Skin Cancer Detection using Convolutional Neural Network

Carter Brinton

Introduction

- Skin cancer is the most common cancer in the United States and worldwide
- More than two people die of skin cancer in the U.S every hour.
- What if we could reduce that number by catching early symptoms before it’s too late.
- We can do this by using a form of Skin Cancer Detection through a Convolutional Neural Network (CNN) to accurately identify skin cancer lesions from dermoscopic images.
- The goal, prove a CNN trained model can significantly improve early detection of skin cancer and save the lives of many around the globe.

Datasets

- The dataset: Skin Cancer MNIST: HAM10000 (6GB)
  - Types: Actinic keratoses, Basal cell carcinoma, Benign keratosis-like lesions, Dermatofibroma, Melanocytic nevi, Melanoma, Vascular lesions

- Contains two folders with 10,000 images divided between them
  - Trained my model first using the images directly, converting them to vectors containing the image data
  - The images weren't resized already

- CSV file that contains the features
  - lesion_id, image_id, dx, dx_type, age, sex, localization
    - dx: different types of skin lesions
    - dx_type: how diagnosis was made

- Csv file that contains pixel values that make up these images.
  - Trained my final model using the pixel file with 3 RGB channels

Datasets

dx:
nv, mel, bkl, bcc, akiec, vasc, df

dx_type

- "histo": Diagnosis based on a histopathology examination, which involves studying tissue samples under a microscope.
- "follow_up": Diagnosis based on a follow-up examination or evaluation of the skin lesion.
- "consensus": Diagnosis based on a consensus among multiple medical experts or practitioners.
- "confocal": Diagnosis made using confocal microscopy, which is a specialized imaging technique for examining skin lesions.

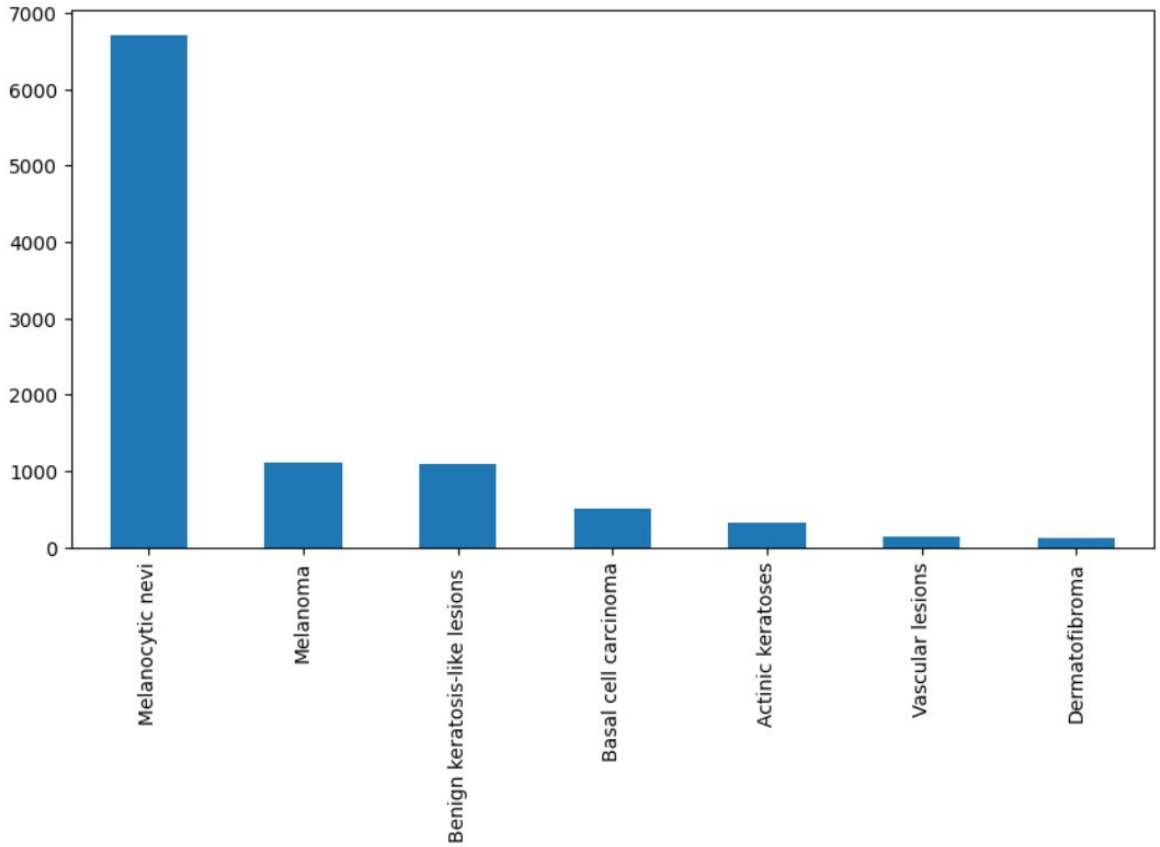
```
skin_df = pd.read_csv(os.path.join(data_path, 'HAM10000_metadata.csv'))
skin_df.tail(10)
```

	lesion_id	image_id	dx	dx_type	age	sex	localization
10005	HAM_0005579	ISIC_0028393	akiec	histo	80.0	male	face
10006	HAM_0004034	ISIC_0024948	akiec	histo	55.0	female	face
10007	HAM_0001565	ISIC_0028619	akiec	histo	60.0	female	face
10008	HAM_0001576	ISIC_0033705	akiec	histo	60.0	male	face
10009	HAM_0005705	ISIC_0031430	akiec	histo	75.0	female	lower extremity
10010	HAM_0002867	ISIC_0033084	akiec	histo	40.0	male	abdomen
10011	HAM_0002867	ISIC_0033550	akiec	histo	40.0	male	abdomen
10012	HAM_0002867	ISIC_0033536	akiec	histo	40.0	male	abdomen
10013	HAM_0000239	ISIC_0032854	akiec	histo	80.0	male	face
10014	HAM_0003521	ISIC_0032258	mel	histo	70.0	female	back

```
lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
}
```

