Assignment 2 Team Report

Abstract

Objective: Image classification problem

Datasets used: Tiny-ImageNet 200, CIFAR 10

Tasks Performed:

Task 1: Building a CNN model using Keras framework.

Dataset: Tiny ImageNet

Task2: Implementation of Autokeras to tune hyperparameters

Dataset: Tiny Imagenet

Task3: Implementation of Alexnet model in Keras

Dataset: Tiny Imagenet

Task4: Implementation of pre-trained model

Dataset: Tiny Imagenet

Pre-trained model: Xception Architecture

Task5: Transfer learning using Xception Architecture

Dataset: CIFAR10

Task6: Fine tuning for model with Xception Architecture

Dataset: CIFAR10

Task 1: Building a CNN Model in Keras

Steps:

1. Resized the Tiny-ImageNet data which was in 64*64 dimensions using bilinear algorithm to 32*32 dimensions.

Bilinear algorithm: Interpolation technique based on surrounding pixels which gives smooth scaling. It is a combination of two linear interpolations where arbitrary points are estimated between two other points and image shrinks as per the required dimension.

   a. Link to code for resizing images

2. Processed and mapped the images with corresponding labels.
3. Tried different models by tuning the parameters and choose the best model.

   Hyperparameters settings:

   Number of layers: 18

   Optimizer: Adam

   Learning rate: .00001

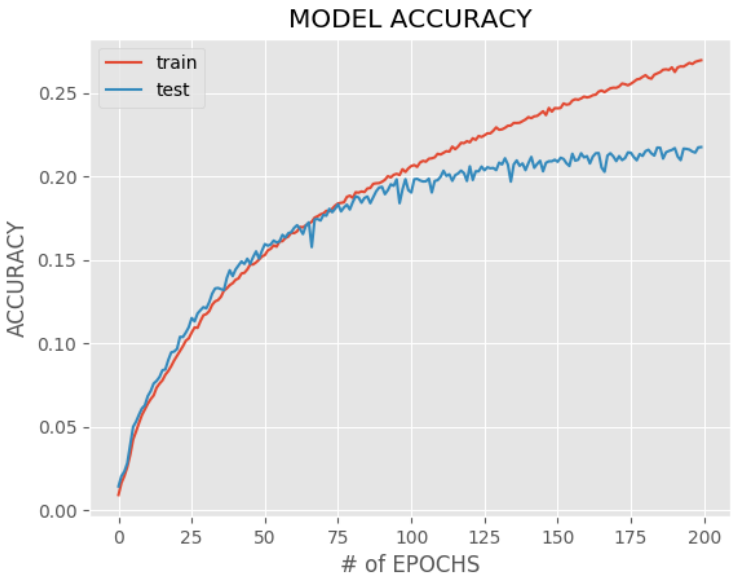
   Batch Size: 64

   Number of epochs: 200

Observations & Inference:

- Model seems to train better without any overfitting issues after tuning of hyperparameters and running different types of models.
- Model Accuracy seems to increase with decrease in model loss as expected.
- With higher end machines, we could solve this problem at much ease as the data is huge along with model being complex.

Model Performance:



Training time: 5 hours in a CPU

Link to experiment

Link to recordings made during optimization

Task 2: Perform hyperparameters tuning with Autokeras

What is AutoKeras?

AutoKeras is an open source software library which aims to provide access to deep learning tools without domain knowledge in the fields of machine learning and data science.

It is used to automate the process of building the best model architecture and hyperparameters given the data input.

Why AutoKeras is preferred?

Autokeras uses network morphism which reduces the time taken for neural architecture search. It uses a Gaussian Process for Bayesian optimization of guiding network morphism.

Neural architecture search (NAS) has been proposed to automatically tune deep neural networks, but existing search algorithms usually suffer from expensive computational cost. Using network morphism, which keeps the functionality of a neural network while changing its neural architecture, could be helpful for NAS by enabling a more efficient training during the search.

Procedure followed:
By running models with different time frames, we obtained the best hyperparameters for the model along with time taken.

Note: By default, Autokeras takes 24 hours to build model and tune hyperparameters.

Observations:

1. We observed that giving time frame as 1 hour, it was not able to find any optimal model. This explains the complexity of our dataset.
2. Increasing the time frame slowly, we could find better results where model metrics seem to improve.
3. We choose our final model to be the one obtained from running it for 10 hours. We found this as the most optimal one with reasonable timeframe as well.

Inference:

1. Using Autokeras could help one in tuning hyperparameters at much faster rate when compared to the usual trial and error method.
2. The difference of model performance could be observed while comparing the best models obtained from task 1 and task 2. Autokeras could clearly give a better optimized model.
3. One could save a lot of time from trial and error when Autokeras is applied.
4. Obtained best model could be easily saved and the model architecture can be visualized at ease.

