CPSC 8650 Data Mining

Quality Evaluation of Skull Stripped Brain MRI Images

Team 3 - Manish Enishetty, Tejaswi Namburu, Shruti Kondur

Abstract

Deep 2D convolutional neural networks are commonly used in medical image processing on volumetric data (CNNs). This is mostly due to the difficulties imposed by the nature of 3D data: changing volume size, GPU exhaustion during optimization. However, in 2D CNNs, dealing with individual slices independently discards the depth information, resulting in poor performance for the desired job. As a result, it is critical to design approaches that not only overcome the high memory and computing needs, but also make use of 3D information. We have come up with a 3D CNN to solve the problem of quality evaluation of skull-stripped Brain MRI images. In this work, we have first prepared data with the help of data pre-process such as Normalization, interpolation of volume over an axis and volume reduction to make CNN training . With this 3D CNN model, we have generated two models which help to classify the problem of Facial Features and Brain Feature Loss in the Images. We report that our newly built models have a very good accuracy and very less loss during training and Validation. The main objective of this work is to develop a hybrid technique, which can classify the brain MRI images successfully and efficiently via 3D CNN model.The Recognizable Facial feature model is around 97% confident about not having facial features and the Brain Feature Loss model is around 98% confident that MRI Images doesn't have any brain loss features.

Keywords: 3D data processing · CT images · convolutional neural networks.

Problem Statement

Sometimes facial features are retained in the skull-stripped MRI Images, and then there are instances where voxels depicting brain tissues are eliminated. Ensuring a patient's confidentiality and imaging quality are always major considerations in medical diagnosis. The aim of our project is to build an algorithm that determines if there are any recognizable facial features and brain feature loss in the given skull stripped MRI images.

Introduction

Skull stripping is one of the first steps in the process of diagnosing brain disorders. It is the separation of brain tissue from non-brain tissue in an MRI imaging of the brain. Even for professional radiologists, segmenting the brain from the skull is a time-consuming task, and the accuracy of the results varies greatly from person to person. We are attempting to automate the procedure by developing an end-to-end pipeline that requires only the raw MRI image to be input and should generate a segmented image of the brain after the necessary preprocessing.

A patient is put into a tunnel with a magnetic field inside to obtain an MR image. As a result, all protons in the body 'align' themselves so that their quantum spins are the same. An oscillating magnetic field pulse is then utilized to disturb this alignment. When the protons re-establish their equilibrium, they emit an electromagnetic wave. Different images will be acquired depending on the fat content, chemical makeup, and, most crucially, the sort of stimulation (i.e. sequences) employed to disrupt the protons. T1, T1 with contrast (T1C), T2, and FLAIR are the four most typical sequences obtained.

Skull-stripped image: This can be thought of as part of the brain stripped from the above T1-weighted image. This is similar to overlaying masks to actual images.

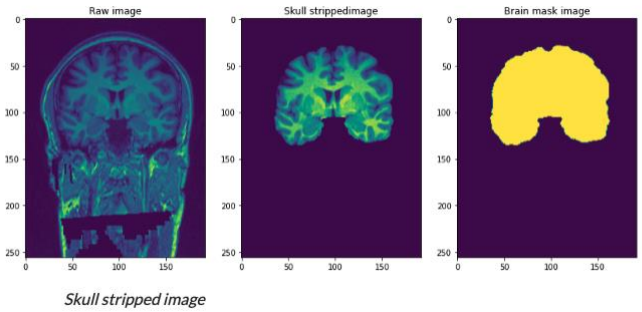


Fig 1. Figure showing Raw Image, Skull Stripped Image, Brain Mask Image

There are obstacles to learning the geometric features of volumetric data imposed by the data itself[1]. Fitting the data into GPU memory during optimization is a significant difficulty. Furthermore, complication occurs while dealing with the data's changeable depth size. As a result, the data preparation method is critical for developing robust systems based on volumetric image data. Deep learning has been widely applied in the context of medical imaging in a range of tasks and domains[2].

