Shreyas Kapoor | Neural Nets and Deep Learning
Lab Assignment 4

Validation Set

Method

The assignment was done on Google colab with GPU turned on. The dataset was a ViZWiz dataset which had 20523 images in training set, 4319 in validation set and 8000 in test set. Each image also featured a question and an answer associated with it. For the assignment, I used training images from 17000 to 18999, images 1000 to 1600 for validation and 1000 images for testing.

The first step was to import all the necessary libraries required for this assignment. After that, the annotation files containing the images were downloaded from https://vizwiz.cs.colorado.edu//VizWiz_visualization_img/. Training, validation, and test annotation files were taken out of it. After that, I wrote functions to extract features from images and question. To extract features from images, I used the VGG16 model with its default parameters. In the function, the image was extracted from the dataset, fed into the model which predicted its features, and it was returned from the function. For the question part, BERT based tokenizer was used with its default parameters. For this function, tokenizer extracted the features of the question and returned it. The next part was to iterate the for loop for 2000 images, call the above-mentioned functions and extract the features, concatenate the features into a numpy array and add the features to another numpy array. The string answers were converted into integer values using the LabelEncoder library.

The first model that I created consisted of 2 layers of Dense layers with input as the number of features in the concatenated array and output as the number of classes identified by the LabelEncoder. For activation function, I used sigmoid. While compiling, I used Adam as the optimizer and sparce cross entropy as the loss function. For the second model, I used 3 Dense layers with optimizer as RMSprop. For the third model, I used 4 Dense layers and Adagrad as the optimizer. The number of epochs that I used were 30 epochs.

The next part was to create the validation set. For this, I used the same code block that I used for training dataset to create the features. After that, I fed the features to all the three models and calculated the average precision value from all the models.

The next part was to create the test set of 1000 images. Again, I used the same code block to extract features from images and question. After this, the features were fed to the third model as it had the highest accuracy. Results were calculated and then saved into the results.csv file.

Results

The following graphs shows the loss functions with respect to number of epochs for each of the model

