Skin Cancer Classifier

CSCI 49369
Computational Vision

Daniel Kaijzer

The Importance of Early Skin Cancer Detection

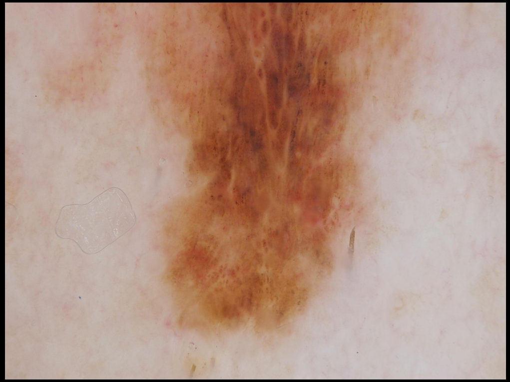
Problem Motivation:

- Skin cancer is the most common form of cancer globally
- Early and accurate detection is crucial for successful treatment
  - Localized melanoma has a 98% survival rate
- But there's limited access:
  - Diagnosis requires extensive clinical experience
  - Limited access to dermatologists in many regions

The Importance of Early Skin Cancer Detection

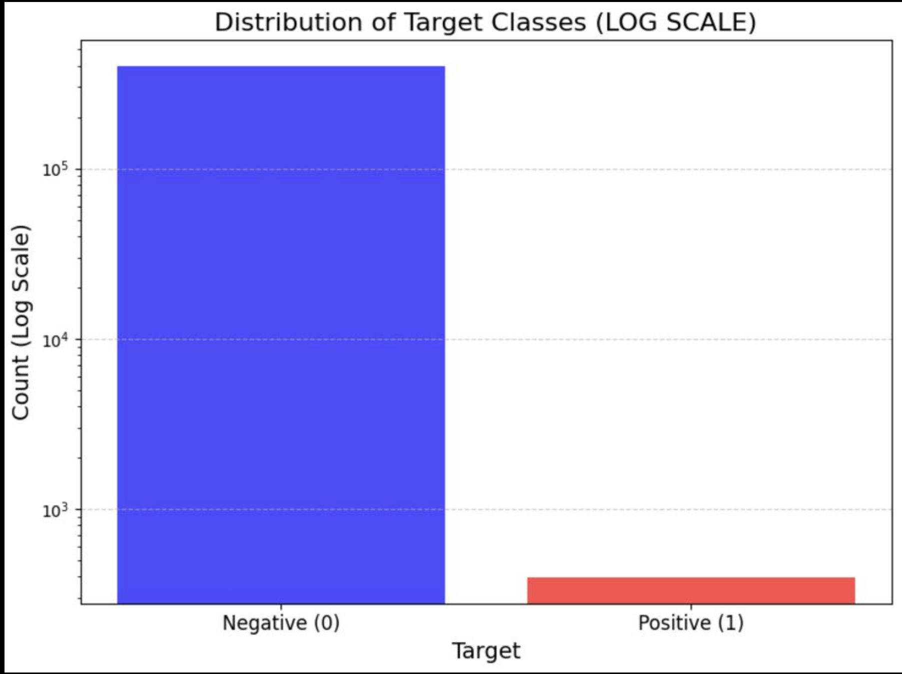
Challenges in Visual Diagnosis:

- *Skin lesion*: any area of skin that differs from surrounding skin (e.g., bump, sore)
- Subtle differences between benign and malignant lesions
  a. Benign = non-cancerous
  b. Malignant = cancerous
- Variation in lesion appearance across patients



