CO542 - NEURAL NETWORKS AND FUZZY SYSTEMS

CONVOLUTIONAL NEURAL NETWORKS (CNN)

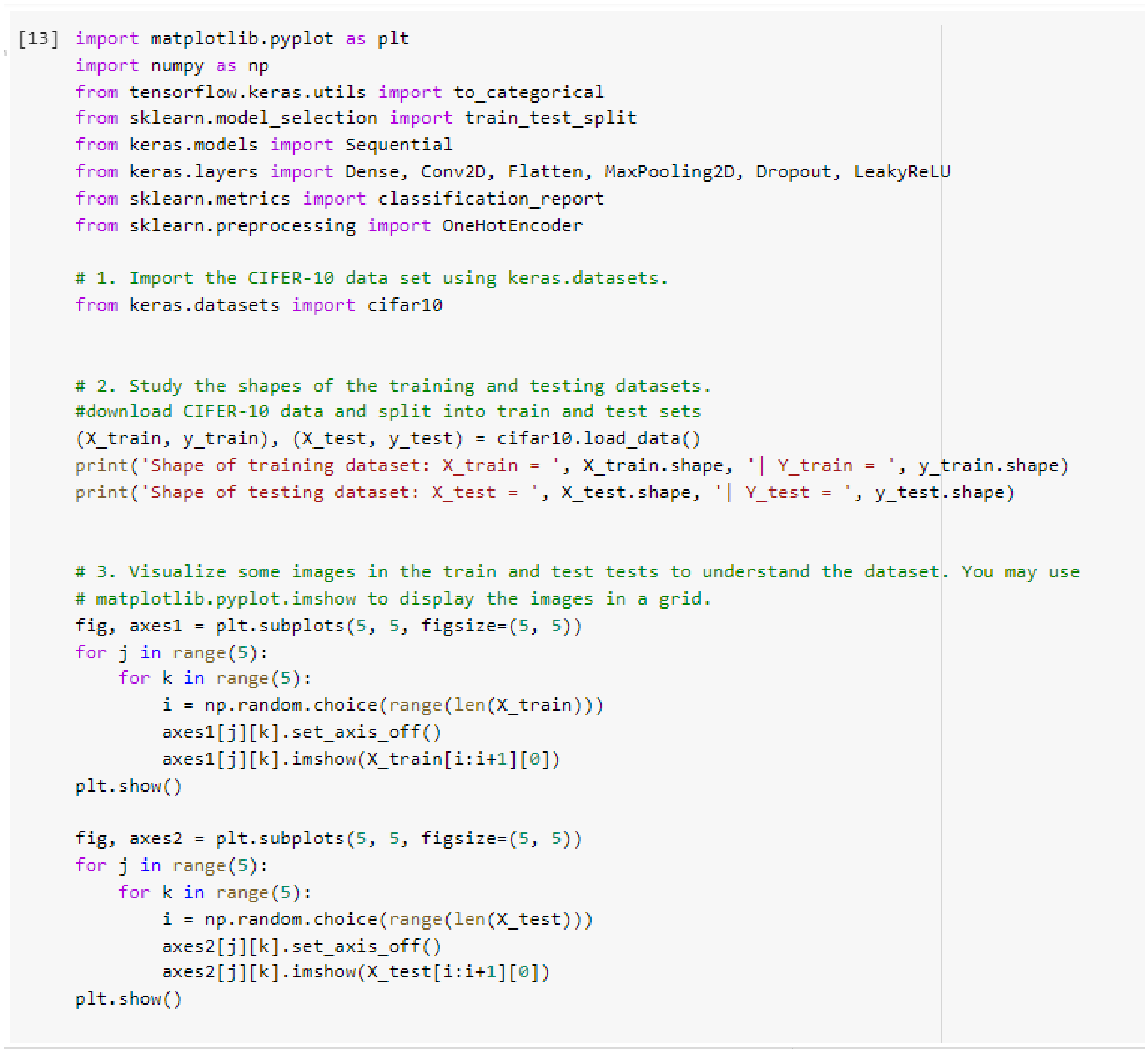
F.S.MARZOOK

E/16/232

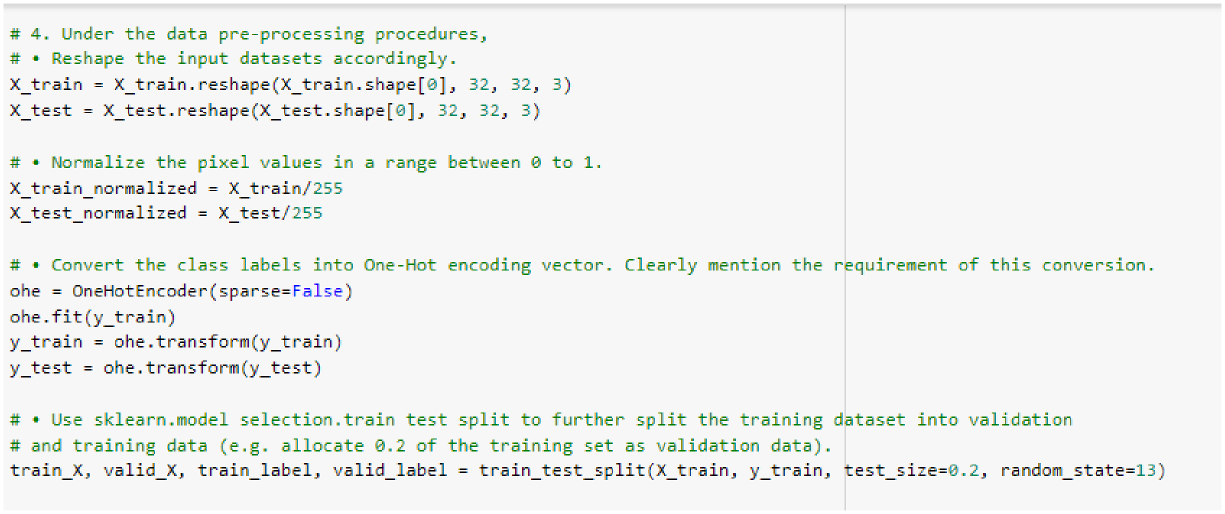
LAB 06

15/10/2021

CODE



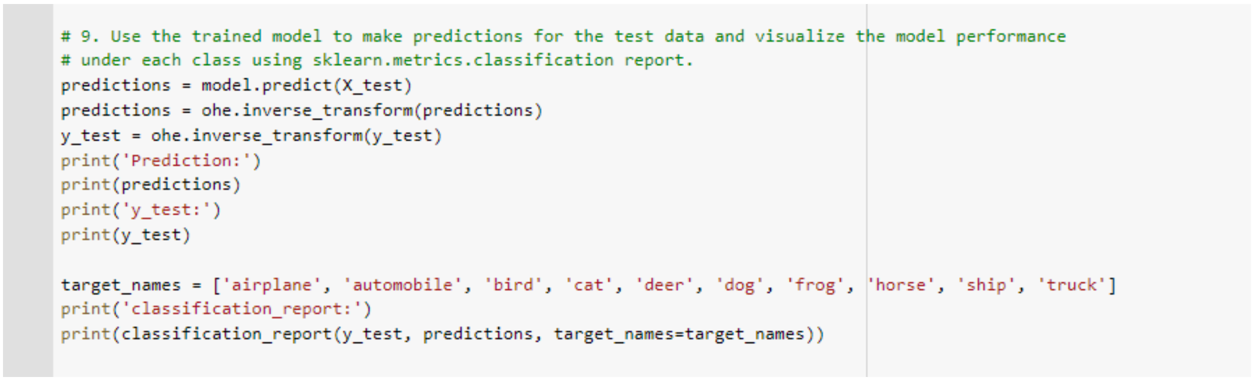
```
[13] import matplotlib.pyplot as plt
     import numpy as np
     from tensorflow.keras.utils import to_categorical
     from sklearn.model_selection import train_test_split
     from keras.models import Sequential
     from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout, LeakyReLU
     from sklearn.metrics import classification_report
     from sklearn.preprocessing import OneHotEncoder

     # 1. Import the CIFER-10 data set using keras.datasets.
     from keras.datasets import cifar10


     # 2. Study the shapes of the training and testing datasets.
     #download CIFER-10 data and split into train and test sets
     (X_train, y_train), (X_test, y_test) = cifar10.load_data()
     print('Shape of training dataset: X_train = ', X_train.shape, '| Y_train = ', y_train.shape)
     print('Shape of testing dataset: X_test = ', X_test.shape, '| Y_test = ', y_test.shape)


     # 3. Visualize some images in the train and test tests to understand the dataset. You may use
     # matplotlib.pyplot.imshow to display the images in a grid.
     fig, axes1 = plt.subplots(5, 5, figsize=(5, 5))
     for j in range(5):
         for k in range(5):
             i = np.random.choice(range(len(X_train)))
             axes1[j][k].set_axis_off()
             axes1[j][k].imshow(X_train[i:i+1][0])
     plt.show()

     fig, axes2 = plt.subplots(5, 5, figsize=(5, 5))
     for j in range(5):
         for k in range(5):
             i = np.random.choice(range(len(X_test)))
             axes2[j][k].set_axis_off()
             axes2[j][k].imshow(X_test[i:i+1][0])
     plt.show()
```



```