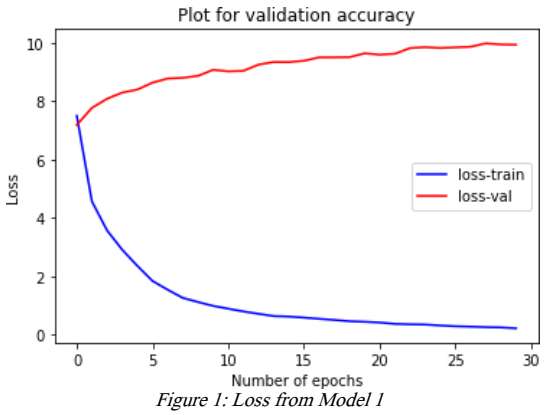

Figure 1: Loss from Model 1

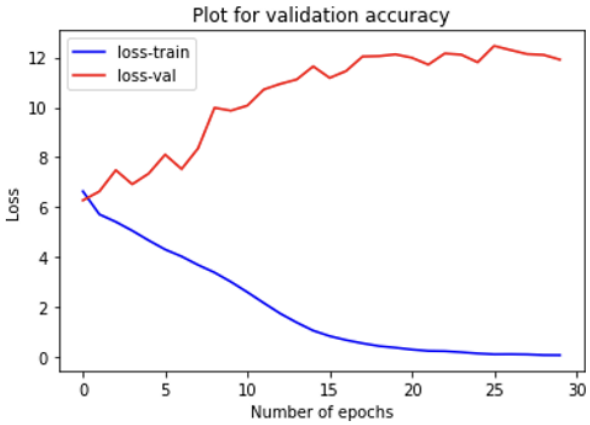

Figure 2: Loss from Model 2

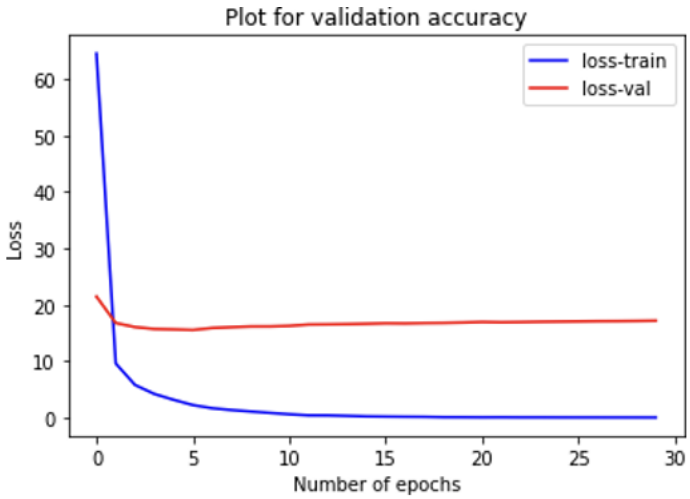

Figure 3: Loss from Model 3

The following graphs shows the Accuracy functions with respect to number of epochs for each of the model

