**Deep Learning Project Report: Image Classification using Custom CNN**

**Introduction**

This report presents the implementation of an image classification system using deep learning techniques, applied to the Labeled Faces in the Wild (LFW) dataset. The project aimed to explore and compare a custom convolutional neural network (CNN) model. The core objective was to assess how effectively these architectures could learn discriminative features from facial images and generalize across unseen data. The study also involved applying transfer learning and interpretability techniques like Grad-CAM to understand model behavior. By conducting thorough training, evaluation, and visualization, this project demonstrates how deep learning can solve complex image classification tasks with relatively limited data. The following sections describe the dataset, preprocessing steps, model architectures, training procedures, evaluation metrics, and insights gained from visualizations.

**Dataset Summary**

**Dataset Name:** Labeled Faces in the Wild (LFW)  
**Source:** [Kaggle - LFW Dataset](https://www.kaggle.com/datasets/jessicali9530/lfw-dataset)

The Labeled Faces in the Wild (LFW) dataset is a benchmark dataset widely used for studying face recognition problems in unconstrained environments. It contains over 13,000 images of faces collected from the web, each labeled with the name of the person pictured. The images vary in pose, lighting, and facial expressions, making it a challenging yet suitable dataset for deep learning-based classification tasks.

**Data Preparation**

* The dataset was downloaded programmatically using the opendatasets Python package.
* Only classes (individuals) with at least 10 associated images were retained for analysis.
* The top 10 most frequent classes were selected to ensure a balanced representation for model training.
* All images were resized to a consistent dimension of 128x128 pixels to standardize the input size across the dataset.

**Data Augmentation and Preprocessing**

To increase the robustness and generalization ability of the models, data augmentation was applied to the training set. Transformations included:

* Random horizontal flipping
* Random rotation up to 15 degrees
* Color jittering for brightness, contrast, and saturation
* Tensor conversion and normalization to center pixel values around zero

train\_transform = transforms.Compose([

transforms.Resize((128, 128)),

transforms.RandomHorizontalFlip(),

transforms.RandomRotation(15),

transforms.ColorJitter(0.3, 0.3, 0.2),

transforms.ToTensor(),

transforms.Normalize([0.5]\*3, [0.5]\*3)

])

val\_transform = transforms.Compose([

transforms.Resize((128, 128)),

transforms.ToTensor(),

transforms.Normalize([0.5]\*3, [0.5]\*3)

])

**Data Splitting**

* The dataset was split into a 70% training set and a 30% validation/test set using random\_split.
* Stratified sampling ensured an even distribution of classes in both subsets.

A screenshot of a computer

AI-generated content may be incorrect.

**Model Architecture**

**Models Used**

Custom CNN Model built from scratch using nn.Sequential

Custom CNN Architecture

The custom CNN consists of two convolutional layers followed by ReLU activation and max pooling, feeding into a fully connected layer for classification. This model serves as a baseline to compare with more complex architectures.

self.features = nn.Sequential(

nn.Conv2d(3, 16, kernel\_size=3, stride=1, padding=1),

nn.ReLU(),

nn.MaxPool2d(kernel\_size=2, stride=2),

nn.Conv2d(16, 32, kernel\_size=3, stride=1, padding=1),

nn.ReLU(),

nn.MaxPool2d(kernel\_size=2, stride=2)

)

self.classifier = nn.Sequential(

nn.Linear(32 \* 32 \* 32, 128),

nn.ReLU(),

nn.Linear(128, num\_classes)

)

This architecture was selected to provide a lightweight model for comparison and faster training, though it might not perform as well as more advanced models.

**Training Configuration**

* **Loss Function:** CrossEntropyLoss
* **Optimizer:** Adam
* **Device:** CUDA (GPU)

**Training Process**

Training was conducted over multiple combinations of learning rates and batch sizes. For each combination, a fresh instance of the model was trained using the selected hyperparameters. The validation accuracy and loss were monitored at the end of each epoch.