# 4. Under the data pre-processing procedures,
# • Reshape the input datasets accordingly.
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)

# • Normalize the pixel values in a range between 0 to 1.
X_train_normalized = X_train/255
X_test_normalized = X_test/255

# • Convert the class labels into One-Hot encoding vector. Clearly mention the requirement of this conversion.
ohe = OneHotEncoder(sparse=False)
ohe.fit(y_train)
y_train = ohe.transform(y_train)
y_test = ohe.transform(y_test)

# • Use sklearn.model selection.train test split to further split the training dataset into validation
# and training data (e.g. allocate 0.2 of the training set as validation data).
train_X, valid_X, train_label, valid_label = train_test_split(X_train, y_train, test_size=0.2, random_state=13)
```

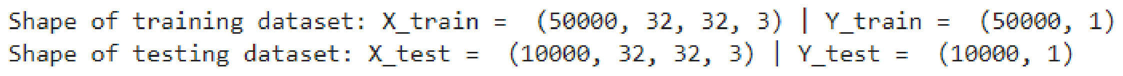
```python
# 5. Build the CNN model with three convolutional layers followed by a dense layer and an output layer
# accordingly. In this case,
# • Select 3 X 3 as the kernal size of each filter.
# • Use different number of filters in each convolutional layer (e.g. first layer 32 filters, second
# layer 64 filters, third layer 128 filters).
# • Use LeakyReLU as the activation function. Mention the advantage of using LeakyReLU over
# ReLU activation function.
# • Use 2 X 2 MaxPooling layers, and Dropout layers according to the requirements and mention
# the purpose behind the usage of Dropout Layers.
#create model
model = Sequential()
#add model layers
model.add(Conv2D(32, kernel_size=(3, 3), activation=LeakyReLU(alpha=0.1), input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=(3, 3), activation=LeakyReLU(alpha=0.1)))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=(3, 3), activation=LeakyReLU(alpha=0.1)))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Dropout(0.25))
```

```python
model.add(Flatten())
model.add(Dense(10, activation='softmax'))


