Machine Learning for the classifaction of pigmented lesions in the skin

week 10-11 project

Pigmented Skin Lesions

Lesions that are brown, black or blue in colour under dermascopy due to melanin (most often), blood or exogenous pigment

Melanocytic:
Melanocytic nevi (benign), Melanoma

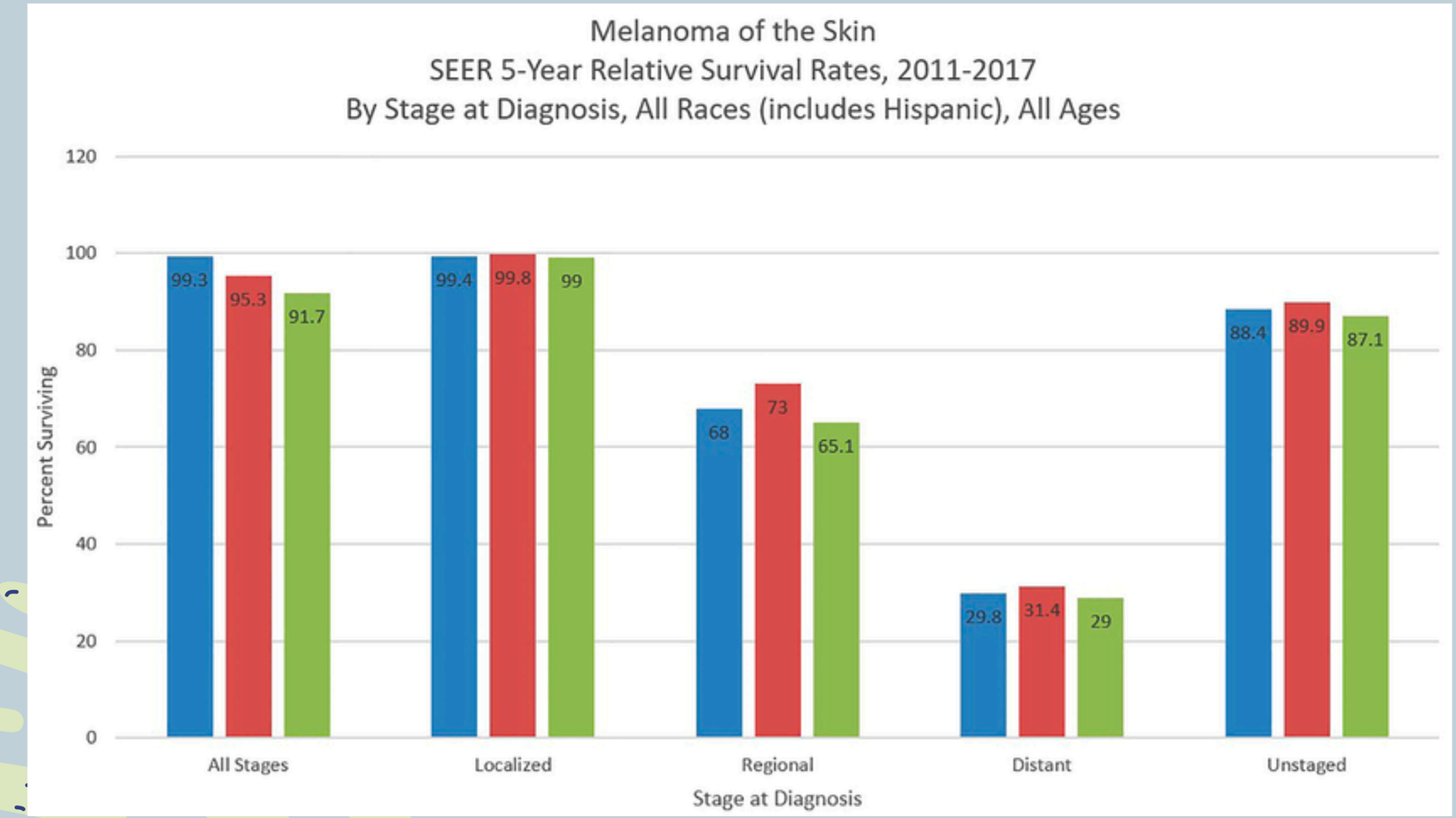
Keratinocytic:
Seborrhoeic keratosis, lentigo, pigmented actinic keratosis, pigmented basal cell and squamous cell carcinoma

Vascular:
Trauma, vasculitis, angiomas

source: dermnetnz

SO..?

