Scientific Computing Project

Image Classification with AlexNET Light

Technische Universität Bergakademie Freiberg
Institut für Numerische Mathematik und Optimierung
Dr. Oliver Rheinbach
Dr. Stephan Köhler

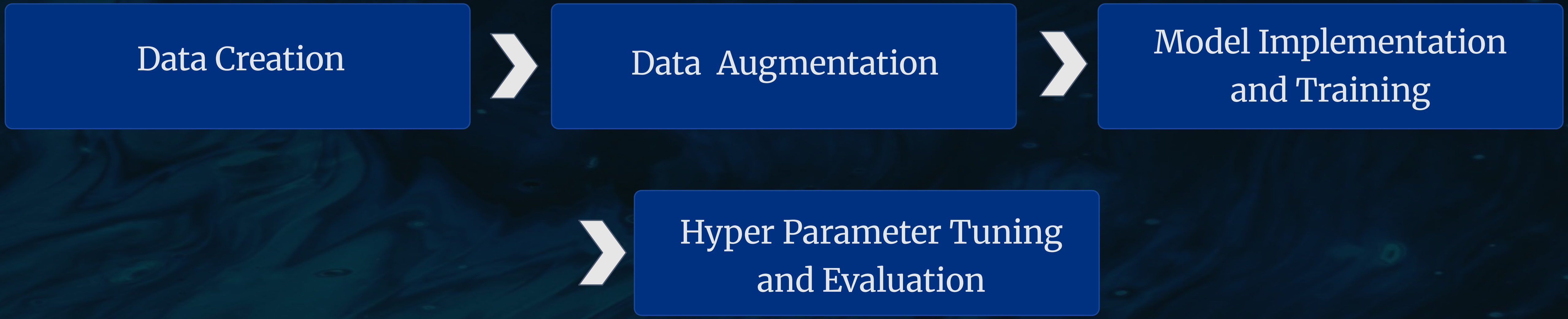
Presented by
Abhijith Sai Thirunahari
Harsha Ramachandra

TUBAF - Winter Semester 2024/25

Project Objective

- ➔ Implementing a Deep Neural Network to classify the image dataset of 10 distinct categories
- ➔ Implementing appropriate data augmentation strategies to generalize the model
- ➔ Classification of the data using AlexNET light Deep Neural Network Architecture.
- ➔ Improve the model performance using Hyper Parameter Tuning to obtain High Accuracy

Project Overview


Data Creation
Data Augmentation
Model Implementation and Training
Hyper Parameter Tuning and Evaluation

Data Creation

- ➔ 10 classes of images were set as a governing domain under which the images were collected.
- ➔ Obtaining the image dataset by manually photographing objects from the surroundings was the prerequisite step for the project to commence successfully
- ➔ Several sources for images were used, including but not restricted to: homes, malls, public places, college premises
- ➔ The 10 classes for the dataset are as follows

Chairs	Spoons	Plants	T-shirts	Forks
Knives	Bikes	Cups	Shoes	Bottles

- ➔ However, the data created using the above methods contains the noise.
- ➔ The presence of extraneous object in the image (Image on left and center) negatively impact the training process.
- ➔ Images related to one class contains the objects related to the other classes being trained. (Image on Right)
- ➔ However, denoising the data is difficult but not impossible.

