Main

[ ]:
```
import pandas as pd
import numpy as np
import itertools
import keras
import tensorflow as tf


from sklearn import metrics
from sklearn.metrics import confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator,␣
 ↪img_to_array, load_img
from tensorflow.keras.models import Sequential
from tensorflow.keras import optimizers
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras import applications
from keras.utils.np_utils import to_categorical


import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
import math
import datetime
import time

import warnings
warnings.filterwarnings("ignore")
```

[ ]:
```
#Default dimensions we found online
img_width, img_height = 224, 224

#Create a bottleneck file
top_model_weights_path = 'bottleneck_fc_model.h5'


# loading up our datasets
```

```
train_data_dir = 'C:/Users/arman/OneDrive/School/Spring 2022/ML/Final Project/
↪blood cell dataset/dataset2-master/dataset2-master/images/TRAIN'
validation_data_dir = "C:/Users/arman/OneDrive/School/Spring 2022/ML/Final␣
↪Project/blood cell dataset/dataset2-master/dataset2-master/images/VAL"
test_data_dir = "C:/Users/arman/OneDrive/School/Spring 2022/ML/Final Project/
↪blood cell dataset/dataset2-master/dataset2-master/images/TEST"

# number of epochs to train top model

# batch size used by flow_from_directory and predict_generator
batch_size = 50
```

[ ]:

```
# Loading vgc16 model
# Here we're making use of transfer learning!? This is a great tool to make use␣
↪of when building CNN. We can make use of a prebuilt model to help provide␣
↪some foundation to our model.

vgg16 = applications.VGG16(include_top=False, weights='imagenet')
datagen = ImageDataGenerator(rescale=1. / 255) # needed to create the␣
↪bottleneck .npy files
```

[ ]:

```
# Loading resnet50 model
# Here we're making use of transfer learning!? This is a great tool to make use␣
↪of when building CNN. We can make use of a prebuilt model to help provide␣
↪some foundation to our model.

# resnet50 = applications.ResNet50V2(include_top=False, weights='imagenet')
# datagen = ImageDataGenerator(rescale=1. / 255) # needed to create the␣
↪bottleneck .npy files

# RUNNING OUT OF MEMORY (OOM) WHEN USING RESNET50V2 AND AN RTX 3070!? What!?
```

[ ]:

```
# TRAINING HERE - making use of transfer training to create our .npy files for␣
↪our data preparation below

start = datetime.datetime.now()

generator = datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_train_samples = len(generator.filenames)
num_classes = len(generator.class_indices)
```

```
predict_size_train = int(math.ceil(nb_train_samples / batch_size))

bottleneck_features_train = vgg16.predict_generator(generator,␣
 ↪predict_size_train)
# bottleneck_features_train = resnet50.predict_generator(generator,␣
 ↪predict_size_train)

np.save('bottleneck_features_train.npy', bottleneck_features_train)
end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)
```

```
Found 9957 images belonging to 4 classes.
Time:  0:00:23.749635
```

[ ]:
```
# TESTING HERE - making use of transfer training to create our .npy files for␣
 ↪our data preparation below

start = datetime.datetime.now()

generator = datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_test_samples = len(generator.filenames)
num_classes = len(generator.class_indices)

predict_size_test = int(math.ceil(nb_test_samples / batch_size))

bottleneck_features_test = vgg16.predict_generator(generator, predict_size_test)
# bottleneck_features_test = resnet50.predict_generator(generator,␣
 ↪predict_size_test)

np.save('bottleneck_features_test.npy', bottleneck_features_test)
end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)
```

```
Found 2239 images belonging to 4 classes.
Time:  0:00:06.330560
```

[ ]:
```
# VALIDATION HERE - making use of transfer training to create our .npy files␣
 ↪for our data preparation below

start = datetime.datetime.now()
```

```
generator = datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_val_samples = len(generator.filenames)
num_classes = len(generator.class_indices)

predict_size_val = int(math.ceil(nb_val_samples / batch_size))

bottleneck_features_val = vgg16.predict_generator(generator, predict_size_val)
# bottleneck_features_val = resnet50.predict_generator(generator,␣
 ↪predict_size_val)

np.save('bottleneck_features_val.npy', bottleneck_features_val)
end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)
```

```
Found 248 images belonging to 4 classes.
Time:  0:00:02.942569
```

[ ]:
```
# TRAINING DATA - preparing our training data for our CNN below

generator_top = datagen.flow_from_directory(
   train_data_dir,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='categorical',
   shuffle=False)

nb_train_samples = len(generator_top.filenames)
num_classes = len(generator_top.class_indices)

# load the bottleneck features saved earlier
train_data = np.load('bottleneck_features_train.npy')

# get the class labels for the training data, in the original order
train_labels = generator_top.classes

# convert the training labels to categorical vectors
train_labels = to_categorical(train_labels, num_classes=num_classes)

print(len(train_labels))
```

```
print(len(train_data))
```

```
Found 9957 images belonging to 4 classes.
9957
9957
```

