Self-Learning AI Model on Limited Biomedical Imaging Data and Labels

Niraj Gupta, Saniya Khalil, Jolie Li, Iris Ochoa, Elisa Torres.
Mentor: David J. Florez Rodriguez.

Abstract

This research explores self-learning AI using Google Colab by pre-training a general TensorFlow-coded model on recognizing patterns in limited, unlabeled biomedical image data. This allows the model to understand the basic underlying patterns and structures in the images. After self-learning, we train the model with labeled data. We hypothesize that self-learning will decrease how much we depend on extensively labeled data for the development of accurate AI models.

Introduction

The development of reliable visual artificial intelligence (AI) models in the biomedical field usually requires a substantial quantity of high-quality, accurately labeled imaging data by professionals. Unfortunately, obtaining such data in the real world is typically limited or inaccessibly expensive. This poses one of the most significant challenges faced in the intersection of machine learning applications in the biomedical field. This hinders the development of potentially high-performing AI models that could help develop effective strategies for disease prevention, early detection, management, etc. In order to combat this, AI models can be trained using easily accessible unlabeled data at first. This allows the nascent model to grasp basic visual patterns that may arise in the data in its unsupervised self-learning phase. Then, labeled data can be utilized in order to improve the efficiency of the model and adjust the parameters that compose it during the supervised self-learning phase.

Our team developed a model that follows these criteria and analyzed its results to answer the question "How does self-learning affect the accuracy of AI models trained on limited, labeled data (primarily breast cancer histopathology images) compared to self-supervised AI models?”

This research focuses on programming an AI model that utilizes a breast cancer dataset that contains limited data, which was sometimes not labeled. Breast cancer, a disease in which the cells in the breast grow uncontrollably and form tumors, is the most common type of cancer in the world. According to the World Health Organization, in 2020, over 2.3 million individuals worldwide were diagnosed with breast cancer. Furthermore, over 685,000 deaths occurred due

to this fatal disease. The growing prevalence of breast cancer makes it imperative for new models and technology to be developed in order to classify possible tumors accurately.

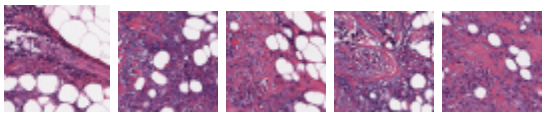
Materials and Methods

Dataset

Both the self-learning model and supervised learning model were trained on a dataset titled “Breast Histopathology Images.” This dataset was taken from Kaggle, a data science and AI platform under Google LLC. The images of the dataset focused on 250 x 250 sized .png images of patients with and without Invasive Ductal Carcinoma (IDC). IDC is the most common subtype of all breast cancers. The full 3 GB dataset consists of 198,738 IDC(-) images and 78,786 IDC(+) images. For the models of this investigation, 772 class 0 (non-IDC) images and 207 class 1 (IDC positive) images fwere used for training and validation purposes. The demographic information of the patients is unstated through the data source.

Breast cancer is important to investigate as it is the most common type of cancer in women. Accurately identifying breast cancer subtypes is an important biomedical task that AI can save time on, decrease cost, and reduce error on.

Sample images:

Class 0 (non-IDC cells)	Class 1 (IDC positive cells)
	

Programming

The AI models were coded on Google Colab notebooks using Python. Functions were imported from Tensorflow, Matplotlib, Numpy, and Pandas.

Self Supervised Learning

The Self Supervised Learning AI model used both a littleNtrain and Ntrain variable. LittleNtrain is a quantity of labeled data for training. This number influences fitting (overfitting or proper learning) and the validation accuracy and loss of the model. Ntrain is a quantity of unlabeled data used in training self-learning AI models. In particular, Ntrain prepares the AI model to recognize visual patterns in the available data so that this recognition ability can be used in training with labeled data later on.

```
### PICTURE EDITING FUNCTIONS

def flip_random_crop(image):
    # With random crops we also apply horizontal flipping.
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_crop(image, (CROP_TO, CROP_TO, 3))
    return image

def color_jitter(x, strength=[0.4, 0.4, 0.4, 0.1]):
    x = tf.image.random_brightness(x, max_delta=0.8 * strength[0])
    x = tf.image.random_contrast(
        x, lower=1 - 0.8 * strength[1], upper=1 + 0.8 * strength[1]
    )
    x = tf.image.random_saturation(
        x, lower=1 - 0.8 * strength[2], upper=1 + 0.8 * strength[2]
    )
    x = tf.image.random_hue(x, max_delta=0.2 * strength[3])
    # Affine transformations can disturb the natural range of
    # RGB images, hence this is needed.
    x = tf.clip_by_value(x, 0, 255)
    return x

def color_drop(x):
    x = tf.image.rgb_to_grayscale(x)
    x = tf.tile(x, [1, 1, 3])
    return x

def random_apply(func, x, p):
    if tf.random.uniform([], minval=0, maxval=1) < p:
        return func(x)
    else:
        return x

def custom_augment(image):
    # As discussed in the SimCLR paper, the series of augmentation
    # transformations (except for random crops) need to be applied
    # randomly to impose translational invariance.
    image = flip_random_crop(image)
    image = random_apply(color_jitter, image, p=0.8)
    image = random_apply(color_drop, image, p=0.2)
    return image
```

The self-learning model defines the different functions shown above which can essentially crop, change the colors, remove the colors, and rotate pictures from the training dataset.

```
### SELF LEARNING

# Create a cosine decay learning scheduler.
num_training_samples = len(x_train)
steps = EPOCHS * (num_training_samples // BATCH_SIZE)
lr_decayed_fn = tf.keras.optimizers.schedules.CosineDecay(
    initial_learning_rate=0.03, decay_steps=steps
)

# Create an early stopping callback.
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor="loss", patience=15, restore_best_weights=True
)

# Compile model and start training.
simsiam = SimSiam(get_encoder(), get_predictor())
simsiam.compile(optimizer=tf.keras.optimizers.SGD(lr_decayed_fn, momentum=0.6))
ssHistory = simsiam.fit(ssl_ds, epochs=EPOCHS, callbacks=[early_stopping])

# Visualize the training progress of the model.
plt.plot(ssHistory.history["loss"])
plt.grid()
plt.title("Negative Cosine Similairty")
plt.show()

pd.DataFrame(ssHistory.history).to_csv(loc + "Results/cosineLoss_" + str(Ntrain) + "_" + str(littleNtrain) + "_" + dataset_name + ".csv")

print('SELF SUPERVISED RESULTS SAVED!!!\n'*4)
```

The model is compiled and the randomly modified images are fed to the model. The cosine loss function demonstrates the progress of the model as it is trained by producing values used for measuring how similar or different two inputs are. The model is trained to recognize whether two images are the same, even when one version of the image was modified.

```
backbone = tf.keras.Model(
    simsiam.encoder.input, simsiam.encoder.get_layer("backbone_pool").output
)

# We then create our linear classifier and train it.
backbone.trainable = False # locking self learning in place
inputs = layers.Input((CROP_TO, CROP_TO, 3))

x = backbone(inputs, training=False)
x = layers.Dropout(.125)(x)
x = layers.Dense(32, activation= "tanh")(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(.1)(x)
x = layers.Dense(32, activation= "tanh")(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(.1)(x)
x = layers.Dense(32, activation = "sigmoid")(x)

outputs = layers.Dense(num_cat, activation="softmax")(x)
linear_model = tf.keras.Model(inputs, outputs, name="linear_model")

# Compile model and start training.
linear_model.compile(
    loss="categorical_crossentropy",
    metrics=["accuracy"],
    optimizer=tf.keras.optimizers.SGD(lr_decayed_fn, momentum=0.9),
)
linear_model.summary()
history = linear_model.fit(
    train_ds, validation_data=test_ds, epochs=EPOCHS, callbacks=[early_stopping]
```

The layers of the self-learning model include TensorFlow layers dropout, dense, and batch normalization.

Full code of Self Supervised Learning model:
CODE - FriendlySelfSupervisedLearningFINAL.ipynb

