Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Part A

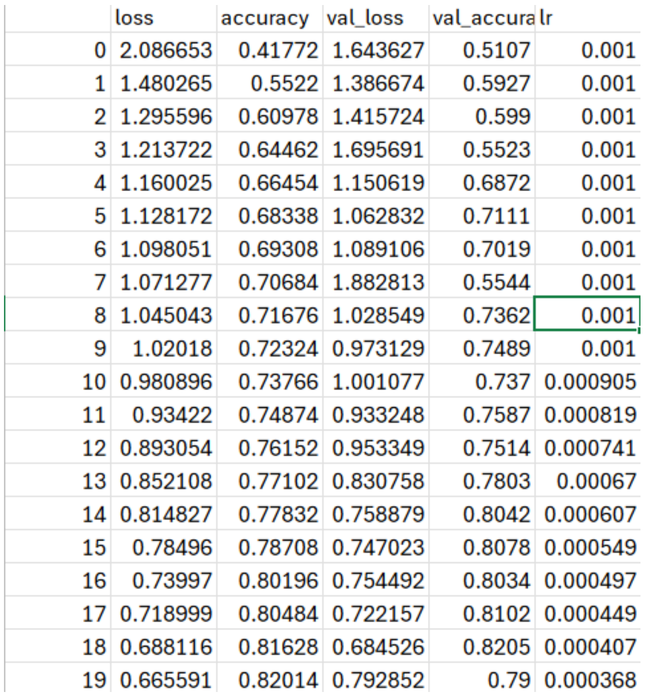
For experiment 1, we were tasked to train several custom-made convolutional neural networks on the CIFAR-10 dataset. How I approached this was to start with a simple topology with very few modifications such as learning rate and regularization (Batch and/or L2), However I quickly learned that I had to at least implement one of these techniques for the training/test accuracy to at least get to 80%. Hence instead of making 5-7 configurations, I made 9. The three things in common with all models that I created were the input shape, the output layer, and early stopping, with the input shape as 32x32 and the output layer 10 units respectively.

The first model (Appendix A) had a learning rate of 1e-3, a batch size of 64, and ran for 50 epochs. I decided to start with a small learning rate just to get a baseline metric to compare when I start improving. This one was surprisingly not bad but not how I wanted it, with a test accuracy of 76.82999968528748%. I needed to boost this metric to at least 80% for the model to pass with adequacy. While training, even though with a low learning rate, the first epoch ended with an approximate 43% accuracy. I think the key thing here is simplicity, because as soon as I make it complicated by changing the number of units, such as in model 2 (Appendix A) I boost the first 2 Convolution layers from 32 to 48, and then the next 64 to 96. That learned a lot slower than the first one, with the first epoch dropping to 33% accuracy. I also noticed that adding Dropout layers with increasing rates towards the top of the model significantly helped with converging faster.

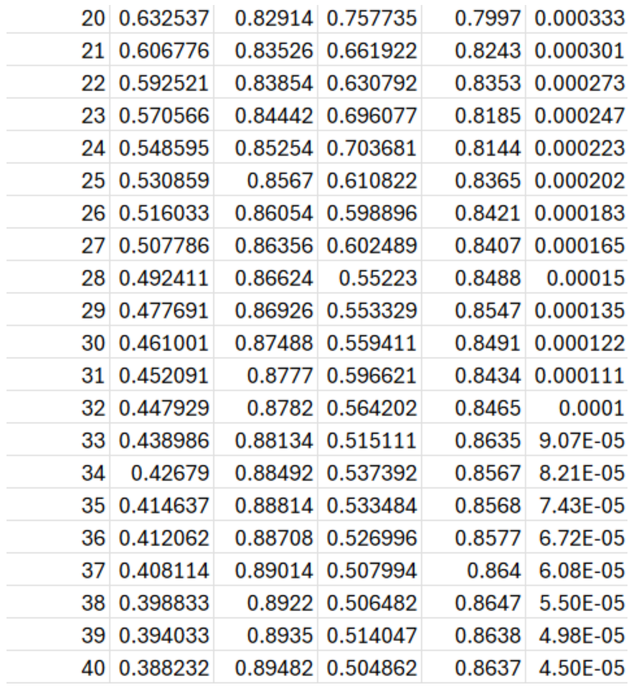
Model 9 (Appendix A) had the best test accuracy, with a score of 81.5999984741211%. The approach that helped this converge to the best accuracy but not necessarily faster was a learning rate scheduler, batch normalization, and l2 regularization as well as the obvious change of more layers. Basically, the learning rate scheduler will decrease the learning rate by a certain amount after running 10 epochs on a standard learning rate of 0.001. I have noticed after implementing this, it did take longer to converge but it reduced overfitting by doing so because this technique is used to fine-tune the learning process, enabling quicker convergence at the beginning by taking larger steps, and then slowing down to ensure more precise adjustments as the model approaches an optimal point. This technique is officially called learning rate decay and I used exponential decay to be exact.

In CNNs, model complexity is a significant factor. If a model is too complex relative to the data and task, it might overfit, learning noise in the training data rather than generalizing from it. The use of validation performance metrics helps in detecting this issue. The calculations of test accuracies after training sessions (e.g., "Model 6 Test accuracy: 79.58%") is that the final model performance is evaluated on unseen data, which is essential to assess the true effectiveness of the model. The CNN configurations on the CIFAR-10 dataset, employing methods like multiple epochs training, early stopping, and detailed performance logging to optimize model tuning and prevent overfitting. The models are assessed on both training and unseen test data, with final accuracies suggesting a focus on generalization. Things that I could improve on, or change could be implemented by hyperparameter optimization and different learning rate schedules such as cyclical learning instead of exponential.

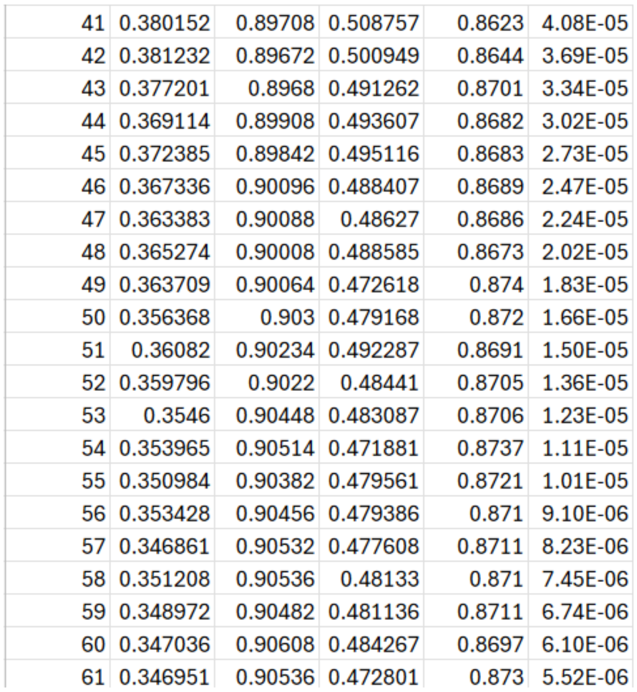
Training Log for Model 9



	loss	accuracy	val_loss	val_accura	lr
0	2.086653	0.41772	1.643627	0.5107	0.001
1	1.480265	0.5522	1.386674	0.5927	0.001
2	1.295596	0.60978	1.415724	0.599	0.001
3	1.213722	0.64462	1.695691	0.5523	0.001
4	1.160025	0.66454	1.150619	0.6872	0.001
5	1.128172	0.68338	1.062832	0.7111	0.001
6	1.098051	0.69308	1.089106	0.7019	0.001
7	1.071277	0.70684	1.882813	0.5544	0.001
8	1.045043	0.71676	1.028549	0.7362	0.001
9	1.02018	0.72324	0.973129	0.7489	0.001
10	0.980896	0.73766	1.001077	0.737	0.000905
11	0.93422	0.74874	0.933248	0.7587	0.000819
12	0.893054	0.76152	0.953349	0.7514	0.000741
13	0.852108	0.77102	0.830758	0.7803	0.00067
14	0.814827	0.77832	0.758879	0.8042	0.000607
15	0.78496	0.78708	0.747023	0.8078	0.000549
16	0.73997	0.80196	0.754492	0.8034	0.000497
17	0.718999	0.80484	0.722157	0.8102	0.000449
18	0.688116	0.81628	0.684526	0.8205	0.000407
19	0.665591	0.82014	0.792852	0.79	0.000368

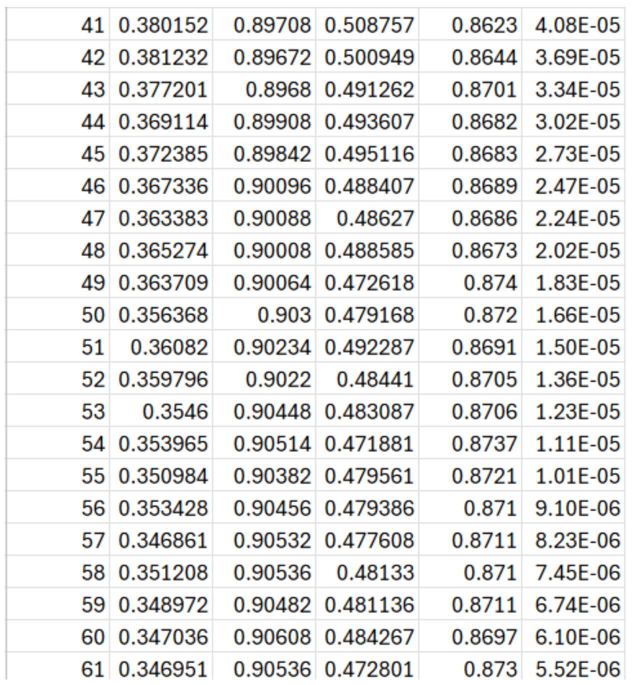


20	0.632537	0.82914	0.757735	0.7997	0.000333
21	0.606776	0.83526	0.661922	0.8243	0.000301
22	0.592521	0.83854	0.630792	0.8353	0.000273
23	0.570566	0.84442	0.696077	0.8185	0.000247
24	0.548595	0.85254	0.703681	0.8144	0.000223
25	0.530859	0.8567	0.610822	0.8365	0.000202
26	0.516033	0.86054	0.598896	0.8421	0.000183
27	0.507786	0.86356	0.602489	0.8407	0.000165
28	0.492411	0.86624	0.55223	0.8488	0.00015
29	0.477691	0.86926	0.553329	0.8547	0.000135
30	0.461001	0.87488	0.559411	0.8491	0.000122
31	0.452091	0.8777	0.596621	0.8434	0.000111
32	0.447929	0.8782	0.564202	0.8465	0.0001
33	0.438986	0.88134	0.515111	0.8635	9.07E-05
34	0.42679	0.88492	0.537392	0.8567	8.21E-05
35	0.414637	0.88814	0.533484	0.8568	7.43E-05
36	0.412062	0.88708	0.526996	0.8577	6.72E-05
37	0.408114	0.89014	0.507994	0.864	6.08E-05
38	0.398833	0.8922	0.506482	0.8647	5.50E-05
39	0.394033	0.8935	0.514047	0.8638	4.98E-05
40	0.388232	0.89482	0.504862	0.8637	4.50E-05

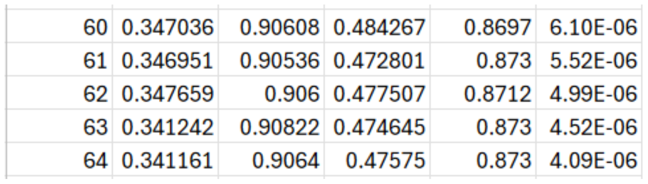


41	0.380152	0.89708	0.508757	0.8623	4.08E-05
42	0.381232	0.89672	0.500949	0.8644	3.69E-05
43	0.377201	0.8968	0.491262	0.8701	3.34E-05
44	0.369114	0.89908	0.493607	0.8682	3.02E-05
45	0.372385	0.89842	0.495116	0.8683	2.73E-05
46	0.367336	0.90096	0.488407	0.8689	2.47E-05
47	0.363383	0.90088	0.48627	0.8686	2.24E-05
48	0.365274	0.90008	0.488585	0.8673	2.02E-05
49	0.363709	0.90064	0.472618	0.874	1.83E-05
50	0.356368	0.903	0.479168	0.872	1.66E-05
51	0.36082	0.90234	0.492287	0.8691	1.50E-05
52	0.359796	0.9022	0.48441	0.8705	1.36E-05
53	0.3546	0.90448	0.483087	0.8706	1.23E-05
54	0.353965	0.90514	0.471881	0.8737	1.11E-05
55	0.350984	0.90382	0.479561	0.8721	1.01E-05
56	0.353428	0.90456	0.479386	0.871	9.10E-06
57	0.346861	0.90532	0.477608	0.8711	8.23E-06
58	0.351208	0.90536	0.48133	0.871	7.45E-06
59	0.348972	0.90482	0.481136	0.8711	6.74E-06
60	0.347036	0.90608	0.484267	0.8697	6.10E-06
61	0.346951	0.90536	0.472801	0.873	5.52E-06

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

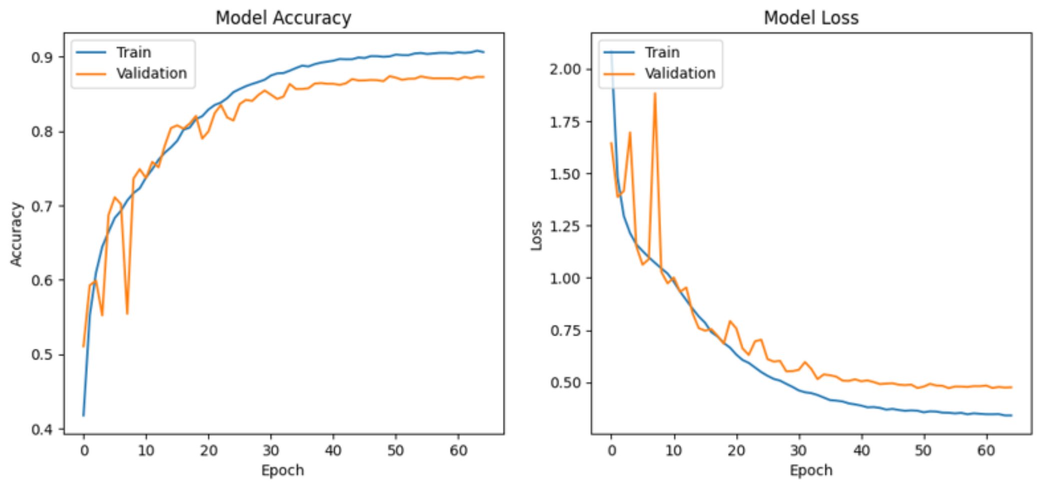


41	0.380152	0.89708	0.508757	0.8623	4.08E-05
42	0.381232	0.89672	0.500949	0.8644	3.69E-05
43	0.377201	0.8968	0.491262	0.8701	3.34E-05
44	0.369114	0.89908	0.493607	0.8682	3.02E-05
45	0.372385	0.89842	0.495116	0.8683	2.73E-05
46	0.367336	0.90096	0.488407	0.8689	2.47E-05
47	0.363383	0.90088	0.48627	0.8686	2.24E-05
48	0.365274	0.90008	0.488585	0.8673	2.02E-05
49	0.363709	0.90064	0.472618	0.874	1.83E-05
50	0.356368	0.903	0.479168	0.872	1.66E-05
51	0.36082	0.90234	0.492287	0.8691	1.50E-05
52	0.359796	0.9022	0.48441	0.8705	1.36E-05
53	0.3546	0.90448	0.483087	0.8706	1.23E-05
54	0.353965	0.90514	0.471881	0.8737	1.11E-05
55	0.350984	0.90382	0.479561	0.8721	1.01E-05
56	0.353428	0.90456	0.479386	0.871	9.10E-06
57	0.346861	0.90532	0.477608	0.8711	8.23E-06
58	0.351208	0.90536	0.48133	0.871	7.45E-06
59	0.348972	0.90482	0.481136	0.8711	6.74E-06
60	0.347036	0.90608	0.484267	0.8697	6.10E-06
61	0.346951	0.90536	0.472801	0.873	5.52E-06



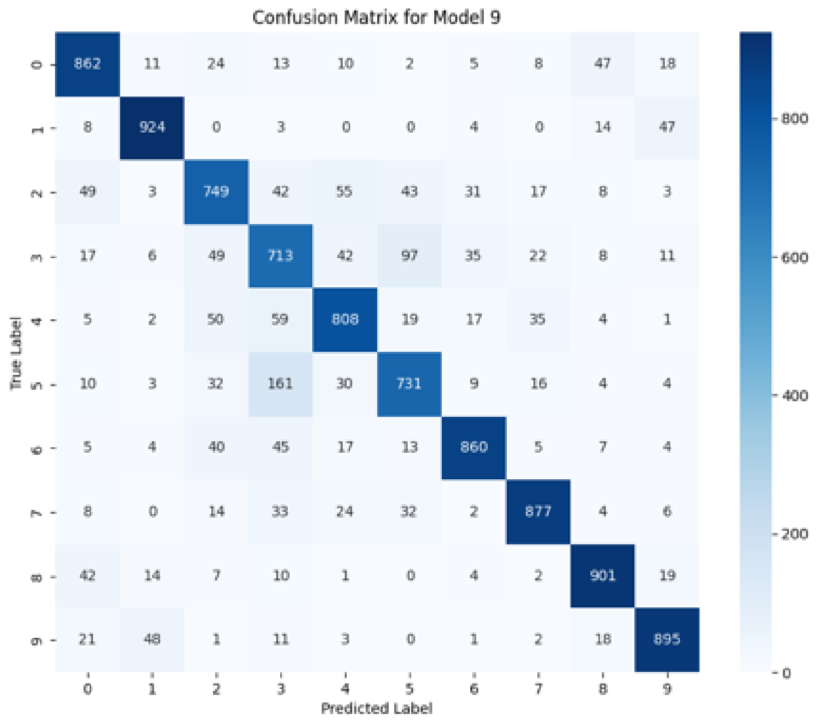
60	0.347036	0.90608	0.484267	0.8697	6.10E-06
61	0.346951	0.90536	0.472801	0.873	5.52E-06
62	0.347659	0.906	0.477507	0.8712	4.99E-06
63	0.341242	0.90822	0.474645	0.873	4.52E-06
64	0.341161	0.9064	0.47575	0.873	4.09E-06