[ ]:
```
# TESTING DATA - preparing our testing data for our CNN below

generator_top = datagen.flow_from_directory(
   test_data_dir,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='categorical',
   shuffle=False)

nb_test_samples = len(generator_top.filenames)
num_classes = len(generator_top.class_indices)

# load the bottleneck features saved earlier
test_data = np.load('bottleneck_features_test.npy')

# get the class labels for the testing data, in the original order
test_labels = generator_top.classes

# convert the testing labels to categorical vectors
test_labels = to_categorical(test_labels, num_classes=num_classes)

print(len(test_labels))
print(len(test_data))
```

```
Found 2239 images belonging to 4 classes.
2239
2239
```

[ ]:
```
# VALIDATION - preparing our validation data for our CNN below

generator_top = datagen.flow_from_directory(
   validation_data_dir,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='categorical',
   shuffle=False)

nb_val_samples = len(generator_top.filenames)
num_classes = len(generator_top.class_indices)

# load the bottleneck features saved earlier
val_data = np.load('bottleneck_features_val.npy')
```

```
# get the class labels for the validation data, in the original order
val_labels = generator_top.classes

# convert the validation labels to categorical vectors
val_labels = to_categorical(val_labels, num_classes=num_classes)

print(len(val_labels))
print(len(val_data))
```

```
Found 248 images belonging to 4 classes.
248
248
```

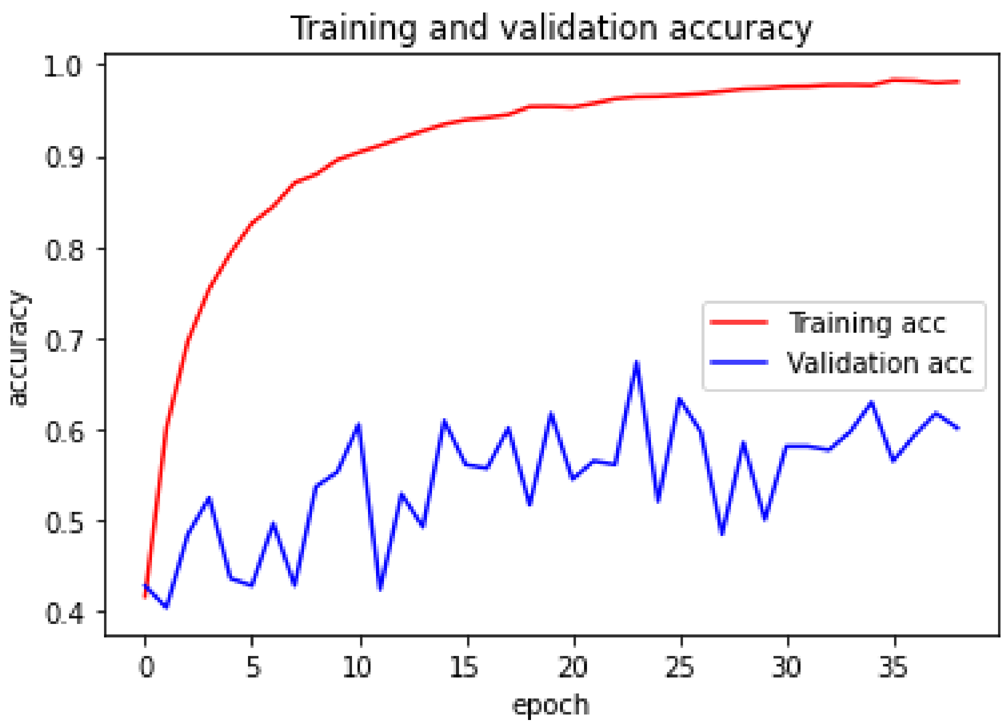
[ ]:
```
# Setting up callback for the different models below
# EarlyStopping with patience 3 allows Keras to stop at an epoch count within␣
↪the epoch count given- 100- when the accuracy of a given epoch doesn't␣
↪improve after at least 3 epochs

callback = tf.keras.callbacks.EarlyStopping(monitor = "acc", patience = 3)
```

