University of Missouri-Kansas City

School of Computing and Engineering

ICP: ICP-04

Course Name: Big Data Analytics and Applications
Course ID: COMP-SCI 5542

Semester /Session : Spring 2022

Student ID : **Student Name**

16334245 Mustavi Islam

16321217 Keenan Flynn

Description of the Problem :
ICP 4 focused on the tuning of neural network hyperparameters. Hyperparamaters include the toggleable features of each layer in the network, number of neurons, activation function, optimizer, batch size, epochs and others. Hyperparameter tuning can cause subtle differences in the accuracy of your model and those differences can be both positive and negative. Tuning can change the learning rate, variance, computational cost, and metrics so it is important to understand why this happens.

Description of Solution :

1. The model provided in the source code for the ICP was adapted for the CIFAR-10 dataset.
    a. The data was already cleaned and split into training and testing sets.
    b. The labels were transformed using one hot encoding.
    c. The same Keras sequential model from the source code was then used, but with num_classes = 10.
2. Tuning the convolutional layers.
    a. The number of convolutional filters in each Conv2D() layer was increased. Because our images have relatively few dimensions, it is very important that we extract all the features we can. We found through experimentation that we can increase the number of filters for the cost of longer computational time. These filters increase the amount of data that we have about an image and thus increase the accuracy of the model.
    b. We are currently using 4x the number of filters than what was in the source code. We found this helped bump the model to 80% accuracy
3. Tuning Epochs and Batch size
    a. Epochs and Batch size were found to be correlated in the performance of the model.
    b. Initially, a low batch size and low epoch size were used. A low batch size increases learning rate, but also the variance and thus loss of the model. To make sure this loss was not too great, a low number of epochs were used.
    c. After experimentation, a higher batch size and higher number of epochs were used. Higher batch size decreases the learning rate because the learning of the model occurs after each batch. Increasing batch size decreases the chance for outliers to have a big effect and thus decreases variance and loss. To account for the slow learning rate, a high number of epochs needs to be used.
    d. We are currently using a batch size of 200 and 35 epochs. We tested all the way up to 50 epochs but found this gave marginal improvement at the cost of marginal deterioration of loss score.

4. Tuning activation functions
    a. The activation function provides non-linearity to the model which separates a neural network problem from a regression problem.
    b. The hidden layers use ‘Relu’ as the activation function. Relu is a good activation function for the hidden layers because it acts like a switch. When a feature is not wanted or needed, Relu ‘turns it off’. When a feature is wanted, Relu is simply the identity function which saves computational cost. For the hidden Dense layer, the activation was changed from softmax to Relu
    c. The output layer is where the model chooses which label to give to a piece of data. This layer was changed to have an activation of ‘softmax’. Softmax outputs a vector of decimals that sum to equal 1. This creates a set of probabilities that the model can choose from. Softmax is a good activation function for multiclass classification problems because of this property.
5. The final part of this ICP was to test the model with random images.
    a. These images were scraped from stock photo repos on the web. We created a loadImage() function that received a URL as input and output a resized 32x32 image in the form of a numpy array. This function also plots the image so you can visualize the transformation.
6. Results
    a. The model classified the validation data at around 80%. This percentage was reflected by how it classified our random images. It classified 4 out of 5 pictures on a typical run. The random images used were particularly unique images and it is impressive that the model captured these classifications. The images we used are pictured below.