Brain MRI imaging has been widely used by neurologists and neuroscientists in disease diagnosis and human brain study since the 1980s. There are quite a few different imaging modalities or sequences used in imaging the nervous system and producing different kinds of brain MRI images, such as T1-weighted (T1W) images, T2-weighted (T2W) images, Diffusion-weighted images (DWI), etc. To advance neuroimaging research, more and more researchers share neuroimaging data in open access data repositories for collaboration and knowledge dissemination.

To ensure HIPAA compliance, researchers often use skull-stripping tools to remove the image voxels that represent the skull structure and facial features from the raw 3D MRI images before sharing data. Skull stripping is a segmentation process to extract brain tissues from a raw 3D MRI image of a brain. This is one of the preliminary steps in neuroimaging research. Various skull-stripping tools and algorithms have been proposed to extract brain images however, using these algorithms and tools for brain image segmentation is a tedious

task even for expert radiologists and the accuracy of results varies a lot from person to person.

Sometimes, the facial features are still in the processed images, and the other times, the voxels representing actual brain tissues were removed. To ensure the quality of the skull stripping process, researchers often spend a lot of time and effort inspecting the quality of the segmentation results.

We have used the data set provided by professor Wang Research team i.e BET-BSE-DATA which has skull stripped images and Labels for each scan showing whether it has facial features and brain feature loss.

We summarize our contributions as follows:

- Pre-process the data using Min-Max Normalization, resize around the z-axis (interpolation around z-axis).
- We developed a 14 layer 3D Convolutional neural network.
- Trained the CNN, to get two models for Recognizing facial feature and Brain Feature loss in the MRI scans.
- Tested this model against a testing dataset to get the model accuracy and model loss of both models.

Related Works

3D Based Approaches: Studies based on 3D convolutions for 3D medical picture analysis have been demonstrated [3, 4, 5], rather than using 3D spatial information as the input channel in 2D based approaches. These methods are capable of capturing 3D context along any axis and alleviate the limitation of 2D approaches in capturing 3D context along a certain axis (depth/z-axis). As a result, 3D approaches are often superior when a 3D context is necessary. UUIP [6] conducts a related work in which they utilize 3D CNN as an autoencoder followed by a random forest classifier. To maintain the 3D context, downsampling was conducted at the volume level prior to training the autoencoder. UoAP [7] used a 3D CNN

(VoxNet) with either 16 or 32 slices that were selected from the top, middle, and bottom of the volume

Experiment

Dataset

The dataset we used was developed by professor Wang's research team. They conducted a skull stripping process on these images with BET and BSE tools with different parameters. Their group then examined images to determine whether the skull-stripped images had recognizable facial features or had brain tissue images removed, and created an associated label file. The dataset has around 1311 MRI scans of the Human Brain. Each image is a .gz file. A GZ file is an archive file compressed by the standard GNU zip (gzip) compression algorithm. It typically contains a single compressed file but may also store multiple compressed files. gzip is primarily used on Unix operating systems for file compression.

We have to unzip these images to get the .nii format of scans, which are used to prepare pre-process data.

NII (or NIfTI) files are probably the most commonly used format for multi-dimensional neuroimaging data as of this writing (late 2012). NIfTI is a raster format, with files generally containing at least 3-dimensional data: voxels, or pixels with a width, height, and depth. NIfTI files can be up to 7-dimensional: the first four dimensions are defined as 3 spatial dimensions and time, with the other dimensions used for "other stuff" -- though the time dimension is often to encode something other than time.

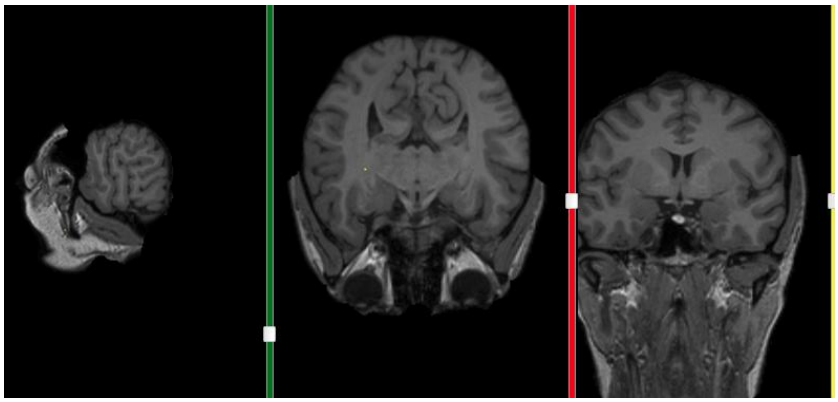


Fig 2: Skull Stripped Images in 2D view

The dataset also has a label file that shows labels for recognizable facial features and Brain Feature loss in the MRI scan.

