PAPER REPORT - MACHINE LEARNING PROJECT 2

Student: Imbrea Giulia Stefania-343C4

1.Introduction

- **Purpose of the Report**: To analyze and compare the performance of various neural network models on the Fashion-MNIST and Fruits-360 datasets.
- **Overview of Approaches**: MLP on extracted features, MLP on raw images, convolutional network, and fine-tuning with ResNet-18.

---

2. Dataset Description

- **Fashion-MNIST** is a dataset of 70,000 grayscale images, each sized 28x28 pixels, categorized into 10 classes representing different clothing items.
- **Fruits-360** is a dataset containing approximately 55,000 images, each representing a single fruit. The images are in RGB color format, and the dataset includes 121 fruit classes

---

3. Model Descriptions

3.1. MLP on Extracted Features

Model Architecture:

- Input layer: it matches the number of selected features
  - 64 for Fashion-MNIST
  - 128 for Fruits-360
- Two hidden layers: Each with 512 neurons and ReLU activation.
- **Dropout**: Applied with a rate of 0.3 to prevent overfitting.
- **Output Layer**: <number_of_classes> neurons with softmax activation for classification.
  - where <number_of_classes> is 10 for Fashion-MNIST and 121 for Fruits-360

```
mlp = Sequential([
    Dense(512, input_shape=(X_train_selected.shape[1],)),  # Input shape matches selected features
    Activation('relu'),                                    # ReLU activation
    Dropout(0.3),                                          # Dropout for regularization
    Dense(512),                                            # Hidden layer with 512 units
    Activation('relu'),                                    # ReLU activation
    Dropout(0.3),                                          # Dropout for regularization
    Dense(num_classes),                                    # Output layer with 10 units (num_classes)
    Activation('softmax')                                  # Softmax activation for probabilities
])
```

Figure 1: MLP Architecture for Extracted Features

Training Process :

- **Compilation Details**:
  - **Loss Function**: `categorical_crossentropy`, chosen for its suitability in multi-class classification tasks.
  - **Optimizer**: `adam`, which provides adaptive learning rates for efficient convergence.
  - **Metric**: Accuracy, to monitor the model's performance during training.

```
mlp.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 2: MLP Model Compilation with Adam Optimizer and Categorical Crossentropy Loss

- Training Configuration
  - Batch size: 128
  - Number of epochs: 25
  - Validation data (20%) used during training for real-time monitoring of model generalization.

