VICTORIA UNIVERSITY
MELBOURNE AUSTRALIA

NIT 3251
Advanced Data Science
(Block 3, Semester 4)

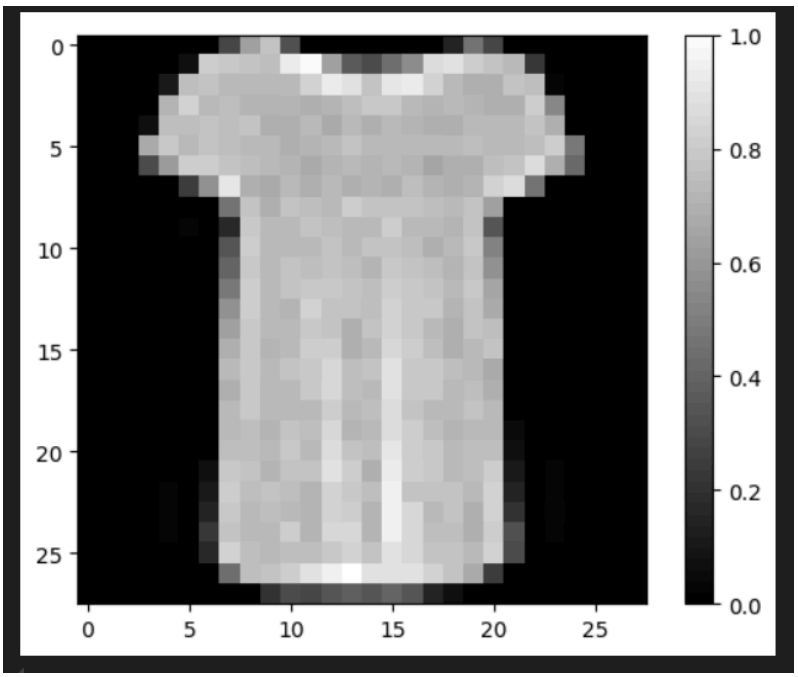
Hernandez Romero, Diego Alejandro, s8146903

Image Classification

The goal of this project was to develop a machine learning model capable of automatically categorizing images of fashion items using the Fashion MNIST dataset. This dataset contains grayscale images of 10 different clothing categories, including items like T-shirts, dresses, and sneakers. The primary objective was to build a Convolutional Neural Network (CNN) that could classify these images into their respective categories and evaluate its performance.

The first step involved loading and preprocessing the Fashion MNIST dataset. The images were provided in CSV format, so they were converted into NumPy arrays. Each image was reshaped to a 28x28 grid and normalized by scaling the pixel values between 0 and 1 to prepare them for model training. Additionally, the labels were one-hot encoded, representing the 10 distinct clothing categories. This preprocessing was crucial to ensure that the images were formatted correctly for the CNN and to enable efficient training.

```
data_train = pd.read_csv('fashion-mnist_train.csv')
data_test = pd.read_csv('fashion-mnist_test.csv')

img_rows, img_cols = 28, 28
input_shape = (img_rows, img_cols, 1)

X = np.array(data_train.iloc[:, 1:])
y = to_categorical(np.array(data_train.iloc[:, 0]))


X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=13)

#Test data
X_test = np.array(data_test.iloc[:, 1:])
y_test = to_categorical(np.array(data_test.iloc[:, 0]))



X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
X_val = X_val.reshape(X_val.shape[0], img_rows, img_cols, 1)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_val = X_val.astype('float32')
X_train /= 255
X_test /= 255
X_val /= 255
```



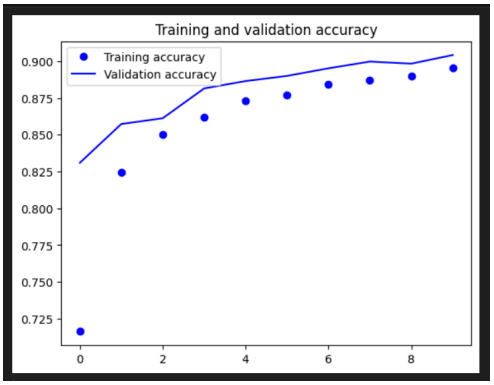
A CNN model was then built using the Keras library. The model was designed with three convolutional layers, each followed by max-pooling layers to reduce the spatial dimensions of the feature maps, and dropout layers to prevent overfitting. The convolutional layers utilized ReLU activation to introduce non-linearity, and the model concluded with a fully connected (Dense) layer containing 128 neurons. Finally, a softmax activation layer was used to output a probability distribution across the 10 clothing categories. The model was compiled using the Adam optimizer and categorical cross-entropy loss function.

