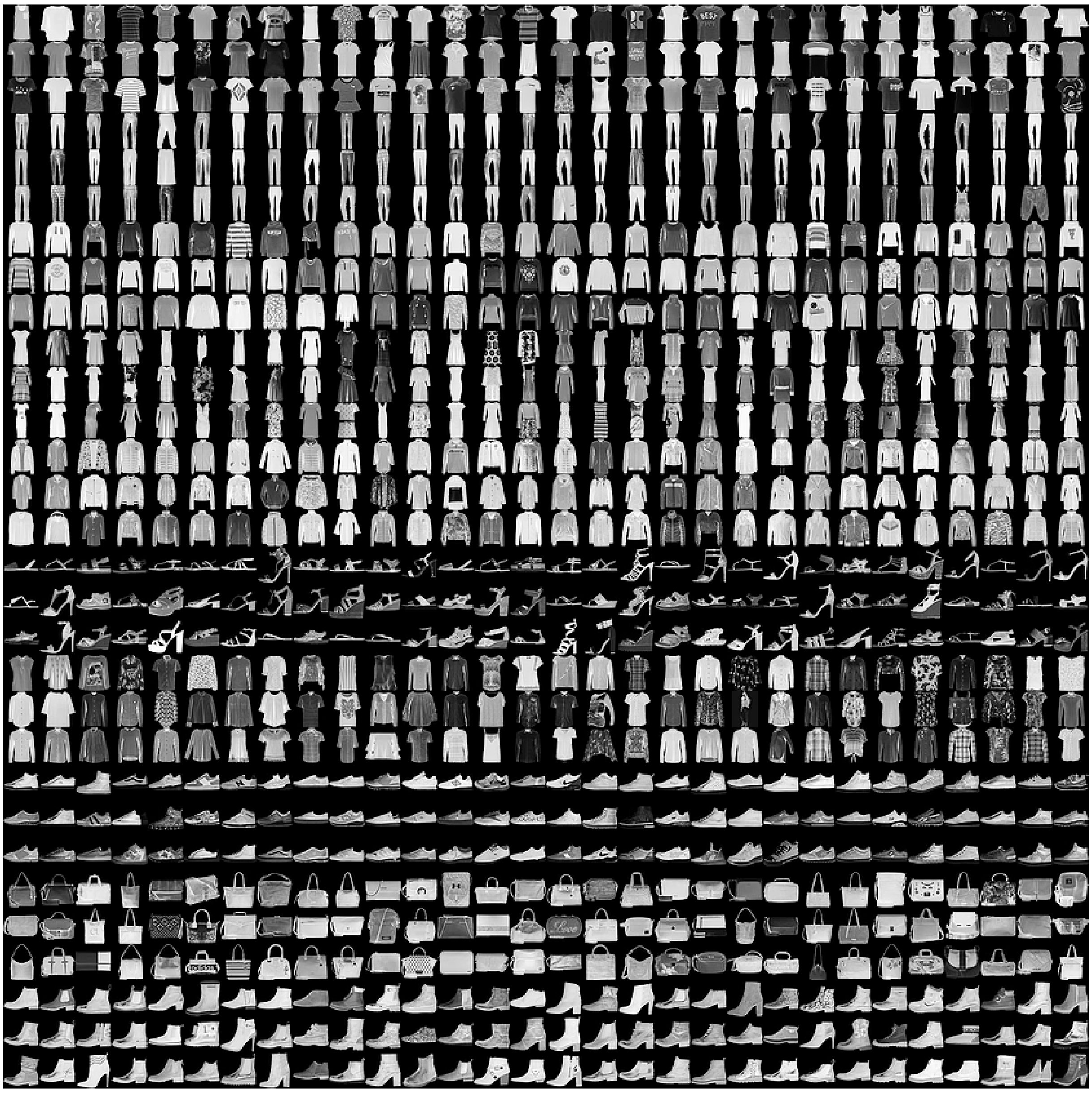
Report for Fashion MNIST Classification



Abstract:

In this technical report, we explore the enhancement of a Convolutional Neural Network (CNN) for the Fashion MNIST dataset, aiming to optimise classification accuracy. We delve into the initial model that exhibited overfitting, followed by systematic refinements, learning rate scheduling, and data augmentation. These strategic adjustments mitigated overfitting and improved model performance, culminating in a notable test accuracy of 91.3%.

The report discusses the challenges encountered, the solutions implemented, and the comprehensive methodology that guided the successful model improvement.

