:41:59 - Accuracy: 0.9083

Epoch 18 - Time: 2023-06-29 08:43:16 - Accuracy: 0.9104

Epoch 19 - Time: 2023-06-29 08:44:36 - Accuracy: 0.9095

Epoch 20 - Time: 2023-06-29 08:45:54 - Accuracy: 0.9111

Epoch 21 - Time: 2023-06-29 08:47:13 - Accuracy: 0.9112

Epoch 22 - Time: 2023-06-29 08:48:34 - Accuracy: 0.9149

Epoch 23 - Time: 2023-06-29 08:49:51 - Accuracy: 0.9135

Epoch 24 - Time: 2023-06-29 08:51:11 - Accuracy: 0.9153

Epoch 25 - Time: 2023-06-29 08:52:27 - Accuracy: 0.9139

Epoch 26 - Time: 2023-06-29 08:53:47 - Accuracy: 0.9150

Epoch 27 - Time: 2023-06-29 08:55:08 - Accuracy: 0.9172

Epoch 28 - Time: 2023-06-29 08:56:25 - Accuracy: 0.9144

Epoch 29 - Time: 2023-06-29 08:57:40 - Accuracy: 0.9161

Epoch 30 - Time: 2023-06-29 08:58:57 - Accuracy: 0.9182

Epoch 31 - Time: 2023-06-29 09:00:13 - Accuracy: 0.9201

Epoch 32 - Time: 2023-06-29 09:01:30 - Accuracy: 0.9184

Epoch 33 - Time: 2023-06-29 09:02:46 - Accuracy: 0.9207

Epoch 34 - Time: 2023-06-29 09:04:07 - Accuracy: 0.9190

Epoch 35 - Time: 2023-06-29 09:05:25 - Accuracy: 0.9190

Epoch 36 - Time: 2023-06-29 09:06:44 - Accuracy: 0.9210

Epoch 37 - Time: 2023-06-29 09:08:01 - Accuracy: 0.9197

Epoch 38 - Time: 2023-06-29 09:09:21 - Accuracy: 0.9210

Epoch 39 - Time: 2023-06-29 09:10:39 - Accuracy: 0.9203

Epoch 40 - Time: 2023-06-29 09:11:54 - Accuracy: 0.9220

Test Loss: 0.2266

**Test Accuracy: 0.9196**

Total Training Time: 3118.77 seconds

## CONCLUSION:

To conquer the Fashion MNIST dataset with CNNs, we pre-process the data, design a CNN model, and train it using appropriate loss functions and optimization techniques. We evaluate the model's accuracy, fine-tune hyperparameters, and iterate to improve performance. By following this approach, we can effectively tackle the Fashion MNIST dataset and achieve accurate results using CNNs.