Label_file

Filename	Recognizable-Facial-Feature	Brain-Feature-Loss
IXI369-Guys-0924-T1_bet_03.nii	Yes	No
IXI448-HH-2393-T1_bet_07.nii	Yes	No
IXI252-HH-1693-T1_bet_08.nii	Yes	No
IXI188-Guys-0798-T1_bet_17.nii	Yes	No
IXI182-Guys-0792-T1_bet_17.nii	Yes	No

Fig 3: Label_file.csv file image showing yes/no for facial features and Brain feature loss

Tools

We have used Tensorflow to work on the project. The main library we used is Keras from TensorFlow. We also used python 2.7

TensorFlow is an end-to-end platform for machine learning. It supports the following:

- Multidimensional-array-based numeric computation (similar to NumPy.)
- GPU and distributed processing
- Automatic differentiation
- Model construction, training, and export
- And more

Keras is **a powerful and easy-to-use free open-source Python library for developing and evaluating deep learning models**. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code

We also used Numpy and Pandas libraries for various functionalities in the network to read and work on various kinds of data. We also used pickle files to store images.

Pickle can be used to serialize Python object structures, which refers to the process of converting an object in the memory to a byte stream that can be stored as a binary file on disk. When we load it back to a Python program, this binary file can be de-serialized back to a Python object.

NiBabel supports an ever growing collection of neuroimaging file formats. Every file format has its own features and peculiarities that need to be taken care of to get the most out of it. To this end, NiBabel offers both high-level format-independent access to neuroimages, as well as an API with various levels of format-specific access to all available information in a particular file format.

Environment

If you are working with Neural Networks, you will have to train the neural network with training data. This training of neural networks deals with updating weights and backpropagation and this is the most intense phase in working with them.

A neural network takes in inputs during training, which are then processed in hidden layers using weights that are modified throughout training, and the model spits out a prediction. Weights are modified in order to uncover trends and create better forecasts.

So, working on a CPU is nothing but overloading your computer with enormous data and it consumes all of your RAM to train the neural network. You need to run at least 10 epochs to build a mediocre model which has better performance than a model generated after the first epoch. But for a better performance you need to run more than 20 epochs. Each epoch may

take around 40 mins on the CPU. This consumes a lot of time just for training and if the results are not satisfactory you need to again run this model for hours and hours.

To overcome this we implemented all of our work in Palmetto Clusters provided by Clemson High performance computing systems. We ran our model on a GPU model k40. We have created an interactive node of tensorflow on the cluster to get all the required libraries such as keras, numpy, NiBabel, pandas etc which are used in the project.

We can say that we saved a lot of time by running on GPU, we ran 20 epochs , it took only 40 mins to build an optimal model with significant accuracy and low loss during validation.

Loading Data and Preprocessing Data

Neural networks don't process raw data, like text files, encoded JPEG image files, or CSV files. They process **vectorized** & **standardized** representations.

- Text files need to be read into string tensors, then split into words. Finally, the words need to be indexed & turned into integer tensors.
- Images need to be read and decoded into integer tensors, then converted to floating point and normalized to small values (usually between 0 and 1).
- CSV data needs to be parsed, with numerical features converted to floating point tensors and categorical features indexed and converted to integer tensors. Then each feature typically needs to be normalized to zero-mean and unit-variance.

Inorder to achieve these features, we have used following functions to normalize and resize the image over z-axis(interpolation).