Supervised Learning
Unlike the Self Supervised Learning AI model, the Supervised Learning AI model only uses the littleNtrain variable. This is because selfless (supervised training without self supervised learning) AI models train only on labeled data, a difference that gives self-learning AI models the advantage since the selflessmodel can only train on a few labeled data sources.

```
###  TRAINING WITHOUT SELF SUPERVISED LEARNING

simsiam = SimSiam(get_encoder(), get_predictor())

# Create a cosine decay learning scheduler.
num_training_samples = len(x_train)
steps = EPOCHS * (num_training_samples // BATCH_SIZE)
lr_decayed_fn = tf.keras.optimizers.schedules.CosineDecay(
    initial_learning_rate=0.03, decay_steps=steps
)

# Create an early stopping callback.
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor="loss", patience=15, restore_best_weights=True
)

def self_less_model(indim):
  inputs = layers.Input((indim, indim, 3))
  resnet = tf.keras.Model(
    simsiam.encoder.input, simsiam.encoder.get_layer("backbone_pool").output
  )
  # We then create our linear classifier and train it.
  x = resnet(inputs)
  outputs = layers.Dense(num_cat, activation="softmax")(x)
  linear_model = tf.keras.Model(inputs, outputs, name="linear_model")
  # Compile model and start training.
  return linear_model

self_less = self_less_model(x_train[0].shape[0])
self_less.trainable=True
self_less.summary()
self_less.compile(
    loss="categorical_crossentropy",
    metrics=["accuracy"],
    optimizer=tf.keras.optimizers.SGD(lr_decayed_fn, momentum=0.9),
)


history = self_less.fit(x_train,y_train, epochs=EPOCHS)
pd.DataFrame(history.history).to_csv(loc + "Results/selfLess_"  + str(littleNtrain) + "_" + dataset_name + ".csv")
print('SELF LESS TRAINING RESULTS SAVED!!!\n'*4)
```

The selfless learning model does not train on any modified data. Instead, this model trains only on the original, labeled data source.

Full code of the Supervised Learning model:
CODE - FriendlySelfLessLearningFINAL.ipynb

Results

Self-Supervised Learning [AKA Selfless] Notebook

With the self-supervised learning notebook we ran the code with up to 800 breast cancer unlabeled images, and 160 labeled images. We observed an overfitting tendency until the sample size of the unlabeled data was 40, which means that the model didn't capture underlying patterns, but unimportant fluctuations. As the sample sizes increased, our validation accuracy increased to 90% (from the original 78% from overfitting), suggesting a large improvement in the pattern recognition as well as the accuracy of predictions for the images.

Moreover, our validation (Val) accuracy parameter starts at 78%, suggesting that our models' predictions are accurate for an estimated 78% of data points, this percentage is also the accuracy for guessing all the images are healthy, or in other words overfitting. It later increased to 85%, which similarly to the training data, improves its accuracy. Although we observed a 6% increase throughout the models, signaling that it is learning and getting proficient, it still doesn´t reach an ideal performance of 100%.

Supervised Learning [AKA Self] Notebook

For our Supervised Learning notebook, 160 labeled images were run through the model. We detected that the model's accuracy overfits similar amounts to selfless ones as it goes down to 40%, and goes up to 90%, possibly due to the differing amounts of diseased data in the samples.

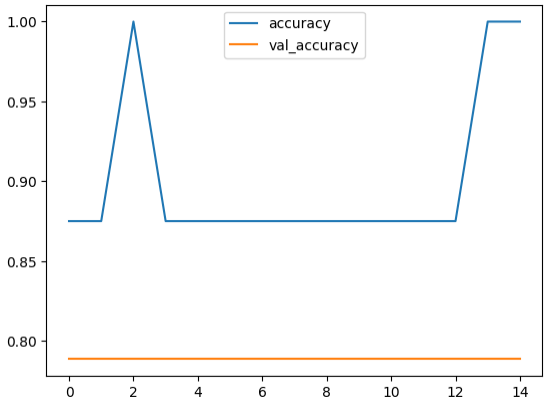
For our Val accuracy, we obtained an average of 78%, which indicates that our model's prediction is 22% off target. The supervised learning model has a larger max validation loss across the sample when compared with the self-supervised learning model.

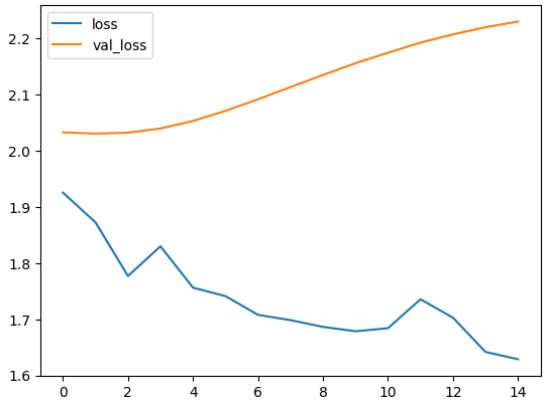
The results suggest that the model's predictions in both notebooks showed a significant improvement, but these results are not conclusive. The val accuracy still needs to go up by several percent to reach an ideal performance of imaging analysis and pattern detection. Therefore, we will need to run a larger amount of similar models to examine the algorithms for any possible changes, specifically increases, in the values' accuracy.