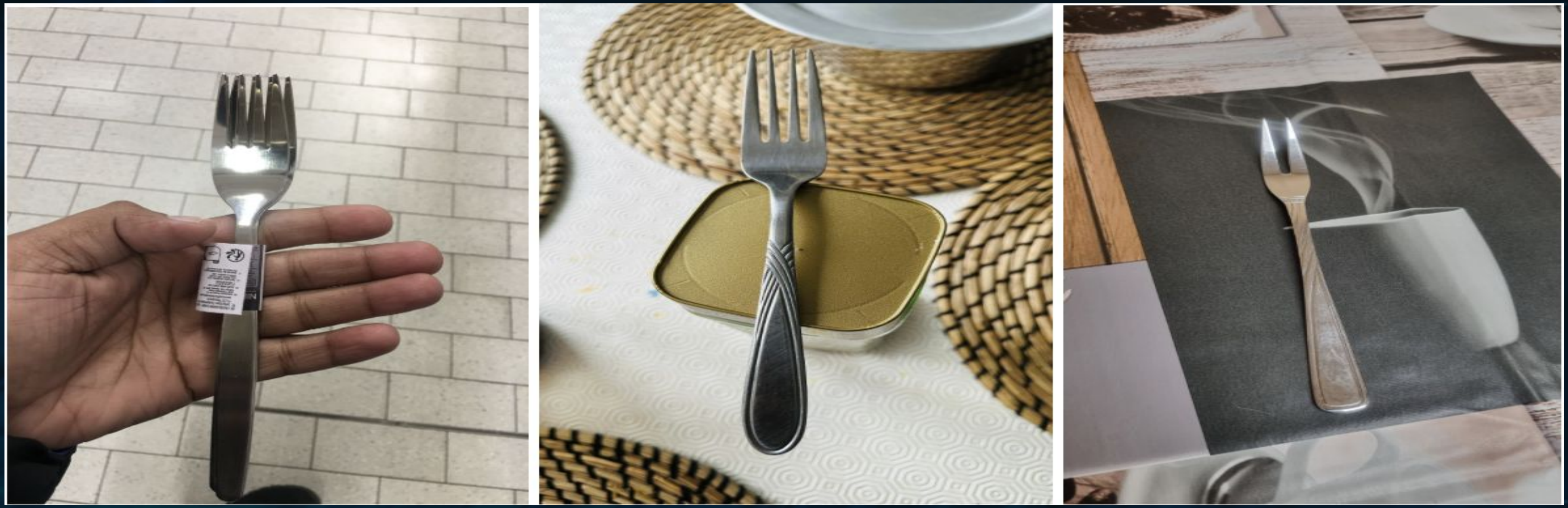


Figure 1: Images showing Noise present in the data
[Left: hand , middle: unknown object, left: multiple classes present in single image (cup, fork)]

Data Augmentation

- ➢ The dataset constituted over a 3,000+ original images in all, including all the 10 classes, as a collective effort of every other participating project groups, which was amalgamated into a central dataset and provisioned to every team for training the data
- ➢ However, this size of the original dataset, in terms of the number of images, is minimalistic and inadequate to output an efficient classifying model
- ➢ Due to the nature of the original data size being extremely small, the procedure of data augmentation was employed in order to enhance the number of images to a higher order

Data augmentation in this project involved generating copies of images from the original ones, with distortions in terms of angles, color, gradient and other pixel based changes, which retain the descriptive features of the image, but changing them so as to not be identical to the original images obtained from manual photography

Code snippet

```
image_augment = {
    "train": v2.Compose([
        v2.Resize((224, 224)),
        v2.RandomHorizontalFlip(p=0.5),
        v2.RandomRotation(20),
        v2.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
        v2.RandomAffine(degrees=0, translate=(0.1, 0.1)),
        v2.ToDtype(torch.float32, scale=True)
    ])
}
image_transforms = {
    "train": v2.Compose([
        v2.ToImage(),
        v2.ToDtype(torch.float32, scale=True),
        v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
}
```

Model Implementation and Training

- Model has to classify the images based on class [cups, plants, tshirts etc]
- Implemented the AlexNET light architecture from scratch.
- Experimented on different Model Architectures to create a robust model. [Better Accuracy]

AlexNET Light

32 x 32 Image
parameters: ~43 Thousand

AlexNET Light

224 x 224 Image
parameters: ~51.5 Million

Training

➔ Augmentation of Images to enhance the Sample Size
➔ Suitable use of parameters to obtain high accuracy
➔ Enabling the Graphic Memory for quicker learning

What is AlexNet ?

➔ Introduced in 2012 by Krizhevsky et al.