The following code shows limits for Min-Max Normalization. Hounsfield units are used to store raw voxel intensity in CT scans (HU). In this dataset, they range from -1024 to more

than 2000. Because bones with varying radio intensities exist over 400, this is considered as an upper bound. CT scans are routinely normalized using a threshold between -1000 and 400.

```
def normalize(volume):
    """Normalize the volume"""
    min = -1000
    max = 400
    volume[volume < min] = min
    volume[volume > max] = max
    volume = (volume - min) / (max - min)
    return volume.astype("float32")
```

To process the data, we do the following:

- We first rotate the volumes by 90 degrees, so the orientation is fixed
- We scale the HU values to be between 0 and 1.
- We resize width, height and depth

```
def resize_volume(img):
    """Resize across z-axis"""
    # Compute depth factor
    depth_factor = 1 / (img.shape[-1] / 64)
    width_factor = 1 / (img.shape[0] / 128)
    height_factor = 1 / (img.shape[1] / 128)
    # Rotate
    img = ndimage.rotate(img, 90, reshape=False)
    # Resize across z-axis
    img = ndimage.zoom(img, (width_factor, height_factor, depth_factor), order=1)
    return img
```

We load and read data using get_fdata() function to read .nii images from nib class.

We will preprocess data using the above steps. Now we will split the dataset into two parts, one for training and other for testing. We took the split 70:30 for training and testing.

We now store the preprocessed data in pickle files.

```
X_test_facial = dataset['Recognizable-Facial-Feature'][:918]
X_test_facial = dataset['Recognizable-Facial-Feature'][918:]
X_test_brain = dataset['Brain-Feature-Loss'][:918]
X_test_brain = dataset['Brain-Feature-Loss'][918:]
```

```
with open('X_train.pickle', 'wb') as f:
    pickle.dump(X_train, f)
with open('X_test.pickle', 'wb') as f:
    pickle.dump(X_test, f)
```

We stored our preprocessed data into two pickle files. These will be used in building training and validation datasets.

We consider the problem of the CNN as Binary Classification, so we will make the label_file.csv file contents to '1' and '0' for 'Yes' and 'No'. Since CNN is a supervised learning, we use the image and corresponding label for supervised learning.

Build train and validation datasets

Read the scans from the class directories and assign labels. Downsample the scans to have a shape of 128x128x64. Rescale the raw HU values to the range 0 to 1. Lastly, split the dataset into train and validation subsets

We have to unload the pickle files into two files.

```
with open('X_train.pickle', 'rb') as f:
    X_train = pickle.load(f)
with open('X_test.pickle', 'rb') as f:
    X_test = pickle.load(f)
```

We have loaded the preprocessed data into these files with corresponding labels.

```
y_train_facial = rec_df['Recognizable-Facial-Feature'][:len(X_train)]
y_test_facial = rec_df['Recognizable-Facial-Feature'][len(X_train):]

y_train_brain = rec_df['Brain-Feature-Loss'][:918]
y_test_brain = rec_df['Brain-Feature-Loss'][918:]
```

Data augmentation

During training, the CT scans were also supplemented by spinning at different angles. Because the data is stored as rank-3 shape tensors (samples, height, breadth, depth), we add a dimension of size 1 at axis 4 to perform 3D convolutions on it. As a result, the new shape (samples, height, width, depth, 1). There are several preprocessing and augmentation techniques available; this example demonstrates a few basic ones.

```
@tf.function
def rotate(volume):
    """Rotate the volume by a few degrees"""

    def scipy_rotate(volume):
        # define some rotation angles
        angles = [-20, 20]
        # pick angles at random
        angle = random.choice(angles)
        # rotate volume
        volume = ndimage.rotate(volume, angle, reshape=False)
        volume[volume < 0] = 0
        volume[volume > 1] = 1
        return volume

    augmented_volume = tf.numpy_function(scipy_rotate, [volume], tf.float32)
    return augmented_volume
```

The training data is supplied through an augmentation function that randomly rotates volume at different angles while defining the train and validation data loader. It should be noted that both the training and validation data have previously been rescaled to have values between 0 and 1.

```
def train_preprocessing(volume, label):
    """Process training data by rotating and adding a channel."""
    # Rotate volume
    volume = rotate(volume)
    volume = tf.expand_dims(volume, axis=3)
    return volume, label


def test_preprocessing(volume, label):
    """Process validation data by only adding a channel."""
    volume = tf.expand_dims(volume, axis=3)
    return volume, label
```

```
train_loader_facial = tf.data.Dataset.from_tensor_slices((X_train, y_train_facial))
test_loader_facial = tf.data.Dataset.from_tensor_slices((X_test, y_test_facial))
```