Pigmented lesions range from benign lesions such as nevi, to lesions associated to a very high mortality (melanoma). As such, correct diagnosis is important for patient outcome



Diagnostic Challenges

Skin lesions are usually examined and diagnosed using a dermascope or the naked eye

Difficult for primary care physicians

Need for technologies to aid diagnosis(?)

HAM10000

(Human Against Machine)

Dataset consists of 10015 dermatoscopic images which are released as a training set for academic machine learning purposes and are publicly available through the ISIC archive.

More than 50% of lesions have been confirmed by **pathology,** while the ground truth for the rest of the cases was either follow-up, expert consensus, or confirmation by in-vivo confocal microscopy.

This benchmark dataset can be used for machine learning and for comparisons with human experts. Cases include a representative collection of all important diagnostic categories in the realm of pigmented lesions.

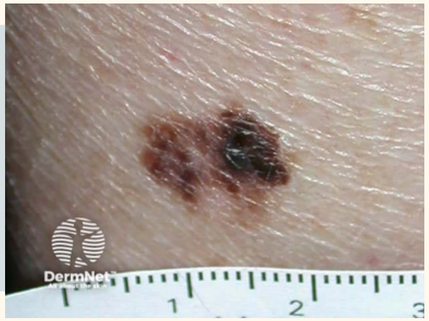
Categories in HAM10000

Melanocytic nevi

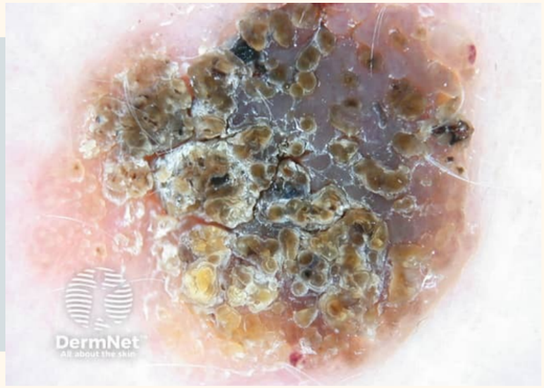
Actinic Keratoses

Basal cell carcinoma

Melanoma



Benign keratoses



Dermato-fibroma

Vascular skin lesions

My little ML Model

Goal: Creating a deep learning model to classify seven types of skin lesions from images

Overview

Technical Approach & Architecture

- Data Preparation
- Model Architecture
- Training Process
- Results and Evaluation
- Model Interpretability
- Limitations and Challenges
- Future Improvements

Data Preparation

Data Preprocessing Pipeline

- Standardization: All images resized to 224×224 pixels for consistent input to CNN
- Normalization: Pixel values normalized using ImageNet mean (0.485, 0.456, 0.406) and std (0.229, 0.224, 0.225)
- Training augmentations: RandomHorizontalFlip, RandomVerticalFlip, RandomRotation(20°), ColorJitter

```
# Use glob to find all jpg images in any subfolder of lesion_images
# The * wildcard will match both HAM10000_images_part_1 and HAM10000_imag
def create_image_path_dict(base_dir=base_skin_dir):
    """Create a dictionary mapping image IDs to file paths"""
    imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
        for x in glob(os.path.join(base_dir, 'lesion_images', '*', '*.jpg
    return imageid_path_dict
```

Data was stored as png. files in two folders and a seperate metadata csv data sheet

```
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),  # Takes an image of a
        transforms.RandomHorizontalFlip(),  # Flips the image
        transforms.RandomVerticalFlip(),    # as above, just
        transforms.RandomRotation(20),      # Rotates the ima
        transforms.ColorJitter(brightness=0.1, contrast=0.1,
        transforms.ToTensor(), #Converts a PIL Image or numpy
        #Normalizes tensor images with mean and standard devi
        # First parameter [0.485, 0.456, 0.406]: These are th
```

