PATTERN RECOGNITION

VQA Report

Basma Hesham	ID: 5432
Mariam Medhat	ID: 5351
Toka Sherif	ID: 5492

Collab link:
https://colab.research.google.com/drive/1fc0wF4HzZE5NmEPV9QE8cSsEY7SUYEh5?usp=sharing

Problem Statement

Our goal is to develop a machine learning model to answer any question posed on any given image using only the features in the image.

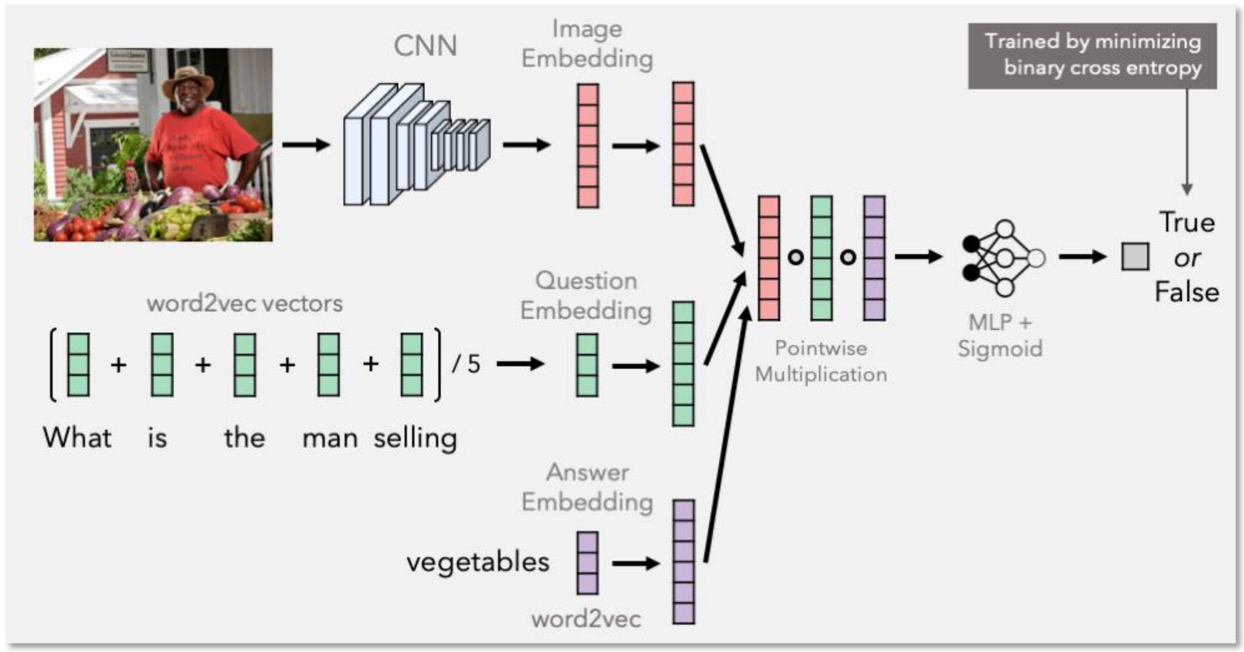


Fig1: VQA architectural design

Dataset used

Vizwiz dataset containing around 20k for training and 4k for validation and 8k for testing. Each question/ image has approximately 10 answers. We used around 1000 for training and validation and 100 for testing, due to lack of computation resources.

Image processing

In the image processing we used the Vgg16 model to extract the image features.

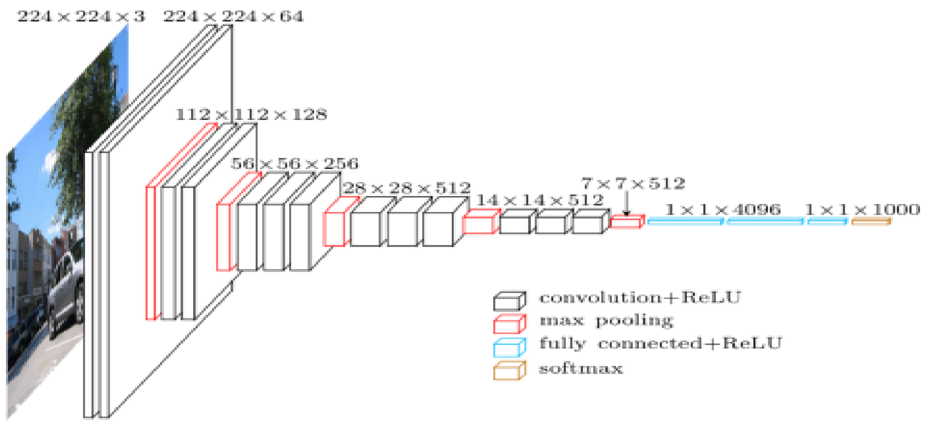
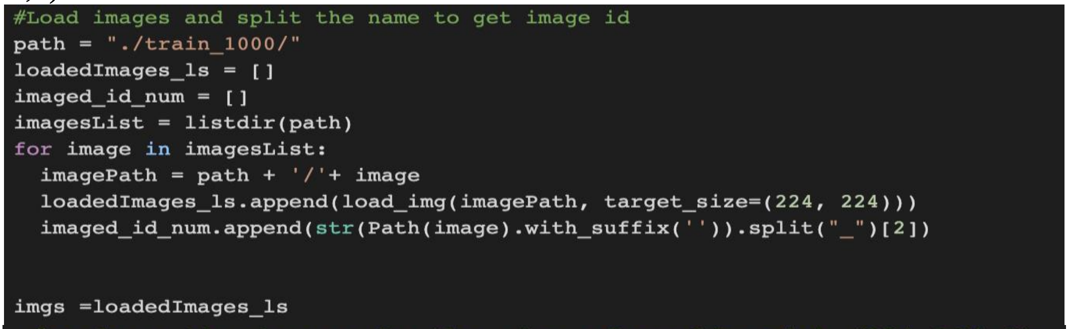
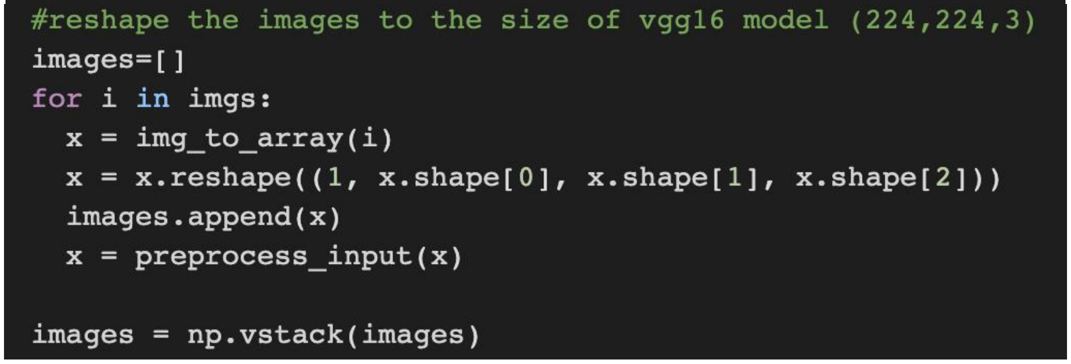