This is how we build a training and testing loader for giving input to CNN. We are slicing the images into tensors, so that it becomes easier for training the neural network.

```
batch_size = 16
# Augment the on the fly during training.
train_dataset_facial = (
    train_loader_facial.shuffle(len(X_train))
    .map(train_preprocessing)
    .batch(batch_size)
    .prefetch(2)
)
# Only rescale.
test_dataset_facial = (
    test_loader_facial.shuffle(len(X_test))
    .map(test_preprocessing)
    .batch(batch_size)
    .prefetch(2)
)
```

We have given the batch size as 16 here. This screenshot shows the Augmentation on the fly during training of the Recognizable facial feature model and also Brain Feature Loss Model. The test and train loader are loaded from pickle files in a random manner using .shuffle() function.

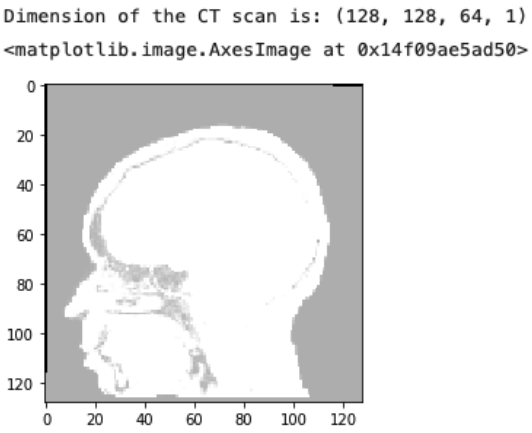


Fig: Image showing Augmented CT scan

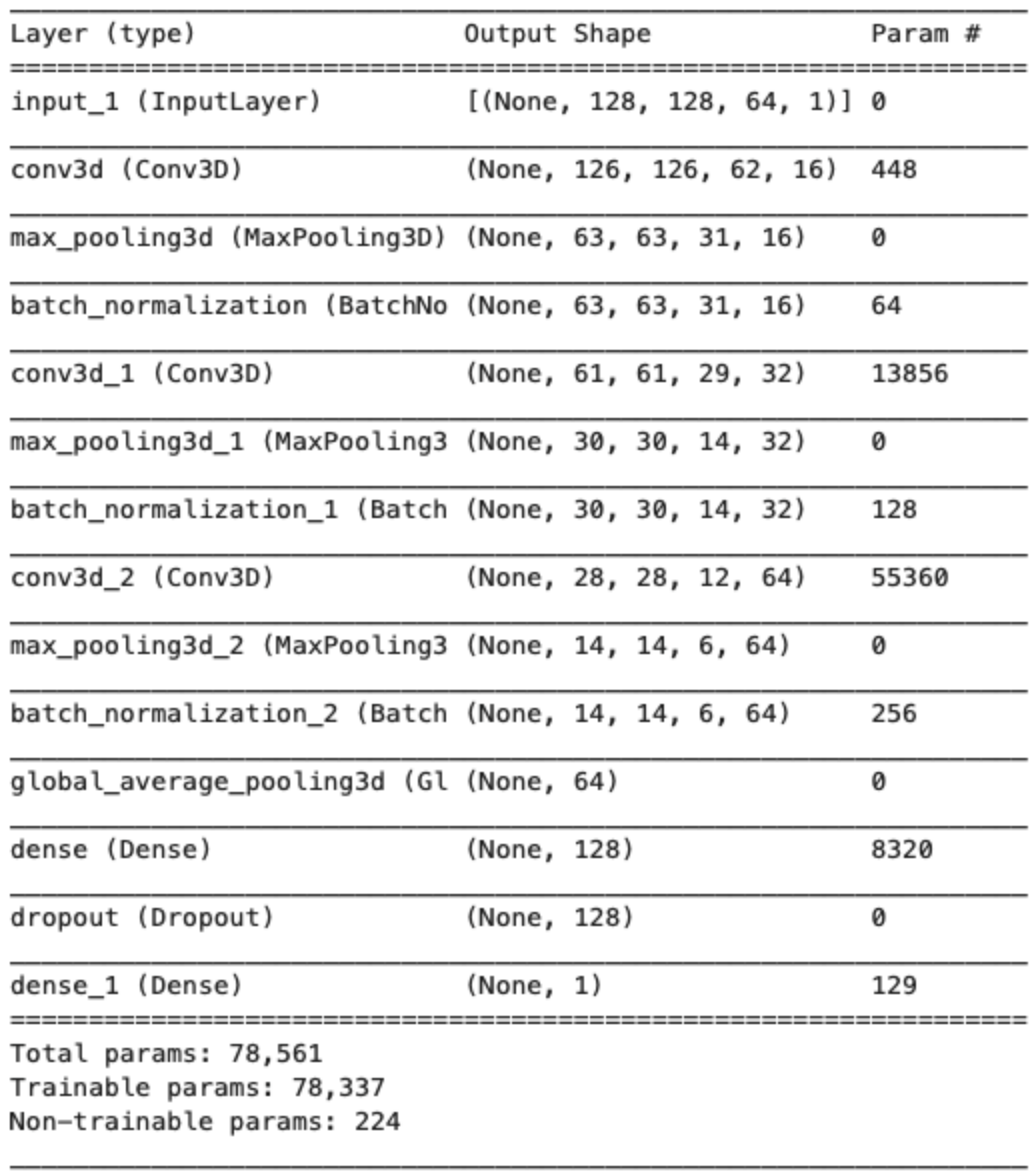
Define a 3D convolutional neural network

