PROJECT - 1

Implementing a simple CNN architecture on the MNIST Database

Ahsan Khan
104100196

Pooja Chandrasekharan
110091306

Srivatsan Vasudevan
119003177

***Abstract*— The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.**

I. INTRODUCTION

A variety of current, approachable research challenges require the use of intelligent picture processing, which is an intriguing study field in artificial intelligence. The recognition of pre-segmented hand-written digits using learning models is the topic of the well-researched sub-area of the discipline known as hand-written digit recognition. It is one of the most crucial problems in machine learning, data retrieval, deep learning, and pattern recognition, along with several other artificial intelligence disciplines. Additionally, not all of these specific models' components have been thoroughly examined before.

This report shows the implementation of a simple CNN on the MNSIT dataset. We will first create a simple CNN model and apply that model to the MNSIT dataset. We will compare the results and performances with the existing CNN implementation.

II. THE PROBLEM—MOTIVATION

Image processing advanced techniques are generally implemented using deep learning and artificial neural network techniques. Simple machine learning algorithms will be attempted to break this perception and compare the performance of other algorithms. If we have low-sized images, we can run supervised and unsupervised learning algorithms using machine learning without consuming too many resources. In this way, we can use our computers in a very effective way in terms of performance.

Repo - https://github.com/Ahsan-khan/ml-course-first-project/settings

USING MNIST

The MNIST database (Modified National Institute of Standards and Technology database) of handwritten digits consists of a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST. Additionally, the black and white images from NIST were size-normalized and centred to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.
This database is well-liked for training and testing in the fields of machine learning and image processing. It is a remixed subset of the original NIST datasets. One-half of the 60,000 training images consist of images from NIST's testing dataset and the other half from Nist's training set. The 10,000 images from the testing set are similarly assembled.

Figure 1: MNIST Example Dataset

III. PROPOSED METHOD

To build a baseline CNN model with a single convolution layer. Experimenting with the model by adding more convolution layers Fine-tuning the model by changing hyperparameters like dropouts and adding batch normalization layers. Then to compare the current model to the existing architecture, vgg16.

IV. SOFTWARE AND PACKAGES

For this project, we used Python 3.6 as the programming language. Python is open-source software and is a cross-platform language. The following table lists the package and its applications.

PACKAGE	APPLICATION
NumPy	*Data processing/ wrangling, data calculations.*
Keras	*Open-source deep learning*
TensorFlow	*Open-source platform for machine learning.*
Keras.applications.vgg16	*Comparison with existing architecture*
wandb	*To stream training data*
Keras.layers	*To import dense, dropout, flatten, conv2D, and maxpooling2D*
keras.datasets	*To import the MNIST dataset*
keras.models	*To import the sequential model.*

V. READING THE DATA

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML.

For importing the data, we use Keras and then load the training and test images.

You can also find the MNIST dataset on Kaggle: https://www.kaggle.com/c/digit-recognizer/dat


```
# Importing Libraries

import urllib.request
from PIL import Image
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D, BatchNormalization
from tensorflow.keras.applications import vgg16
from tensorflow.keras.datasets import mnist
import wandb
from wandb.keras import WandbCallback
```

```
(train_X, train_y), (test_X, test_y) = mnist.load_data()
plt.imshow(train_X[5])
```

VI. DATA PREPARATION

In order to make sure that Keras is able to read the images and use them effectively, we need to reshape the images into one dimension instead of two by multiplying 28 by 28.

We also need to scale the values in the [0,1] interval by first transforming to float32 and dividing by the maximum value, which is 255.

```
train_X = train_X/255
test_X = test_X/255
```

Reshaped the image so that there is an extra dimension for the input into the neural network.