```
start_time = time.time()
history_mlp = mlp.fit(
    X_train_small,                # 80% X_train_normalized
    y_train_small_one_hot,
    batch_size=128,
    epochs=25,
    verbose=1,
    validation_data=(X_val, y_val_one_hot)
)
training_time = time.time() - start_time
print(f"Training Time: {training_time:.2f} seconds")
```

Figure 3: Training Process for MLP on Extracted Features

Performance Metrics :

	Test Accuracy	Avg Precision	Avg recall	Avg f1-score	Training Time
Fashion-MNIST	0.8773	0.8768	0.8774	0.8766	42.62 s
Fruits-360	0.8802	0.8818	0.8778	0.8739	81.50 s

Graphs:

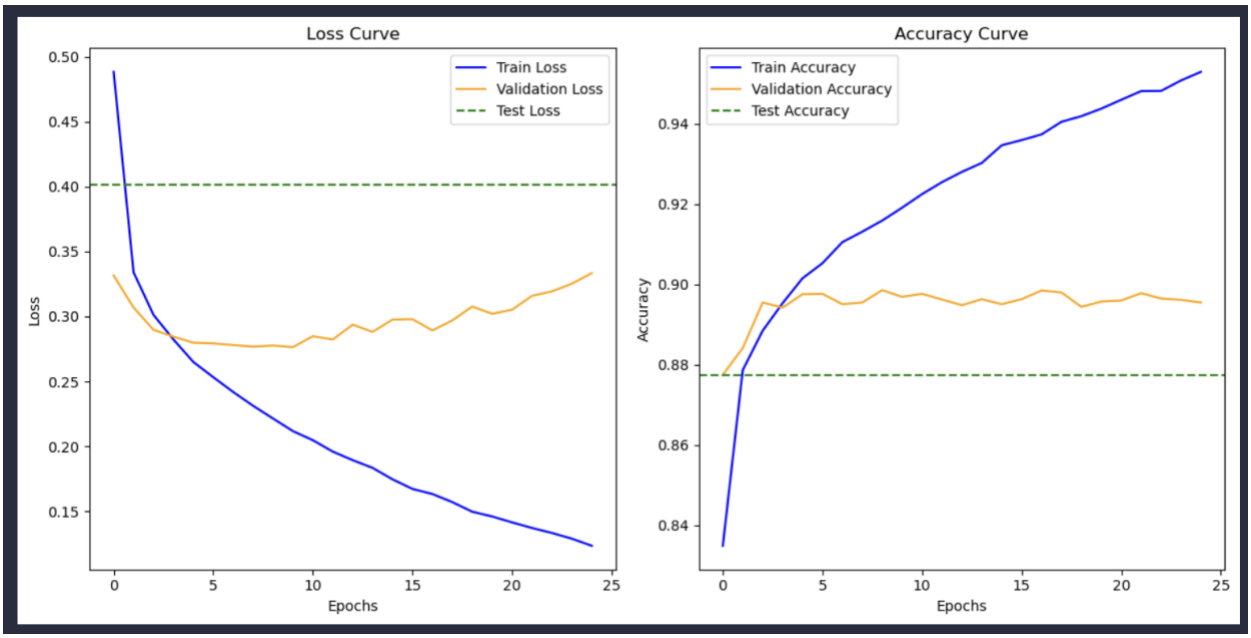

Figure 4: Loss and Accuracy curves for MLP on Extracted Features ***(Fashion-MNIST)***

- The validation loss and accuracy curves indicate that the model is moderately effective but begins to overfit after a certain point.
- Strategies such as early stopping, increasing dropout rates, or introducing regularization could help mitigate overfitting.
- Despite the overfitting signs, the final validation accuracy (**88%**) and stable loss demonstrate the MLP's ability to generalize reasonably well to unseen Fashion-MNIST data.

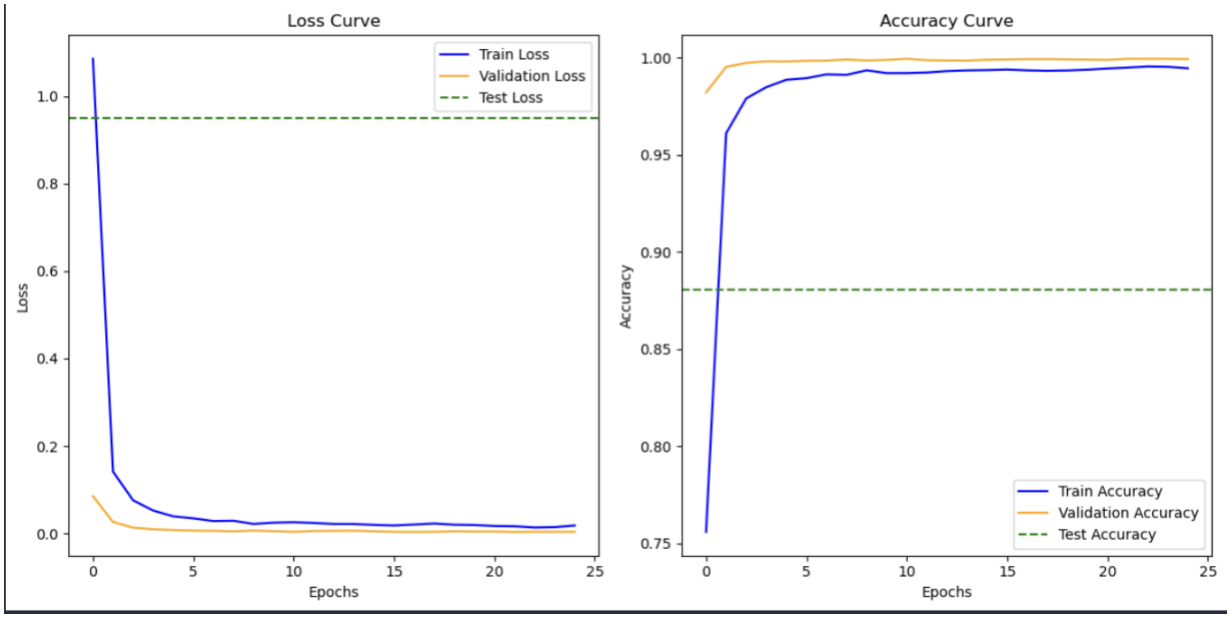

Figure 5: Loss and Accuracy curves for MLP on Extracted Features ***(Fruits-360)***

- The close alignment between training and validation curves (both loss and accuracy) demonstrates a well-regularized model with minimal signs of overfitting.
- The results show that using extracted features as input for the MLP is highly effective for the Fruits-360 dataset, achieving strong classification performance while keeping the training process efficient.

- Strategies such as further feature selection or increasing dropout might marginally improve validation accuracy but are not strictly necessary, given the balanced performance.

3.2. MLP applied on Raw Images

Model Architecture:

- Input layer: A flattened image input of
  - 784 for Fashion-MNIST
  - 30.000 for Fruits-360
- Two hidden layers: Each with 512 neurons and ReLU activation.
- **Dropout**: Applied with a rate of 0.3 to prevent overfitting.
- **Output Layer**: <number_of_classes> neurons with softmax activation for classification.
  - where <number_of_classes> is 10 for Fashion-MNIST and 121 for Fruits-360

```
mlp_img = Sequential([
    Dense(512, input_shape=(30000,)),  # Input layer with 512 units
    Activation('relu'),              # ReLU activation
    Dropout(0.3),                    # Dropout for regularization
    Dense(512),                      # Hidden layer with 512 units
    Activation('relu'),              # ReLU activation
    Dropout(0.3),                    # Dropout for regularization
    Dense(num_classes),                      # Output layer with 10 units (num_classes)
    Activation('softmax')            # Softmax activation for probabilities
])
```

Figure 6: MLP Architecture for Original Images

Training Process :

- **Compilation Details**:
  - **Loss Function**: `categorical_crossentropy`, chosen for its suitability in multi-class classification tasks.
  - **Optimizer**: `adam`, which provides adaptive learning rates for efficient convergence.
  - **Metric**: Accuracy, to monitor the model's performance during training.