Model correctly classified as deer

Model correctly classified as cat

Original

Compressed to 32x32x3

Model incorrectly classified as truck

Original

Compressed to 32x32x3

Model correctly classified as ship

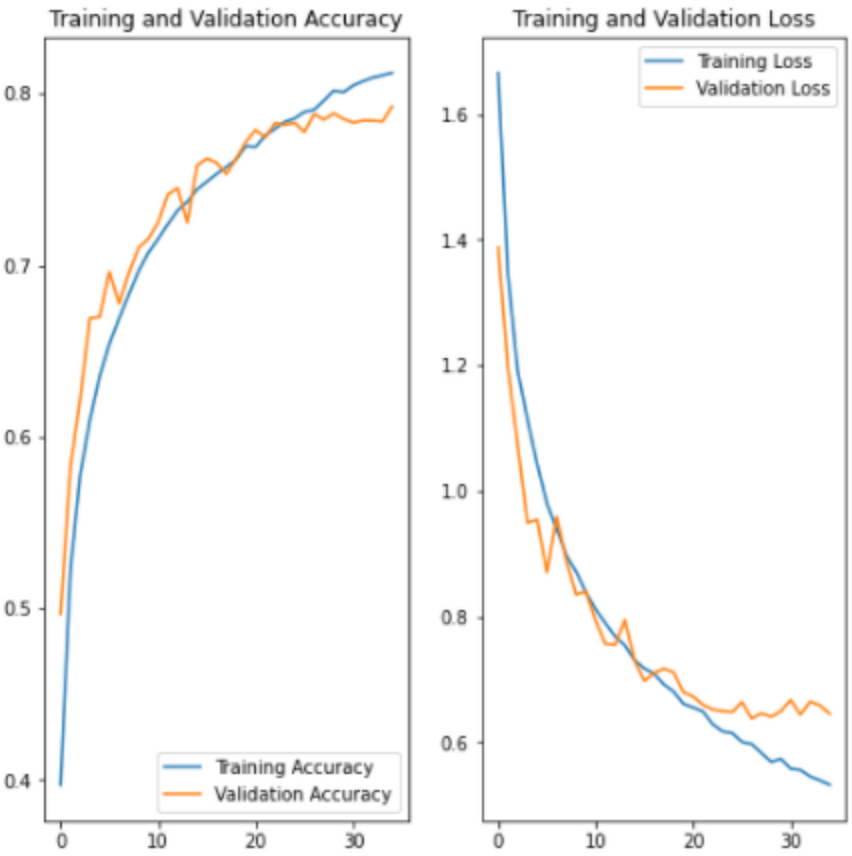
Original

Compressed to 32x32x3

Model correctly classified as frog

Challenges:
The biggest challenge this ICP faced was the tuning of hyperparameters. In industry, a grid search can be used to automate this process, but it was informative for us to manually change each hyperparameter and see the effect it had on the model's performance. We tried optimizing many parameters such as filter size, optimizer, and loss function.

We also learned how changing 1 hyperparameter can cause the model to throw errors. One example of this was when we changed the output layer to have an activation function of ‘softmax’. Our model started to give a warning, which we had to fix by modifying the loss function. The Categorical cross entropy function had a parameter of ‘from_logits=True’. This was removed because the softmax function does not output logits.



The model classified the validation set with ~80% accuracy. While we could have improved the accuracy further by adding new layers to the model, we decided to solely focus on the tuning of hyperparameters.

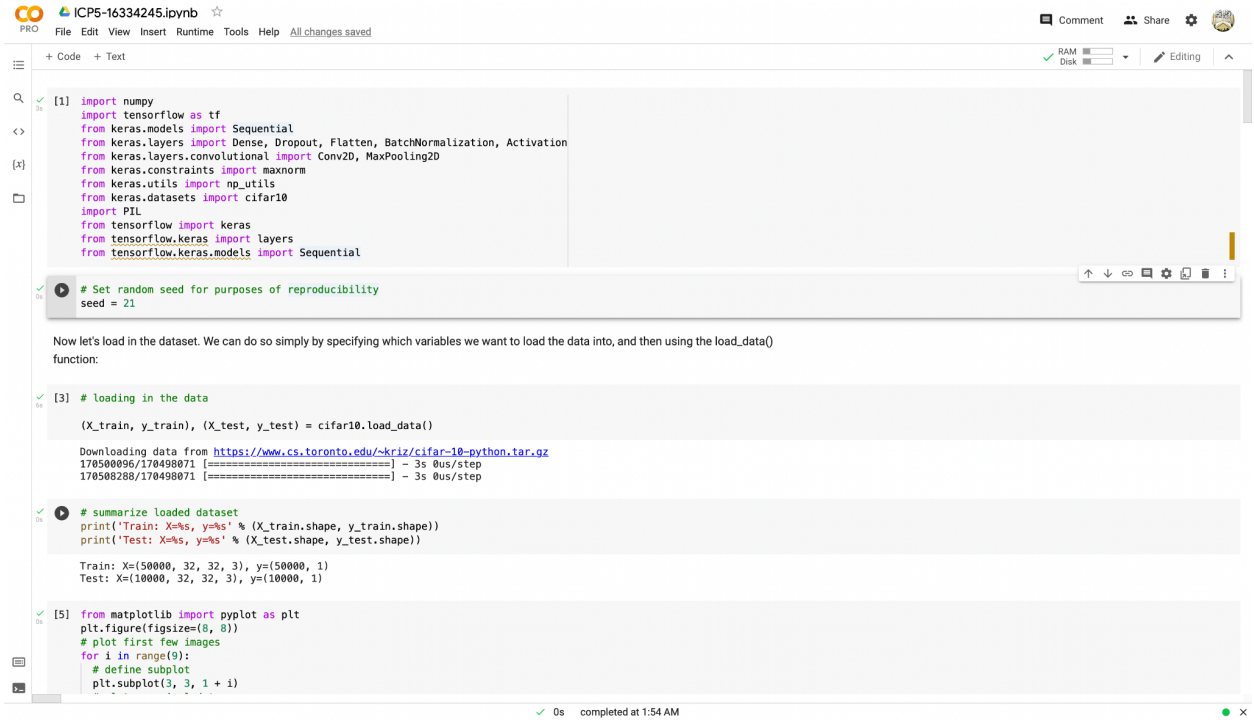
Learning Outcomes :