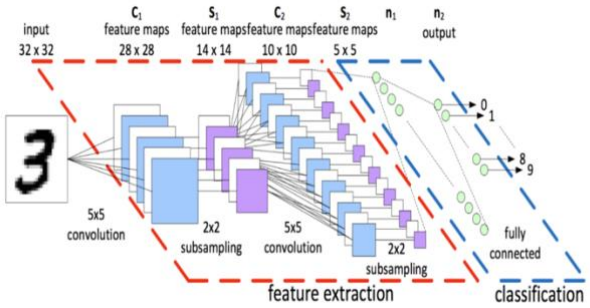
```
train_X = np.expand_dims(train_X, -1)
test_X = np.expand_dims(test_X, -1)
```

VII. SUPERVISED LEARNING TECHNIQUES

Everyone who is new to computer vision realizes that one of the most powerful supervised deep learning techniques is the Convolutional Neural Networks (abbreviated as “CNN”). The final structure of a CNN is actually very similar to Regular Neural Networks (Regular Nets) where there are neurons with weights and biases. In addition, just like in Regular Nets, we use a loss function (e.g. cross-entropy or softmax) and an optimizer (e.g. Adam optimizer) in CNNs.

CONVOLUTIONAL NEURAL NETWORK (CNN)

A typical CNN design begins with feature extraction and finishes with classification. Feature extraction is performed to convolution layers with sub-sampling. Classification is performed with dense layers followed by a final SoftMax layer. For image classification, this architecture performs better than an entirely fully connected feed-forward neural network.



THE 5-STEP MODEL LIFE-CYCLE

A model has a life-cycle, and this very simple knowledge provides the backbone for both modelling a dataset and understanding the tf.keras API.

There are 5 steps for this model:

1. Define the model.

Next, we need to define a baseline convolutional neural network model for the problem.

For the convolutional front-end, we can start with a single convolutional layer with a small filter size (3,3) and a modest number of filters (32) followed by a max pooling layer. The filter maps can then be flattened to provide features to the classifier.

```
#Simple CNN Model
wandb.init(
    project = 'Topics in AI Report 1',
    name = 'Plain CNN Model'
)
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.1))
model.add(Dense(10,activation=tf.nn.softmax))
```

2. Compile the model.

This is a multi-class classification problem, so categorical_crossentropy is the loss function. For optimizing our weights, we used Adam as the optimizer, and the loss method we are using is spare categorical cross entropy.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

3. Fit the model.

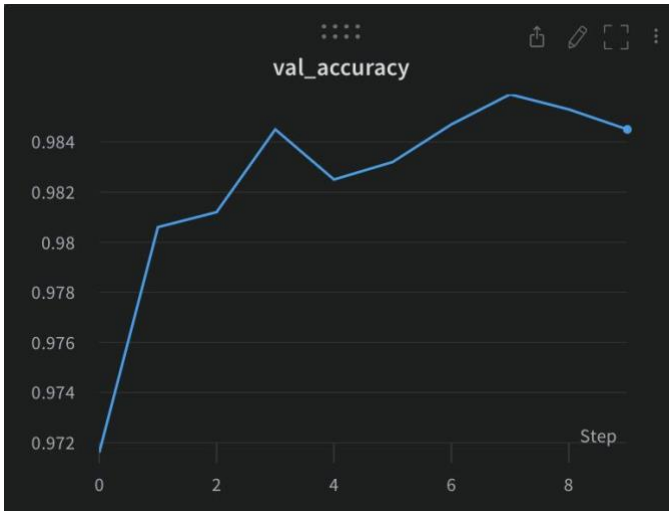
Each image will be trained over 10 epochs. The validation rate is 0.16.

