

UNIVERSITY PARTNER



# Image Classification

<Brain tumor detection>

University Id:	2227738
Name:	Nishant Niraula
Group:	L6CG2
Lecturer:	Siman Giri
Tutor:	Sunita Parajuli

## Table of Contents

1. Introduction .....	1
2. Methodology .....	2
2.1 Image Classification with Fully Connected Neural Network .....	2
2.2 Image Classification with Convolutional Neural Network .....	9
2.3 Image Classification with Transfer Learning .....	15
3 Final Discussion.....	20

## Table of Figure

Figure 1:Model Summary FCN.....	2
Figure 2:training loss FCN.....	3
Figure 3:Building model FCN .....	4
Figure 4:Training loss FCN .....	5
Figure 5:Trainin Accracy FCN.....	6
Figure 6:Accuracy FCN.....	7
Figure 7:Evaluation FCN.....	8
Figure 8:Model Summary CNN .....	9
Figure 9:Model summary CNN.....	10
Figure 10:History CNN .....	10
Figure 11:Training Loss CNN.....	11
Figure 12: Plotting CNN .....	12
Figure 13:Training Loss CNN.....	12
Figure 14:Training accuracy CNN .....	13
Figure 15: Evaluation metrics CNN .....	14
Figure 16: Accuracy cnn.....	15
Figure 17:Model Transfer learning .....	16
Figure 18:Training Loss Transfer Learning .....	17
Figure 19:Plotting Transfer Learning.....	17
Figure 20:Plotting Transfer Learning.....	18
Figure 21:Train accuracy Transfer Learning .....	19
Figure 22:Accuracy Transfer learning .....	19
Figure 23:Code Transfer learning .....	20
Figure 24:Comparision of the 3 architecture .....	21

## 1. Introduction

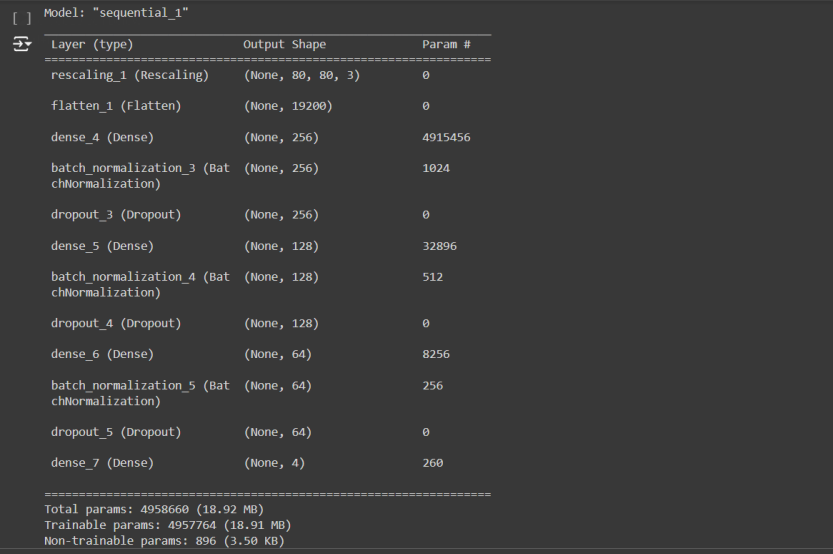
- The project we have chosen for the image classification is Brain tumor classification. The tumor here consists of MRI scan that is collected different patients with brain tumors which is classified into 4 types of brain tumor. They are: meningioma, glioma, pituitary and no tumor. The main task of the image classification includes training the different models of the neural network. So the main goal of the project is to classify the tumor with high accuracy. (Mohsen, 2023)  
The aims and objective includes: -
  - a. Preprocessing the brain tumor data
  - b. Evaluating the different neural network models for classification of image
  - c. Implementing optimization which helps in improving the performance of the model
- Firstly, we need to understand the data. Also we need to research about the image format of the brain tumor images. And we need to check for any imbalance in the dataset as it can create biasness for certain type of tumor classes. For this project we are going to implementing 3 different models. They are: Fully Connected Neural Network(FCNN), Convolutional Neural Network(CNN) and Transfer Learning. The outline of the report is structure as follows:
  - i. Introduction
  - ii. Understanding of data
  - iii. Data preparation and preprocessing
  - iv. Development of the model
  - v. Evaluation of the model and performance
  - vi. Conclusion from the evaluation metrics

