CONQUERING FASHION MNIST WITH CNN’S USING COMPUTER VISION

TEAM NAME - CODERED

MEMBERS - JENAT CICI ANIL

Date of submission – 15th July 2023

ABSTRACT

Conquering Fashion MNIST with CNN using computer vision has been a significant achievement in the field of image classification. The Fashion MNIST dataset consists of 60,000 training images and 10,000 testing images, each labeled with one of ten different clothing categories. This project focuses on solving the Fashion MNIST problem using Convolutional Neural Networks (CNNs). In this project, the CNN architecture is utilized to train a model that accurately identifies fashion items from the input images. The model's architecture includes convolutional layers for feature extraction, pooling layers for downsampling, and fully connected layers for classification. By employing techniques such as dropout regularizations helps mitigate overfitting and improved generalization. The project aims to leverage deep learning techniques to achieve high accuracy in fashion item classification, with potential applications in e-commerce and fashion recommendation systems.

INTRODUCTION

The field of computer vision has seen significant advancements in recent years, enabling machines to interpret and understand visual data with increasing accuracy. One notable application of computer vision is in the recognition and classification of fashion items. The Fashion MNIST dataset, consisting of grayscale images of various fashion products, has become a popular benchmark for evaluating the performance of computer vision models.

In this project, we explore the task of conquering the Fashion MNIST dataset using Convolutional Neural Networks (CNNs). CNNs have proven to be highly effective in image classification tasks, leveraging their ability to extract relevant features from images and learn complex patterns. By training a CNN model on the Fashion MNIST dataset, the aim is to achieve high accuracy in classifying different fashion items.

The objective of this report is to provide a comprehensive overview of the approach, methodology, and results in tackling the Fashion MNIST problem. We will discuss the motivation behind this project, review prior work in the field, describe the architecture of our CNN model, and present the experimental setup and results. By successfully conquering the Fashion MNIST dataset using CNNs, the aim is to demonstrate the power of deep learning and computer vision in solving complex image classification problems.

MOTIVATION

The motivation behind tackling the Fashion MNIST problem using CNN and computer vision techniques stems from the increasing importance of accurate and efficient fashion product classification in the digital age.The Fashion MNIST dataset serves as an ideal benchmark for this task, as it provides a diverse range of fashion items in a standardized format. By conquering this dataset, we aim to develop a model that can accurately identify and categorize different fashion items. Such a model can have numerous applications, including product recommendation systems, visual search engines, and inventory management tools.

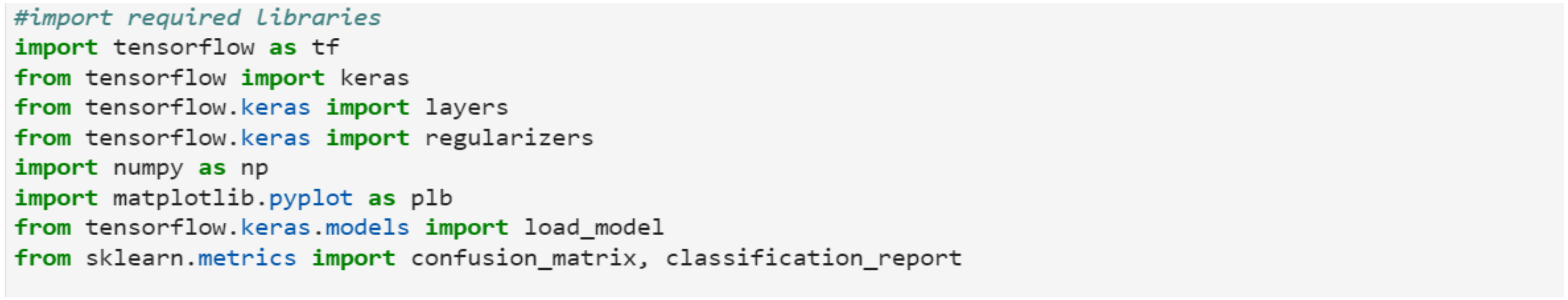
By solving the Fashion MNIST problem, I hope to contribute to the advancement of computer vision in the fashion industry, enabling retailers, designers, and consumers to better navigate the vast world of fashion products. Additionally, this work can serve as a stepping stone for more complex and challenging fashion classification problems, opening up avenues for further research and innovation in the field of computer vision and deep learning.