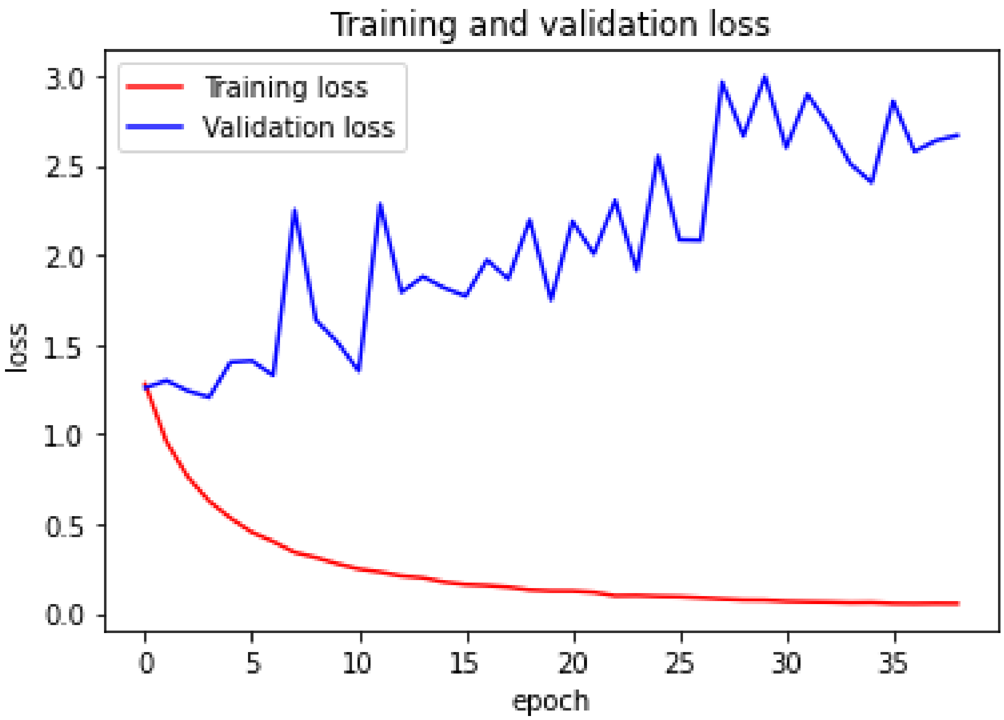
0.0.1 Base Model

[ ]:
```
# This is our base model!

start = datetime.datetime.now()

modelBase = Sequential()

modelBase.add(Flatten(input_shape=train_data.shape[1:]))
modelBase.add(Dense(100, activation=keras.layers.LeakyReLU(alpha=0.3)))
modelBase.add(Dropout(0.5))
modelBase.add(Dense(50, activation=keras.layers.LeakyReLU(alpha=0.3)))
modelBase.add(Dropout(0.3))
modelBase.add(Dense(num_classes, activation='softmax'))


modelBase.compile(optimizer = optimizers.RMSprop(lr=1e-4),␣
↪loss='categorical_crossentropy', metrics=['acc'])


model_output = modelBase.fit(train_data, train_labels, epochs=100,␣
↪batch_size=32, callbacks = [callback], validation_data=(val_data,␣
↪val_labels))


modelBase.save_weights(top_model_weights_path)
```

```python
(eval_loss, eval_accuracy) = modelBase.evaluate(val_data, val_labels) #,␣
↪batch_size=32, verbose=1)


print("[INFO] accuracy: {:.2f}%".format(eval_accuracy * 100))
print("[INFO] Loss: {}".format(eval_loss))
print("[INFO] EarlyStopping Epoch Count:", len(model_output.history["acc"]))


end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)


# Graphing our training and validation

acc = model_output.history['acc']
val_acc = model_output.history['val_acc']
loss = model_output.history['loss']
val_loss = model_output.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```