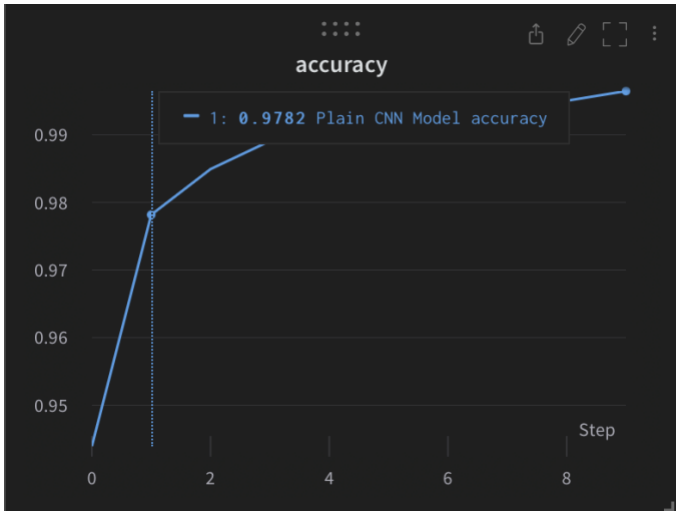
```
model.fit(x=train_X,y=train_y, validation_data=(test_X, test_y),epochs=10, callbacks=[WandbCallback()])
```

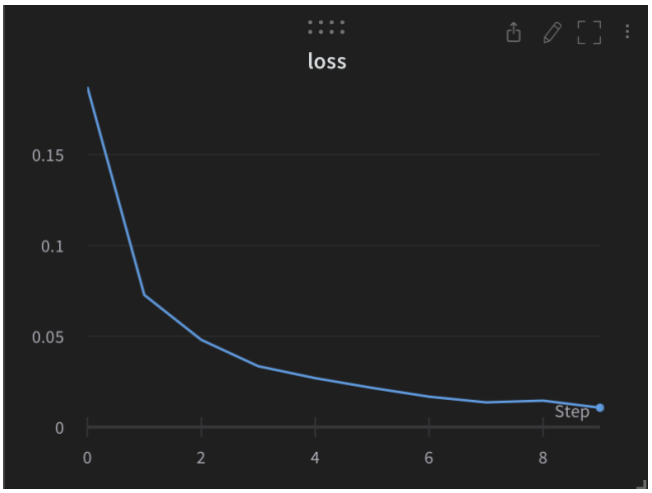
4. Evaluate the model.

In order to evaluate our neural network's performance, we used accuracy as an evaluation metric.

The following graphs are for the base model:







5. Make predictions.

We got around **98.65%** accuracy.

Train Loss: 0.01
Train Accuracy: 0.99

Test Loss: 0.069
Test Accuracy: 0.98

After the base model, we decided to do 3 more models in order to increase accuracy and compare with other CNN architectures.

1. Adding more layers to make it a deeper CNN

```
#Experiment 1 Deeper CNN
wandb.init(
    project = 'Topics in AI Report 1',
    name = 'Deeper CNN Model'
)
model2 = Sequential()
model2.add(Conv2D(28, kernel_size=(3,3), input_shape=(28,28,1)))
model2.add(Conv2D(64, kernel_size=(3,3)))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Conv2D(128, kernel_size=(3,3)))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model2.add(Dense(128, activation=tf.nn.relu))
model2.add(Dropout(0.1))
model2.add(Dense(10,activation=tf.nn.softmax))
model2.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
model2.fit(x=train_X,y=train_y, validation_data=(test_X, test_y),epochs=10, callbacks=[WandbCallback()])
```

2. We increased the number of dropout layers.

```
#Experiment 2 Increased Dropoout
wandb.init(
    project = 'Topics in AI Report 1',
    name = 'Deeper CNN Model with higher dropouts'
)
model3 = Sequential()
model3.add(Conv2D(28, kernel_size=(3,3), input_shape=(28,28,1)))
model3.add(Conv2D(64, kernel_size=(3,3)))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Conv2D(128, kernel_size=(3,3)))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model3.add(Dense(128, activation=tf.nn.relu))
model3.add(Dropout(0.6))
model3.add(Dense(10,activation=tf.nn.softmax))
model3.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
model3.fit(x=train_X,y=train_y, validation_data=(test_X, test_y),epochs=10, callbacks=[WandbCallback()])
```