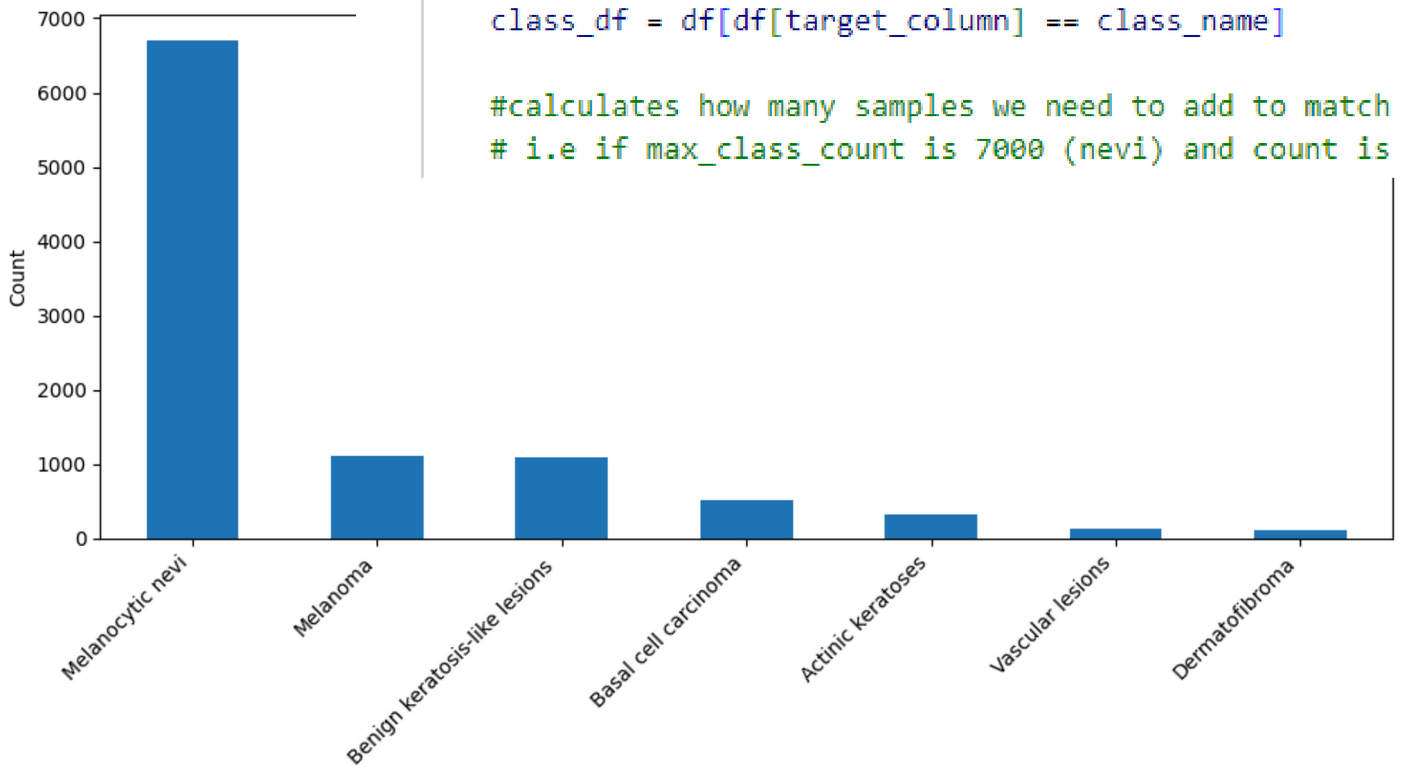
Data Preparation

Class imbalance strategy

- Oversampling minority classes to match majority class distribution

- Without oversampling, model would be biased toward the majority class

```python
# This duplicates samples from minority classes to balance the datase
def oversample_minority_classes(df, target_column):
    # Get the class counts
    class_counts = df[target_column].value_counts()
    max_class_count = class_counts.max()

    # Create a list to hold the balanced dataframe
    balanced_dfs = []

    # Loops through each class and its count (e.g., 'nevi': 7000, 'me
    for class_name, count in class_counts.items():
        # Creates a subset of the dataframe containing only rows of t
        # i.e if class_name is 'melanoma', class_df contains only mel
        class_df = df[df[target_column] == class_name]

        #calculates how many samples we need to add to match the majo
        # i.e if max_class_count is 7000 (nevi) and count is 1000 (me
```