```
[INFO] accuracy: 60.08%
[INFO] Loss: 2.6685335636138916
[INFO] EarlyStopping Epoch Count: 39
Time:  0:00:39.187392
```

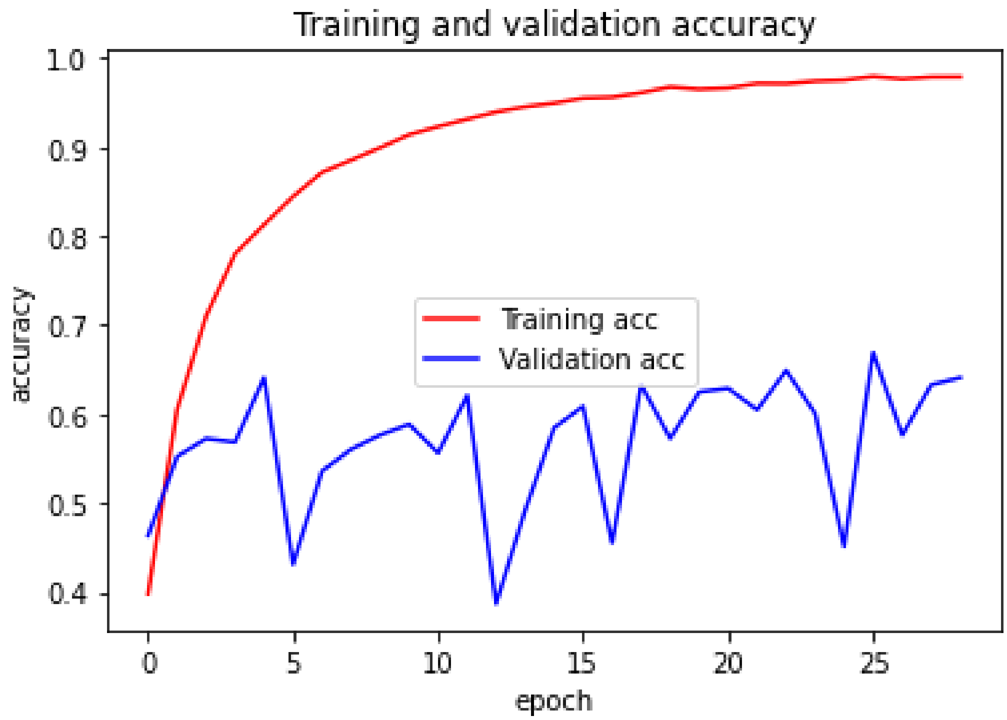


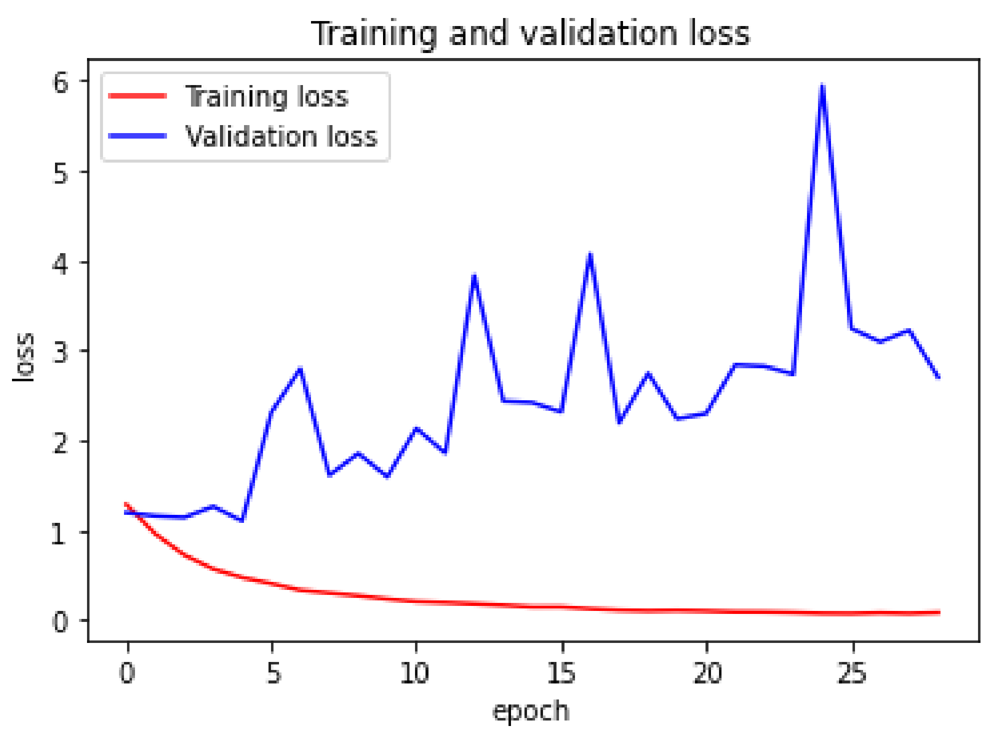


0.0.2 4 Hidden Layer CNN

[ ]:
```
# This is a new variant of our base model! We're going to add some extra layers␣
↪here!

start = datetime.datetime.now()

model4 = Sequential()

model4.add(Flatten(input_shape=train_data.shape[1:]))
model4.add(Dense(256, activation=keras.layers.LeakyReLU(alpha=0.3)))
model4.add(Dropout(0.2))
model4.add(Dense(128, activation=keras.layers.LeakyReLU(alpha=0.3)))
model4.add(Dropout(0.2))
model4.add(Dense(64, activation=keras.layers.LeakyReLU(alpha=0.3)))
model4.add(Dropout(0.2))
model4.add(Dense(32, activation=keras.layers.LeakyReLU(alpha=0.3)))
model4.add(Dropout(0.2))

model4.add(Dense(num_classes, activation='softmax'))
```

```
model4.compile(optimizer = optimizers.RMSprop(lr=1e-4),␣
↪loss='categorical_crossentropy', metrics=['acc'])


model_output = model4.fit(train_data, train_labels, epochs=100, batch_size=32,␣
↪callbacks = [callback], validation_data=(val_data, val_labels))


model4.save_weights(top_model_weights_path)


(eval_loss, eval_accuracy) = model4.evaluate(val_data, val_labels) # ,␣
↪batch_size=32, verbose=1)


print("[INFO] accuracy: {:.2f}%".format(eval_accuracy * 100))
print("[INFO] Loss: {}".format(eval_loss))
print("[INFO] EarlyStopping Epoch Count:", len(model_output.history["acc"]))


end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)