Training/Test Accuracy-Loss Curves for Model 9



Confusion Matrix for Model 9

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024



	0	1	2	3	4	5	6	7	8	9
0	862	11	24	13	10	2	5	8	47	18
1	8	924	0	3	0	0	4	0	14	47
2	49	3	749	42	55	43	31	17	8	3
3	17	6	49	713	42	97	35	22	8	11
4	5	2	50	59	808	19	17	35	4	1
5	10	3	32	161	30	731	9	16	4	4
6	5	4	40	45	17	13	860	5	7	4
7	8	0	14	33	24	32	2	877	4	6
8	42	14	7	10	1	0	4	2	901	19
9	21	48	1	11	3	0	1	2	18	895

For experiment 2, we were tasked to first conduct transfer learning on two pretrained models from the Keras library: Resnet-50 and VGG-19. Secondly, from our best saved model from experiment 1 (in this case for me, model 9) conduct transfer learning on that. The key thing is that most layers from each model cannot be trained when doing transfer learning and only the newly added layers will be trained. For the Resnet and VGG, I trained them on the CIFAR-10, CIFAR-100, and Tiny ImageNet datasets. Each model had its input shape and output layer adjusted for each dataset i.e. 100 units for the output layer for CIFAR-100 and 200 units for the Tiny ImageNet. Resnet and VGG also had their weights imported relative to the ImageNet dataset. I ran each model for 50 epochs

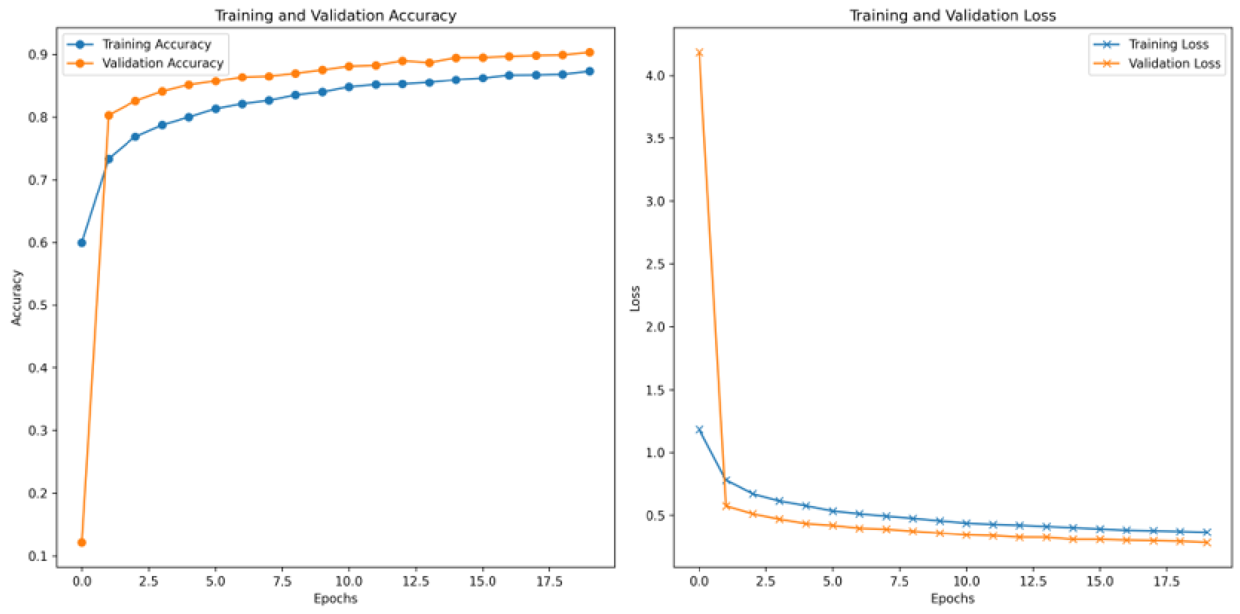
For the Resnet-50, it did fantastic on the CIFAR-10 dataset with a test accuracy of 80.7999643%. I think logically this makes sense. Out of the three datasets, the CIFAR-10 is the least complex with only 10 classes to make features out of. However, when it came to the other two, it did increasingly worse than CIFAR-10, with a test accuracy of 70.89999924% for the CIFAR-100 and about 68% on the Tiny ImageNet. What could be inferred from these results as to why the test accuracy is getting worse with each dataset, is that the Tiny ImageNet and CIFAR-100 are just more complex and there might be some class imbalances in those datasets. This means there might be an imbalance in the number of training examples per class. I also used a constant learning rate of 0.001 for all experiments with Resnet-50. While this is a common practice, it might not be optimal throughout the entire training process. Implementing a learning rate schedule that decreases the learning rate as the model approaches convergence could potentially improve performance, especially in later epochs where finer adjustments are necessary. However, with these experiments, I tried not to implement that, just to keep it simple anyways. In general, ResNet-50 uses residual blocks that include skip connections to allow gradients to flow through the network more effectively during training, which can prevent the vanishing gradient problem and allow for deeper networks.

For the VGG-19 model, results were kind of all over the place with no consistent pattern. The test accuracy for the CIFAR-10 dataset was around 60.25000214576721%,32.25000214576721% on the CIFAR-100, and about 99% for the Tiny ImageNet. The Tiny ImageNet surprised me the most and the reason why is even when both Resnet and VGG-19 were initialized with weights according to the ImageNet dataset, VGG-19 did significantly better at classifying. VGG-19's straightforward stacking of layers might have provided more robust feature extraction for the Tiny ImageNet dataset's larger and more diverse images compared to CIFAR-10 and CIFAR-100. ResNet-50, while generally more efficient due to its skip connections, might not always translate to better performance depending on how well the residual learning is adapted to the specifics of the dataset and task. Both CIFAR-10 and CIFAR-100 are relatively small in image size (32x32 pixels), which might not leverage the depth and complexity of VGG-19 effectively, leading to potential overfitting compared to Tiny ImageNet, where the images are larger (typically resized to 224x224 for these models). This gives the model more information to work with and might explain why VGG-19 performed relatively better on Tiny ImageNet.

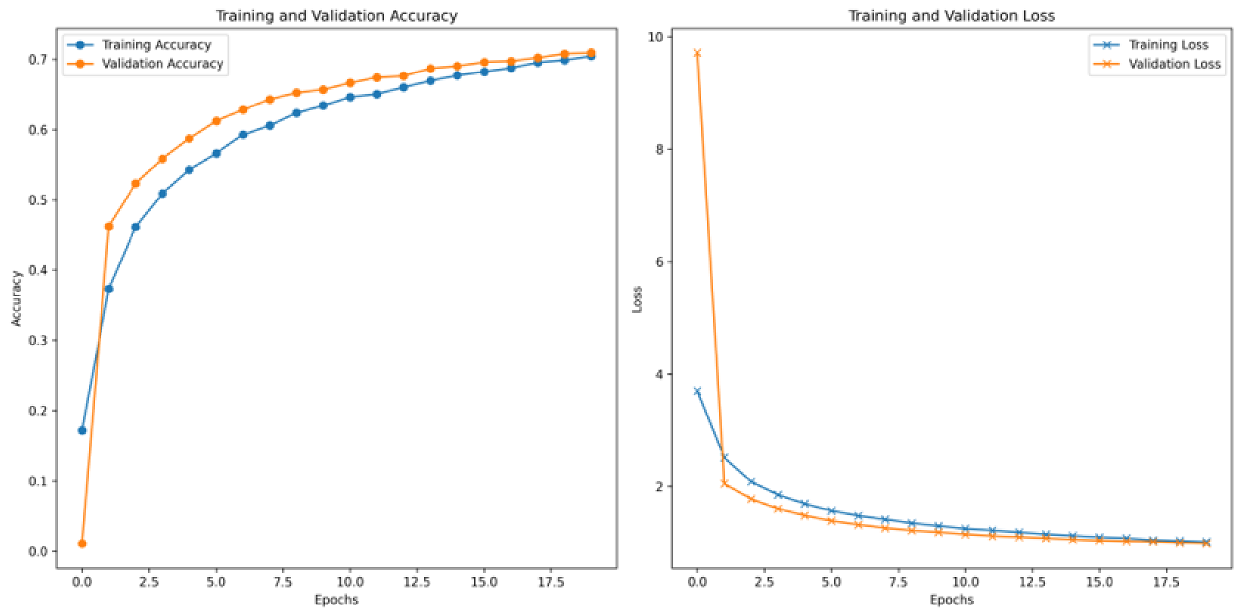
For the custom topology from experiment 1, my 9th configuration did not do well on the CIFAR-100 and Tiny ImageNet, with test accuracies 71.89234% and 40.02387% respectively. The reasoning behind this could be that The custom CNN topology is somewhat typical, featuring convolutional layers, batch normalization, pooling layers, and dropout for regularization. The use of dropout and batch normalization should theoretically help prevent overfitting by regularizing the model during training. However, the success heavily depends on whether the features learned from CIFAR-10 are generalizable enough for CIFAR-100 and Tiny ImageNet. The architecture is deep and complex enough (with layers going up to 128 filters) for CIFAR-10 but may not generalize well without adjustments to CIFAR-100 or Tiny ImageNet, which have more classes and potentially more complex image features. CIFAR-100 is much more complex than CIFAR-10, with 100 classes compared to 10. This means that the intra-class variation is higher, and the model needs to learn more fine-grained features to perform well. This dataset is even more complex than CIFAR datasets, with more classes and larger image sizes. It requires the model to handle a higher resolution and more varied imagery, which could be challenging if the model has primarily learned lower-level features relevant to the smaller images and fewer classes of CIFAR-10.

For experiment 2, please refer to appendices B-G for model implementation.

Training/Validation or Test Accuracy-Loss Curve for Resnet-50: CIFAR-10

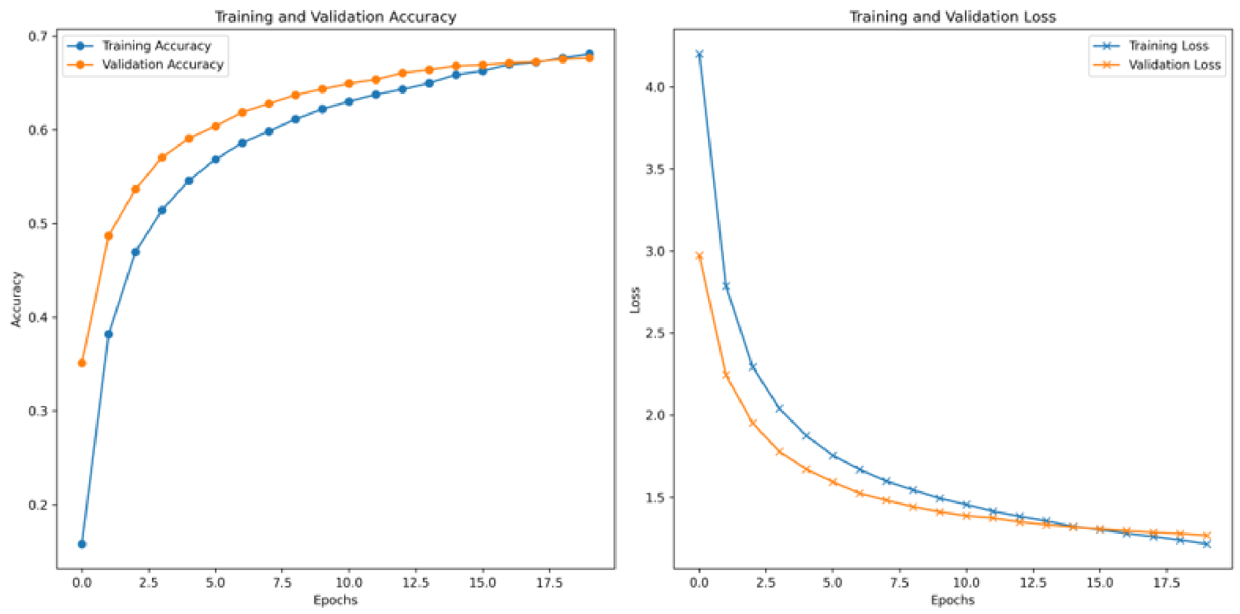


Training/Validation or Test Accuracy-Loss Curve for Resnet-50: CIFAR-100

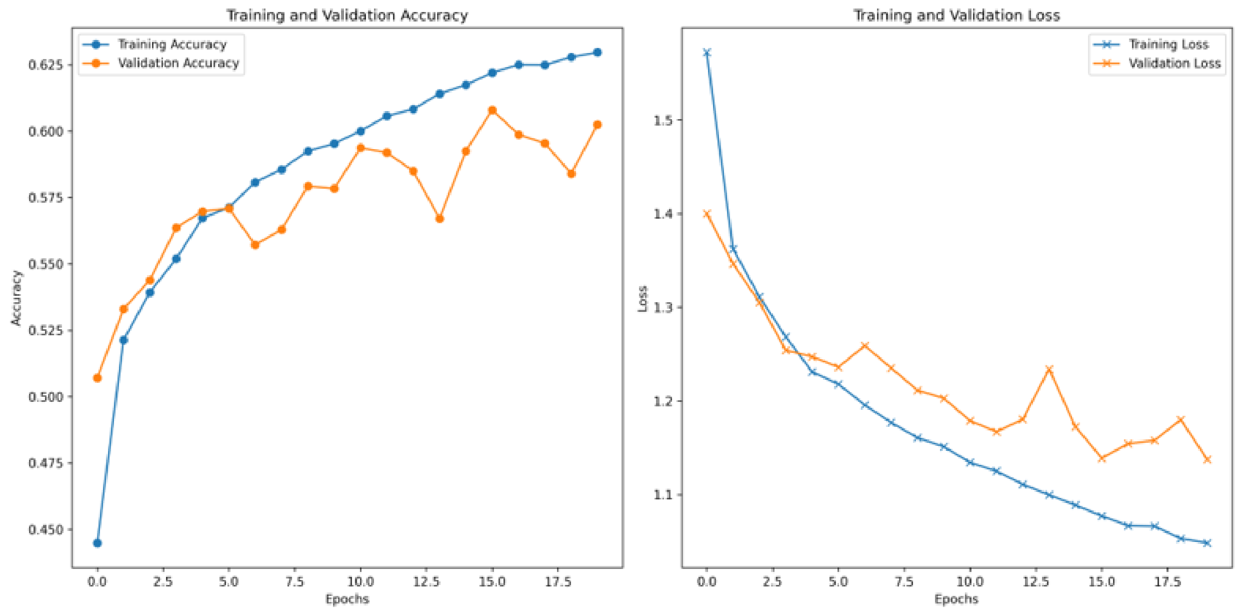


Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Training/Validation or Test Accuracy-Loss Curve for Resnet-50: Tiny ImageNet