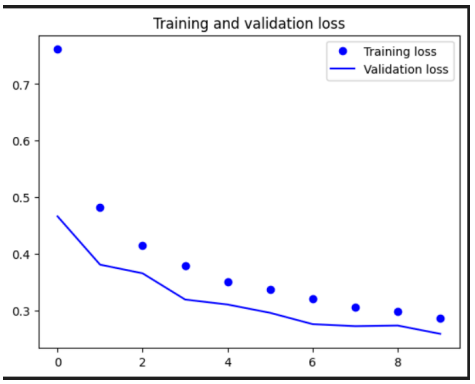
```
img_rows, img_cols = 28, 28

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 kernel_initializer='he_normal',
                 input_shape=input_shape))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(10, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])

model.summary()
```

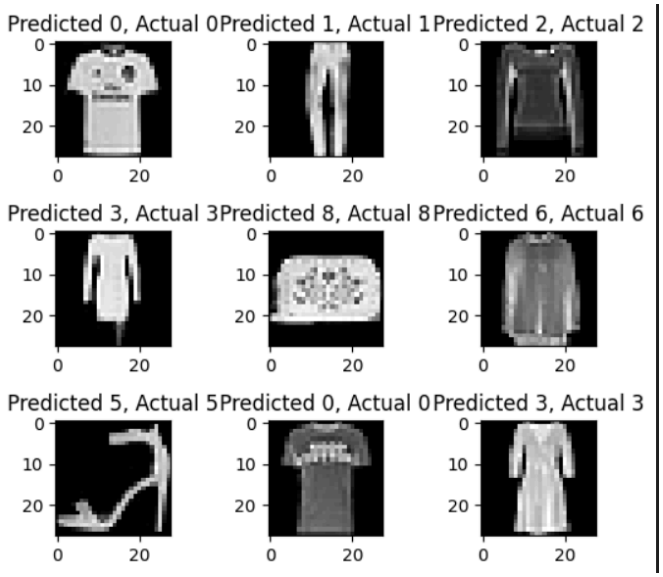
Once the model architecture was established, it was trained on the Fashion MNIST dataset. The data was split into training and validation sets, with 80% of the data used for training and 20% for validation. The model was trained over 10 epochs, and the training history was monitored through accuracy and loss curves. As the training progressed, both the training and validation accuracies steadily improved, while the loss function converged, indicating successful learning. After training, the model was evaluated on a separate test set, achieving satisfactory accuracy, demonstrating its ability to generalize to unseen data.

```
history = model.fit(X_train, y_train,
                    batch_size=128,
                    epochs=10,
                    verbose=1,
                    validation_data=(X_val, y_val))
score = model.evaluate(X_test, y_test, verbose=0)
```

While the model performed well, there were some challenges faced during the process. Preprocessing the data, particularly ensuring consistent formatting and normalization, was key to ensuring accurate predictions. Balancing the model complexity to avoid overfitting while maintaining high accuracy required careful use of dropout layers and tuning of hyperparameters. Additionally, the Fashion MNIST dataset, with its relatively small size and low-resolution images, limited the model's potential performance, and more advanced techniques could be explored for better accuracy in future work.







In conclusion, the CNN model successfully achieved the task of classifying clothing images from the Fashion MNIST dataset. By effectively preprocessing the data, designing a robust CNN, and carefully evaluating the model, the project demonstrated the power of deep learning in automating fashion categorization. Further improvements could include experimenting with more complex architectures or using transfer learning on larger datasets to enhance the model's accuracy and generalization.

Movie Recommendation

In this project, I developed a movie recommendation system using the MovieLens 100k dataset and cosine similarity to measure both user and movie similarities. The process began by loading the dataset into a Pandas DataFrame and examining its structure, such as identifying the number of users and movies. A user-movie rating matrix was created where rows represented users and columns represented movies, with the matrix values corresponding to the ratings given by users.

```
header = ['user_id', 'item_id', 'rating', 'timestamp']
movielens_data = pd.read_csv('ml-100k/u.data', sep='\t', names=header)
movielens_data.head()
```

✓ 0.0s

	user_id	item_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

```
n_users, n_movies = movielens_data['user_id'].nunique(), movielens_data['item_id'].nunique()
n_users, n_movies
```

To generate recommendations, I used the cosine similarity metric. By computing pairwise cosine distances between users and between movies, we were able to gauge how similar they were. These distances were converted into similarity scores, which helped identify the closest matches. Additionally, movie IDs from the dataset were mapped to their corresponding titles to make the system user-friendly and enable interaction using movie names instead of numerical IDs.

```
train_data_matrix = np.zeros((n_users, n_movies))

for line in movielens_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1] = line[3]

train_data_matrix.shape
```

```
user_distances = pairwise_distances(train_data_matrix, metric="cosine")

# ".T" below is to transpose our 2D matrix.
train_data_matrix_transpose = train_data_matrix.T
movie_distances = pairwise_distances(train_data_matrix_transpose, metric="cosine")

user_distances.shape, movie_distances.shape
```

```
user_similarity = 1 - user_distances
movie_similarity = 1 - movie_distances
```

```
idx_to_movie = {}

with open('ml-100k/u.item', 'r', encoding="ISO-8859-1") as f:
    for line in f.readlines():
        info = line.split('|')
        idx_to_movie[int(info[0])-1] = info[1]

movie_to_idx = {v: k for k, v in idx_to_movie.items()}
```