# Graphing our training and validation

acc = model_output.history['acc']
val_acc = model_output.history['val_acc']
loss = model_output.history['loss']
val_loss = model_output.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```

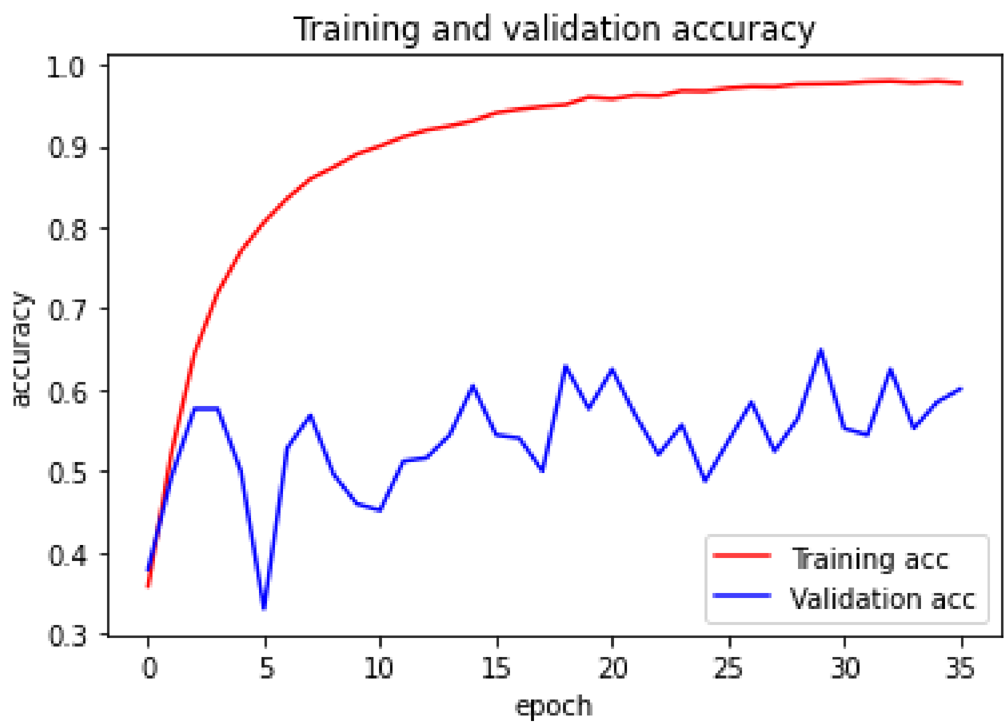

Training and validation accuracy
Training acc
Validation acc
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
5
10
15
20
25

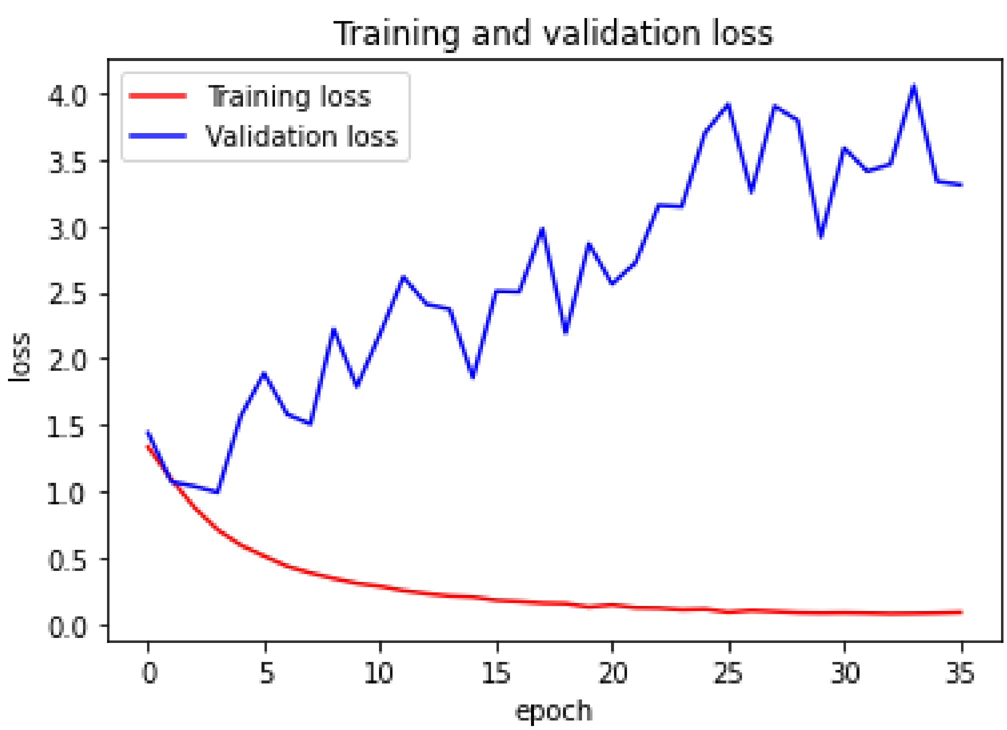

Training and validation loss
Training loss
Validation loss
loss
epoch
6
5
4
3
2
1
0
0
5
10
15
20
25