Link to Code:

https://github.com/Shravsridhar/INFO-7374-Assignment-2/blob/master/AutoKeras.ipynb

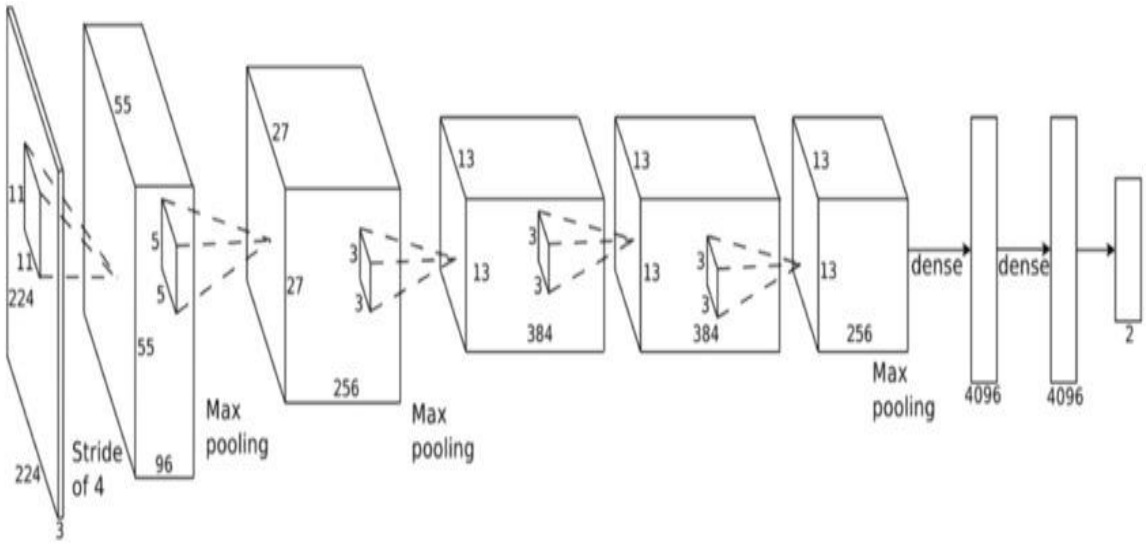
Link to recorded observations:

https://docs.google.com/spreadsheets/d/1hl7vIWUg05e_pCSIjPQ3AnZwxRS-TTpvdjYA8EIwp0Y/edit#gid=1236221929

Task 3: Implementation of Alexnet model in Keras

About Alexnet:

The AlexNet architecture consists of five convolutional layers, some of which are followed by maximum pooling layers and then three fully-connected layers and finally a 1000-way softmax classifier. Originally, all the layers were divided into 2 and trained in separate GPU's to speed process. Below diagram depicts the architecture layout with a single layer which we followed for our implementation with tinyimagenet data.



Performed Steps:

Using keras, we replicated the Alexnet architecture. We modified the final fully connected layer according to our tinyimagent data which has 200 classes.

Best model was found post tuning the hyperparameters.

Trial with resizing image to 224*224 and running the AlexNet model.

Link to Code

Link to recordings of different models

Observations:

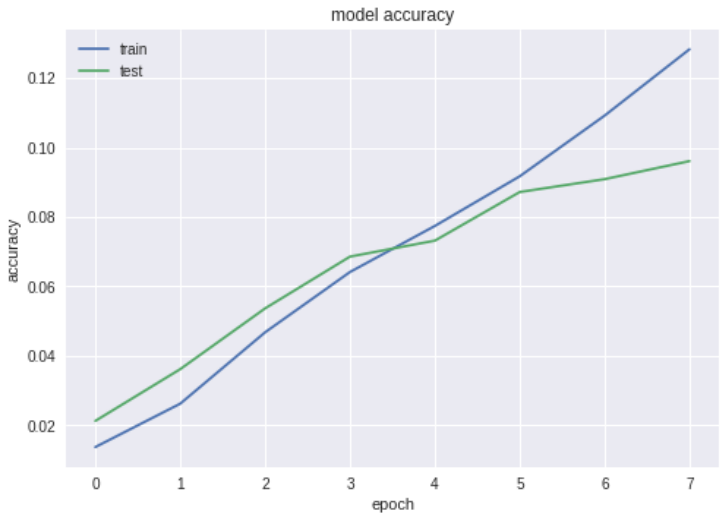
- Tiny ImageNet seems to perform well with AlexNet architecture.
- Resizing the data to 224*224 didn't seem to provide feasible results as 1 epoch was taking around 8 hours and still performance was same as one with 32*32 size. This shows that size doesn't matter in implementing the architecture.