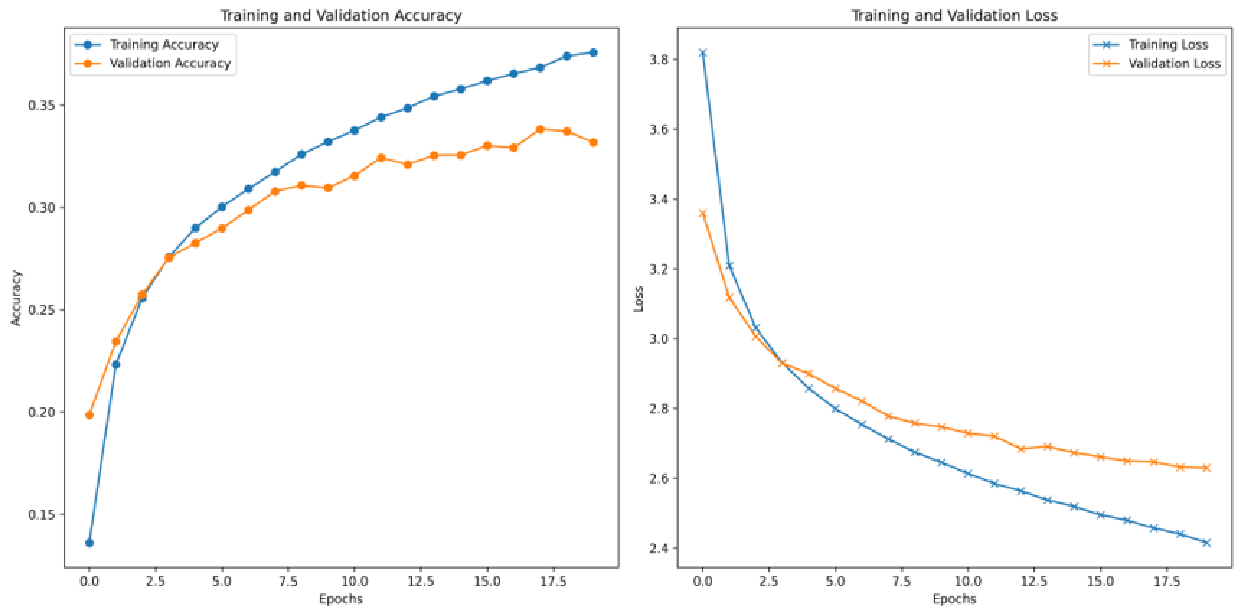


Training/Validation or Test Accuracy-Loss Curve for VGG-19: CIFAR-10

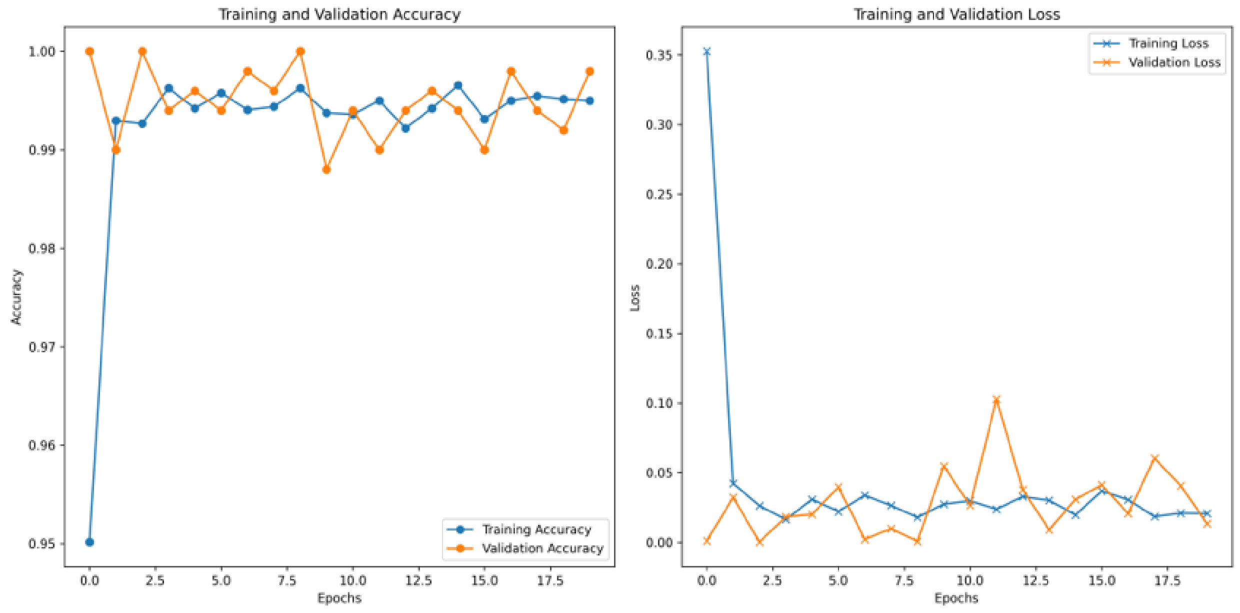


Training/Validation or Test Accuracy-Loss Curve for VGG-19: CIFAR-100

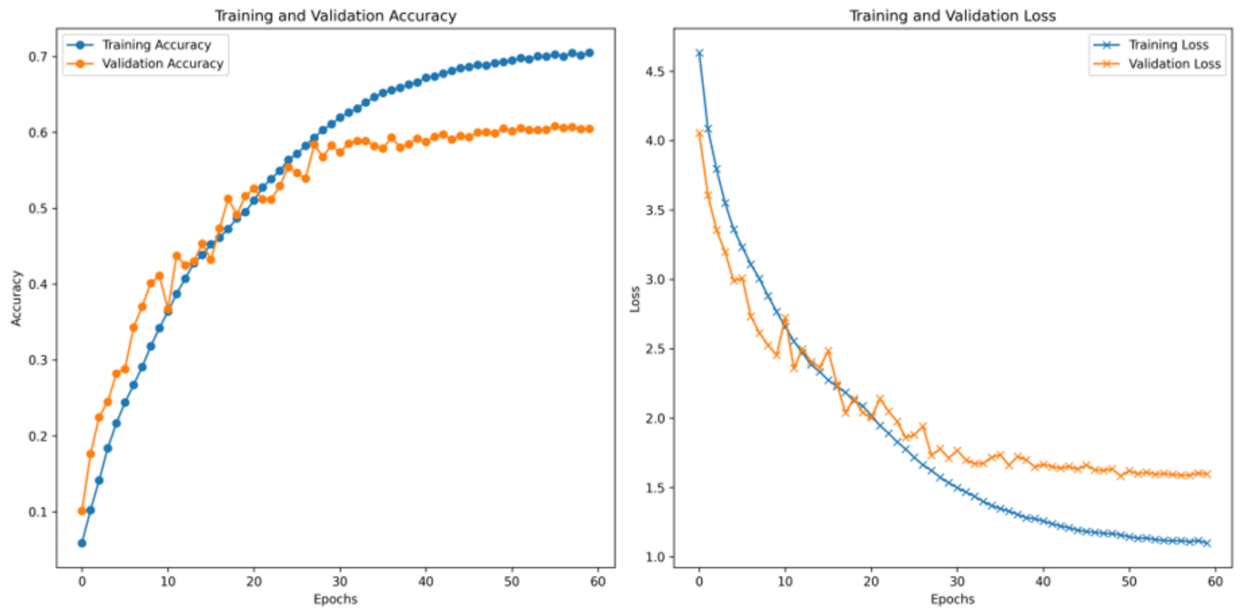
Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024



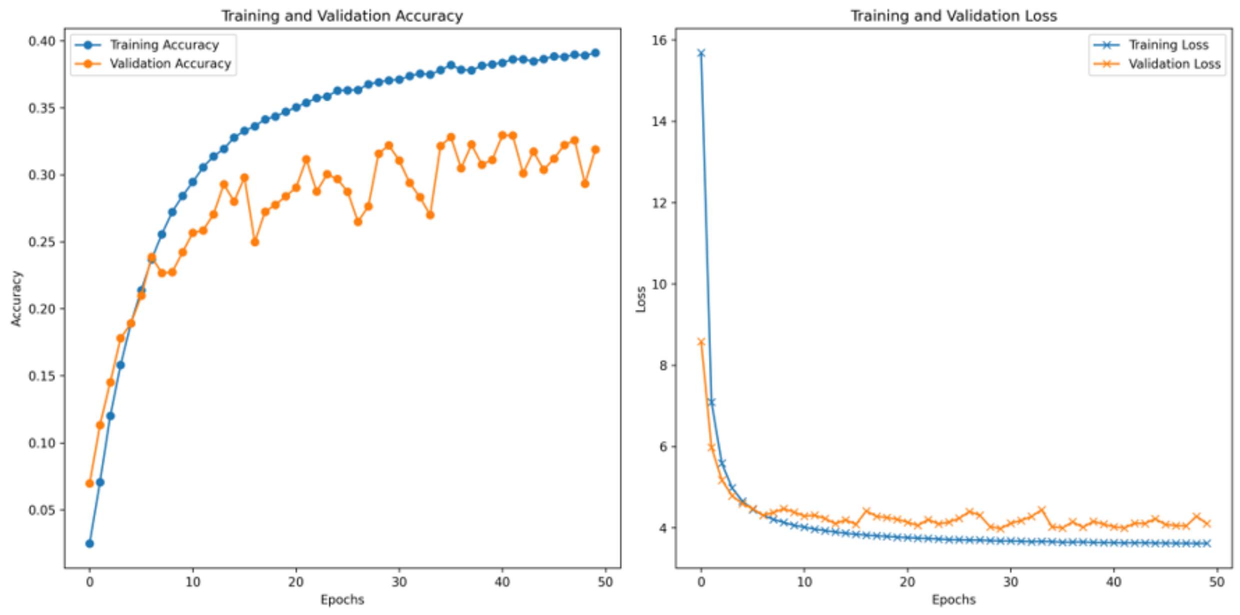
Training/Validation or Test Accuracy-Loss Curve for VGG-19: Tiny ImageNet



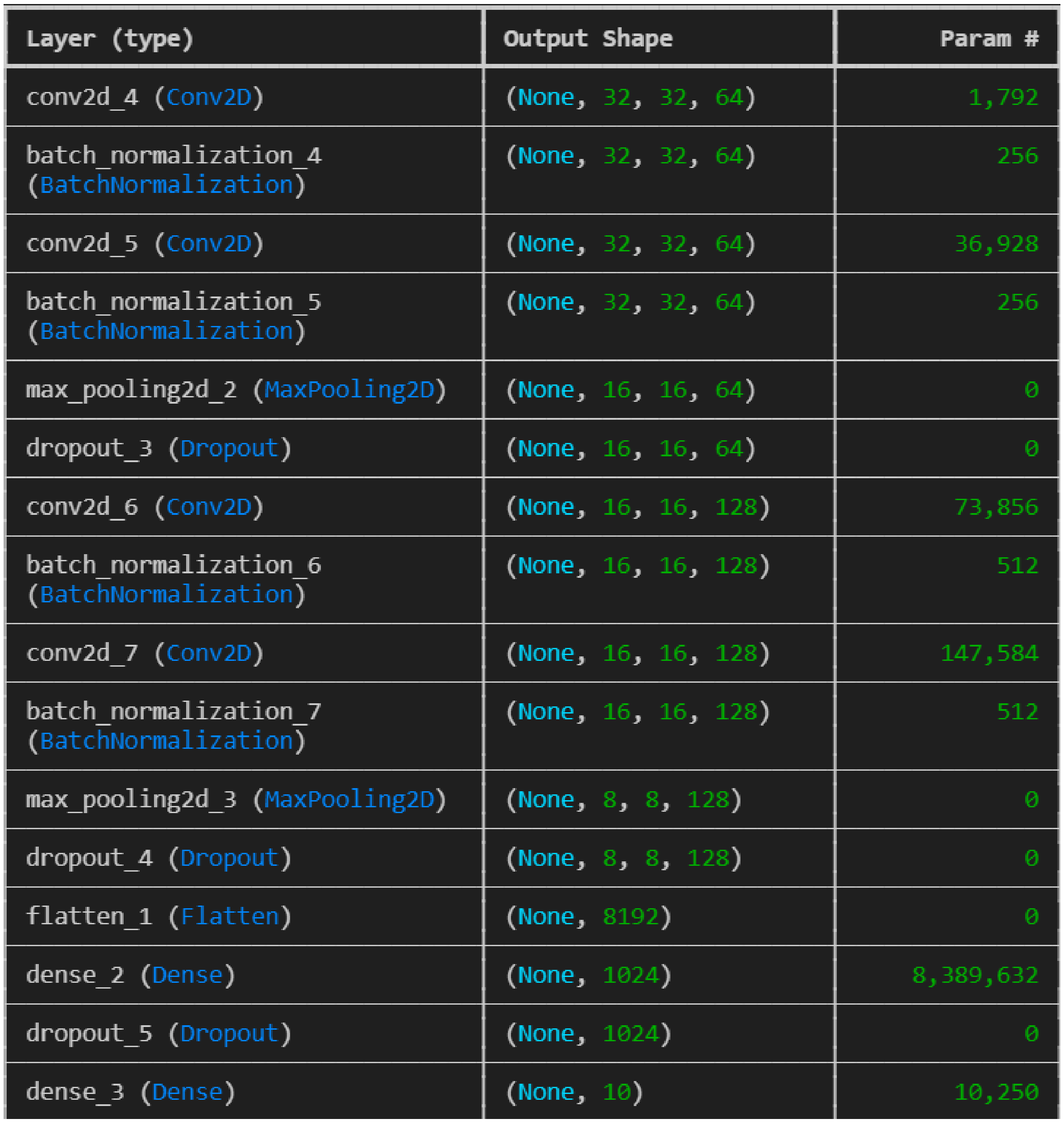
Training/Validation or Test Accuracy-Loss Curve for Model 9 from Experiment 1: CIFAR-100



Training/Validation or Test Accuracy-Loss Curve for Model 9 from Experiment 1: Tiny ImageNet



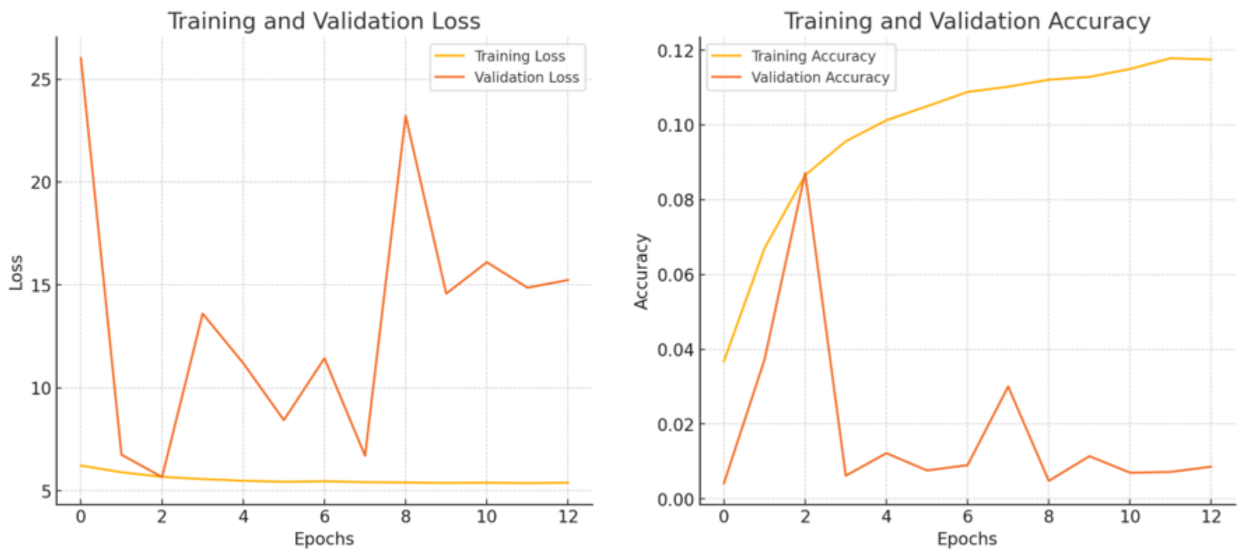
Custom Topology



Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_4 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_5 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_5 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_3 (Dropout)	(None, 16, 16, 64)	0
conv2d_6 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_6 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_7 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_7 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_4 (Dropout)	(None, 8, 8, 128)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 1024)	8,389,632
dropout_5 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 10)	10,250

For experiment 3, we were tasked to fine-tune our best custom model from experiment 1 on the Tiny ImageNet dataset. The process of fine-tuning is supposed to improve the model by introducing a more complex dataset than what it has been trained on. Therefore, when applying a simpler dataset to it later after fine-tuning, it will pick out more complex features. I did a bit of research on the best practices for fine-tuning and many sources have told me to run the model for a small number of epochs, like 10 or 20. I wasn't looking for a best accuracy score here because later in experiment 4, we will see the effects of this fine-tuning. This resulted in a test accuracy score of 10.03489%. I kept the learning rate at a standard of 0.001. See Appendix H for more details.

Training/Validation or Test Accuracy-Loss Curve for Fine-Tuning Model 9 from Experiment 1: Tiny ImageNet



For experiment 4, I saved the model generated after fine-tuning in experiment 3 and ran it again on the CIFAR-10 dataset to see if it retained its previous level of performance, or if it gained any feature extraction/representation power. I ran it with a very low learning rate at 1.00E-4 and implemented early stopping. The test accuracy resulted in 67.0799970626831%. After retraining on CIFAR-10, the model does not seem to retain its initial performance level. While the test accuracy is around 67%, which is reasonable, it doesn't match the higher performance typically expected from models trained directly on CIFAR-10 without the intermediate fine-tuning on a more complex dataset like ImageNet. This drop in performance could be due to several factors. Training on ImageNet might have led to weights that are not optimal for CIFAR-10, due to the vast differences in image features and class distributions. This can cause the model to learn features that are not useful or even detrimental when the model is brought back to CIFAR-10. After adapting to a complex task like ImageNet classification, the model parameters might have diverged significantly from those optimal for CIFAR-10, leading to a form of "forgetting" the simpler features that are more relevant to CIFAR-10. The learning rate and other hyperparameters optimized for CIFAR-10 initially might not have been ideal once the model was re-adapted to CIFAR-10 post-ImageNet training. See Appendix I for more details.

Part B

For experiment 1, we were tasked to replicate the segmentation process of the yolov8 model for 3 new images from a dataset we picked ourselves. I decided to go with the COCO

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

dataset, a very common one used for training and validation of segmentation models. The segmentation results are shown below.