5 Hidden Layer CNN

[ ]:
```
# This is a new variant of our base model! We're going to add some extra layers␣
↪here!

start = datetime.datetime.now()

model5 = Sequential()

model5.add(Flatten(input_shape=train_data.shape[1:]))
model5.add(Dense(256, activation=keras.layers.LeakyReLU(alpha=0.3)))
model5.add(Dropout(0.2))
model5.add(Dense(128, activation=keras.layers.LeakyReLU(alpha=0.3)))
model5.add(Dropout(0.2))
model5.add(Dense(64, activation=keras.layers.LeakyReLU(alpha=0.3)))
model5.add(Dropout(0.2))
model5.add(Dense(32, activation=keras.layers.LeakyReLU(alpha=0.3)))
model5.add(Dropout(0.2))
model5.add(Dense(16, activation=keras.layers.LeakyReLU(alpha=0.3)))
model5.add(Dropout(0.2))

model5.add(Dense(num_classes, activation='softmax'))



model5.compile(optimizer = optimizers.RMSprop(lr=1e-4),␣
↪loss='categorical_crossentropy', metrics=['acc'])


model_output = model5.fit(train_data, train_labels, epochs=100, batch_size=32,␣
↪callbacks = [callback], validation_data=(val_data, val_labels))


model5.save_weights(top_model_weights_path)


(eval_loss, eval_accuracy) = model5.evaluate(val_data, val_labels) # ,␣
↪batch_size=32, verbose=1)


print("[INFO] accuracy: {:.2f}%".format(eval_accuracy * 100))
print("[INFO] Loss: {}".format(eval_loss))
print("[INFO] EarlyStopping Epoch Count:", len(model_output.history["acc"]))


end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)
```

```
# Graphing our training and validation

acc = model_output.history['acc']
val_acc = model_output.history['val_acc']
loss = model_output.history['loss']
val_loss = model_output.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```


Training and validation accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
accuracy
0
5
10
15
20
25
30
35
epoch
Training acc
Validation acc


Training and validation loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
loss
0
5
10
15
20
25
30
35
epoch
Training loss
Validation loss

[ ]:
```
preds = np.round(model5.predict(test_data), 0)
# print('round test_labels', preds)

bloodcells = ['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']
classification_metrics = metrics.classification_report(test_labels, preds,␣
↪target_names = bloodcells)
print(classification_metrics)
```

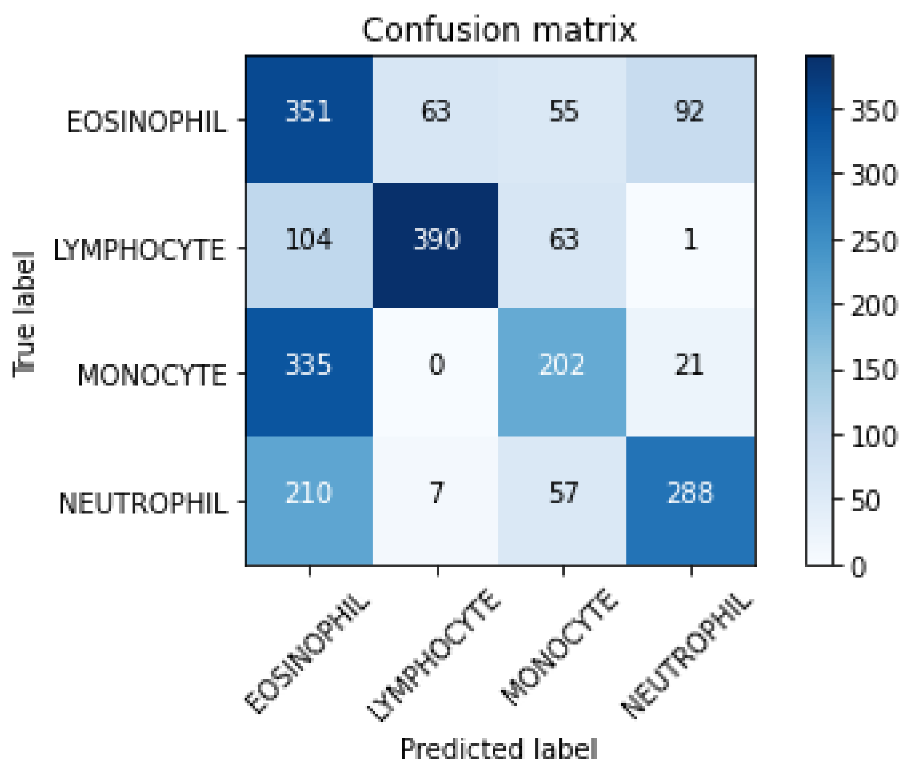
```
              precision    recall  f1-score   support