# 6. Compile the model using appropriate parameters and generate the model summery using
# model.summary() function (In this case make sure to specify the metrics as accuracy).
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
summary = model.summary()
print(summary)


# 7. Train the compiled model using model.fit function and observe the train and validation set performances.
# In this case, you may have to select an appropriate number of epochs (e.g. 25) and
# batch size (e.g. 64, 128 or 256).
model_hostory = model.fit(train_X, train_label, validation_data=(valid_X, valid_label), epochs=35, batch_size=64)

fig, axs = plt.subplots(1, 2, figsize=(15, 5))

# summarize history for accuracy
axs[0].plot(model_hostory.history['accuracy'])
axs[0].plot(model_hostory.history['val_accuracy'])
axs[0].set_title('Model Accuracy')
axs[0].set_ylabel('Accuracy')
axs[0].set_xlabel('Epoch')
axs[0].legend(['train', 'validate'], loc='upper left')

# summarize history for loss
axs[1].plot(model_hostory.history['loss'])
axs[1].plot(model_hostory.history['val_loss'])
axs[1].set_title('Model Loss')
axs[1].set_ylabel('Loss')
axs[1].set_xlabel('Epoch')
axs[1].legend(['train', 'validate'], loc='upper right')
plt.show()


# 8. Evaluate the model performance using test set. Identify the test loss and test accuracy.
test_eval = model.evaluate(X_test, y_test, verbose=0)
print("Test loss:", test_eval[0])
print("Test accuracy:", test_eval[1])
```



```
# 9. Use the trained model to make predictions for the test data and visualize the model performance
# under each class using sklearn.metrics.classification report.
predictions = model.predict(X_test)
predictions = ohe.inverse_transform(predictions)
y_test = ohe.inverse_transform(y_test)
print('Prediction:')
print(predictions)
print('y_test:')
print(y_test)

target_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
print('classification_report:')
print(classification_report(y_test, predictions, target_names=target_names))
```