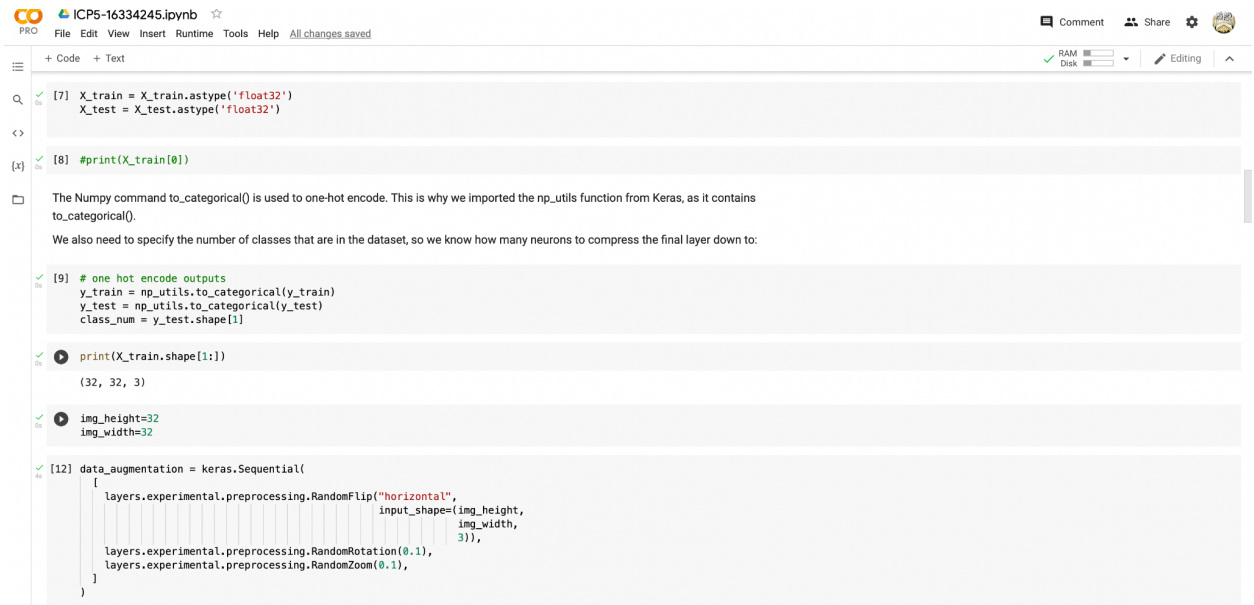
1. We learned how activation functions work.
2. We learned what a grid search was.
3. We learned how to tune a model without adding layers.
4. We learned how to modify hyperparameters to get the best performance.
5. We learned how loss was affected by batch and epoch size.