We have made the code into each block to make it clear to understand. Each block has a 3D Convolutional Layer with kernel size 3, activation function as “relu” and filter size is different to each layer to accommodate convolution. This Convolutional Layer is followed by a MaxPooling layer with pool size =2. Then we Normalize it and send it to the next block.

The final block consists of Global average pooling 3D layer , Dense layer and then we have Dropout layer.

Finally we return the model generated.

```python
def get_model(width=128, height=128, depth=64):
    """Build a 3D convolutional neural network model."""
    inputs = keras.Input((width, height, depth, 1))

    x = layers.Conv3D(filters=16, kernel_size=3, activation="relu")(inputs)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv3D(filters=32, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv3D(filters=64, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.GlobalAveragePooling3D()(x)
    x = layers.Dense(units=128, activation="relu")(x)
    x = layers.Dropout(0.3)(x)

    outputs = layers.Dense(units=1, activation="sigmoid")(x)

    # Define the model.
    model = keras.Model(inputs, outputs, name="3dcnn")
    return model


# Build model.
model = get_model(width=128, height=128, depth=64)
model.summary()
```



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 128, 128, 64, 1)]	0
conv3d (Conv3D)	(None, 126, 126, 62, 16)	448
max_pooling3d (MaxPooling3D)	(None, 63, 63, 31, 16)	0
batch_normalization (BatchNo	(None, 63, 63, 31, 16)	64
conv3d_1 (Conv3D)	(None, 61, 61, 29, 32)	13856
max_pooling3d_1 (MaxPooling3	(None, 30, 30, 14, 32)	0
batch_normalization_1 (Batch	(None, 30, 30, 14, 32)	128
conv3d_2 (Conv3D)	(None, 28, 28, 12, 64)	55360
max_pooling3d_2 (MaxPooling3	(None, 14, 14, 6, 64)	0
batch_normalization_2 (Batch	(None, 14, 14, 6, 64)	256
global_average_pooling3d (Gl	(None, 64)	0
dense (Dense)	(None, 128)	8320
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 78,561
Trainable params: 78,337
Non-trainable params: 224

Fig: 3D CNN model layers and no.of parameters in each layer.