  EOSINOPHIL       0.37      0.49      0.42       561
  LYMPHOCYTE       0.84      0.77      0.80       558
    MONOCYTE       0.54      0.32      0.40       558
  NEUTROPHIL       0.62      0.68      0.65       562

   micro avg       0.57      0.56      0.57      2239
   macro avg       0.59      0.56      0.57      2239
weighted avg       0.59      0.56      0.57      2239
 samples avg       0.56      0.56      0.56      2239
```

[ ]:
```
# Converting our data from it's numeric form into a form we can assign␣
↪predictive labels too!

categorical_test_labels = pd.DataFrame(test_labels).idxmax(axis=1)
categorical_preds = pd.DataFrame(preds).idxmax(axis=1)

confusion_matrix = confusion_matrix(categorical_test_labels, categorical_preds)

#To get better visual of the confusion matrix:
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion␣
↪matrix', cmap=plt.cm.Blues):

#Add Normalization Option
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print('Normalized confusion matrix')
    else:
        print('Confusion matrix, without normalization')

# print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
```

```
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), horizontalalignment='center',␣
 ↪color='white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

[ ]:
```
plot_confusion_matrix(confusion_matrix, ['EOSINOPHIL', 'LYMPHOCYTE',␣
 ↪'MONOCYTE', 'NEUTROPHIL'], normalize = False)
```

```
Confusion matrix, without normalization
```



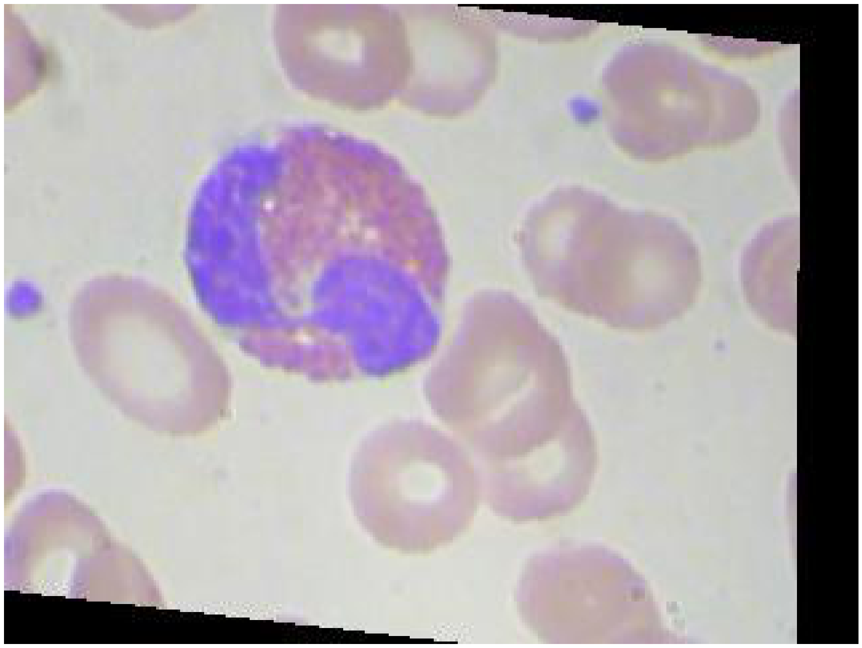
[ ]:
```
def read_image(file_path):
    print('[INFO] loading and preprocessing image…')
    image = load_img(file_path, target_size=(224, 224))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
```

```
    image /= 255.
    return image

def test_single_image(path):
  BC = ['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']
  images = read_image(path)
  time.sleep(.5)
  bt_prediction = vgg16.predict(images)
  preds = model5.predict(bt_prediction)
  for idx, BCs, x in zip(range(0,6), BC , preds[0]):
    print('ID: {}, Label: {} {}%'.format(idx, BCs, round(x*100,2) ))
  print('Final Decision:')
  time.sleep(.5)
  for x in range(3):
    print('.'*(x+1))
    time.sleep(.2)
  class_predicted = model5.predict(bt_prediction)
  class_dictionary = generator_top.class_indices
  # inv_map = {v: k for k, v in class_dictionary.items()}
  # print('ID: {}, Label: {}'.format(class_predicted[0], 
↪inv_map[class_predicted[0]]))
  # print('ID: {}, Label: {}'.format(class_predicted, inv_map))
  return load_img(path)
```