2. Shape of the training and testing dataset:



```
Shape of training dataset: X_train =  (50000, 32, 32, 3) | Y_train =  (50000, 1)
Shape of testing dataset: X_test =  (10000, 32, 32, 3) | Y_test =  (10000, 1)
```

3.

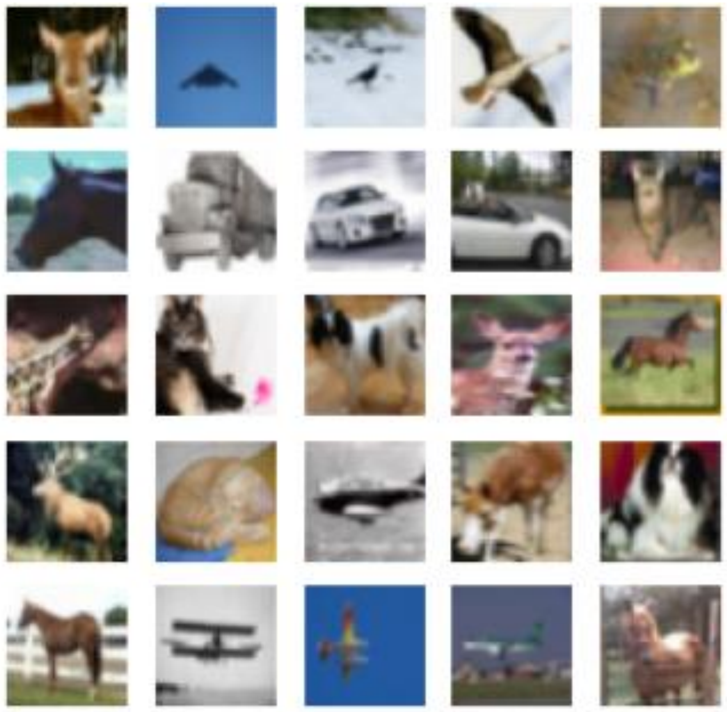


Figure 1: Some Images in Train Set