Introduction:

Our project commenced with a basic CNN architecture, designed for image classification on the Fashion MNIST dataset. This model includes multiple layers, such as Conv2D for extracting features from images, MaxPooling2D to reduce the spatial dimensions, and Dense layers for decision-making. BatchNormalization is employed to normalise the input of each layer, aiming to improve training efficiency and stability. The model is compiled with a specific learning rate and momentum to optimise the training process, and its fit on the training data while validating on a separate validation set to monitor and avoid overfitting. The results are visualised to evaluate the training and validation accuracy and loss, providing insight into the model's learning performance.

Methodology:

Our methodology was systematic and iterative. Initially, we loaded and preprocessed the Fashion MNIST dataset, ensuring it was in a format compatible with our CNN. This involved reshaping and normalising the images and encoding the labels.

Initially, we designed a basic CNN model as a baseline for the Fashion MNIST dataset. Identifying overfitting as a key challenge, we focused on stabilising the learning process. Our approach included implementing batch normalisation following each convolutional layer to normalise inputs within the network, facilitating faster convergence and reducing sensitivity to initial weights.

We adopted a consistent learning rate throughout the training process, using Stochastic Gradient Descent (SGD) with momentum. This optimiser was effective in navigating the loss landscape more efficiently than standard gradient descent.

Additionally, data augmentation played a crucial role. Our CNN model leveraged various data augmentation techniques to enhance generalisation and performance. Utilizing 'ImageDataGenerator' from TensorFlow's Keras API, we introduced diverse transformations to the training images. These included a rotation range of 6 degrees, width and height shifts of 5% (0.05), a shear range of 5%, and a zoom range of 5%. Additionally, we enabled horizontal flipping of images. These augmentations created a rich variety of training samples, simulating different orientations and perspectives, thus making our model more robust against variations in new, unseen data.

Initial Model and Challenges:

The initial CNN model, designed for Fashion MNIST classification, was structured with essential Conv2D layers for feature extraction, complemented by Batch Normalisation and MaxPooling2D layers. These layers were effective in pattern recognition but led to overfitting. The model's high training performance contrasted with its limited ability to generalise to unseen data, indicating a complexity that surpassed the dataset's scope.

Final Model:
The final model shifted focus from layer addition to optimising the training process. Key strategies included:

> **Batch Normalisation Continuation**: Preserving batch normalisation across Conv2D layers ensured consistent training experiences across different batches, contributing to a stable learning environment.
>
> **Learning Rate Adjustment**: A schedule for learning rate reduction was implemented, facilitating gradual, more precise tuning across 100 epochs. This approach allowed for initial broad learning adjustments, with refinement as the model progressed, enhancing its ability to find an optimal solution.
>
> **Data Augmentation Emphasis**: The use of ImageDataGenerator for varying training data presented a broader spectrum of input scenarios, critical for handling the diverse Fashion MNIST dataset. This technique bolstered the model's generalisation capability.

In essence, the transition from the initial to the final model marked a shift from adding complexity through layers to refining the training regimen and data handling strategies. This pivot played a crucial role in overcoming the initial overfitting challenge and achieving a more robust and accurate model.

Refinements for Improvement:

In our journey to refine the CNN model's performance, we focused on techniques to mitigate overfitting and enhance accuracy.

Batch Normalisation Integration: We incorporated batch normalisation layers following each convolutional layer. This approach normalised the inputs within the network, leading to faster convergence and reduced sensitivity to initial network weights. It played a crucial role in stabilising the learning process, contributing significantly to the model's robustness.

Data Augmentation Strategy: Recognising the diversity of the Fashion MNIST dataset, we employed a comprehensive data augmentation strategy using ImageDataGenerator. This included adjustments in rotation, width shift, height shift, shear range, and zoom range, as well as horizontal flips. These augmentations introduced a wide range of variations in the training data, enhancing the model's ability to generalise to new, unseen data.

Learning Rate and Optimizer Choice: Contrary to an adaptive learning rate schedule, our model used a consistent learning rate throughout the training process. The optimiser chosen was Stochastic Gradient Descent (SGD) with momentum, known for its effectiveness in navigating complex loss landscapes and accelerating convergence.
These strategic decisions, especially the focus on batch normalisation and data augmentation, were key in achieving the final model's test accuracy of approximately 91.3%, successfully addressing overfitting challenges and enhancing generalisation capabilities.

Results:

The optimised CNN, through its final configuration, achieved a remarkable test accuracy of 91.3%. This significant enhancement from the initial baseline testifies to the efficacy of the chosen model architecture and training strategies.