Training Parameters

- Learning Rate: 0.00001
- We used Exponential Decay Function for scheduling learning rate.
- Loss Function: Binary Cross Entropy
- Optimizer: Adam Optimizer

```
initial_learning_rate = 0.00001
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=100000, decay_rate=0.96, staircase=True
)
model.compile(
    loss="binary_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),
    metrics=["acc"],
)

# Define callbacks.
checkpoint_cb = keras.callbacks.ModelCheckpoint(
    "Facial_feature_model.h5", save_best_only=True
)
early_stopping_cb = keras.callbacks.EarlyStopping(monitor="val_acc", patience=15)

# Train the model, doing validation at the end of each epoch
epochs = 20
model.fit(
    train_dataset_facial,
    validation_data=test_dataset_facial,
    epochs=epochs,
    shuffle=True,
    verbose=1,
    callbacks=[checkpoint_cb, early_stopping_cb],
)
```

Learning Rate in a Neural network:

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process

Exponential Decay Function:

A LearningRateSchedule that uses an exponential decay schedule. When training a model, it is often useful to lower the learning rate as the training progresses. This schedule applies an exponential decay function to an optimizer step, given a provided initial learning rate.

Binary Cross Entropy Function:

Neural networks are trained using stochastic gradient descent and require that you choose a loss function when designing and configuring your model.There are many loss functions to choose from and it can be challenging to know what to choose, or even what a loss function

is and the role it plays when training a neural network.

Computes the cross-entropy loss between true labels and predicted labels. Use this cross-entropy loss for binary (0 or 1) classification applications. The loss function requires the following inputs:

- y_true (true label): This is either 0 or 1.
- y_pred (predicted value): This is the model's prediction, i.e, a single floating-point value which either represents a logit, (i.e, value in [-inf, inf] when from_logits=True) or a probability (i.e, value in [0., 1.] when from_logits=False).

Optimizer:

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problems involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm. The **Adam optimization algorithm** is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.

Checkpoint is used to save the model generated.

This is the output from the Jupyter kernel once we train the model for 20 epochs. This shows the training_loss, training_acc, Validation_acc and validation_loss.

```
Epoch 17/20
58/58 [==============================] - 135s 2s/step - loss: 0.1470 - acc: 0.9677 - val_loss: 0.1506 - val_acc: 0.9644
Epoch 18/20
58/58 [==============================] - 135s 2s/step - loss: 0.1216 - acc: 0.9846 - val_loss: 0.1493 - val_acc: 0.9644
Epoch 19/20
58/58 [==============================] - 135s 2s/step - loss: 0.1309 - acc: 0.9846 - val_loss: 0.1469 - val_acc: 0.9644
Epoch 20/20
58/58 [==============================] - 135s 2s/step - loss: 0.1156 - acc: 0.9827 - val_loss: 0.1443 - val_acc: 0.9644
```

After 20 epochs we have achieved very good results. It is not overfitted or under-fitted. Since

we ran everything on GPU , it took us around 2mins for each epoch and 40 mins for training.

Model Accuracy and Loss Plots

Loss can be seen as a distance between the true values of the problem and the values predicted by the model. Greater the loss is, more huge is the errors you made on the data. Accuracy can be seen as the number of errors you made on the data.