Fig2: VGG16 model

To apply the vgg16 model, first we needed to extract the images then reshaped them to (224,224,3).

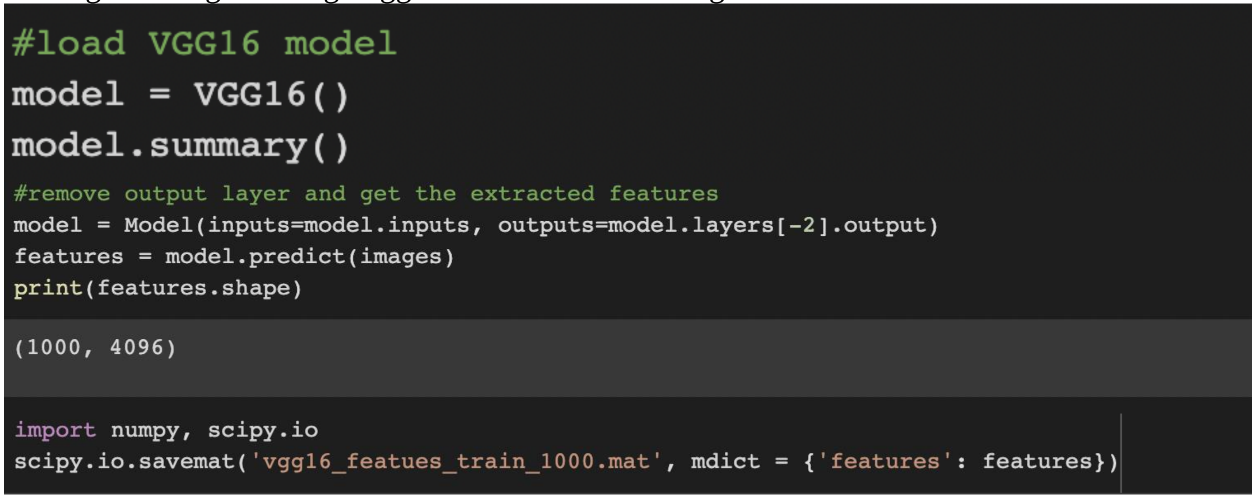


```
#Load images and split the name to get image id
path = "./train_1000/"
loadedImages_ls = []
imaged_id_num = []
imagesList = listdir(path)
for image in imagesList:
  imagePath = path + '/'+ image
  loadedImages_ls.append(load_img(imagePath, target_size=(224, 224)))
  imaged_id_num.append(str(Path(image).with_suffix('')).split("_")[2])


imgs =loadedImages_ls
```



```
#reshape the images to the size of vgg16 model (224,224,3)
images=[]
for i in imgs:
  x = img_to_array(i)
  x = x.reshape((1, x.shape[0], x.shape[1], x.shape[2]))
  images.append(x)
  x = preprocess_input(x)

images = np.vstack(images)
```

Passing the images through vgg16 Model and extracting the features.



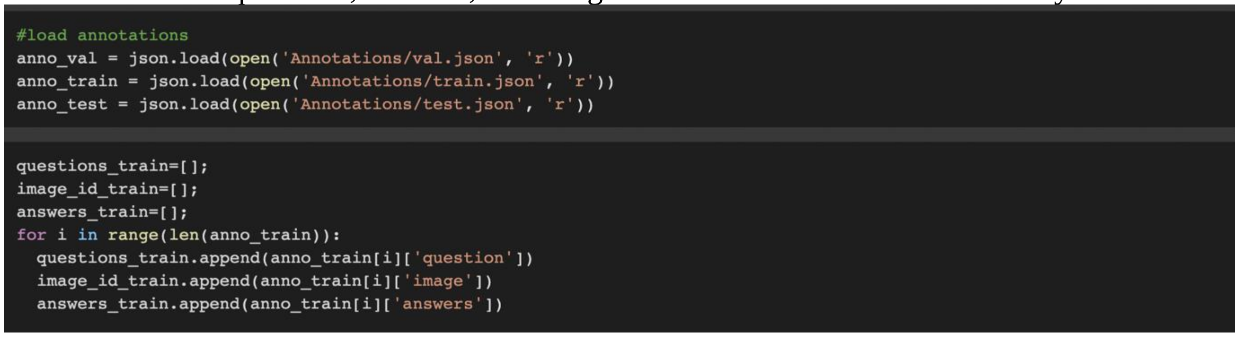
```
#load VGG16 model
model = VGG16()
model.summary()
#remove output layer and get the extracted features
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
features = model.predict(images)
print(features.shape)
```

```
(1000, 4096)
```

```
import numpy, scipy.io
scipy.io.savemat('vgg16_featues_train_1000.mat', mdict = {'features': features})
```