ISIC Dataset Overview

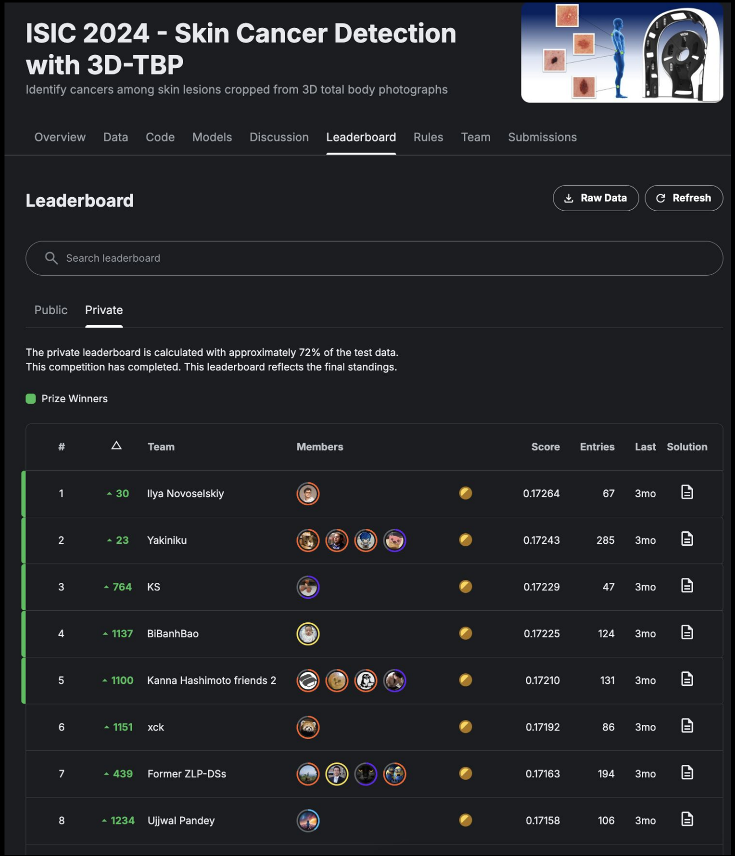
- Taken from recent ISIC 2024 Kaggle Competition
- Image quality resembles close-up smartphone photos
- Main challenge: **class imbalance**
  a. ~400k images
  b. ~400 malignant samples
- Only about **0.01% of data points are malignant cases**



Another Problem: Reliance on Tabular (CSV) data

- In Kaggle competition the winners relied heavily on tabular data

- In the real world diagnosis there usually isn't any tabular data except maybe age, sex, etc.



ISIC 2024 - Skin Cancer Detection with 3D-TBP

Identify cancers among skin lesions cropped from 3D total body photographs

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

Leaderboard

Raw Data Refresh

Search leaderboard

Public Private

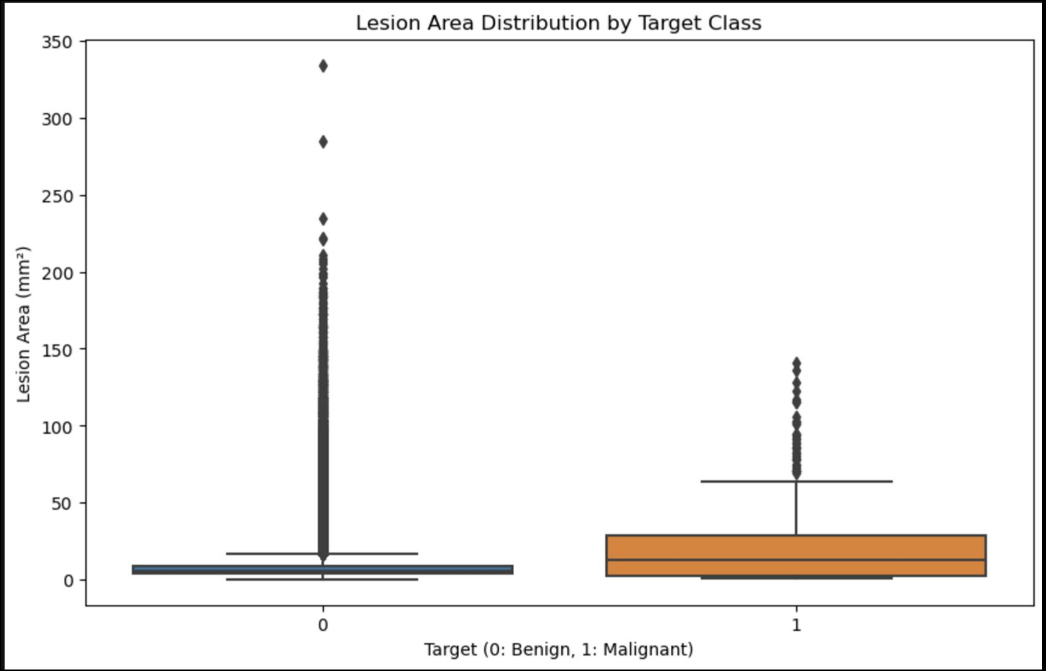
The private leaderboard is calculated with approximately 72% of the test data.
This competition has completed. This leaderboard reflects the final standings.

