Skin Lesion Classification

Deep Learning Application

60.001 Applied Deep Learning

Problem Background

1 in 5 people develop skin cancer

35% survival rate when the cancer **spreads to distant organs**

74% survival rate when the cancer becomes **lymph nodes**

Problem Background

99%

survival rate when skin cancer is **detected early**

Problem Statement

How might we develop an automated and reliable system to identify the presence or absence of skin cancer in individuals using limited information?

Scope & Constraints

HAM10000 - Dataset

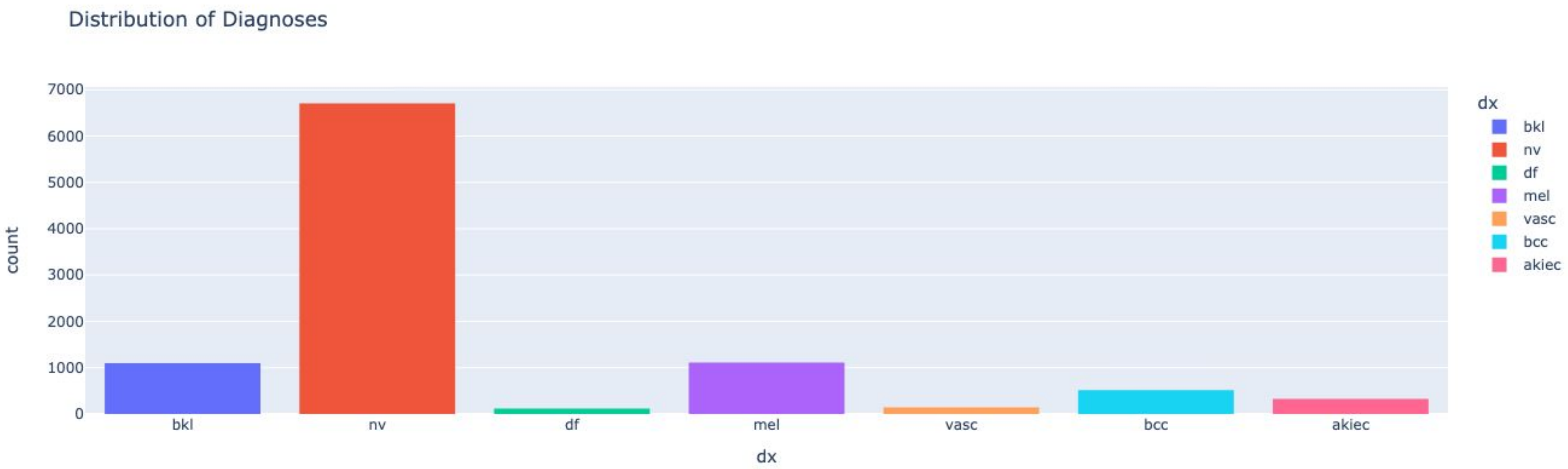
10,015

dermatoscopic images

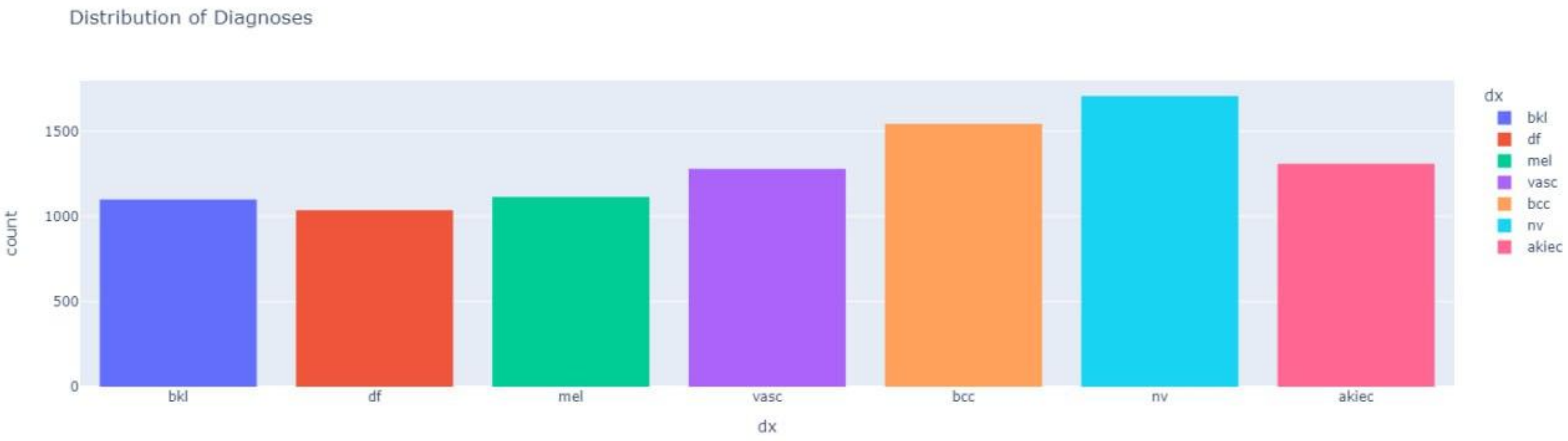
Meta-Data

skin lesion type and location
patient’s age and gender

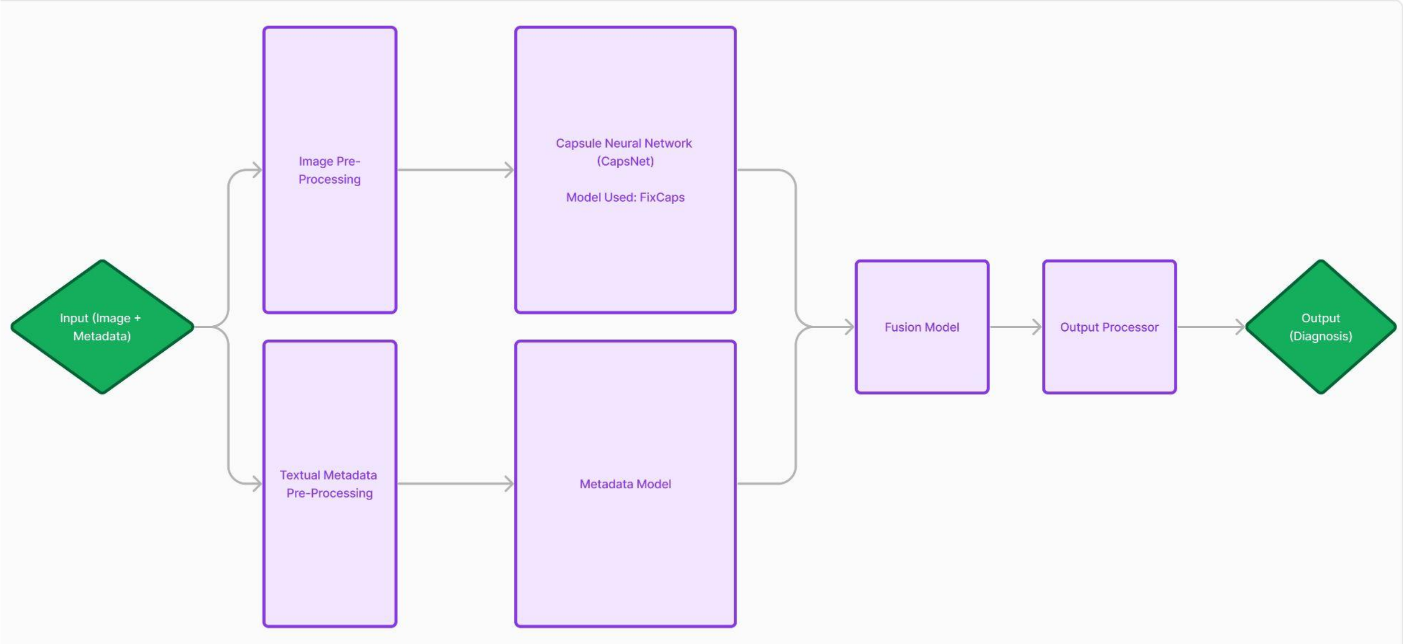
Class Distribution



Class Distribution (Data Augmentation)