Image 1

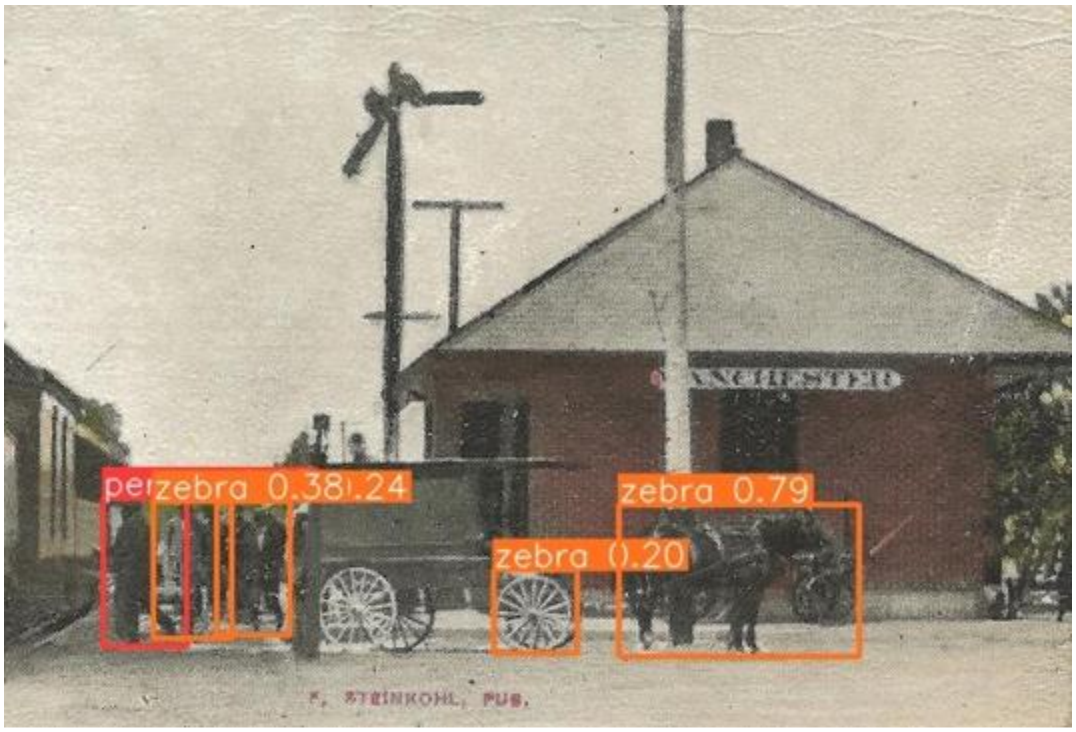
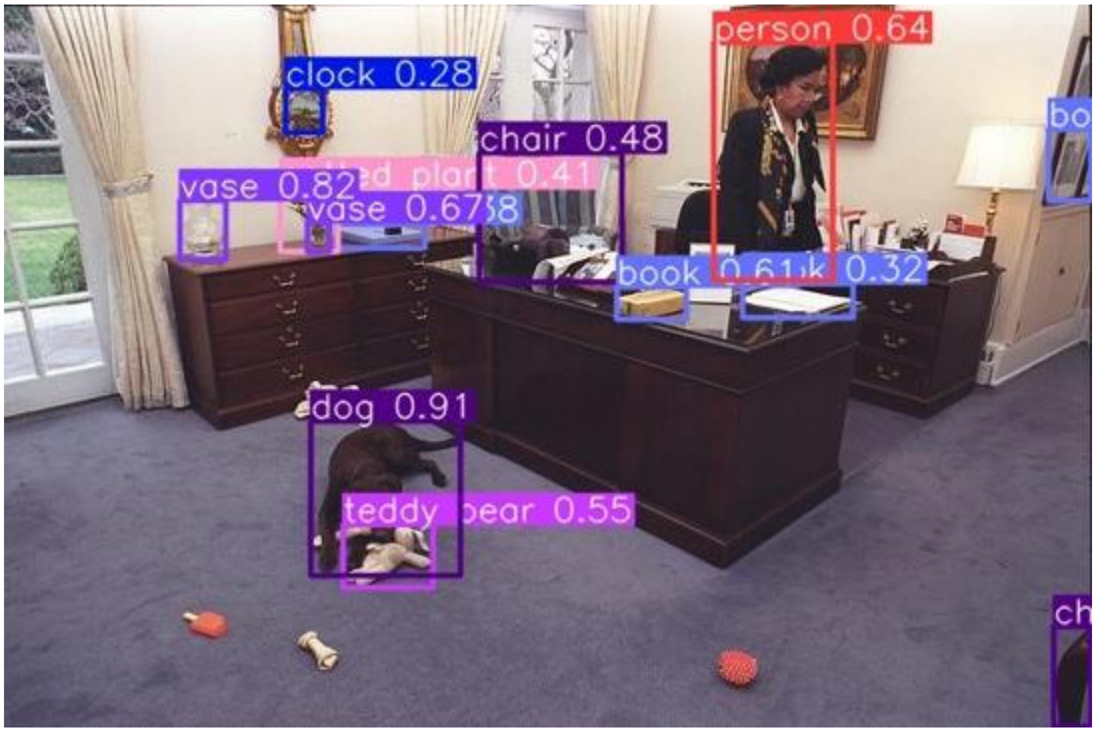
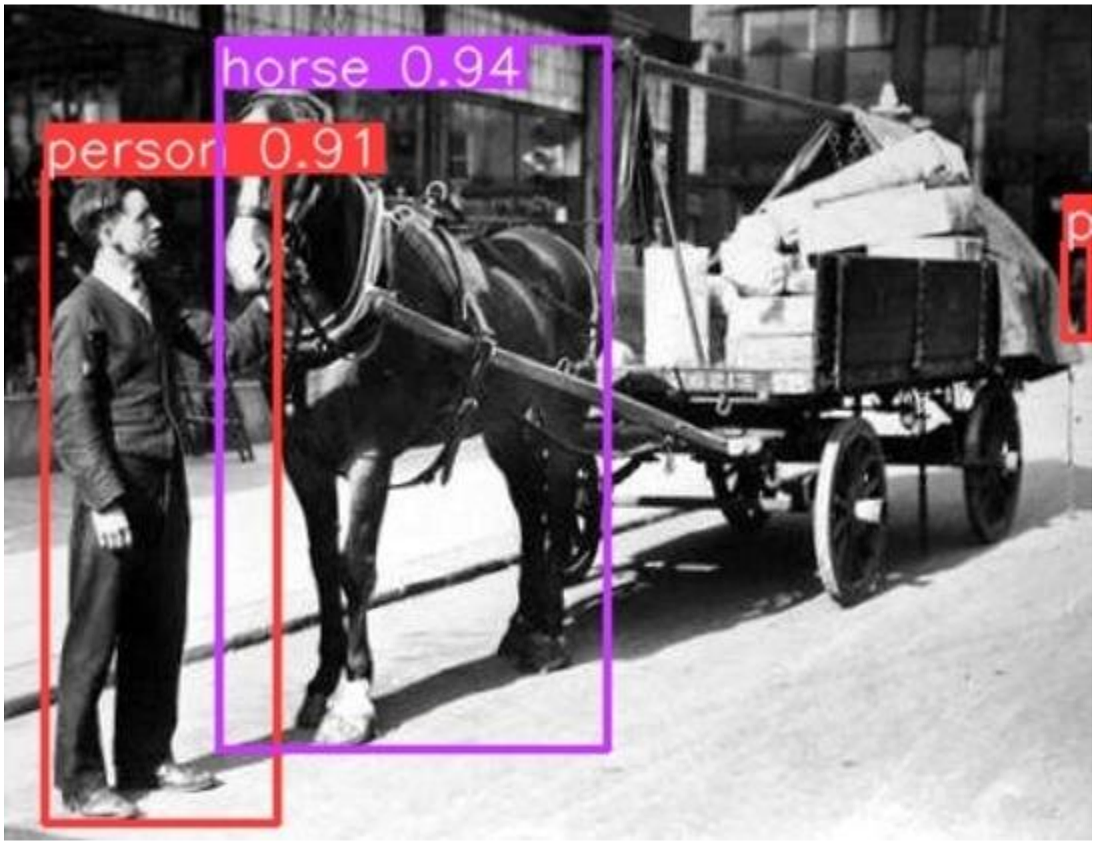


Image 2



Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Image 3



When gathering results, I only grabbed the best IoU score for each object respective to the image. It seems to me that it is very good at identifying people, animals, and objects with clearly defined shapes like vases. For the first image, it seemed to have misclassified the horse as a zebra probably due to the quality of the image or due to the reins on the horse that give a striped pattern on it, like a zebra. It also misclassified the wheel as a zebra because it does have a striped pattern. So there seems to be class consistency issues when running the yolov8 model. See Appendix J for more details on implementation.

For experiment 2, we were tasked to run the Segformer model on three random images from the ADE20K toy dataset. The three images before and after segmentation are as follows:

Image 1

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

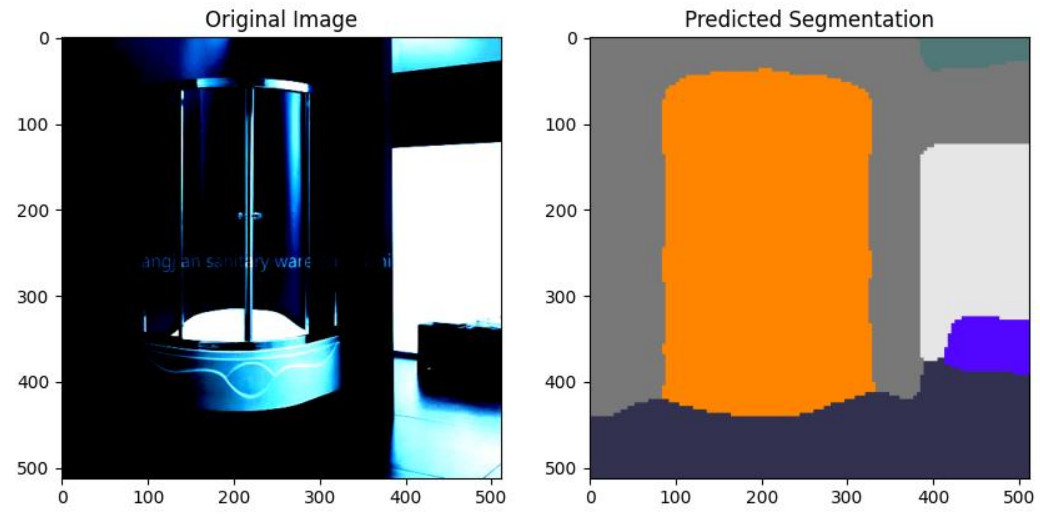


Image 2

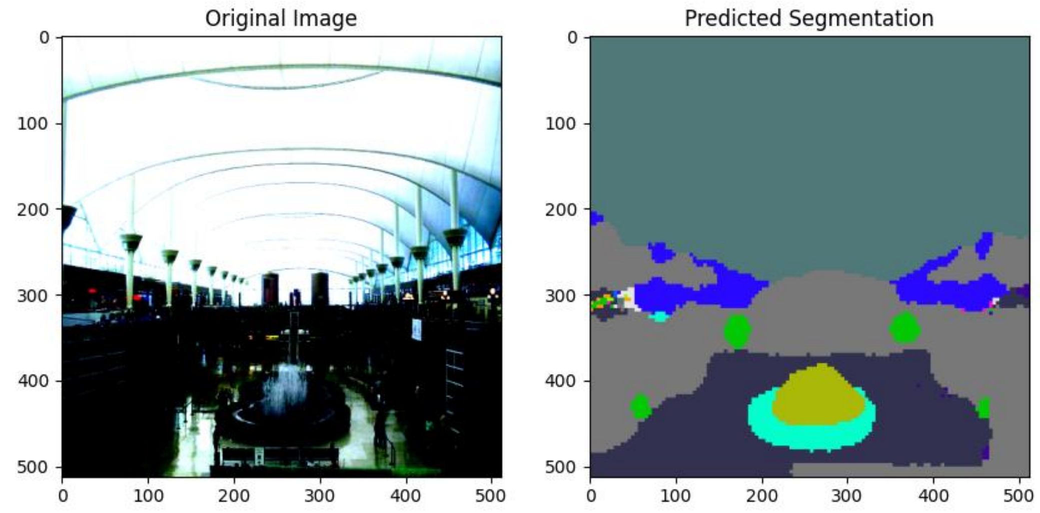
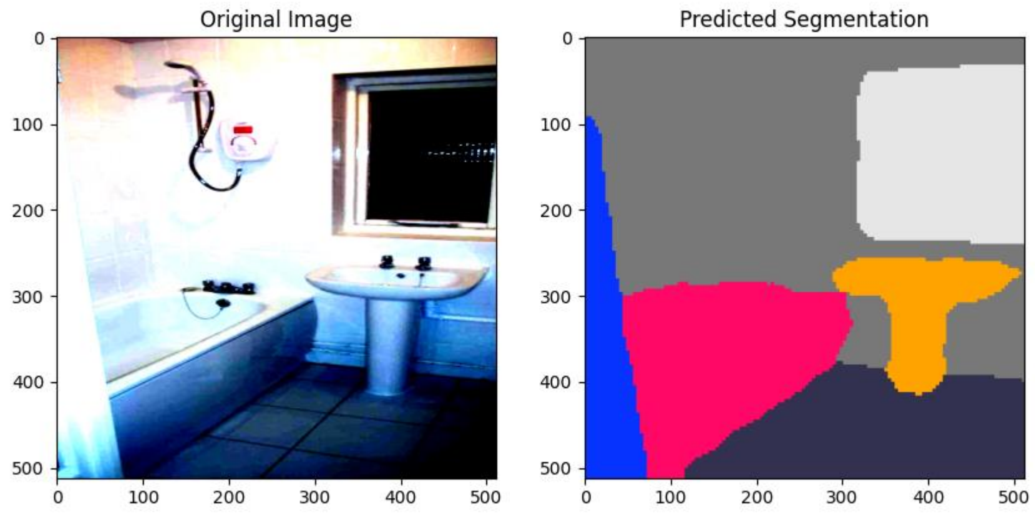


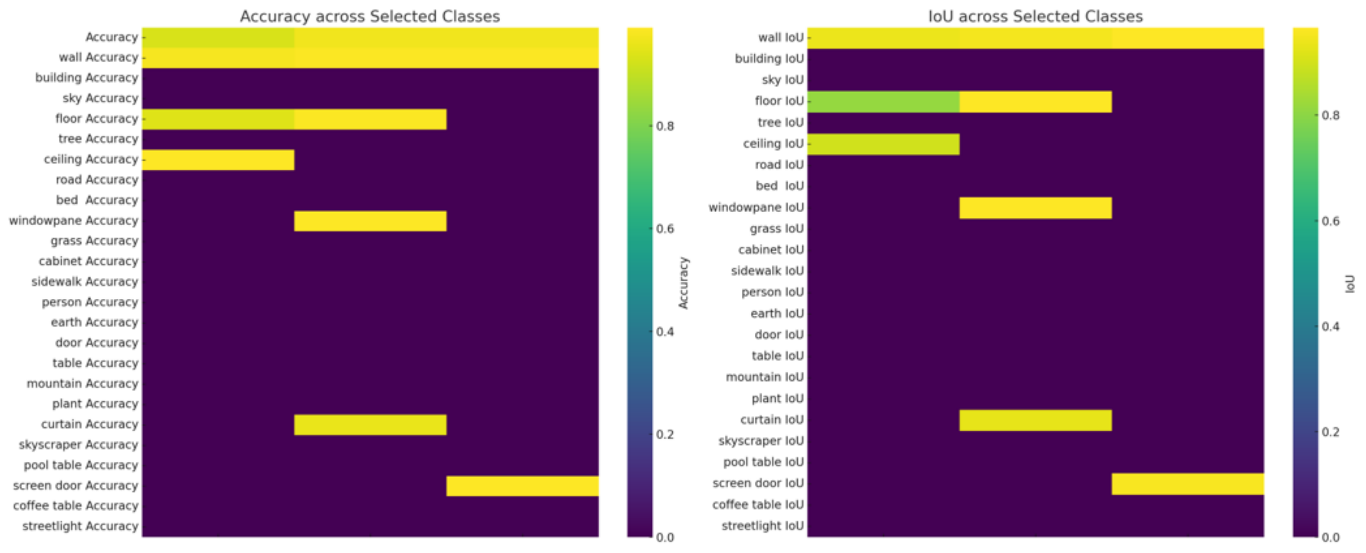
Image 3

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024



The accuracies of Image 1 and Image 3 were pretty good at about 96.1601% and 95.8504% respectively. However, Image 2 fell short at about 79.6494%. All images seem to have some augmentation done to them with contrast being different from the original image. For gathering results, I decided to extract all labels that each image had and their respective IoU scores (please see attached spreadsheet for more details). To better understand the data collection, I generated heatmaps for each feature of the image that was detected.

Heatmaps across all 3 Images



Some classes show consistently darker colors across their rows, indicating high accuracy. These classes are likely well-represented in the training data or are easier for the model to identify due to distinct features. Classes with lighter colors suggest lower accuracy. This could mean these classes are underrepresented in the data, have less distinct features, or are commonly confused with other classes. If there's significant variation in color within a single row (across the images), it suggests that the model's performance for that class varies significantly depending on the image. This could be due to variations in image conditions like lighting, presence of occlusions, or angles. Classes with uniformly dark colors across the row have high IoU scores, indicating that not only is the model accurately identifying these classes, but it is also precise in delineating the class boundaries in the segmentation. Rows with variable colors (from light to dark) suggest that the model's precision in segmenting the class fluctuates across images. This might be impacted by similar factors as mentioned for accuracy—image quality, class overlap, etc. Lighter rows indicate a consistent struggle with segmenting that class accurately across images. It may require revisiting either the model's ability to generalize for that class or the need for more representative training samples.

If the task involves understanding detailed geometric boundaries and the precise layout of objects, SegFormer is likely more advantageous. Its segmentation capabilities provide a deeper structural understanding of each part of the image at the pixel level, which is essential for applications requiring fine-grained analysis of the geometry and extent of objects. On the other hand, YOLOv8 would be more appropriate where the priority is identifying the presence and rough location of objects quickly rather than understanding their exact shapes or sizes in detail. See Appendix K for more details on implementation.

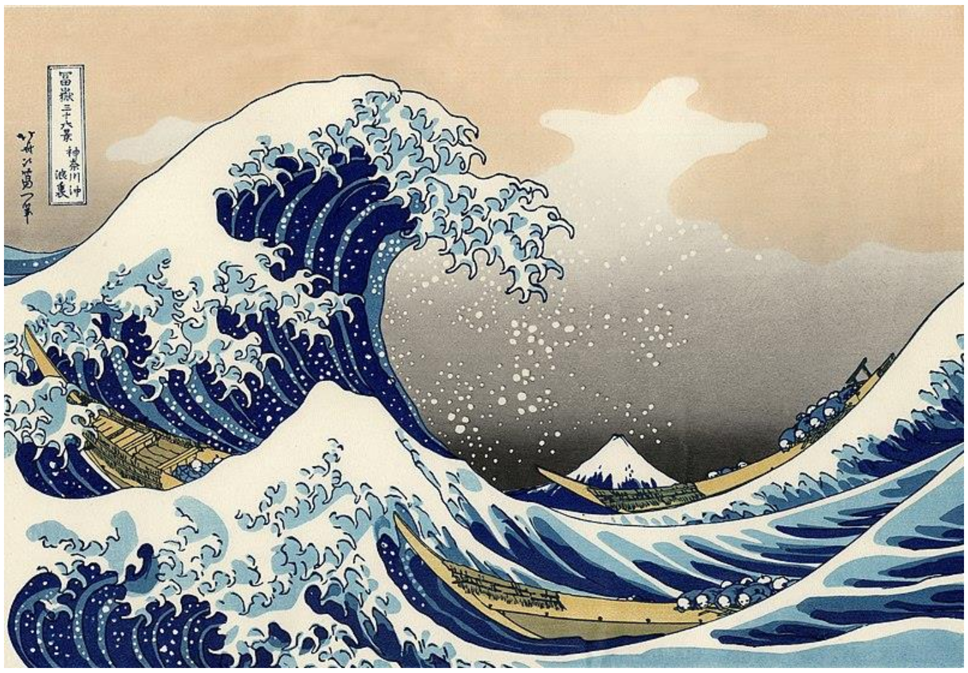
Generative Models

The best images that worked for me and that I used for style transfer was Starry Night by Van Gogh and . I used several images from my own album of photos with various features like people, water, flowers, landscapes, etc. The best that I generated were a picture of Igauzu Falls and this picture of purple flowers. The original images and the styled images are as follows.