```
mlp.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 7: MLP Model Compilation with Adam Optimizer and Categorical Crossentropy Loss (for original Images)

- Training Configuration
  - Batch size: 128
  - Number of epochs: 25
  - Validation data (20%)

```
start_time = time.time()
history_mlp_img = mlp_img.fit(
    X_train_small,
    y_train_small_one_hot,
    batch_size=128,
    epochs=25,
    verbose=1,
    validation_data=(X_val, y_val_one_hot)
)
training_time = time.time() - start_time
print(f"Training Time: {training_time:.2f} seconds")
```

Figure 7: Training Process for MLP on Images

Performance Metrics :

	Test Accuracy	Avg Precision	Avg recall	Avg f1-score	Training Time
Fashion-MNIST	0.8948	0.8946	0.8948	0.8945	66.99 s
Fruits-360	0.8843	0.8897	0.8718	0.8694	535.54 s

Graphs:

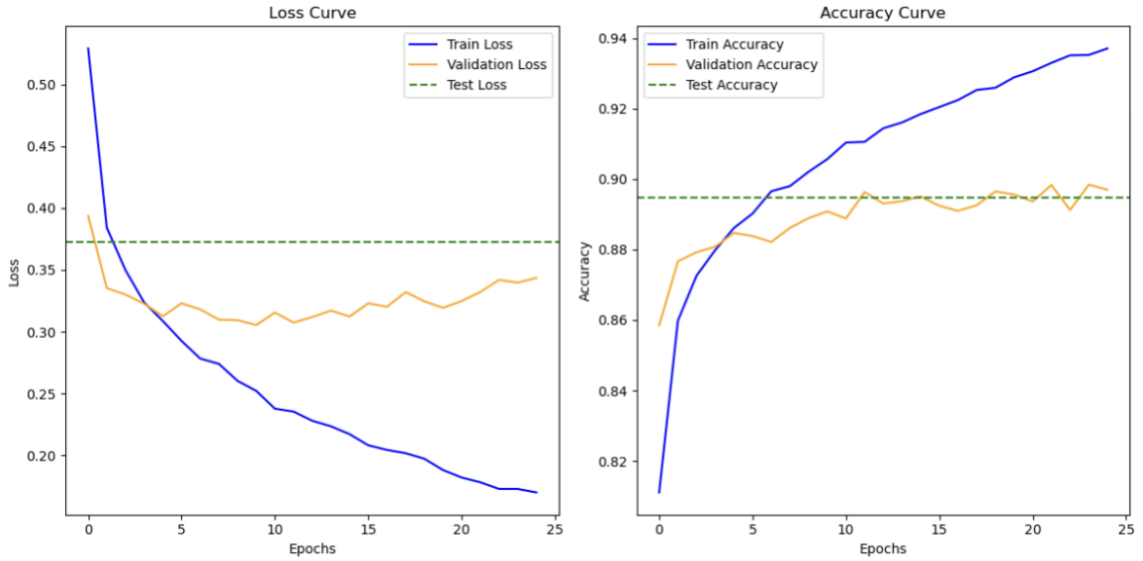

Figure 8: Loss and Accuracy curves for MLP on Images ***(Fashion-MNIST)***

- The model achieves a final test accuracy of approximately **89%**, outperforming the MLP on extracted features for Fashion-MNIST.
- The slight overfitting, as indicated by the small gap between training and validation curves, could be mitigated by employing techniques like increased dropout or early stopping.
- The plateau in validation performance after a few epochs suggests that the model has reached its capacity to learn from the current configuration and dataset.