- Data is skewed, (comes back to bite me…)

- Skin Lesion images next slide



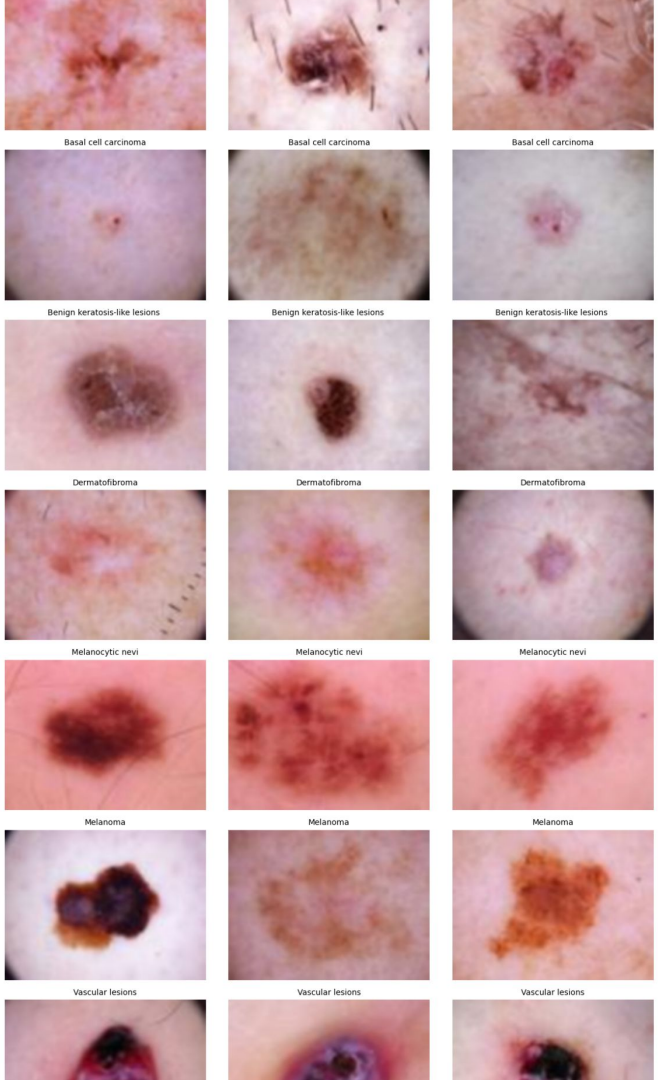
Resizing Images

```python
# Function to load and preprocess images
def load_and_preprocess_image(image_path):
    image = Image.open(image_path)
    image = image.resize((100, 75))  # Resize the image
    return np.asarray(image)

# Iterate through the DataFrame and save resized images using image_id
for index, row in skin_df.iterrows():
    image_id = row['image_id']
    image_path = row['path']
    resized_image = load_and_preprocess_image(image_path)
    resized_image_path = os.path.join(output_dir, f'{image_id}_resized.jpg')
    Image.fromarray(resized_image).save(resized_image_path)
    # Add the path to the resized image to the DataFrame
    skin_df.at[index, 'path_resized'] = resized_image_path
```

```python
#  Store resized images reference in dataframe
skin_df['image'] = skin_df['path_resized'].map(lambda x: np.asarray(Image.open(x)))
```

Image Names Top to Bottom

- Actinic keratoses
- Basal cell carcinoma
- Benign keratosis-like lesions
- Dermatofibroma
- Melanocytic nevi
- Melanoma
- Vascular lesions



Tools and Frameworks

- The primary tools and frameworks I used were TensorFlow and Keras.

```
from tensorflow.keras.utils import to_categorical

# Load and preprocess images for training and testing
x_train = np.asarray(train_set_x['image'].tolist())
x_test = np.asarray(test_set_x['image'].tolist())

# Normalize the data
x_train_mean = np.mean(x_train)
x_train_std = np.std(x_train)
x_test_mean = np.mean(x_test)
x_test_std = np.std(x_test)

x_train = (x_train - x_train_mean) / x_train_std
x_test = (x_test - x_test_mean) / x_test_std

# Perform one-hot encoding on the labels
y_train = to_categorical(train_set_y, num_classes=7)
y_test = to_categorical(test_set_y, num_classes=7)

# Reshape image in 3 dimensions (height = 75px, width = 100px , canal = 3)
x_train = x_train.reshape(x_train.shape[0], *(75, 100, 3))
x_test = x_test.reshape(x_test.shape[0], *(75, 100, 3))
```

- Lack of addressing skewed data.

- Options to do next time:
  - Min-Max Scaling
  - Undersampling the Majority Class
  - Oversampling the Minority Class
  - Weighted Loss Function

- A Sequential model can be used for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

- Input to output, passes through a series of specified neural layers, one after the other.