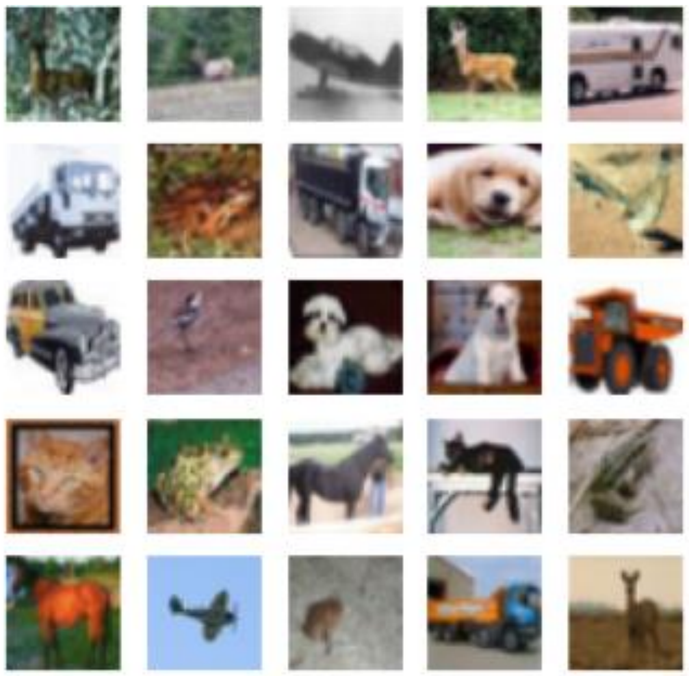


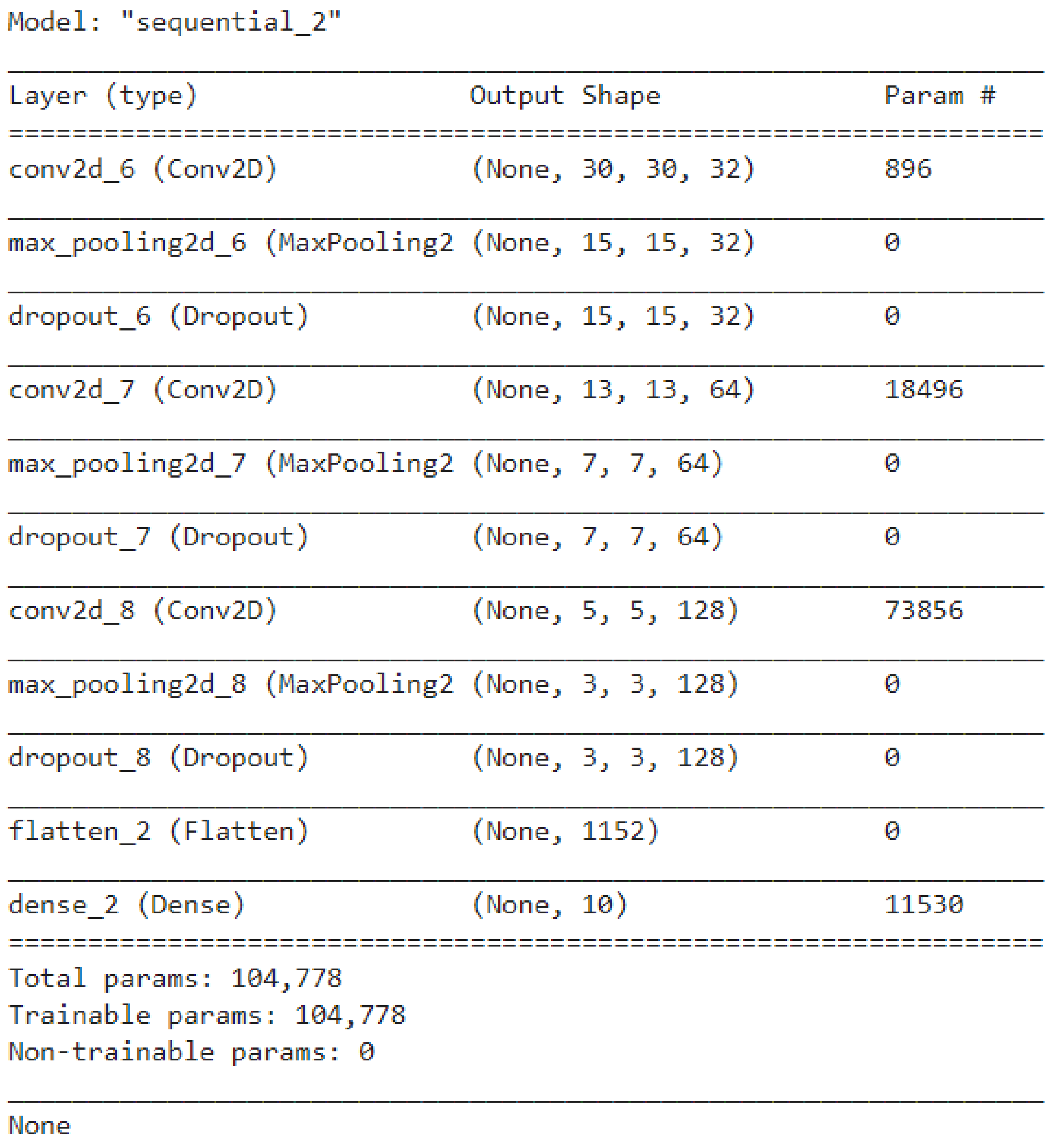
Figure 2: Some Images in Test Set

4. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical. For categorical variables where no such ordinal relationship exists, the integer encoding is not enough. In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