Model Architecture

Transfer Learning Architecture

Base model: ResNet-18 pre-trained on ImageNet

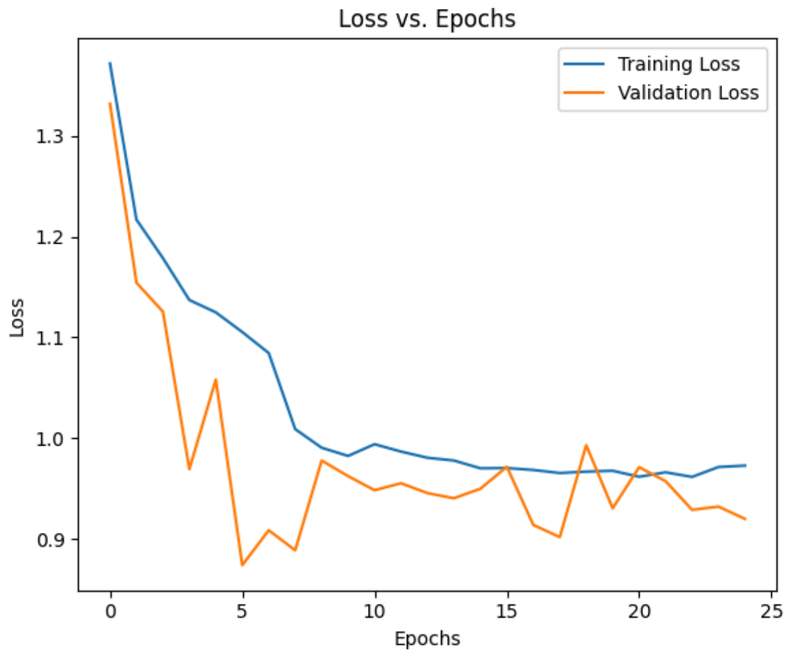
ResNet-18 is a Convolutional Neural Network (CNN)

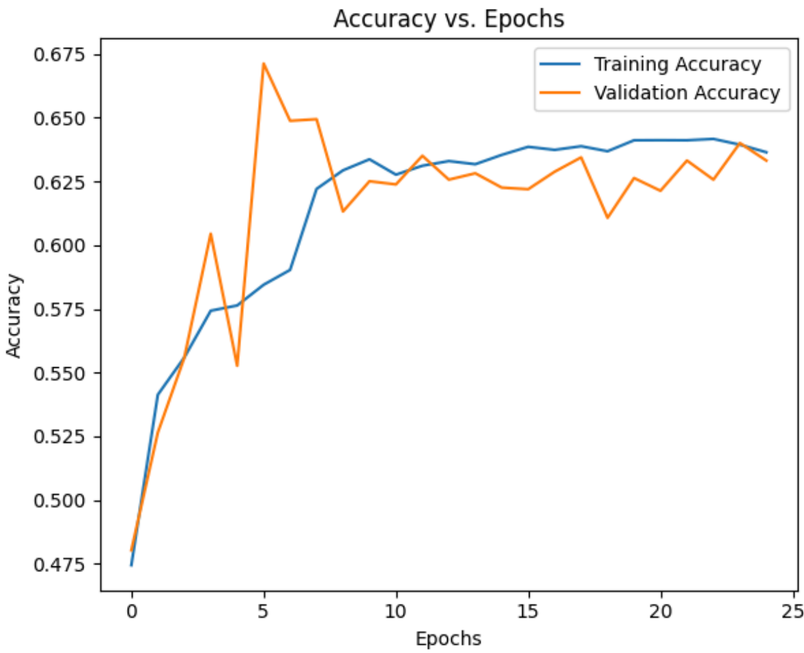
Reason: Relatively lightweight (18 layers) but still powerful and known to perfrom well with image classification

```python
def create_model(num_classes):
    # first load a pre-trained ResNet-18 model
    resnet_model = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)


    # Freeze all the parameters to prevent them from updating
    # Preserve learned features -> pre-trained model has already learned useful f
    # Prevent overfitting -> with limited data, retraining all parameters might l
    # Efficiency: Training fewer parameters requires less computation
    # Basically: use the pre-trained features but adapt the final classification
    # Freezes: All convolutional layers: The feature extraction pipeline and All
    # --> freezing everything except the final fully connected layer: We're freez
    for param in resnet_model.parameters():
        param.requires_grad = False

    # This code replaces the original fully connected layer (which was designed f
    # The last layer has 512 input features and outputs num_classes
    resnet_model.fc = nn.Sequential(
        nn.Linear(512, 256), #First fully connected layer -> 512 input features (
        nn.ReLU(), # Activation function (non-linearity), allows the network to l
        nn.Dropout(0.5), # Randomly turns off 50% of neurons during trainingto pr
        nn.Linear(256, num_classes) # Final output layer, has 256 input features,
```