Style References

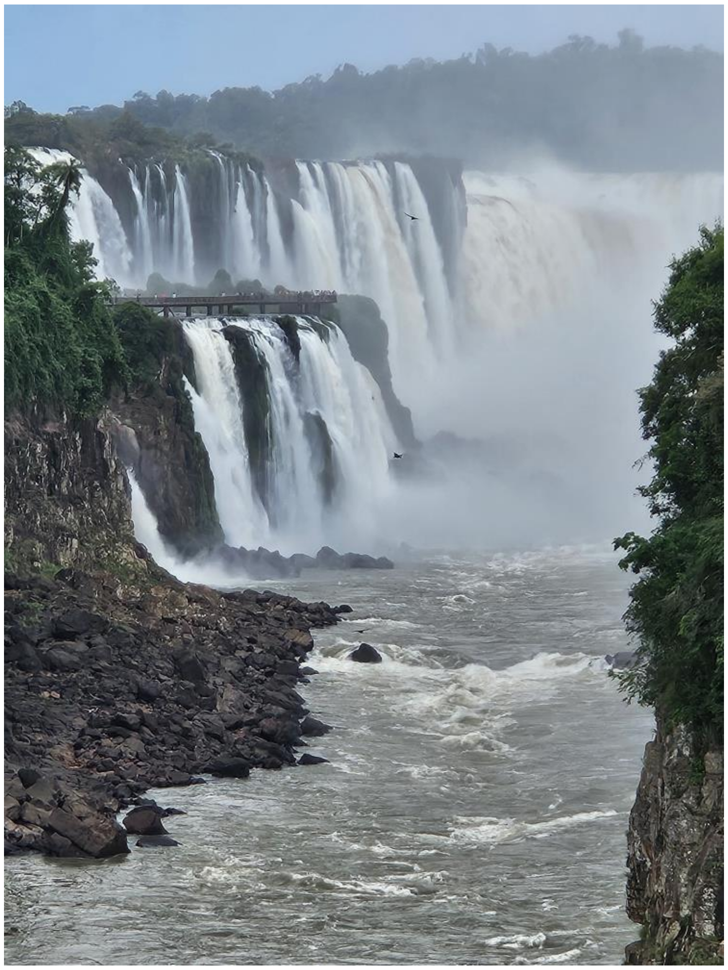
Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024



Original Image/Styled Image: Iguazu Falls (Starry Night)

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024



Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Original Image/Styled Image: Purple Flowers (The Great Wave)

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Reflection on Final and Course

I honestly would like to learn more advanced skills on how to troubleshoot on deep learning models when encountering problems like overfitting, the vanishing gradient problem, etc. Artificial intelligence is going to be our future when it comes to mundane tasks. In fact, as a software engineer, I am pretty sure AI generated programs that can be reused will be more of a common thing. Deep learning is a subset of machine learning where artificial neural networks—algorithms inspired by the human brain—learn from large amounts of data. These models automatically extract and learn features from data, which is a significant step beyond the manual feature extraction typical of traditional algorithms. In my opinion, intelligence typically involves the ability to: learn from experience, solve problems, understand complex ideas, adapt to new situations. However, unlike human intelligence, deep learning models are often not good at generalizing beyond their training data without further training. These models require large amounts of data to learn effectively and are only as good as the data they are trained on. Deep learning models are often described as "black boxes" because it can be challenging to understand how they have arrived at a particular decision.

In terms of this course, I had a lot of fun learning skills and learning about fundamental deep learning concepts. My advice for future students is to outsource and research some of these topics outside of class, especially if you don’t understand what is going on. Another thing is discussing (not cheating off) people in class on assignments and projects. This will help you understand the topics better and one person might have a better approach on solving the problem. Utilize your professor as a resource for help as well. They are professionals and are qualified on the topics they are teaching you. They will give you better insight.

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Appendix A – Experiment 1 Part A

```
# %%
!pip install -q transformers datasets evaluate

# %% [markdown]
# Import Libraries and Load Data

# %%
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import accuracy_score
from keras.utils import to_categorical

# Load CIFAR-10 dataset
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()

# %% [markdown]
# Data Preprocessing

# %%
# Normalize data
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Convert class vectors to binary class matrices
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)

# %% [markdown]
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# Define the Model Architecture

# %% [markdown]
# Configuration 1

# %%
model1 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])



# %% [markdown]
# Configuration 2 - Change in Layer Depth and Filters

# %%
model2 = Sequential([
    Conv2D(48, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    Conv2D(48, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    Conv2D(96, (3, 3), activation='relu'),
    Conv2D(96, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.3),

    Flatten(),
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    Dense(1024, activation='relu'),
    Dropout(0.4),
    Dense(10, activation='softmax')
])



# %% [markdown]
# Configuration 3 - Introduction of Batch Normalization

# %%
from keras.layers import BatchNormalization

model3 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    BatchNormalization(),
    Conv2D(32, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])




# %% [markdown]
# Configuration 4 - Different Activation Functions and Optimizer

# %%
model4 = Sequential([
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    Conv2D(32, (3, 3), activation='elu', input_shape=(32, 32, 3)),
    Conv2D(32, (3, 3), activation='elu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='elu'),
    Conv2D(64, (3, 3), activation='elu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(512, activation='elu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])




# %% [markdown]
# Configuration 5 - Varying Dropout Rates

# %%
model5= Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.35),

    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.45),
    Dense(10, activation='softmax')
])
```

```
# %% [markdown]
# Configuration 6 - Improving on Configuration 1 (Increased Learning Rate)
#

# %%
model6 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# %% [markdown]
# Configuration 7 - Improving on Configuration 6 (Increased Learning Rate, Data
Augmentation)

# %%

model7 = Sequential([
    Conv2D(64, (3, 3), padding='same', activation='relu', input_shape=(32, 32,
3)),
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.3),

    Conv2D(128, (3, 3), padding='same', activation='relu'),
    Conv2D(128, (3, 3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.4),
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    Flatten(),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=True,
    fill_mode='nearest',
    zoom_range=0.1
)
datagen.fit(X_train)

# %% [markdown]
# Configuration 9 - Improving on Configuration 7 (Batch Normalization, Learning
Rate Scheduler)

# %%
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.regularizers import l2

# Adjusting the model by adding Batch Normalization
model9 = Sequential([
    Conv2D(64, (3, 3), padding='same', activation='relu',
kernel_regularizer=l2(0.001), input_shape=(32, 32, 3)),
    BatchNormalization(),
    Conv2D(64, (3, 3), padding='same', activation='relu',
kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.3),

    Conv2D(128, (3, 3), padding='same', activation='relu',
kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Conv2D(128, (3, 3), padding='same', activation='relu',
kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.4),

    Flatten(),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Implementing a Learning Rate Scheduler
from keras.callbacks import LearningRateScheduler
from keras.optimizers import Adam

def scheduler(epoch, lr):
    if epoch < 10:
        return float(lr)
    else:
        return float(lr * tf.math.exp(-0.1))

lr_schedule = LearningRateScheduler(scheduler)
optimizer = Adam(learning_rate=0.001)

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=True,
    fill_mode='nearest',
    zoom_range=0.1
)
datagen.fit(X_train)

# %%
import numpy as np
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

model8 = model7
models = [model1, model2, model3, model4, model5, model6, model7, model8, model9]

test_accuracies = {}

for model in models:
    model_num = models.index(model) + 1
    learning_rate = 0.001  # Default learning rate
    if 1 <= model_num <= 3 or model_num == 5:
        learning_rate = 0.0001
    elif model_num == 4 or model_num == 6 or model_num == 7:
        learning_rate = 0.001
    elif model_num == 8:
        learning_rate = 0.01

    # Reinitialize the optimizer with the specific learning rate for each model
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

    checkpoint = ModelCheckpoint(f'model{model_num}.keras', save_best_only=True)
    early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=10,
restore_best_weights=True)
    epochs = 50 if model_num <= 6 else 100

    # Decide the fitting strategy based on model number
    if model_num == 9:
        history = model.fit(X_train, Y_train, batch_size=64, epochs=epochs,
validation_data=(X_test, Y_test), callbacks=[checkpoint, early_stopping])
    else:
        history = model.fit(datagen.flow(X_train, Y_train, batch_size=64),
epochs=epochs, validation_data=(X_test, Y_test), callbacks=[checkpoint,
early_stopping])

    # Evaluate model on test data
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    scores = model.evaluate(X_test, Y_test, verbose=1)
    print(f'Model {model_num} Test accuracy: {scores[1]*100}%')
    test_accuracies[model_num] = scores[1] * 100

    # Predict the outputs on the test set
    Y_pred = model.predict(X_test)
    Y_pred_classes = np.argmax(Y_pred, axis=1)
    Y_true = np.argmax(Y_test, axis=1)

    '''

    # Generate the confusion matrix
    cm = confusion_matrix(Y_true, Y_pred_classes)
    # Optionally visualize the confusion matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.title(f'Confusion Matrix for Model {model_num}')
    plt.savefig(f'confusion_matrix_model{model_num}.png')
    plt.close()
    '''



pd.DataFrame(test_accuracies, index=['Test
Accuracy']).to_csv('test_accuracies.csv')











# %%
model8 = model7
optimizer = Adam(learning_rate=0.01)
model8.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
checkpoint = ModelCheckpoint('model8.keras', save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=10,
restore_best_weights=True)
epochs = 50
history = model8.fit(datagen.flow(X_train, Y_train, batch_size=64),
epochs=epochs, validation_data=(X_test, Y_test), callbacks=[checkpoint,
early_stopping])
# Evaluate model on test data
scores = model8.evaluate(X_test, Y_test, verbose=1)
print(f'Model 8 Test accuracy: {scores[1]*100}%')
# Predict the outputs on the test set
Y_pred = model8.predict(X_test)
Y_pred_classes = np.argmax(Y_pred, axis=1)
Y_true = np.argmax(Y_test, axis=1)

# Generate the confusion matrix
cm = confusion_matrix(Y_true, Y_pred_classes)
# Optionally visualize the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title(f'Confusion Matrix for Model 8')
plt.savefig(f'confusion_matrix_model8.png')
plt.close()

# %%
import os
from google.colab import files
filename = f'confusion_matrix_model{8}.png'
files.download(filename)

# %%
model1.summary()
model2.summary()
model3.summary()
model4.summary()
model5.summary()
model6.summary()
model7.summary()
model8.summary()
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
model9.summary()
```

Appendix B – Resnet-50, CIFAR-10, CIFAR-100

```
# -*- coding: utf-8 -*-
"""resnet_cifar10_cifar100.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1OviNBvezdEoVxwXZD82WZo-BoP21inrS
"""


# Step 1: Import necessary libraries
import numpy as np
import tensorflow as tf
from keras.datasets import cifar10, cifar100
from keras.applications import ResNet50
from keras.models import Model
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
from keras.layers import GlobalAveragePooling2D, Dense, BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.utils import to_categorical
import pandas as pd

# Step 2: Load and prepare the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar100.load_data()
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
y_train = to_categorical(y_train, 100)
y_test = to_categorical(y_test, 100)

# Step 3: Load the ResNet50 base model
base_model = ResNet50(include_top=False, weights='imagenet', input_shape=(32, 32,
3))

# Step 4: Freeze layers without BatchNormalization
for layer in base_model.layers:
    if not isinstance(layer, BatchNormalization):
        layer.trainable = False

# Step 5: Add custom layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(100, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# Step 6: Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Step 7: Set up callbacks
checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True,
monitor='val_loss')
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)

# Step 8: Train the model
```

```
history = model.fit(x_train, y_train, epochs=20, batch_size=64,
validation_data=(x_test, y_test), callbacks=[checkpoint, early_stopping])

# Step 9: Evaluate the model
test_accuracy = {}
score = model.evaluate(x_test, y_test)
print('Test accuracy:', score[1])
test_accuracy['ResNet50_CIFAR10'] = score[1]*100
pd.DataFrame(test_accuracy, index=['Test
Accuracy']).to_csv('resnet_cifar_100_test_accuracy.csv')

import os
from google.colab import files
filename = 'resnet_cifar_100_test_accuracy.csv'
files.download(filename)

'''
# Additional import for plotting
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Step 10: Make predictions
predictions = model.predict(x_test)
y_pred = np.argmax(predictions, axis=1)
y_true = np.argmax(y_test, axis=1)

# Step 11: Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", cm)

# Step 12: Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=np.arange(10),
yticklabels=np.arange(10))
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
# Step 13: save history to csv file
import pandas as pd
history_df = pd.DataFrame(history.history)
history_df.to_csv('history.csv', index=False)
'''
```

Appendix C – Resnet-50, Tiny ImageNet

```
# -*- coding: utf-8 -*-
"""resnet_imageNet.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1NOc0fbf8I-S7qGx-W2nK5_pbZlGHwvdn
"""
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
! git clone https://github.com/seshuad/IMagenet
! ls 'IMagenet/tiny-imagenet-200/'

import time
import imageio
import numpy as np

path = 'IMagenet/tiny-imagenet-200/'

def get_id_dictionary():
    id_dict = {}
    for i, line in enumerate(open( path + 'wnids.txt', 'r')):
        id_dict[line.replace('\n', '')] = i
    return id_dict

def get_class_to_id_dict():
    id_dict = get_id_dictionary()
    all_classes = {}
    result = {}
    for i, line in enumerate(open( path + 'words.txt', 'r')):
        n_id, word = line.split('\t')[:2]
        all_classes[n_id] = word
    for key, value in id_dict.items():
        result[value] = (key, all_classes[key])
    return result

def get_data(id_dict):
    print('starting loading data')
    train_data, test_data = [], []
    train_labels, test_labels = [], []
    t = time.time()
    for key, value in id_dict.items():
        train_data += [imageio.imread( path + 
'train/{}/images/{}_{}.JPEG'.format(key, key, str(i)), mode='RGB') for i in 
range(500)]
        train_labels_ = np.array([[0]*200]*500)
        train_labels_[:, value] = 1
        train_labels += train_labels_.tolist()

    for line in open( path + 'val/val_annotations.txt'):
        img_name, class_id = line.split('\t')[:2]
        test_data.append(imageio.imread( path +
'val/images/{}'.format(img_name) ,mode='RGB'))
        test_labels_ = np.array([[0]*200])
        test_labels_[0, id_dict[class_id]] = 1
        test_labels += test_labels_.tolist()

    print('finished loading data, in {} seconds'.format(time.time() - t))
    return np.array(train_data), np.array(train_labels), np.array(test_data),
np.array(test_labels)

train_data, train_labels, test_data, test_labels = get_data(get_id_dictionary())

print( "train data shape: ",  train_data.shape )
print( "train label shape: ", train_labels.shape )
print( "test data shape: ",   test_data.shape )
print( "test_labels.shape: ", test_labels.shape )

"""## Using Resnet50 for Transfer Learning and classify the Cifar - 100 data

### The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes,
with 6000 images per class. There are 50000 training images and 10000 test images

## Importing Libraries
"""

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from datetime import datetime
from keras.preprocessing import image
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Flatten, Conv2D, UpSampling2D, Dropout,
BatchNormalization, GlobalAveragePooling2D
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
"""## Importing the Cifar 100 Dataset"""

#cifar100 = tf.keras.datasets.cifar100
#(X_train, Y_train), (X_test,Y_test) = cifar100.load_data()

def plot_acc_loss(result):
  # function to plot the accuracy and loss graphs
  acc = result.history['accuracy']
  val_acc = result.history['val_accuracy']
  loss = result.history['loss']
  val_loss = result.history['val_loss']

  plt.figure(figsize=(20, 10))
  plt.subplot(1, 2, 1)
  plt.title("Training and Validation Accuracy")
  plt.plot(acc,color = 'green',label = 'Training Acuracy')
  plt.plot(val_acc,color = 'red',label = 'Validation Accuracy')
  plt.legend(loc='lower right')
  plt.ylabel('accuracy')
  plt.xlabel('epoch')
  plt.subplot(1, 2, 2)
  plt.title('Training and Validation Loss')
  plt.plot(loss,color = 'blue',label = 'Training Loss')
  plt.plot(val_loss,color = 'purple',label = 'Validation Loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(loc='upper right')
  #plt.show()
  plt.savefig('resnet_plot_CIFAR100')

"""##  Plotting some images from the dataset

## Splitting the train data again - we use the val set as test set and previous
test set for final predictions
"""


x_train,x_val,y_train,y_val = train_test_split(train_data, train_labels,
test_size = 0.2)

"""##  Onehot encoding of the outputs"""
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
x_train = x_train * 1.0/255

x_val = x_val * 1.0/255

X_test = test_data * 1.0/255

print(x_train.shape, x_val.shape, X_test.shape)
print(y_train.shape, y_val.shape, test_labels.shape)

"""##  Image Data Augmentation"""

train_datagen = ImageDataGenerator(
        rotation_range = 10,
        zoom_range = 0.1,
        width_shift_range = 0.1,
        height_shift_range = 0.1,
        shear_range = 0.1,
        horizontal_flip = True,
        vertical_flip = False
        )
train_datagen.fit(x_train)

"""##  Reduce Learning Rate if accuracy is not improving for 3 epochs"""

from keras.callbacks import ReduceLROnPlateau
learning_rate_reduction = ReduceLROnPlateau(
    monitor='val_accuracy',
    patience=3,
    verbose=1,
    factor=0.6,
    min_lr=1e-6)

"""##  Importing the Resnet Model"""

from tensorflow.keras.applications.resnet50 import ResNet50
resnet_model = ResNet50(
    include_top = False,
    weights = 'imagenet',
    input_shape = (224,224,3)
)

for layer in resnet_model.layers:
    if isinstance(layer, BatchNormalization):
        layer.trainable = True
    else:
        layer.trainable = False

resnet_model.summary()

"""##  Converting the output layer as per our dataset"""

from tensorflow.keras.layers import Resizing
model=tf.keras.models.Sequential()
model.add(Resizing(224, 224, interpolation='bilinear'))
model.add(resnet_model)
model.add(GlobalAveragePooling2D())
model.add(Dropout(.25))
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(200, activation='softmax'))

"""### **Issue** : The Cifar images are of the shape 32,32,3 and resnet model is
trained on images of 224,224,3

### *Solution* : Rezise Images - this is a costly operation and i eventually ran
out of memory many a times

### **Used Solution** : Keras provides an upsampling layer – called UpSampling2D
– which allows to perform upsampling operation within neural networks
* ### 32 * 7 = 224

### Stochastic gradient descent optimizer with momentum.
"""

optimizer = tf.keras.optimizers.SGD(learning_rate=1e-3, momentum=0.9)

"""Compile the model"""
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

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

"""We now Train the model on images. we are also checking to see if validation
accuracy doesnt improve we will reduce learning rate though the callback"""

result=model.fit(
    train_datagen.flow(x_train, y_train, batch_size = 64),
    validation_data = (x_val, y_val),
    epochs = 20,
    verbose = 1,
    callbacks = [learning_rate_reduction]
)

import pandas as pd
pd.DataFrame(result.history).to_csv('resnet_training_history_IMageNet.csv')

test_accuracy = {}
scores = model.evaluate(X_test, test_labels, verbose=1)
print("Accuracy: %.2f%%" % (scores[1]*100))
test_accuracy['resnet_50_imageNet'] = scores[1]
pd.DataFrame(test_accuracy, index=[0]).to_csv('resnet_50_imageNet_test_acc.csv')
import os
from google.colab import files
filename = 'resnet_50_imageNet_test_acc.csv'
files.download(filename)

model.summary()

"""## Plot accuracy and Loss"""

plot_acc_loss(result)

"""## Predictions,Accuracy and Confusion Matrix
### ** Requires modification for proper format **
"""
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
y_pred = np.argmax(model.predict(X_test), axis=-1)
y_true = test_labels.ravel()
print(y_pred.shape,y_true.shape)

print("Testing Accuracy: ", accuracy_score(y_true,y_pred))

cm = confusion_matrix(y_true,y_pred)
cm
```

