Fashion MNIST Image Classification Project Report

1. Data Preprocessing and Feature Engineering

Steps taken are:

1) Dataset Selection:
   We used the Fashion MNIST dataset, which offers 28×28 grayscale images across 10 classes. This dataset is simple yet effective for benchmarking classification models.

2) Preprocessing:
   a) Normalization: Convert image pixel values to tensors and scale them to a range of [-1, 1]. This standardizes input data and improves model convergence.
   b) Data Augmentation: Apply random horizontal flips and small rotations to improve model generalization.
   c) Dataset Splitting: Divide the data into training (80%), validation (20%), and test sets to ensure reliable evaluation.

Key Code Snippet:

```
import torchvision.transforms as transforms
from torchvision.datasets import FashionMNIST
from torch.utils.data import random_split, DataLoader

# Define transformations for training and testing
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Load the dataset
train_dataset_full = FashionMNIST(root='./data', train=True,
transform=transform_train, download=True)
test_dataset = FashionMNIST(root='./data', train=False,
transform=transform_test, download=True)

# Split training dataset into training and validation sets (80/20 split)
train_size = int(0.8   len(train_dataset_full))
val_size = len(train_dataset_full) - train_size
train_dataset, val_dataset = random_split(train_dataset_full,
[train_size, val_size])

# DataLoaders for batching
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)
```

Justification: These preprocessing steps ensure that the model sees varied and normalized inputs, which is critical for achieving stable and high performance during training.

2. Model Selection and Optimization Approach

Our Model Choice:

1) Architecture:
   We selected a modified ResNet18 because it offers a good balance between performance and computational efficiency. Given that Fashion MNIST images are grayscale, we adjusted the first convolution layer to accept one channel and modified the final fully connected layer for 10 classes.

2) Optimization:
   a) Loss Function: CrossEntropyLoss for multi-class classification.
   b) Optimizer: Adam with a learning rate of 0.001 for efficient convergence.
   c) Learning Rate Scheduler: ReduceLROnPlateau to adjust the learning rate based on validation loss.
   d) Early Stopping: To prevent overfitting, training stops when validation loss does not improve for a defined number of epochs.

Key Code Snippet:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import in models

# Modify ResNet18 for grayscale images and 10 output classes
model = models.resnet18(pretrained=False)
model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3,
bias=False)
model.fc = nn.Linear(model.fc.in_features, 10)
```

```
model = model.to(device)

# Define loss function, optimizer, and learning rate scheduler
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',
factor=0.5, patience=2, verbose=True)
```

Justification: ResNet18 is well-suited for this task due to its proven architecture in image classification. The optimization choices (Adam, LR scheduler, and early stopping) ensure efficient training and help mitigate overfitting.

3. Deployment Strategy and API Usage Guide

Our Deployment Approach:

1) API Framework:
   We used FastAPI to build a REST API with a `/predict` endpoint. FastAPI was chosen for its speed, ease of use, and built-in support for asynchronous operations.

2) Endpoint Design:
   a) /predict: Accepts an image file (JPEG/PNG), preprocesses it (grayscale conversion, resizing, normalization), and returns the predicted Fashion MNIST class.
   b) Security: Implements basic HTTP authentication to restrict access. (<u>username and password are kept as ‘user’ and ‘pass’ for this submission</u>)

3) Containerization:
   A Dockerfile is included to create a consistent, portable environment. This facilitates deployment across different platforms (AWS, GCP, etc.).

Key Code Snippet (FastAPI Endpoint):

```
from fastapi import FastAPI, File, UploadFile, HTTPException, Depends
from fastapi.security import HTTPBasic, HTTPBasicCredentials
from PIL import Image
import io, torch
from torchvision import transforms

app = FastAPI()
security = HTTPBasic()

def get_current_username(credentials: HTTPBasicCredentials =
Depends(security)):
    if credentials.username != "user" or credentials.password != "pass":
```

```python
        raise HTTPException(status_code=401, detail="Unauthorized")
    return credentials.username

# Preprocessing transformation for inference
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Endpoint for prediction
@app.post("/predict")
async def predict(file: UploadFile = File(...), username: str =
Depends(get_current_username)):
    if file.content_type not in ["image/jpeg", "image/png"]:
        raise HTTPException(status_code=400, detail="Unsupported file
type")
    try:
        image = Image.open(io.BytesIO(await file.read()))
        image = transform(image).unsqueeze(0).to(device)
        with torch.no_grad():
            outputs = model(image)
            predicted = torch.argmax(outputs, dim=1).item()
        class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
'Coat',
                       'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle
boot']
        return {"predicted_class": class_names[predicted]}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Key Dockerfile Snippet:

```dockerfile
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Justification:

1) FastAPI offers a clean, high-performance platform for deploying machine learning models.
2) Basic authentication secures the endpoint without overcomplicating the deployment.
3) Docker ensures the application runs consistently in any environment.

4. Conclusion

Our provided approach focuses on both clarity and efficiency. The preprocessing ensures better generalization, the modified ResNet18 provides robust performance as it is coupled with thoughtful optimization, the FastAPI deployment strategy makes the model accessible and secure, and the Docker implementation allows for easier deployment. We hope that the decisions made fulfill the requirements for this assessment.