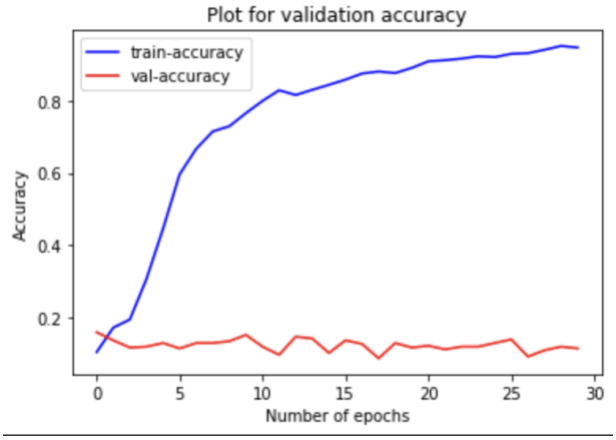

Figure 4: Accuracy from Model 1

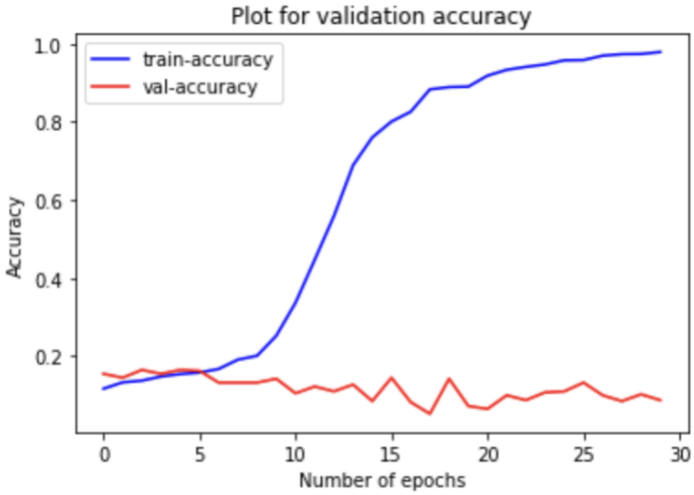

Figure 5: Accuracy from Model 2

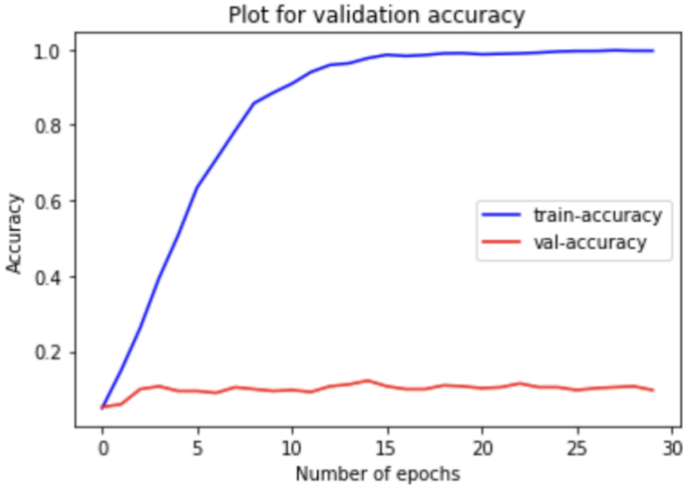

Figure 6: Accuracy from Model 3

The following table shows the Average Precision from the models.

Table 1: Average precision values per model

Model	*Average Precision*
Model 1	*68*
Model 2	*68*
Model 3	*68*

Analysis

Effect of number of layers in the model: - As I increased the number of layers in the models, the training accuracy increased for the model. Both model 2 and model 3 almost reached 99% accuracy but model 3 reached it faster than the model 2. This trend can be attributed to the fact that with more layers, more knowledge is learned by the models and hence better accuracy on the training dataset.

Validation Accuracy vs Training Accuracy: - From the plots, we can see that even though the training accuracy reached almost 100%, validation accuracy remained very low for all the models. This can be attributed to the reason that the label class in validation set is very different from the training set. Has there been less biasness between the training set and validation set, the validation accuracy may have increased.

Average Precision for the models: - The average precision (AP) value remained same for all the models. What was surprising was that the value was almost like that demonstrated in the coding tutorial where all the values in the array was 0 and the AP was 69%. I tried to change the code to make my precision better, but it was same no matter what. With the values that I have in my results.csv, I see that the values are very small. And with the function that I have written to calculate the AP, which appends 0 to the pred array if value is < 0.5 and 1 otherwise, I guess that the array would be of 600 values of 0 for me as well.

Test Set

Method

For the test set, I used the third model to predict the values. I used the same code block that I had used to extract the features from training and validation dataset. After extracting the features, I fed the values to the model to predict the values. After this, I used the np.amex function to get the max value from the arrays and then stored it into the results dataframe. This dataframe was then saves into the results.csv file.