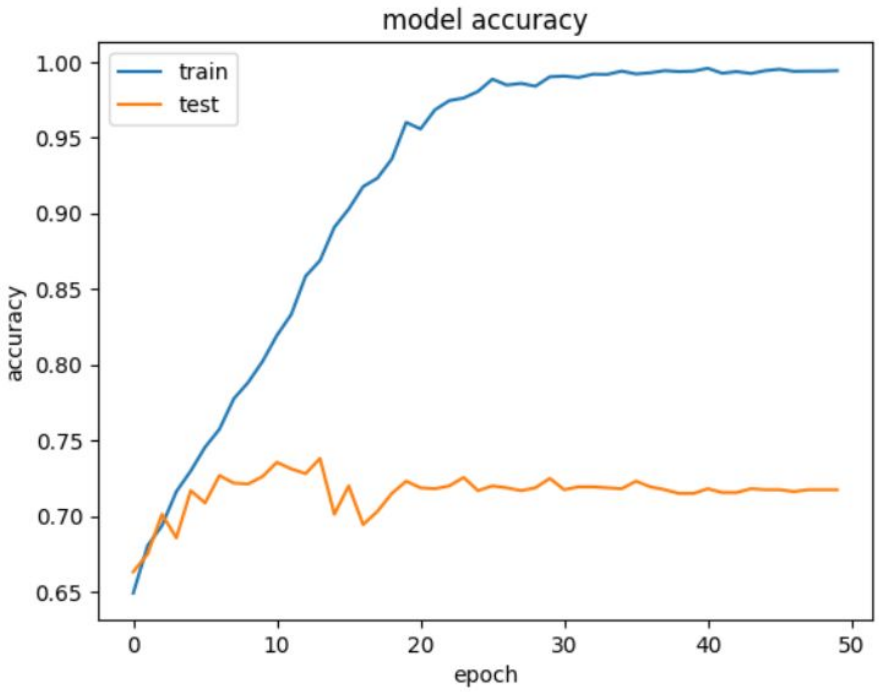
- Adam stands for "Adaptive Moment Estimation" and is a popular optimization algorithm for training neural networks.

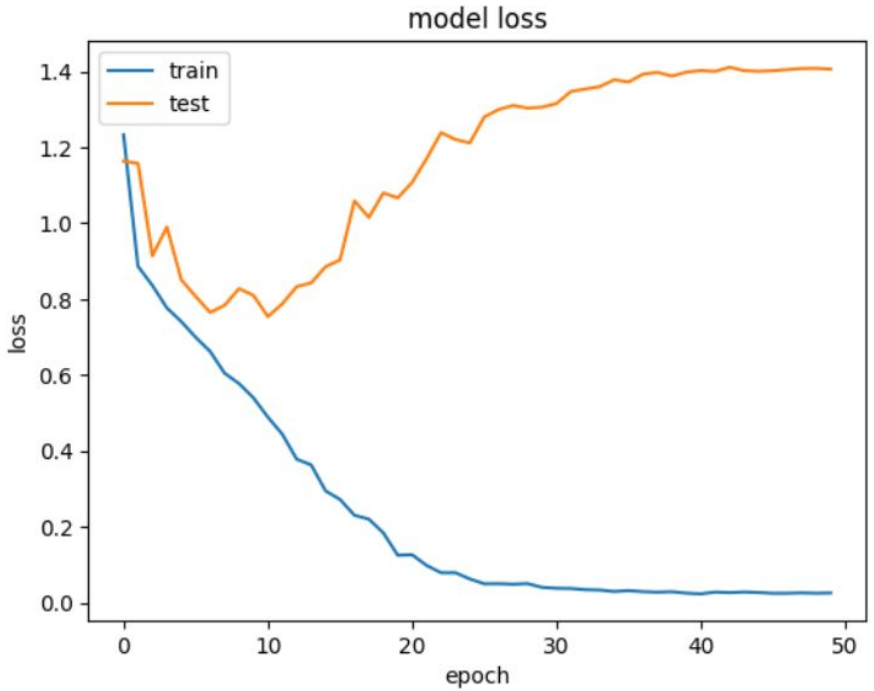
```python
def build_one(input_shape=(75, 100, 3), num_classes=7, learning_rate=1e-3):
    # Create a Sequential model
    model = Sequential()

    # Add the first convolutional layer with 64 filters, 3x3 kernel size, ReLU activation, and Glorot uniform kernel initializer
    model.add(Conv2D(64, kernel_size=(3, 3), input_shape=input_shape, activation='relu', kernel_initializer='glorot_uniform', padding='same'))

    # Add max-pooling layer with a 2x2 pooling size
    model.add(MaxPool2D(pool_size=(2, 2)))

    # Add dropout layer with a dropout rate of 25%
    model.add(Dropout(0.25))

    # Add the second convolutional layer with 64 filters, 3x3 kernel size, ReLU activation, and Glorot uniform kernel initializer
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', kernel_initializer='glorot_uniform', padding='same'))

    # Add another max-pooling layer with a 2x2 pooling size
    model.add(MaxPool2D(pool_size=(2, 2)))

    # Add dropout layer with a dropout rate of 25%
    model.add(Dropout(0.25))

    # Flatten the output of the previous layers
    model.add(Flatten())

    # Add a fully connected layer with 128 neurons, ReLU activation, and normal kernel initializer
    model.add(Dense(128, activation='relu', kernel_initializer='normal'))

    # Add the output layer with num_classes neurons and softmax activation
    model.add(Dense(num_classes, activation='softmax'))

    # Print a summary of the model's architecture
    model.summary()

    # Configure the optimizer with specified learning rate and other hyperparameters
    optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False)

    # Compile the model with categorical cross-entropy loss and accuracy metric
    model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])

    return model
```

```
input_shape = (75, 100, 3)
num_classes = 7
learning_rate = 1e-5

# Create the convolutional neural network model using the build_one function
model = build_one(input_shape=input_shape, num_classes=num_classes, learning_rate=learning_rate)
```

```
epochs = 50
batch_size = 128

# Create a learning rate reduction callback
# This callback monitors the validation accuracy and reduces the learning rate if it doesn't improve for a specified number of epochs
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience=5, verbose=1, factor=0.5, min_lr=0.00001)

# Train the model using the specified hyperparameters and callbacks
history = model.fit(
    x_train,               # Training data
    y_train,               # Training labels
    validation_split=0.2,  # Split a portion of the data for validation
    epochs=epochs,         # Number of training epochs
    batch_size=batch_size, # Batch size for training
    shuffle=True,          # Shuffle the training data
    callbacks=[learning_rate_reduction]  # List of callbacks to apply during training
)
```