Language Processing

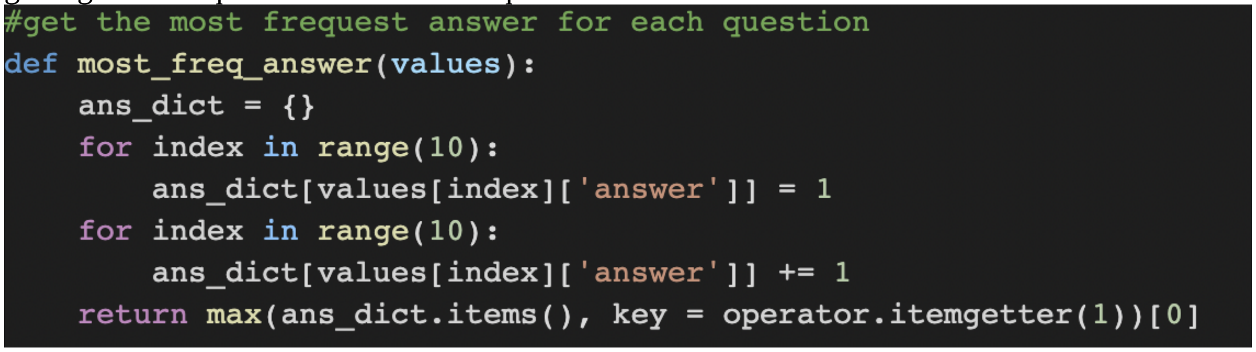
We extracted the questions, answers, and image ids from annotations file into arrays.



```
#load annotations
anno_val = json.load(open('Annotations/val.json', 'r'))
anno_train = json.load(open('Annotations/train.json', 'r'))
anno_test = json.load(open('Annotations/test.json', 'r'))
```

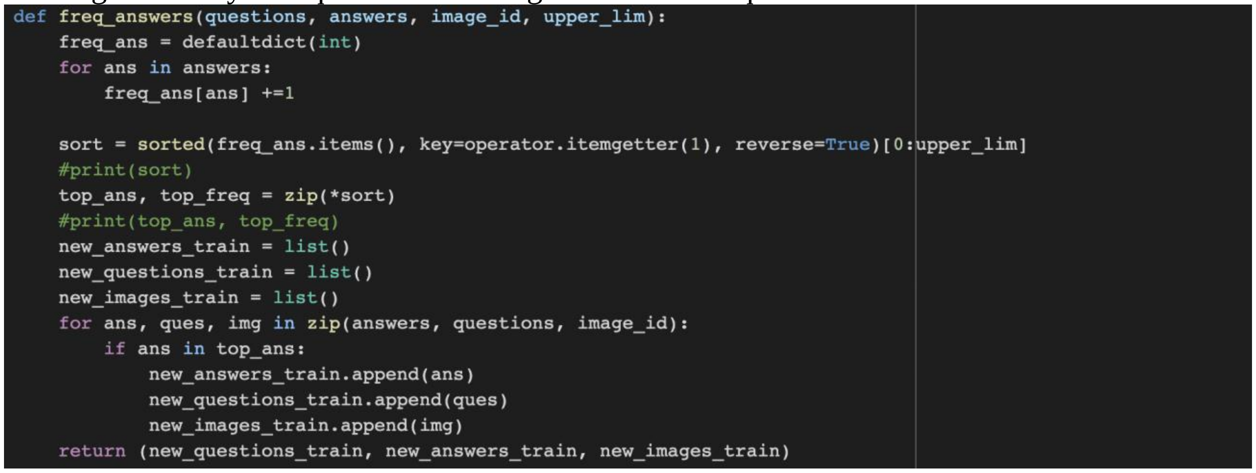
```
questions_train=[];
image_id_train=[];
answers_train=[];
for i in range(len(anno_train)):
  questions_train.append(anno_train[i]['question'])
  image_id_train.append(anno_train[i]['image'])
  answers_train.append(anno_train[i]['answers'])
```

getting most frequent answer for each question.



```
#get the most frequest answer for each question
def most_freq_answer(values):
    ans_dict = {}
    for index in range(10):
        ans_dict[values[index]['answer']] = 1
    for index in range(10):
        ans_dict[values[index]['answer']] += 1
    return max(ans_dict.items(), key = operator.itemgetter(1))[0]
```

creating new arrays for question and image id of most frequent answers.