Results

The loss plot for the final model is same as the loss plot for model 3

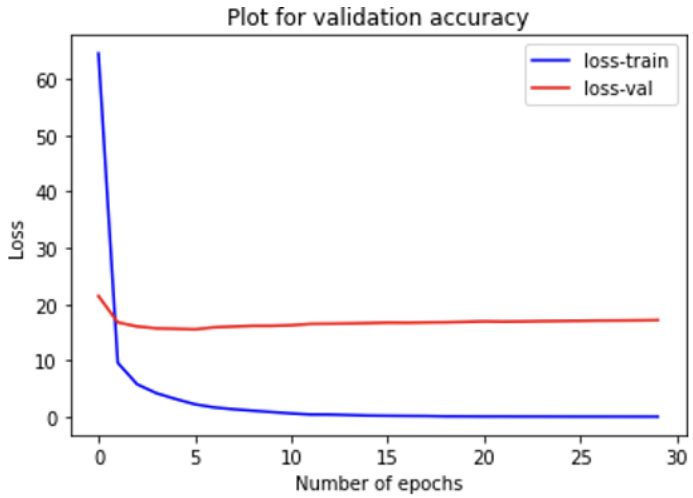

Figure 7: Loss for the final model

Analysis

From the plot, it looks like that the training loss is decreasing but the validation loss decreases and then increases. This looks like a case of overfitting the model. This trend of validation loss increasing happened in all my models but model 3 was the best model out of all the models. And hence I decided to go forward with this model.

```
! pip install transformers
import requests
from sklearn import preprocessing
from keras.applications.vgg16 import VGG16
from skimage import io
import numpy as np
from keras.utils import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input
from urllib import request
from io import BytesIO
from PIL import Image
from transformers import AutoTokenizer
from collections import Counter
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Embedding, Dense, LSTM
from sklearn.metrics import average_precision_score
import pandas as pd

img_dir = "https://vizwiz.cs.colorado.edu//VizWiz_visualization_img/"

##3. Training data
split = "train"
annotation_file_train = "https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/Annotations/%s.json" %split
split_data = requests.get(annotation_file_train, allow_redirects=True)
train_data = split_data.json()
print(len(train_data))
## validation dataset

split = "val"
annotation_file_val = "https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/Annotations/%s.json" %split
split_data_val = requests.get(annotation_file_val, allow_redirects=True)
val_data = split_data_val.json()
print(len(val_data))
# test dataset

split = "test"
annotation_file_test = "https://vizwiz.cs.colorado.edu/VizWiz_final/vqa_data/Annotations/%s.json" %split
split_data = requests.get(annotation_file_test, allow_redirects=True)
test_data = split_data.json()
print(len(test_data))
le = preprocessing.LabelEncoder()

def clean_answers_data(answers,common_answers):
  result = []
  answers_len = len(answers)
  for i in range(answers_len):
    if answers[i] in common_answers:
      result.append(answers[i])
    else:
      result.append(common_answers[0]) #print(result)
  return result

def extract_image_features(image_url):
  res = request.urlopen(image_url).read()
  img = Image.open(BytesIO(res)).resize((128,128))
  img_data = img_to_array(img)
  img_data = np.expand_dims(img_data, axis = 0)
  img_data = preprocess_input(img_data)
  features = image_model.predict(img_data)
  arr = np.array(features)
  image_feature_vector = arr.ravel()
  #print(image_feature_vector)
  #print(len(image_feature_vector))
  return image_feature_vector

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

def tokenize_function(examples):
  return tokenizer(examples, padding="max_length", truncation=True)

def extract_question_features(question):
  tokenized_question = tokenize_function(question)
  return tokenized_question['input_ids']
answer_list = []
X = [] ## features
y = [] ## target labels
for vq in train_data[17000:19000]:
  # Extract features describing the image
  image_name = vq['image']
  image_url = img_dir + image_name
  image_feature = extract_image_features(image_url)
  question = vq['question']
  question_feature = extract_question_features(question)
```

```
  # Create a multimodal feature to represent both the image and question (e.g. concate
  multimodal_features = np.concatenate((question_feature, image_feature))

  # Prepare features and labels
  X.append(multimodal_features)

  answers = vq['answers']

  label = answers[0]['answer']

  answer_list.append(label)

multimodal_features_len = len(X)

clean_answer_list = answer_list

frequent_answers_counter = Counter(answer_list).most_common(10)
frequent_answers = [x[0] for x in frequent_answers_counter]
class_labels_len = len(le.fit_transform(answer_list))
print(frequent_answers)

Y = tf.stack(le.transform(clean_answer_list))
X = tf.stack(X)

def plot_graph(graph_val):
  plt.plot(graph_val.history["accuracy"], color ="blue", label = "train-accuracy")
  plt.plot(graph_val.history["val_accuracy"], color ="red", label = "val-accuracy")
  plt.title("Plot for validation accuracy")
  plt.ylabel("Accuracy")
  plt.xlabel("Number of epochs")
  plt.legend()
  plt.show()

model_1 = tf.keras.Sequential([