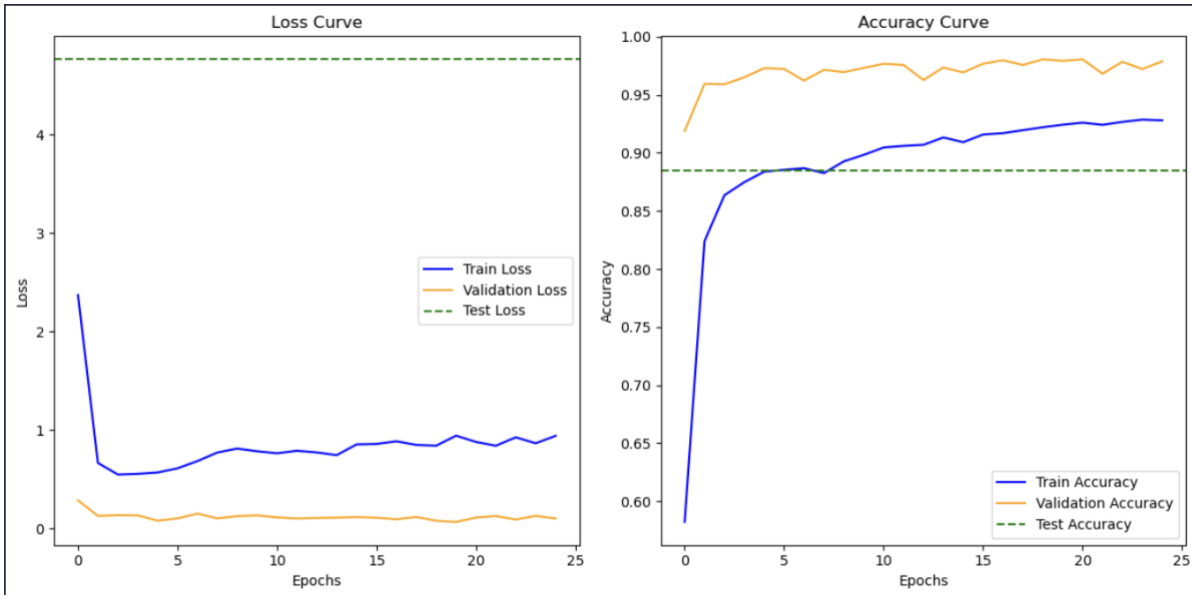

*Figure 9: Loss and Accuracy curves for MLP on Imagess **(Fruits-360)***

- The final test accuracy of approximately **88%** shows that the MLP on raw images achieves strong performance on the Fruits-360 dataset, surpassing the extracted features MLP in accuracy.
- The close alignment between training and validation curves for both loss and accuracy indicates a well-regularized model with minimal signs of overfitting.
- The slightly lower validation accuracy compared to training accuracy suggests room for improvement, potentially through additional regularization or tuning the learning rate.

3.3. Convolutional Network

Model Architecture:

The CNN model was inspired by DeepConvNet and designed with:

- **Three Convolutional Layers**:
  - First layer: 32 filters (for Fashion-MNIST) or 16 filters (for Fruits-360), kernel size of 3×33 \times 33×3, ReLU activation, and Batch Normalization.
  - Second layer: 64 filters (for Fashion-MNIST) or 32 filters (for Fruits-360), similar setup.
  - Third layer: 128 filters (for Fashion-MNIST) or 64 filters (for Fruits-360), similar setup.
- **Global Average Pooling**: Reduces the spatial dimensions.
- **Fully Connected Layer**: 256 neurons (Fashion-MNIST) or 128 neurons (Fruits-360) with ReLU activation.
- **Output Layer**: Softmax activation to classify into 10 (Fashion-MNIST) or 121 (Fruits-360) categories.

```python
# DeepConvNet-like architecture
def create_cnn_model():
    model = Sequential([
        # layer 1 - convolutional
        Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
        BatchNormalization(),
        Activation('relu'),

        # layer 2 - convolutional
        Conv2D(64, (3, 3), padding='same'),
        BatchNormalization(),
        Activation('relu'),

        # layer 3 - convolutional
        Conv2D(128, (3, 3), padding='same'),
        BatchNormalization(),
        Activation('relu'),

        # Global Average Pooling
        GlobalAveragePooling2D(),

        # Fully Connected Layer (Linear)
        Dense(256, activation='relu'),
        Dropout(0.5),

        # Output Layer (Softmax)
        Dense(10, activation='softmax')  # 10 classes
    ])
    return model
```

```python
def create_cnn_model():
    model = Sequential([
        # Layer 1 - Convolutional
        Conv2D(16, (3, 3), padding='same', input_shape=(100, 100, 3)),
        BatchNormalization(),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        # Layer 2 - Convolutional
        Conv2D(32, (3, 3), padding='same'),
        BatchNormalization(),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),

        # Layer 3 - Convolutional
        Conv2D(64, (3, 3), padding='same'),
        BatchNormalization(),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        # Flattening
        Flatten(),

        # Fully Connected Layer
        Dense(128, activation='relu'),
        Dropout(0.5),

        # Output Layer
        Dense(121, activation='softmax')
    ])
    return model
```

Figures 9 and 10: CNN Architecture for Fashion-MNIST and Fruits-360

Compilation Details

- **Loss Function**: `sparse_categorical_crossentropy` for multi-class classification.
- **Optimizer**: Adam, chosen for its adaptive learning rates.
- **Metric**: Accuracy, to monitor training performance.:

Data Augmentation

- To enhance model generalization, the following augmentations were applied:
  - Random rotations (up to 15 degrees).
  - Horizontal flips.
  - Width and height shifts (10%).
- The augmented dataset was created using TensorFlow's `ImageDataGenerator`.