Resources:
https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/ **-> We used this site for references on how to validate a model with independent images.**

Video Link: https://youtu.be/dci2wmBiqBs

Screenshots :





colab.research.google.com

ICP5-16334245.ipynb - Colaboratory | ICP4 - Google Docs

ICP5-16334245.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
[13] num_classes = 10

     model = Sequential([
       data_augmentation,
       layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
       layers.Conv2D(64, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       #layers.Dropout(0.2),         #I think we should keep the baseline layers that were provided in the original source code
       #layers.BatchNormalization(),
       layers.Conv2D(128, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       #layers.Dropout(0.2),
       #layers.BatchNormalization(),
       layers.Conv2D(256, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Dropout(0.2),
       #layers.BatchNormalization(),
       layers.Flatten(),
       layers.Dense(128, activation='relu'),
       layers.Dense(num_classes, activation='softmax')
     ])
```

```
[14] model.compile(optimizer='adam',
                   loss=tf.keras.losses.CategoricalCrossentropy(),
                   metrics=['accuracy'])
```

```
[15] model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 32, 32, 3)         0

rescaling (Rescaling)        (None, 32, 32, 3)         0

conv2d (Conv2D)              (None, 32, 32, 64)        1792

max_pooling2d (MaxPooling2D  (None, 16, 16, 64)        0
)

conv2d_1 (Conv2D)            (None, 16, 16, 128)       73856

max_pooling2d_1 (MaxPooling  (None, 8, 8, 128)         0
2D)
```

0s completed at 1:54 AM

+ Code + Text

Editing

```
[16] epochs=35
     numpy.random.seed(seed)
     history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=200)
```

```
250/250 [==============================] - 4s 18ms/step - loss: 0.9850 - accuracy: 0.6528 - val_loss: 0.9766 - val_accuracy: 0.6665
Epoch 7/35
```

ICP5-16334245.ipynb

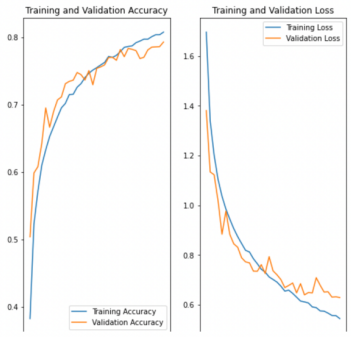
File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

Editing

```
[18] val_acc = history.history['val_accuracy']

     loss=history.history['loss']
     val_loss=history.history['val_loss']

     epochs_range = range(epochs)

     plt.figure(figsize=(8, 8))
     plt.subplot(1, 2, 1)
     plt.plot(epochs_range, acc, label='Training Accuracy')
     plt.plot(epochs_range, val_acc, label='Validation Accuracy')
     plt.legend(loc='lower right')
     plt.title('Training and Validation Accuracy')

     plt.subplot(1, 2, 2)
     plt.plot(epochs_range, loss, label='Training Loss')
     plt.plot(epochs_range, val_loss, label='Validation Loss')
     plt.legend(loc='upper right')
     plt.title('Training and Validation Loss')
     plt.show()
```



0s completed at 1:54 AM

```
0   10   20   30   0   10   20   30
```

```
from keras.backend import clear_session
clear_session()
```

```
[20] from keras.preprocessing.image import load_img
     from keras.preprocessing.image import img_to_array
     from keras.models import load_model
     from io import BytesIO
     from urllib.request import urlopen
     from PIL import Image
     import numpy as np
```

```
def loadImage(URL):
    res = urlopen(URL).read()
    img = Image.open(BytesIO(res))
    fig = plt.figure(figsize=(18, 16))
    ax1 = fig.add_subplot(1, 2, 1)
    ax1.imshow(img)
    plt.axis('off')
    plt.title('Original')
    img = img.resize((32,32))
    ax2 = fig.add_subplot(1, 2, 2)
    ax2.imshow(img)
    plt.axis('off')
    plt.title('Compressed to 32x32x3')
    fig.show()
    #convert to array
    mod = img_to_array(img)
    # reshape into a single sample with 3 channels
    mod = mod.reshape(1, 32, 32, 3)
    # prepare pixel data
    mod = mod.astype('float32')
    return mod
```