5. Advantage of Leaky ReLU over ReLU:
   With a Leaky ReLU (LReLU), we won't face the "dead ReLU" problem which happens when the ReLU always have values under 0 - this completely blocks learning in the ReLU because of gradients of 0 in the negative part. The derivative of the ReLU is 1 in the positive part, and 0 in the negative part. The derivative of the LReLU is 1 in the positive part, and is a small fraction in the negative part.

   Purpose behind the usage of Dropout Layer:
   Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase.

6.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 30, 30, 32)        896       
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 15, 15, 32)        0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 15, 15, 32)        0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 13, 13, 64)        18496     
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 64)          0         
_________________________________________________________________
dropout_7 (Dropout)          (None, 7, 7, 64)          0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 5, 5, 128)         73856     
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 3, 3, 128)         0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 3, 3, 128)         0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 1152)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                11530     
=================================================================
Total params: 104,778
Trainable params: 104,778
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 3: Model Summary

7.

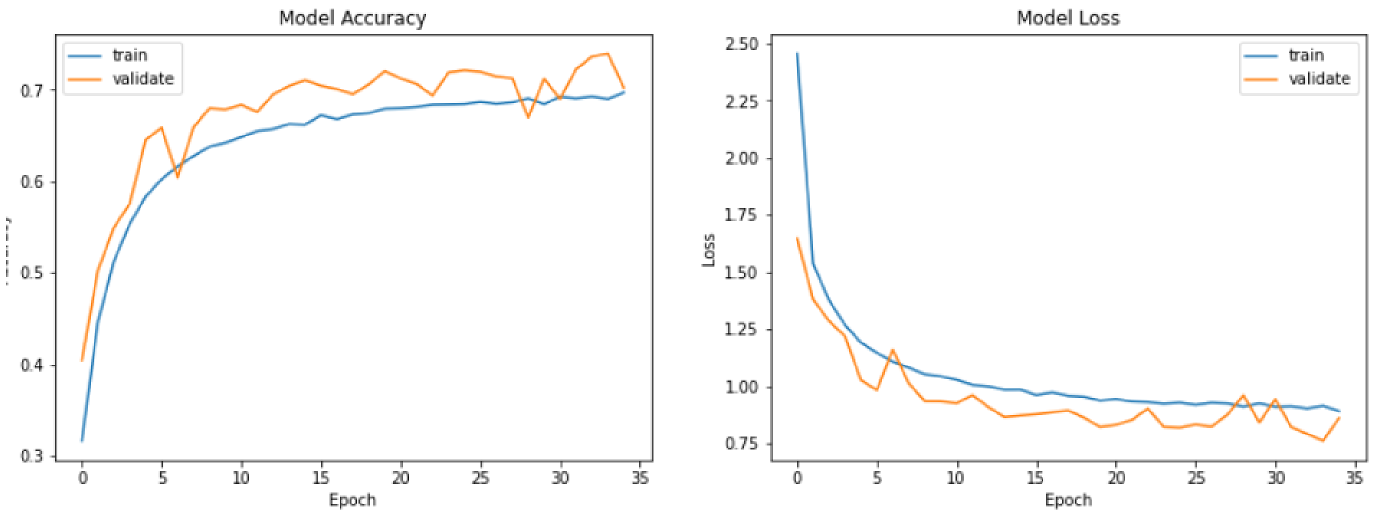
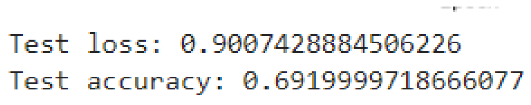