THE APPROACH

- A **convolutional layer** applies the ReLU activation function. The input shape of the images is (28, 28, 1), representing grayscale images. Three convolutional layers are added with filters 32,64 and 128 etc.
- After each convolutional layer, a **max-pooling layer** with a pool size of 2x2 is added. It performs downsampling by selecting the maximum value within each 2x2 region.
- To prevent overfitting, **dropout layers** are introduced after each max-pooling layer. They randomly deactivate a certain percentage of the neurons during training, forcing the model to learn more robust and generalized representations.
- To feed the extracted features into the dense layers, a **flatten layer** is used to reshape the 3D feature maps into a 1D vector.
- **Dense layers** with 256 units and ReLU activation are added. They serve as fully connected layers to learn high-level abstractions from the flattened features.
- The last dense layer consists of 512 units, followed by a softmax activation function with 10 units, representing the 10 fashion categories. It produces the final classification probabilities for each category.

My primary objective in this project was to develop an architecture for CNNs that would perform image classification on the fashion MNIST dataset, and I used Tensorflow to accomplish this. The process is divided into steps such as data analysis, designing the architecture, training the model, and making predictions.

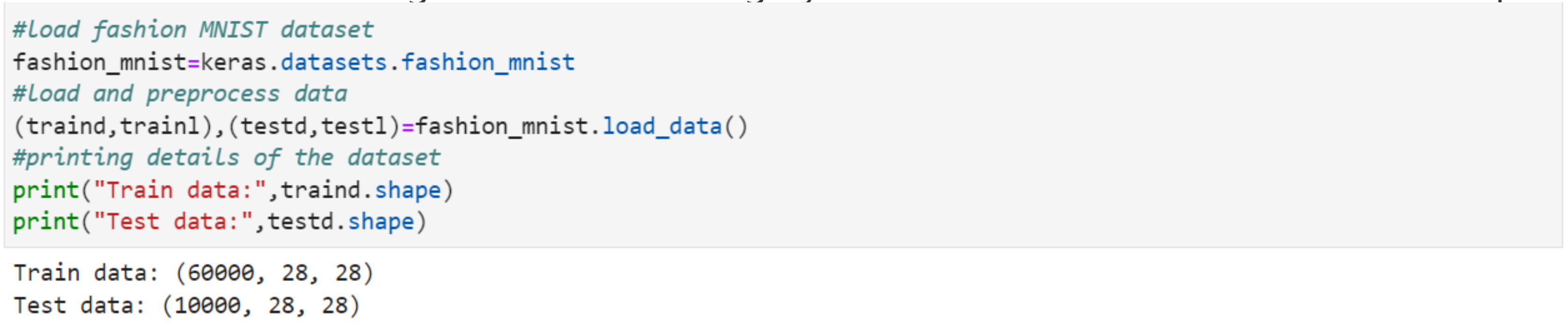
- As the initial step the required libraries were imported.



```
#import required libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
import numpy as np
import matplotlib.pyplot as plb
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, classification_report
```

- Loading the dataset and data analysis.

  The fashion MNIST dataset consists of 60,000 images for the training set and 10,000 images for the testing set where each image is a 28 x 28 size grayscale. The dataset is loaded and then split into training and testing.

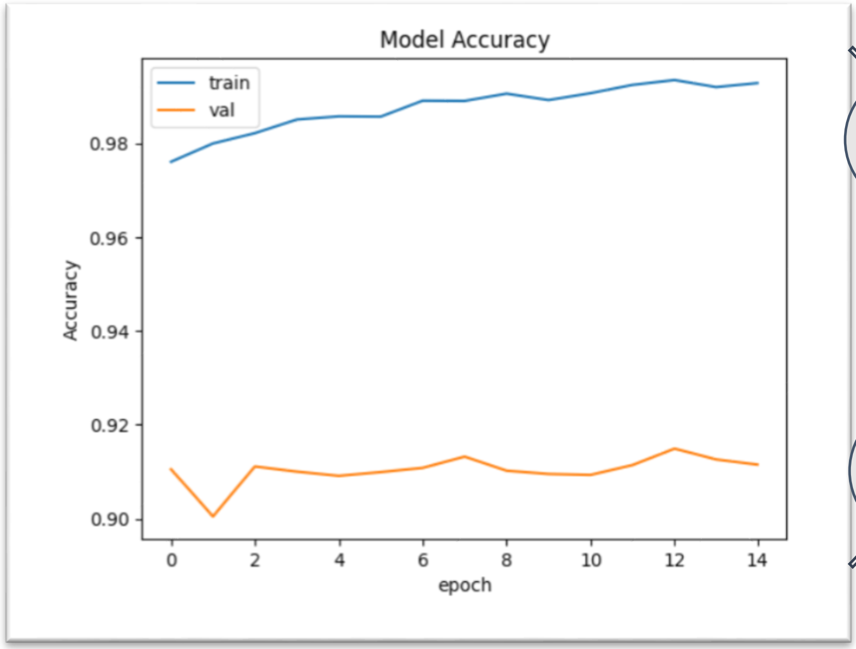


```
#load fashion MNIST dataset
fashion_mnist=keras.datasets.fashion_mnist
#load and preprocess data
(traind,trainl),(testd,testl)=fashion_mnist.load_data()
#printing details of the dataset
print("Train data:",traind.shape)
print("Test data:",testd.shape)
```