Key to this success was the integration of batch normalisation in each convolutional layer, as seen in the code. This technique, indicated by *'tf.keras.layers.BatchNormalization'*, played a pivotal role in stabilising the learning process. It ensured uniform updates across training batches, contributing to the model's efficiency and stability.

The data augmentation techniques, implemented via *'ImageDataGenerator'*, were instrumental in introducing diverse variations to the training dataset. Adjustments like rotation range and width shifts, precisely coded, significantly broadened the model's exposure to varied image representations. This diversity was crucial in enhancing the model's ability to generalise to new, unseen data, a factor particularly important for the diverse Fashion MNIST dataset.

Furthermore, the use of a consistent learning rate in the SGD optimiser, as opposed to a dynamic schedule, demonstrated its effectiveness in the learning process. The model's training and validation loss metrics showed a consistent decrease, and the accuracy metrics exhibited an upward trend in both the training and validation phases. These metrics reflect the model's ability not just to memorise the training data, but to effectively generalise, affirming the success of our strategies in combating overfitting and enhancing model robustness.

This approach underscores the importance of a comprehensive strategy in CNN optimisation for image classification tasks, balancing model complexity with computational efficiency and data representation to achieve high generalisation capabilities.

Discussion:

Our CNN model's performance enhancement is a testament to the effectiveness of batch normalisation and data augmentation. The incorporation of *'tf.keras.layers.BatchNormalization'* played a crucial role in stabilising the learning process, leading to more efficient and stable training. This, combined with 'ImageDataGenerator' for data augmentation, introduced vital variability in the training data for a diverse dataset like Fashion MNIST. Image transformations such as rotations, shifts, and flips significantly improved our model's generalisation capabilities.

We employed a consistent learning rate throughout the model's training, using *'tf.keras.optimizers.SGD'* with a fixed rate of 0.01 and momentum of 0.9. This approach ensured steady and controlled learning, contributing to the model's effective convergence and robust performance.

The combination of these strategies not only boosted the model's accuracy but also underscored the importance of a comprehensive approach to CNN optimisation for robust image classification tasks.

Cell-by-Cell Report:

Cell 1 - Setup and Importing Libraries:

This cell sets up the environment by importing necessary libraries. numpy for numerical operations, gzip for data extraction, TensorFlow and Keras for building the neural network and plotting libraries for visualization.

```
import numpy as np
import gzip
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
```

Cell 2 - Data Loading Function:

A function is defined to load the Fashion MNIST dataset. It reads label and image files, converting them into a format suitable for the neural network.

```
def load_mnist(path, kind='train'):
    labels_path = f'{path}/{kind}-labels-idx1-ubyte.gz'
    images_path = f'{path}/{kind}-images-idx3-ubyte.gz'

    with gzip.open(labels_path, 'rb') as lbpath:
        labels = np.frombuffer(lbpath.read(), dtype=np.uint8, offset=8)

    with gzip.open(images_path, 'rb') as imgpath:
        images = np.frombuffer(imgpath.read(), dtype=np.uint8, offset=16).reshape(len(labels), 784)

    return images, labels
train_images, train_labels = load_mnist('.', kind='train')
test_images, test_labels = load_mnist('.', kind='t10k')
```

Cell 3 - Label Encoding:

Labels are converted to a one-hot encoded format, making them compatible with the categorical_crossentropy loss function used later in model training.

```
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

Cell 4 - Image Processing:

The images are reshaped to fit the CNN input requirements and normalised to improve the training process.

```
train_images = train_images.reshape((60000, 28, 28, 1)) / 255.0
test_images = test_images.reshape((10000, 28, 28, 1)) / 255.0
```

Cell 5 - Data Splitting:

The dataset is divided into training and validation sets using train_test_split from sklearn. This is crucial for evaluating the model's performance on unseen data.

```
from sklearn.model_selection import train_test_split
# Splitting training data into training and validation sets
```

```
X_train, X_val, y_train, y_val = train_test_split(train_images, train_labels, test_size=0.2,
random_state=42)
```

Cell 6 - Data Augmentation:

ImageDataGenerator is used for augmenting the training images, introducing variations like rotation and shifting. This helps the model generalise better.