➔ Lightweight, Computationally efficient.

➔ Architecture consists of 5 CNNs each with a kernel size of 11 x 11 [227 x 227 Image size]

➔ Requires huge computational capacity

ReLU Activation

Faster, efficient training

Max Pooling

Enhances accuracy

Batch Normalization

Faster and stable Training

Dropout

Regularizes to prevent overfitting

Model: 32 x 32 Image

- ➔ Light weight variant of AlexNET architecture which is implemented from scratch.
- ➔ Very fewer number of Parameters, Hence easily trained with less computational power.
- ➔ Limited model ability to learn complex features.
- ➔ Easily prone to Overfitting.

Model: 32 x 32 Image

```
Input (32×32×3)
│
├─[Block 1]
│  ├─Conv3×3 (16 filters, padding=1) → [32×32×16]
│  ├─ReLU
│  ├─BatchNorm
│  ├─MaxPool 2×2 (stride=2) → [16×16×16]
│
├─[Block 2]
│  ├─Conv3×3 (32 filters, padding=1) → [16×16×32]
│  ├─ReLU
│  ├─BatchNorm
│  ├─MaxPool 2×2 (stride=2) → [8×8×32]
│
├─[Block 3]
│  ├─Conv3×3 (32 filters, padding=1) → [8×8×32]
│  ├─ReLU
│  ├─BatchNorm
│
├─[Block 4]
│  ├─Conv3×3 (64 filters, padding=1) → [8×8×64]
│  ├─ReLU
│  ├─BatchNorm
│  ├─MaxPool 2×2 (stride=2) → [4×4×64]
│
├─Flatten → [4×4×64 = 1024 units]
├─FC1: 1024 → 512
│  ├─ReLU
│  └─Dropout (p=0.5)
└─FC2: 512 → 10 (output classes)
```

Model: 224 x 224 Image

- ➔ Complex variant of AlexNET architecture which is from scratch.
- ➔ ~51.5 Million Parameters, Hence requires high computational power.
- ➔ Higher model ability to learn complex features from the data.
- ➔ Builds the robust model with less chance of overfitting with better accuracy.

Model: 224 x 224 Image

```
Input (224×224×3)
│
├─[Block 1]
│   ├─Conv3×3 (32 filters, padding=1) → [224×224×32]
│   ├─ReLU
│   ├─BatchNorm
│   ├─MaxPool 2×2 (stride=2) → [112×112×32]
│   └─Dropout2d (p=0.2)
│
├─[Block 2]
│   ├─Conv3×3 (64 filters, padding=1) → [112×112×64]
│   ├─ReLU
│   ├─BatchNorm
│   ├─MaxPool 2×2 (stride=2) → [56×56×64]
│   └─Dropout2d (p=0.2)
│
│
├─[Block 3]
│   ├─Conv3×3 (128 filters, padding=1) → [56×56×128]
│   ├─ReLU
│   ├─BatchNorm
│   └─MaxPool 2×2 (stride=2) → [28×28×128]
Flatten → [28×28×128 = 100,352 units]
│
├─FC1: 100,352 → 512
│   ├─ReLU
│   └─Dropout (p=0.5)
│
└─FC2: 512 → 10 (output classes)
```

Training

- ➔ AlexNet was trained using Adam optimizer with a batch size of 128 for 80 epochs, with learning rate of 1e-2
- ➔ Introduced the Learning rate scheduler to schedule the change in the learning rate for every few epochs to avoid overfitting. [StepLR, Reduced LR on Plateau; Step size: 10]
- ➔ Experimented on different loss functions which best converge the loss functions during training without overfitting. [Cross Entropy ,Weighted cross entropy, Nll Loss]
- ➔ Integrated **TensorBoard** to monitor the performance during training. [Visualises the loss and accuracy curve during training and validation.]

1 **Optimizer:** Adam, SGD

2 **Loss :** Cross Entropy ,Weighted cross entropy, Nll Loss.

3 **Hardware:** TUBAF HPC Cluster , Apple Silicon 14 Core GPU & 8 Core CPU.

4 **LR Scheduler:** StepLR, Reduced LR on Plateau.

Hyper Parameter Tuning

- ➔ Experiment with different hyperparameter values to find the combination that yields the best performance on the validation set.
- ➔ Introduced new schedulers to enforce the training which was halting after few epochs.
- ➔ Evaluated all runs and chose the best parameter which provides the higher accuracy value.
- ➔ Trained and Saved the model on best parameters.

Batch Size	128 - 512
Learning Rate	1e-2 to 1e-5
weight decay	1e-3 to 1e-4
Optimizers	Adam, SGD
Loss Functions	Cross Entropy, Weighted cross Entropy, NLL Loss
LR Scheduler	StepLR, Reduced LR on Plateau

Code Snippet: config.json

```
"optimizer": {
"type": "Adam",
 "args": {
   "lr": 1e-2,
   "weight_decay": 1e-3,
   "amsgrad": false,
   "betas": [0.9, 0.999]
 }
},
"loss": "nll_loss",
"metrics": [
 "accuracy",
 "top_k_acc"
],
"lr_scheduler": {
    "type": "StepLR",
    "args": {
      "step_size": 50,
      "gamma": 0.1
    }
  },
```

Results

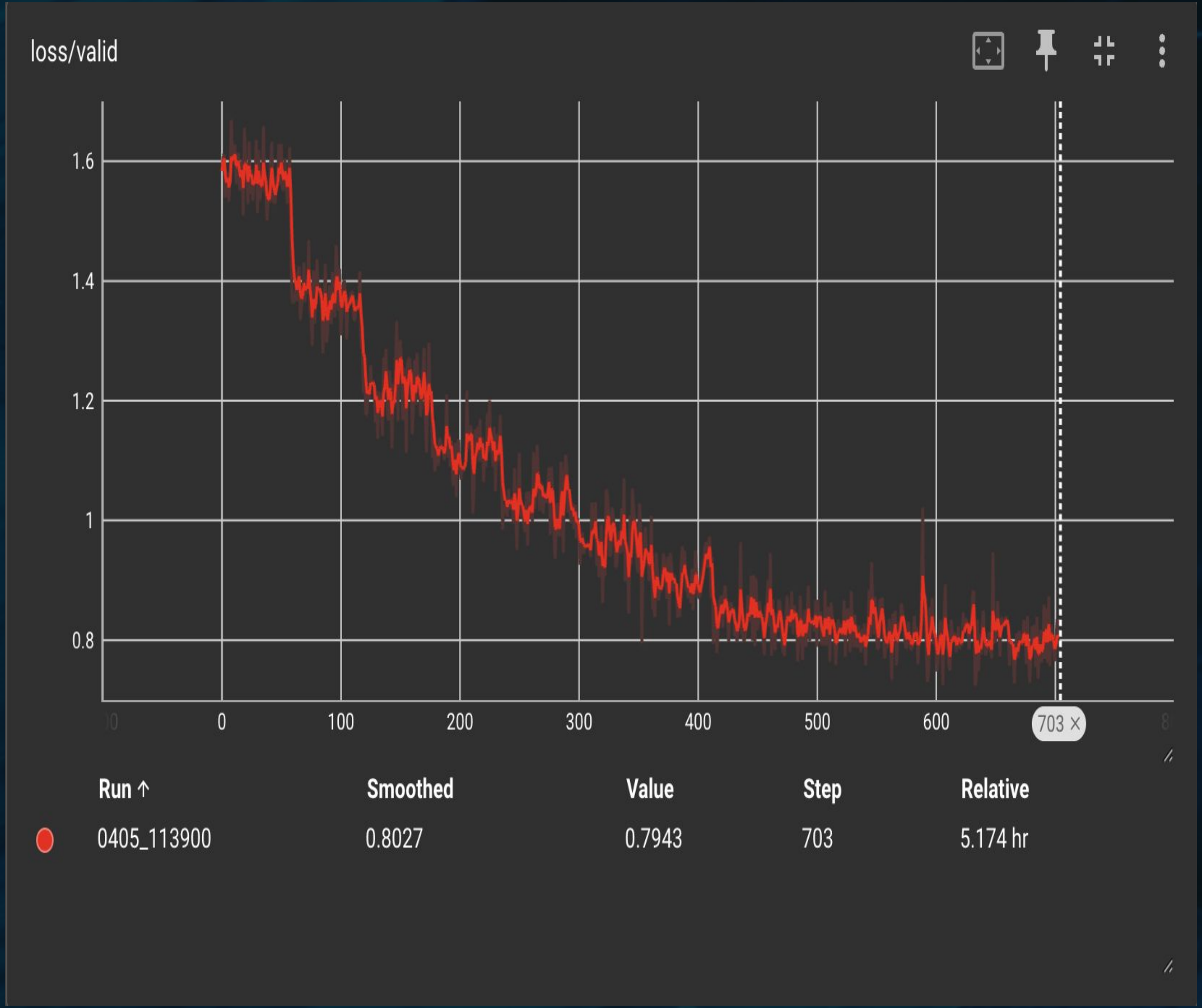
Validation Accuracy As per the Image size:

Model Setup	Accuracy [32 x 32 Image]	Accuracy [224 x 224 Image] [Stopped Training after few 30 Epochs]
Adam , lr=1e-2, decay=1e-3 StepLR, NLL Loss	43%	15%
Adam , lr=1e-3, decay=1e-4 StepLR, Cross Entropy Loss	63%	19%
Adam , lr=1e-3, decay=1e-4, ReduceLROnPlateau, Weighted Cross Entropy Loss	71%	28 %

Best Parameters:

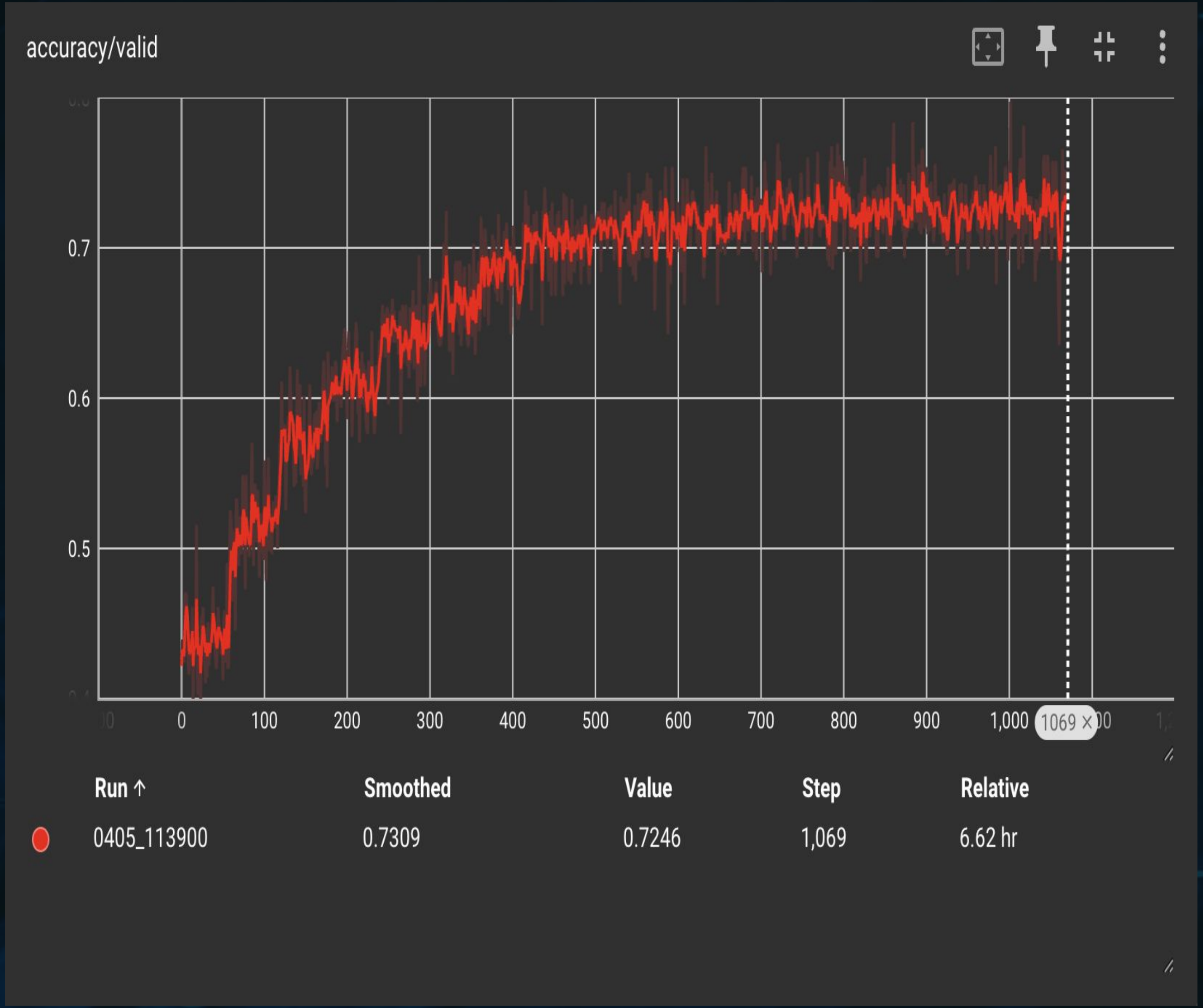
Batch Size:	512 (32 x 32) , 128 (224 x 224)
Learning Rate:	1e-3
weight decay:	1e-4
Optimizer:	Adam
Loss Function:	Cross Entropy Loss
LR Scheduler:	ReduceLROnPlateau