Appendix D – VGG-19, CIFAR-10

```
# -*- coding: utf-8 -*-
"""vgg19_fixed_for_cifar10.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1J8__F3GuIHtuuwkAlO1kJupuOn_2rTfy
"""

# Import necessary libraries
import numpy as np
import tensorflow as tf
from keras.datasets import cifar100, cifar10
from keras.applications import VGG19
from keras.models import Model
from keras.layers import Dense, Flatten
from keras.optimizers import SGD
from keras.utils import to_categorical

# Load CIFAR-100 data
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Preprocess data
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# Load the VGG19 model pre-trained on ImageNet
vgg19_model = VGG19(include_top=False, weights='imagenet', input_shape=(32, 32,
3))

vgg19_model.trainable = False

# Modify the model to include new top layers for CIFAR-100 classification
x = Flatten()(vgg19_model.output)
x = Dense(512, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Create the full model
model = Model(inputs=vgg19_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=SGD(learning_rate=0.01, momentum=0.9),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, batch_size=128, epochs=20,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

import pandas as pd
pd.DataFrame(history.history).to_csv('history.csv')
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Appendix E – VGG-19, CIFAR-100

```
# -*- coding: utf-8 -*-
"""vgg19_updated_for_cifar100.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1Bl70x20PEueiAclw4DooCuTBg6A0k8RF
"""

# Import necessary libraries
import tensorflow as tf
from keras.datasets import cifar100
from keras.applications import VGG19
from keras.layers import Flatten, Dense, Dropout
from keras.models import Model
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator

# Load the CIFAR-100 dataset
(train_images, train_labels), (test_images, test_labels) = cifar100.load_data()

# Preprocess the data
train_images = tf.keras.applications.vgg19.preprocess_input(train_images)
test_images = tf.keras.applications.vgg19.preprocess_input(test_images)

# Set up the VGG19 base model
base_model = VGG19(include_top=False, weights='imagenet', input_shape=(32, 32,
3))
```

```python
# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Create custom layers for our classifier
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(100, activation='softmax')(x)

# Build the model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Create data augmentation configuration
train_datagen = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.2
)

# Fit the model
history = model.fit(train_datagen.flow(train_images, train_labels,
batch_size=64),
                    epochs=20,
                    validation_data=(test_images, test_labels))

# Evaluate the model
performance = model.evaluate(test_images, test_labels)
test_accuracy = {}
print('Test Loss:', performance[0])
print('Test Accuracy:', performance[1])
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Appendix F – VGG-19, Tiny ImageNet

```
# -*- coding: utf-8 -*-
"""vgg19_imageNet.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1FM-FpY7cqSmMEKxsskSkqOmx4awWYuei
"""

! git clone https://github.com/seshuad/IMagenet
! ls 'IMagenet/tiny-imagenet-200/'

import time
import imageio
import numpy as np

path = 'IMagenet/tiny-imagenet-200/'

def get_id_dictionary():
    id_dict = {}
    for i, line in enumerate(open( path + 'wnids.txt', 'r')):
        id_dict[line.replace('\n', '')] = i
    return id_dict
```

```
def get_class_to_id_dict():
    id_dict = get_id_dictionary()
    all_classes = {}
    result = {}
    for i, line in enumerate(open( path + 'words.txt', 'r')):
        n_id, word = line.split('\t')[:2]
        all_classes[n_id] = word
    for key, value in id_dict.items():
        result[value] = (key, all_classes[key])
    return result

def get_data(id_dict):
    print('starting loading data')
    train_data, test_data = [], []
    train_labels, test_labels = [], []
    t = time.time()
    for key, value in id_dict.items():
        train_data += [imageio.imread( path +
'train/{}/images/{}_{}.JPEG'.format(key, key, str(i)), mode='RGB') for i in
range(500)]
        train_labels_ = np.array([[0]*200]*500)
        train_labels_[:, value] = 1
        train_labels += train_labels_.tolist()

    for line in open( path + 'val/val_annotations.txt'):
        img_name, class_id = line.split('\t')[:2]
        test_data.append(imageio.imread( path +
'val/images/{}'.format(img_name) ,mode='RGB'))
        test_labels_ = np.array([[0]*200])
        test_labels_[0, id_dict[class_id]] = 1
        test_labels += test_labels_.tolist()

    print('finished loading data, in {} seconds'.format(time.time() - t))
    return np.array(train_data), np.array(train_labels), np.array(test_data),
np.array(test_labels)

train_data, train_labels, test_data, test_labels = get_data(get_id_dictionary())

print( "train data shape: ",  train_data.shape )
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```python
print( "train label shape: ", train_labels.shape )
print( "test data shape: ",   test_data.shape )
print( "test_labels.shape: ", test_labels.shape )

def shuffle_data(train_data, train_labels ):
    size = len(train_data)
    train_idx = np.arange(size)
    np.random.shuffle(train_idx)

    return train_data[train_idx], train_labels[train_idx]

train_data, train_labels = shuffle_data(train_data, train_labels)

# Keras, dataset, and VGG19 imports
import keras
from keras.datasets import cifar100, cifar10
from keras.applications import VGG19

# Loading VGG19 with imagenet weights
from keras.layers import Input

vgg19_model = VGG19(include_top = True, weights='imagenet')
vgg19_model.summary()

from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout

# define new empty model
model = Sequential()

# add all layers except output from VGG19 to new model
for layer in vgg19_model.layers[:-1]:
  model.add(layer)

# freeze all weights
for layer in model.layers:
  layer.trainable = False

# add dropout layer and new output layer
```

```
model.add(Dropout(0.5))
model.add(Dense(200, activation='softmax'))
model.summary()

# load dataset
#(x_train, y_train) , (x_val, y_val) = cifar10.load_data()

import numpy as np
import cv2

import matplotlib.pyplot as plt

NUM_CLASSES = 200

# Onehot encode labels

train_labels = keras.utils.to_categorical(train_labels, NUM_CLASSES)
test_labels = keras.utils.to_categorical(test_labels, NUM_CLASSES)

model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["categorical_accuracy"])

# returns batch_size random samples from either training set or validation set
# resizes each image to (224, 244, 3), the native input size for VGG19
def getBatch(batch_size, train_or_val='train'):
  x_batch = []
  y_batch = []
  if train_or_val == 'train':
    idx = np.random.randint(0, len(train_data), (batch_size))

    for i in idx:
      img = cv2.resize(train_data[i], (224, 224), interpolation=cv2.INTER_CUBIC)
      x_batch.append(img)
      y_batch.append(train_labels[i] if np.isscalar(train_labels[i]) else
train_labels[i][0])
  elif train_or_val == 'val':
    idx = np.random.randint(0, len(test_data), (batch_size))

    for i in idx:
      img = cv2.resize(test_data[i], (224, 224), interpolation=cv2.INTER_CUBIC)
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
      x_batch.append(img)
      y_batch.append(test_labels[i] if np.isscalar(test_labels[i]) else
test_labels[i][0])
  else:
    print("error, please specify train or val")

  x_batch = np.array(x_batch)
  y_batch = np.array(y_batch)
  #print(x_batch.shape)
  #print(y_batch.shape)
  return x_batch, y_batch




import pandas as pd
EPOCHS = 20
BATCH_SIZE = 128
VAL_SIZE = 500
STEPS = 50

df = pd.DataFrame(columns=['Epoch', 'Training Loss', 'Training Acc', 'Validation
Loss', 'Validation Acc'])




for e in range(EPOCHS):
  train_loss = 0
  train_acc = 0

  for s in range(STEPS):
    x_batch, y_batch = getBatch(BATCH_SIZE, "train")
    out = model.train_on_batch(x_batch, y_batch)
    train_loss += out[0]
    train_acc += out[1]

  print(f"Epoch: {e}\nTraining Loss = {train_loss / STEPS}\tTraining Acc =
{train_acc / STEPS}")

  x_v, y_v = getBatch(VAL_SIZE, "val")
  eval = model.evaluate(x_v, y_v)
  print(f"Validation loss: {eval[0]}\tValidation Acc: {eval[1]}\n")
  df.loc[len(df)] = [e, train_loss / STEPS, train_acc / STEPS, eval[0], eval[1]]


df.to_csv("vgg19_training_history_ImageNet.csv")

x_v, y_v = getBatch(VAL_SIZE, "val")
eval1 = model.evaluate(x_v, y_v)
print(f"Validation loss: {eval1[0]}\tValidation Acc: {eval1[1]}\n")
```

Appendix G – Custom Topology, CIFAR-100, Tiny ImageNet

```
# %% [markdown]
# Import Libraries

# %%
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG19, ResNet50
from tensorflow.keras.datasets import cifar10, cifar100
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import to_categorical
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
from tensorflow.keras.optimizers import Adam
from keras.callbacks import EarlyStopping
import pandas as pd

# %% [markdown]
# Load Custom Model From Filesystem

# %%
custom_model = load_model('Exp1_results/best_model/exp1_best_model.h5')
custom_model.summary()
custom_model.trainable = False

# %% [markdown]
# Get ImageNet Data

# %%
import time
import imageio
import numpy as np
path = 'IMagenet/tiny-imagenet-200/'

def get_id_dictionary():
    id_dict = {}
    for i, line in enumerate(open( path + 'wnids.txt', 'r')):
        id_dict[line.replace('\n', '')] = i
    return id_dict

def get_class_to_id_dict():
    id_dict = get_id_dictionary()
    all_classes = {}
    result = {}
    for i, line in enumerate(open( path + 'words.txt', 'r')):
        n_id, word = line.split('\t')[:2]
        all_classes[n_id] = word
    for key, value in id_dict.items():
        result[value] = (key, all_classes[key])
    return result

def get_imagenet_data(id_dict):
    print('starting loading data')
    train_data, test_data = [], []
    train_labels, test_labels = [], []
    t = time.time()
    for key, value in id_dict.items():
        train_data += [imageio.imread( path +
'train/{}/images/{}_{}.JPEG'.format(key, key, str(i)), mode='RGB') for i in
range(500)]
        train_labels_ = np.array([[0]*200]*500)
        train_labels_[:, value] = 1
        train_labels += train_labels_.tolist()

    for line in open( path + 'val/val_annotations.txt'):
        img_name, class_id = line.split('\t')[:2]
        test_data.append(imageio.imread( path +
'val/images/{}'.format(img_name) ,mode='RGB'))
        test_labels_ = np.array([[0]*200])
        test_labels_[0, id_dict[class_id]] = 1
        test_labels += test_labels_.tolist()

    print('finished loading data, in {} seconds'.format(time.time() - t))
    return np.array(train_data), np.array(train_labels), np.array(test_data),
np.array(test_labels)

# %% [markdown]
# Prepare Datasets

# %%

def prepare_data(dataset_name):
    x_train = None
    x_test = None
    y_train = None
    y_test = None
    # CIFAR-100 data preparation
    if dataset_name == 'CIFAR100':
        (x_train, y_train), (x_test, y_test) = cifar100.load_data()
        num_classes = 100
        x_train = x_train.astype('float32') / 255
        x_test = x_test.astype('float32') / 255
        y_train = to_categorical(y_train, num_classes)
        y_test = to_categorical(y_test, num_classes)
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    elif dataset_name == 'IMagenet':
        train_data, train_labels, test_data, test_labels = 
get_imagenet_data(get_id_dictionary())
        x_train = train_data.astype('float32') / 255
        x_test = test_data.astype('float32') / 255
        y_train = train_labels
        y_test = test_labels

    # Resizing images to 32x32
    x_train = tf.image.resize(x_train, (32, 32))
    x_test = tf.image.resize(x_test, (32, 32))

    return (x_train, y_train), (x_test, y_test)



# %% [markdown]
# Data Augmentation

# %%
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def data_augmentation(x_train):
    datagen = ImageDataGenerator(
        rotation_range=10,
        width_shift_range=0.05,
        height_shift_range=0.05,
        horizontal_flip=True,
        fill_mode='nearest',
        zoom_range=0.1
    )
    datagen.fit(x_train)
    return datagen

# %% [markdown]
# Train Model

# %%
def train_model(model, train_data, test_data, lr_schedule):
    x_train, y_train = train_data
    x_test, y_test = test_data
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    datagen = data_augmentation(x_train)
    model.compile(optimizer=Adam(learning_rate=0.01),
loss='categorical_crossentropy', metrics=['accuracy'])
    early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=10,
restore_best_weights=True)
    history = model.fit(datagen.flow(x_train, y_train, batch_size=64),
epochs=100, validation_data=(x_test, y_test), callbacks=[early_stopping,
lr_schedule])
    return history

# %% [markdown]
# Save Results

# %%
def save_results(history, dataset_name):
    results_df = pd.DataFrame(history.history)
    if dataset_name == "CIFAR100":
        base_path = 'Exp2_results/Custom/CIFAR100'
        results_df.to_csv(base_path + 'training_history.csv')
    else:
        base_path = 'Exp2_results/Custom/ImageNet'
        results_df.to_csv(base_path + 'training_history.csv')




# %% [markdown]
# Define List of Dataset Names

# %%
datasets = ['CIFAR100', 'IMageNet']

# %% [markdown]
# Modify Last Layer of Model for different datasets

# %%

def modify_model_for_classes(model, num_classes):
    # Explicitly define a new input layer that matches the shape of the original
model's input
    if not model.inputs:
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
        raise ValueError("Model does not have any inputs defined.")
    input_shape = model.inputs[0].shape[1:]  # Get shape, excluding the batch
size
    new_input = Input(shape=input_shape)

    # Rebuild the model from the new input using all layers except the last
    x = new_input
    for layer in model.layers[:-1]:  # Skip the last layer
        if not layer.__class__.__name__ == 'InputLayer':  # Skip the original
input layer
            # Ensure each layer has a unique name by appending a suffix
            layer_config = layer.get_config()
            layer_config['name'] = layer_config['name'] + '_reused'
            new_layer = layer.__class__.from_config(layer_config)
            x = new_layer(x)

    # Add a new Dense layer as the output layer with a unique name
    output = Dense(num_classes, activation='softmax', name='output_layer')(x)
    new_model = Model(inputs=new_input, outputs=output)
    return new_model




# %% [markdown]
# Main Training Loop

# %%
# Implementing a Learning Rate Scheduler
from tensorflow.keras.callbacks import LearningRateScheduler
from keras.optimizers import Adam

def scheduler(epoch, lr):
    if epoch < 10:
        return float(lr)
    else:
        return float(lr * tf.math.exp(-0.1))

lr_schedule = LearningRateScheduler(scheduler)

for dataset in datasets:
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    if dataset == 'CIFAR100':
        num_classes = 100
    elif dataset == 'IMageNet':
        num_classes = 200
    model = modify_model_for_classes(custom_model, num_classes)
    train_data, test_data = prepare_data(dataset)
    history = train_model(model, train_data, test_data, lr_schedule)
    save_results(history, dataset)
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024
Appendix H – Experiment 3, Part A

```
# %%
! git clone https://github.com/seshuad/IMagenet
! ls 'IMagenet/tiny-imagenet-200/'

# %%
from keras.models import load_model

model = load_model('exp1_best_model.h5')
model.summary()

# %%
import time
import imageio as nd
import numpy as np

path = 'IMagenet/tiny-imagenet-200/'

def get_id_dictionary():
    id_dict = {}
    for i, line in enumerate(open( path + 'wnids.txt', 'r')):
        id_dict[line.replace('\n', '')] = i
    return id_dict

def get_class_to_id_dict():
    id_dict = get_id_dictionary()
    all_classes = {}
    result = {}
    for i, line in enumerate(open( path + 'words.txt', 'r')):
        n_id, word = line.split('\t')[:2]
        all_classes[n_id] = word
    for key, value in id_dict.items():
        result[value] = (key, all_classes[key])
    return result

def get_data(id_dict):
    print('starting loading data')
    train_data, test_data = [], []
    train_labels, test_labels = [], []
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    t = time.time()
    for key, value in id_dict.items():
        train_data += [nd.imread( path + 'train/{}/images/{}_{}.JPEG'.format(key,
key, str(i)), mode='RGB') for i in range(500)]
        train_labels_ = np.array([[0]*200]*500)
        train_labels_[:, value] = 1
        train_labels += train_labels_.tolist()

    for line in open( path + 'val/val_annotations.txt'):
        img_name, class_id = line.split('\t')[:2]
        test_data.append(nd.imread( path +
'val/images/{}'.format(img_name) ,mode='RGB'))
        test_labels_ = np.array([[0]*200])
        test_labels_[0, id_dict[class_id]] = 1
        test_labels += test_labels_.tolist()

    print('finished loading data, in {} seconds'.format(time.time() - t))
    return np.array(train_data), np.array(train_labels), np.array(test_data),
np.array(test_labels)

train_data, train_labels, test_data, test_labels = get_data(get_id_dictionary())

print( "train data shape: ",  train_data.shape )
print( "train label shape: ", train_labels.shape )
print( "test data shape: ",   test_data.shape )
print( "test_labels.shape: ", test_labels.shape )

# %%
# Import necessary libraries
import tensorflow as tf
from keras.models import load_model
from keras.layers import Input, Dense, Flatten, Dropout
from keras.models import Model
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.callbacks import LearningRateScheduler




# Assume new input size for Tiny ImageNet
```

```
new_input = Input(shape=(64, 64, 3))

# Create a new input layer that matches the size of Tiny ImageNet images
input_shape = (64, 64, 3)
new_input = Input(shape=input_shape)

model.trainable = True
# Pass new_input through the layers of the old model, except for the output
layers
x = new_input
for layer in model.layers[:-3]:  # skip the last three layers (Flatten, Dense,
Dense)
    x = layer(x)

# Calculate the correct output dimensions after the last pooling layer
shape_before_flatten = x.shape[1]
print("Shape before flattening:", shape_before_flatten)



# Flatten and adjust dense layers
# x = Flatten()(x)
x = Dense(1024, activation='relu', input_shape=(shape_before_flatten,),
kernel_regularizer=l2(0.01))(x)  # Adjusted dense layer
x = Dropout(0.5)(x)
x = Dense(200, activation='softmax')(x)  # New output layer for 200 classes

# Create new model
new_model = Model(inputs=new_input, outputs=x)
new_model.summary()



# Learning rate scheduler function
def scheduler(epoch, lr):
    if epoch < 20:
        return lr
    else:
        return lr * np.exp(-0.1)

# Compile the new model
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```python
new_model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])




# %%
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping, ModelCheckpoint

early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1,
mode='min')
model_checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True,
monitor='val_loss', mode='min')

# Ensure the data is in the correct shape and datatype
train_data = train_data.astype('float32')
test_data = test_data.astype('float32')

# Normalize the data
train_data /= 255.0
test_data /= 255.0

# Verify shapes
print("Train data shape:", train_data.shape)
print("Test data shape:", test_data.shape)

# Create image data generators for data augmentation (optional but recommended)
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(
    rescale=1./255
)
```

```python
# Assuming the model is already compiled from your previous setup
# Define batch size and number of epochs
batch_size = 64
epochs = 50
# Callback for learning rate adjustment
lr_scheduler = LearningRateScheduler(scheduler)

# Train the model
history = new_model.fit(
    train_datagen.flow(train_data, train_labels, batch_size=batch_size),
    validation_data=test_datagen.flow(test_data, test_labels),
    epochs=epochs,
    steps_per_epoch=len(train_data) // batch_size,
    validation_steps=len(test_data) // batch_size,
    callbacks=[early_stopping, model_checkpoint, lr_scheduler]
)

# Optionally, save the trained model
new_model.save('trained_tiny_imagenet_model.h5')

# Evaluate the model
print(new_model.evaluate(test_datagen.flow(test_data, test_labels)))



# %%
import pandas as pd

pd.DataFrame(history.history).to_csv('trained_tiny_imagenet_history.csv')
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Appendix I – Experiment 4, Part A

```
# %%
!pip install -q transformers datasets evaluate

# %%
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import accuracy_score
from keras.utils import to_categorical

# Load CIFAR-10 dataset
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()

# %%
# Normalize data
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Convert class vectors to binary class matrices
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# %%
# import model from experiment 3
from keras.models import load_model
import tensorflow as tf
from keras.models import Model
from keras.layers import Input, Dense, Conv2D, BatchNormalization, MaxPooling2D,
Dropout, Flatten
from keras.regularizers import l2
from keras.callbacks import LearningRateScheduler
from keras.optimizers import Adam


imported_model = load_model('trained_tiny_imagenet_model.h5')

imported_model.summary()

# Create a new input layer that matches the size of Tiny ImageNet images
input_shape = (32, 32, 3)
new_input = Input(shape=input_shape)

# Pass new_input through the layers of the old model, except for the output
layers
x = new_input
for layer in imported_model.layers[:-3]:  # skip the last three layers (Flatten,
Dense, Dense)
    x = layer(x)

# Calculate the correct output dimensions after the last pooling layer
shape_before_flatten = x.shape[1]
print("Shape before flattening:", shape_before_flatten)

x = Dense(1024, activation='relu', input_shape=(shape_before_flatten,),
kernel_regularizer=l2(0.01))(x)  # Adjusted dense layer
x = Dropout(0.5)(x)
x = Dense(10, activation='softmax')(x)  # New output layer for 10 classes

# Learning rate scheduler function
def scheduler(epoch, lr):
    if epoch < 55:
        return lr
    else:
        return lr * np.exp(-0.1)

# Create new model
new_model = Model(inputs=new_input, outputs=x)
new_model.summary()









# %%
from keras.preprocessing.image import ImageDataGenerator
# data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=True,
    fill_mode='nearest',
    zoom_range=0.1
)
datagen.fit(X_train)

checkpoint = ModelCheckpoint('exp4_model.keras', save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=10,
restore_best_weights=True)

# %%
lr_schedule = LearningRateScheduler(scheduler)

# Compile the new model
new_model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])

history = new_model.fit(datagen.flow(X_train, Y_train, batch_size=64),
                     epochs=200,
                     validation_data=(X_test, Y_test),
                     callbacks=[early_stopping, checkpoint, lr_schedule])

# %%
model_scores = new_model.evaluate(X_test, Y_test, verbose=1)
print(f'Model Test accuracy: {model_scores[1]*100}%')

# %%
import pandas as pd

pd.DataFrame(history.history).to_csv('exp4_history.csv')
```

Appendix J – Experiment 1, Part B

```
# %% [markdown]
# Import Libraries

# %%
import numpy as np
import pandas as pd
import cv2

from sklearn.utils import shuffle
from matplotlib.patches import Rectangle
import matplotlib.pyplot as plt

import warnings

warnings.simplefilter('ignore')

# %% [markdown]
# Load Data
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# %%
!unzip '/content/drive/MyDrive/2024-
2025/Deep_Learning_Assignments/Datasets/Microsoft_COCO.v2-raw.tensorflow.zip'

# %%
df = pd.read_csv('/content/train/_annotations.csv')
df = shuffle(df)
df.head()

# %%
classes = df['class'].unique()
classes

# %%
 # Creating a dictionary with indices as keys
labels = {i + 1: class_id for i, class_id in enumerate(classes)}
print(labels)

# %% [markdown]
# Show some labeled images

# %%
# Get path images and boxes (x,y) for each class_id
boxes = {}
images = {}

base_path = '/content/train/'
for class_id in classes:
    first_row = df[df['class'] == class_id].iloc[0]
    images[class_id] = cv2.imread(base_path + first_row['filename'])
    boxes[class_id] =
[first_row['xmin'],first_row['xmax'],first_row['ymin'],first_row['ymax']]



# %%
for i in classes:
  # print(i)
  xmin, xmax, ymin, ymax = boxes[i][0], boxes[i][1], boxes[i][2], boxes[i][3]
  # print(f"boxes: {xmin}, {xmax}, {ymin}, {ymax}")
  for label in labels.values():
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    if label == i:
      plt.figure(figsize=(8, 10))
      plt.title("Label " + label)
      plt.imshow(images[i])
      plt.gca().add_patch(plt.Rectangle((xmin, ymin), xmax-xmin, ymax-ymin,
color='yellow', fill=False, linewidth=2))
      plt.show()



'''
for i in classes:
  xmin, xmax, ymin, ymax = boxes[i][0], boxes[i][1], boxes[i][2], boxes[i][3]
  plt.figure(figsize=(8, 10))
  plt.title("Label " + labels[i])
  plt.imshow(images[i])
  plt.gca().add_patch(plt.Rectangle((xmin, ymin), xmax-xmin, ymax-ymin,
color='yellow', fill=False, linewidth=2))
  plt.show()
'''


# %% [markdown]
# Model

# %%
!pip install ultralytics

# %%
from ultralytics import YOLO
from PIL import Image
from IPython.display import display
import os
import pathlib

# %%
model = YOLO("yolov8m.pt")

# %%
def calculate_iou(box1, box2):
    """
    Calculate the Intersection over Union (IoU) of two bounding boxes.
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
    Parameters:
    box1 -- first box, list or tuple (xmin, ymin, xmax, ymax)
    box2 -- second box, list or tuple (xmin, ymin, xmax, ymax)

    Returns:
    float -- IoU of box1 and box2
    """
    x_left = max(box1[0], box2[0])
    y_top = max(box1[1], box2[1])
    x_right = min(box1[2], box2[2])
    y_bottom = min(box1[3], box2[3])

    if x_right < x_left or y_bottom < y_top:
        return 0.0  # No overlap

    intersection_area = (x_right - x_left) * (y_bottom - y_top)
    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])
    union_area = box1_area + box2_area - intersection_area
    return intersection_area / union_area



# %%
base_img_path = '/content/train/'

images = {'image1' : '000000009465_jpg.rf.3d5a6b94f8c1afdc004f2641cd578912.jpg',
          'image2' : '000000432604_jpg.rf.c09c7a2273915bf0fe46ffef4fac09c6.jpg',
          'image3': '000000559907_jpg.rf.1888a00adfb5c7113de8b3f5e92a14a5.jpg'}

ground_truth_boxes = {key: [] for key in images.keys()}

# Extract ground truth bounding boxes for each image
for key, filepath in images.items():
    # Filter rows where filename matches
    filtered_df = df[df['filename'] == filepath]
    for _, row in filtered_df.iterrows():
        box = {
            row['class'] : [row['xmin'], row['ymin'], row['xmax'], row['ymax']]
        }
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
        ground_truth_boxes[key].append(box)




predicted_boxes = {key: [] for key in images.keys()}  # Initialize an empty list
for each image

for image_name, image_path in images.items():
    results = model.predict(source=base_img_path + image_path, save=True,
conf=0.2, iou=0.5)
    for result in results:
        boxes = result.boxes  # Boxes object for bbox outputs
        for box in boxes:
            class_id = result.names[box.cls[0].item()]  # Get the class id
            cords = box.xyxy[0].tolist()  # Get the coordinates
            cords = [round(x) for x in cords]  # Round the coordinates
            predicted_boxes[image_name].append({class_id: cords})  # Append the
bounding box to the list for the image




print("Predicted Boxes:")
for image_name, boxes in predicted_boxes.items():
    for box in boxes:
        for class_id, cords in box.items():
            print(f"Image: {image_name}, Class: {class_id}, Coordinates:
{cords}")

print("Ground Truth Boxes:")
for image_name, boxes in ground_truth_boxes.items():
    for box in boxes:
        for class_id, cords in box.items():
          print(f"Image: {image_name}, Class: {class_id}, Coordinates: {cords}")



iou_results = []

for image_key in predicted_boxes:
    for pred_dict in predicted_boxes[image_key]:
        for pred_class, pred_coords in pred_dict.items():
            best_iou = 0
            best_gt_coords = None
            for gt_dict in ground_truth_boxes[image_key]:
                for gt_class, gt_coords in gt_dict.items():
                    if pred_class == gt_class:
                        iou = calculate_iou(pred_coords, gt_coords)
                        if iou > best_iou:
                            best_iou = iou
                            best_gt_coords = gt_coords
            if best_iou > 0:  # To record only matches found
                iou_results.append({
                    'image': image_key,
                    'predicted_class': pred_class,
                    'predicted_coords': pred_coords,
                    'ground_truth_coords': best_gt_coords,
                    'IoU': best_iou
                })

# Create DataFrame and save to CSV
results_df = pd.DataFrame(iou_results)
results_df.to_csv('iou_results.csv', index=False)
print("IoU results saved to 'iou_results.csv'.")
```