```
[22] def predict_class(img):
       cifar10_classes=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
       prediction = model.predict(img)
       classes=np.argmax(prediction,axis=1)
       print(cifar10_classes[classes[0]])
```

```
    return mod
```

Run cell (⌘/Ctrl+Enter)
cell executed since last change

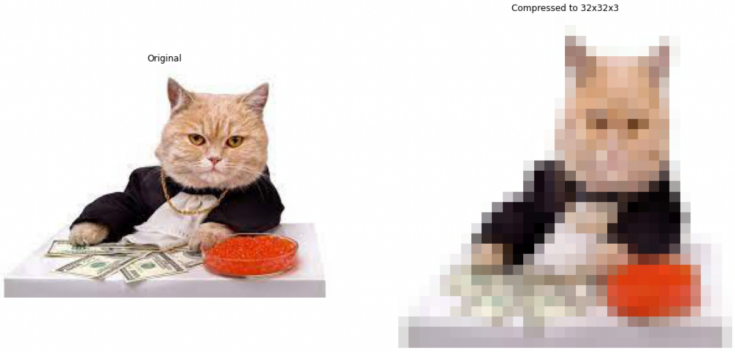
executed by Mustavi Islam
1:54 AM (4 minutes ago)
executed in 0.007s

```
[ (img):
    =['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
    del.predict(img)
    ax(prediction,axis=1)
    print(cifar10_classes[classes[0]])
```

```
[23] deer = loadImage('https://media.istockphoto.com/photos/fallow-deer-picture-id1145651250?k=20&m=1145651250&s=612x612&w=0&h=F8zwduw-t7WCkX4HGTTfe6nTomjoL7S2nrqrUs5jzvU=')
```

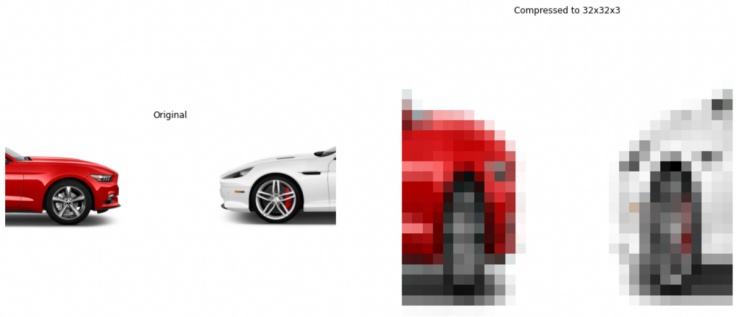

```
[24] predict_class(deer)

     deer
```

```
cat=loadImage('
```



Lets give it a hard image. The model was probably able to tell that the image had wheels and a hood, but it failed to predict 'automobile' over 'truck'

```
[27] car=loadImage('https://www.izmostock.com/wp-content/uploads/2018/04/izmostock_MainBanner_04.jpg')
```




```
predict_class(car)
```

```
truck
```

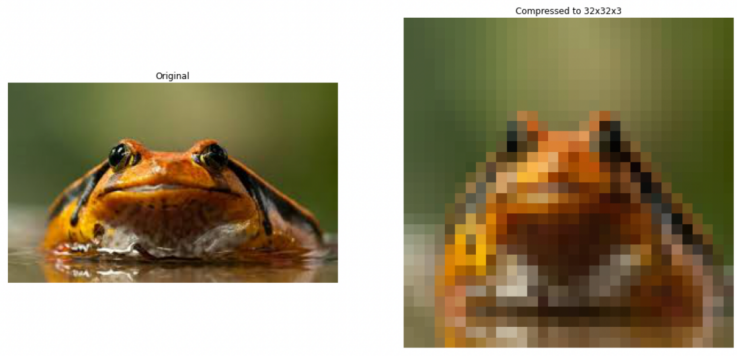
```
[29] ship = loadImage('https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSOD3YIEUhzgWXVMbpLZLFjOVqg6QRH6nzGGw&usqp=CAU')
```


```
[30] predict_class(ship)
```

```
cat
```

+ Code + Text

```
[31] frog = loadImage('https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQMG4yexr2eOfzLDq9C-VZGPU8BL7B0wE2l9g&usqp=CAU')
```




```
[32] predict_class(frog)
```

```
frog
```