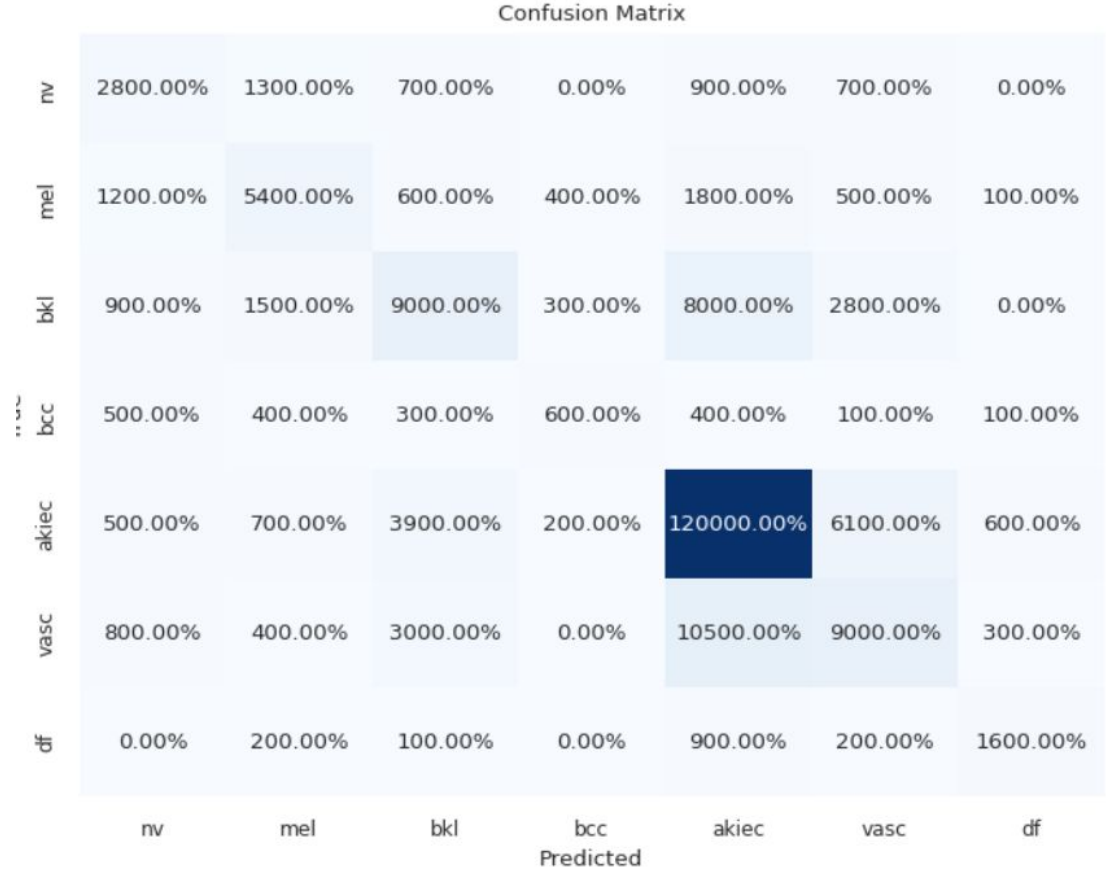
Results - Highest Accuracy: 0.9957





Classification report:

	precision	recall	f1-score	support
nv	0.42	0.44	0.43	64
mel	0.55	0.54	0.54	100
bkl	0.51	0.40	0.45	225
bcc	0.40	0.25	0.31	24
akiec	0.84	0.91	0.87	1320
vasc	0.46	0.38	0.41	240
df	0.59	0.53	0.56	30
accuracy			0.74	2003
macro avg	0.54	0.49	0.51	2003
weighted avg	0.72	0.74	0.73	2003



- Add similar convolutional layers

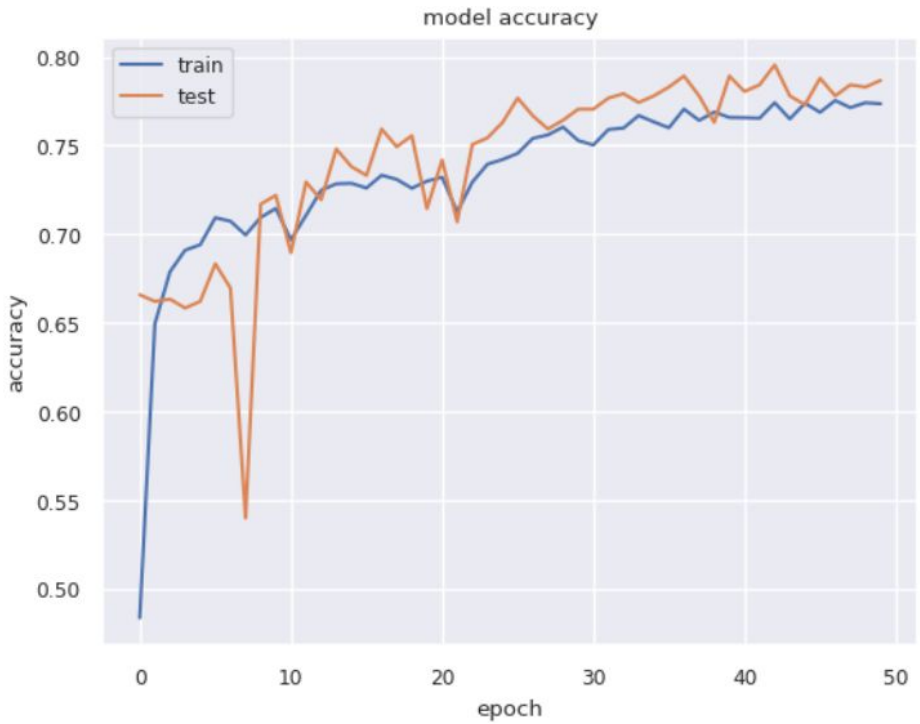
- Adding dense layers, which is a hidden layer that helps the model learn complex patterns in the data.