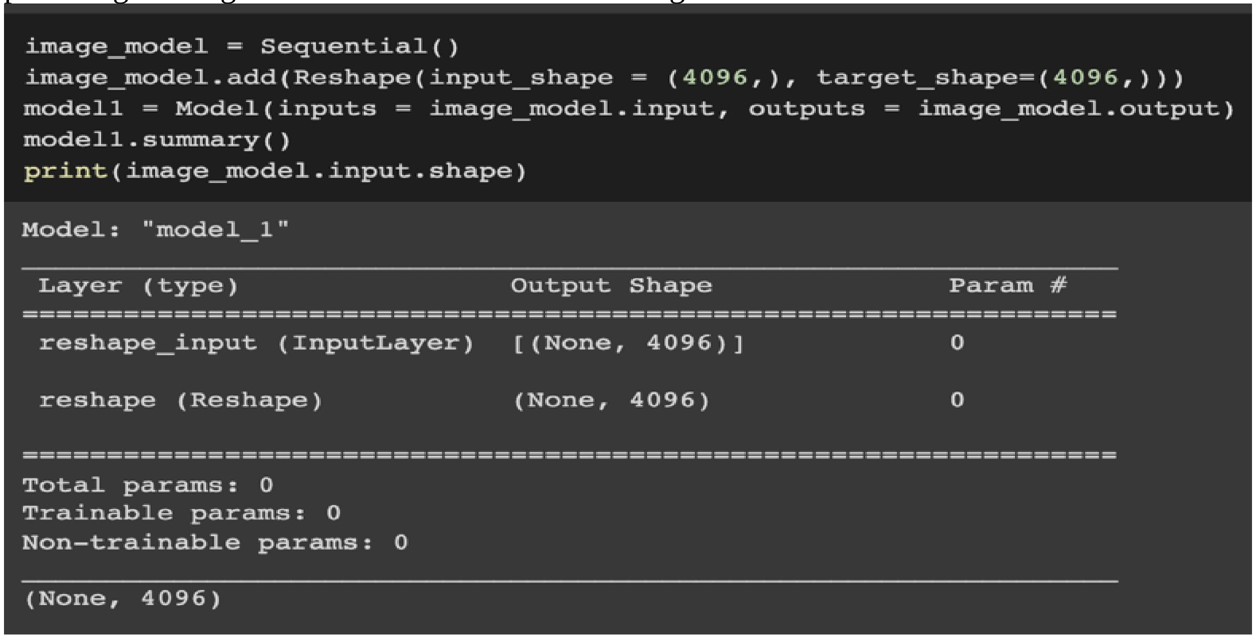


```python
def freq_answers(questions, answers, image_id, upper_lim):
    freq_ans = defaultdict(int)
    for ans in answers:
        freq_ans[ans] +=1

    sort = sorted(freq_ans.items(), key=operator.itemgetter(1), reverse=True)[0:upper_lim]
    #print(sort)
    top_ans, top_freq = zip(*sort)
    #print(top_ans, top_freq)
    new_answers_train = list()
    new_questions_train = list()
    new_images_train = list()
    for ans, ques, img in zip(answers, questions, image_id):
        if ans in top_ans:
            new_answers_train.append(ans)
            new_questions_train.append(ques)
            new_images_train.append(img)
    return (new_questions_train, new_answers_train, new_images_train)
```

Creating model

The model is divided into 2 main parts. CNN applied to analyze the imagery and LSTM to process, classify and predict the language. The third step is combining both CNN and LSTM.

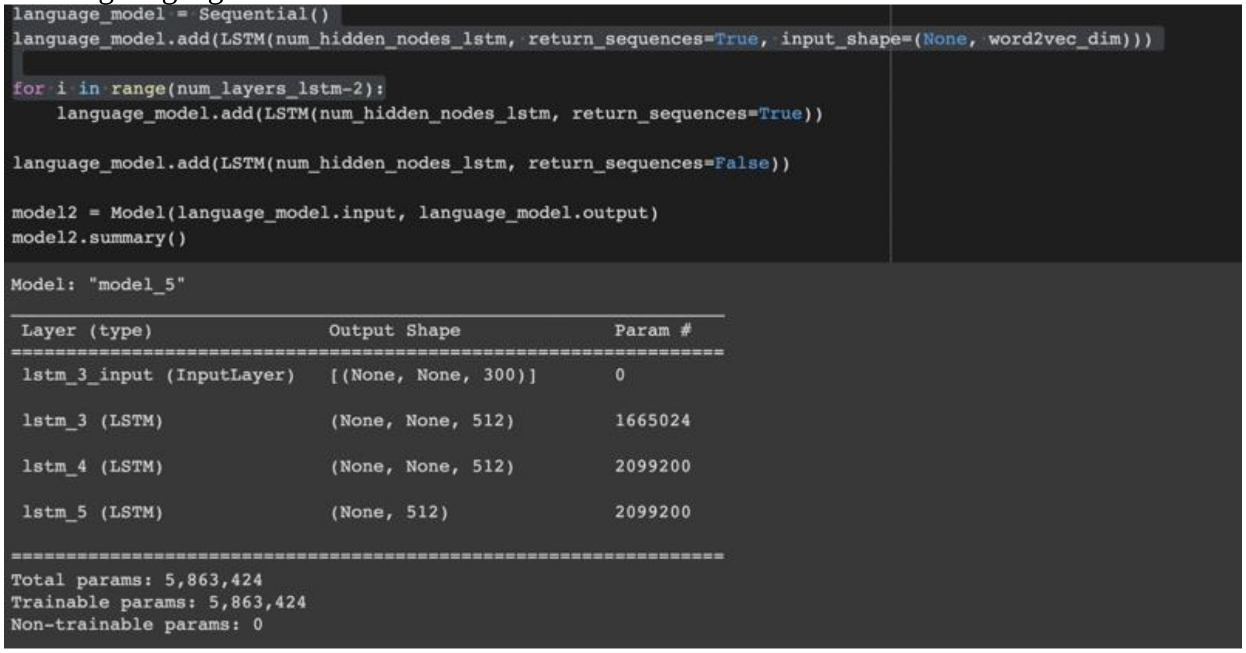
providing training and inference features on the image model.



```python
image_model = Sequential()
image_model.add(Reshape(input_shape = (4096,), target_shape=(4096,)))
model1 = Model(inputs = image_model.input, outputs = image_model.output)
model1.summary()
print(image_model.input.shape)
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape_input (InputLayer)  [(None, 4096)]            0

 reshape (Reshape)           (None, 4096)              0

=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
(None, 4096)
```