Proposed Model Architecture



Base-Comparison of Models

	InceptionV3	DenseNet	ResNeXt
Test Accuracy	71.36%	71.45%	79.74%
Test Loss	1.0066	0.7242	0.7029

Related Work

Vision Transformer (VIT)

- Research paper by Yang, Luo, & Greer in 2023
- ViT processes images as a sequence of patches, utilizing transformer mechanisms to capture complex patterns and relationships between these patches
- Achieving an accuracy of **94.1% in Skin Lesion Classification**

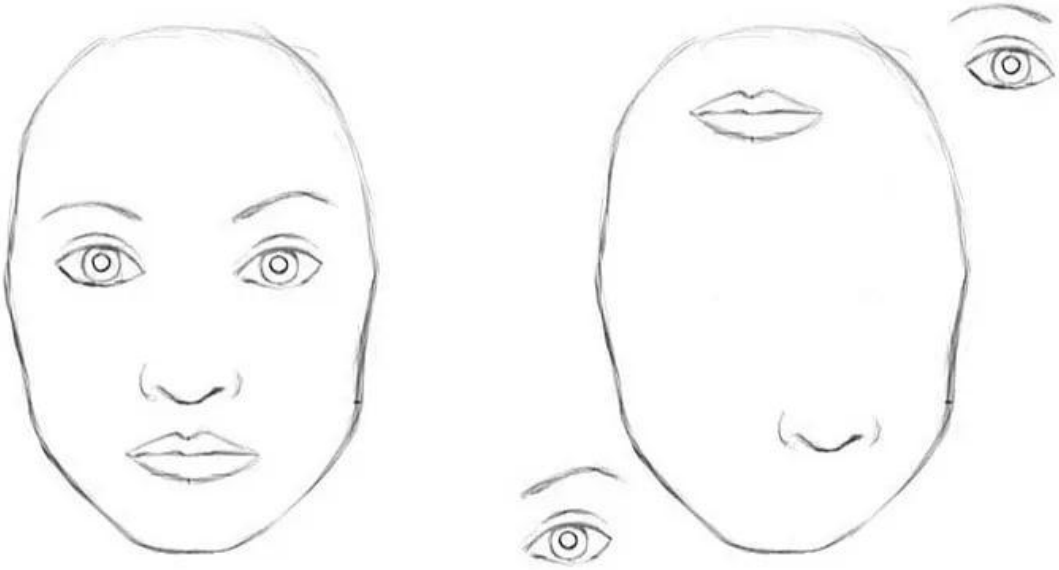
EfficientNet

- Work by Ali, Shaikh, Khan, & Laghari in 2022
- The model that they have tested with, EfficientNet B0, has a **potential accuracy of 83.02%**