## 2. Methodology

### 2.1 Image Classification with Fully Connected Neural Network

#### Model Summary

There are 8 layers in the FCNN architecture. The input layer consists of rescaling layer which rescales the values into the range of [0,1]. The other layer is Flatten layer which flattens the images into one array. The next layer is Dense layer where there is 256 neuron made with relu activation function. The other layer is Batch Normalization layer where this layer normalizes the previous layer and improves the speed. Again the dropout layer is used where there is random connection break which helps in over fitting of the model. Again dense layer is used with 128 neurons and with relu activation function. Again the batch normalization layer is used which normalizes the previous batch size. Again the dropout layer is used with the 30% dropout rate. Lastly in the output layer dense layer is used and softmax activation function is used which helps in different class classification. Altogether there are 8 layers with 3 dense layer and 2 batch normalization layer and 2 dropout layer and other two flattening and rescaling layer. The hyper-parameters are: Activation function, dropout rate, Batch normalization, Input shape etc.



```
[ ] Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 80, 80, 3)	0
flatten_1 (Flatten)	(None, 19200)	0
dense_4 (Dense)	(None, 256)	4915456
batch_normalization_3 (Batch Normalization)	(None, 256)	1024
dropout_3 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 128)	32896
batch_normalization_4 (Batch Normalization)	(None, 128)	512
dropout_4 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 64)	8256
batch_normalization_5 (Batch Normalization)	(None, 64)	256
dropout_5 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 4)	260

=====  
Total params: 4958660 (18.92 MB)  
Trainable params: 4957764 (18.91 MB)  
Non-trainable params: 896 (3.58 KB)

Figure 1: Model Summary FCN

There are several layers in the model architecture as we have discussed above. There is the total parameter of 4,958,660 where 4,957,764 is only trainable and 896 are non-trainable parameters. This is for the multiclass

classification of 4 different classes. The model uses different techniques like normalization, reshape and dropout etc.

### Training of the model

The loss function used in the training of the model is categorical crossentropy. The optimizer used in this model is adam. And the model is trained for 15 epochs.

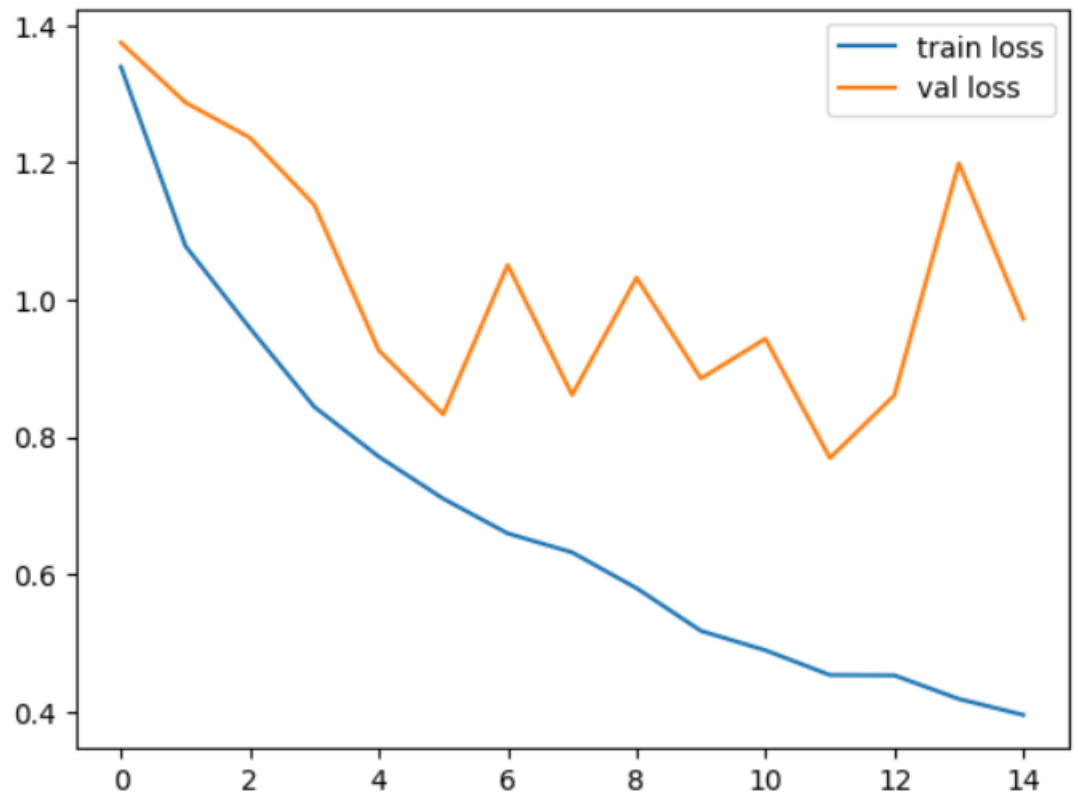


Figure 2:training loss FCN

The training loss tends to decrease which shows the model is learning from the data. However, the validation loss tends to decrease initially but is gradually increase over the more epochs. So we can say that model might be overfitting which requires generalization.

```

# Define the number of classes in the dataset
num_classes = 4

input_size = 80

tf.random.set_seed(42)

# Create a Sequential model
model_fcn = keras.Sequential([

    # Rescale the input pixel values to the range [0, 1]
    layers.Rescaling(1./255, input_shape=(input_size, input_size, 3)),

    # Flatten the input images
    layers.Flatten(),

    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Dense(64, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Dense(num_classes, activation='softmax')

])

```

Figure 3: Building model FCN

Firstly, the input image is rescaled in the input layer. The flatten layer is applied and with ReLu activation function, dropout, there is 3 dense layer passed. For the output layer softmax activation function is used for multi-class classification.

## Finding and Discussion

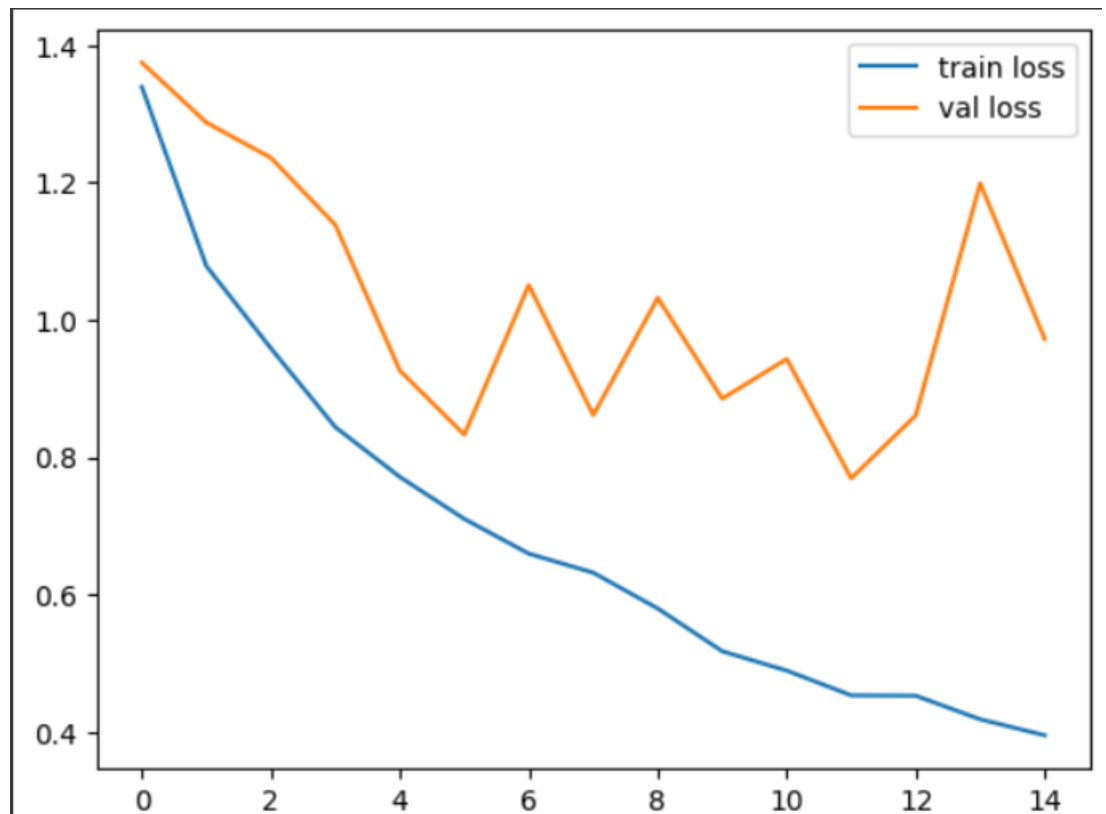


Figure 4: Training loss FCN

The training loss tends to decrease over more number of epochs which indicated that model is learning well and validation loss tends to decrease which indicated the model is learning well on train data.



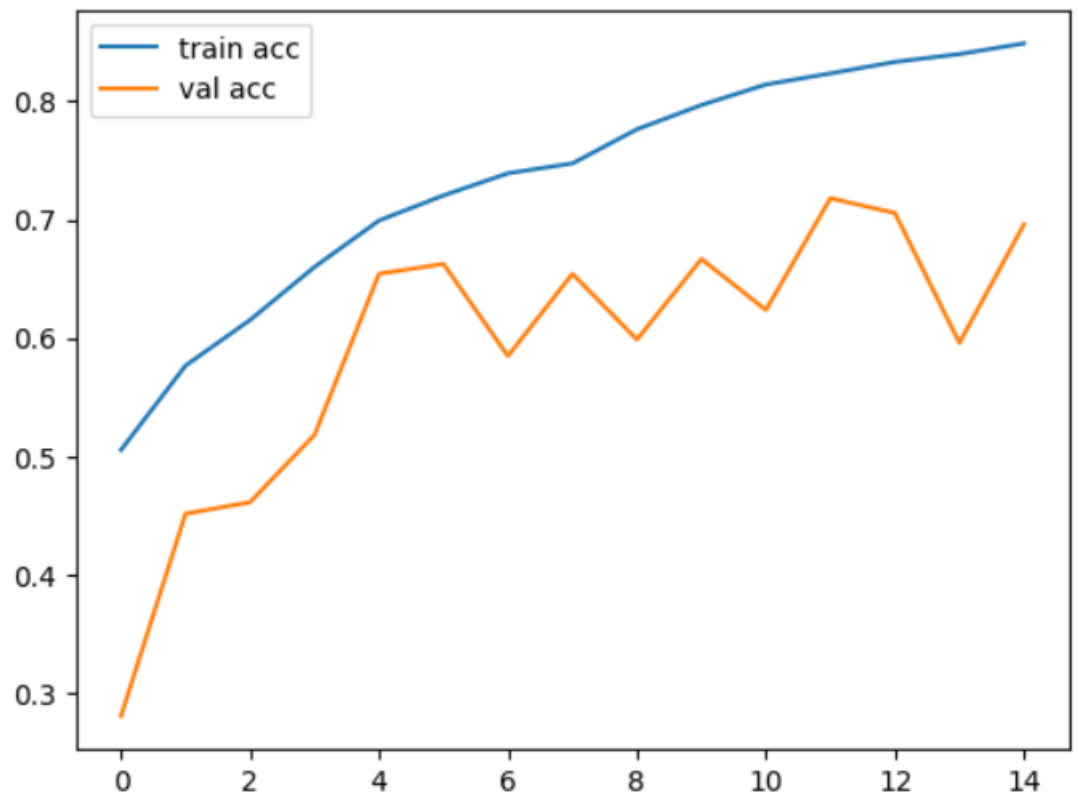


Figure 5: Trainin Accracy FCN

The train accuracy tends to increase when there is increase in the epochs which indicate our model is learning well and validation accuracy tends to increase which is also a good symbol that model is learning well.

```
29/29 [=====] - 0s 4ms/step
Accuracy: 0.7122222222222222
Confusion matrix:
[[217  23   2  11]
 [104 137  10  13]
 [ 14  14  89   4]
 [ 44  12   8 198]]

Classification report:
              precision    recall  f1-score   support

     0           0.57       0.86       0.69       253
     1           0.74       0.52       0.61       264
     2           0.82       0.74       0.77       121
     3           0.88       0.76       0.81       262

 accuracy          0.71       0.71       0.71       900
 macro avg         0.75       0.72       0.72       900
weighted avg         0.74       0.71       0.71       900

Accuracy score: 0.7122222222222222
F1 score: 0.7119267156408192
```

Figure 6:Accuracy FCN

In the unseen data there is the accuracy of 71% percentage which can be considerable.

```

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Get predicted probabilities for test set
y_pred = model_fcn.predict(X_test)

# Get predicted classes by taking argmax of predicted probabilities
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test_one_hot, axis=1)
accuracy = accuracy_score(y_true_classes, y_pred_classes)
print("Accuracy:", accuracy)

print("Confusion matrix:")
confusion_matrix = confusion_matrix(y_true_classes, y_pred_classes)
print(confusion_matrix)

print("\nClassification report:")
print(classification_report(y_true_classes, y_pred_classes))

print("\nAccuracy score:", accuracy_score(y_true_classes, y_pred_classes))
print("F1 score:", f1_score(y_true_classes, y_pred_classes, average='weighted'))

```

Figure 7: Evaluation FCN


The above code calculates the performance of the testing dataset. Here we are calculating the probability of the predicted and also converting of the classes if performed and there is comparison on the true classes. Evaluation metrics like confusion matrix, precision, recall, F-1 score is also calculated to evaluate model performance.

So from these thing we can say that FCNN has the accuracy of 71% in the testing data. But we have other neural network which has high accuracy then the FCNN model so for that we will discuss on the below section.

## 2.2 Image Classification with Convolutional Neural Network

### Model Summary

The model has the total of 13 layers. The layers are Conv2D layer, MaxPooling2D layer, Dropout layer, Flatten layer and Dense layer. The model hyper-parameters are: Optimizer, loss function, batch size, epochs, validation split.



Model: "sequential\_5"

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 78, 78, 32)	896
conv2d_19 (Conv2D)	(None, 76, 76, 32)	9248
max_pooling2d_9 (MaxPooling2D)	(None, 38, 38, 32)	0
dropout_18 (Dropout)	(None, 38, 38, 32)	0
conv2d_20 (Conv2D)	(None, 36, 36, 64)	18496
conv2d_21 (Conv2D)	(None, 34, 34, 64)	36928
max_pooling2d_10 (MaxPooling2D)	(None, 17, 17, 64)	0
dropout_19 (Dropout)	(None, 17, 17, 64)	0
conv2d_22 (Conv2D)	(None, 15, 15, 256)	147712
conv2d_23 (Conv2D)	(None, 13, 13, 256)	590080
max_pooling2d_11 (MaxPooling2D)	(None, 6, 6, 256)	0
dropout_20 (Dropout)	(None, 6, 6, 256)	0

Figure 8: Model Summary CNN

```

dropout_19 (Dropout)      (None, 17, 17, 64)      0
conv2d_22 (Conv2D)        (None, 15, 15, 256)     147712
conv2d_23 (Conv2D)        (None, 13, 13, 256)     590080
max_pooling2d_11 (MaxPooli (None, 6, 6, 256)      0
ng2D)
dropout_20 (Dropout)      (None, 6, 6, 256)      0
flatten_5 (Flatten)       (None, 9216)            0
dense_14 (Dense)          (None, 1024)            9438208
dropout_21 (Dropout)      (None, 1024)            0
dense_15 (Dense)          (None, 4)               4100

=====
Total params: 10245668 (39.08 MB)
Trainable params: 10245668 (39.08 MB)
Non-trainable params: 0 (0.00 Byte)

```

Figure 9:Model summary CNN

There is 3 convolutional layer, 3 max-polling layer, and other 3 dense layer in the CNN architecture. Also there is a flatten layer and 3 dense layer. There is 10245668 parameters which are trainable,

```

[ ] epochs = 15
    history_cnn= model_cnn.fit(
        x_train,
        y_train_one_hot,
        batch_size=32,
        epochs=epochs ,
        validation_split=0.2
    )

```

Figure 10:History CNN

The training of the cnn model is done in this section and the model is trained on Xtrain and ytrain dataset and there is the epochs of 15 and batch size is 32 and also there is 20% data for the validation split. All the progress of training result and validation result are in the history\_cnn variable.

## Training of the Model

The loss function used in the training of the model is categorical crossentropy. The optimizer used here is adam. There are 15 epochs for training the model.

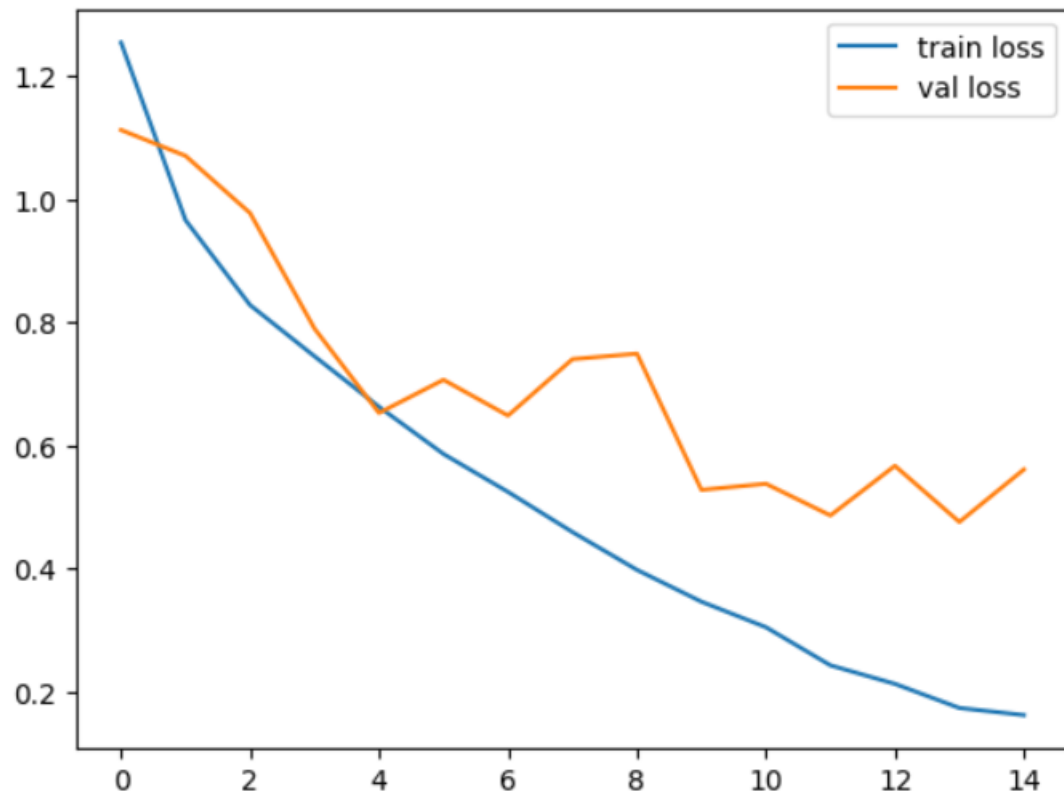


Figure 11: Training Loss CNN

Here the training loss is decreasing over the increase of the epochs which indicates the model is learning well from the training data and validation loss tends decrease which indicates model has the ability to generalize the new data.