```
datagen = ImageDataGenerator(
    rotation_range=6,
    width_shift_range=0.05,
    height_shift_range=0.05,
    shear_range=0.05,
    zoom_range=0.05,
    horizontal_flip=True
)
```

Cell 7 - Model Building and Compilation:

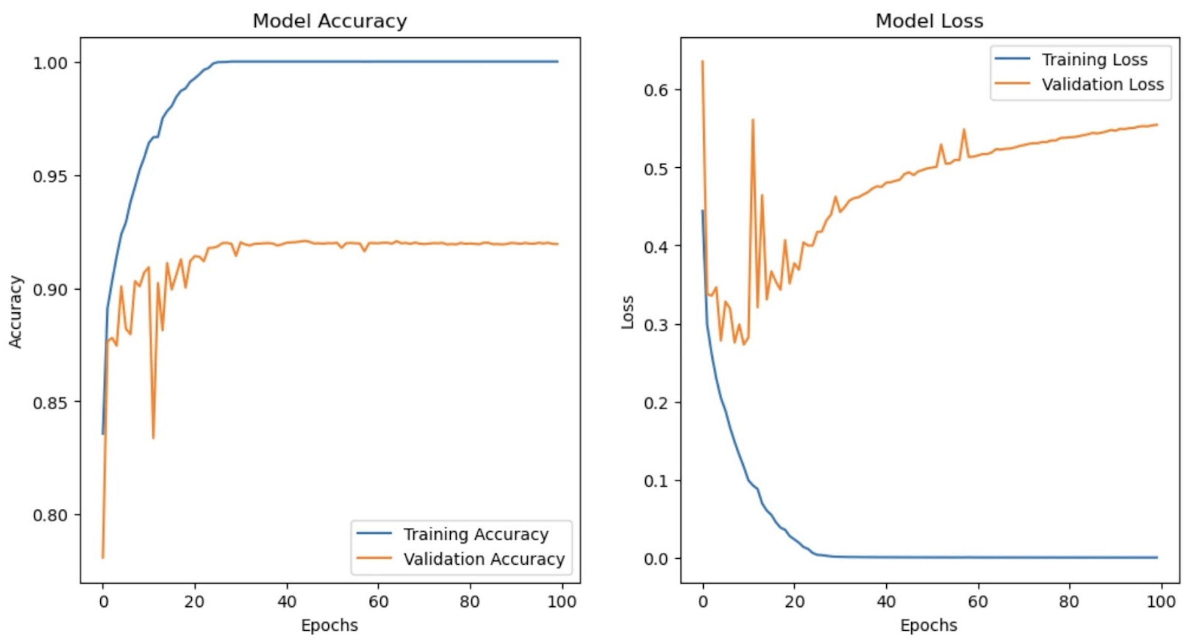
A CNN model is defined and compiled. The architecture includes convolutional, batch normalisation, and max-pooling layers, followed by flattening and dense layers. The model is compiled with SGD optimiser and categorical cross-entropy loss.

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer=tf.keras.optimizers.legacy.SGD(learning_rate=0.01, momentum=0.9),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=128,
    validation_data=(X_val, y_val)
)
```

Cell 8 - Model Training Visualisation:

Training and validation accuracy and loss are plotted to assess the model's learning progress.

```
# Plot for accuracy
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)  # 1 row, 2 columns, first plot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
# Plot for loss
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second plot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



The training visualisation suggests a well-fitted model with both training and validation accuracy trends converging to a high level, which indicates good generalisability on unseen data. The training accuracy starts higher and remains stable, while validation accuracy catches up as epochs increase, showing that the model is learning effectively without overfitting. The loss plots show a rapid decrease in training loss, levelling off as epochs go by. Validation loss decreases alongside training loss but with some fluctuations, which is normal in validation due to its variability. Overall, the model appears to generalise well with consistent accuracy and controlled loss.

Cell 9 - Model Evaluation and Saving:

The final cell evaluates the model's performance on the test dataset and saves the trained model for future use.

```
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f'Test loss: {test_loss}, Test accuracy: {test_accuracy}')
model.save('fashion_mnist_model.keras')
```

output:

Test loss: 0.6320658922195435, **Test accuracy**: 0.9129999876022339

Conclusion:

In conclusion, our refined CNN model for the Fashion MNIST dataset illustrates the power of combining meticulous network architecture with targeted training strategies. By achieving a test accuracy of 91.3%, the model underscores the value of batch normalisation in stabilizing the learning process, the use of a consistent learning rate with SGD for efficient optimization, and the application of data augmentation techniques for enhanced generalisation. These elements collaboratively contributed to the model's robust performance, demonstrating their collective importance in successful machine-learning applications.