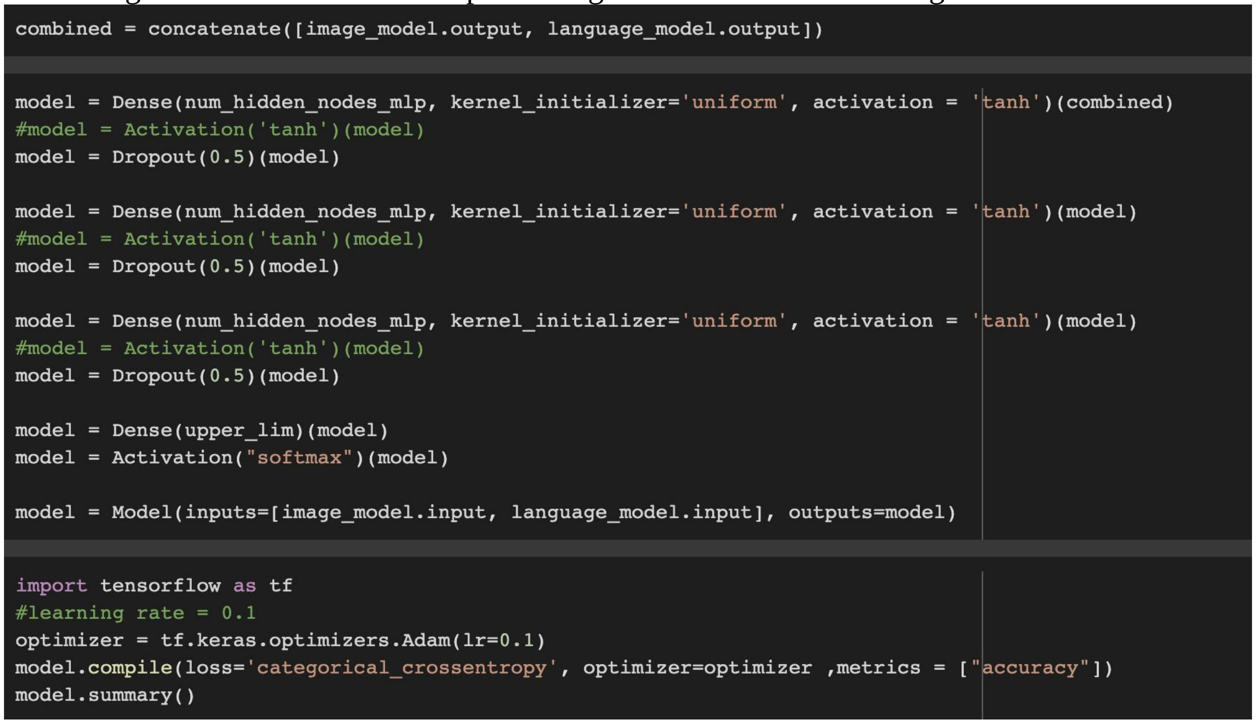
Creating language LSTM model



```
language_model = Sequential()
language_model.add(LSTM(num_hidden_nodes_lstm, return_sequences=True, input_shape=(None, word2vec_dim)))

for i in range(num_layers_lstm-2):
    language_model.add(LSTM(num_hidden_nodes_lstm, return_sequences=True))

language_model.add(LSTM(num_hidden_nodes_lstm, return_sequences=False))

model2 = Model(language_model.input, language_model.output)
model2.summary()
```

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3_input (InputLayer)   [(None, None, 300)]       0

 lstm_3 (LSTM)               (None, None, 512)         1665024

 lstm_4 (LSTM)               (None, None, 512)         2099200

 lstm_5 (LSTM)               (None, 512)               2099200

=================================================================
Total params: 5,863,424
Trainable params: 5,863,424
Non-trainable params: 0
```

combining both models. model.compile configures the model for training.

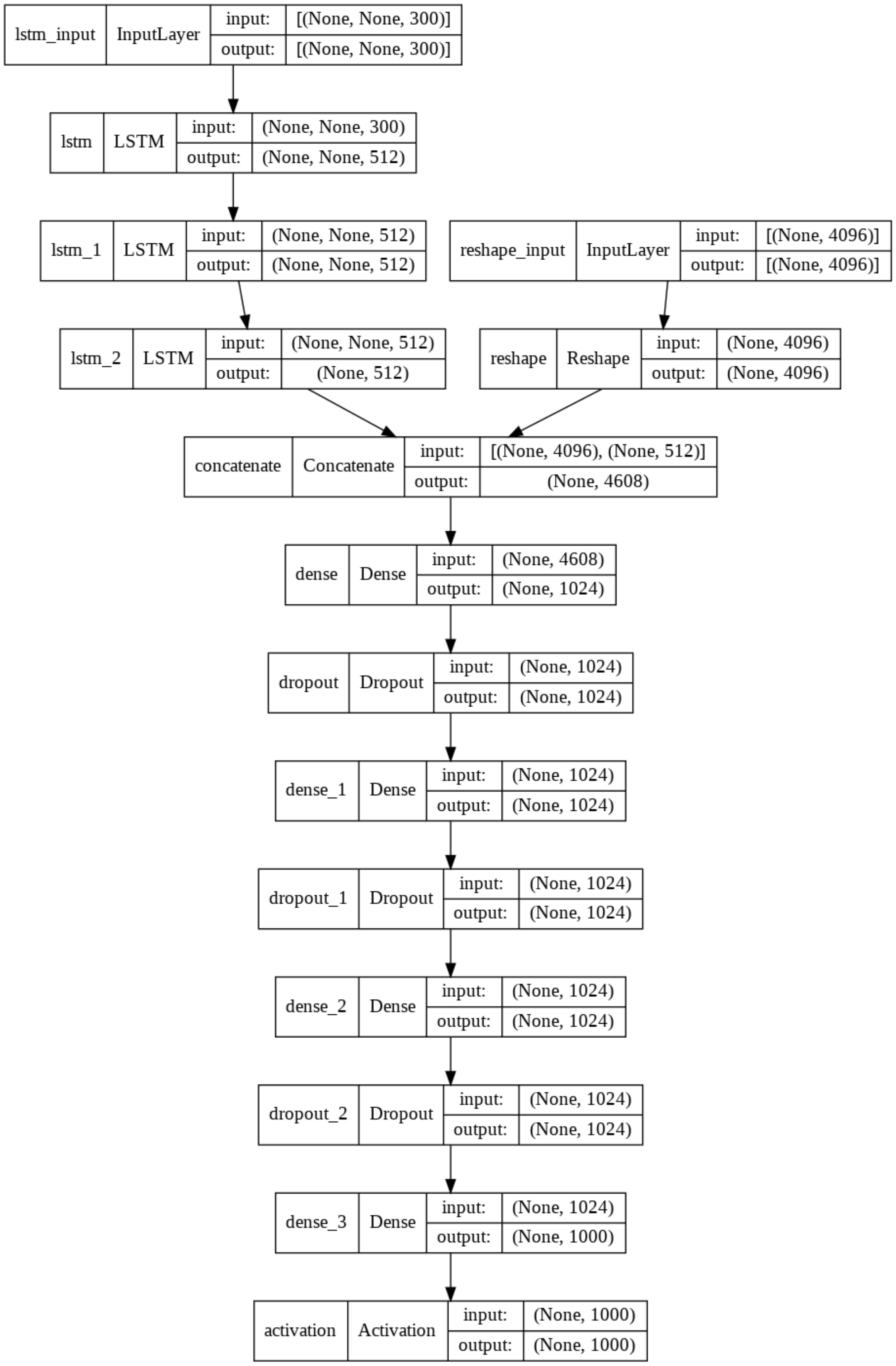