Validation Loss and Accuracy: [32 x 32]



Run ↑	Smoothed	Value	Step	Relative
● 0405_113900	0.8027	0.7943	703	5.174 hr

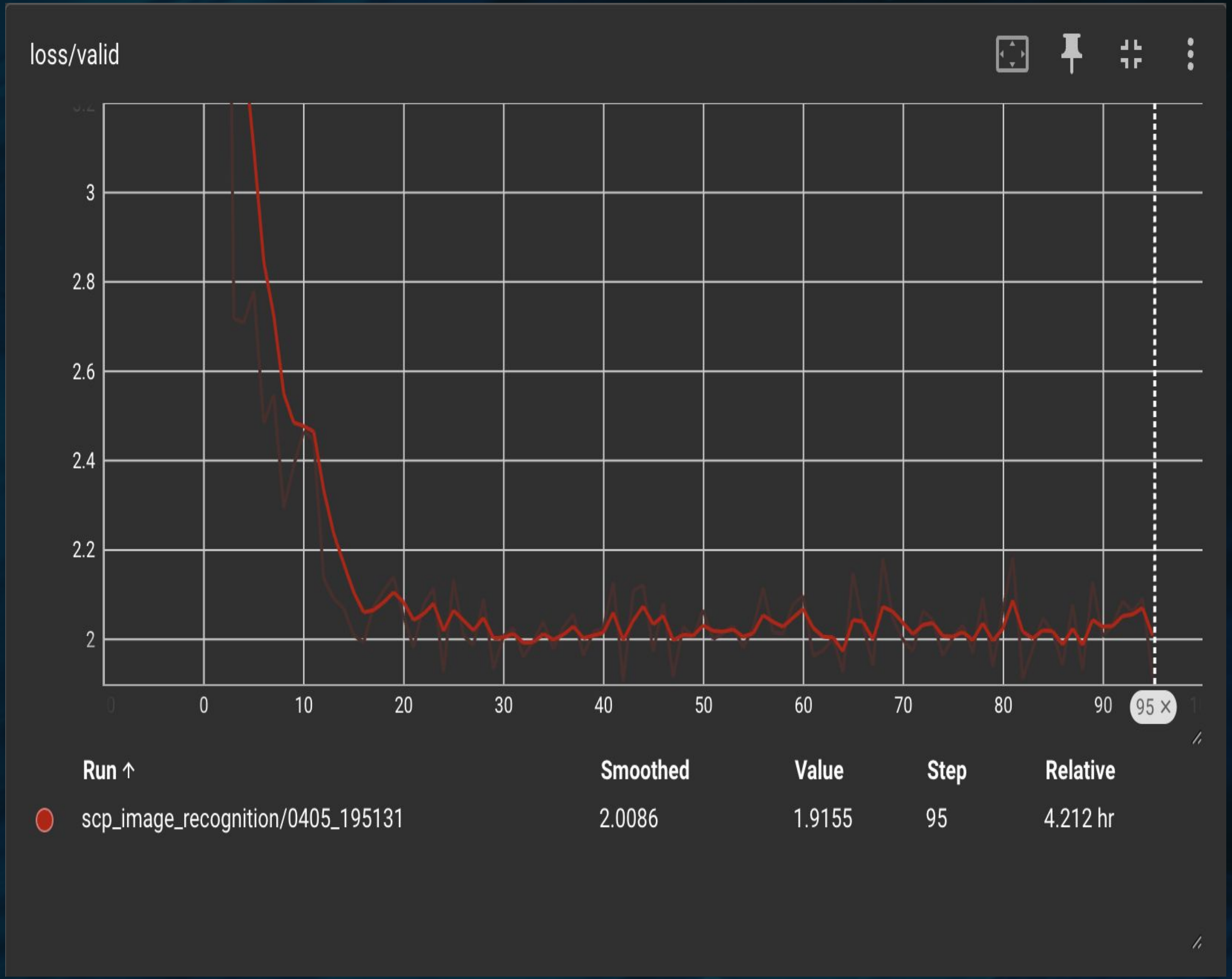
Figure : Validation Loss Curve [32x32 Image]