Training Process

- Batch size: 16 images per batch
- Optimizer: Adam with learning rate 0.001
- Learning rate scheduler: StepLR (reduces by factor 0.1 every 7 epochs)
- Loss function: CrossEntropy (standard for multi-class classification)






```python
def create_optimizer(model, learning_rate=0.001):
    """Create an Adam optimizer for the model's FC layer"""
    return optim.Adam(model.fc.parameters(), lr=learning_rate)

#Learning Rate Scheduler: Adjusts learning rate during training
# -> Starts with lr=0.001
# -> Every 7 epochs, multiplies the learning rate by 0.1 ->Example: l
# Large steps at the beginning for faster progress, Small steps later
def create_scheduler(optimizer, step_size=7, gamma=0.1):
    """Create a learning rate scheduler"""
```

Results and Evaluation

Overall Performance Metrics

Accuracy: 66.1% on the test set

Metric breakdown:

- Weighted avg precision: 0.751
- Weighted avg recall: 0.661
- Weighted avg F1-score: 0.692

Conclusion..
Model still performs remarkably better on majority classes

Lesion Type	Precision	Recall	F1-Score	Support
Vascular lesions	0.936	0.743	0.829	1341
Dermatofibroma	0.465	0.486	0.476	220
Basal cell carcinoma	0.338	0.345	0.341	223
Melanocytic nevi	0.377	0.670	0.483	103
Benign keratosis	0.298	0.477	0.367	65
Actinic keratoses	0.347	0.893	0.500	28
Melanoma	0.149	0.783	0.250	23

```
# calculate accuracy
# accuracy_score scikit-learn function that calculates proportion of correct predictions
accuracy = accuracy_score(all_labels, all_preds)
print(f"Test accuracy: {accuracy:.4f}")

#Generate a report
# classification_report: scikit-learn function that calculates multiple metrics:
# precision (% correct positive preds) -> True Pos / (True pos + False pos)
# recall (% correct positives id'ed) -> true pos / (trus pos + false neg)
# F1: Balances precision and recall into a single score ->  2 * (precision * recall) / (preci
# support: Occurrences of each class in the test set
# target_names=class_names: Uses readable class names in the report (such as Melanoma instead

report = classification_report(all_labels, all_preds, target_names=class_names, output_dict=T
report_df = pd.DataFrame(report).transpose() # .transpose() to flip rows and columns for bett
print(report_df)
```