```
combined = concatenate([image_model.output, language_model.output])
```

```
model = Dense(num_hidden_nodes_mlp, kernel_initializer='uniform', activation = 'tanh')(combined)
#model = Activation('tanh')(model)
model = Dropout(0.5)(model)

model = Dense(num_hidden_nodes_mlp, kernel_initializer='uniform', activation = 'tanh')(model)
#model = Activation('tanh')(model)
model = Dropout(0.5)(model)

model = Dense(num_hidden_nodes_mlp, kernel_initializer='uniform', activation = 'tanh')(model)
#model = Activation('tanh')(model)
model = Dropout(0.5)(model)

model = Dense(upper_lim)(model)
model = Activation("softmax")(model)

model = Model(inputs=[image_model.input, language_model.input], outputs=model)
```

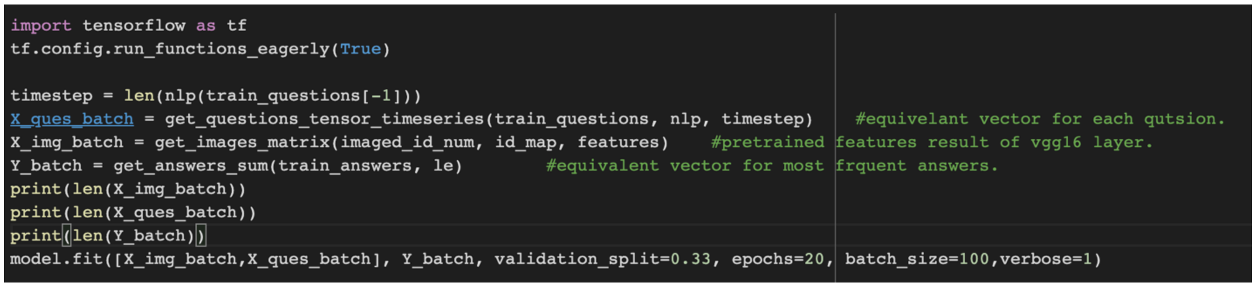
```
import tensorflow as tf
#learning rate = 0.1
optimizer = tf.keras.optimizers.Adam(lr=0.1)
model.compile(loss='categorical_crossentropy', optimizer=optimizer ,metrics = ["accuracy"])
model.summary()
```

The following diagram shows the full model:


lstm_input | InputLayer | input: [(None, None, 300)] | output: [(None, None, 300)]
lstm | LSTM | input: (None, None, 300) | output: (None, None, 512)
lstm_1 | LSTM | input: (None, None, 512) | output: (None, None, 512)
reshape_input | InputLayer | input: [(None, 4096)] | output: [(None, 4096)]
lstm_2 | LSTM | input: (None, None, 512) | output: (None, 512)
reshape | Reshape | input: (None, 4096) | output: (None, 4096)
concatenate | Concatenate | input: [(None, 4096), (None, 512)] | output: (None, 4608)
dense | Dense | input: (None, 4608) | output: (None, 1024)
dropout | Dropout | input: (None, 1024) | output: (None, 1024)
dense_1 | Dense | input: (None, 1024) | output: (None, 1024)
dropout_1 | Dropout | input: (None, 1024) | output: (None, 1024)
dense_2 | Dense | input: (None, 1024) | output: (None, 1024)
dropout_2 | Dropout | input: (None, 1024) | output: (None, 1024)
dense_3 | Dense | input: (None, 1024) | output: (None, 1000)
activation | Activation | input: (None, 1000) | output: (None, 1000)

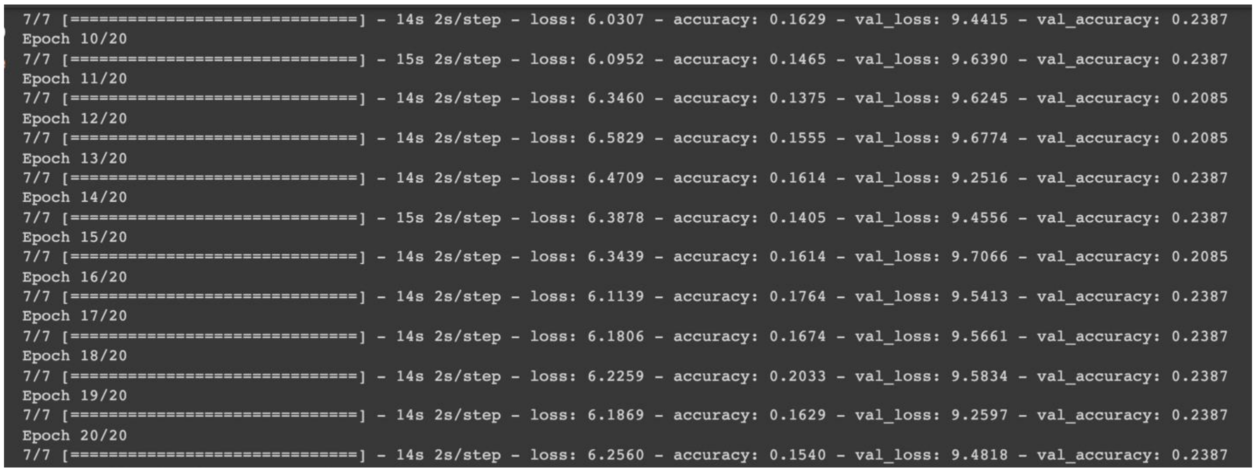
Training the data into the model

model.fit Trains the model for 20 epochs. 30% of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the x (image ids and questions) and y (answers) data provided, before shuffling.