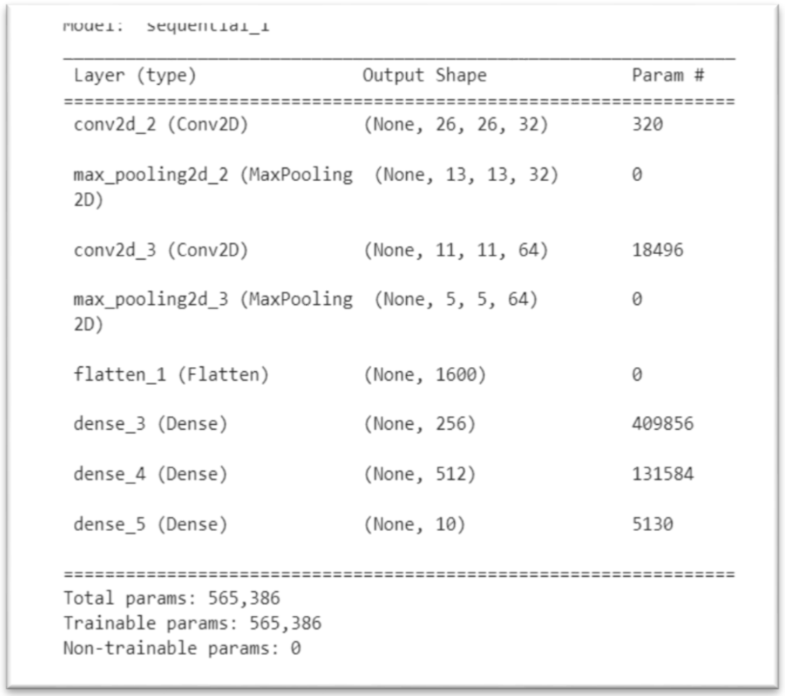
```
Train data: (60000, 28, 28)
Test data: (10000, 28, 28)
```

The model was validated and the hyperparameters were tuned at the training stage itself.

Initially, I tested a CNN model without applying any regularizations, and the following results were obtained:



99.28	Train accuracy
91.15	Validation accuracy
91.14	Test accuracy



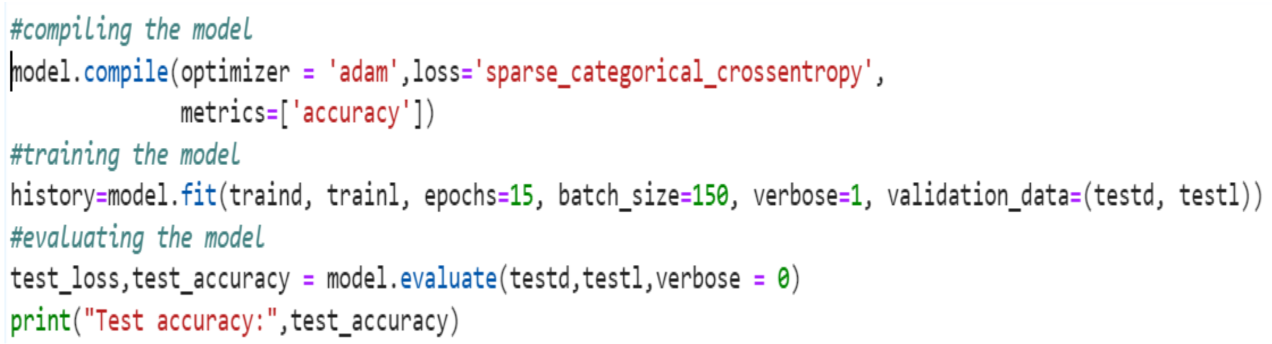
```
Model: sequential_1
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 1600)              0

 dense_3 (Dense)             (None, 256)               409856

 dense_4 (Dense)             (None, 512)               131584

 dense_5 (Dense)             (None, 10)                5130

=================================================================
Total params: 565,386
Trainable params: 565,386
Non-trainable params: 0
```