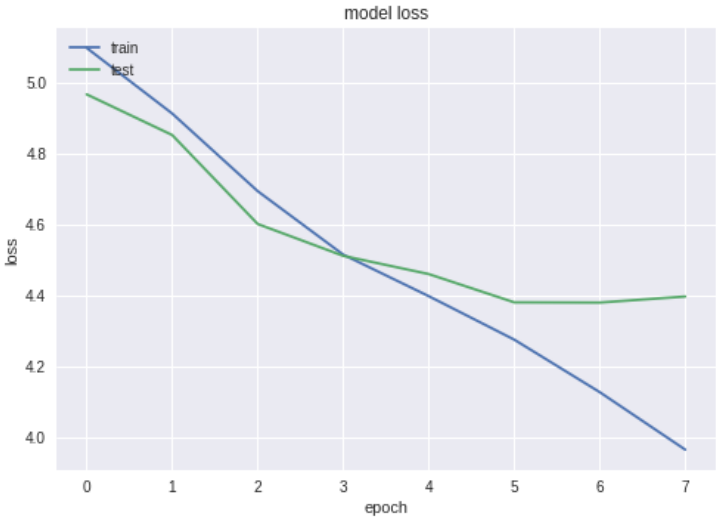
```
In [*]: fit_m =modelAlex.fit(x_train,y_train, batch_size=32, epochs=20,validation_data=(x_test,y_test),shuffle=True)
        scores = modelAlex.evaluate(X_test, y_test, verbose=0)
        print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
        print("Loss:" , scores[0])

        Train on 100000 samples, validate on 10000 samples
        Epoch 1/20
          1728/100000 [..............................] - ETA: 28871s - loss: 5.3619 - acc: 0.0052
```

- The layers were replicated from AlexNet architecture, and it was observed that model performance seems to improve post tuning of hyperparameters.

Model Performance:




model loss
train
test
loss
epoch
5.0
4.8
4.6
4.4
4.2
4.0
0
1
2
3
4
5
6
7

Task 4: Implementation of Xception model on tiny ImageNet data

What is Inception module?

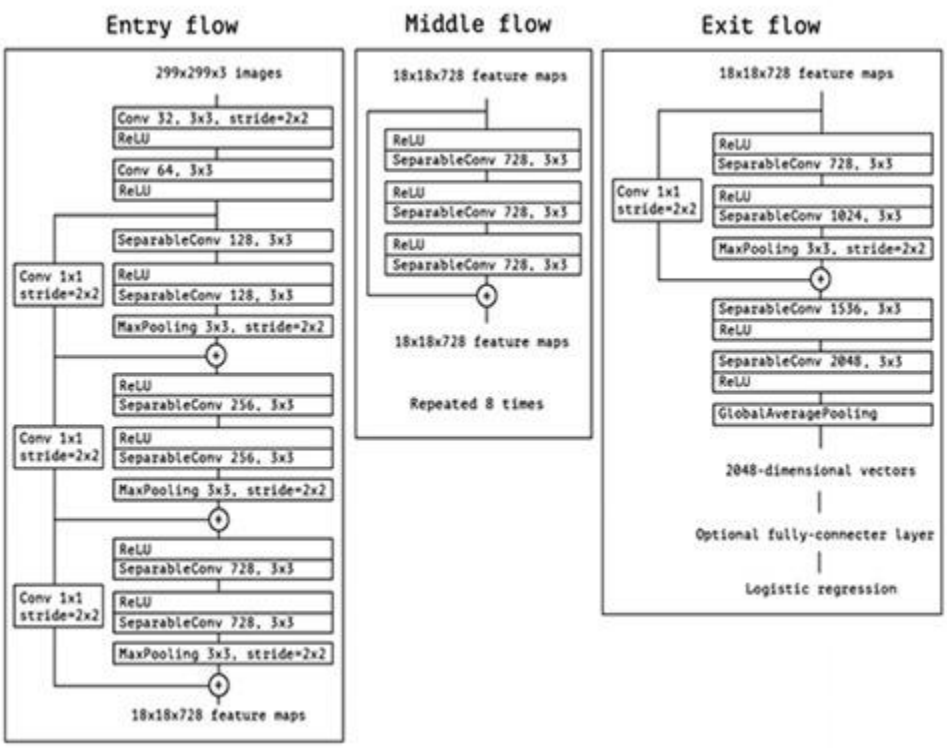
The key idea for devising this model is to deploy multiple convolutions with multiple filters and pooling layers simultaneously in parallel within the same layer.