```
import tensorflow as tf
tf.config.run_functions_eagerly(True)

timestep = len(nlp(train_questions[-1]))
X_ques_batch = get_questions_tensor_timeseries(train_questions, nlp, timestep)    #equivelant vector for each qutsion.
X_img_batch = get_images_matrix(imaged_id_num, id_map, features)    #pretrained features result of vgg16 layer.
Y_batch = get_answers_sum(train_answers, le)        #equivalent vector for most frquent answers.
print(len(X_img_batch))
print(len(X_ques_batch))
print(len(Y_batch))
model.fit([X_img_batch,X_ques_batch], Y_batch, validation_split=0.33, epochs=20, batch_size=100,verbose=1)
```

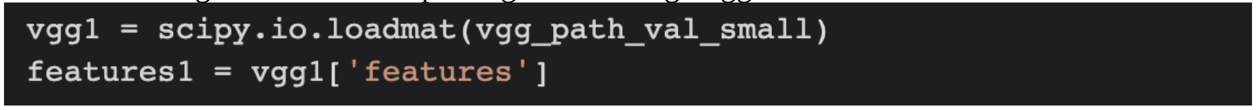
This figure shows the model loss and accuracy results, as well as validation loss and accuracy results.



```
7/7 [==============================] - 14s 2s/step - loss: 6.0307 - accuracy: 0.1629 - val_loss: 9.4415 - val_accuracy: 0.2387
Epoch 10/20
7/7 [==============================] - 15s 2s/step - loss: 6.0952 - accuracy: 0.1465 - val_loss: 9.6390 - val_accuracy: 0.2387
Epoch 11/20
7/7 [==============================] - 14s 2s/step - loss: 6.3460 - accuracy: 0.1375 - val_loss: 9.6245 - val_accuracy: 0.2085
Epoch 12/20
7/7 [==============================] - 14s 2s/step - loss: 6.5829 - accuracy: 0.1555 - val_loss: 9.6774 - val_accuracy: 0.2085
Epoch 13/20
7/7 [==============================] - 14s 2s/step - loss: 6.4709 - accuracy: 0.1614 - val_loss: 9.2516 - val_accuracy: 0.2387
Epoch 14/20
7/7 [==============================] - 15s 2s/step - loss: 6.3878 - accuracy: 0.1405 - val_loss: 9.4556 - val_accuracy: 0.2387
Epoch 15/20
7/7 [==============================] - 14s 2s/step - loss: 6.3439 - accuracy: 0.1614 - val_loss: 9.7066 - val_accuracy: 0.2085
Epoch 16/20
7/7 [==============================] - 14s 2s/step - loss: 6.1139 - accuracy: 0.1764 - val_loss: 9.5413 - val_accuracy: 0.2387
Epoch 17/20
7/7 [==============================] - 14s 2s/step - loss: 6.1806 - accuracy: 0.1674 - val_loss: 9.5661 - val_accuracy: 0.2387
Epoch 18/20
7/7 [==============================] - 14s 2s/step - loss: 6.2259 - accuracy: 0.2033 - val_loss: 9.5834 - val_accuracy: 0.2387