Scatterplots obtained

Our results were analyzed by considering certain parameters and indexes as the x-axis which addresses the little Ntrain values and the y-axis for the validation loss. As shown below, we have two graphs indicating our results, or in other words the accuracy of our models [as seen on the left] and loss [as seen on the right] used to guide the optimization process. In our val accuracy plot, we can state that whereas our accuracy remained stable at a lower 0.80

validation loss with Ntrain values between 0 and 14, our accuracy had a sharp rise when having 2 as our Ntrain value and later on a similar increase when our Ntrain value was 12, yet this time it remained higher until 14.





Discussion

Training Accuracy:

The accuracy was very high at the beginning of the testing, this is most likely due to the overfitting in the smaller datasets. Since the number of images with breast cancer was significantly less than the healthy images, the model most likely determined everything as healthy in the beginning, leading to a higher accuracy due to the low amount of disease data. Once the number of diseased images in the sample increased, the accuracy of the model decreased due to the overfitting done earlier. We can counter this issue by increasing the sample sizes of the images (>100) in order to expose the model to more disease images so that it may familiarize itself with the patterns found in the disease images.

After increasing the labeled image sample size (from 16 to 160 by multiples of 8) and unlabeled image data (from 64 to 800) we noticed a slight gradual increase in validation data. We can assume that due to the higher exposure to the disease images, the model was able to pick up on the difference in patterns between healthy and diseased images, resulting in an increase in accuracy.

Our Hypothesis that training on unlabeled data before using labeled data as a supplement increases the efficiency of the model can be seen in the comparisons of validation accuracy. The validation accuracy for guessing (due to overfitting most likely) is 78%, which also happens to be the validation accuracy percentage for Supervised learning (using only labeled data), while the self-supervised learning model, as mentioned previously was able to increase accuracy.

Potential applications in medical diagnosis and future treatments:

This model if successful will make it more affordable and efficient to sort through diseased and healthy medical images. By using unlabeled data to train the model in pattern detection, the expense of acquiring labeled data is greatly decreased, and the efficiency of finding patterns and determining categories is greatly increased.

Our research could potentially serve and be insightful for image enhancement as we can reduce noise or modify features, leading to nitid images that could be used to do a better disease diagnosis for patients.

Additionally, AI is currently being employed in various medical approaches to facilitate doctors' work when detecting certain anomalies, accelerating drug development, or even when understanding complex disorders.

Future Direction

Research regarding AI models and the usage of unlabeled datasets, particularly biomedical, is critical for the development of new strategies and technology that can have major impacts in the healthcare field. In order to further develop our research and gain more substantial results, a myriad of changes can be employed in the future.

Greater computational power would allow the creation of more intricate and comprehensive models, which would allow for the production of more accurate results. Furthermore, an increased amount of data would allow the AI model created to have more reliable results with decreased margins of error.

Larger data samples create stronger results, decrease chances of common behaviors such as overfitting (a situation caused by small training data sets), allow greater training with varying Ntrain and littleNtrain combinations, and more.

Moreover, it would be imperative to test the AI model created on additional, diverse datasets in order to avoid bias. Training the model on numerous datasets would allow us to create a generalizable machine learning model with an architecture that can be utilized for numerous situations.This could increase the positive impact of our model allowing it to adapt to various circumstances.

Overall, we will continue to test and grow our AI model with various adjustments in order to ensure its efficiency and increase its performance.

Works Cited

Kaggle: Your Machine Learning and Data Science Community, https://www.kaggle.com/. Accessed 20 July 2023.

"Breast cancer." *World Health Organization (WHO)*, 12 July 2023, https://www.who.int/news-room/fact-sheets/detail/breast-cancer. Accessed 2 August 2023.

"Breast Histopathology Images." *Kaggle*, https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images. Accessed 20 July 2023.

"Self-supervised contrastive learning with SimSiam", Keras, https://keras.io/examples/vision/simsiam/ Accessed 01 Aug 2023