About Xception architecture:

Xception architecture is inspired by Inception Architecture and it's also known as extreme inception.

The Inception modules are replaced with depthwise separable convolutions. Xception architecture entirely consists of depthwise separable convolutional layers.

Xception has the same number of parameters as Inception V3. It consists of 36 convolutional layers stacked in 14 modules. All of which have linear residual connections except 1st and the last module. Below is the architecture visualized:



Steps performed:

1. Xception architecture requires the input to be of minimum 70*70 dimensions. Hence, we reshaped the ImageNet input to the required dimensions.
2. Using Xception as a pre-trained model for tiny ImageNet data, we performed transfer learning.
3. Found the best optimal model post tuning of hyperparameters and customizing layers as well.
4. Layers were added to the pretrained model in order to achieve best results.

Observations:

1. Using pretrained models saved time as there were pretrained weights that could be used with our dataset.
2. It could be used as one way to start working with data. It helps get a clearer picture about the data rather than working from scratch.
3. It was noted that there was not a notable improvement in model performance when tuning the parameters and finding the optimal solution.
4. Preferably hyper-tuning parameters more and optimizing could bring in better results.
5. Accuracy could be improved for the pretrained model by hyper tuning the parameters as done in task 6 for cifar10 dataset.
6. Due to the size of data, it takes more time to train.

Using bottleneck features extraction: 'Bottleneck' is an informal term which is often used for the layer just before the final output layer that does the classification. In this case, it is the modified SoftMax layer with 200 neurons.

Model performance:

```
']:  fc_model2 = xception_from_disk()

Train on 100000 samples, validate on 10000 samples
Epoch 1/10
100000/100000 [==============================] - 20s - loss: 5.2989 - acc: 0.0044 - val_loss: 5.2983 - val_acc: 0.0056
Epoch 2/10
100000/100000 [==============================] - 20s - loss: 5.2989 - acc: 0.0043 - val_loss: 5.2983 - val_acc: 0.0052
Epoch 3/10
100000/100000 [==============================] - 20s - loss: 5.2988 - acc: 0.0043 - val_loss: 5.2983 - val_acc: 0.0070
Epoch 4/10
100000/100000 [==============================] - 20s - loss: 5.2989 - acc: 0.0047 - val_loss: 5.2983 - val_acc: 0.0067
Epoch 5/10
100000/100000 [==============================] - 20s - loss: 5.2988 - acc: 0.0041 - val_loss: 5.2983 - val_acc: 0.0068
Epoch 6/10
100000/100000 [==============================] - 21s - loss: 5.2989 - acc: 0.0045 - val_loss: 5.2983 - val_acc: 0.0068
Epoch 7/10
100000/100000 [==============================] - 21s - loss: 5.2989 - acc: 0.0045 - val_loss: 5.2983 - val_acc: 0.0066
Epoch 8/10
100000/100000 [==============================] - 21s - loss: 5.2987 - acc: 0.0044 - val_loss: 5.2983 - val_acc: 0.0066
Epoch 9/10
100000/100000 [==============================] - 20s - loss: 5.2987 - acc: 0.0047 - val_loss: 5.2983 - val_acc: 0.0066
Epoch 10/10
100000/100000 [==============================] - 21s - loss: 5.2987 - acc: 0.0047 - val_loss: 5.2983 - val_acc: 0.0064
```

Link to sample recordings

Link to Code

Task 5: Transfer Learning using Xception model with cifar10 dataset

Task 6: Fine Tuning post transfer learning

What is Transfer learning?

Transfer learning is storing knowledge gained while solving one problem and applying it to a different but associated problem. In this task, it is illustrated with an Image classification problem.

Xception is already used for image classification problems and it has been trained on the ImageNet data. The advantage of Xception architecture is that when compared with conventional convolution, we do not need to perform convolution across all channels. That means the number of connections would be fewer making the model less complex.

Steps Performed:

1. Reshaped the CIFAR10 data input according to required minimum dimensions (71*71) for Xception model.
2. Using transfer learning, built base model for cifar10 data after adjustments of final output layer.
3. Optimized the layers to find the best model for cifar data.
4. Fine-tuned hyperparameters for the best model is performed at much faster pace with this dataset.

Link to Code

Link to recorded observations

Observations:

1. Transfer learning could be an easy way for anyone to start with while building a CNN model.
2. Using pretrained models saved time as there were pretrained weights that could be used with our dataset.
3. Model accuracy seems to increase with decrease in model loss.
4. Time taken for training is very less, so training model becomes easy.

Best Model performance: