Name: Royyuru Sai Prasanna Gangadhar **Roll Number: ME19B190**

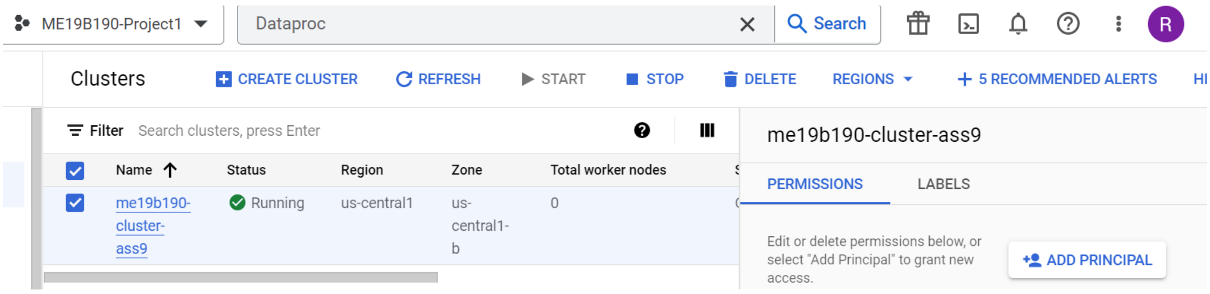
ML Assignment:

In this assignment, you will be working on the Pima Indians Diabetes Database (https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database)

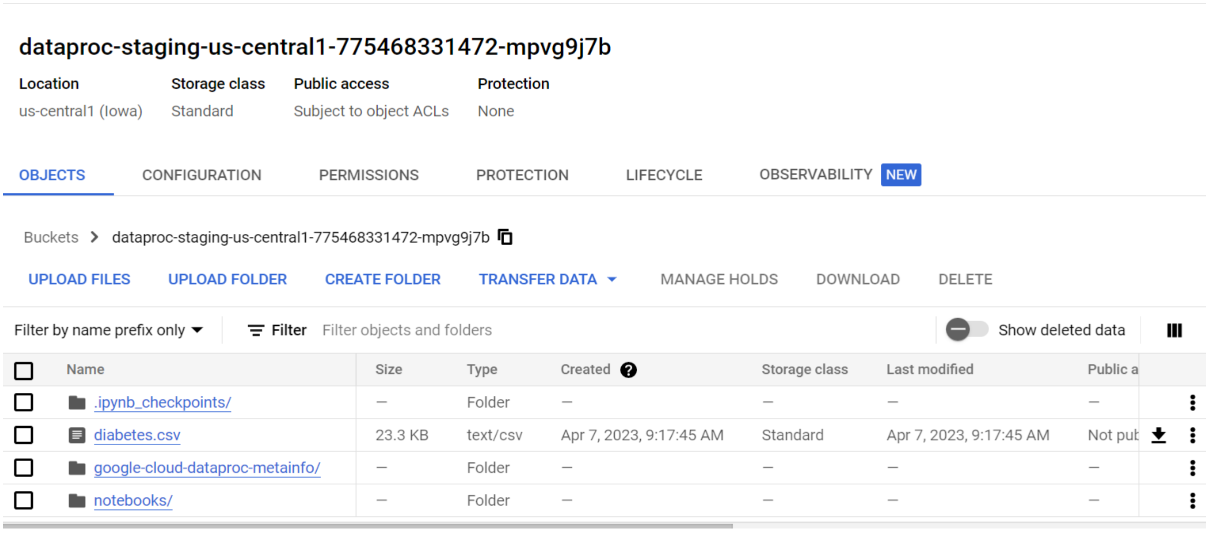
1. Download the dataset and upload it into your bucket.

2. Train an ML model on the dataset to predict the Outcome and report the accuracy for different pre-processing techniques and models. Provide the details of data exploration and feature engineering steps. You also need to submit the code along with the answers to the above questions

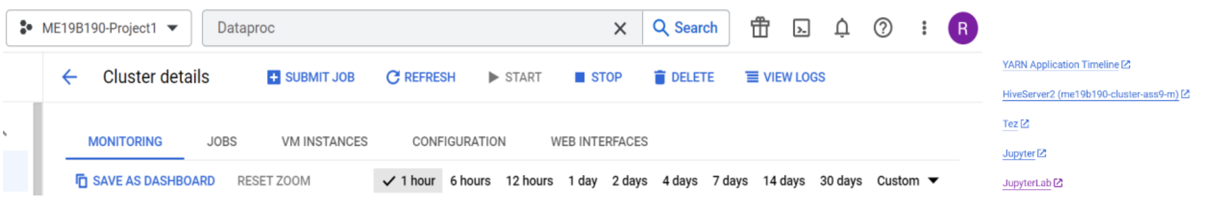
Step 1: Create a Dataproc cluster with



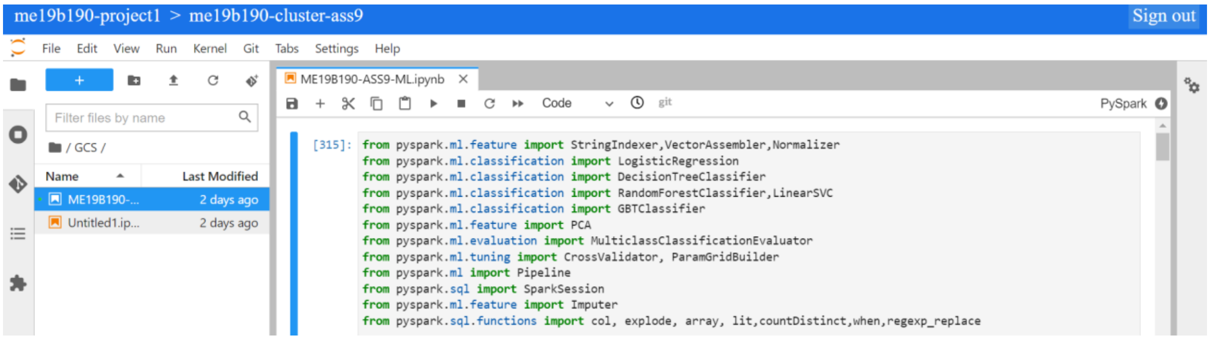
Step 2: Download the Pima Indians Diabetes Database from (pima-indians-dataset)



Step 3: Open Web Interfaces in the DataProc cluster and open Jupyter notebook. Select cluster type as "Single Node (1 master, 0 workers)", 2.0-debian10 image, and select Jupyter Notebook in optional components. In the Configure Nodes section, N2 Series n2-standard-4 Machine Type is selected with 4 vCPUs and ~4GB Memory and everything else kept default.



Step 4: Launch the Jupyter Notebook



Code Explanation:

1. Reading the dataset from the bucket and storing it as a pyspark dataframe

```
[316]: file_location ="gs://dataproc-staging-us-central1-775468331472-mpvg9j7b/diabetes.csv"
training = spark.read.format("csv").option("header","True").load(file_location).toDF("Pregnancies","Glucose","BloodF
training = training.withColumn("Pregnancies",
training["Pregnancies"].cast("float")).withColumn("Glucose",
training["Glucose"].cast("float")).withColumn("BloodPressure",
training["BloodPressure"].cast("float")).withColumn("SkinThickness",
```

2. Null Value Imputation:
   Null values in the dataset have been observed, thus null values have been imputed using the Imputer available in pyspark library

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	label
6.0	148.0	72.0	35.0	null	33.6	0.627	50.0	1.0
1.0	85.0	66.0	29.0	null	26.6	0.351	31.0	0.0
8.0	183.0	64.0	null	null	23.3	0.672	32.0	1.0
1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0	0.0
0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0	1.0
5.0	116.0	74.0	null	null	25.6	0.201	30.0	0.0
3.0	78.0	50.0	32.0	88.0	31.0	0.248	26.0	1.0
10.0	115.0	null	null	null	35.3	0.134	29.0	0.0
2.0	197.0	70.0	45.0	543.0	30.5	0.158	53.0	1.0
8.0	125.0	96.0	null	null	null	0.232	54.0	1.0
4.0	110.0	92.0	null	null	37.6	0.191	30.0	0.0
10.0	168.0	74.0	null	null	38.0	0.537	34.0	1.0