```

] # Plot the training and validation loss
plt.plot(history_cnn.history['loss'], label='train loss')
plt.plot(history_cnn.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# Plot the training and validation accuracy
plt.plot(history_cnn.history['accuracy'], label='train acc')
plt.plot(history_cnn.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')

```

Figure 12: Plotting CNN

This code plots the loss of validation and training loss while in the training of model. first block of code shows the training and validation loss and the second block of code shows the training and validation accuracy.

## Finding and Discussion

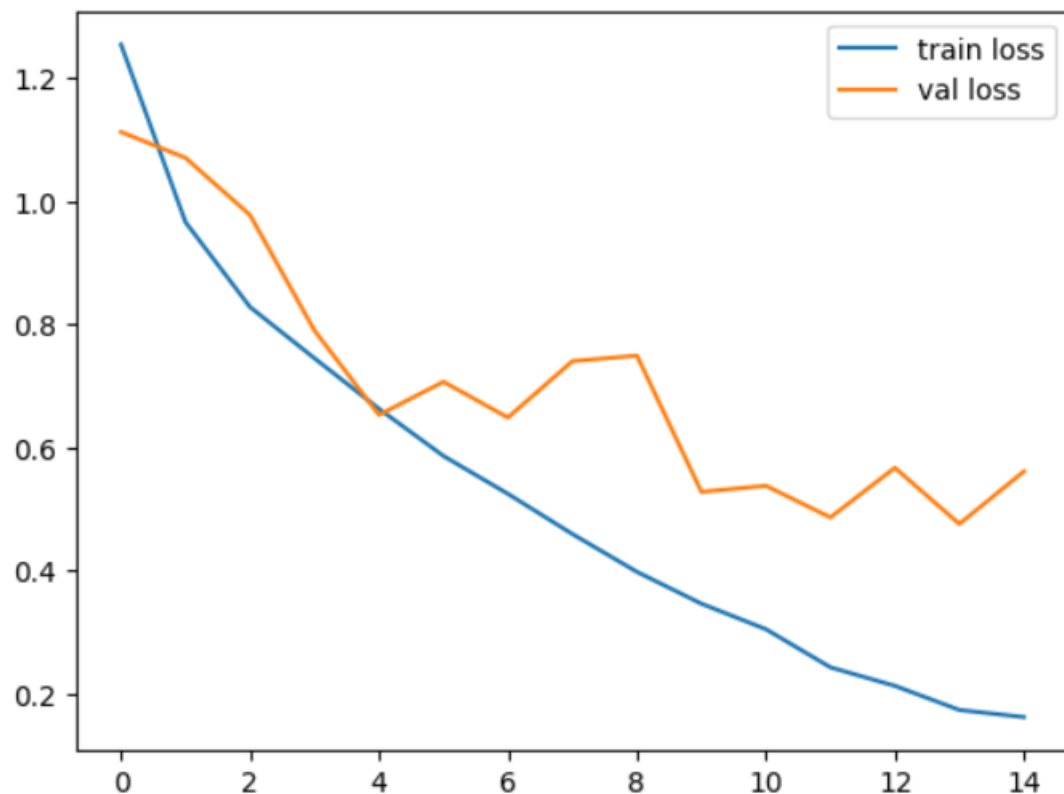


Figure 13: Training Loss CNN

Here the training loss is decreasing over the increase of the epochs which indicates the model is learning well from the training data and validation loss tends decrease which indicates model has the ability to generalize the new data.

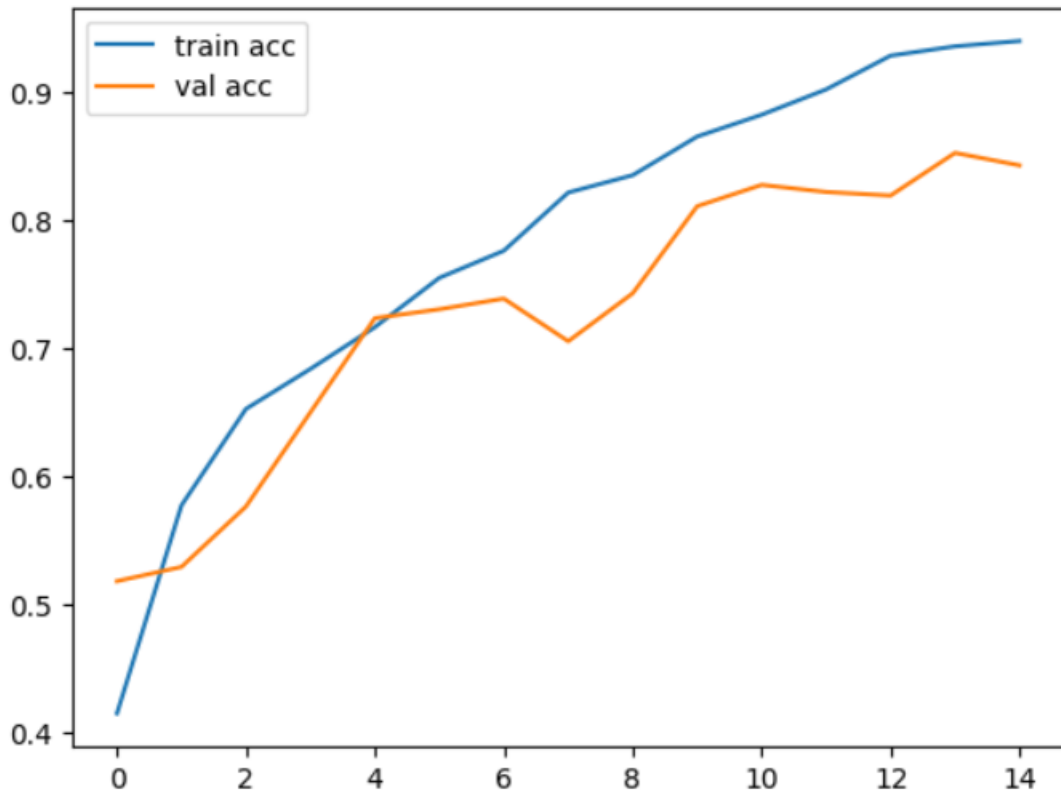


Figure 14: Training accuracy CNN

Here, training and validation accuracy tends to increase over the increasing period of epochs that indicates that the model is learning well over the period of epochs and accuracy is increasing.



