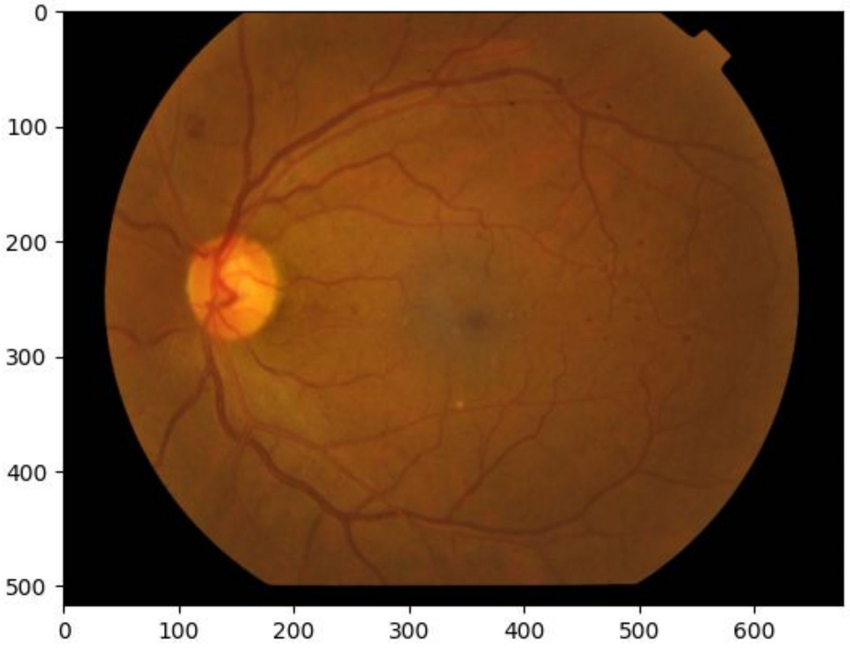
Exploring the Impact of Sample Size, Data Augmentation, and CNN Architectures on Diabetic Retinopathy Classification

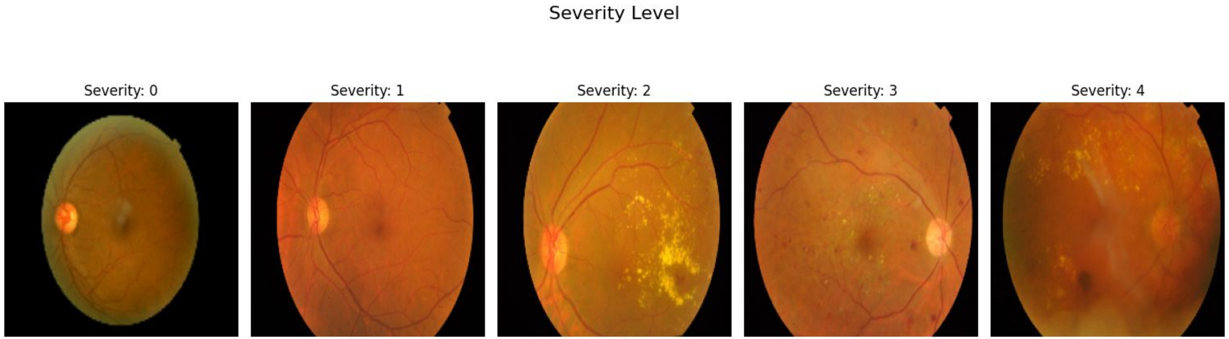


Mandy Chang
High School Division

Background

Diabetic Retinopathy and Deep Learning

- **Diabetic retinopathy (DR) is a leading cause of blindness, demanding early detection for effective treatment.**
- **Deep learning with Convolutional Neural Networks (CNNs) offers a promising approach for automated DR severity classification.**
- **This project investigates the impact of:**
  - **Sample Size**
  - **Data Augmentation (rotation, flipping, scaling, shearing)**
  - **CNN Architecture (depth, skip connections, regularization)**
- **Goal: Explore the impact of preprocessing, data augmentation, and CNN architectures on Diabetic Retinopathy Classification**



Research Questions

How does sample size, data augmentation techniques (rotation, flipping, scaling), and architectural modifications (depth, skip connections, regularization) impact model performance and generalization?

Hypothesis

"Larger sample sizes, data augmentation techniques, and deeper CNN architectures will improve the accuracy, robustness, and interpretability of a deep learning model for diabetic retinopathy severity classification."