   After Null value imputation:

```
imputer = Imputer(
    inputCols = train.columns,
    outputCols = train.columns,
).fit(train)

train = imputer.transform(train)
test = imputer.transform(test)
train.show(5)
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	label
0.0	57.0	60.0	29.363432	154.67284	21.7	0.735	67.0	0.0
0.0	67.0	76.0	29.363432	154.67284	45.3	0.194	46.0	0.0
0.0	73.0	72.5231	29.363432	154.67284	21.1	0.342	25.0	0.0
0.0	74.0	52.0	10.0	36.0	27.8	0.269	22.0	0.0
0.0	84.0	64.0	22.0	66.0	35.8	0.545	21.0	0.0

3. Class Imbalance: Checking if there is any class imbalance in the dataset
   Yes, there is class imbalance in the dataset. The class “0” records are approximately twice as many as class “1” records. Thus, under sampling and oversampling techniques have been explored

```
[ ] major_df = train.filter(train.label == 0.0)
    minor_df = train.filter(train.label == 1.0)
    print("Majority class is 0 with count: ",major_df.count())
    print("Minority class is 1 with count: ",minor_df.count())
    ratio = round(major_df.count()/minor_df.count())
    print("ratio of majority and minority class: {}".format(ratio))
```

```
Majority class is 0 with count:  408
Minority class is 1 with count:  221
ratio of majority and minority class: 2
```

```
#undersampling of the majority class
sampled_majority_df = major_df.sample(False, 1/ratio)
train_undersampling = sampled_majority_df.unionAll(minor_df)
print("Train data with Under Sampling")
train_undersampling.show(5)
print('-----------------------------------------------------------------------')
#oversampling
a = range(ratio)
print("Train data with Over Sampling")
# duplicate the minority rows
train_oversampling = minor_df.withColumn("temp_col",
                         explode(array([lit(x) for x in a]))).drop('temp_col')
train_oversampling.show(5)
```

4. Data pre-processing
   Vector assembler to concatenate the feature columns and Data normalizer to normalize the input dataset

```
[ ] #concatenating  the feature columns to make a feature vector
    assembler = VectorAssembler(inputCols  = ["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction","Age"],
                           outputCol = "temp_features")
```

```
[ ] df = train
    df = assembler.transform(df)
    df.show(5, False)
```

```
---+------------------------+----+-----+-----------------------------------------------------------------------------------------------------------+
MI |DiabetesPedigreeFunction|Age |label|temp_features                                                                                              |
---+------------------------+----+-----+-----------------------------------------------------------------------------------------------------------+
1.7|0.735                   |67.0|0.0  |[0.0,57.0,60.0,29.363431930541992,154.67283630371094,21.700000762939453,0.7350000143051147,67.0]            |
5.3|0.194                   |46.0|0.0  |[0.0,67.0,76.0,29.363431930541992,154.67283630371094,45.29999923706055,0.1940000057220459,46.0]             |
1.1|0.342                   |25.0|0.0  |[0.0,73.0,72.52310180664062,29.363431930541992,154.67283630371094,21.100000381469727,0.3420000762939453,25.0]|
7.8|0.269                   |22.0|0.0  |[0.0,74.0,52.0,10.0,36.0,27.799999237060547,0.26899999380111694,22.0]                                      |
5.8|0.545                   |21.0|0.0  |[0.0,84.0,64.0,22.0,66.0,35.79999923706055,0.5450000166893005,21.0]                                        |
```

```
[ ] normalizer = Normalizer(inputCol  = "temp_features",outputCol="norm_features",p = 1.0)
```

```
[ ] df = normalizer.transform(df)
```

```
[ ] df.select("norm_features").show(5)
```

```
+--------------------+
|       norm_features|
+--------------------+
|[0.0,0.1459774496...|
|[0.0,0.1600840015...|
|[0.0,0.1941482285...|
|[0.0,0.3332297630...|
|[0.0,0.2863522480...|
+--------------------+
only showing top 5 rows
```

5. Feature engineering: Principal Component analysis has been performed to extract the top 5 most important features from the dataset.

