MINI PROJECT

Image Classification with CNN

GROUP 1

Ewa - Marina - Luis

DATASET

CIFAR-10

PROS

- **Widely used in ML**
- **Lots of existing research and models**
- **Small image size**
    1. Easier to train and experiment with
    2. Fast prototyping and feedback on model performance
- **Balanced data**

CONS

- **Limited image resolution:** performance constrained do to limited feature details
- **Generic classes:** no real world application

PREPROCESSING

- **Pixel normalization**

  Scaled values to [0,1]

- **One hot encoding**

- **Data augmentation**

  Rotation_range=15, width_shift_range=0.1, and height_shift_range=0.1

MODEL ARCHITECTURE & METRICS

- **Sequential model**
- **Custom architecture**

Input - 32(max) - 32(max) - 32(max) - Dense(256) - Dense (softmax)

- **Metrics: accuracy, precision, recall, F1-score**
- **Activation function:** ReLu
- **Loss function:** Categorical Cross-Entropy
- **Optimizers:** Adam

OPTIMIZATION

ARCHITECHTURE

- New layers
- Filters
- Batch normalization

HYPERPARAMETERS

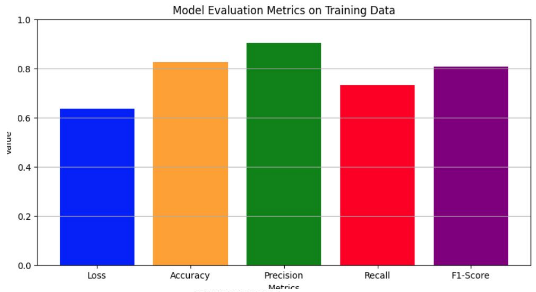
- Learning rate
- Batch size
- Epochs

OPTIMIZERS AND REGULARIZATION METHODS

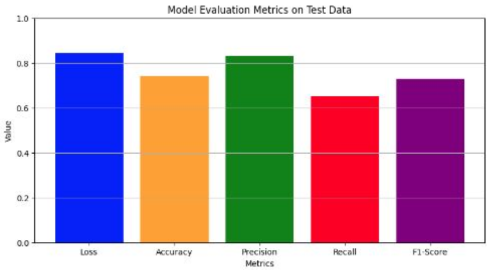
- Dropout
- Optimizers

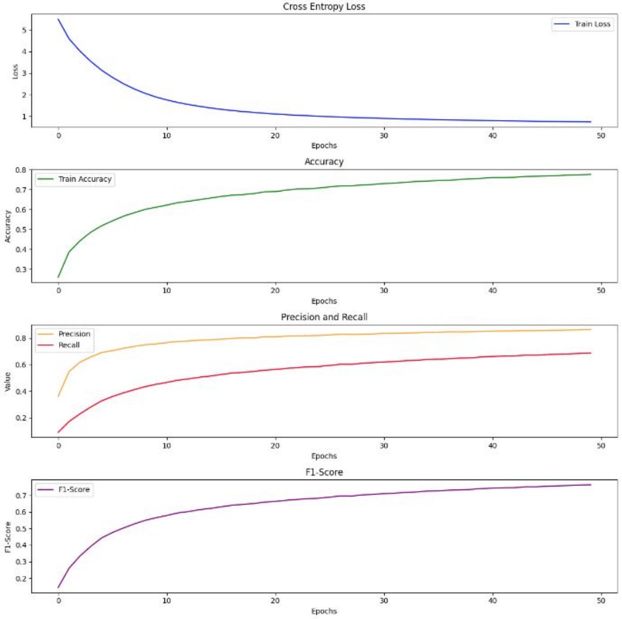
MODEL 1: RESULTS AND MODEL PERFORMANCE

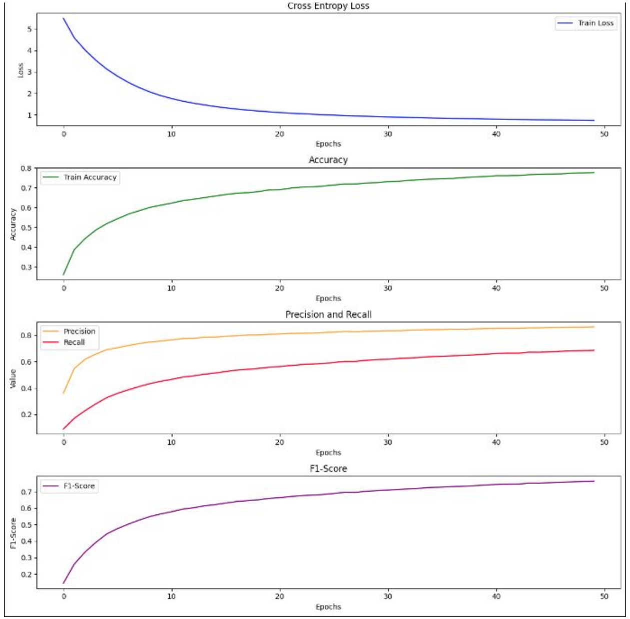
Train loss: 0.6360201835632324
Train accuracy: 0.8245000243186951
Train precision: 0.904063880443573
Train recall: 0.7314599752426147
Train F1-score: 0.8070929050445557



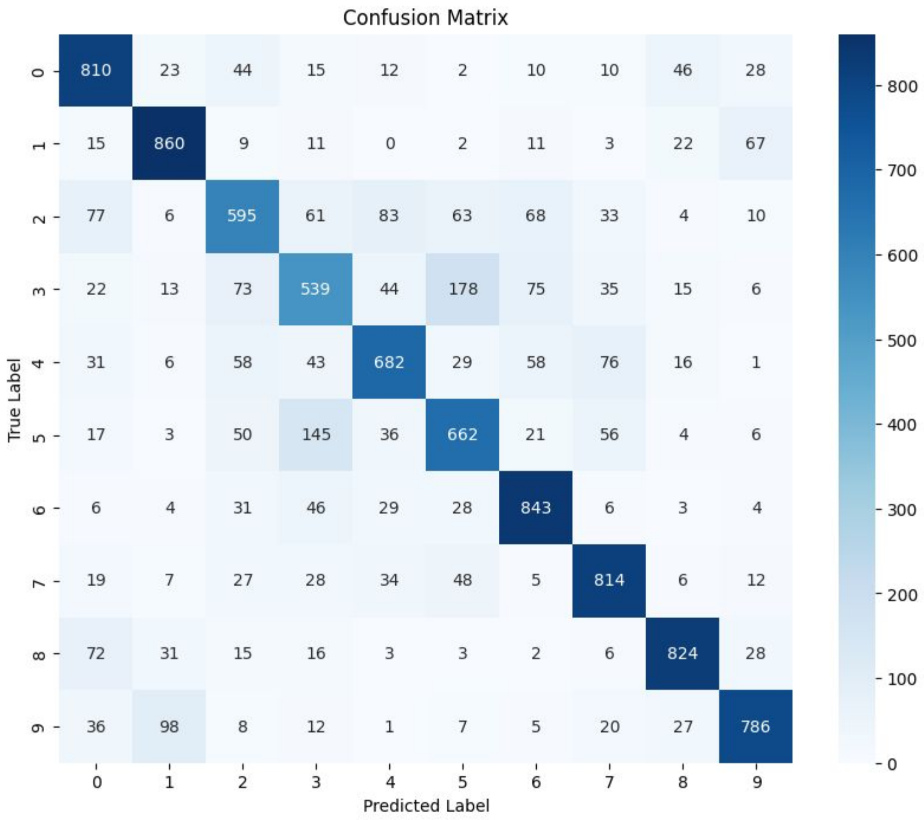
Test loss: 0.8441640734672546
Test accuracy: 0.7415000200271606
Test precision: 0.8298739194869995
Test recall: 0.6517000198364258
Test F1-score: 0.7281082272529602







MODEL 1: RESULTS AND MODEL PERFORMANCE



MODEL 2: ARCHITECTURE

Twelve convolutional layers

- Inspired in VGG16

```
model = Sequential(
    [
        tf.keras.Input(shape=input_shape),
        Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu'),
        Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
        Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu'),
        Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
        Conv2D(128, kernel_size=(3, 3), padding='same', activation='relu'),
        Conv2D(128, kernel_size=(3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
        Conv2D(256, kernel_size=(3, 3), padding='same', activation='relu'),
        Conv2D(256, kernel_size=(3, 3), padding='same', activation='relu'),
        Conv2D(256, kernel_size=(3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
        Conv2D(512, kernel_size=(3, 3), padding='same', activation='relu'),
        Conv2D(512, kernel_size=(3, 3), padding='same', activation='relu'),
        Conv2D(512, kernel_size=(3, 3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
        Flatten(),
        Dropout(0.5),
        Dense(256, activation='relu', kernel_regularizer=l2(0.001)),
        Dense(num_classes, activation='softmax')
    ]
)
```

MODEL 2: OPTIMIZATION TECHNIQUES

ARCHITECHTURE

- Filters
- Batch normalization

HYPERPARAMETERS

- Learning rate
- Batch size
- Epochs

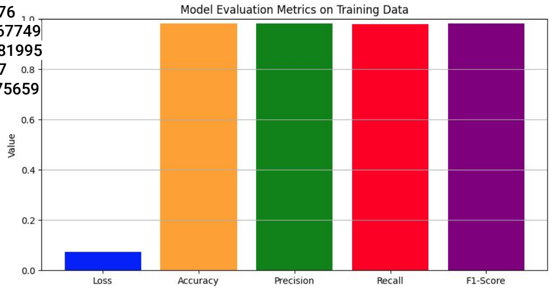
OPTIMIZERS AND REGULARIZATION METHODS

- Dropout
- Optimizers

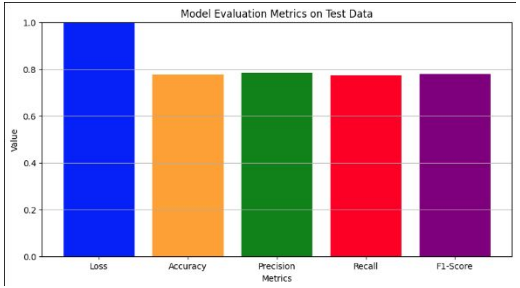
4. MODEL 2: RESULTS AND MODEL PERFORMANCE

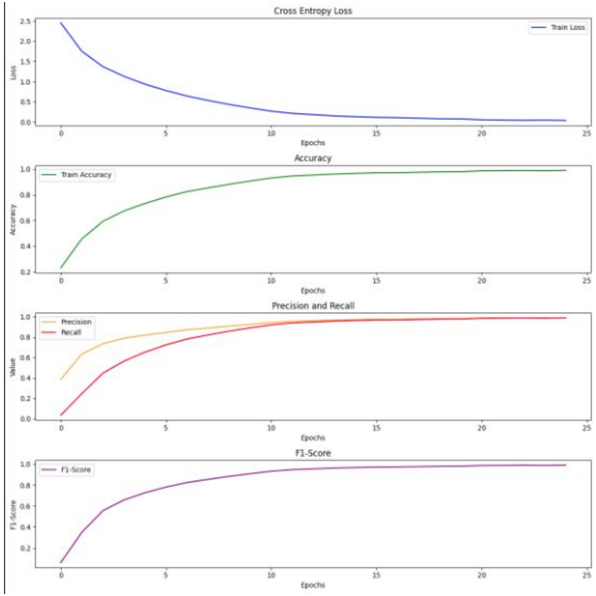
Twelve convolutional layers

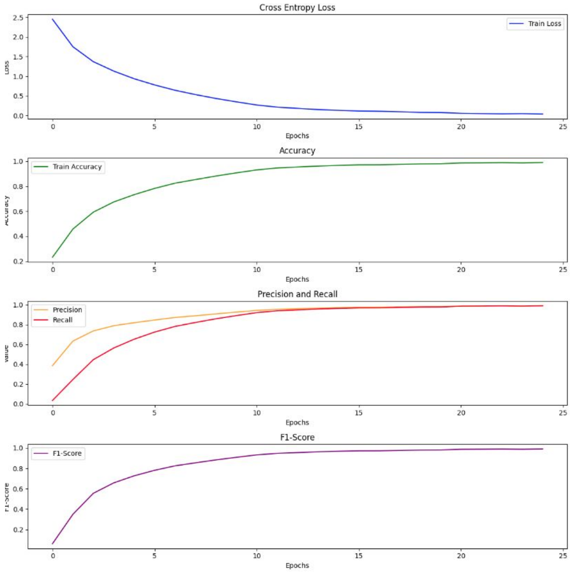
Train loss: 0.07132113724946976
Train accuracy: 0.9800800085067749
Train precision: 0.9812451004981995
Train recall: 0.979420006275177
Train F1-score: 0.9803265333175659



Test loss: 1.267404556274414
Test accuracy: 0.7767999768257141
Test precision: 0.7841208577156067
Test recall: 0.7732999920845032
Test F1-score: 0.7784469723701477

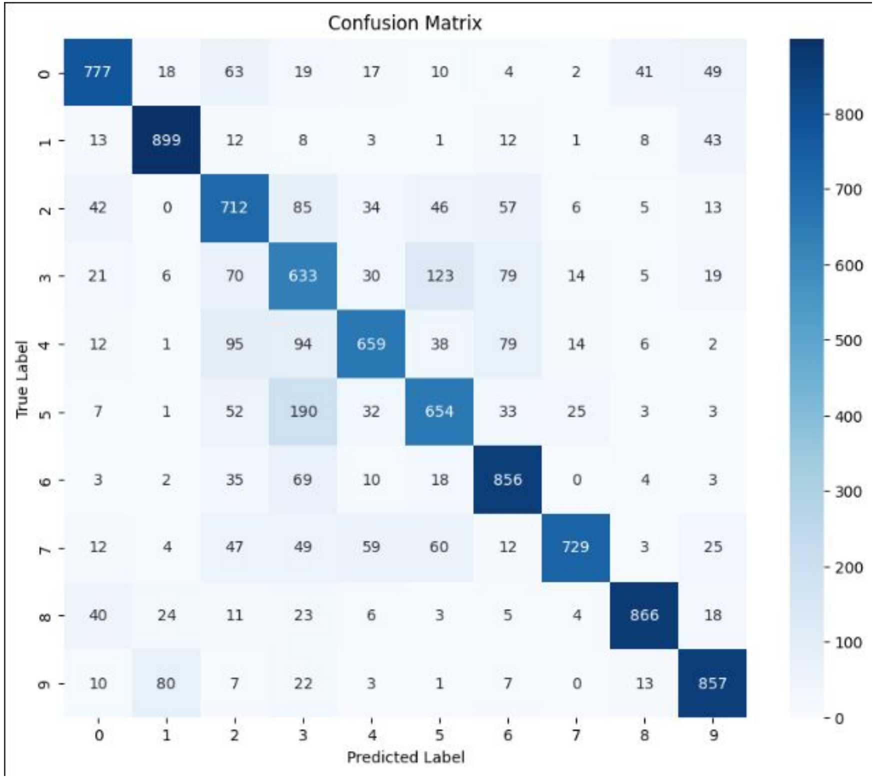






4. MODEL 2: RESULTS AND MODEL PERFORMANCE

Twelve convolutional layers



6. Transfer Learning with ResNet50 and VGG16

Transfer Learning with ResNet50 and VGG16

1. ResNet50

- **Why ResNet50?**: We chose ResNet50 because it's a deep model with strong feature extraction abilities, perfect for image tasks like ours.
- **What We Did**: We used a pre-trained ResNet50 model, kept its layers frozen, and added our own layers to classify CIFAR-10 images.
- **Results**: ResNet50 gave us better accuracy and performance than other models.

2. VGG16

- **Why VGG16?**: We also tried VGG16 because it's simple and often performs well for image classification.
- **Findings**: VGG16 didn't work as well as ResNet50, giving us lower accuracy and taking longer to train.
- **Conclusion**: We stuck with ResNet50 because it gave the best results.

7. EVALUATION

Our best model was **12-Layer Model**:

- Best training metrics (accuracy, precision, recall).
- Test Performance:
- Higher loss, lower accuracy (indicates potential overfitting).
- Overall:
- Strong training, needs improvements for generalization.

Transfer Learning vs. 12-Layer Model

- Transfer learning **outperforms** the 12-layer model in both loss and accuracy.
- Better generalization to unseen data.

Transfer learning is **more effective** for this classification task.