Variables

Independent

- Sample size
- Data augmentation
- CNN architecture modifications

Dependent

- train_acc
- val_acc

Controls

- Dataset
- Preprocessing
- Training Parameters
  - Epoch
  - Batch size
  - Optimizer + Loss function

Background

1. **What are CNNs?**
    - A class of deep learning models designed for processing grid-like data (e.g., images).
    - Inspired by the biological visual cortex.
2. **Key Components**:
    - **Convolutional Layers**:
        - Apply filters to extract features (e.g., edges, textures).
        - Example: 3x3 kernel sliding over the image.
    - **Activation Functions**:
        - Introduce non-linearity (e.g., ReLU).
    - **Pooling Layers**:
        - Reduce spatial dimensions (e.g., max pooling).
    - **Fully Connected Layers**:
        - Combine features for final classification.
3. **Why CNNs for DR Classification?**
    - Automate feature extraction from retinal images.
    - Handle variations in image quality, lighting, and orientation.

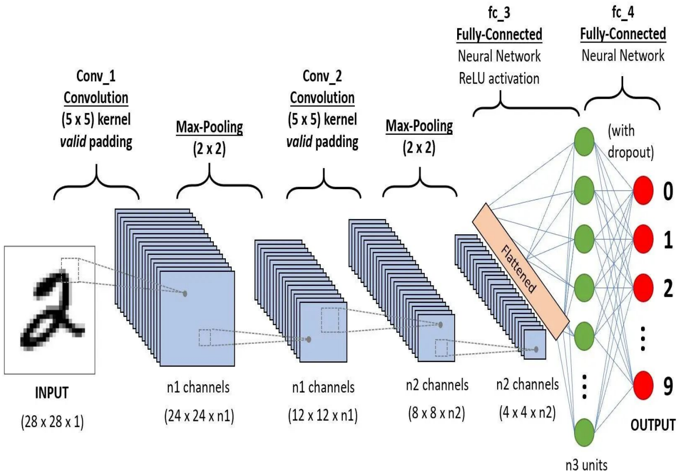


Image source: Analytics Vidhya (2020), https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/

Methodology

Dataset

We used the **Kaggle Diabetic Retinopathy Classification Dataset**, which contains retinal images labeled with five severity levels [0-4]. T

Data augmentation

Different sample sizes were applied

Preprocessing

We applied the following preprocessing and augmentation techniques:

- **Rotation**: Images were rotated between -45° and 45°.
- **Flipping**: Images were flipped horizontally and vertically.
- **Scaling**: Images were scaled between 0.8x and 1.2x.

Model Architectures

We tested three main architectures:

1. 22 layers (8 Conv2d, 8 ReLU, 4 MaxPool2d, 1 Flatten, 1 Linear).
2. 62 layers (12 Conv2d, 24 ReLU, 4 MaxPool2d, BottleneckBlock+Skip connections).
3. 456 layers (151 Conv2d, 151 ReLU, 1 MaxPool2d, BottleneckBlock+Skip connections).
4. 160 layers (67 Conv2d, 23 ReLU, 1 MaxPool2d, BottleneckBlock+Skip connections).

Regularization techniques (BatchNorm2d, Weight Decay) were applied to deeper models to prevent overfitting.

Training and Evaluation

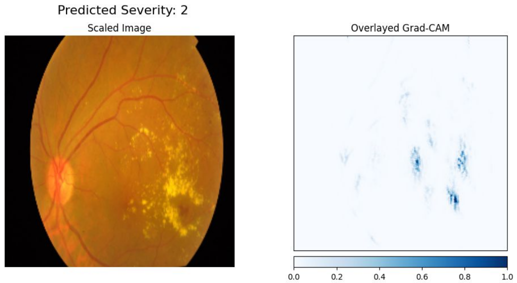
- **Training**: Models were trained for 15 epochs using the Adam optimizer and CrossEntropyLoss.
- **Evaluation**: Performance was measured using training and validation accuracy. Overfitting was assessed by the gap between training and validation accuracy.

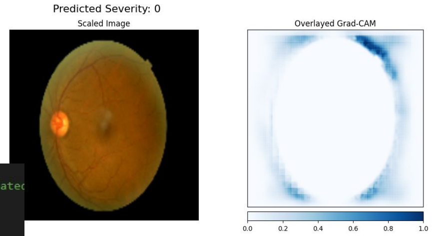
Base Model Development

- **Objective**: Establish a baseline performance for DR classification.
- **Model Architecture**:
  - 22 layers (8 Conv2d, 8 ReLU, 4 MaxPool2d, 1 Flatten, 1 Linear).
- **Training**:
  - Trained on entire available dataset (1750)
  - Trained over 15 epochs each run
  - No data augmentation or regularization.
- **Results**:
  - Training accuracy ranged of 0.69, and validation accuracy of 0.53

Challenges:

- Overfitting (|train_acc - val_acc|)
- Suboptimal accuracy (~0.7))



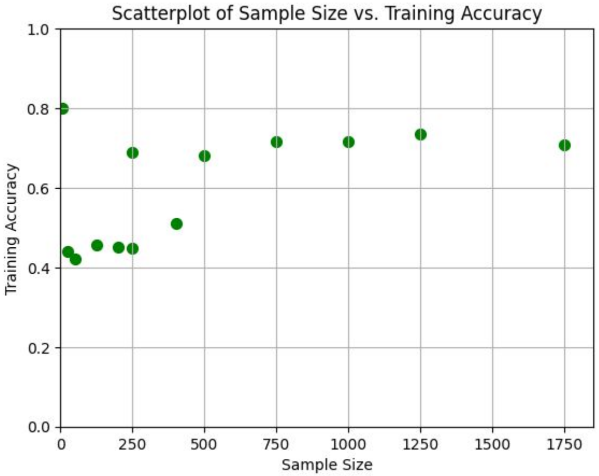


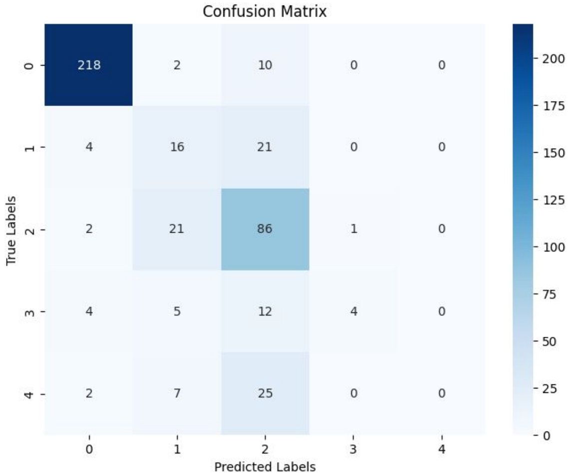

```
class RetinaClassifier(nn.Module):
    def __init__(self, kernel_size=3, num_classes=5):  # Updatec
        super().__init__()
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32,
                      kernel_size=kernel_size,
                      padding='same'),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size, padding='same'),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # img_size/2

            nn.Conv2d(32, 64, kernel_size, padding='same'),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size, padding='same'),
            nn.ReLU(),
            nn.MaxPool2d(2),  # img_size/4

            nn.Conv2d(64, 128, kernel_size, padding='same'),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size, padding='same'),
            nn.ReLU(),
            nn.MaxPool2d(2),  # img_size/8

            nn.Conv2d(128, 256, kernel_size, padding='same'),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size, padding='same'),
            nn.ReLU(),
            nn.MaxPool2d(2),  # img_size/16
        )
        self.flatten = nn.Flatten()  # (256, 16, 16) -> (256*16*
        # calculate the correct input features after feature ext
        # input_features = self.feature_extractor(torch.randn(1,
        self.classifier = nn.Sequential(
            nn.Linear(256 * (IMG_SIZE // 16) * (IMG_SIZE // 16),
        )
```

```
EPOCHS = 15
logs = {
    'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc':
}
# Earlystopping
patience = 500
counter = 0
best_loss = np.inf

for epoch in tqdm(range(EPOCHS)):
    train_loss, train_acc = train(train_loader, model, loss_fn, op
    val_loss, val_acc = test(val_loader, model, loss_fn)

    print(f'EPOCH: {epoch:04d} \
    train_loss: {train_loss:.4f}, train_acc: {train_acc:.3f} \
    val_loss: {val_loss:.4f}, val_acc: {val_acc:.3f} ')

    logs['train_loss'].append(train_loss)
    logs['train_acc'].append(train_acc)
    logs['val_loss'].append(val_loss)
    logs['val_acc'].append(val_acc)

    torch.save(model.state_dict(), "last.pth")
    # chcek improvement
    if val_loss < best_loss:
        counter = 0
        best_loss = val_loss
        torch.save(model.state_dict(), "best.pth")
    else:
        counter += 1
    if counter >= patience:
        print("Earlystop!")
        break
```

Impact of Sample Size - Data Augmentation

Trained the base model with sample sizes of 250, 500, 750, 1000, 1250, 1500, and 1750.





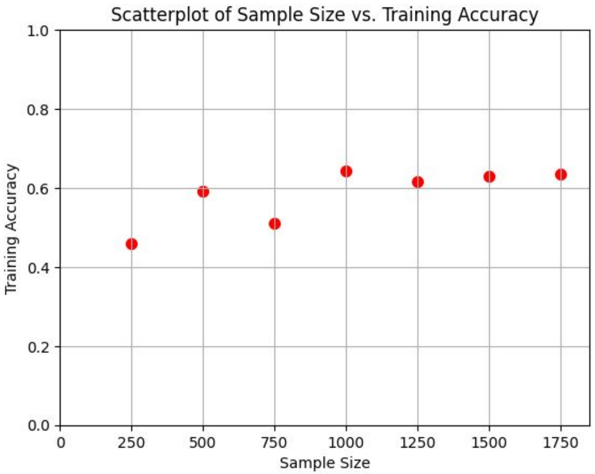
Small sample sizes (e.g., 5, 25) resulted in high training accuracy (e.g., 0.8 for 5 samples) but likely overfitting.

Larger sample sizes (e.g., 1250) showed more stable performance (e.g., 0.734 training accuracy).

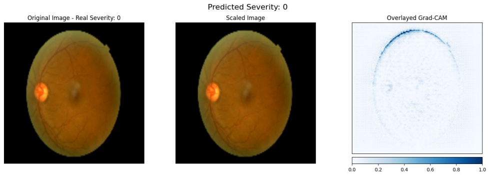
Impact of Preprocessing - Transformations

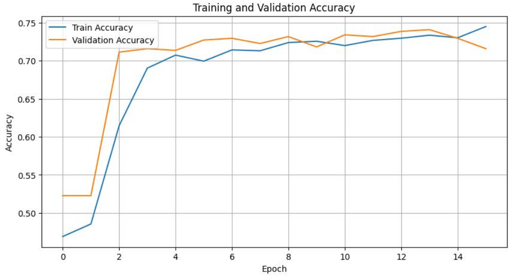
We applied transformation techniques to the training data and compared performance with and without augmentation (Table 2).

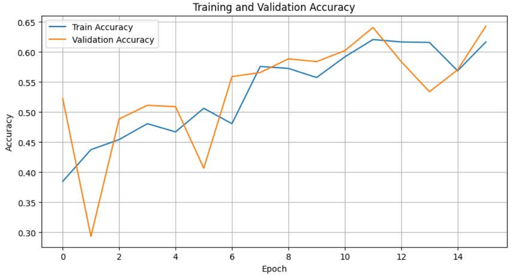
Analysis: Augmentation improved generalization for larger datasets (e.g., 1250 samples) but reduced accuracy for smaller datasets (e.g., 250 samples), likely due to insufficient base data.



Overfitting, val_acc-train_acc was minimized from 0.04 to 0.01, a 300% reduction



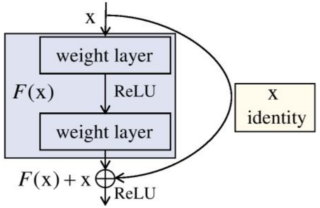


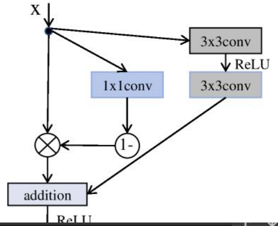


Epoch observations

Impact of Architecture - Layers

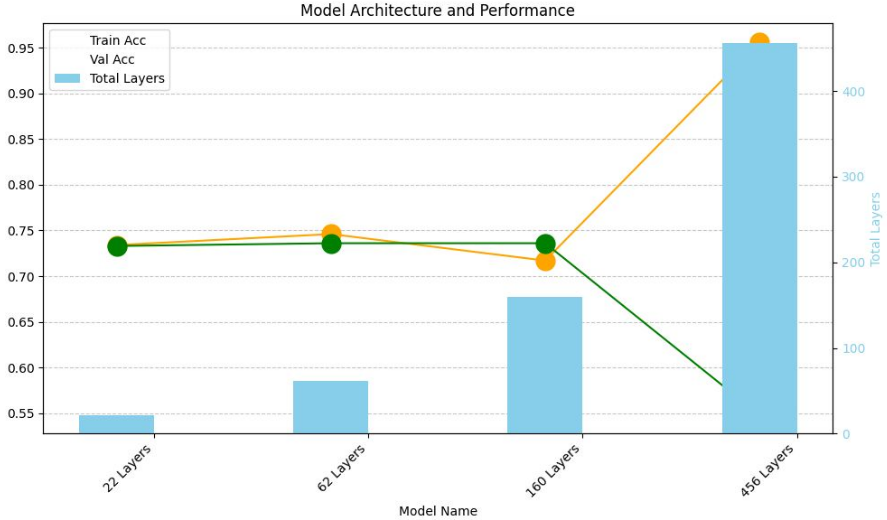
We compared the base model (22 layers) to deeper models (62, 456, and 160 layers) with and without regularization (Table 3).






```python
class BottleneckBlock(nn.Module):
    expansion = 4

    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(BottleneckBlock, self).__init__()
        mid_channels = out_channels // 4  # Reduced channel dimension
        self.conv1 = nn.Conv2d(in_channels, mid_channels, kernel_size=1, stride=1, bias=False)
        self.bn1 = nn.BatchNorm2d(mid_channels)
        self.conv2 = nn.Conv2d(mid_channels, mid_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(mid_channels)
        self.conv3 = nn.Conv2d(mid_channels, out_channels, kernel_size=1, stride=1, bias=False)
        self.bn3 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample  # Assign downsample attribute
```

Analysis: Deeper models improved accuracy but were prone to overfitting without regularization. The 62-layer model with BatchNorm2d and Weight Decay achieved the best balance between accuracy and generalization (0.746 training, 0.736 validation).




```python
class RetinaClassifier(nn.Module):
    def __init__(self, num_classes=5):
        super().__init__()
        self.inplanes = 64  # Initialize inplanes to 64, the initial number of channels

        # Initial convolutional layer (conv1)
        self.conv1 = nn.Conv2d(3, self.inplanes, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(self.inplanes)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # ResNet-152 layers
        self.layer1 = self._make_layer(BottleneckBlock, 64, 3, stride=1)  # conv2_x
        self.layer2 = self._make_layer(BottleneckBlock, 128, 8, stride=2)  # conv3_x
        self.layer3 = self._make_layer(BottleneckBlock, 256, 36, stride=2)  # conv4_x
        self.layer4 = self._make_layer(BottleneckBlock, 1024, 3, stride=2)  # conv5_x
```

Image source: Ayan & Ünlü (2024), https://doi.org/10.1016/j.engappai.2024.109890

Conclusion

This study systematically explored the impact of sample size, data augmentation, and CNN architecture on diabetic retinopathy classification. We had convincing evidence that larger datasets, data augmentation, and deeper architectures improve model performance, but careful regularization is essential to prevent overfitting. Thus, we can reject the null hypothesis and accept the alternative.

Future work will focus on advanced architectures (e.g., ResNet, EfficientNet) and ways they address class imbalance in the dataset.

The mathematical models within each feature, such as transformations or weighted changes can lead to positive and negative fluctuations. For FDA approved medical models, such as Eyeart AI, careful hypertuning is critical for each value.

References

1. Eric Li. Diabetic Retinopathy Classification Dataset, 2015, https://www.kaggle.com/competitions/diabetic-retinopathy-classification-3/data.
2. Krizhevsky, Alex, et al. "ImageNet Classification with Deep Convolutional Neural Networks." Advances in Neural Information Processing Systems, vol. 25, 2012.
3. He, Kaiming, et al. "Deep Residual Learning for Image Recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
4. Ayan, Erkan, and Mehmet Zeki Ünlü. "Explainable Deep Learning Model for Diabetic Retinopathy Detection Using Fundus Images." Engineering Applications of Artificial Intelligence, vol. 129, 2024, p. 109890, https://doi.org/10.1016/j.engappai.2024.109890.
5. Pustokhin, Denis, et al. "An Effective Deep Residual Network Based Class Attention Layer with Bidirectional LSTM for Diagnosis and Classification of COVID-19." Journal of Applied Statistics, vol. 50, 2020, pp. 1–18, https://doi.org/10.1080/02664763.2020.1849057.
6. Analytics Vidhya. "What Is the Convolutional Neural Network Architecture?" Analytics Vidhya, 22 Oct. 2020, https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/.
7. Phani Ratan. "CNN’s Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet, and More." Medium, 22 Oct. 2020, https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5.