Appendix K – Experiment 2, Part B

```
# %%
!pip install -q transformers datasets evaluate

# %% [markdown]
# Download Toy Dataset

# %%
import requests, zipfile, io
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
def download_data():
    url = "https://www.dropbox.com/s/l1e45oht447053f/ADE20k_toy_dataset.zip?dl=1"
    r = requests.get(url)
    z = zipfile.ZipFile(io.BytesIO(r.content))
    z.extractall()

download_data()

# %% [markdown]
# Define Pytorch Dataset and Dataloaders

# %%
from torch.utils.data import Dataset
import os
from PIL import Image

class SemanticSegmentationDataset(Dataset):
    """Image (semantic) segmentation dataset."""

    def __init__(self, root_dir, image_processor, train=True):
        """
        Args:
            root_dir (string): Root directory of the dataset containing the
images + annotations.
            image_processor (SegFormerImageProcessor): image processor to prepare
images + segmentation maps.
            train (bool): Whether to load "training" or "validation" images +
annotations.
        """
        self.root_dir = root_dir
        self.image_processor = image_processor
        self.train = train

        sub_path = "training" if self.train else "validation"
        self.img_dir = os.path.join(self.root_dir, "images", sub_path)
        self.ann_dir = os.path.join(self.root_dir, "annotations", sub_path)

        # read images
        image_file_names = []
        for root, dirs, files in os.walk(self.img_dir):
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
          image_file_names.extend(files)
        self.images = sorted(image_file_names)

        # read annotations
        annotation_file_names = []
        for root, dirs, files in os.walk(self.ann_dir):
          annotation_file_names.extend(files)
        self.annotations = sorted(annotation_file_names)

        assert len(self.images) == len(self.annotations), "There must be as many 
images as there are segmentation maps"

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):

        image = Image.open(os.path.join(self.img_dir, self.images[idx]))
        segmentation_map = Image.open(os.path.join(self.ann_dir, 
self.annotations[idx]))

        # randomly crop + pad both image and segmentation map to same size
        encoded_inputs = self.image_processor(image, segmentation_map, 
return_tensors="pt")

        for k,v in encoded_inputs.items():
          encoded_inputs[k].squeeze_() # remove batch dimension

        return encoded_inputs

# %% [markdown]
# Initialize training and validation datasets

# %%
from transformers import SegformerImageProcessor

root_dir = '/content/ADE20k_toy_dataset'
image_processor = SegformerImageProcessor(reduce_labels=True)
```

```
train_dataset = SemanticSegmentationDataset(root_dir=root_dir,
image_processor=image_processor)
valid_dataset = SemanticSegmentationDataset(root_dir=root_dir,
image_processor=image_processor, train=False)

# %%
print("Number of training examples:", len(train_dataset))
print("Number of validation examples:", len(valid_dataset))

# %% [markdown]
# Verify random sample

# %%
encoded_inputs = train_dataset[0]

# %%
encoded_inputs["pixel_values"].shape

# %%
encoded_inputs["labels"].shape

# %%
encoded_inputs["labels"]

# %%
encoded_inputs["labels"].squeeze().unique()

# %% [markdown]
# Define corresponding dataloaders

# %%
from torch.utils.data import DataLoader

train_dataloader = DataLoader(train_dataset, batch_size=2, shuffle=True)
valid_dataloader = DataLoader(valid_dataset, batch_size=2)

# %%
batch = next(iter(train_dataloader))
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# %%
for k,v in batch.items():
  print(k, v.shape)

# %%
batch["labels"].shape

# %%
mask = (batch["labels"] != 255)
mask

# %%
batch["labels"][mask]

# %% [markdown]
# Define the model

# %%
from transformers import SegformerForSemanticSegmentation
import json
from huggingface_hub import hf_hub_download

# load id2label mapping from a JSON on the hub
repo_id = "huggingface/label-files"
filename = "ade20k-id2label.json"
id2label = json.load(open(hf_hub_download(repo_id=repo_id, filename=filename,
repo_type="dataset"), "r"))
id2label = {int(k): v for k, v in id2label.items()}
label2id = {v: k for k, v in id2label.items()}

# define model
model = SegformerForSemanticSegmentation.from_pretrained("nvidia/mit-b0",
                                                         num_labels=150,
                                                         id2label=id2label,
                                                         label2id=label2id,
)

# %%
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# %% [markdown]
# Fine-tune the model

# %%
import evaluate

metric = evaluate.load("mean_iou")

# %%
image_processor.do_reduce_labels

# %%
import torch
from torch import nn
from sklearn.metrics import accuracy_score
from tqdm.notebook import tqdm
import pandas as pd

# define optimizer
optimizer = torch.optim.AdamW(model.parameters(), lr=0.00006)
# move model to GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

model.train()

results = []  # Initialize a list to store results for each logging step
for epoch in range(200):  # loop over the dataset multiple times
   print("Epoch:", epoch)
   for idx, batch in enumerate(tqdm(train_dataloader)):
        # get the inputs;
        pixel_values = batch["pixel_values"].to(device)
        labels = batch["labels"].to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(pixel_values=pixel_values, labels=labels)
        loss, logits = outputs.loss, outputs.logits
        loss.backward()
        optimizer.step()

        # evaluate
        with torch.no_grad():
          upsampled_logits = nn.functional.interpolate(logits,
size=labels.shape[-2:], mode="bilinear", align_corners=False)
          predicted = upsampled_logits.argmax(dim=1)

          # note that the metric expects predictions + labels as numpy arrays
          metric.add_batch(predictions=predicted.detach().cpu().numpy(),
references=labels.detach().cpu().numpy())

        # let's print loss and metrics every 100 batches
        if idx % 100 == 0:
          # currently using _compute instead of compute
          # see this issue for more info:
https://github.com/huggingface/evaluate/pull/328#issuecomment-1286866576
          metrics = metric._compute(
                  predictions=predicted.cpu(),
                  references=labels.cpu(),
                  num_labels=len(id2label),
                  ignore_index=255,
                  reduce_labels=False, # we've already reduced the labels
ourselves
              )

          print("Loss:", loss.item())
          print("Mean_iou:", metrics["mean_iou"])
          print("Mean accuracy:", metrics["mean_accuracy"])
          # Append current results to the list
          results.append({
              "epoch": epoch,
              "batch_idx": idx,
              "loss": loss.item(),
              "mean_iou": metrics["mean_iou"],
              "mean_accuracy": metrics["mean_accuracy"]
          })
```

```
# After training, convert the results list to a DataFrame and save it to CSV
df = pd.DataFrame(results)
df.to_csv('training_results.csv', index=False)

# %% [markdown]
# Inference

# %%
image =
Image.open('/content/ADE20k_toy_dataset/images/training/ADE_train_00000002.jpg')
image

# %%
# prepare the image for the model
pixel_values = image_processor(image,
return_tensors="pt").pixel_values.to(device)
print(pixel_values.shape)

# %%
import torch

# forward pass
with torch.no_grad():
  outputs = model(pixel_values=pixel_values)

# %%
# logits are of shape (batch_size, num_labels, height/4, width/4)
logits = outputs.logits.cpu()
print(logits.shape)

# %%
def ade_palette():
    """ADE20K palette that maps each class to RGB values."""
    return [[120, 120, 120], [180, 120, 120], [6, 230, 230], [80, 50, 50],
            [4, 200, 3], [120, 120, 80], [140, 140, 140], [204, 5, 255],
            [230, 230, 230], [4, 250, 7], [224, 5, 255], [235, 255, 7],
            [150, 5, 61], [120, 120, 70], [8, 255, 51], [255, 6, 82],
            [143, 255, 140], [204, 255, 4], [255, 51, 7], [204, 70, 3],
            [0, 102, 200], [61, 230, 250], [255, 6, 51], [11, 102, 255],
            [255, 7, 71], [255, 9, 224], [9, 7, 230], [220, 220, 220],
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
            [255, 9, 92], [112, 9, 255], [8, 255, 214], [7, 255, 224],
            [255, 184, 6], [10, 255, 71], [255, 41, 10], [7, 255, 255],
            [224, 255, 8], [102, 8, 255], [255, 61, 6], [255, 194, 7],
            [255, 122, 8], [0, 255, 20], [255, 8, 41], [255, 5, 153],
            [6, 51, 255], [235, 12, 255], [160, 150, 20], [0, 163, 255],
            [140, 140, 140], [250, 10, 15], [20, 255, 0], [31, 255, 0],
            [255, 31, 0], [255, 224, 0], [153, 255, 0], [0, 0, 255],
            [255, 71, 0], [0, 235, 255], [0, 173, 255], [31, 0, 255],
            [11, 200, 200], [255, 82, 0], [0, 255, 245], [0, 61, 255],
            [0, 255, 112], [0, 255, 133], [255, 0, 0], [255, 163, 0],
            [255, 102, 0], [194, 255, 0], [0, 143, 255], [51, 255, 0],
            [0, 82, 255], [0, 255, 41], [0, 255, 173], [10, 0, 255],
            [173, 255, 0], [0, 255, 153], [255, 92, 0], [255, 0, 255],
            [255, 0, 245], [255, 0, 102], [255, 173, 0], [255, 0, 20],
            [255, 184, 184], [0, 31, 255], [0, 255, 61], [0, 71, 255],
            [255, 0, 204], [0, 255, 194], [0, 255, 82], [0, 10, 255],
            [0, 112, 255], [51, 0, 255], [0, 194, 255], [0, 122, 255],
            [0, 255, 163], [255, 153, 0], [0, 255, 10], [255, 112, 0],
            [143, 255, 0], [82, 0, 255], [163, 255, 0], [255, 235, 0],
            [8, 184, 170], [133, 0, 255], [0, 255, 92], [184, 0, 255],
            [255, 0, 31], [0, 184, 255], [0, 214, 255], [255, 0, 112],
            [92, 255, 0], [0, 224, 255], [112, 224, 255], [70, 184, 160],
            [163, 0, 255], [153, 0, 255], [71, 255, 0], [255, 0, 163],
            [255, 204, 0], [255, 0, 143], [0, 255, 235], [133, 255, 0],
            [255, 0, 235], [245, 0, 255], [255, 0, 122], [255, 245, 0],
            [10, 190, 212], [214, 255, 0], [0, 204, 255], [20, 0, 255],
            [255, 255, 0], [0, 153, 255], [0, 41, 255], [0, 255, 204],
            [41, 0, 255], [41, 255, 0], [173, 0, 255], [0, 245, 255],
            [71, 0, 255], [122, 0, 255], [0, 255, 184], [0, 92, 255],
            [184, 255, 0], [0, 133, 255], [255, 214, 0], [25, 194, 194],
            [102, 255, 0], [92, 0, 255]]


# %%
predicted_segmentation_map =
image_processor.post_process_semantic_segmentation(outputs,
target_sizes=[image.size[::-1]])[0]
predicted_segmentation_map = predicted_segmentation_map.cpu().numpy()
print(predicted_segmentation_map)


# %%
import matplotlib.pyplot as plt
import numpy as np

color_seg = np.zeros((predicted_segmentation_map.shape[0],
                      predicted_segmentation_map.shape[1], 3), dtype=np.uint8) #
height, width, 3

palette = np.array(ade_palette())
for label, color in enumerate(palette):
    color_seg[predicted_segmentation_map == label, :] = color
# Convert to BGR
color_seg = color_seg[..., ::-1]

# Show image + mask
img = np.array(image) * 0.5 + color_seg * 0.5
img = img.astype(np.uint8)

plt.figure(figsize=(15, 10))
plt.imshow(img)
plt.show()

# %% [markdown]
# Compare to the ground truth

# %%
map =
Image.open('/content/ADE20k_toy_dataset/annotations/training/ADE_train_00000002.p
ng')
map

# %%
# convert map to NumPy array
import cv2
map = np.array(map)
map[map == 0] = 255 # background class is replaced by ignore_index
map = map - 1 # other classes are reduced by one
map[map == 254] = 255

classes_map = np.unique(map).tolist()
unique_classes = [model.config.id2label[idx] if idx!=255 else None for idx in
classes_map]
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
print("Classes in this image:", unique_classes)

# create coloured map
color_seg = np.zeros((map.shape[0], map.shape[1], 3), dtype=np.uint8) # height,
width, 3
palette = np.array(ade_palette())
for label, color in enumerate(palette):
    color_seg[map == label, :] = color
# Convert to BGR
color_seg = cv2.resize(color_seg, (img.shape[1], img.shape[0]))

# Show image + mask
img = np.array(image) * 0.5 + color_seg * 0.5
img = img.astype(np.uint8)

plt.figure(figsize=(15, 10))
plt.imshow(img)
plt.show()

# %%
# metric expects a list of numpy arrays for both predictions and references
metrics = metric._compute(
                  predictions=[predicted_segmentation_map],
                  references=[map],
                  num_labels=len(id2label),
                  ignore_index=255,
                  reduce_labels=False, # we've already reduced the labels
ourselves
              )

# %%
metrics.keys()

# %%
import pandas as pd

# print overall metrics
for key in list(metrics.keys())[:3]:
  print(key, metrics[key])
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# pretty-print per category metrics as Pandas DataFrame
metric_table = dict()
for id, label in id2label.items():
    metric_table[label] = [
                           metrics["per_category_iou"][id],
                           metrics["per_category_accuracy"][id]
    ]

print("---------------------")
print("per-category metrics:")
pd.DataFrame.from_dict(metric_table, orient="index", columns=["IoU", "accuracy"])

# %%
metric_table

# %%
import torch
from torch.utils.data import DataLoader, Subset
import random
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from torchvision.transforms import functional as F
import os
from torchvision import transforms
import pandas as pd

# Assuming 'train_dataset' is your PyTorch dataset for validation/testing
# Randomly sample 3 indices
indices = random.sample(range(len(train_dataset)), 3)

# Create a subset and corresponding DataLoader for these indices
sample_dataset = Subset(train_dataset, indices)
sample_dataloader = DataLoader(sample_dataset, batch_size=1, shuffle=False)

# Set the device to CUDA if available, otherwise use CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Ensure your model is on the correct device
model = model.to(device)
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# Function to decode segmentation maps into colored images
def decode_segmap(image, nc=150):
    palette = np.array(ade_palette())  # Define your palette function
appropriately
    color_seg = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)
    for label, color in enumerate(palette):
        color_seg[image == label, :] = color
    return color_seg[..., ::-1]  # Convert RGB to BGR if needed

# Metric initialization and update functions
def init_metrics(num_classes):
    return {
        'class_correct': np.zeros(num_classes),
        'class_total': np.zeros(num_classes),
        'class_intersection': np.zeros(num_classes),
        'class_union': np.zeros(num_classes)
    }

def update_metrics(metrics, predictions, labels, num_classes):
    for cls in range(num_classes):
        class_mask = (labels == cls)
        pred_mask = (predictions == cls)
        metrics['class_correct'][cls] += np.sum((predictions == labels) &
class_mask)
        metrics['class_total'][cls] += np.sum(class_mask)
        metrics['class_intersection'][cls] += np.sum(pred_mask & class_mask)
        metrics['class_union'][cls] += np.sum(pred_mask | class_mask)

def finalize_metrics(metrics, num_classes):
    per_class_accuracy = np.divide(metrics['class_correct'],
metrics['class_total'], out=np.zeros_like(metrics['class_correct']),
where=metrics['class_total']!=0)
    per_class_iou = np.divide(metrics['class_intersection'],
metrics['class_union'], out=np.zeros_like(metrics['class_intersection']),
where=metrics['class_union']!=0)
    return per_class_accuracy, per_class_iou

def calc_mean_iou(per_class_iou):
    valid_iou = per_class_iou[~np.isnan(per_class_iou)]
    if valid_iou.size == 0:
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
        return float('nan')  # Return nan if no valid IoU values are present
    return np.mean(valid_iou)

from collections import defaultdict

def calculate_metrics(predictions, labels, num_classes):
    metrics = init_metrics(num_classes)
    update_metrics(metrics, predictions, labels, num_classes)
    per_class_accuracy, per_class_iou = finalize_metrics(metrics, num_classes)
    return per_class_accuracy, per_class_iou

# Directory to save images corresponding to accuracy and IoU
image_dir = '/content/results/images_with_metrics'
os.makedirs(image_dir, exist_ok=True)

# Processing loop
data_list = []
num_classes = 150  # Update this based on your model's output
metrics = init_metrics(num_classes)
model.eval()

with torch.no_grad():
    for i, batch in enumerate(sample_dataloader):
        images = batch['pixel_values'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(images)
        logits = outputs.logits
        probs = torch.softmax(logits, dim=1)
        predictions = torch.argmax(probs, dim=1).cpu().numpy()

        for j, (image, prediction, label) in enumerate(zip(images.cpu(),
predictions, labels.cpu())):
            data_entry = {}
            prediction_resized =
F.resize(torch.from_numpy(prediction).unsqueeze(0).unsqueeze(0),
                                          size=(label.shape[-2], label.shape[-
1]),

interpolation=F.InterpolationMode.NEAREST).squeeze().numpy()
```

```
            acc = (prediction_resized == label.numpy()).mean()
            # Calculate per-class metrics for the current image
            per_class_accuracy, per_class_iou =
calculate_metrics(prediction_resized, label.numpy(), num_classes)
            # calculate mean iou
            mean_iou = calc_mean_iou(per_class_iou)

            pred_img = decode_segmap(prediction_resized)
            plt.figure(figsize=(10, 5))
            plt.subplot(1, 2, 1)
            plt.imshow(np.transpose(image.numpy(), (1, 2, 0)))
            plt.title('Original Image')
            plt.subplot(1, 2, 2)
            plt.imshow(pred_img)
            plt.title('Predicted Segmentation')
            image_path = os.path.join(image_dir, f'sample_image_{i}_{j}.png')
            plt.savefig(image_path)
            plt.close()
            data_entry['Accuracy'] = acc
            data_entry['Mean IoU'] = mean_iou
            data_entry['Image Path'] = image_path
            # Adding per-class metrics
            for cls in range(num_classes):
              class_name = id2label[cls]
              data_entry[f'{class_name} Accuracy'] = per_class_accuracy[cls]
              data_entry[f'{class_name} IoU'] = per_class_iou[cls]
            data_list.append(data_entry)
df = pd.DataFrame(data_list)
df.to_csv('/content/results/metrics_with_images_and_classes.csv', index=False)



# %%
# code to generate heatmaps

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
```

```
file_path = '/content/results/metrics_with_images_and_classes.csv'  # Replace
with the path to your CSV file
data = pd.read_csv(file_path)

# Extract columns for Accuracy and IoU
accuracy_columns = [col for col in data.columns if 'Accuracy' in col]
iou_columns = [col for col in data.columns if 'IoU' in col]

# Create heatmaps for Accuracy and IoU
fig, axes = plt.subplots(2, 1, figsize=(20, 20))

# Accuracy Heatmap
accuracy_data = data[accuracy_columns]
sns.heatmap(accuracy_data.T, ax=axes[0], cmap='viridis', cbar_kws={'label':
'Accuracy'})
axes[0].set_title('Accuracy across All Classes')
axes[0].set_xticklabels([])
axes[0].set_yticklabels([col.replace(' Accuracy', '') for col in
accuracy_columns], rotation=0)

# IoU Heatmap
iou_data = data[iou_columns]
sns.heatmap(iou_data.T, ax=axes[1], cmap='viridis', cbar_kws={'label': 'IoU'})
axes[1].set_title('IoU across All Classes')
axes[1].set_xticklabels([])
axes[1].set_yticklabels([col.replace(' IoU', '') for col in iou_columns],
rotation=0)

plt.tight_layout()
plt.show()

# Analysis output (You might want to customize this part based on specific needs
or insights you gain from the heatmaps)
print("Heatmap analysis complete. Check the visual output for detailed insights
into class performance.")
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

Appendix L – Generative Models

```python
# %%
def preprocess_image(image_path):
    # Util function to open, resize and format pictures into appropriate tensors
    img = keras.utils.load_img(image_path, target_size=(img_nrows, img_ncols))
    img = keras.utils.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = vgg19.preprocess_input(img)
    return tf.convert_to_tensor(img)



def deprocess_image(x):
    # Util function to convert a tensor into a valid image
    x = x.reshape((img_nrows, img_ncols, 3))
    # Remove zero-center by mean pixel
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    # 'BGR'->'RGB'
    x = x[:, :, ::-1]
    x = np.clip(x, 0, 255).astype("uint8")
    return x
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# %%
# The gram matrix of an image tensor (feature-wise outer product)


def gram_matrix(x):
    x = tf.transpose(x, (2, 0, 1))
    features = tf.reshape(x, (tf.shape(x)[0], -1))
    gram = tf.matmul(features, tf.transpose(features))
    return gram


# The "style loss" is designed to maintain
# the style of the reference image in the generated image.
# It is based on the gram matrices (which capture style) of
# feature maps from the style reference image
# and from the generated image


def style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
    return tf.reduce_sum(tf.square(S - C)) / (4.0 * (channels**2) * (size**2))


# An auxiliary loss function
# designed to maintain the "content" of the
# base image in the generated image


def content_loss(base, combination):
    return tf.reduce_sum(tf.square(combination - base))


# The 3rd loss function, total variation loss,
# designed to keep the generated image locally coherent
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
def total_variation_loss(x):
    a = tf.square(
        x[:, : img_nrows - 1, : img_ncols - 1, :] - x[:, 1:, : img_ncols - 1, :]
    )
    b = tf.square(
        x[:, : img_nrows - 1, : img_ncols - 1, :] - x[:, : img_nrows - 1, 1:, :]
    )
    return tf.reduce_sum(tf.pow(a + b, 1.25))



# %%
# Build a VGG19 model loaded with pre-trained ImageNet weights
model = vgg19.VGG19(weights="imagenet", include_top=False)

# Get the symbolic outputs of each "key" layer (we gave them unique names).
outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])

# Set up a model that returns the activation values for every layer in
# VGG19 (as a dict).
feature_extractor = keras.Model(inputs=model.inputs, outputs=outputs_dict)

# %%
# List of layers to use for the style loss.
style_layer_names = [
    "block1_conv1",
    "block2_conv1",
    "block3_conv1",
    "block4_conv1",
    "block5_conv1",
]
# The layer to use for the content loss.
content_layer_name = "block5_conv2"



def compute_loss(combination_image, base_image, style_reference_image):
    input_tensor = tf.concat(
        [base_image, style_reference_image, combination_image], axis=0
    )
    features = feature_extractor(input_tensor)

    # Initialize the loss
    loss = tf.zeros(shape=())

    # Add content loss
    layer_features = features[content_layer_name]
    base_image_features = layer_features[0, :, :, :]
    combination_features = layer_features[2, :, :, :]
    loss = loss + content_weight * content_loss(
        base_image_features, combination_features
    )
    # Add style loss
    for layer_name in style_layer_names:
        layer_features = features[layer_name]
        style_reference_features = layer_features[1, :, :, :]
        combination_features = layer_features[2, :, :, :]
        sl = style_loss(style_reference_features, combination_features)
        loss += (style_weight / len(style_layer_names)) * sl

    # Add total variation loss
    loss += total_variation_weight * total_variation_loss(combination_image)
    return loss



# %%
@tf.function
def compute_loss_and_grads(combination_image, base_image, style_reference_image):
    with tf.GradientTape() as tape:
        loss = compute_loss(combination_image, base_image, style_reference_image)
    grads = tape.gradient(loss, combination_image)
    return loss, grads

# %%
import os

os.environ["KERAS_BACKEND"] = "tensorflow"

import keras
import numpy as np
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
import tensorflow as tf
from keras.applications import vgg19
from PIL import Image

base_image_path = '/content/drive/MyDrive/2024-
2025/Deep_Learning_Assignments/Datasets/custom_images/'
style_ref_path = '/content/style_refs/'
images = {'image1': base_image_path + 'image1_me_and_sis.jpg',
          'image2': base_image_path + 'image2_me_dad_sis.jpg',
          'image3': base_image_path + 'image3_iguazu_falls.jpg',
          'image4': base_image_path + 'image4_purple_flowers.jpg',
          'image5': base_image_path + 'image5_mural.jpg',
          'image6': base_image_path + 'image6_cake.jpg',
          'image7': base_image_path + 'image7_lucky_dog.jpg'}
style_refs = {'image1': style_ref_path + 'scream.jpg',
              'image2': style_ref_path + 'scream.jpg',
              'image3': style_ref_path + 'starry_night.jpg',
              'image4': style_ref_path + 'wave.jpg',
              'image5': style_ref_path + 'starry_night.jpg',
              'image6': style_ref_path + 'colorful.jpg',
              'image7': style_ref_path + 'black_white.jpg'}

for image_name, image_path in images.items():
  for style_ref_name, style_ref_path in style_refs.items():
    if image_name == style_ref_name and ( image_name == 'image7'):
      style_reference_image_path = style_ref_path
      result_prefix = image_name + '_generated'
      total_variation_weight = 1e-6
      style_weight = 1e-6
      content_weight = 2.5e-8
      width, height = keras.utils.load_img(images[image_name]).size
      img_nrows = 400
      img_ncols = int(width * img_nrows / height)
      img_path = images[image_name]
      img = Image.open(img_path)
      style_image = Image.open(style_reference_image_path)
      img_resized = img.resize(style_image.size)
      img_resized.save(img_path, 'JPEG')
      optimizer = keras.optimizers.SGD(
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=100.0,
decay_steps=100, decay_rate=0.96)
          )
      base_image = preprocess_image(img_path)
      style_reference_image = preprocess_image(style_reference_image_path)
      combination_image = tf.Variable(preprocess_image(img_path))
      iterations = 4000
      for i in range(1, iterations + 1):
        loss, grads = compute_loss_and_grads(
          combination_image, base_image, style_reference_image
        )
        optimizer.apply_gradients([(grads, combination_image)])
        if i % 100 == 0:
          print("Iteration %d: loss=%.2f" % (i, loss))
          img = deprocess_image(combination_image.numpy())
          fname = result_prefix + "_at_iteration_%d.png" % i

keras.utils.save_img(f'/content/results/{image_name}/results_during_optimization/
' + fname, img)









# %%
import zipfile
import os

def zipdir(path, ziph):
    # ziph is zipfile handle
    for root, dirs, files in os.walk(path):
        for file in files:
            ziph.write(os.path.join(root, file),
                       os.path.relpath(os.path.join(root, file),
                                       os.path.join(path, '..')))
```

Ashna Ali
COMP-SCI 5567-0001
Final Report
5/11/2024

```
# Specifying the directory
results_directory = '/content/results'
style_refs_directory = '/content/style_refs'


# Writing files to a zipfile
with zipfile.ZipFile('results.zip', 'w', zipfile.ZIP_DEFLATED) as zipf:
    zipdir(results_directory, zipf)
with zipfile.ZipFile('style_refs.zip', 'w', zipfile.ZIP_DEFLATED) as zipf:
    zipdir(style_refs_directory, zipf)
```