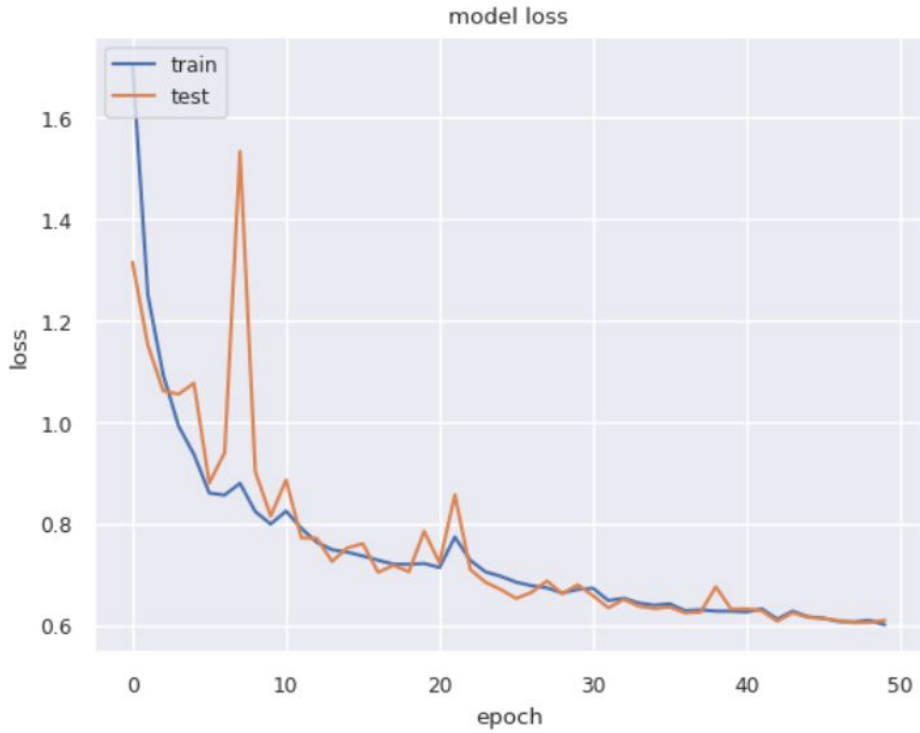
- Potentially a culprit for some of my overfitting

```python
def build_two(input_shape, num_classes, learning_rate, loss):
    model = Sequential()
    model.add(Conv2D(16, kernel_size=(3, 3), input_shape=input_shape, activation='relu', padding='same'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
    model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
    model.add(Flatten())
    model.add(tf.keras.layers.Dropout(0.2))
    model.add(Dense(256, activation='relu'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Dropout(0.2))
    model.add(Dense(128, activation='relu'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(Dense(64, activation='relu'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Dropout(0.2))
    model.add(Dense(32, activation='relu'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(Dense(num_classes, activation='softmax'))
    model.summary()
    optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False)
    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
    return model
```

```python
# Creating validation varibles
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.1, random_state=123)


# Create an ImageDataGenerator object for data augmentation and transformation
datagen = ImageDataGenerator(
    featurewise_center=False,             # Don't set input mean to 0 over the dataset
    samplewise_center=False,              # Don't set each sample mean to 0
    featurewise_std_normalization=False,  # Don't divide inputs by std of the dataset
    samplewise_std_normalization=False,   # Don't divide each input by its std
    zca_whitening=False,                  # Don't apply ZCA whitening
    rotation_range=90,                    # Randomly rotate images in the range (degrees, 0 to 180)
    zoom_range=0.1,                       # Randomly zoom images
    width_shift_range=0.1,                # Randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1,               # Randomly shift images vertically (fraction of total height)
    horizontal_flip=True,                 # Randomly flip images horizontally
    vertical_flip=True,                   # Randomly flip images vertically
    shear_range=10                        # Shear angle in counter-clockwise direction in degrees
)


# Fit the model using data augmentation with ImageDataGenerator
history = model.fit(
    datagen.flow(x_train, y_train, batch_size=batch_size),  # Data generator for training data
    validation_data=(x_val, y_val),                         # Use Validation Data, since 'split' isn't compatible with ImageDataGenerator
    epochs=epochs,                                          # Number of training epochs
    batch_size=batch_size,                                  # Batch size
    shuffle=True,                                           # Shuffle the training data
    callbacks=[learning_rate_reduction]                     # Callbacks for training
)
```