```
29/29 [=====] - 1s 17ms/step
Accuracy: 0.8377777777777777
Confusion matrix:
[[225  15   0  13]
 [ 55 179  12  18]
 [  9   6 103   3]
 [ 10   3   2 247]]

Classification report:
              precision    recall  f1-score   support

     0       0.75       0.89       0.82       253
     1       0.88       0.68       0.77       264
     2       0.88       0.85       0.87       121
     3       0.88       0.94       0.91       262

 accuracy          0.84          900
  macro avg       0.85       0.84       0.84       900
 weighted avg     0.84       0.84       0.84       900

Accuracy score: 0.8377777777777777
F1 score: 0.8352439146170718
```

Figure 15: Evaluation metrics CNN

Here the accuracy is shown in the unseen data and F1 score is also displayed of the classification model of the 4 classes of the dataset. The model has the accuracy of 83% and F1 score of 83%. The confusion matrix and other evaluation other detailed result of the model of each classes.

```

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Generate predictions for the test dataset
y_pred = model_cnn.predict(X_test)

y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test_one_hot, axis=1)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_true_classes, y_pred_classes)
print("Accuracy:", accuracy)

print("Confusion matrix:")
confusion_matrix = confusion_matrix(y_true_classes, y_pred_classes)
print(confusion_matrix)

print("\nClassification report:")
print(classification_report(y_true_classes, y_pred_classes))

# Print the accuracy and F1-score
print("\nAccuracy score:", accuracy_score(y_true_classes, y_pred_classes))
print("F1 score:", f1_score(y_true_classes, y_pred_classes, average='weighted'))

```

Figure 16: Accuracy cnn

The above code calculates the performance of the testing dataset. Here we are calculating the probability of the predicted and also converting of the classes if performed and there is comparison on the true classes. Evaluation metrics like confusion matrix, precision, recall, F-1 score is also calculated to evaluate model performance.

## 2.3 Image Classification with Transfer Learning

### Model Summary

The model has 19 layers. The layers are: Input layer, Conv2D layer, MaxPooling layer, flatten layer and lastly Dense layer. The model hyper-parameters are: Number of filters, kernel size, Pool size, Activation Function, Loss Function, Optimizer, Epochs, Batch size, Validation split.

```

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
import tensorflow as tf
tf.random.set_seed(666)

X_train = X_train
y_train = y_train_one_hot
input_shape = (input_size, input_size, 3)

base_model = MobileNetV2(include_top=False, input_shape=input_shape)

```

*Figure 17: Model Transfer learning*

Here Mobile NetV2 pre trained model is used which is the base model for the extraction of the feature. The shape of the input is defined and random seed is also set for its reproducibility.

## **Training of the Model**

Here, I have frozen the layers of the pre-trained model which means the weights of the layers will not update while training. The weights update while training of the model which is also known to be fine-tuning. Through these we can use pre-trained model to learn the features from our data and fine-tune.

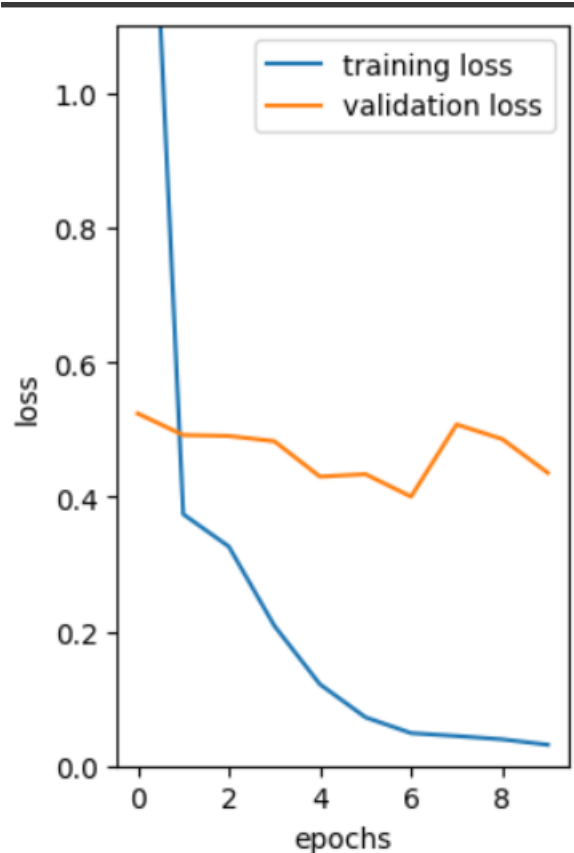


Figure 18: Training Loss Transfer Learning

The training loss tends to decreasing over the period of epochs which shows the model accuracy is increasing and also the validation loss tends to decrease which tells us that model accuracy is increasing.