Prize Winners

#	△	Team	Members	Score	Entries	Last	Solution
1	30	Ilya Novoselskiy		0.17264	67	3mo	
2	23	Yakiniku		0.17243	285	3mo	
3	764	KS		0.17229	47	3mo	
4	1137	BiBanhBao		0.17225	124	3mo	
5	1100	Kanna Hashimoto friends 2		0.17210	131	3mo	
6	1151	xck		0.17192	86	3mo	
7	439	Former ZLP-DSs		0.17163	194	3mo	
8	1234	Ujjwal Pandey		0.17158	106	3mo	

Idea: Important tabular features can be approximated from images directly

- E.g., area, perimeter, minimum axis,




```
Most correlated features with target:
tbp_lv_dnn_lesion_confidence    0.054766
tbp_lv_areaMM2                  0.045139
tbp_lv_H                        0.044884
tbp_lv_perimeterMM              0.036188
tbp_lv_minorAxisMM              0.035757
tbp_lv_deltaB                   0.035069
clin_size_long_diam_mm          0.032682
tbp_lv_Hext                     0.032671
tbp_lv_B                        0.026366
tbp_lv_stdLExt                  0.026084
tbp_lv_radial_color_std_max     0.025441
tbp_lv_color_std_mean           0.024271
tbp_lv_Aext                     0.023206
tbp_lv_norm_color               0.022264
tbp_lv_A                        0.019788
tbp_lv_deltaLBnorm              0.015172
tbp_lv_Bext                     0.013711
tbp_lv_nevi_confidence          0.013341
tbp_lv_stdL                     0.012669
tbp_lv_deltaLB                  0.012237
Name: target, dtype: float64
```

Competition Metric: pAUC

- Sets a threshold for minimum true positive rate (>80%)
- False negatives should be minimized
- Good score => correctly identify positive cases while maintaining an acceptable level of false positives.
- Maximum score possible: 0.2
- Highest score in competition: 0.17264

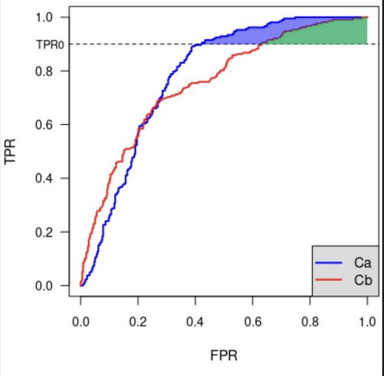
Evaluation

Primary Scoring Metric

Submissions are evaluated on **partial area under the ROC curve (pAUC)** above 80% true positive rate (TPR) for binary classification of malignant examples. (See the implementation in the notebook ISIC pAUC-aboveTPR.)

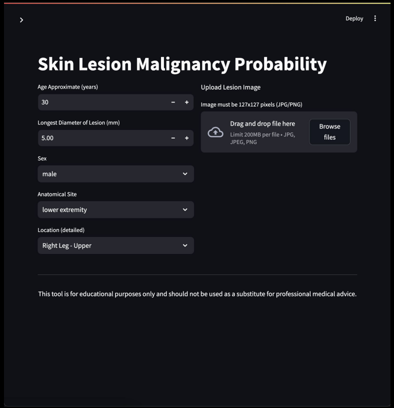
The receiver operating characteristic (ROC) curve illustrates the diagnostic ability of a given binary classifier system as its discrimination threshold is varied. However, there are regions in the ROC space where the values of TPR are unacceptable in clinical practice. Systems that aid in diagnosing cancers are required to be highly-sensitive, so this metric focuses on the area under the ROC curve AND above 80% TRP. Hence, scores range from [0.0, 0.2].