```
[14]: pca  = PCA(k = 5, inputCol = 'norm_features')
      pca.setOutputCol('features')

      model_pca = pca.fit(df)
      df = model_pca.transform(df)
      df.select("features").show(5,truncate = False)
```

```
23/04/09 19:10:35 WARN com.github.fommil.netlib.LAPACK: Failed to load implementation from: com.github.fommil.netli
b.NativeSystemLAPACK
23/04/09 19:10:35 WARN com.github.fommil.netlib.LAPACK: Failed to load implementation from: com.github.fommil.netli
b.NativeRefLAPACK
+---------------------------------------------------------------------------------------------------+
|features                                                                                           |
+---------------------------------------------------------------------------------------------------+
|[0.23471773600388526,-0.05495898059059314,-0.037257758048048206,0.11606018091238766,0.0376103331051093]  |
|[0.19963916199366438,-0.05582877593762735,-0.10533508333468442,0.08563876542886653,0.0010375302056054138]|
|[0.23166956014258616,-0.09876342303641925,-0.14764875497375302,0.09734157052893108,0.05438359013801711]  |
|[-0.054834510446004694,-0.15676612084398914,-0.13853308773482853,0.10933285345673513,-0.012097852450469453]|
|[0.021445914206695067,-0.1321544705807052,-0.15076564575273374,0.07488399408294193,0.00508108516937833]  |
+---------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

6. Building the pipeline and deploying models: Various models like Logistic regression, Decision Tree classifier, Random Forest Classifier, GBT Classifier (Gradient boost), and Support vector machines have been trained to give predictions on Pima Indians Diabetes Database

```
[ ] lr = LogisticRegression(maxIter = 10)
    pipeline =  Pipeline(stages = [assembler,normalizer,pca,lr])

    model = pipeline.fit(train)
    pred = model.transform(test)
    print("Accuracy - Logistic regression: {} ".format(MulticlassClassificationEvaluator(metricName='accuracy').evaluate(pred)))
    print("F1 score - Logistic regression: {} ".format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
[ ] dt = DecisionTreeClassifier(maxDepth = 5)
    pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,dt])
    model = pipeline.fit(train)
    pred = model.transform(test)
    print("Accuracy - Decision Tree Classifier: {} ".format(MulticlassClassificationEvaluator(metricName='accuracy').evaluate(pred)))
    print("F1 score - Decision Tree Classifier: {} ".format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
[ ] rf = RandomForestClassifier(maxDepth = 6,numTrees = 100,featuresCol = 'features',featureSubsetStrategy = 'sqrt')
    pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,rf])
    model = pipeline.fit(train)
    pred = model.transform(test)
    print("Accuracy - Random Forest Classifier: {} ".format(MulticlassClassificationEvaluator(metricName='accuracy').evaluate(pred)))
    print("F1 score - Random Forest Classifier: {} ".format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
[ ] gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=10)
    pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,gbt])
    model = pipeline.fit(train)
    pred = model.transform(test)
    print("Accuracy - GBTClassifier: {} ".format(MulticlassClassificationEvaluator(metricName='accuracy').evaluate(pred)))
    print("F1 score - GBTClassifier: {} ".format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

```
[ ] svc = LinearSVC(labelCol="label", featuresCol="features", maxIter=10,regParam=0.1)
    pipeline =  Pipeline(stages = [assembler,normalizer,model_pca,svc])
    model = pipeline.fit(train)
    pred = model.transform(test)
    print("Accuracy - GBTClassifier: {} ".format(MulticlassClassificationEvaluator(metricName='accuracy').evaluate(pred)))
    print("F1 score - GBTClassifier: {} ".format(MulticlassClassificationEvaluator(metricName='f1').evaluate(pred)))
```

7. Results:

Results of various models have been tabulated below

Sampling \ Model	Logistic regression		Decision Tree		Random forest		GBT Classifier		SVC	
	Accuracy	**F1**	**Accuracy**	**F1**	**Accuracy**	**F1**	**Accuracy**	**F1**	**Accuracy**	**F1**
No sampling	**0.741**	**0.724**	0.669	0.677	**0.776**	**0.768**	0.683	0.686	0.661	0.527
Under sampling	0.640	0.649	0.597	0.599	0.705	0.712	0.640	0.648	0.640	0.649
Over sampling	0.33	0.17	0.33	0.17	0.33	0.17	0.33	0.17	0.33	0.17

8. **Conclusions:** Random Forest classifier without under sampling or oversampling is giving best results followed by logistic regression (without under sampling or oversampling)

Name: Royyuru Sai Prasanna Gangadhar **Roll Number: ME19B190**

Deep learning Assignment:

The DL.ipynb file uploaded on moodle uses a pre-trained mobilenet model to run inference on the flowers dataset using Pyspark. Modify the above code to run inference on CIFAR 10 dataset using Pyspark. Try a few different models pre-trained on Imagenet and report which works better

1. Install the required packages