```
# Plot the training and validation accuracy of the model
plt.plot(history_MobileNetV2.history['accuracy'])
plt.plot(history_MobileNetV2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

Figure 19: Plotting Transfer Learning

```
[ ] # Plot the training and validation loss of the MobileNetV2 model
plt.subplot(1,2,1)
plt.plot(history_MobileNetV2.history['loss'], label='training loss')
plt.plot(history_MobileNetV2.history['val_loss'], label='validation loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.ylim(0,1.1)
plt.show()
```

*Figure 20:Plotting Transfer Learning*

This code plots the loss of validation and training loss while in the training of model. first block of code shows the training and validation loss and the second block of code shows the training and validation accuracy.

## **Findings and Discussion**

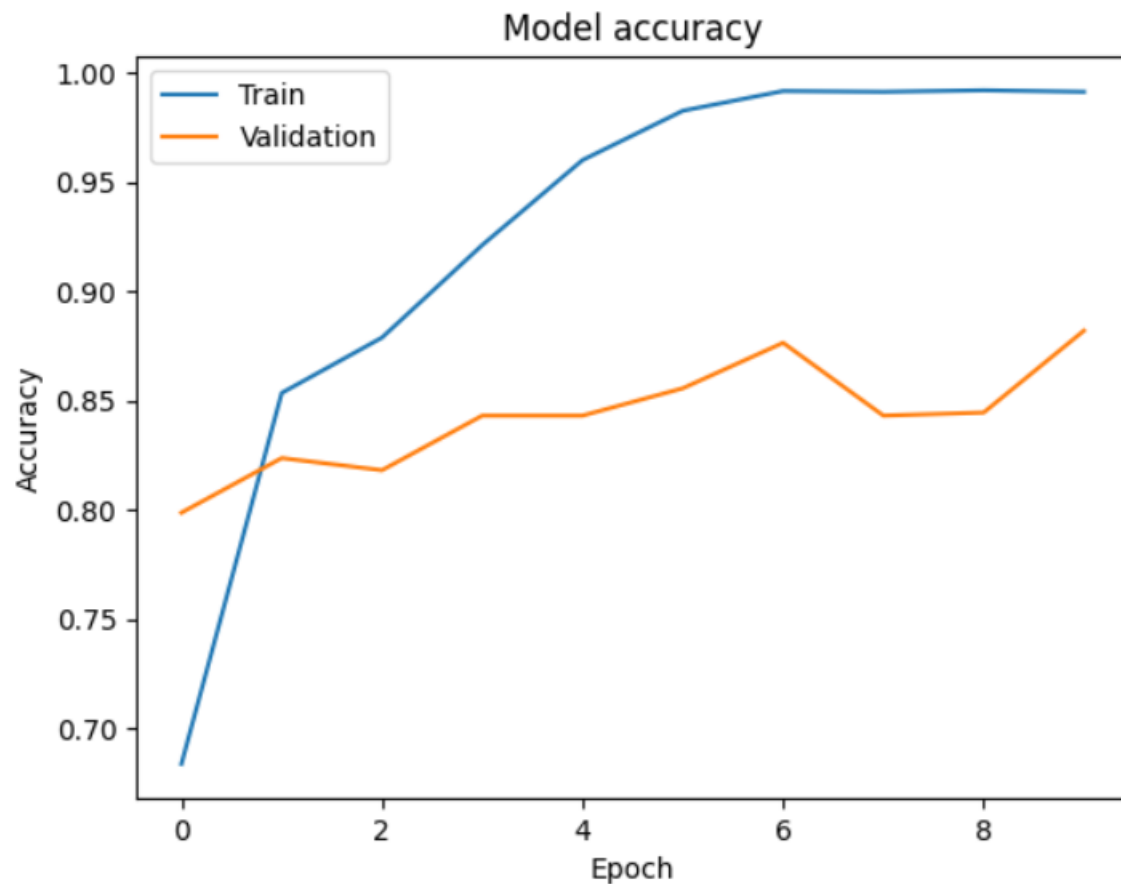


Figure 21: Train accuracy Transfer Learning

The model accuracy tends to increase over the increasing period of epochs. And also the validation is slightly increase which indicated the model is learning well and the accuracy is increase over the period of epochs.

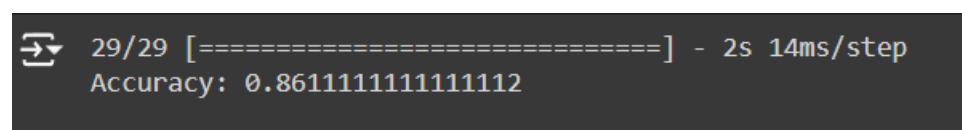
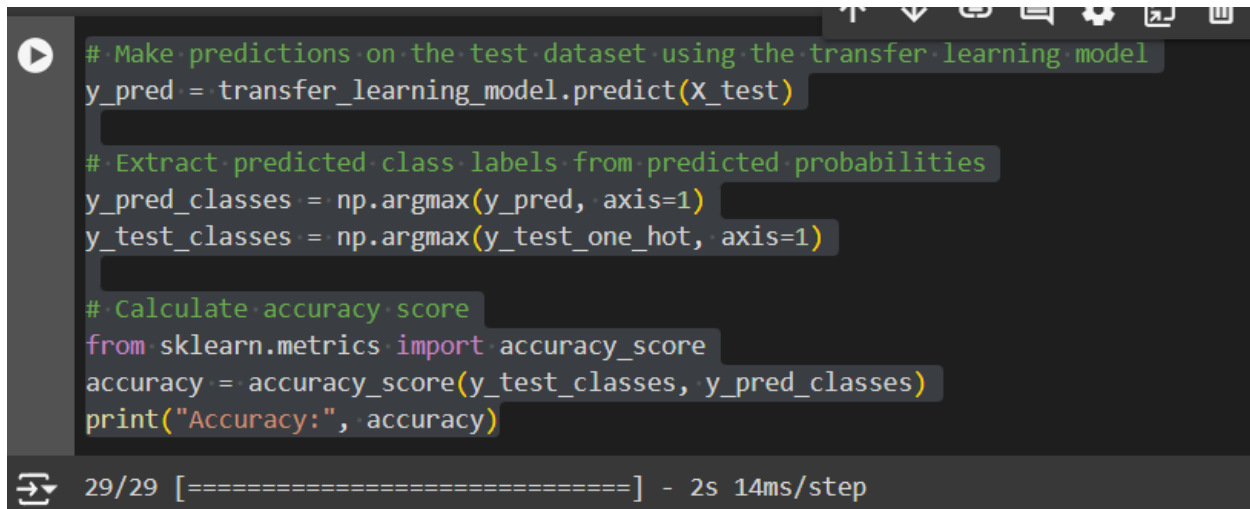


Figure 22: Accuracy Transfer learning

The model has the test accuracy of 86% which is also the highest of all the models. This indicates that transfer learning is way more effective than other models.



```
# Make predictions on the test dataset using the transfer learning model
y_pred = transfer_learning_model.predict(x_test)

# Extract predicted class labels from predicted probabilities
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test_one_hot, axis=1)

# Calculate accuracy score
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test_classes, y_pred_classes)
print("Accuracy:", accuracy)
```

29/29 [=====] - 2s 14ms/step

Figure 23:Code Transfer learning

Transfer learning model performance is evaluated here. Prediction is made using the model and the class of the predicted label is extracted and accuracy is calculated comparing with predicted and true labels. Lastly the accuracy score is printed using the model performance.

So, finally we can say that transfer learning has the highest accuracy among the other models followed by CNN and FCN architecture. Also the increasing epochs may help in more training iteration which helps in increasing the accuracy of the model.

### 3. Final Discussion

- Overall, we can say that CNN has outperformed the FCNN models for classification of brain tumor as they have the ability to capture more information. On the other hand transfer learning with fine tuning has the great accuracy and has the advantage of pre-trained model which can adapt quickly in learning new information which makes the transfer learning the most powerful approach. However, the models have their own advantage and disadvantage we can not say that transfer learning is the best due to its accuracy. Each model architecture has

its own importance. So below here there is the accuracy comparison of the 3 different architecture.

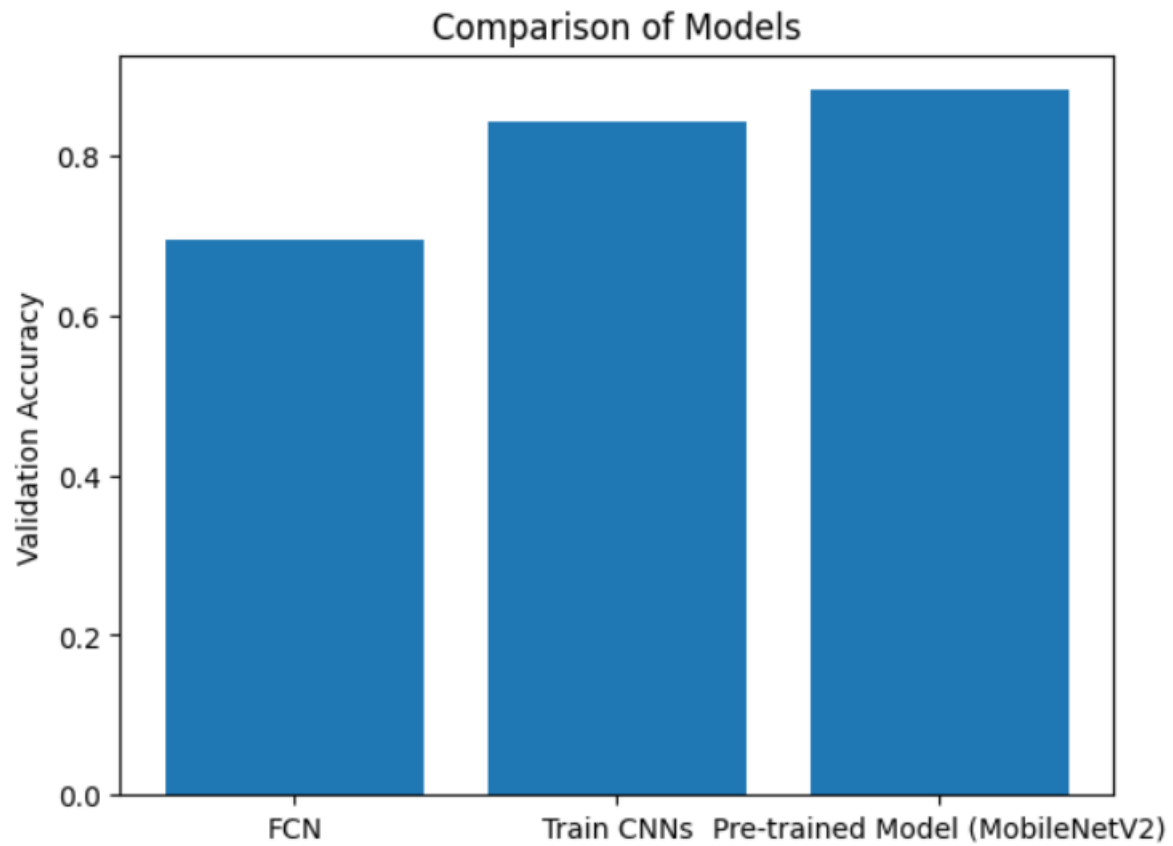


Figure 24: Comparison of the 3 architecture



#### **4. References**

Mohsen, H., 2023. *Classification using deep learning neural networks for brain tumors - ScienceDirect*. [Online]  
Available at: <https://www.sciencedirect.com/science/article/pii/S2314728817300636>  
[Accessed 10 5 2024].