SATELLITE IMAGE CLASSIFICATION

IEE 577: Term Project Report

AJEY MUSUWATHI RAJKUMAR - 1220003616

VIVIEK NAGARAJAN - 1219798034

ARCHITTA BASKAR - 1219973144

Table of Contents

1 Introduction to Problem Setup

Satellite imagery data provides essential information that helps in monitoring various applications such as image fusion, change detection and land cover classification. Satellite image analysis is a key technique used to obtain information related to the earth's resources and environment. There are many different use cases for the Satellite images such as

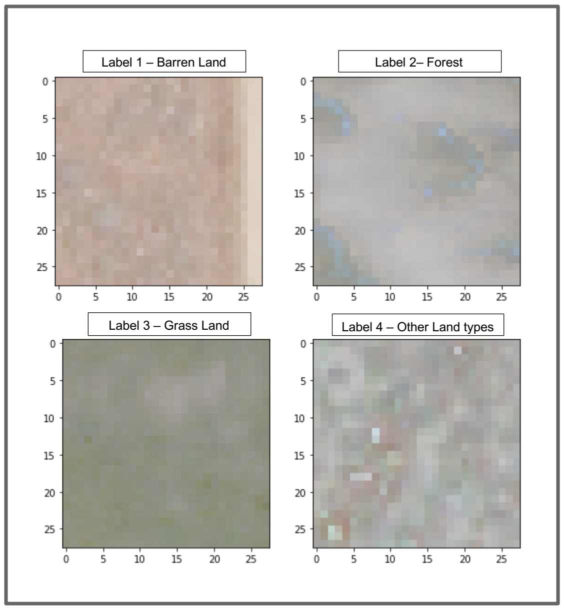
- Providing a base map for graphical reference and assisting planners and engineers
- Extracting mineral deposits with remote sensing based spectral analysis
- Disaster mitigation planning and recovery
- Agriculture Development

Our Project focuses on Agricultural and rural development by zoning the large land area. With a growing demand for agriculture satellite image analysis will play a very important role in in the improvement of the present systems of acquiring and generating agricultural maps and resource data.

Our central point of our project will be to classify the images into four different label which are as follows

- Barren Land
- Forest
- Grass Land
- Other Land types

2 Dataset Exploration and Visualization



The images were extracted from the National Agriculture Imagery Program (NAIP) dataset. The authors used the uncompressed digital Ortho quarter quad tiles (DOQQs) which are GeoTIFF images, and the area corresponds to the United States Geological Survey (USGS) topographic quadrangles. The average image tiles are ~6000 pixels in width and ~7000 pixels in height, measuring around 200 megabytes each. The entire NAIP dataset for CONUS is ~65 terabytes. The imagery is acquired at a 1-m ground sample distance (GSD) with a horizontal accuracy that lies within six meters of photo-identifiable ground control points.

The images consist of 4 bands - red, green, blue, and Near Infrared (NIR). To maintain the high variance inherent in the entire NAIP dataset, we sample image patches from a multitude of scenes (a total of 1500 image tiles) covering different landscapes like rural areas, urban areas, densely forested, mountainous terrain, small to large water bodies, agricultural areas, etc. covering the whole state of California. An image labelling tool developed as part of this study was used to manually label uniform image patches belonging to a particular land cover class.

Once labelled, 28x28 non-overlapping sliding window blocks were extracted from the uniform image patch and saved to the dataset with the corresponding label. We chose 28x28 as the window size to maintain a significantly bigger context, and at the same time not to make it as big as to drop the relative statistical properties of the target class conditional distributions within the contextual window. Care was taken to avoid interclass overlaps within a selected and labelled image patch.

2.1 Contents of the Data:

- Each sample image is 28x28 pixels and consists of 4 bands - red, green, blue and near infrared.
- The training and test labels are one-hot encoded 1x4 vectors
- The four labels represent the four broad land covers which include barren land, trees, grassland and a class that consists of all land cover classes other than the above three.
- Training and test datasets belong to a disjoint set of image tiles.
- Each image patch is size normalized to 28x28 pixels.

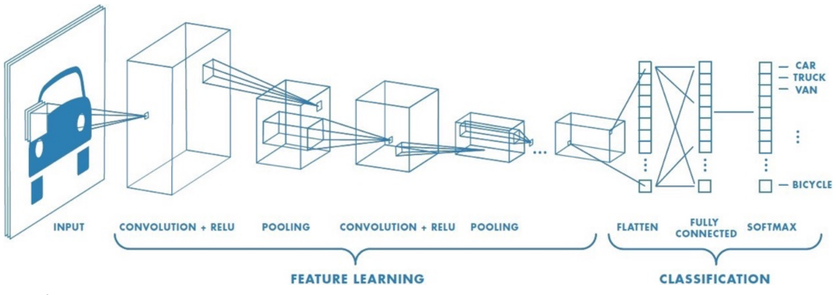
- Once generated, both the training and testing datasets were randomized using a pseudo-random number generator.

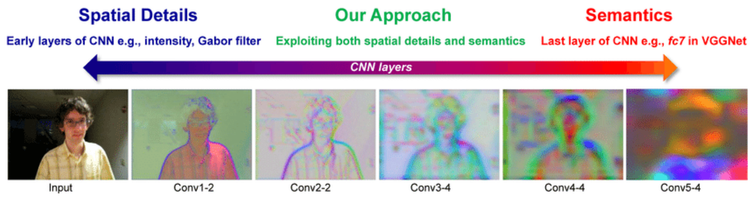
2.2 Data Files

- Xtrainsat4.csv: 400,000 training images, 28x28 images each with 4 channels
- ytrainsat4.csv: 400,000 training labels, 1x4 one-hot encoded vectors
- Xtestsat4.csv: 100,000 training images, 28x28 images each with 4 channels
- ytestsat4.csv: 100,000 training labels, 1x4 one-hot encoded vectors

3 Model Solutions and Interpretations

3.1 Convolution Neural Network




Spatial Details
Early layers of CNN e.g., intensity, Gabor filter
Our Approach
Exploiting both spatial details and semantics
Semantics
Last layer of CNN e.g., fc7 in VGGNet
CNN layers
Input
Conv1-2
Conv2-2
Conv3-4
Conv4-4
Conv5-4

For our project we have used Convolutional Neural Network because CNN are complex feed forward neural networks, and they are very appropriate for image classification and recognition because of its high accuracy. The architecture of a CNN is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area. The CNN follows a hierarchical model which works on building a network, like a funnel, and finally gives out a fully connected layer where all the neurons are connected to each other and the output is processed.

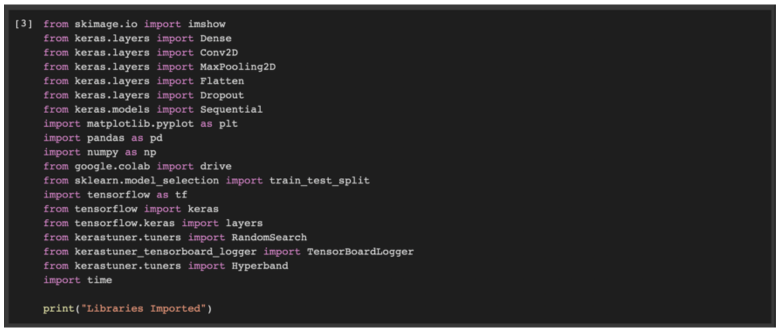
The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics. The CNN generates different set of filters to extract feature information from the images. Each filter generated could extract a different set of categories of features such as horizontal lines or vertical lines or contrast or bright spots, etc. Different filters enable the CNN neural network to deep learn more from the limited set of features provided by the data set.

3.2 TensorFlow

We have used TensorFlow to build our Neural Network because it can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training which has allowed for it to become popular. TensorFlow is an open-source library with various different built-in functions for the construction of the neural network, augmentation of the data, evaluation of the model and for hyperparameter tune the model for best performance.

3.3 Libraries used

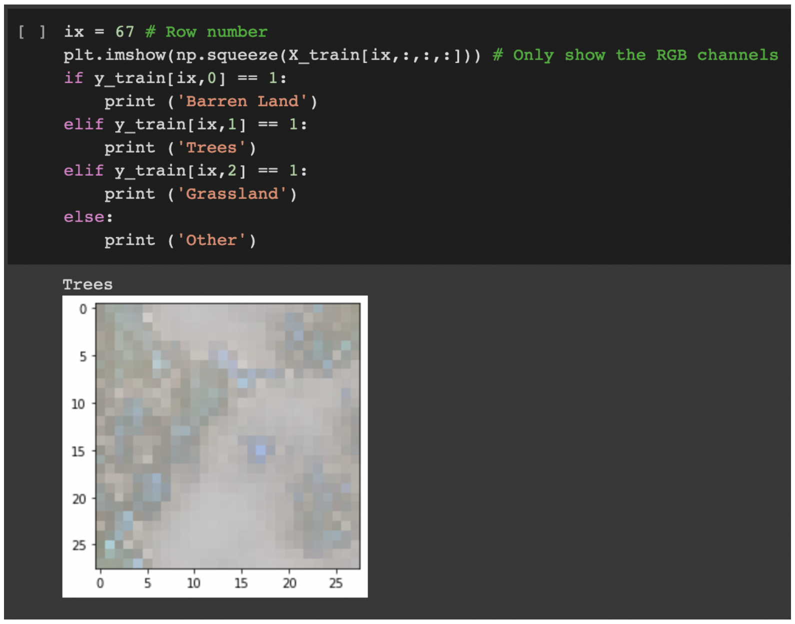
We have used a range of different libraries and function in this project, and it's all mentioned below in the figure,



```
[3] from skimage.io import imshow
    from keras.layers import Dense
    from keras.layers import Conv2D
    from keras.layers import MaxPooling2D
    from keras.layers import Flatten
    from keras.layers import Dropout
    from keras.models import Sequential
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np
    from google.colab import drive
    from sklearn.model_selection import train_test_split
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    from kerastuner.tuners import RandomSearch
    from kerastuner_tensorboard_logger import TensorBoardLogger
    from kerastuner.tuners import Hyperband
    import time

    print("Libraries Imported")
```

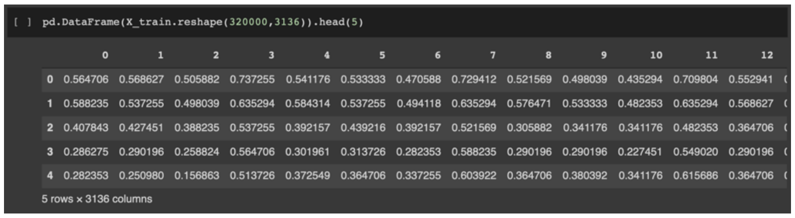
3.4 Data Handling and Data Exploration.

As mentioned in the previous we are using 400000 images are that are translated into .csv file with 3136 features. The data is uploaded to google drive and the drive is mounted to the colab and all the files are access through this method so circumvent to limitations of our local machines. The complete project is completely performed on the colab platform. Here is the visualization of the dataset as a table and images.



```
[ ] ix = 67 # Row number
    plt.imshow(np.squeeze(X_train[ix,:,:,:])) # Only show the RGB channels
    if y_train[ix,0] == 1:
        print ('Barren Land')
    elif y_train[ix,1] == 1:
        print ('Trees')
    elif y_train[ix,2] == 1:
        print ('Grassland')
    else:
        print ('Other')
```

```
Trees
```



```
[ ] pd.DataFrame(X_train.reshape(320000,3136)).head(5)
```

```
          0         1         2         3         4         5         6         7         8         9        10        11        12
0  0.564706  0.568627  0.505882  0.737255  0.541176  0.533333  0.470588  0.729412  0.521569  0.498039  0.435294  0.709804  0.552941
1  0.588235  0.537255  0.498039  0.635294  0.584314  0.537255  0.494118  0.635294  0.576471  0.533333  0.482353  0.635294  0.568627
2  0.407843  0.427451  0.388235  0.537255  0.392157  0.439216  0.392157  0.521569  0.305882  0.341176  0.341176  0.482353  0.364706
3  0.286275  0.290196  0.258824  0.564706  0.301961  0.313726  0.282353  0.588235  0.290196  0.290196  0.227451  0.549020  0.290196
4  0.282353  0.250980  0.156863  0.513726  0.372549  0.364706  0.337255  0.603922  0.364706  0.380392  0.341176  0.615686  0.364706
5 rows × 3136 columns
```

3.5 Image Classification Architecture

As you'll see, almost all CNN architectures follow the same general design principles of successively applying convolutional layers to the input, periodically down sampling the spatial dimensions while increasing the number of feature maps.

While the classic network architectures were comprised simply of stacked convolutional layers, modern architectures explore new and innovative ways for constructing convolutional layers in a way which allows for more efficient learning. Almost all of these architectures are based on a repeatable unit which is used throughout the network.

These architectures serve as general design guidelines which machine learning practitioners will then adapt to solve various computer vision tasks. These architectures serve as rich feature extractors which can be used for image classification, object detection, image segmentation, and many other more advanced tasks.

Classic network architectures (included for historical purposes)

- LeNet-5
- AlexNet
- VGG 16

Modern network architectures

- Inception
- ResNet
- ResNeXt
- DenseNet

In our project we will be using an architecture that’s constructed based on the AlexNet and VGG 16. The architecture used in our final model is visualized along with the different architecture.

3.6 Data Augmentation

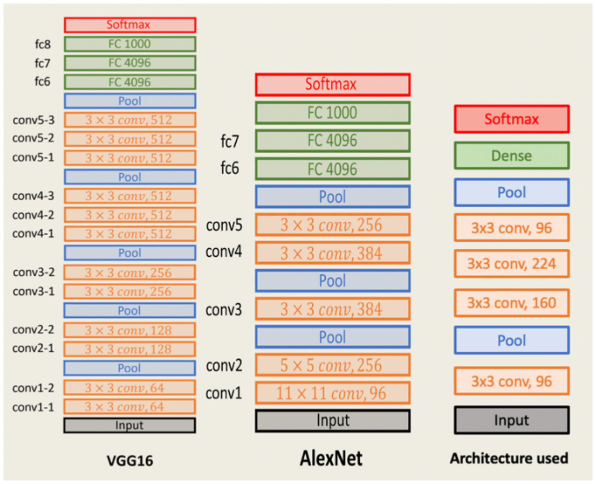
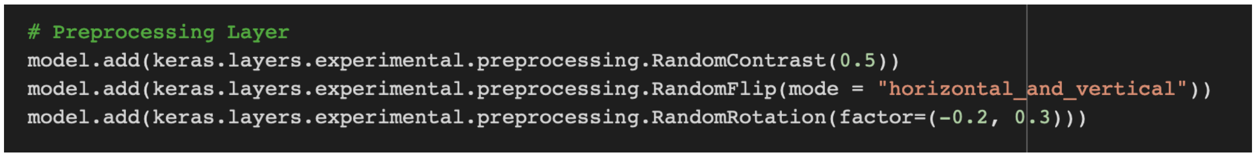


Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images.

As such, it is clear that the choice of the specific data augmentation techniques used for a training dataset must be chosen carefully and within the context of the training dataset and knowledge of the problem domain.

In addition, we performed the data augmentation methods in isolation and in concert to see if they result in a measurable improvement to model performance during the hyperparameter tuning and observed data augmentation resulted in a 5-6% increase in the validation accuracy.

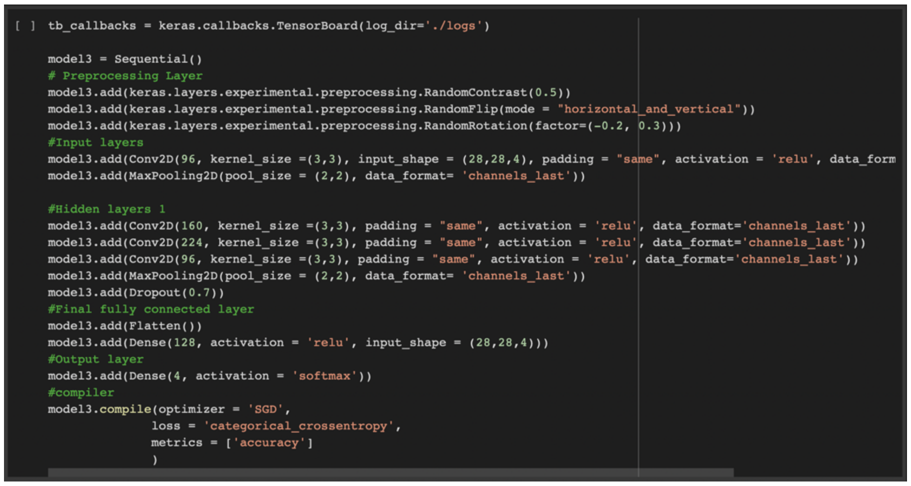
The code used for data augmentation,



```
# Preprocessing Layer
model.add(keras.layers.experimental.preprocessing.RandomContrast(0.5))
model.add(keras.layers.experimental.preprocessing.RandomFlip(mode = "horizontal_and_vertical"))
model.add(keras.layers.experimental.preprocessing.RandomRotation(factor=(-0.2, 0.3)))
```

3.7 Model Explanation

The complete code for our best CNN is shown below,



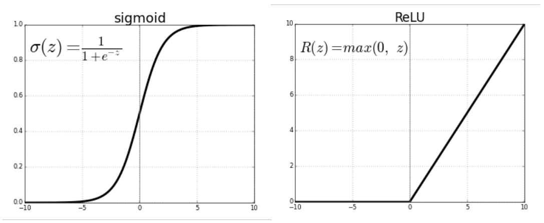
```
[ ] tb_callbacks = keras.callbacks.TensorBoard(log_dir='./logs')

    model3 = Sequential()
    # Preprocessing Layer
    model3.add(keras.layers.experimental.preprocessing.RandomContrast(0.5))
    model3.add(keras.layers.experimental.preprocessing.RandomFlip(mode = "horizontal_and_vertical"))
    model3.add(keras.layers.experimental.preprocessing.RandomRotation(factor=(-0.2, 0.3)))
    #Input layers
    model3.add(Conv2D(96, kernel_size =(3,3), input_shape = (28,28,4), padding = "same", activation = 'relu', data_form
    model3.add(MaxPooling2D(pool_size = (2,2), data_format= 'channels_last'))

    #Hidden layers 1
    model3.add(Conv2D(160, kernel_size =(3,3), padding = "same", activation = 'relu', data_format='channels_last'))
    model3.add(Conv2D(224, kernel_size =(3,3), padding = "same", activation = 'relu', data_format='channels_last'))
    model3.add(Conv2D(96, kernel_size =(3,3), padding = "same", activation = 'relu', data_format='channels_last'))
    model3.add(MaxPooling2D(pool_size = (2,2), data_format= 'channels_last'))
    model3.add(Dropout(0.7))
    #Final fully connected layer
    model3.add(Flatten())
    model3.add(Dense(128, activation = 'relu', input_shape = (28,28,4)))
    #Output layer
    model3.add(Dense(4, activation = 'softmax'))
    #compiler
    model3.compile(optimizer = 'SGD',
                  loss = 'categorical_crossentropy',
                  metrics = ['accuracy']
                  )
```

3.7.1 Input and Hidden Activation layers: ReLU

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input. The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. The rectified linear activation is the default activation when developing multilayer Perceptron and convolutional neural networks.

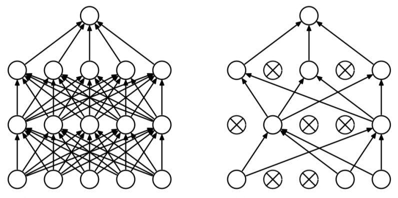
3.7.2 Pooling Layer: Max Pooling



Pooling layers provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map. Two common pooling methods are average pooling and max pooling that summarize the average presence of a feature and the most activated presence of a feature respectively.

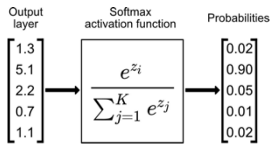
We cannot say that a particular pooling method is better over other generally. The choice of pooling operation is made based on the data at hand. Average pooling method smooths out the image and hence the sharp features may not be identified when this pooling method is used. Max pooling selects the brighter pixels from the image. We have chosen the max pooling because we our data set comprises of the images that's carry a lot of weightage in bright areas.

3.7.3 Penalization: Dropout function (Bayesian Penalty)


(a) Standard Neural Net (b) After applying dropout.

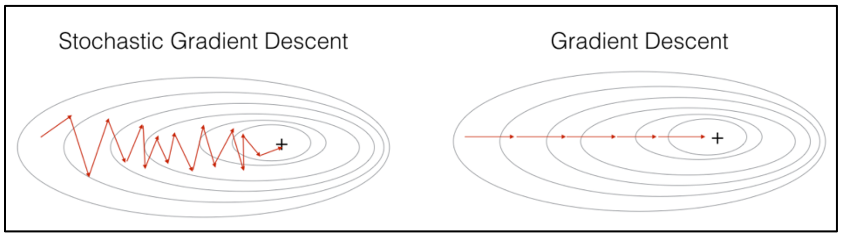
The Dropout function is a penalization function that acts very similar to the Bayesian penalty by decreasing the weightage of layers. The weightage of the layers is decreased by randomly deactivating the certain neurons.

3.7.4 Output Activation layer: Softmax


Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

The Softmax activation layer is used for our output layer because our problem deals with a multiclass classification which suits the Softmax. The Softmax function enforces the sum of the probabilities to be equal to one. So, to increase the probability of one class the function forces the other classes to reduce giving an accurate decision when it comes to multiclass classification.

3.7.5 Optimizer: Stochastic Gradient Descent



We have used SGD because SGD often converges much faster compared to Gradient Descent, but the error function is not as well minimized as in the case of GD. Often in most cases, the close approximation that you get in SGD for the parameter values are enough because they reach the optimal values and keep oscillating there.

The reason for the SGD to converge quicker is because while in GD, you have to run through all the samples in your training set to do a single update for a parameter in a particular iteration, in SGD, on the other hand, you use only one or subset of training sample from your training set to do the update for a parameter in a particular iteration.

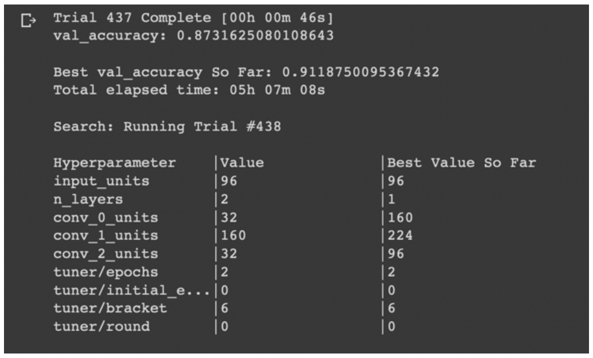
Thus, if the number of training samples are large, in fact very large, then using gradient descent may take too long because in every iteration when you are updating the values of the parameters, you are running through the complete training set. On the other hand, using SGD will be faster because you use only one training sample and it starts improving itself right away from the first sample.

3.7.6 Loss function: Categorical Cross entropy

Categorical cross entropy loss function is used when our model deals with the multi class classification and when our labels are one hot encoded variable.

3.8 Model tuning

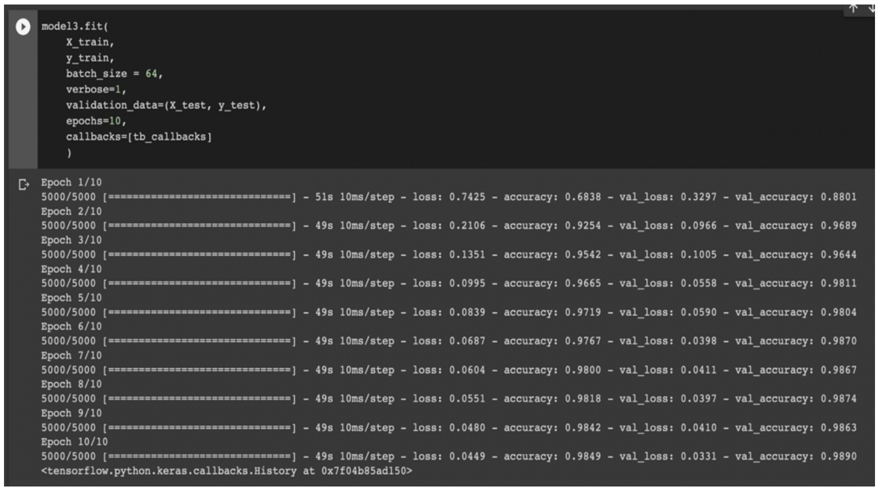
We have tuned our convolutional neural network using the keras tuner function and have employed the Hyperband function. The factors that we have tuned are the number of filters for the input 2d convolution layer and the number of the hidden convolution 2d layers and the number of filters used in each of the convolution hidden layers. We conducted around 400+ trials and got the best parameters for which the validation accuracy is maximum, and the loss function is minimum.



```
Trial 437 Complete [00h 00m 46s]
val_accuracy: 0.8731625080108643

Best val_accuracy So Far: 0.9118750095367432
Total elapsed time: 05h 07m 08s

Search: Running Trial #438

Hyperparameter    |Value              |Best Value So Far
input_units       |96                 |96
n_layers          |2                  |1
conv_0_units      |32                 |160
conv_1_units      |160                |224
conv_2_units      |32                 |96
tuner/epochs      |2                  |2
tuner/initial_e...|0                  |0
tuner/bracket     |6                  |6
tuner/round       |0                  |0
```

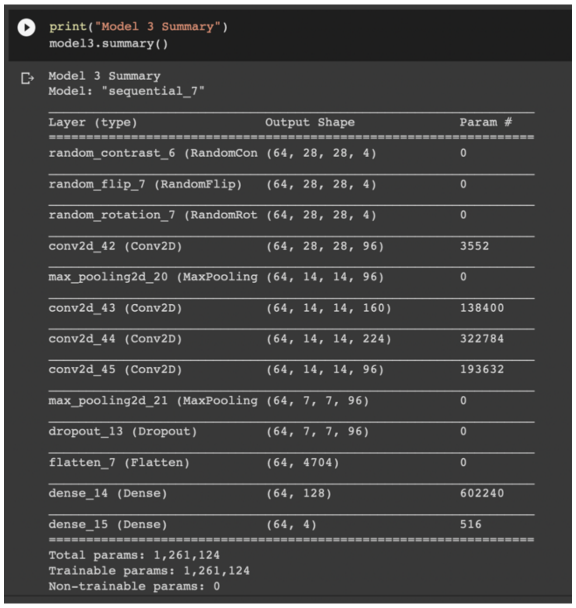
4 Model Summary and Future Improvement Plans

Our final convolutional neural network with the fine-tuned parameters that we collected from the hyperband keras function. We were able to achieve a validation accuracy of 0.9890 and a validation loss function of 0.0331. Here is our model training summary and the complete model summary



```
model3.fit(
    X_train,
    y_train,
    batch_size = 64,
    verbose=1,
    validation_data=(X_test, y_test),
    epochs=10,
    callbacks=[tb_callbacks]
    )
```

```
Epoch 1/10
5000/5000 [==============================] - 51s 10ms/step - loss: 0.7425 - accuracy: 0.6838 - val_loss: 0.3297 - val_accuracy: 0.8801
Epoch 2/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.2106 - accuracy: 0.9254 - val_loss: 0.0966 - val_accuracy: 0.9689
Epoch 3/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.1351 - accuracy: 0.9542 - val_loss: 0.1005 - val_accuracy: 0.9644
Epoch 4/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.0995 - accuracy: 0.9665 - val_loss: 0.0558 - val_accuracy: 0.9811
Epoch 5/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.0839 - accuracy: 0.9719 - val_loss: 0.0590 - val_accuracy: 0.9804
Epoch 6/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.0687 - accuracy: 0.9767 - val_loss: 0.0398 - val_accuracy: 0.9870
Epoch 7/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.0604 - accuracy: 0.9800 - val_loss: 0.0411 - val_accuracy: 0.9867
Epoch 8/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.0551 - accuracy: 0.9818 - val_loss: 0.0397 - val_accuracy: 0.9874
Epoch 9/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.0480 - accuracy: 0.9842 - val_loss: 0.0410 - val_accuracy: 0.9863
Epoch 10/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.0449 - accuracy: 0.9849 - val_loss: 0.0331 - val_accuracy: 0.9890
<tensorflow.python.keras.callbacks.History at 0x7f04b85ad150>
```



```
print("Model 3 Summary")
model3.summary()
```

```
Model 3 Summary
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
random_contrast_6 (RandomCon (64, 28, 28, 4)           0
_________________________________________________________________
random_flip_7 (RandomFlip)   (64, 28, 28, 4)           0
_________________________________________________________________
random_rotation_7 (RandomRot (64, 28, 28, 4)           0
_________________________________________________________________
conv2d_42 (Conv2D)           (64, 28, 28, 96)          3552
_________________________________________________________________
max_pooling2d_20 (MaxPooling (64, 14, 14, 96)          0
_________________________________________________________________
conv2d_43 (Conv2D)           (64, 14, 14, 160)         138400
_________________________________________________________________
conv2d_44 (Conv2D)           (64, 14, 14, 224)         322784
_________________________________________________________________
conv2d_45 (Conv2D)           (64, 14, 14, 96)          193632
_________________________________________________________________
max_pooling2d_21 (MaxPooling (64, 7, 7, 96)            0
_________________________________________________________________
dropout_13 (Dropout)         (64, 7, 7, 96)            0
_________________________________________________________________
flatten_7 (Flatten)          (64, 4704)                0
_________________________________________________________________
dense_14 (Dense)             (64, 128)                 602240
_________________________________________________________________
dense_15 (Dense)             (64, 4)                   516
=================================================================
Total params: 1,261,124
Trainable params: 1,261,124
Non-trainable params: 0
```

4.1 Improvement Plans:

There are many state of the art image processing algorithms that have been developed. First priority would be explore different algorithms and then fine tuning to compare which is the best suited. There is also a way to make our model more robust, it would be by feeding more data and different types of data augmentation layer and further increase the model accuracy.