Training Process

- For both original and augmented datasets:
  - **Fashion-MNIST**: Trained for 20 epochs with a batch size of 128.
  - **Fruits-360**: Same configuration.
- Training times were recorded and compared.

```python
model_without_aug = create_cnn_model()

model_without_aug.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])


start_time = time.time()
history_without_aug = model_without_aug.fit(
    X_train_small, y_train_small,
    validation_data=(X_val, y_val),
    epochs=20,
    batch_size=128,
    verbose=1
)
training_time = time.time() - start_time
print(f"Training Time: {training_time:.2f} seconds")
```

Figure 11: e.g. of CNN Training for Augmented Dataset

Performance Metrics :

- On original dataset:

	Test Accuracy	Avg Precision	Avg recall	Avg f1-score	Training Time
Fashion-MNIST	0.7868	0.8297	0.7869	0.7781	2008.75 s
Fruits-360	0.8560	0.8624	0.8186	0.8194	1669.83 s

- On augmented dataset:

	Test Accuracy	Avg Precision	Avg recall	Avg f1-score	Training Time
Fashion-MNIST	0.7868	0.8117	0.7733	0.7691	1689.41 s
Fruits-360	0.8560	0.8215	0.7696	0.7556	1585.25 s

Graphs:

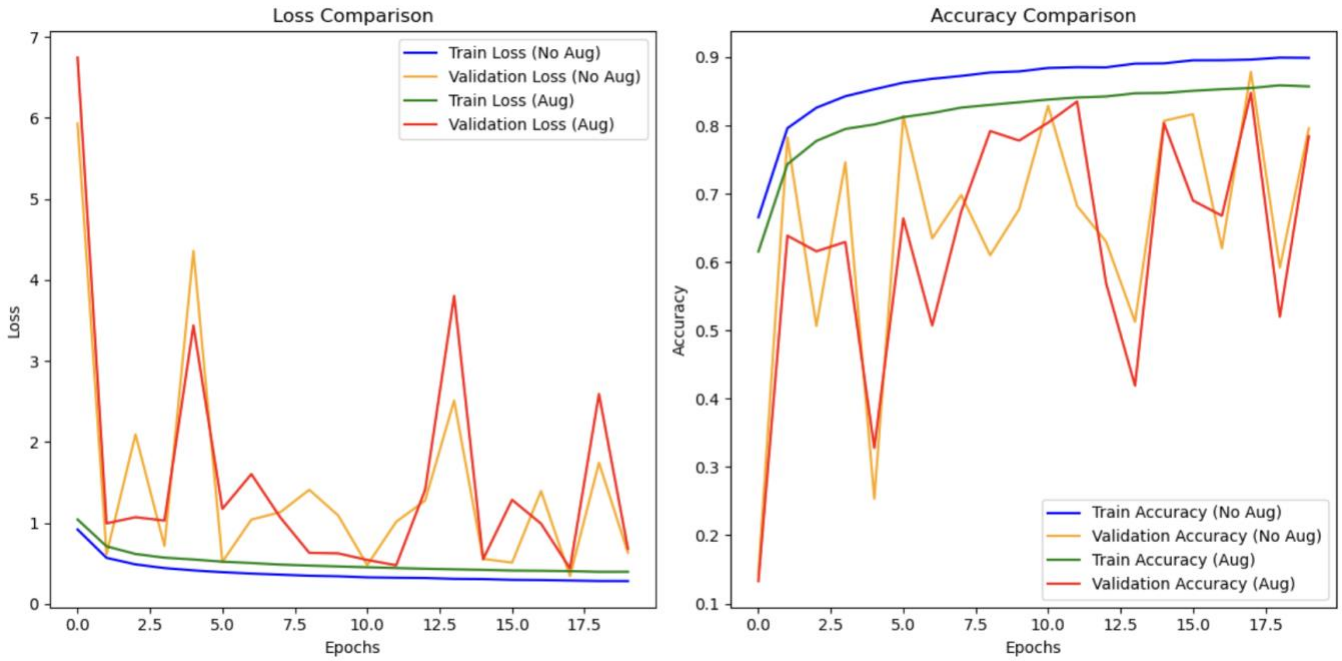

Figure 12: Loss and Accuracy curves for CNN ***(Fashion-MNIST)***

- **Impact of Augmentation**:
    - While data augmentation can improve generalization by introducing variability, in this case, it seems to have introduced too much noise or irrelevant transformations, leading to instability in both loss and accuracy curves.
    - The validation performance with augmentation is comparable to or slightly worse than without augmentation, indicating that the augmentation approach needs refinement (e.g., adjusting parameters such as rotation range or flipping).

- **Overfitting**:
    - The model trained without augmentation shows clear signs of overfitting, with a significant gap between training and validation performance.
    - Augmentation reduces overfitting slightly by increasing the diversity of training data, but it is not sufficient to achieve better overall performance.