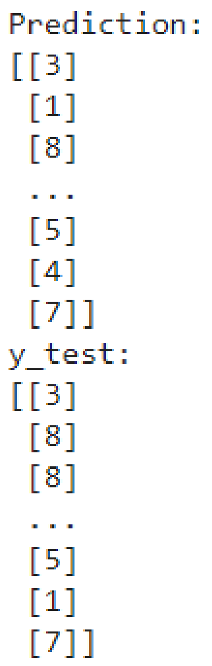
Figure 4: Train and Validation Set Performances

8. Test loss and Test accuracy:



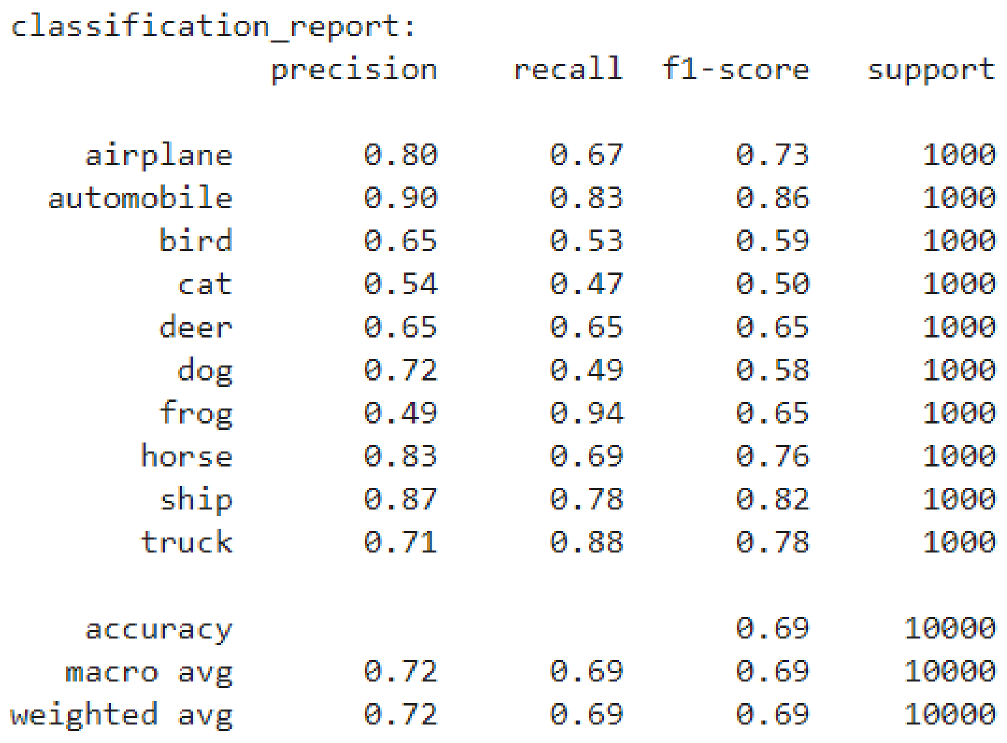
```
Test loss: 0.9007428884506226
Test accuracy: 0.6919999718666077
```

9. Prediction:



```
Prediction:
[[3]
 [1]
 [8]
 ...
 [5]
 [4]
 [7]]
y_test:
[[3]
 [8]
 [8]
 ...
 [5]
 [1]
 [7]]
```

Classification Report:



```
classification_report:
              precision    recall  f1-score   support

    airplane       0.80      0.67      0.73      1000
  automobile       0.90      0.83      0.86      1000
        bird       0.65      0.53      0.59      1000
         cat       0.54      0.47      0.50      1000
        deer       0.65      0.65      0.65      1000
         dog       0.72      0.49      0.58      1000
        frog       0.49      0.94      0.65      1000
       horse       0.83      0.69      0.76      1000
        ship       0.87      0.78      0.82      1000
       truck       0.71      0.88      0.78      1000

    accuracy                           0.69     10000
   macro avg       0.72      0.69      0.69     10000
weighted avg       0.72      0.69      0.69     10000
```