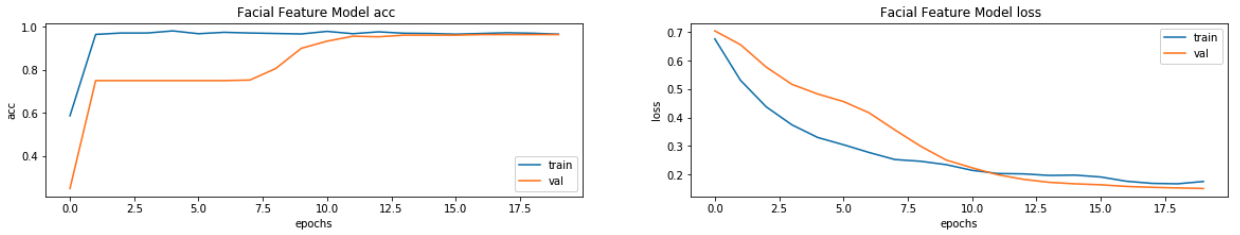


Fig: Facial Feature Model accuracy and model loss

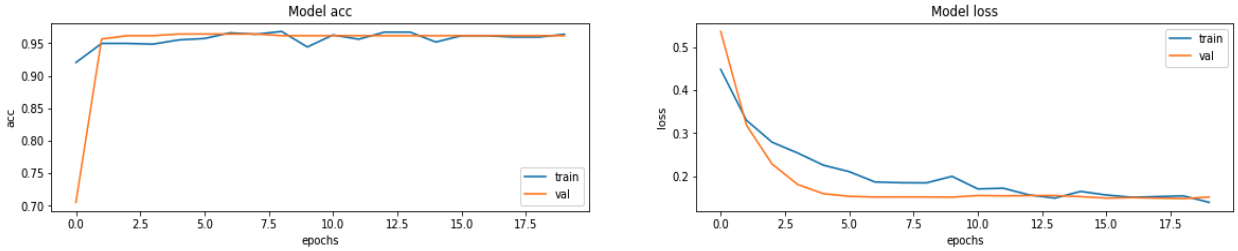


Fig: Brain Feature Loss Model accuracy and model loss

The above figure has Epochs on x-axis and accuracy , loss on y-axis respectively on first and second plots.

Predictions

```
class_names = ["Model  having", "Model not having"]
for score, name in zip(scores, class_names):
    print(
        " Confidence of our %s Recognizable facial features in MRI : %.2f percent "%(name, (100 * score))
    )
```

```
Confidence of our Model  having Recognizable facial features in MRI : 3.98 percent
Confidence of our Model not having Recognizable facial features in MRI : 96.02 percent
```

```
class_names = ["Model not having", "Model having"]
for score, name in zip(scores, class_names):
    print(
        " %s Brain feature loss:%.2f percent"
        % (name, (100 * score))
    )
```

```
Model not having Brain feature loss:98.29 percent
Model having Brain feature loss:1.71 percent
```

The above two screenshots show our predictions for the Recognizable facial features model and Brain Feature Loss prediction model.

Conclusion

As per our problem statement, we are able to predict if there are any Facial features in the Brain MRI. We also built another model for predicting Brain Feature loss in the MRI scans. The task has been made easy by choosing a 3D CNN model which reduced our effort to predict the requirements mentioned in the problem statement. We achieved very good results in terms of accuracy and loss for validation. We also achieved very good prediction results for facial feature recognition and Brain Feature loss. CNN's are very useful in image processing so it is highly recommended when we are dealing with classification of images. Facial feature model prediction is 96% confident and Brain feature loss model prediction is 98%. We also suggest training and running the model on GPU in order to save time and effort when dealing with Neural networks.

Takeaways:

We really appreciate professor Wang for providing dataset for the project and which has made the task easy. We also appreciate our TA for clarifying doubts whenever we reach him.

We learnt a lot of interesting details of medical data and use of data mining in those fields. We feel happy to work on good projects like this.

References

[1]Ahmed, E., Saint, A., Shabayek, A.E.R., Cherenkova, K., Das, R., Gusev, G., Aouada, D., Ottersten, B.: A survey on deep learning advances on different 3d data representations. arXiv preprint arXiv:1808.01462 (2018)

[2]LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature 521(7553), 436 (2015)

[3]Gordaliza, P.M., Vaquero, J.J., Sharpe, S., Gleeson, F., Munoz-Barrutia, A.: A multi-task self-normalizing 3d-cnn to infer tuberculosis radiological manifestations. arXiv preprint arXiv:1907.12331 (2019)

[4]Wu, W., Li, X., Du, P., Lang, G., Xu, M., Xu, K., Li, L.: A deep learning system that generates quantitative ct reports for diagnosing pulmonary tuberculosis. arXiv preprint arXiv:1910.02285 (2019)

[5]Yang, J., Huang, X., Ni, B., Xu, J., Yang, C., Xu, G.: Reinventing 2d convolutions for 3d images. arXiv pp. arXiv–1911 (2019)

[6]Kazlouski, S.: Imageclef 2019: Ct image analysis for tb severity scoring and ct report generation using autoencoded image features. CLEF2019 Working Notes 2380, 9–12 (2019)

[7]Gao, X.W., Hui, R., Tian, Z.: Classification of ct brain images based on deep learning networks. Computer methods and programs in biomedicine 138, 49–56 (2017)