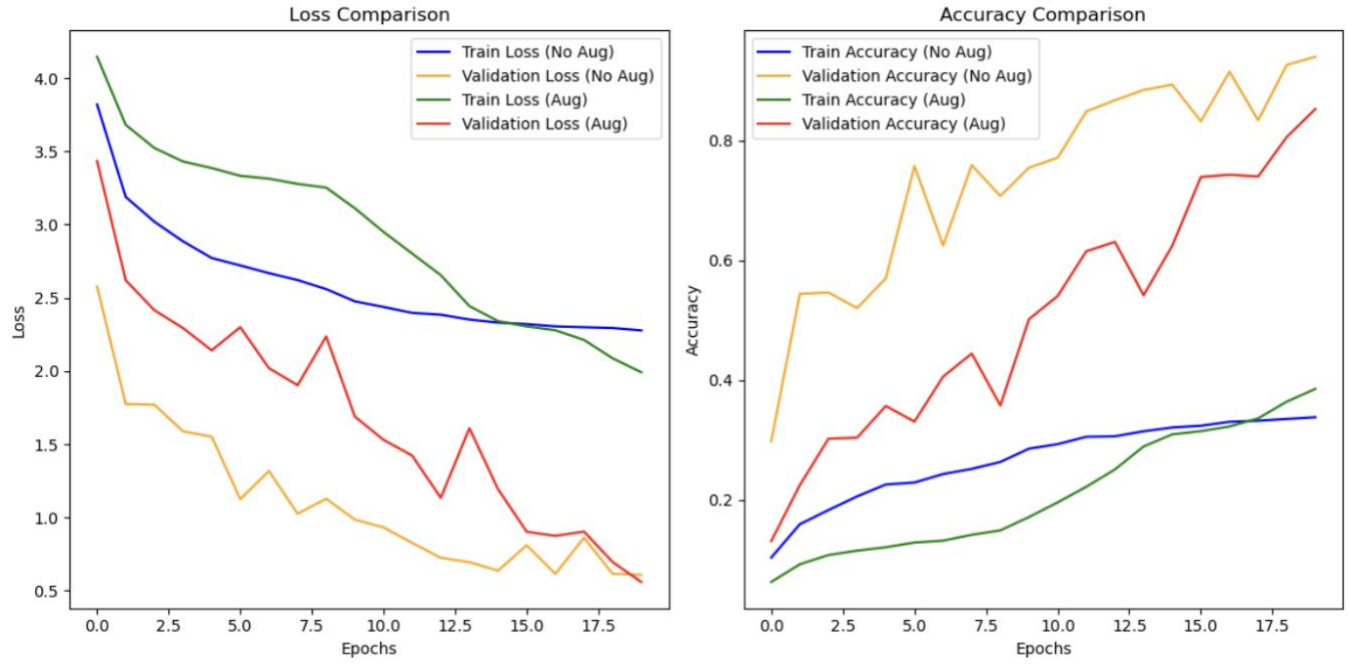

Figure 13: Loss and Accuracy curves for CNN ***(Fruits-360)***

- **Impact of Augmentation**:
  - Data augmentation reduces the overfitting observed in the model trained on non-augmented data.
  - The smaller gap between training and validation curves for the augmented dataset suggests improved generalization.
- **Performance Comparison**:
  - The augmented model achieves better overall validation accuracy and stability compared to the non-augmented model.
  - The trade-off is a slightly slower learning curve for the augmented model due to the added complexity.

3.4. Fine-Tuning with ResNet-18

Preprocessing

- **Data Preparation**:
  - **Fashion-MNIST**: Converted grayscale images to RGB by duplicating the channel three times.
  - Resized images to 32×32 for compatibility with ResNet-18 pretrained on CIFAR-10.
  - **Fruits-360**: Resized images to 32×32 and normalized pixel values to the range [0, 1].

```
from torch.utils.data import TensorDataset, DataLoader

train_dataset = TensorDataset(X_train_small, y_train_small)
val_dataset = TensorDataset(X_val, y_val)
test_dataset = TensorDataset(X_test_transformed, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=256, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False)
✓ 0.0s
```

Figure 14: DataLoader Creation for ResNet-18 Fine-Tuning

Model Setup

- **Pretrained Model**:
  - Loaded the ResNet-18 architecture pretrained on CIFAR-10.
  - Replaced the final fully connected layer with a custom linear layer:
    - Fashion-MNIST: Output layer with 10 classes.
    - Fruits-360: Output layer with 121 classes.

```
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

Figure 14: Modifications to Pretrained ResNet-18 for Fine-Tuning

Training Configuration

- **Optimizer and Loss Function**:
  - Optimizer: SGD with momentum (0.90.90.9) and learning rate scheduling (StepLR with decay factor 0.10.10.1 every 7 steps).
  - Loss function: CrossEntropyLoss.

```
criterion = nn.CrossEntropyLoss()
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

Figure 15: Optimizer, Loss Function, and Learning Rate Scheduler

- **Device Utilization**:
  - Model trained on an Apple M1 device using Metal Performance Shaders (MPS), falling back to CPU if unavailable.

Training Process

- Batch size: 256.
- Number of epochs: 20.