```
!sudo add-apt-repository ppa:openjdk-r/ppa
!sudo apt-get install openjdk-11-jdk
# To Install Oracke JDK version 8
# !sudo add-apt-repository ppa:webupd8team/java
# !sudo apt-get install oracle-java8-installer
!sudo apt install -y openjdk-8-jdk
```

```
[ ] !wget -q https://downloads.apache.org/spark/spark-3.2.3/spark-3.2.3-bin-hadoop3.2.tgz
    !tar xvzf spark-3.2.3-bin-hadoop3.2.tgz
    !pip install pyspark
    !pip install -q findspark
    !pip install pyarrow
    try:
      # %tensorflow_version only exists in Colab.
      !pip install  tf-estimator-nightly==2.8.0.dev2021122109
    except Exception:
      pass
```

2. Download CIFAR 10 dataset

```
[ ] data_dir = tf.keras.utils.get_file(origin='https://mmlspark.azureedge.net/datasets/CIFAR10/train.zip',extract = True)

    print(data_dir)

    Downloading data from https://mmlspark.azureedge.net/datasets/CIFAR10/train.zip
    127708997/127708997 [==============================] - 6s 0us/step
    /root/.keras/datasets/train.zip
```

3. Store each image in the dataset as a binary array and store its path and corresponding label

```
df = images.select(
  col("path"),
  col("modificationTime"),
  extract_label(col("path")).alias("label"),
  extract_size_udf(col("content")).alias("size"),
  col("content"))

df  = df.withColumn("label",
                    when(col("label")=="00","airplanes" )
                    .when(col("label")=="01", "cars")
                    .when(col("label")=="02", "birds")
                    .when(col("label")=="03", "cats")
                    .when(col("label")=="04", "deer")
                    .when(col("label")=="05", "dogs")
                    .when(col("label")=="06", "frogs")
                    .when(col("label")=="07", "horses")
                    .when(col("label")=="09", "ships")
                    .otherwise("trucks"))
df.show(5)
```

```
+--------------------+--------------------+-----+--------+--------------------+
|                path|    modificationTime|label|    size|             content|
+--------------------+--------------------+-----+--------+--------------------+
|file:/root/.keras...|2023-04-09 15:20:...|birds|{40, 40}|[89 50 4E 47 0D 0...|
|file:/root/.keras...|2023-04-09 15:20:...|frogs|{40, 40}|[89 50 4E 47 0D 0...|
|file:/root/.keras...|2023-04-09 15:20:...| deer|{40, 40}|[89 50 4E 47 0D 0...|
|file:/root/.keras...|2023-04-09 15:20:...|frogs|{40, 40}|[89 50 4E 47 0D 0...|
|file:/root/.keras...|2023-04-09 15:20:...|frogs|{40, 40}|[89 50 4E 47 0D 0...|
+--------------------+--------------------+-----+--------+--------------------+
only showing top 5 rows
```

4. Visualizing the dataset

5. A Imagenet dataset class has been created and a method _preprocess has been defined to convert all the input images to standard ImageNet dataset sizes

```
def _preprocess(self, content):
  """
  Preprocesses the input image content using standard ImageNet normalization.

  See https://pytorch.org/docs/stable/torchvision/models.html.
  """
  image = Image.open(io.BytesIO(content))
  transform = transforms.Compose([
    transforms.Resize(256),                        ### Smaller lenght is convertd to 256
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
  ])
  return transform(image)
```

6. Predict function has been defined with returns predictions of the given model on each image

```
def predict(content_series_iter : pd.Series) -> pd.DataFrame:
  model = model_fn() # Once per Map
  model.eval()      # Once Per Map ,model file read from memory
  for content_series in content_series_iter:
    dataset = ImageNetDataset(list(content_series))
    loader = DataLoader(dataset, batch_size=64)
    with torch.no_grad():
      for image_batch in loader:
        predictions = model(image_batch).numpy()
        predicted_labels = [x[0] for x in decode_predictions(predictions, top=1)]
        yield pd.DataFrame(predicted_labels)


return_type = "class: string, desc: string, score:float"
return pandas_udf(predict,return_type, PandasUDFType.SCALAR_ITER)
```

7. Various pretrained models such as MobileNet, ResNet50, GoogleNet and VGG16 has been used to give predictions on ImageNet dataset.The below table summarizes the results of various pretrained models.

Predictions of MobileNet:

label	MobileNet prediction	MobileNet Score
birds	comic_book	9.823734
frogs	frilled_lizard	11.226815
deer	orangutan	8.612705
frogs	revolver	9.759069
frogs	waffle_iron	9.964415
frogs	book_jacket	10.09747
frogs	muzzle	11.94763
frogs	three-toed_sloth	13.096301
birds	patas	10.979576
frogs	bolo_tie	10.29219
frogs	cheetah	9.640568
frogs	muzzle	10.303909
frogs	fox_squirrel	8.597261
frogs	leopard	11.523518
birds	safety_pin	10.109551
deer	limpkin	10.380197
frogs	cheetah	9.61996
deer	fox_squirrel	12.25812
cats	English_foxhound	10.38315
frogs	cheetah	12.006193

Name: Royyuru Sai Prasanna Gangadhar **Roll Number: ME19B190**

Predictions of ResNet50 architecture:

label	ResNet50 prediction	ResNet50 Score
birds	brambling	6.5817957
frogs	tailed_frog	5.7556643
deer	tusker	5.8913913
frogs	muzzle	5.5295134
frogs	cheetah	7.1604624
frogs	screen	5.916838
frogs	shoji	4.5856814
frogs	tailed_frog	6.1878433
birds	patas	5.433097
frogs	sidewinder	5.265415
frogs	sidewinder	5.7072563
frogs	book_jacket	5.4684696
frogs	tailed_frog	6.578742
frogs	milk_can	5.7039857
birds	howler_monkey	5.801503
deer	limpkin	4.8295956
frogs	limpkin	7.5168877
deer	screen	5.94618
cats	book_jacket	5.341413
frogs	German_short-haired_pointer	6.310394

Predictions of GoogleNet :

label	GoogleNet prediction	GoogleNet Score
birds	brambling	8.454064
frogs	book_jacket	6.4626827
deer	book_jacket	6.2683506
frogs	book_jacket	6.6099167
frogs	cheetah	7.160299
frogs	screen	7.0333133
frogs	safety_pin	6.7185173
frogs	screen	5.701735
birds	patas	9.558256
frogs	bolo_tie	6.728154
frogs	book_jacket	6.834465
frogs	screen	6.812574
frogs	fox_squirrel	6.750074
frogs	screen	6.3442874
birds	screen	7.789497
deer	book_jacket	6.203725
frogs	book_jacket	6.9980235
deer	sorrel	8.453428
cats	screen	7.7593718
frogs	book_jacket	7.0712056

Predictions of VGG16 Network:

label	VGG16 prediction	VGG16 Score
birds	screen	8.6330185
frogs	frilled_lizard	11.306978
deer	bulletproof_vest	7.5320554
frogs	comic_book	9.264689
frogs	cheetah	10.472668
frogs	book_jacket	8.955809
frogs	safety_pin	8.636215
frogs	three-toed_sloth	9.986124
birds	fox_squirrel	11.699705
frogs	rock_python	8.877836
frogs	hen-of-the-woods	7.9704266
frogs	book_jacket	7.852457
frogs	fox_squirrel	13.552772
frogs	plate_rack	10.658487
birds	fox_squirrel	6.7122188
deer	barn_spider	9.036906
frogs	cheetah	6.843171
deer	fox_squirrel	8.38976
cats	Windsor_tie	8.3282795
frogs	German_short-haired_pointer	7.845697

8. Conclusions:
   ImageNet dataset has different label names than that of CIFAR10, the predicted labels of these pre trained models do not match the exact labels of CIFAR10 dataset.

- From the above predictions reported, ResNet 50 is performing better compared all other models like GoogleNet, MobileNet and VGG16.
- The reason being, it has predicted
  a. Brambling (a bird) - for the class "bird"
  b. tailed frog - for the class "frogs
  c. tusker - for the class "deer" (may it is drawing correlation between the tusks of the two classes)

- These predictions are better compared to the predictions of other models which are completely different like screen for birds(VGG16), book_jacket for frogs(GoogleNet) etc.

- Thus, we can conclude that ResNet50 is working better on CIFAR10 dataset compared to other pre-trained models which are trained on ImageNet dataset.