Epoch 19/20
7/7 [==============================] - 14s 2s/step - loss: 6.1869 - accuracy: 0.1629 - val_loss: 9.2597 - val_accuracy: 0.2387
Epoch 20/20
7/7 [==============================] - 14s 2s/step - loss: 6.2560 - accuracy: 0.1540 - val_loss: 9.4818 - val_accuracy: 0.2387
```

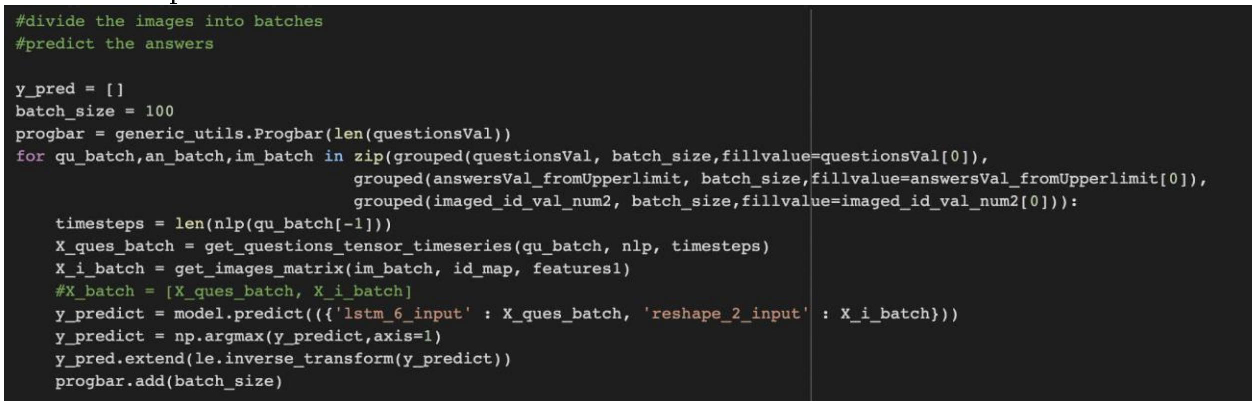
Testing the model

extract the images features after passing them through vgg16 model.



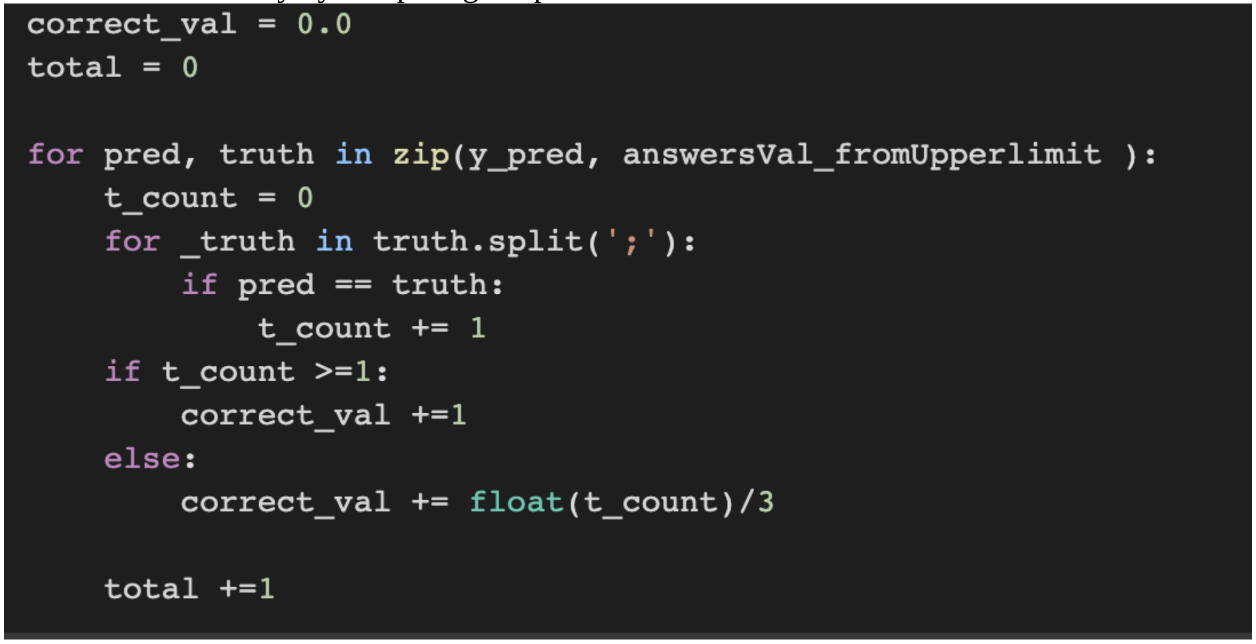
```
vgg1 = scipy.io.loadmat(vgg_path_val_small)
features1 = vgg1['features']
```

Then used model.predict to pass question vector through the model and return the answer tensor for each datapoint.



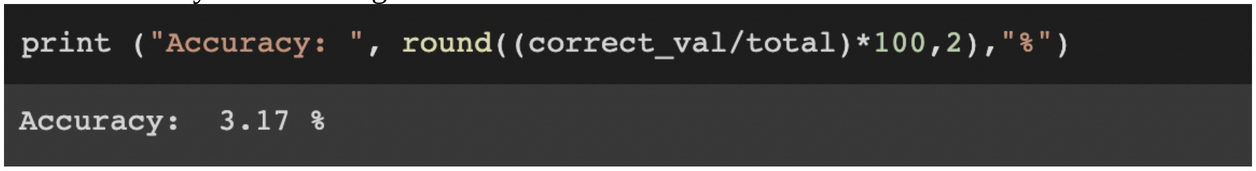
```
#divide the images into batches
#predict the answers

y_pred = []
batch_size = 100
progbar = generic_utils.Progbar(len(questionsVal))
for qu_batch,an_batch,im_batch in zip(grouped(questionsVal, batch_size,fillvalue=questionsVal[0]),
                                  grouped(answersVal_fromUpperlimit, batch_size,fillvalue=answersVal_fromUpperlimit[0]),
                                  grouped(imaged_id_val_num2, batch_size,fillvalue=imaged_id_val_num2[0])):
    timesteps = len(nlp(qu_batch[-1]))
    X_ques_batch = get_questions_tensor_timeseries(qu_batch, nlp, timesteps)
    X_i_batch = get_images_matrix(im_batch, id_map, features1)
    #X_batch = [X_ques_batch, X_i_batch]
    y_predict = model.predict(({'lstm_6_input' : X_ques_batch, 'reshape_2_input' : X_i_batch}))
    y_predict = np.argmax(y_predict,axis=1)
    y_pred.extend(le.inverse_transform(y_predict))
    progbar.add(batch_size)
```

Calculate the accuracy by comparing the predicted answers to the real one



```
correct_val = 0.0
total = 0

for pred, truth in zip(y_pred, answersVal_fromUpperlimit ):
    t_count = 0
    for _truth in truth.split(';'):
        if pred == truth:
            t_count += 1
    if t_count >=1:
        correct_val +=1
    else:
        correct_val += float(t_count)/3

    total +=1
```

Model accuracy with learning rate = 0.1



```
print ("Accuracy: ", round((correct_val/total)*100,2),"%")
```

```
Accuracy:  3.17 %
```