  tf.keras.layers.Dense(multimodal_features_len, activation='sigmoid'),
  tf.keras.layers.Dense(class_labels_len, activation='sigmoid')
  ])
model_1.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
graph_val = model_1.fit(X, Y, epochs=30,  validation_split= 0.2)
plot_graph(graph_val)
model_1.summary()

model_2 = tf.keras.Sequential([
    tf.keras.layers.Dense(multimodal_features_len, activation='sigmoid'),
  tf.keras.layers.Dense(multimodal_features_len, activation='sigmoid'),
  tf.keras.layers.Dense(class_labels_len, activation='sigmoid')
])
model_2.compile(optimizer='RMSProp', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
graph_val = model_2.fit(X, Y, epochs=30,  validation_split= 0.2)
plot_graph(graph_val)
model_2.summary()

model_3 = tf.keras.Sequential([
  tf.keras.layers.Dense(multimodal_features_len, activation='relu'),
  tf.keras.layers.Dense(multimodal_features_len, activation='relu'),
  tf.keras.layers.Dense(multimodal_features_len, activation='relu'),
  tf.keras.layers.Dense(class_labels_len, activation='sigmoid')

])
model_3.compile(optimizer='Adagrad', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
graph_val = model_3.fit(X, Y, epochs=30, validation_split= 0.2)
plot_graph(graph_val)
model_3.summary()

def validate(results,data):
  # All answers
  gtlist = [x['answerable'] for x in data]
  # Save the accuracies
  y_test = []
  pred_list = []
  i=0
  #print(results)

  # Compute accuracy for each image
  for i in range(0, 600):
    # Get the GT answer list and preprocess
    y_test.append(gtlist[i])
    if results[i] >0.5:
      pred_list.append(1)
    else:
      pred_list.append(0)
  y_test = np.array(y_test)
```

```
  pred = np.array(pred_list)
  average_precision = average_precision_score(y_test, pred)
  print("AP: {}".format(round(100*average_precision, 4)))

X = [] ## features
y = [] ## target labels
multimodal_features_set = []

for vq in val_data[1000:1600]:
  # Extract features describing the image
  image_name = vq['image']

  image_url = img_dir + image_name
  image_feature = extract_image_features(image_url)
  #print(image_feature) # Extract features describing the question
  question = vq['question']

  question_feature = extract_question_features(question)
  #print(question_feature)
  # Create a multimodal feature to represent both the image and question (e.g. concate
  multimodal_features = np.concatenate((question_feature, image_feature))
  multimodal_features_set.append(multimodal_features)
multimodal_features_tensor = tf.convert_to_tensor(multimodal_features_set)
predicted_values_1 = model_1.predict(multimodal_features_tensor)
results_1 = [np.amax(predicted_values_1[i]) for i in range(len(predicted_values_1))]
validate(results_1,val_data[1000:1600])
multimodal_features_tensor = tf.convert_to_tensor(multimodal_features_set)
predicted_values_2 = model_2.predict(multimodal_features_tensor)
results_2 = [np.amax(predicted_values_2[i]) for i in range(len(predicted_values_2))]
validate(results_2,val_data[1000:1600])
multimodal_features_tensor = tf.convert_to_tensor(multimodal_features_set)
predicted_values_3 = model_3.predict(multimodal_features_tensor)
results_3 = [np.amax(predicted_values_3[i]) for i in range(len(predicted_values_3))]
validate(results_3,val_data[1000:1600])
num_VQs = 1000
results = []
multimodal_features_set = [] ##X = [] ## features y = [] ## target labels

for vq in test_data[0:num_VQs]:
  # Extract features describing the image
  image_name = vq['image']
  image_url = img_dir + image_name
  image_feature = extract_image_features(image_url)
  question = vq['question']
  question_feature = extract_question_features(question)
  multimodal_features = np.concatenate((question_feature, image_feature))
  multimodal_features_set.append(multimodal_features)

multimodal_features_tensor = tf.convert_to_tensor(multimodal_features_set)
predicted_values_final = model_3.predict(multimodal_features_tensor)
#index_of_maximums = [np.where(output == max(output)) for output in predicted_values]
#result = [x[0].tolist()[0] for x in index_of_maximums]
#results = le.inverse_transform(result)
results = [np.amax(predicted_values_final[i]) for i in range(len(predicted_values_final))]
#print(results)
import pandas as pd
df = pd.DataFrame(results)
df.to_csv("results.csv", header = None, index = None)
```