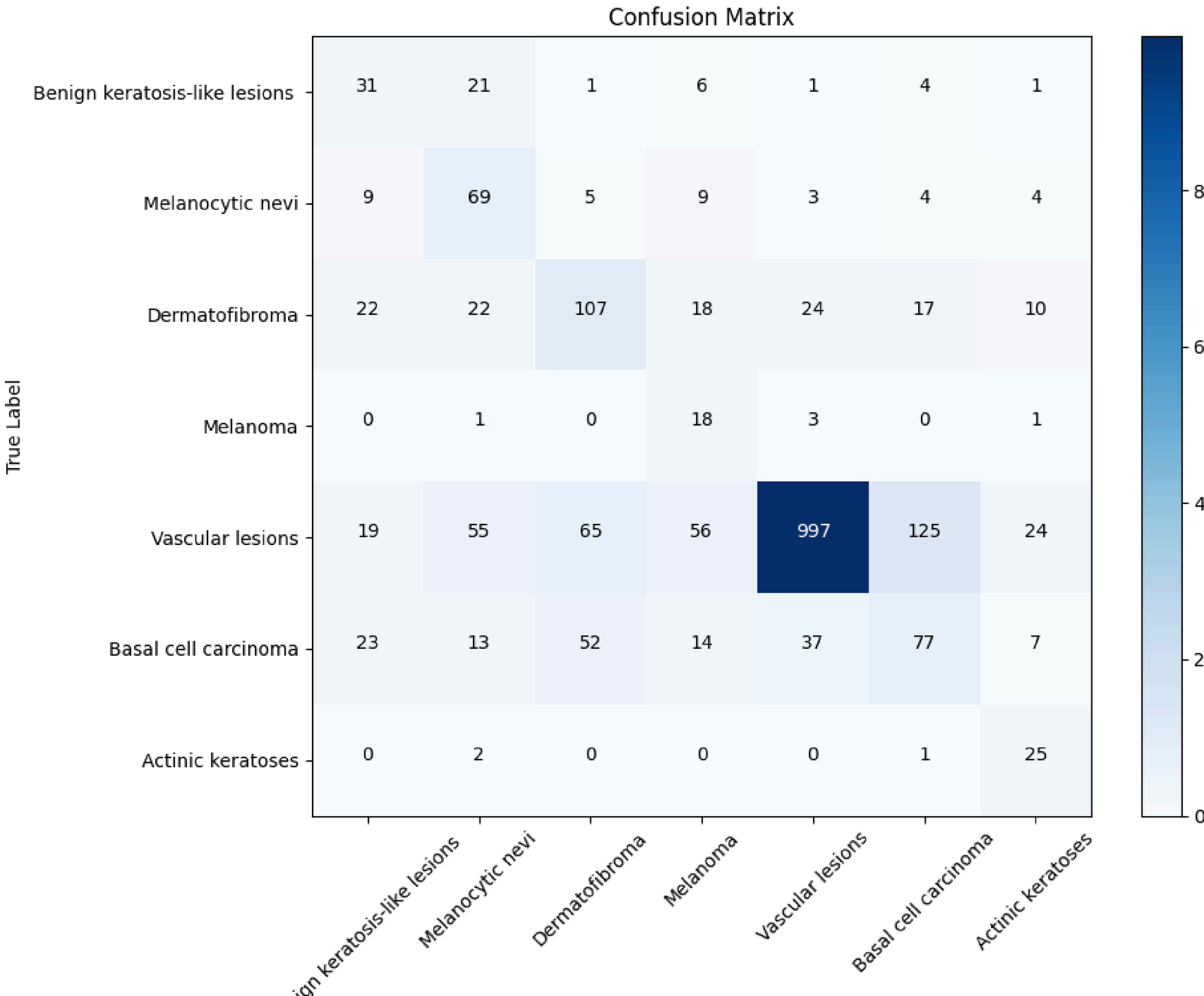
Results and Evaluation

COnfusion Matrix

...Reflects class imbalances

```
cm = confusion_matrix(all_labels, all
plt.figure(figsize=(10, 8))
plt.imshow(cm, interpolation="nearest
plt.title("Confusion Matrix")
plt.colorbar() # color scale referenc
# adding class labels:
tick_marks = np.arange(len(class_name
plt.xticks(tick_marks, class_names, r
plt.yticks(tick_marks, class_names) #
# adding value annotations:
thresh = cm.max() / 2. # calculates a
for i in range (cm.shape[0]): # i for
```

Higher sensitivity for some critical lesions (Melanoma: 78.3% recall, Actinic keratoses: 89.3% recall) But.. lower precision means many false positives (Melanoma precision only 14.9% ie many false positives)