**Hyperparameters**

* Epochs: 30
* Learning Rates Tested: 0.01, 0.1, 0.0001 and 0.001
* Batch Sizes Tested:32, 64 and 128

Loop Implementation

for lr in learning\_rates:

for batch\_size in batch\_sizes:

model = get\_model(num\_classes).to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=lr)

for epoch in range(num\_epochs):

model.train()

for inputs, labels in train\_loader:

inputs, labels = inputs.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

Validation was conducted using the held-out set with performance metrics including accuracy, precision, and recall. The final trained model was tested to assess generalization.

**Graph before hyperparameters:**

learning\_rate **=** 0.001,batch\_size **=** 32,num\_epochs **=** 30

A graph with lines and dots

AI-generated content may be incorrect.

**Graph after hyperparameters**

**A graph of different colored lines

AI-generated content may be incorrect.**

**Evaluation Results**

After training, the best-performing model (based on validation performance) was evaluated on the held-out test set. The goal was to measure how well the model could generalize to unseen data. The evaluation focused on core classification metrics: accuracy, precision, and recall. These metrics were calculated using the true and predicted labels of the test dataset.

The classification accuracy represents the ratio of correctly predicted samples to the total number of samples. Precision evaluates how many of the predicted positive classes were correctly labeled, while recall indicates how many of the actual positive classes were correctly identified by the model. Macro-averaging was used to ensure equal weight was given to each class, which is important in our multi-class setup with potential class imbalance.

Evaluation Metrics

* Accuracy: Measures overall correct predictions
* Precision (Macro Average): Measures average correctness among predicted classes
* Recall (Macro Average): Measures average coverage of actual classes

test\_accuracy = correct\_test / total\_test

test\_precision = precision\_score(all\_labels\_test, all\_preds\_test, average='macro', zero\_division=0)

test\_recall = recall\_score(all\_labels\_test, all\_preds\_test, average='macro', zero\_division=0)

In addition to these metrics, a confusion matrix was plotted to visualize the distribution of misclassifications across the 10 target classes. This helped identify which classes were more frequently confused and provided insights into where the model struggled.

**Insights and Visualizations**

**Grad-CAM Visualization**

To interpret model decisions, Grad-CAM (Gradient-weighted Class Activation Mapping) was applied. It highlights image regions most influential in decision-making.

* **Target Layer:** model.layer4[1].conv2
* Heatmaps were generated and overlaid on original face images.

fig, axes = plt.subplots(1, n, figsize=(15, 3))

plt.tight\_layout()

plt.show()

**Observations**

Grad-CAM outputs confirmed that the model primarily focused on facial regions (e.g., eyes, nose, and mouth), demonstrating valid learning behavior. These visual explanations helped validate that the network was not relying on irrelevant features during classification.

**Conclusion and Future Work**

**Summary**

* Both the custom CNN and ResNet18 effectively classified face images from the LFW dataset.
* ResNet18 provided higher performance due to transfer learning benefits.
* Data augmentation and normalization techniques helped enhance generalization.
* Grad-CAM aided in interpreting predictions.

**Improvement required**

* Introduce Early Stopping to prevent overfitting.
* Apply Learning Rate Scheduling for optimized training.
* Experiment with deeper models like ResNet50 or EfficientNet.
* Include additional evaluation metrics (e.g., F1-Score, AUC).

**Visuals of dataset without GramCAM**

**A person speaking into a microphone

AI-generated content may be incorrect.**

A person speaking into a microphone

AI-generated content may be incorrect.

**Visual with GramCAM**

**A person with a red and blue image

AI-generated content may be incorrect.** **A person's face with a blurry image

AI-generated content may be incorrect.** A person with glasses

AI-generated content may be incorrect.

**Classification Report:**

A screenshot of a computer

AI-generated content may be incorrect.

**Confusion martix :**

**A screenshot of a graph

AI-generated content may be incorrect.**

***Misclassified Images***

**A collage of a person

AI-generated content may be incorrect.**