Run ↑	Smoothed	Value	Step	Relative
● 0405_113900	0.7309	0.7246	1,069	6.62 hr

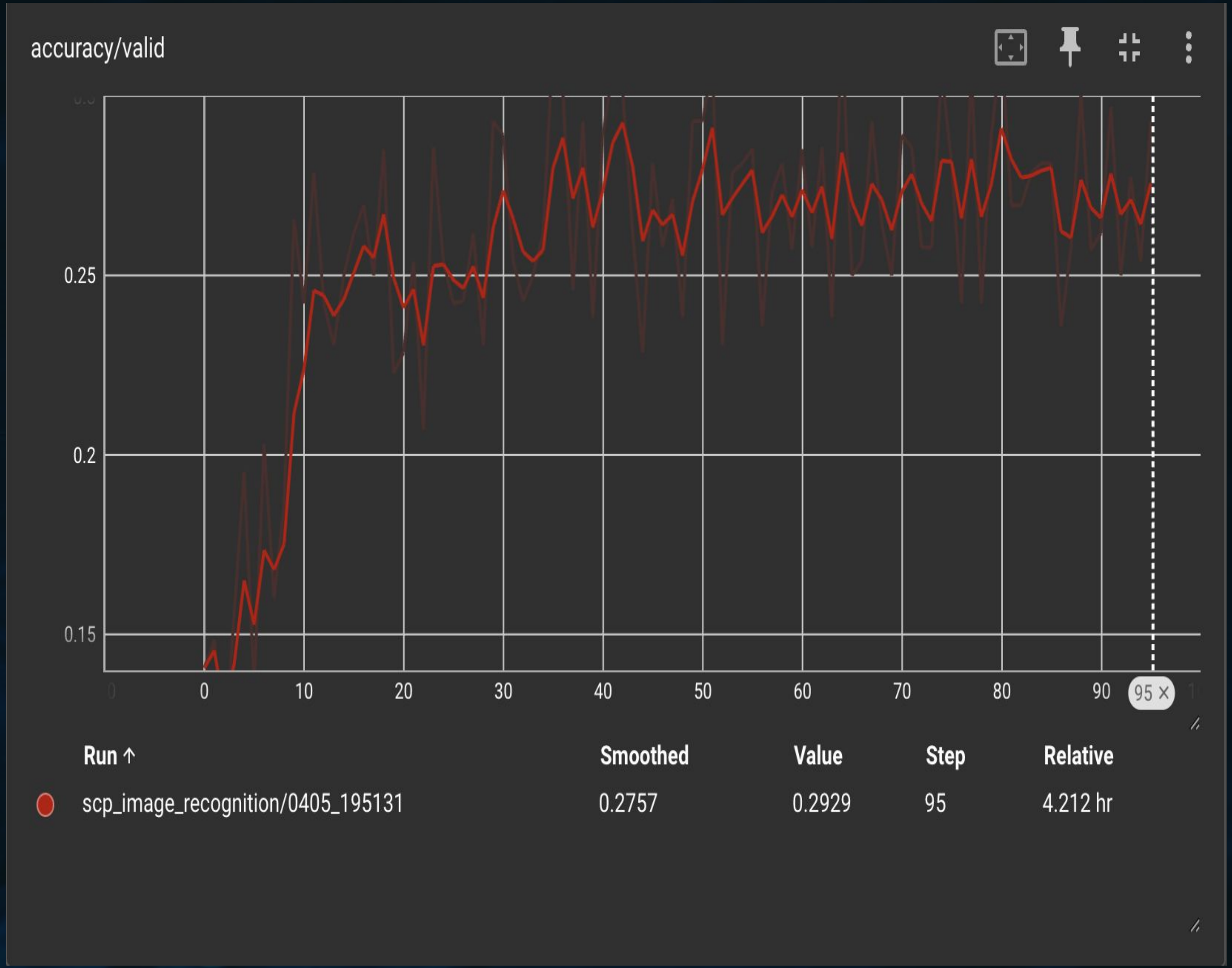
Figure : Validation Accuracy Curve [32x32 Image]

Validation Loss and Accuracy: [224 x 224]



Run ↑ Smoothed Value Step Relative
● scp_image_recognition/0405_195131 2.0086 1.9155 95 4.212 hr

Figure : Validation Loss Curve [224x224 Image]



Run ↑ Smoothed Value Step Relative
● scp_image_recognition/0405_195131 0.2757 0.2929 95 4.212 hr

Figure : Validation Accuracy Curve [224x224 Image]

Conclusion

- ➢ The project gave critical machine learning implications and insights into the realm of image processing , specific to the AlexNet Light architecture
- ➢ Fundamental understanding of stages of image processing, parameter-tuning, optimization and accuracy monitoring and enhancement were garnered
- ➢ Comprehensive report of experimentations, drawbacks and critical observations for possible novel future works

Key Achievements

- Successful implementation of AlexNet Light
- Enhanced accuracy of model to over 70%
- Validation of the AlexNet Light approach

Trade Offs

- Modest accuracy, but scope for better quality classifier
- Original image dataset size not optimal
- Scope for comparative analysis with other models

Future Work

- Research on specific tuning relevant to the dataset
- Achievement of higher competitive performance
- Hybrid approaches with AlexNet Light

Questions ?

References

[1] Research Paper : ImageNet Classification with Deep Convolutional Neural Networks

[2] AlexNet https://en.wikipedia.org/wiki/AlexNet

[3] AlexNet https://www.youtube.com/watch?v=UZDiGooFs54&ab_channel=WelchLabs

[4] Schedulers:https://www.kaggle.com/code/isbhargav/guide-to-pytorch-learning-rate-scheduling

[5] CNN: https://en.wikipedia.org/wiki/CNN

[6] Loss Functions: https://neptune.ai/blog/pytorch-loss-functions