Performance Metrics :

	Test Accuracy	Avg Precision	Avg recall	Avg f1-score	Training Time
Fashion-MNIST	0.9315	0.9313	0.9315	0.9314	225.44 s
Fruits-360	0.9461	0.9451	0.9461	0.9451	267.58 s

Graphs

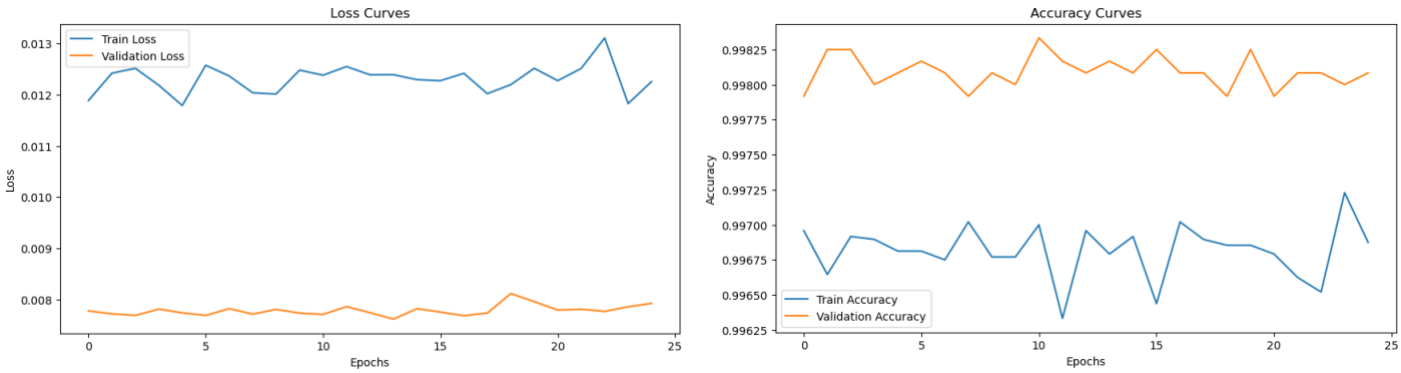

Figure 16: Loss and Accuracy Curves for Fashion-MNIST

- **Model Performance**:
  - The low and stable loss values, combined with the high and consistent accuracy scores, highlight the effectiveness of ResNet's transfer learning for the Fashion-MNIST dataset.
  - The model exhibits near-perfect performance with minimal overfitting, as evidenced by the close alignment between training and validation curves.

- **Stability**:
  - The small fluctuations in accuracy and loss curves suggest a well-optimized model and training process.
  - The pretrained ResNet model benefits from a stable training regime, requiring minimal tuning to achieve high performance.

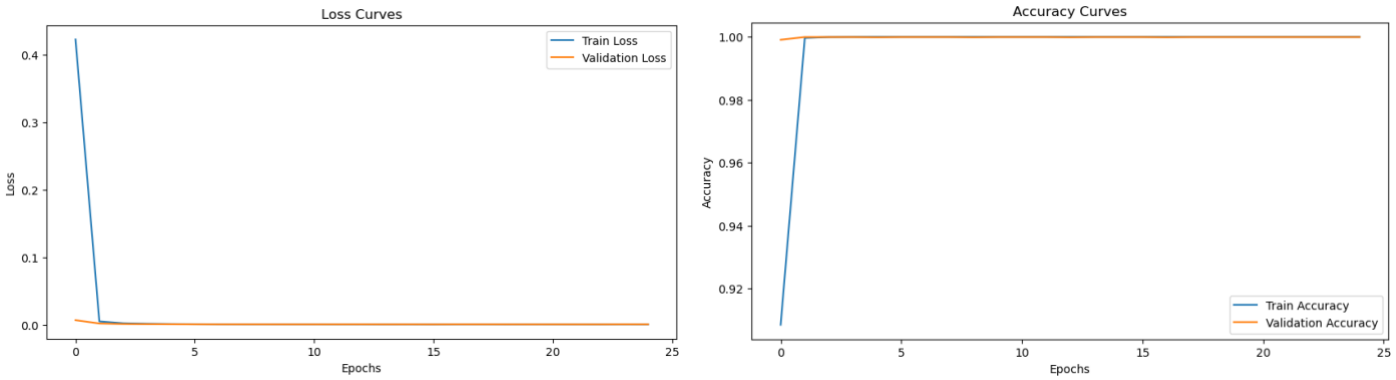

Figure 17: Loss and Accuracy Curves for Fruits-360

- **Model Performance**:

- The ResNet model achieves near-perfect performance on the Fruits-360 dataset, with both training and validation accuracy and loss curves converging closely.
- This suggests that the dataset's characteristics align well with the pretrained features of ResNet, leading to highly efficient learning.

- **Stability**:

  - The stability of the curves across epochs, with no significant fluctuations, indicates a well-optimized model and effective fine-tuning process.
  - The rapid convergence within the first epoch highlights the benefits of transfer learning, where pretrained weights accelerate learning.