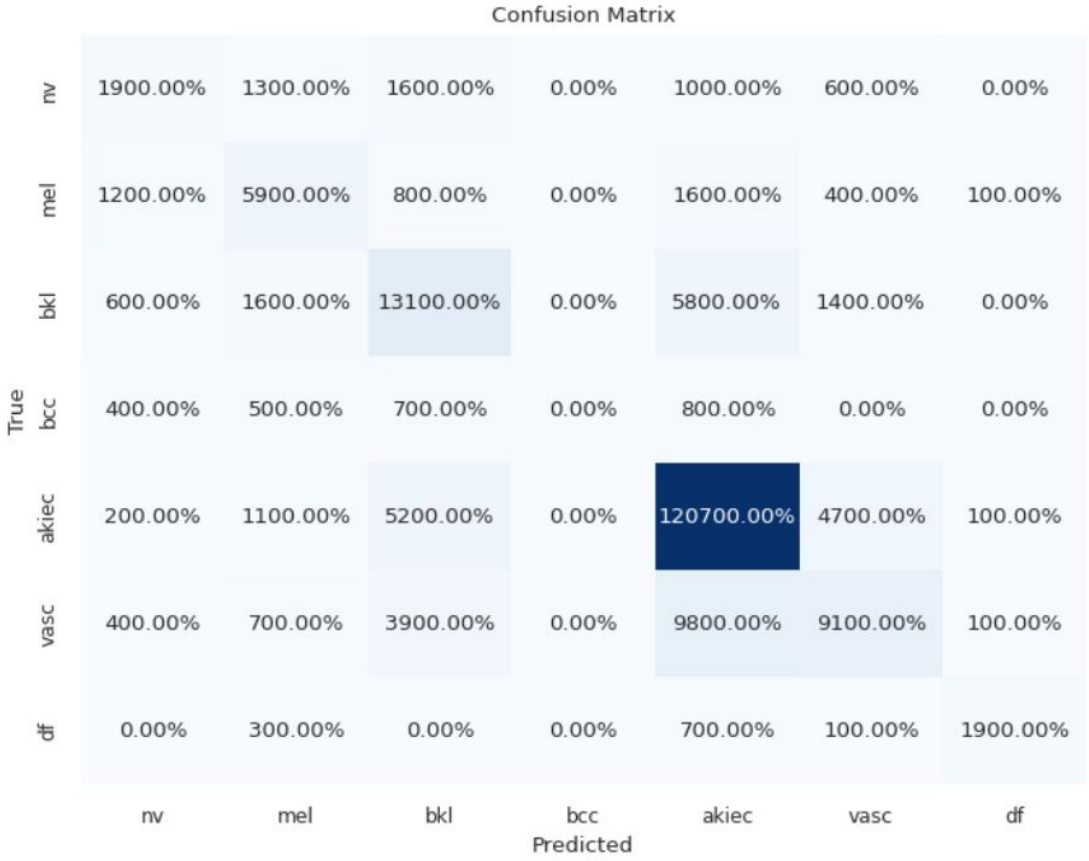
Results - Highest Accuracy: 0.77545





Classification report:

	precision	recall	f1-score	support
nv	0.40	0.30	0.34	64
mel	0.52	0.59	0.55	100
bkl	0.52	0.58	0.55	225
bcc	0.00	0.00	0.00	24
akiec	0.86	0.91	0.89	1320
vasc	0.56	0.38	0.45	240
df	0.86	0.63	0.73	30
accuracy			0.76	2003
macro avg	0.53	0.49	0.50	2003
weighted avg	0.74	0.76	0.75	2003



```
# Load and preprocess the image Basal_Cell_Carcinoma
image = Image.open('Basal_Cell_Carcinoma.jpg')
image
```

```
Melanocytic nevi (nv): 0.00%
Melanoma (mel): 0.00%
Benign keratosis-like lesions  (bkl): 0.00%
Basal cell carcinoma (bcc): 0.00%
Actinic keratoses (akiec): 0.00%
Vascular lesions (vasc): 0.00%
Dermatofibroma (df): 100.00%

Model's Answer: Dermatofibroma (df)
```

```
image = image.resize((75, 100))
img = np.asarray(image).reshape(-1, 75, 100, 3)

# Make a prediction using the model
result = model.predict(img)
result = result.tolist()
print(f"{result[0]}\n")

max_prob = max(result[0])
class_ind = result[0].index(max_prob)

lesion_type_keys = list(lesion_type_dict.keys())
class_shorthand = lesion_type_keys[class_ind]

if class_shorthand in lesion_type_dict:
    correct_name = lesion_type_dict[class_shorthand]
    correct_shorthand = class_shorthand

for i, probability in enumerate(result[0]):
    class_shorthand = lesion_type_keys[i]
    if class_shorthand in lesion_type_dict:
        class_names = lesion_type_dict[class_shorthand]
        print(f"{class_names} ({class_shorthand}): {probability * 100:.2f}%")

print(f"\nModel's Answer: {correct_name} ({correct_shorthand})")
```

Let's look at the Pixel CSV file (RGB channels, 28x28x3)