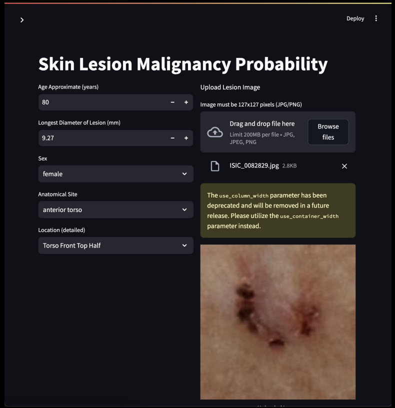
The shaded regions in the following example represents the pAUC of two arbitrary algorithms (Ca and Cb) at an arbitrary minimum TPR:

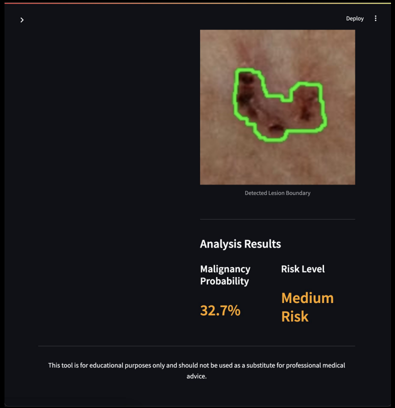


Streamlit App

- **Aim**:
  - Provide a user-friendly interface for the models
  - Reduce reliance on tabular data sets
  - Made two versions: one that uses a tabular only model and one that uses CNN + tabular model







Streamlit App

- **Challenges**:
  - Getting inference to work properly for hybrid model
  - Currently requires a file to store encodings, feature order, CNN weights and the model itself (tabular or hybrid)
  - With feature extractor and CNN program is a bit slow
    - Need to regenerate embeddings for new image

```python
st.write("About to load CNN model...")

# Load checkpoint directly to CPU
checkpoint = torch.load(
    'cnn_model_hdf5_balanced.pth',
    map_location='cpu',
    weights_only=True
)
print(checkpoint.keys())
st.write("Loaded checkpoint...")


# Load CNN model at startup
# @st.cache_resource
def load_cnn_model():
    st.write("Inside load_cnn_model...")
    try:
        # st.write("Test...")
        # print(timm.list_models())
        # Create model with no initial weights
        model = EmbeddingModel(CNN_CONFIG["pretrained_model"])
        st.write("Created EmbeddingModel...")

        # Load state dict
        # checkpoint = torch.load('cnn_model_hdf5_balanced.pth', map_location='cpu')
        # model.load_state_dict(checkpoint)
        model.load_state_dict(checkpoint['model_state_dict'])
        st.write("Loaded state dict...")

        # Set to eval mode
        model.eval()
        st.write("Set model to eval mode...")

        # Wrap in try-except
        try:
            st.write("Testing model...")
            dummy_input = torch.randn(1, 3, 224, 224)
            with torch.no_grad():
                _ = model(dummy_input)
            st.write("Model test successful!")
        except Exception as e:
            st.error(f"Model test failed: {str(e)}")
            raise e

        return model
    except Exception as e:
        st.error(f"Error loading model: {str(e)}")
        raise e

# Function to generate embeddings for a single image
def generate_embedding(img_array, model):
    transforms_fn = A.Compose([
        A.Resize(height=CNN_CONFIG["img_size"], width=CNN_CONFIG["img_size"]),
        A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ToTensorV2()
    ])
```

image_feature_extractor.py

- Goal: Reverse-engineer the ISIC csv dataset by computing geometric and color features directly from images

