**Deep Learning Assignment Report – 2024-25**

Name: Iason-Christoforos Asproudis  
Student ID: p3352318

### Part 1: Image Classification

#### Dataset Description

We utilized two datasets for initial image classification experiments:

* **Fashion-MNIST**: 28x28 grayscale images categorized into 10 clothing item classes.
* **CIFAR-10**: 32x32 RGB images across 10 object classes such as airplanes, cats, and cars.

Both datasets are provided through tensorflow.keras.datasets.

#### Model Architectures

**Fashion-MNIST (MLP)**:

* Fully connected neural network using Functional API
* Two hidden layers with 256 and 128 units
* ReLU activation and Dropout (0.3)
* Glorot uniform initialization

**CIFAR-10 (CNN)**:

* Three convolutional blocks with Conv2D, BatchNormalization, MaxPooling, and Dropout
* Flatten and dense layers at the end
* Trained with EarlyStopping

#### Training Strategy

* Optimizer: Adam
* EarlyStopping with patience of 3
* Training epochs: 10–30
* Batch size: 128
* Dropout and BatchNormalization to reduce overfitting and stabilize training

#### Challenges & Solutions

* **Slow convergence on CIFAR-10**: Deepened architecture and added normalization layers
* **Overfitting in Fashion-MNIST**: Resolved with EarlyStopping and Dropout

#### Results Summary

* **Fashion-MNIST (MLP)**:
  + Test Accuracy: 88.7%
  + Validation Accuracy: 89.0%
* **CIFAR-10 (CNN)**:
  + Test Accuracy: ~71%

### Part 2: X-ray Classification – MURA Dataset

#### Project Strategy

We built an efficient, multitask deep learning pipeline to classify musculoskeletal radiographs from the MURA dataset:

* **Binary Classification**: Normal vs. Abnormal
* **Multitask Learning**: Added body part prediction as a secondary task
* **Efficiency**: Optimized for Colab Pro (GPU & RAM), with modular reusable code
* **Robustness**: Introduced an "Other" class for body parts not in the primary set
* **Transfer Learning**: Used EfficientNetB0 as a strong pretrained baseline
* **Modularity**: Code organized into modules: data\_loader.py, model\_pretrained.py, metrics.py, config.py

#### Implementation Steps

1. **Data Preparation**:
   * Preprocessed and cached training and validation data locally
   * Used float16 format to reduce memory
   * All image inputs resized and normalized
2. **Model Development**:
   * Custom CNN from scratch with multitask heads (binary + body part)
   * EfficientNetB0 pretrained backbone model with fine-tuning support
3. **Evaluation**:
   * Metrics: F1-score, accuracy, precision, recall (binary task)
   * Body part classification evaluated using categorical accuracy

#### Challenges & Solutions

| **Challenge** | **Solution** |
| --- | --- |
| Slow preprocessing | Cached dataset locally in compressed format |
| Colab memory exceeded (83.5 GB) | Used float16 arrays and reduced batch size; introduced generators |
| Class weight errors in multitask model | Switched to sample\_weight via custom generator |
| Input shape mismatch with EfficientNetB0 | Updated preprocessing to output 244x244 RGB images |
| Runtime environment inconsistency | Centralized config and file path logic using config.py |
| TensorFlow package compatibility | Downgraded Python and manually adjusted dependencies |
| Monitoring data loading | Integrated tqdm progress bars during preprocessing |

### Results Summary

#### CNN Baseline (Multitask Learning)

* Binary Accuracy: 64.9% (Validation)
* Body Part Accuracy: ~100%
* F1-Score (Abnormal): 0.60

| **Class** | **Precision** | **Recall** | **F1-Score** |
| --- | --- | --- | --- |
| Normal | 0.64 | 0.73 | 0.69 |
| Abnormal | 0.66 | 0.56 | 0.60 |
| **Accuracy** |  |  | **0.65** |

The CNN model successfully learned both tasks, but showed overfitting and limited generalization in binary classification.

### Next Steps (Pretrained Model)

1. **EfficientNetB0 Model Completion**:
   * Use MURAGenerator for memory-efficient training
   * Ensure float16 input handling and 244x244 resizing
2. **Evaluation**:
   * Compare performance with baseline CNN
   * Evaluate binary classification and body part accuracy
3. **Fine-Tuning**:
   * Unfreeze top layers of EfficientNetB0
   * Retrain with a low learning rate
4. **Final Analysis**:
   * Compare learning curves, training duration, and classification metrics
   * Document trade-offs between scratch vs. transfer learning approaches

### Final Submission

* ✅ Clean and modular .ipynb notebooks
* ✅ Complete report with visualizations and metrics
* ✅ Preprocessing modules and training scripts
* ✅ GitHub repository: [<https://github.com/jasproudis/deep-learning-assignment>]