3. Adding more batch normalization layers.

```
#Experiment 3 Adding Batch Normlization layers
wandb.init(
    project = 'Topics in AI Report 1',
    name = 'Deeper CNN Model with Batch Normalization'
)
model4 = Sequential()
model4.add(Conv2D(28, kernel_size=(3,3), input_shape=(28,28,1)))
model.add(BatchNormalization())
model4.add(Conv2D(64, kernel_size=(3,3)))
model.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model4.add(Conv2D(128, kernel_size=(3,3)))
model.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model4.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model4.add(Dense(128, activation=tf.nn.relu))
model.add(BatchNormalization())
model4.add(Dropout(0.6))
model4.add(Dense(10,activation=tf.nn.softmax))
model4.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
model4.fit(x=train_X,y=train_y, validation_data=(test_X, test_y),epochs=10, callbacks=[WandbCallback()])
```

4. Comparing with existing CNN architecture and the architecture we selected is MobileNet.

```
import tensorflow as tf
wandb.init(
    project = 'Topics in AI Report 1',
    name = 'Mobile Net for comparision'
)
mnet_model = tf.keras.applications.mobilenet.MobileNet(include_top=False, input_tensor=Input(shape=(128, 128, 3)))
mnet_model.summary()
(trainVGG_X, trainVGG_y), (testVGG_X, testVGG_y) = mnist.load_data()
print(trainVGG_X.shape, testVGG_X.shape)
trainVGG_X.resize((60000, 128, 128, 3))
testVGG_X.resize(10000, 128, 128, 3)
mnet_model.summary()
pretrained_model = Sequential()
for layer in mnet_model.layers:
    pretrained_model.add(layer)
pretrained_model.summary()
for layer in mnet_model.layers:
    layer.trainable = False
pretrained_model.add(Flatten())
pretrained_model.add(Dense(10,activation=tf.nn.softmax))
pretrained_model.summary()
type(pretrained_model)
pretrained_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
pretrained_model.fit(x=trainVGG_X,y=trainVGG_y, validation_data=(testVGG_X, testVGG_y),epochs=10, callbacks=[WandbCa
```

VIII. RESULT

We trained each model:

BASE: The plain model has the best accuracy of 0.983
Exp 1: Deeper CNN model adds a best accuracy of 0.987

Exp 2: Deeper CNN model with higher dropouts has 0.986

Exp 3: Deeper CNN model with batch normalization has an accuracy of 0.989

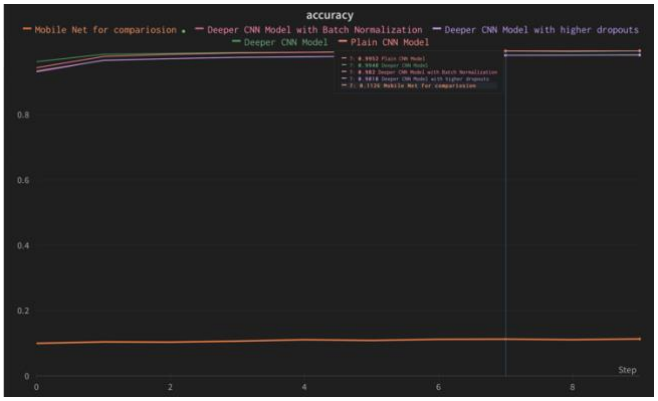
Exp 4: Comparing with MobileNet: have an accuracy of 0.10

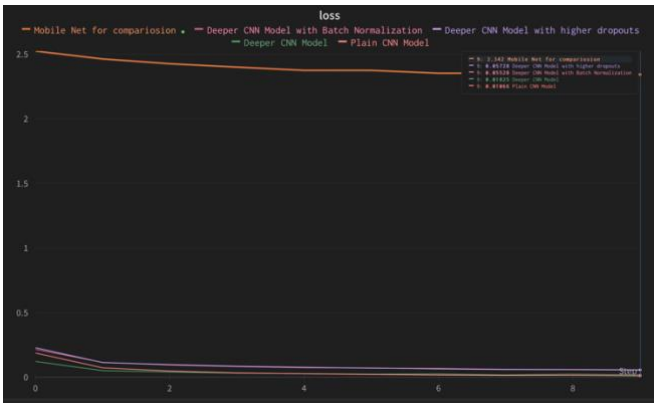
From the base model, we build a deeper model to improve the performance of the base model and we found there was a slight improvement of 0.03. We then increased the drop-out layers to prevent overfitting and we found there was a slight decrease in the accuracy. We also added batch norm layers to ensure faster training and we found that it increased our accuracy by 0.06 and we also found that the run time decreased by 3 seconds.

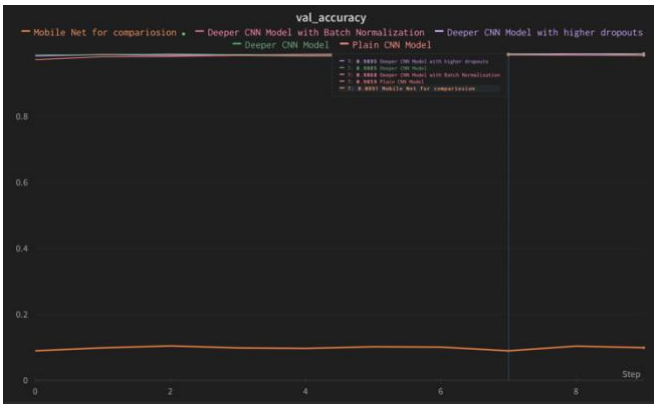
Finally, we compared our model with an existing architecture MobileNet and found out that MobileNet performed extremely poor. From this, we can see using

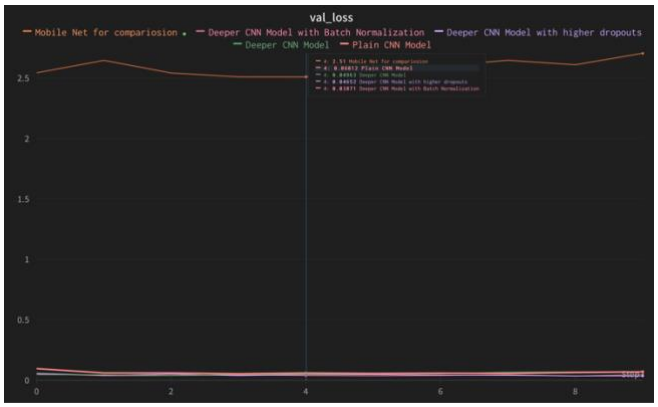
existing pre trained weights did not improve our model's performance. This was because the images on which MobileNet was trained on as a completely different characteristic to MNIST dataset.

The following graphs shows the Training loss and accuracy, Test/Val loss and accuracy:









IX. CONCLUSION

To conclude, from Exp 4, we learnt that transfer learning works best with images of similar characteristic because the pre-trained weights that is used by MobileNet are trained on images with three channels whereas as MNIST dataset has images of one channel.

We also found out that a deeper CNN model perform better than base model and batch normalization helped in reducing training time.

Findings:
https://wandb.ai/srivatsan25/Topics%20in%20AI%20Report%201?workspace=user-srivatsan25

REFERENCES

[1] M. Z. Alom, P. Sidike, T. M. Taha, and V. K. Asari, "Handwritten Bangla Digit Recognition Using Deep Learning," vol. 6, no. 7, pp. 990–997, 2017.

[2] S. Majumder, "Handwritten Digit Recognition by Elastic Matching," J. Comput., vol. 4, no. 04, pp. 1067–1074, 2018.

[3] G. Cheedella, "Critique of Various Algorithms for Handwritten Digit Recognition Using Azure ML Studio," Glob. J. Comput. Sci. Technol., vol. 20, no. 1, pp. 1–5, 2020.

[4] S. M. Shamim, M. B. A. Miah, A. Sarker, M. Rana, and A. Al Jobair, "Handwritten digit recognition using machine learning algorithms," Indones. J. Sci. Technol., vol. 3, no. 1, pp. 29–39, 2018.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Ha, "LeNet," Proc. IEEE, no. November, pp. 1–46, 1998.