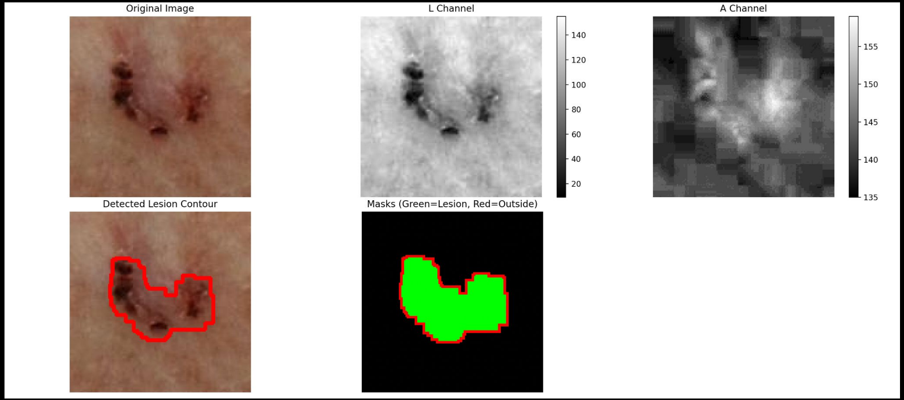
```python
def create_masks(img):
    """Create lesion and surrounding area masks."""
    # Convert to LAB
    lab_img = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    L = lab_img[:,:,0]
    A = lab_img[:,:,1]
    B = lab_img[:,:,2]  # Add B channel

    # Threshold L channel for dark spots
    thresh_L = np.percentile(L, 20)
    _, binary_L = cv2.threshold(L, thresh_L, 255, cv2.THRESH_BINARY_INV)

    # Threshold A channel for reddish areas
    _, binary_A = cv2.threshold(A, 128, 255, cv2.THRESH_BINARY)

    # Threshold B channel using mean
    _, binary_B = cv2.threshold(B, np.mean(B), 255, cv2.THRESH_BINARY)

    # Combine conditions
    binary = cv2.bitwise_and(binary_L, binary_A)

    # Use larger kernel for morphological operations
    kernel = np.ones((7,7), np.uint8)
    binary = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)
    binary = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)

    # Find contours
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    # Filter by size and compactness
    min_area = 200
    valid_contours = []
    for c in contours:
        area = cv2.contourArea(c)
        perimeter = cv2.arcLength(c, True)
        if area > min_area:
            compactness = 4 * np.pi * area / (perimeter * perimeter)
            if compactness > 0.15:
                valid_contours.append(c)

    if not valid_contours:
        raise ValueError("No valid lesion contours found")

    # Get darkest contour
    contour = min(valid_contours, key=lambda c: np.mean(L[cv2.drawContours(
        np.zeros_like(L), [c], -1, 255, cv2.FILLED) > 0]))

    # Create masks
    mask = np.zeros_like(L)
    cv2.drawContours(mask, [contour], -1, 255, -1)
    mask_bool = mask > 0

    # Create dilated mask for outside region using larger kernel
    kernel = np.ones((5,5), np.uint8)
    dilated_mask = cv2.dilate(mask, kernel, iterations=1)
    outside_mask = (dilated_mask > 0) & (~mask_bool)

    return contour, mask_bool, outside_mask
```

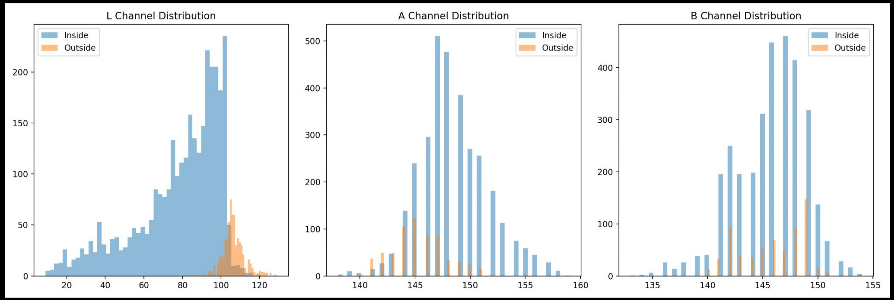
```python
def calculate_shape_features(contour, mm_per_pixel):
    """Calculate shape-related features."""

    rect = cv2.minAreaRect(contour)
    (x, y), (width, height), angle = rect

    # Calculate area and perimeter
    area_pixels = cv2.contourArea(contour)
    area_mm2 = area_pixels * (mm_per_pixel ** 2)
    perimeter_pixels = cv2.arcLength(contour, True)
    perimeter_mm = perimeter_pixels * mm_per_pixel

    # Calculate moments for axis and eccentricity
    moments = cv2.moments(contour)

    if moments['m00'] == 0:
        return None

    # Central moments
    mu20 = moments['mu20'] / moments['m00']
    mu02 = moments['mu02'] / moments['m00']
    mu11 = moments['mu11'] / moments['m00']

    # Calculate eigenvalues for axes
    delta = np.sqrt((mu20 - mu02)**2 + 4*mu11**2)
    major_axis = 2 * np.sqrt(2 * (mu20 + mu02 + delta)) * mm_per_pixel
    minor_axis = 2 * np.sqrt(2 * (mu20 + mu02 - delta)) * mm_per_pixel

    # Calculate eccentricity
    lambda1 = (mu20 + mu02 + delta) / 2
    lambda2 = (mu20 + mu02 - delta) / 2
    eccentricity = np.sqrt(1 - (lambda2 / lambda1)) if lambda1 != 0 else 0

    area_perim_ratio = (perimeter_mm ** 2) / (area_mm2)

    return {
        'tbp_lv_areaMM2': area_mm2,
        'tbp_lv_perimeterMM': perimeter_mm,
        'tbp_lv_minorAxisMM': minor_axis, # Axis of least second moment
        'tbp_lv_eccentricity': eccentricity,
        'tbp_lv_area_perim_ratio': area_perim_ratio
    }
```

```python
def calculate_color_features(lab_img, mask_bool, outside_mask):
    """Calculate color-related features."""
    # Split LAB channels
    L, A, B = cv2.split(lab_img)

    # Normalize L to 0-100 range, A and B to -128 to +127
    L_raw = L
    L = L * (100/255)
    A = A - 128
    B = B - 128

    # Calculate means for inside lesion
    L_in = np.mean(L[mask_bool])
    A_in = np.mean(A[mask_bool])
    B_in = np.mean(B[mask_bool])

    # Calculate means for outside lesion
    L_ext = np.mean(L[outside_mask])
    A_ext = np.mean(A[outside_mask])
    B_ext = np.mean(B[outside_mask])

    # Calculate deltas
    deltaL = L_in - L_ext
    deltaA = A_in - A_ext
    deltaB = B_in - B_ext

    # Calculate deltaLBnorm
    deltaLBnorm = np.sqrt(deltaL**2 + deltaB**2)

    # Calculate standard deviations
    # stdL_in = np.std(L[mask_bool])
    # stdL_ext = np.std(L[outside_mask])

    # Calculate Hue (degrees)
    H_in = np.degrees(np.arctan2(B_in, A_in))
    H_in = H_in + 360 if H_in < 0 else H_in

    H_ext = np.degrees(np.arctan2(B_ext, A_ext))
    H_ext = H_ext + 360 if H_ext < 0 else H_ext

    # Calculate Chroma
    C_in = np.sqrt(A_in**2 + B_in**2)
    C_ext = np.sqrt(A_ext**2 + B_ext**2)
```

image_feature_extractor.py

- Some features were easier to calculate than others
- Challenging to guess how people calculated certain metrics
- Subtle changes in one feature can throw off other features
- All calculations reliant on good masking and segmentation
- Using LAB color space has been tricky (because of reverse engineering formulas)






```
tbp_lv_areaMM2:
  Calculated: 42.064574507290494
  Original:   40.9645336836179
  Difference: 1.1000408236725931

tbp_lv_perimeterMM:
  Calculated: 34.03888383494147
  Original:   32.5980153581636
  Difference: 1.4408684767778723

tbp_lv_minorAxisMM:
  Calculated: 6.15927655266766
  Original:   6.67713322051584
  Difference: 0.5178566678481795

tbp_lv_eccentricity:
  Calculated: 0.812694889109181
  Original:   0.76384134918667
  Difference: 0.04885353992251096

tbp_lv_area_perim_ratio:
  Calculated: 27.54445103272891
  Original:   25.9402587979666
  Difference: 1.604192234762312

tbp_lv_L:
  Calculated: 30.703402212481734
  Original:   23.3950878565758
  Difference: 7.308314355905935

tbp_lv_Lext:
  Calculated: 41.76775057849226
  Original:   34.7843406576673
  Difference: 6.983409920824961

tbp_lv_A:
  Calculated: 20.484032561051972
  Original:   18.0933675045165
  Difference: 2.3906650565354717

tbp_lv_Aext:
  Calculated: 17.309006211180126
  Original:   13.0547724013607
  Difference: 4.254233809819427

tbp_lv_B:
  Calculated: 17.676581089542893
  Original:   19.0904579402873
  Difference: 1.4138768507444084

tbp_lv_Bext:
  Calculated: 17.874223602484474
  Original:   21.211776263998
  Difference: 3.337552661513527

tbp_lv_C:
  Calculated: 27.05655390062494
  Original:   26.3023864320972
  Difference: 0.75416746852774
```

Models

Tabular-only model

- **Best mean Cross Validation pAUC: ~0.1587**
- **Best test set pAUC: ~0.1695**
- Trains fast, usually less than minute
- Best tabular-only model on Kaggle has pAUC of ~0.166 (on larger hidden test set)
  - Mine has about 40% less features and trains in about half the time
  - Some features were unfeasible to reproduce without knowing methodology
- An ensemble of LightGBM models
  - 5 LGBM + VotingRegressor

Hybrid Model (CNN + Tabular)

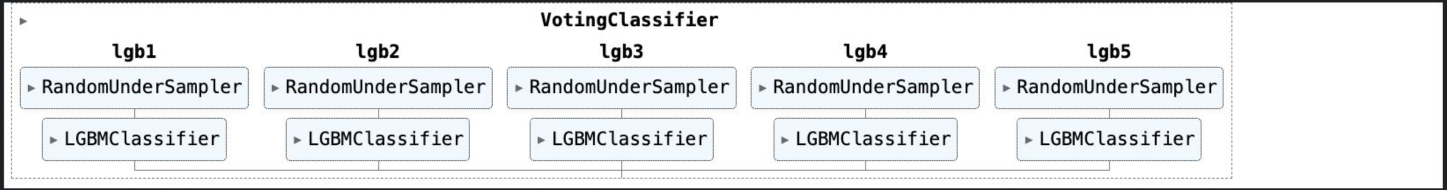
- **Mean CV pAUC:**
  - **~0.1671 using 20k images**
  - **~0.1662 using all 400k images**
- **Test set pAUC: ~0.1495**
- Process: create embeddings from images, merge with data frame, use tabular model on updated data frame.
  - Embeddings used as numeric columns
- Time to generate embeddings depends on sample size. Training on full dataset takes about 1 - 1.5 hours with GPU T4 x2

For reference: Best pAUC from Kaggle competition: 0.17264

Ensembling

Training Final Model (Tabular only)

```python
# X, y = df_train[feature_cols], df_train[target_col]
X, y = df_train[new_feature_cols], df_train[target_col]

estimator.fit(X, y)
```



+ Code + Markdown

Save model using Pickle

```python
with open('model.pkl', 'wb') as file:
    pickle.dump(estimator, file)
```

Hybrid Model Design

- Combines ResNet18 CNN embeddings with tabular features

- Used transfer learning by leveraging a pre-trained ResNet18 model, removing its classification layer to extract 512-dimensional image embeddings that capture visual patterns

- Handled extreme class imbalance by:
    - Data augmentation on positive samples using rotations, flips, and minor color adjustments
    - Undersampling negative cases to achieve a better balance
    - Using an ensemble of 5 LightGBM models with different random seeds for stability

- Extracts CNN features in batches to handle large dataset efficiently

Most Important Features

Tabular:

```
Training Complete!
Total time: 62.86 seconds (1.05 minutes)

Model Performance:
Mean score: 0.1587
Score std: 0.0102
All scores: [0.15318169 0.15151431 0.17028419 0.17141857 0.1469753 ]

Top 20 Most Important Features:
                   feature  importance
18            tbp_lv_deltaB      264.24
8               tbp_lv_Aext      259.64
13                 tbp_lv_H      258.00
21        lesion_size_ratio      254.80
34   overall_color_difference      253.48
29     color_contrast_index      240.04
1       clin_size_long_diam_mm      237.24
16              tbp_lv_Lext      237.04
5         tbp_lv_eccentricity      236.44
20        tbp_lv_deltaLBnorm      230.44
15                 tbp_lv_L      229.96
23             hue_contrast      226.44
14              tbp_lv_Hext      217.96
7                  tbp_lv_A      210.84
17            tbp_lv_deltaA      210.52
32      mean_hue_difference      209.36
9                  tbp_lv_B      207.16
31    normalized_lesion_size      206.60
35  size_color_contrast_ratio      206.00
10              tbp_lv_Bext      202.76
```

Hybrid:

```
Top 20 Most Important Features:
                      feature  importance
1      clin_size_long_diam_mm      162.56
13                   tbp_lv_H      134.68
3           tbp_lv_perimeterMM      102.60
32        mean_hue_difference       93.16
4          tbp_lv_minorAxisMM       85.56
2              tbp_lv_areaMM2       82.32
239           cnn_feature_163       81.72
20          tbp_lv_deltaLBnorm       79.80
18              tbp_lv_deltaB       78.48
31      normalized_lesion_size       69.60
34    overall_color_difference       59.96
8                 tbp_lv_Aext       59.16
14                tbp_lv_Hext       51.72
470           cnn_feature_394       50.48
30            log_lesion_area       50.12
28       size_age_interaction       49.00
26     perimeter_to_area_ratio       48.64
23               hue_contrast       47.36
237           cnn_feature_161       45.48
376           cnn_feature_300       44.88
```

Currently hybrid model is worse than tabular only
This limits the ability of my program as it is too reliant on feature generator output alone

```
Training Complete!
Total time: 3441.07 seconds (57.35 minutes)
Hybrid model mean CV score: 0.1662 (±0.0087)

Top 20 Most Important Features:
                      feature  importance
1      clin_size_long_diam_mm      153.68
239           cnn_feature_163      149.52
13                   tbp_lv_H      132.80
18              tbp_lv_deltaB      108.72
237           cnn_feature_161      105.72
8                 tbp_lv_Aext       99.00
20          tbp_lv_deltaLBnorm      95.56
31      normalized_lesion_size      94.44
32         mean_hue_difference      86.76
4          tbp_lv_minorAxisMM       86.48
3          tbp_lv_perimeterMM       85.28
376           cnn_feature_300       82.84
34   overall_color_difference       79.04
2              tbp_lv_areaMM2       77.60
29       color_contrast_index       75.92
7                    tbp_lv_A       75.60
28       size_age_interaction       75.36
21          lesion_size_ratio       72.04
23               hue_contrast       71.96
14                tbp_lv_Hext       71.64

Training final hybrid model...
```

Comparison on holdout test set:

```
Model Comparison Results:
Tabular Model pAUC: 0.1623
Hybrid Model pAUC:  0.1485
```

When hybrid uses 20k samples

```
Model Comparison Results:
Tabular Model pAUC: 0.1623
Hybrid Model pAUC:  0.1495
```

Next steps

- Improve hybrid model or try ensemble of CNN only model and tabular only model
- Do feature reduction on CNN generated features
- Improve openCV generated features and calculate more features
- Try using NN approaches to generate the features themselves from an image
- Use better CNN model (currently using ResNet18)
- Gather more labeled data
- Generate synthetic data for oversampling
- Make program more invariant to lighting changes, skintone, angle, resolution
- Allow for more kinds of images to be uploaded
- Improve app user interface

Ethical Analysis

- False negatives can have deadly consequences
- How accurate does this technology need to be to be ready?
- Should doctors always play a role in diagnosis?
- Access and affordability
- Who is held responsible if a person dies because of a false negative diagnosis (even if app is more reliable than a person)?