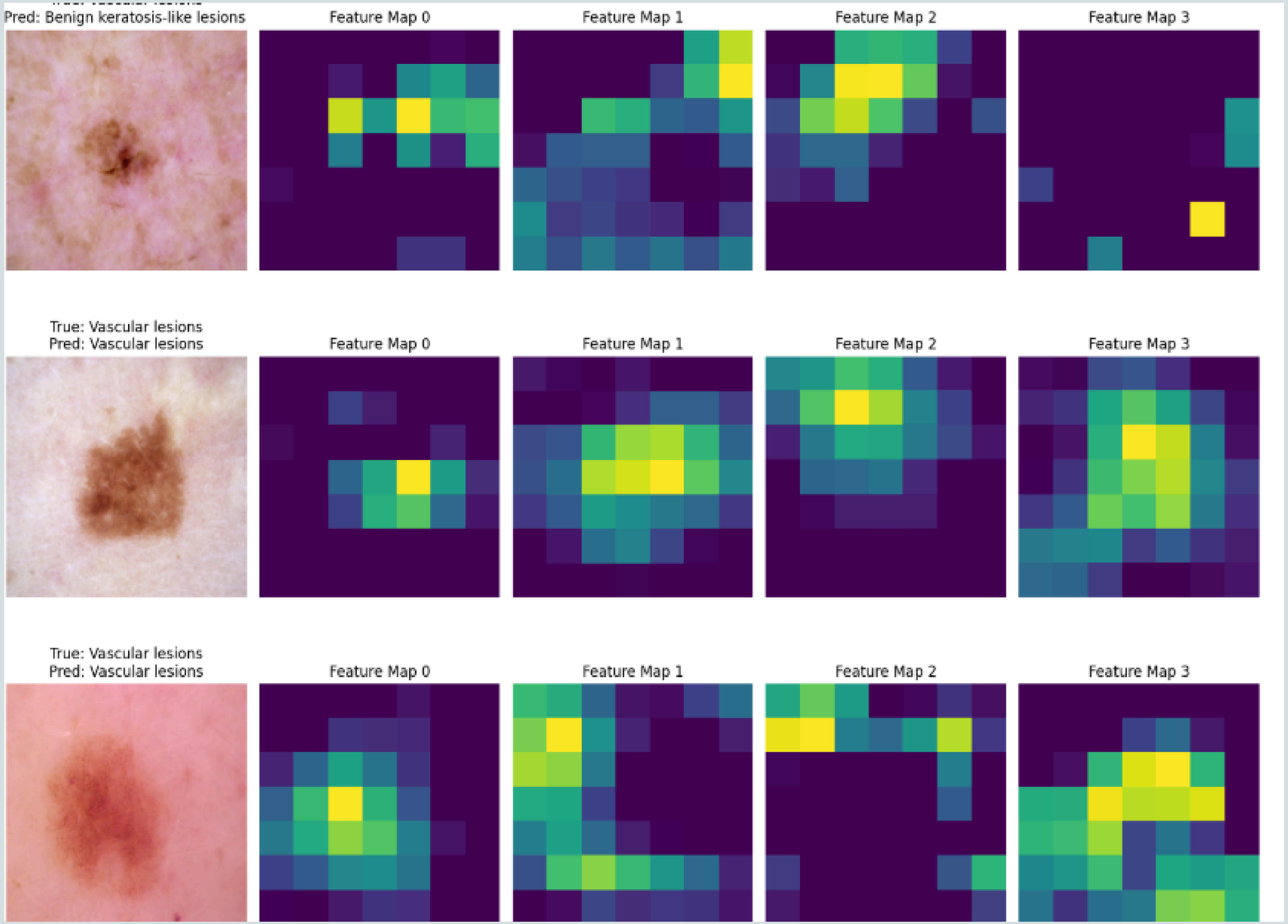
Interpretability – Feature Map Visualization

Visualization that helps understanding what the model "sees" internally..

Visualizes feature maps from an intermediate layer of the model:

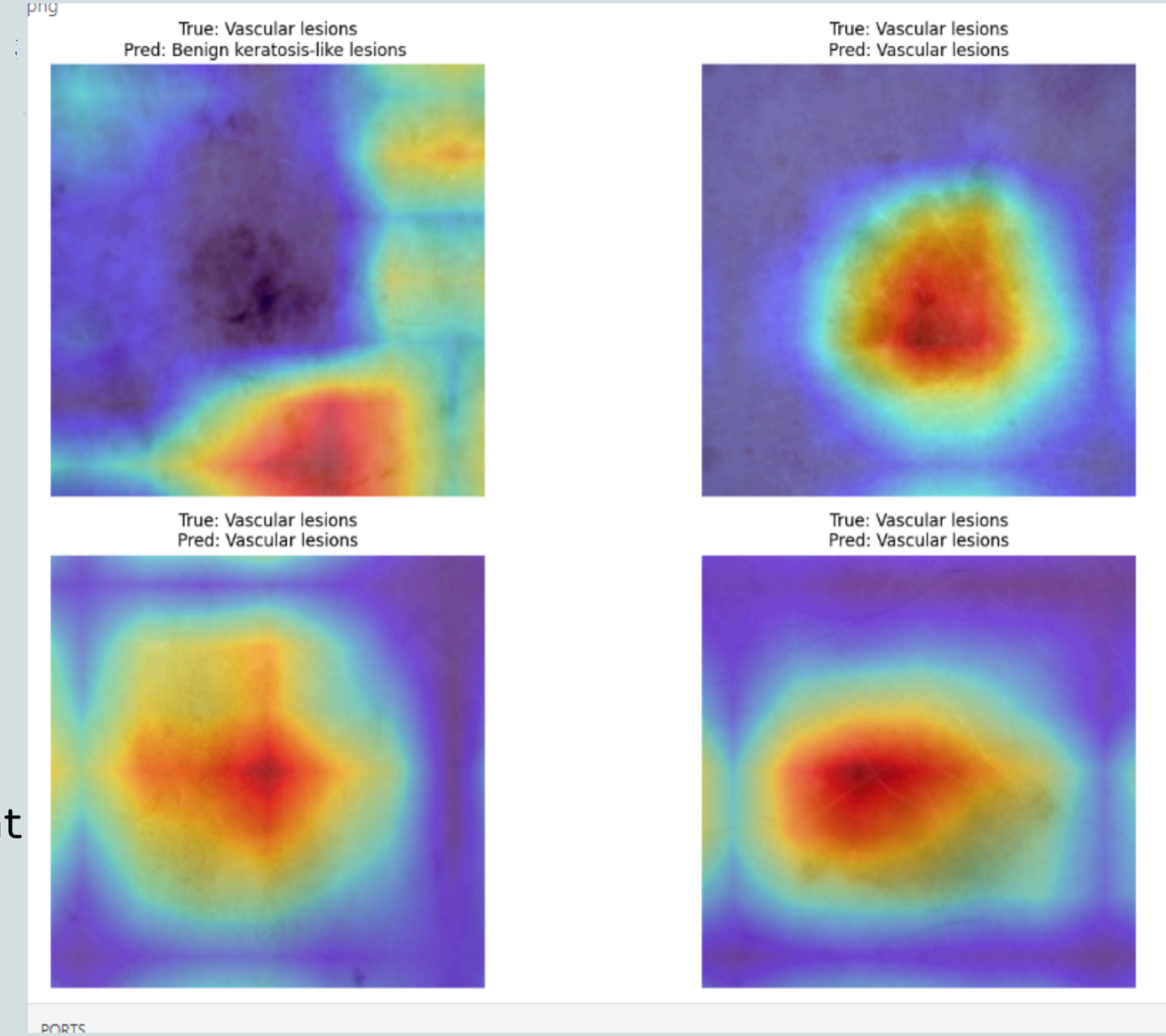
- What patterns it has learned to recognize
- Identify if it focuses on irrelevant features

Shows all features extracted, regardless of their importance to final classification



Interpretability – Grad-CAM visualization

- Grad-CAM = Gradient-weighted Class Activation Mapping
- Show which area of the lesion influenced the model's "diagnosis"
- Confirms whether the model focuses on the lesion itself rather the irrelevant background elements, shows biases
- I.e for melanoma should highloght border and color irregularity, assymmetry etc
- In case of incorrect predictions, can help highlight how it was "misled"
- Combines feature activations with the gradients flowing back from the final classification



Limitations and Challenges

- Class imbalance effects on performance
- Limited dataset size for some classes (particularly melanoma)
- Frozen model limitations (transfer learning tradeoffs)
- Computational constraints affecting model complexity
- Generalizability concerns to real-world clinical settings

Future Improvements

- Architecture enhancements: use different premade model e.g. ResNet-50
- Training improvements: Fine-tuning more layers, enhanced augmentation
- Better handling of class imbalance
- Maybe combine multiple models specialized for different lesions
- Additional data sources to enhance training

Thank you for you for your attention! (THE END)