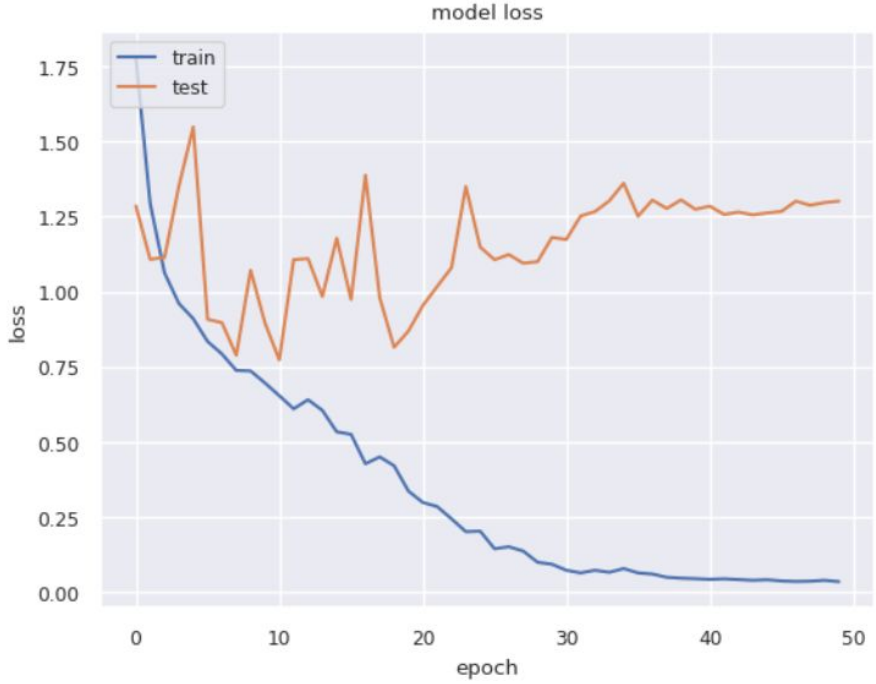
```
skin_csv_df = pd.read_csv(os.path.join(data_path, 'hmnist_28_28_RGB.csv'))
skin_csv_df.tail(10)
```

	pixel0000	pixel0001	pixel0002	pixel0003	pixel0004	pixel0005	pixel0006	pixel0007	pixel0008	pixel0009	...	pixel2343	pixel2344	pixel2345	pixel2346	pixel2347	pixel2348	pixel2349	pixel2350	pixel2351	label
10005	141	103	90	155	113	105	166	122	112	176	...	186	147	126	177	140	121	163	129	110	0
10006	201	150	151	209	163	159	220	175	164	227	...	138	98	93	113	72	67	114	73	70	0
10007	26	13	19	25	10	17	24	6	5	23	...	22	6	9	27	9	10	23	5	6	0
10008	178	134	148	178	142	158	201	171	192	208	...	196	152	148	198	158	160	197	155	158	0
10009	166	157	156	166	157	156	165	156	158	167	...	172	163	154	165	155	146	159	151	143	0
10010	183	165	181	182	165	180	184	166	182	188	...	208	185	187	208	186	186	206	187	189	0
10011	2	3	1	38	33	32	121	104	103	132	...	96	79	76	24	23	21	3	4	1	0
10012	132	118	118	167	149	149	175	156	160	184	...	204	181	178	181	159	153	172	151	145	0
10013	160	124	146	164	131	152	167	127	146	169	...	185	162	167	184	157	166	185	162	172	0
10014	175	142	121	181	150	134	181	150	133	178	...	159	79	82	174	137	125	175	139	126	6

```
# reference: https://www.kaggle.com/kmader/skin-cancer-mnist-ham10000/discussion/183083

skin_lesion_labels={
    0:('akiec', 'actinic keratoses and intraepithelial carcinomae'),
    1:('bcc' , 'basal cell carcinoma'),
    2:('bkl', 'benign keratosis-like lesions'),
    3:('df', 'dermatofibroma'),
    4:('nv', ' melanocytic nevi'),
    5:('vasc', ' pyogenic granulomas and hemorrhage'),
    6:('mel', 'melanoma'),
}
```

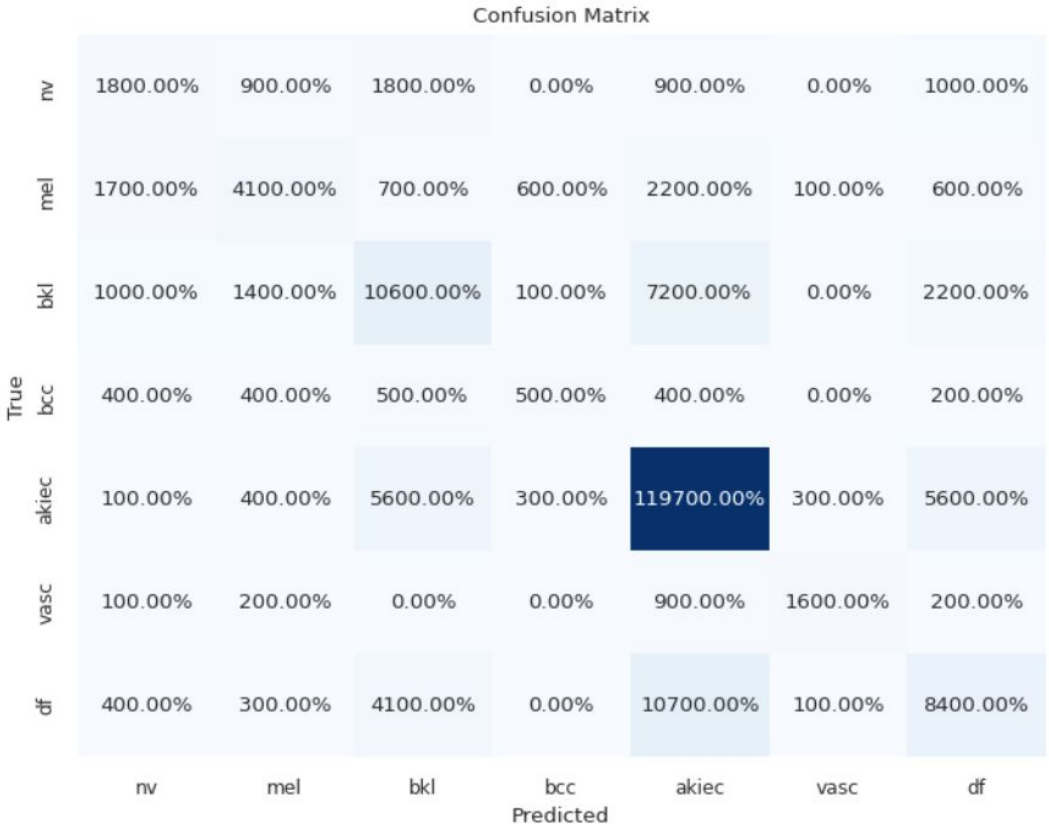
Results - Highest Accuracy: 0.99485

- Used build_two model function for my third training



Classification report:

	precision	recall	f1-score	support
nv	0.33	0.28	0.30	64
mel	0.53	0.41	0.46	100
bkl	0.45	0.47	0.46	225
bcc	0.33	0.21	0.26	24
akiec	0.84	0.91	0.87	1320
vasc	0.76	0.53	0.63	30
df	0.46	0.35	0.40	240
accuracy			0.73	2003
macro avg	0.53	0.45	0.48	2003
weighted avg	0.71	0.73	0.72	2003

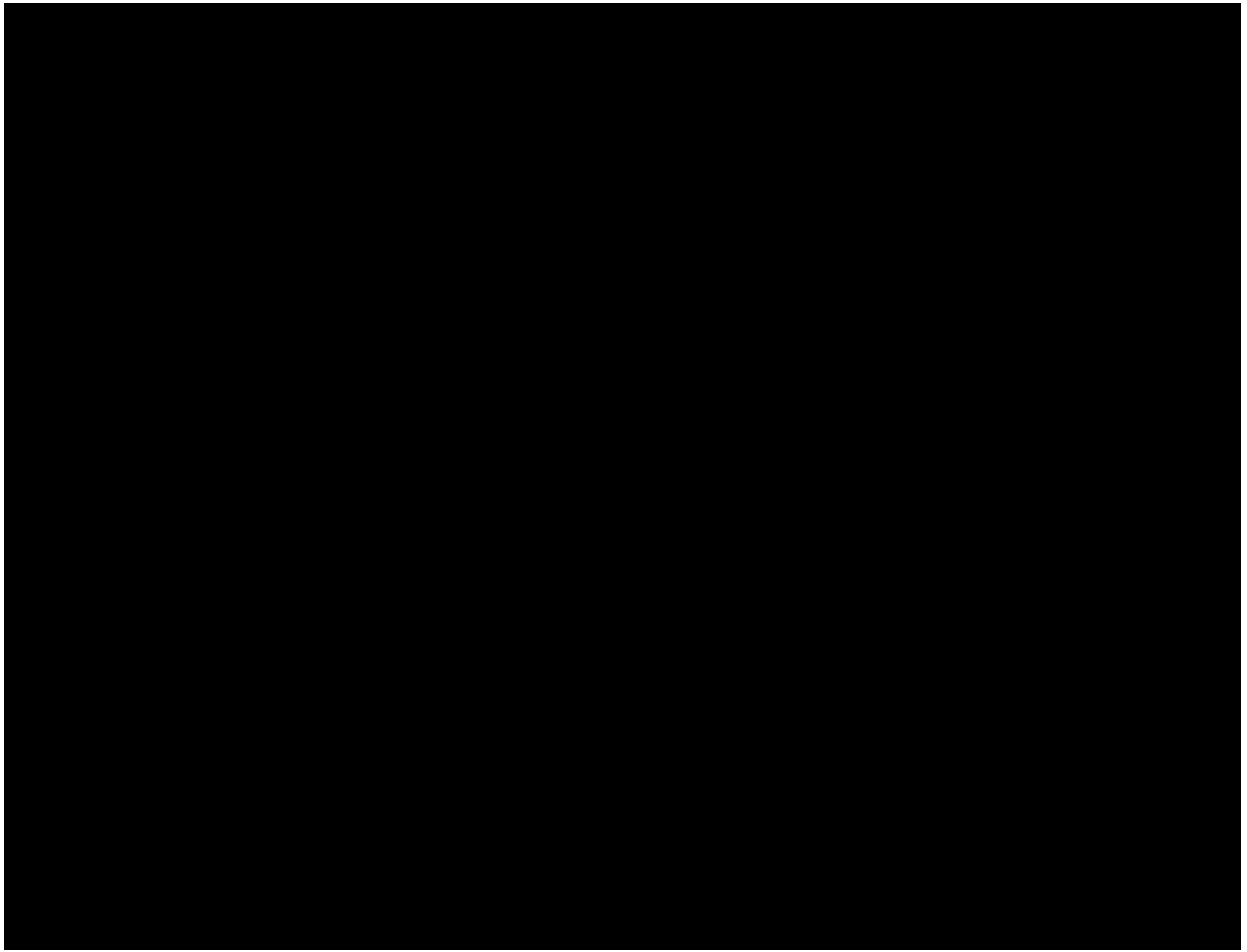


Using same 'Basal_Cell_Carcinoma' image

Melanocytic nevi (nv): 0.62%
Melanoma (mel): 74.49%
Benign keratosis-like lesions (bkl): 1.69%
Basal cell carcinoma (bcc): 5.46%
Actinic keratoses (akiec): 12.37%
Vascular lesions (vasc): 4.58%
Dermatofibroma (df): 0.78%

Model's Answer: Melanoma (mel)

I built an App???



Conclusion

- Possibly bit off more than I could chew….
- Don’t forget about accounting for skewed data
- Challenges with lots of fine tuning/hyperparameters
- Library and dependency errors are stressful
- Training a model of this scale takes a lot of time
- Won’t get model working on the first try, and if you do, something is most likely wrong

- Future goals:
  - Train a more consistent and reliable model
  - Advance model to distinguishing various stages of cancer