The architecture

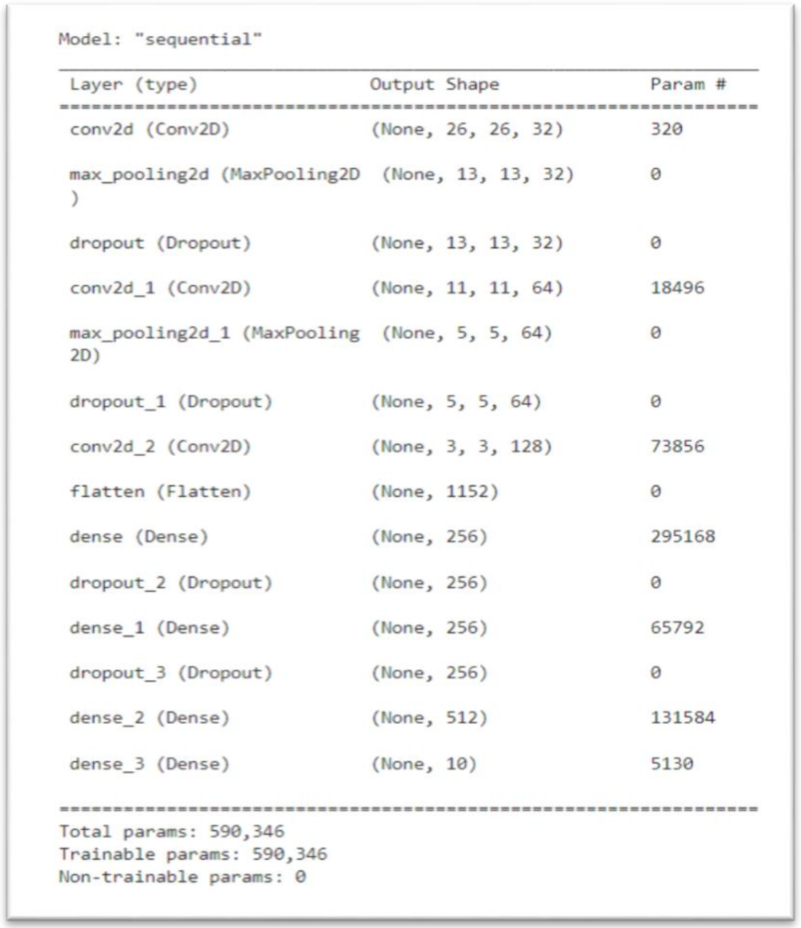
In general, deep learning models typically have a high tendency to overfit (perform better on the training dataset as compared to the validation/test dataset). The above results demonstrate that fact.
A training accuracy of 99% and test accuracy of 91% confirms that model is overfitting.

In order to solve the overfitting issue, Dropout layers were added for regularization, and convolutional and dense layers were added. The modified architecture is :

- Compiling and training of the model



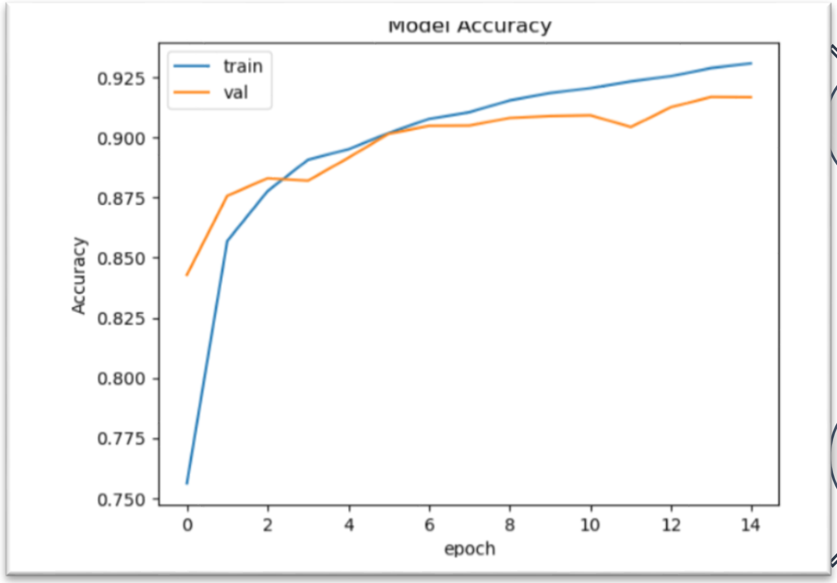
```
#compiling the model
model.compile(optimizer = 'adam',loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
#training the model
history=model.fit(traind, trainl, epochs=15, batch_size=150, verbose=1, validation_data=(testd, testl))
#evaluating the model
test_loss,test_accuracy = model.evaluate(testd,testl,verbose = 0)
print("Test accuracy:",test_accuracy)
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 256)               295168

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 dropout_3 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 512)               131584

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 590,346
Trainable params: 590,346
Non-trainable params: 0
```

- The model is compiled using ‘adam’ optimizer.
- The ‘sparse_categorical_crossentropy’ is used as the loss function to minimize the loss.
- After compiling the model was trained by adjusting and experimenting with various batch size and number of epochs .
- After number of experiments the number of epochs were set to 15 and batch size is set to 150 for better accuracy.
- The model is validated based on the test data

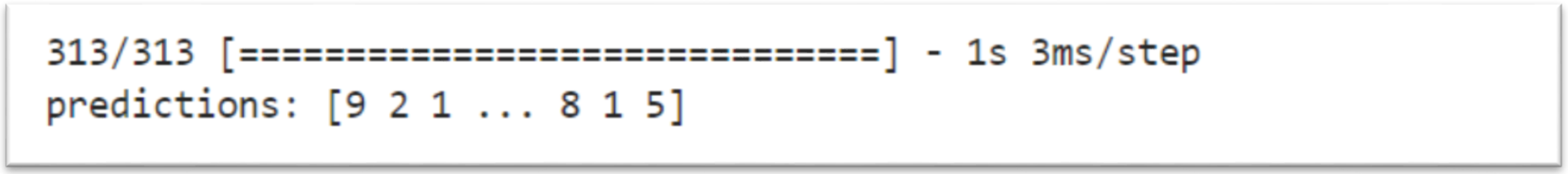
After compiling and training the model, the following results were obtained



- 93.09 Train accuracy
- 91.68 Validation accuracy
- 91.68 Test accuracy

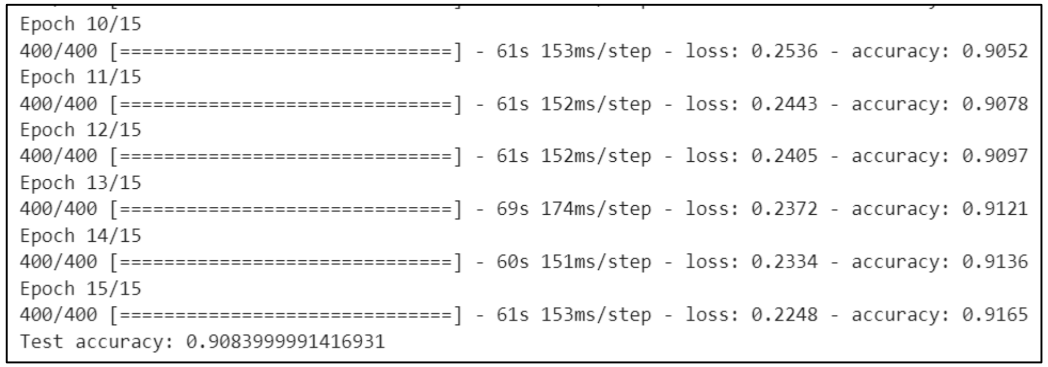
The model performed well in terms of training accuracy, with 92%, and test accuracy, with 91%. It is evident that the model is performing well and there is no over fitting. So the model is finalized.

Predictions were made with the model and the resultant array of prediction is obtained.



```
313/313 [==============================] - 1s 3ms/step
predictions: [9 2 1 ... 8 1 5]
```

The model was compiled and trained on different platforms, such as Google Colab, Intel Developer Cloud, etc. The model on the Intel DevCloud seemed to perform significantly better than those on other platforms.



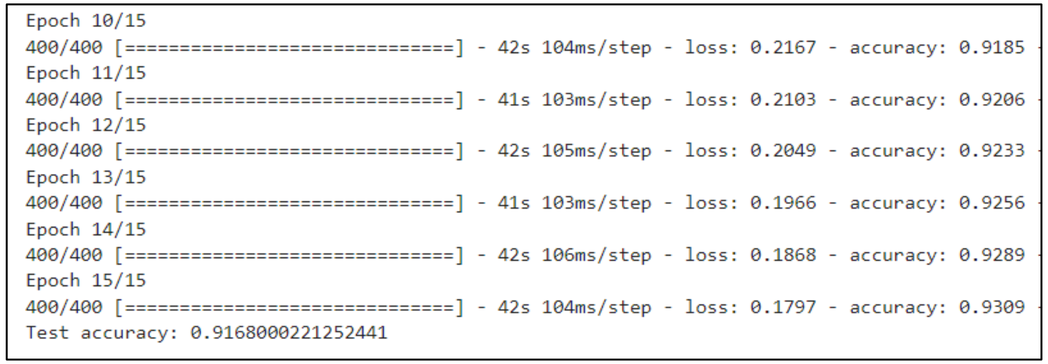
```
Epoch 10/15
400/400 [==============================] - 61s 153ms/step - loss: 0.2536 - accuracy: 0.9052
Epoch 11/15
400/400 [==============================] - 61s 152ms/step - loss: 0.2443 - accuracy: 0.9078
Epoch 12/15
400/400 [==============================] - 61s 152ms/step - loss: 0.2405 - accuracy: 0.9097
Epoch 13/15
400/400 [==============================] - 69s 174ms/step - loss: 0.2372 - accuracy: 0.9121
Epoch 14/15
400/400 [==============================] - 60s 151ms/step - loss: 0.2334 - accuracy: 0.9136
Epoch 15/15
400/400 [==============================] - 61s 153ms/step - loss: 0.2248 - accuracy: 0.9165
Test accuracy: 0.9083999991416931
```

Model in google colab

63 Average time

90.8 Test accuracy

91.6 Train Accuracy



```
Epoch 10/15
400/400 [==============================] - 42s 104ms/step - loss: 0.2167 - accuracy: 0.9185 -
Epoch 11/15
400/400 [==============================] - 41s 103ms/step - loss: 0.2103 - accuracy: 0.9206 -
Epoch 12/15
400/400 [==============================] - 42s 105ms/step - loss: 0.2049 - accuracy: 0.9233 -
Epoch 13/15
400/400 [==============================] - 41s 103ms/step - loss: 0.1966 - accuracy: 0.9256 -
Epoch 14/15
400/400 [==============================] - 42s 106ms/step - loss: 0.1868 - accuracy: 0.9289 -
Epoch 15/15
400/400 [==============================] - 42s 104ms/step - loss: 0.1797 - accuracy: 0.9309 -
Test accuracy: 0.9168000221252441
```

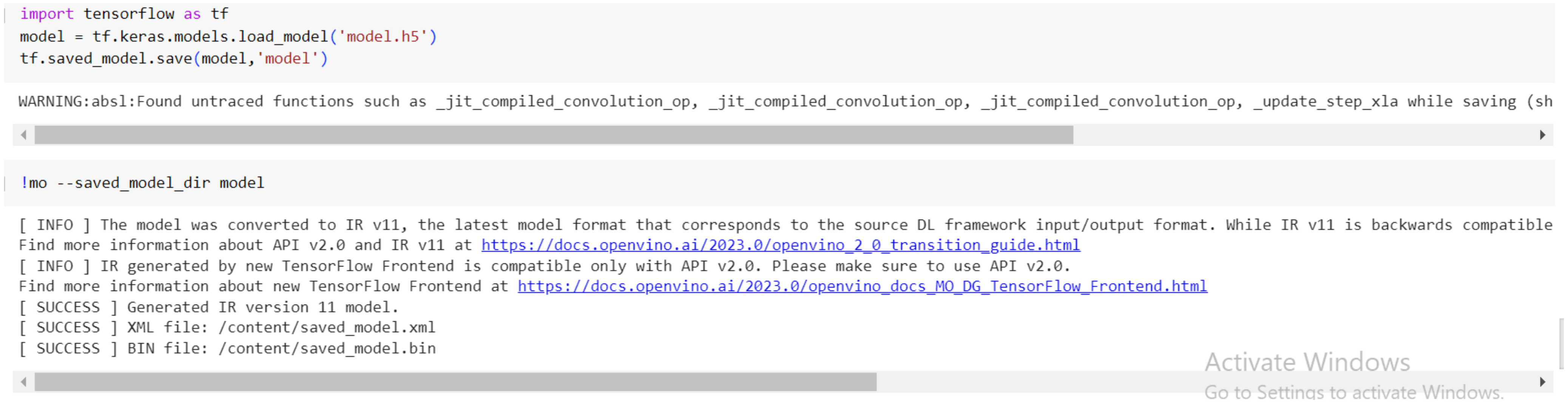
Model in Intel DevCloud

Average time 41

Test accuracy 91.7

Train accuracy 93.0

In order for generating better results in the other platform, **Intel optimization** was used in the model. Model was optimized by using openvino by using the required code and environment. The saved file was converted from .h5 format to the .xml format to perform optimization, and it was successfully done.



```python
import tensorflow as tf
model = tf.keras.models.load_model('model.h5')
tf.saved_model.save(model,'model')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _update_step_xla while saving (sh
```

```
!mo --saved_model_dir model
```

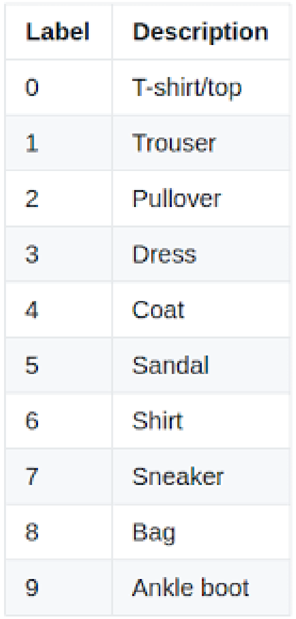
```
[ INFO ] The model was converted to IR v11, the latest model format that corresponds to the source DL framework input/output format. While IR v11 is backwards compatible
Find more information about API v2.0 and IR v11 at https://docs.openvino.ai/2023.0/openvino_2_0_transition_guide.html
[ INFO ] IR generated by new TensorFlow Frontend is compatible only with API v2.0. Please make sure to use API v2.0.
Find more information about new TensorFlow Frontend at https://docs.openvino.ai/2023.0/openvino_docs_MO_DG_TensorFlow_Frontend.html
[ SUCCESS ] Generated IR version 11 model.
[ SUCCESS ] XML file: /content/saved_model.xml
[ SUCCESS ] BIN file: /content/saved_model.bin
```

However, this operation was not supported in my current jupyter lab environment.

RESULTS

The model was compiled and trained with 15 epochs and a batch size of 150 on the Intel DevCloud platform, and results were obtained. For a better understanding about the results few of the prediction values were printed to analyze the errors.

It was possible to identify which category makes more errors in the prediction by comparing predicted values with the target data.



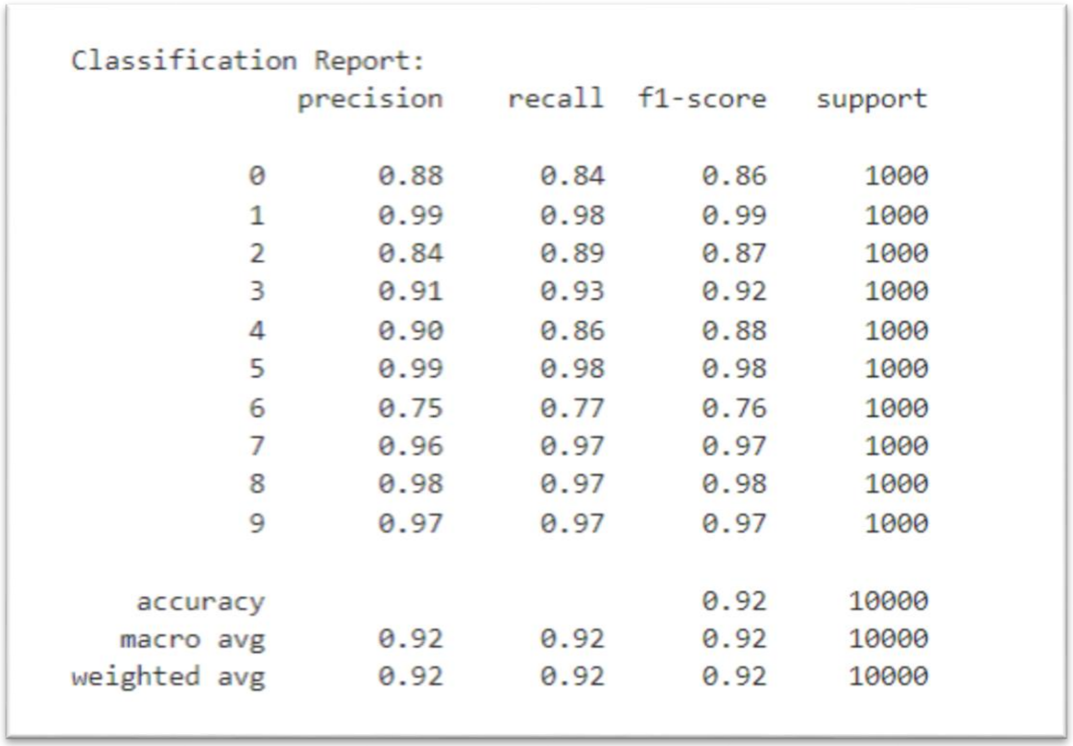
Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

The pullover(2) and the coat(4) categories are often confused by the model as it have similar structure

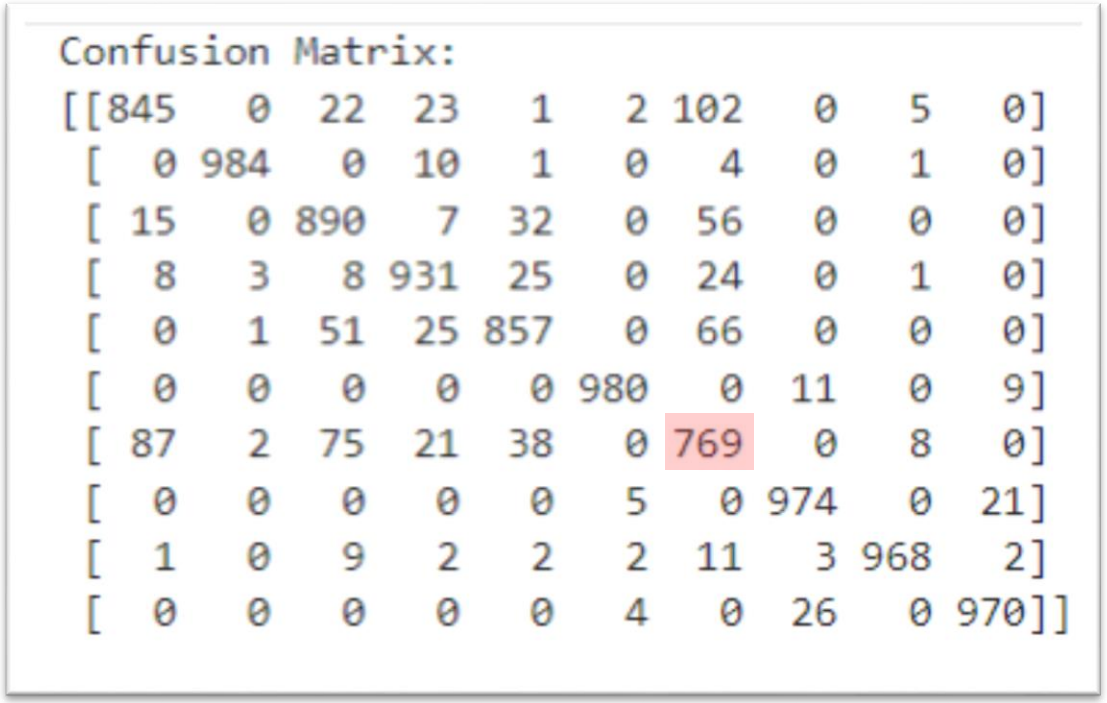
```
9 9
2 2
1 1
1 1
0 6
1 1
4 4
6 6
5 5
7 7
4 4
5 5
7 7
3 3
4 4
1 1
```

```
2 2
2 4
8 8
0 0
2 2
7 5
7 7
5 9
1 1
2 4
6 6
0 0
9 9
3 3
8 8
8 8
```

The performance of the model was evaluated on the basis of various evaluation metrics such as accuracy, precision, recall etc.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      1000
           1       0.99      0.98      0.99      1000
           2       0.84      0.89      0.87      1000
           3       0.91      0.93      0.92      1000
           4       0.90      0.86      0.88      1000
           5       0.99      0.98      0.98      1000
           6       0.75      0.77      0.76      1000
           7       0.96      0.97      0.97      1000
           8       0.98      0.97      0.98      1000
           9       0.97      0.97      0.97      1000

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```



```
Confusion Matrix:
[[845   0  22  23   1   2 102   0   5   0]
 [  0 984   0  10   1   0   4   0   1   0]
 [ 15   0 890   7  32   0  56   0   0   0]
 [  8   3   8 931  25   0  24   0   1   0]
 [  0   1  51  25 857   0  66   0   0   0]
 [  0   0   0   0   0 980   0  11   0   9]
 [ 87   2  75  21  38   0 769   0   8   0]
 [  0   0   0   0   0   5   0 974   0  21]
 [  1   0   9   2   2   2  11   3 968   2]
 [  0   0   0   0   0   4   0  26   0 970]]
```

From the confusion matrix, we can find that the class 6(Shirt) is often confused with class 0(T-shirt) Because of their similarity in structures.

CONCLUSION

In conclusion, this project on Fashion MNIST using Convolutional Neural Networks (CNNs) has highlighted the benefits of learning the problem and employing CNNs. In the process of exploring the Fashion MNIST dataset, I gained insight into computer vision and image classification that can be applied to other fields. The effectiveness of CNNs in capturing meaningful patterns and structures within the images resulted in improved accuracy. My skills in building deep learning models for image analysis have increased as a result of this project, which is useful for a variety of applications in different areas of real life. As a whole, it has expanded the knowledge in the field of computer vision and has paved the way for future endeavors into this field.

REFERENCES

https://rb.gy/vwu4x

https://rb.gy/ahreu

https://rb.gy/k8o8r

LINKS TO SOLUTION

https://github.com/Jenat14/intelunnati_Codered