---

4. Summary of Performance Metrics

Fashion-MNIST	Test Accuracy	Avg Precision	Avg recall	Avg f1-score	Training Time
MLP on selected features	0.8773	0.8768	0.8774	0.8766	42.62 s
MLP on raw Images	0.8948	0.8946	0.8948	0.8945	66.99 s
CNN (orginal dataset)	0.7868	0.8297	0.7869	0.7781	2008.75 s
CNN (augmented dataset)	0.7868	0.8117	0.7733	0.7691	1689.41 s
Fine-Tuning Resnet-18	0.9315	0.9313	0.9315	0.9314	225.44 s
Logistic Regression	0.8816	0.88	0.88	0.88	
SVM	0.9076	0.91	0.91	0.91	
Random Forest	0.8730	0.84	0.77	0.79	
Gradient Boosted Trees	0.9023	0.90	0.90	0.90	

Fruits-360	Test Accuracy	Avg Precision	Avg recall	Avg f1-score	Training Time
MLP on selected features	0.8802	0.8818	0.8778	0.8739	81.50 s
MLP on raw Images	0.8843	0.8897	0.8718	0.8694	535.54 s
CNN (orginal dataset)	0.8560	0.8624	0.8186	0.8194	1669.83 s
CNN (augmented dataset)	0.8560	0.8215	0.7696	0.7556	1585.25 s
Fine-Tuning Resnet-18	0.9461	0.9451	0.9461	0.9451	267.58 s
Logistic Regression	0.8079	0.82	0.81	0.81	
SVM	0.78	0.87	0.87	0.87	
Random Forest	0.7802	0.84	0.77	0.79	
Gradient Boosted Trees	0.7702	0.80	0.76	0.77	

5.Analysis and Discussions

5.1. Comparative Insights Based on the Performance Metrics Table

1 Fashion-MNIST Dataset

Accuracy Performance:

- **Best Model**: **Fine-Tuning ResNet-18** (93.15% test accuracy) outperforms all other approaches.
- **Close Competitors**: SVM (90.76%) and Gradient Boosted Trees (90.23%) achieve competitive performance but are still inferior to fine-tuning ResNet-18.
- **Lower Performance**: CNN (both original and augmented datasets) underperformed significantly (78.68% accuracy). This could be due to the CNN architecture's inefficiency when trained from scratch compared to leveraging a pretrained network.

Precision, Recall, and F1-Score:

- Fine-Tuning ResNet-18 maintains the highest metrics across precision, recall, and F1-score, further solidifying its robustness.
- SVM and Gradient Boosted Trees also achieve strong precision, recall, and F1-scores, showing they are effective for smaller-scale datasets.

Training Time:

- **Fastest Model**: MLP on selected features (42.62 seconds) is the most efficient but sacrifices performance.
- Fine-Tuning ResNet-18 (225.44 seconds) balances computational cost and accuracy effectively.
- **Longest Training Time**: CNNs (2008.75 seconds for original dataset) require significantly more time without commensurate accuracy gains, highlighting inefficiency.

2. Fruits-360 Dataset

Accuracy Performance:

- **Best Model**: **Fine-Tuning ResNet-18** achieves a dominant 94.61% test accuracy, demonstrating the advantage of transfer learning for complex datasets with a large number of classes.
- **Close Competitor**: MLP on raw images (88.43%) is the next best but lags by over 6%, showing the limitation of simpler architectures in handling large-scale datasets.
- **Lower Performance**: Logistic Regression (80.79%) and Gradient Boosted Trees (77.02%) perform poorly, indicating their limitations for this complex, high-dimensional dataset.

Precision, Recall, and F1-Score:

- Similar to accuracy, Fine-Tuning ResNet-18 achieves the highest precision, recall, and F1-score, confirming its consistency across metrics.

- Other models, such as CNN (original and augmented datasets), exhibit decent metrics (around 85.60% accuracy) but fall short of ResNet-18.

Training Time:

- **Fastest Model**: MLP on selected features (81.50 seconds) is computationally efficient but far from the top performer in terms of accuracy.
- Fine-Tuning ResNet-18 (267.58 seconds) offers a good tradeoff between accuracy and computational cost.
- **Longest Training Time**: MLP on raw images (535.54 seconds) and CNN (1669.83 seconds) require significantly more training time, with limited performance gains for CNN.

5.2. General Observations Across Datasets

- **ResNet-18 Dominance**:
  - Fine-tuning ResNet-18 stands out as the top-performing model for both datasets, showcasing the benefits of transfer learning.
  - Its performance improvement is more significant on the Fruits-360 dataset, which has a higher number of classes and more complex features.
- **Classical ML Models**:
  - Logistic Regression and Random Forest perform relatively well on Fashion-MNIST but struggle with Fruits-360, indicating their limitations for larger-scale, complex datasets.
  - SVM performs well on Fashion-MNIST (90.76%) but falls short on Fruits-360 (78%).
- **CNN Limitations**:
  - Training CNNs from scratch is computationally expensive and less effective than leveraging pretrained networks like ResNet-18.
- **MLP Effectiveness**:
  - MLP on selected features is an efficient option with decent performance for simpler datasets (Fashion-MNIST).
  - However, it underperforms for more complex datasets like Fruits-360 when compared to ResNet-18.

	Strenghts	Weaknesses
MLP on Selected Features	Fast training, good performance on moderate datasets.	Limited by feature extraction quality.
MLP on Raw Images	Better performance by learning directly from raw data	Increased training time and computational cost.
CNN	Capable of learning spatial hierarchies, potential for high accuracy	High computational cost, suboptimal performance for datasets.
Fine-Tuning ResNet-18	Exceptional accuracy and generalization, efficient training via transfer learning.	Requires pretrained weights, slightly higher resource demand.