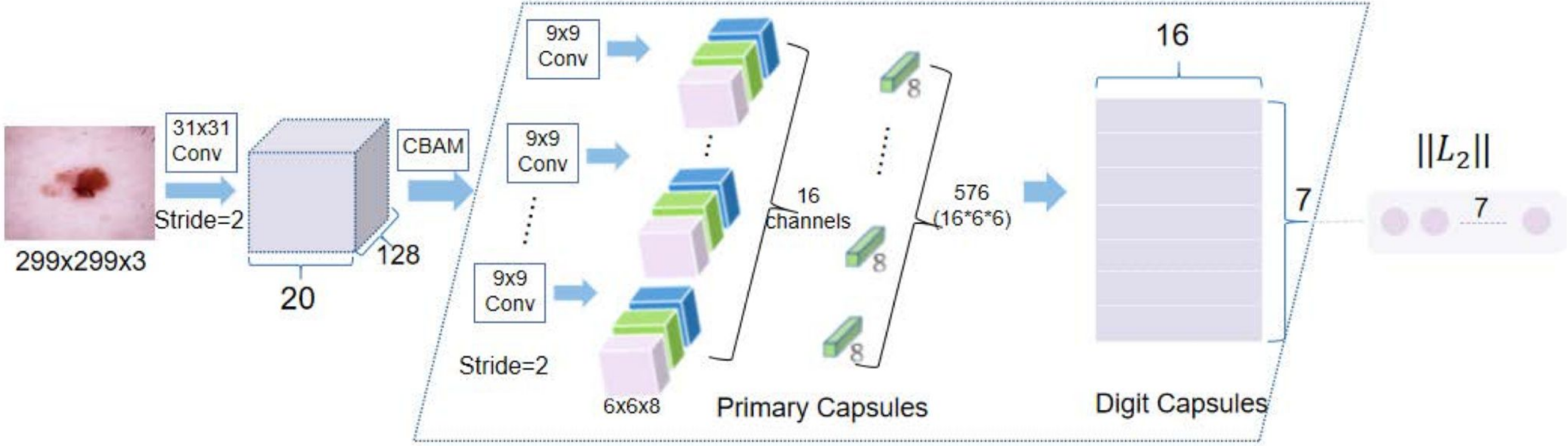
Fixed Capsule Network (FixCaps)

- Research by Chongqing Jiaotong University, 2022).
- A novel approach by encoding spatial hierarchies between features
- It has the potential to reach an **accuracy of 96.49%** with the HAM10000 dataset

Why FixCaps



CapsNet Model



```python
class FixCapsNet(nn.Module):
    def __init__(self,conv_inputs,conv_outputs,
                 primary_units,primary_unit_size,
                 output_unit_size,num_classes=7,
                 init_weights=False,mode="DS"):
        super().__init__()

        self.Convolution = make_features(cfgs[mode],f_c=conv_inputs,out_c=conv_outputs)

        self.CBAM = Conv_CBAM(conv_outputs,conv_outputs)

        self.primary = Primary_Caps(in_channels=conv_outputs,#128
                                    caps_units=primary_units,#8
                                    )

        self.digits = Digits_Caps(in_units=primary_units,#8
                                   in_channels=primary_unit_size,#16*6*6=576
                                   num_units=num_classes,#classification_num
                                   unit_size=output_unit_size,#16
                                   )
        if init_weights:
            self._initialize_weights()
```

```python
class Primary_Caps(nn.Module):
    def __init__(self, in_channels, caps_units):
        super(Primary_Caps, self).__init__()
        self.in_channels = in_channels
        self.caps_units = caps_units
        def create_conv_unit(unit_idx):
            unit = ConvUnit(in_channels=in_channels)
            self.add_module("Caps_" + str(unit_idx), unit)
            return unit
        self.units = [create_conv_unit(i) for i in range(self.caps_units)]

    def forward(self, x):
        u = [self.units[i](x) for i in range(self.caps_units)]
        u = torch.stack(u, dim=1)
        u = u.view(x.size(0), self.caps_units, -1)
        return squash(u)
```

```python
class Digits_Caps(nn.Module):
    def __init__(self, in_units, in_channels, num_units, unit_size):
        super(Digits_Caps, self).__init__()
        self.in_units = in_units
        self.in_channels = in_channels
        self.num_units = num_units
        self.W = nn.Parameter(torch.randn(1, in_channels, self.num_units, unit_size, in_units))

    def forward(self, x):
        batch_size = x.size(0)
        x = x.transpose(1, 2)
        x = torch.stack([x] * self.num_units, dim=2).unsqueeze(4)
        W = torch.cat([self.W] * batch_size, dim=0)
        u_hat = torch.matmul(W, x)
        b_ij = Variable(torch.zeros(1, self.in_channels, self.num_units, 1)).to(device)

        num_iterations = 3
        for iteration in range(num_iterations):
            c_ij = b_ij.softmax(dim=1)
            c_ij = torch.cat([c_ij] * batch_size, dim=0).unsqueeze(4)
            s_j = torch.sum(c_ij * u_hat, dim=1, keepdim=True)
            v_j = squash(s_j)#CapsuleLayer.squash
            v_j1 = torch.cat([v_j] * self.in_channels, dim=1)
            u_vj1 = torch.matmul(u_hat.transpose(3, 4), v_j1).squeeze(4).mean(dim=0, keepdim=True)
            b_ij = b_ij + u_vj1

        return v_j.squeeze(1)
```