The core functionality of the system revolved around recommending similar movies based on a user's favorite film. For example, by selecting "Richard III (1995)" as a reference, the system could return a list of similar movies using the calculated cosine similarity values. This approach worked well for generating recommendations, showing the potential of cosine similarity for this type of task.

```
favorite_movie_name = 'Richard III (1995)'
movie_index = movie_to_idx[favorite_movie_name]
movie_index
```

✓ 0.0s

```
9
```

```
how_much_movie_to_show = 7

movies = top_k_movies(movie_similarity, idx_to_movie, movie_index, k = how_much_movie_to_show)
movies[1:how_much_movie_to_show + 1]
```

✓ 0.0s

```
['Henry V (1989)',
 'Angels and Insects (1995)',
 'Piano, The (1993)',
 'Madness of King George, The (1994)',
 'Othello (1995)',
 'Much Ado About Nothing (1993)',
 'Sense and Sensibility (1995)']
```

However, some challenges were encountered during the process.Mainly that this is not a Tensorflow-based model, it's just a simple recommendations system where the user-movie matrix was sparse, meaning many ratings were missing, which likely affected the accuracy of the similarity calculations. Additionally, the computational cost of calculating pairwise distances increased as the number of users and movies grew, which could limit scalability. Finally, the cold start problem emerged as a limitation, as users who had rated only a few movies lacked sufficient data for generating high-quality recommendations.

In conclusion, while cosine similarity proved to be an effective method for movie recommendations, issues related to data sparsity, computational cost, and limited user data posed challenges. Despite these limitations, the system successfully demonstrated the potential for recommending movies based on similarity metrics.

Music Genre Classification

This project focused on classifying audio files into different musical genres using a neural network. The process began with data preprocessing, where audio data was loaded and analyzed using the librosa library to extract relevant features. In addition to using raw audio data, a CSV file containing pre-extracted features was employed. The filename column in the dataset was dropped as it was not relevant for training. The target labels, representing different genres, were encoded into numerical values using LabelEncoder, preparing them for input into the neural network model.

```
audio_data = 'Data/genres_original/rock/rock.00010.wav'
data , sr = librosa.load(audio_data)
print(type(data), type(sr))
```

✓ 2.9s

```
df1 = pd.read_csv('Data/features_3_sec.csv')
df1.head()
```
✓ 0.1s Python

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521	1773.065032	167541.630869	1972.744388	
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450	1816.693777	90525.690866	2010.051501	
2	blues.00000.2.wav	66149	0.346815	0.092243	0.132003	0.004620	1788.539719	111407.437613	2084.565132	
3	blues.00000.3.wav	66149	0.363639	0.086856	0.132565	0.002448	1655.289045	111952.284517	1960.039988	
4	blues.00000.4.wav	66149	0.335579	0.088129	0.143289	0.001701	1630.656199	79667.267654	1948.503884	

```
df1 = df1.drop(labels='filename',axis=1)
```
✓ 0.0s

```
genre_list = df1.iloc[:, -1]
encoder = LabelEncoder()
```
✓ 0.0s

```
y = encoder.fit_transform(genre_list)
```
✓ 0.0s

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(np.array(df1.iloc[:, :-1], dtype = float))
```

Feature scaling was an important step, as it ensured that all features had a consistent scale, preventing certain features from disproportionately influencing the model. A Standarscaler was used to standardize the features. The dataset was then split into training (67%) and testing (33%), allowing for effective evaluation of the model's performance on unseen data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```
✓ 0.0s

The neural network model was constructed using Keras.It consisted of three hidden layers with 256, 128, and 64 units, respectively, all utilizing the ReLU activation function. The output layer had 10 units (one for each genre), using the softmax activation function to classify the audio files. The model was trained for 100 epochs with a batch size of 128,after training, the model was evaluated on the test set, with both loss and accuracy metrics calculated.

```
# Neural network
model = Sequential()
model.add(layers.Dense(256, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
classifier = model.fit(X_train,
                       y_train,
                       epochs=100,
                       batch_size=128)
```

✓ 9.9s

The model achieved reasonable accuracy on the audio genre classification task. Standardizing the features proved essential for stable and efficient training. Although the architecture of the network was effective.

```
test_loss, test_acc  = model.evaluate(X_test, y_test, batch_size=128)
```

✓ 0.1s

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 958us/step - accuracy: 0.8937 - loss: 0.5706
```

```
print("The test loss is :",test_loss, "\nThe test accuracy is :",test_acc)
```

✓ 0.0s

```
The test loss is : 0.6072977185249329
The test accuracy is : 0.887776792049408
```

In conclusion, the project demonstrated that even a simple feedforward neural network could perform reasonably well in classifying audio genres based on pre-extracted features. However, there is room for improvement. Exploring more complex model architectures, such as convolutional neural networks or recurrent networks tailored to audio data, as well as further experimentation with hyperparameters and data augmentation, could lead to enhanced model performance.