HAM10000

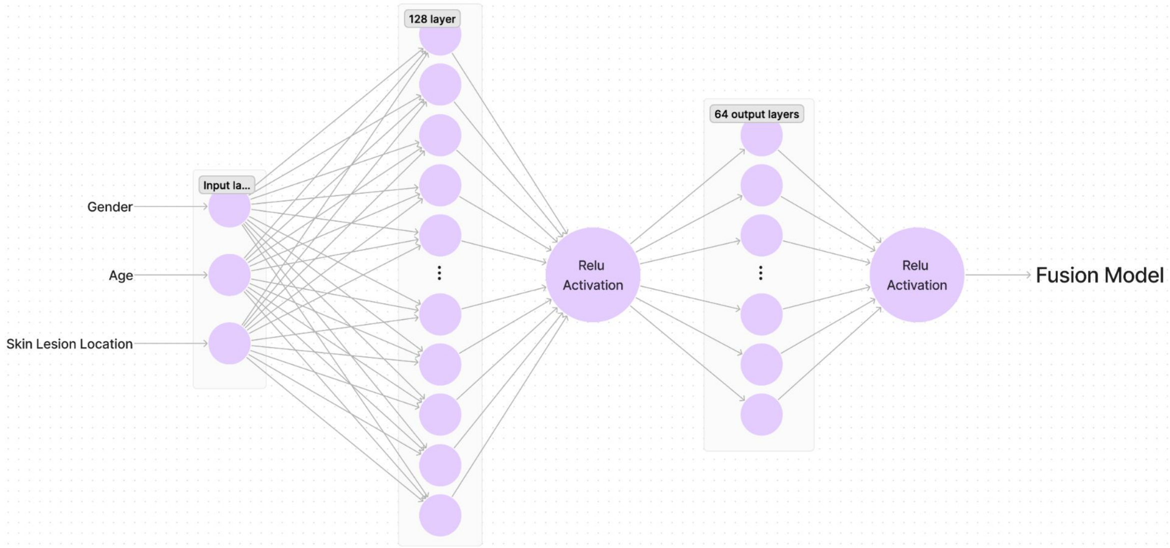
Train Accuracy	Train Loss	Test Accuracy	Test Loss
74.45%	0.1662	81.82%	0.1772

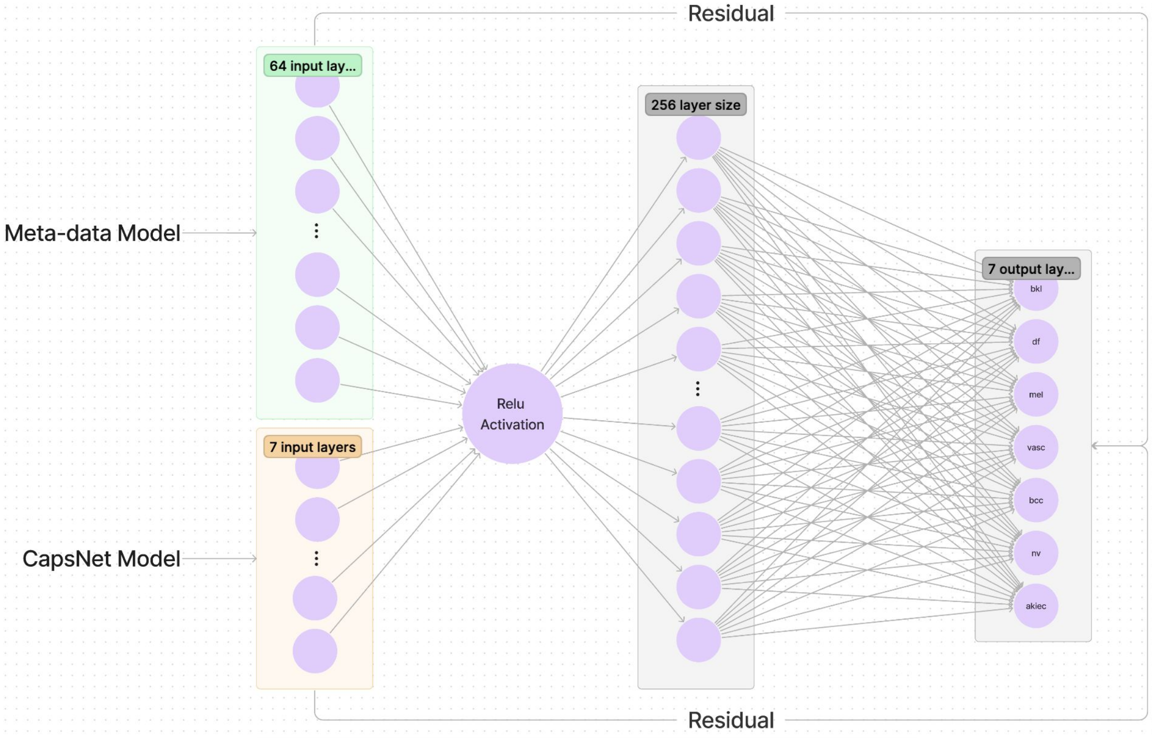
Accuracy	Precision	F1 Score	Recall
80.81%	80.61%	80.61%	80.69%

Augmented HAM10000

Train Accuracy	Train Loss	Test Accuracy	Test Loss
66.20%	0.2330	69.77%	0.24329

Metadata Model





Fusion Model

Evaluation of Proposed Fusion Model

Train Accuracy	Train Loss	Test Accuracy	Test Loss
72.43%	0.7633	68.04%	0.8370

Accuracy	Precision	F1 Score	Recall
72.46%	49.20%	69.05%	72.00%

40,000 images (No Meta-Data)

Train Accuracy	Train Loss	Test Accuracy	Test Loss
70.20%	0.2117	86.43%	0.1343

Accuracy	Precision	F1 Score	Recall
80.81%	80.61%	80.61%	80.69%

40,000 images (No Meta-Data)

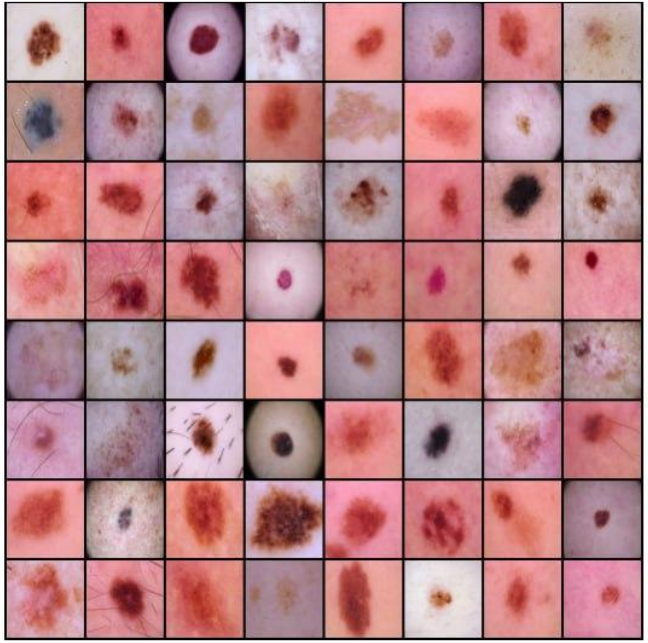
Class Distribution

	akiec	bcc	bkl	df	mel	nv	vasc
Train	5593	6286	6344	4701	6322	6433	5676
Test	1399	1572	1587	1176	1581	1609	1420

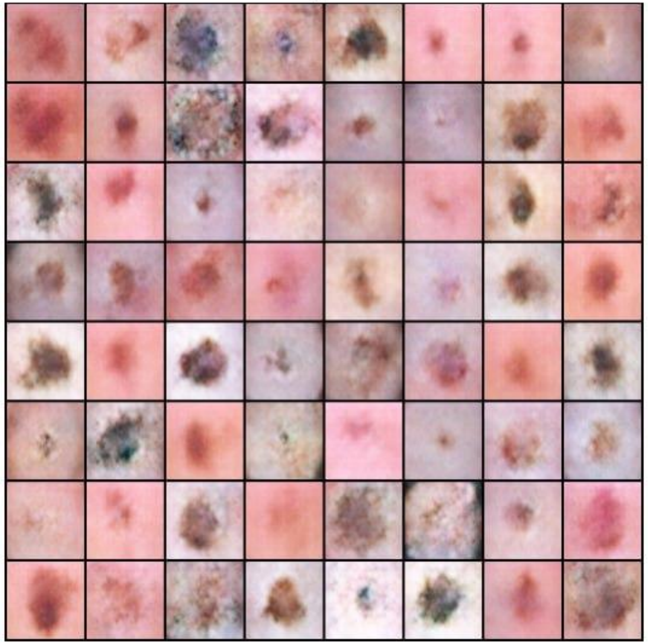
Skin Lesion GAN

Increasing Dataset size by generating realistic synthetic data samples, enhancing the model's robustness and diversity

Real Images Batch 1

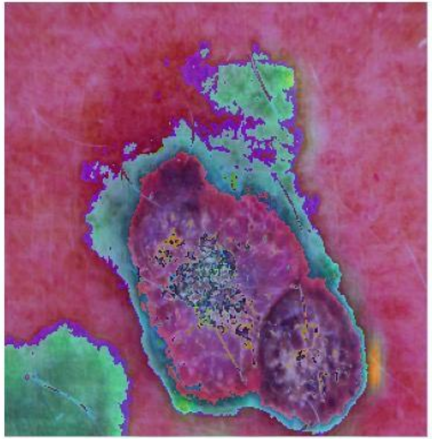


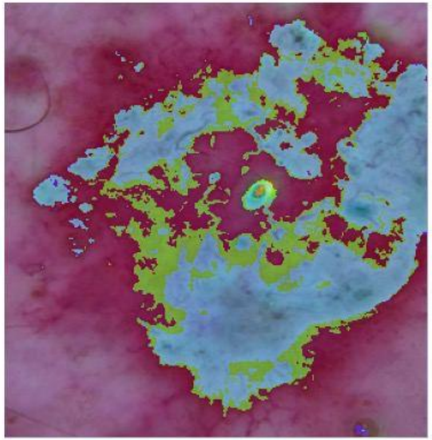
Fake Images Batch 1

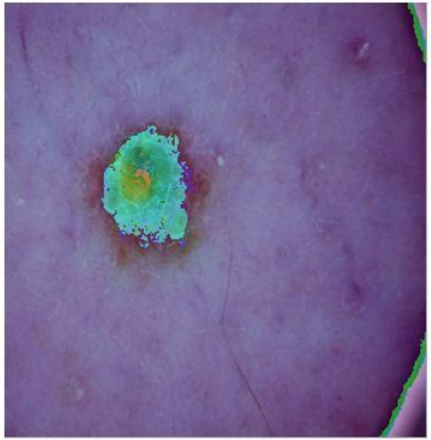


Grad-CAM

Understanding which parts of the image the model focuses on when making predictions by analysing gradients







DEMO

MEDICAL LABORATORY

THANKS

MEDICAL LABORATORY