[ ]:
```
# Testing EOSINOPHIL classification

path = "C:/Users/arman/OneDrive/School/Spring 2022/ML/Final Project/CHECK/EOS.
↪jpeg"
test_single_image(path)
```

```
[INFO] loading and preprocessing image…
ID: 0, Label: EOSINOPHIL 100.0%
ID: 1, Label: LYMPHOCYTE 0.0%
ID: 2, Label: MONOCYTE 0.0%
ID: 3, Label: NEUTROPHIL 0.0%
Final Decision:
.
..
…
```

[ ]:

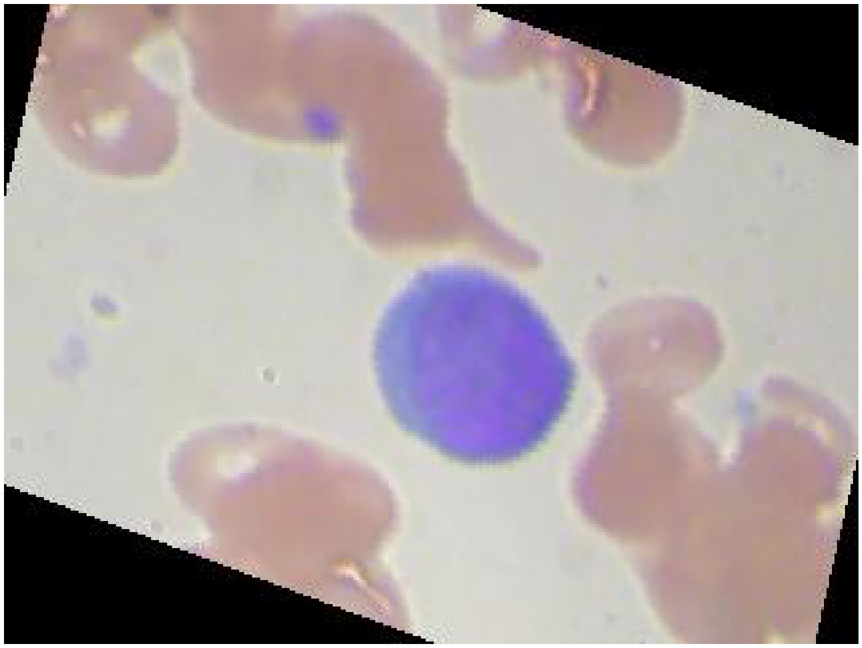


Guessed this was an eosinophile with 100.00% confidence.

[ ]:
```
# Testing LYMPHOCYTE classification

path = "C:/Users/arman/OneDrive/School/Spring 2022/ML/Final Project/CHECK/LYMPH.
↪jpeg"
test_single_image(path)
```

```
[INFO] loading and preprocessing image…
ID: 0, Label: EOSINOPHIL 0.0%
ID: 1, Label: LYMPHOCYTE 100.0%
ID: 2, Label: MONOCYTE 0.0%
ID: 3, Label: NEUTROPHIL 0.0%
Final Decision:
.
..
…
```

[ ]:

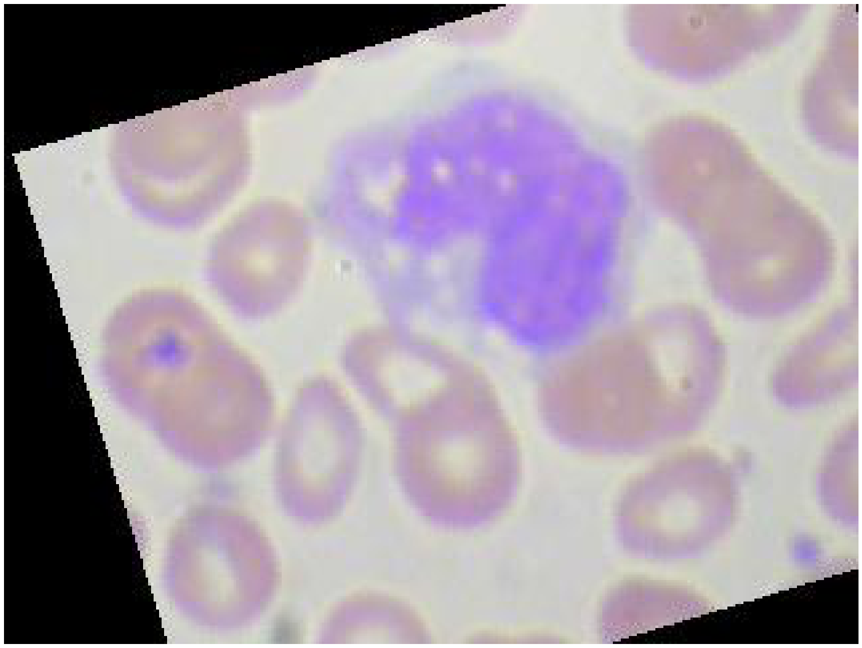


Guessed this was a lymphocyte with 100.00% confidence.

[ ]:
```
# Testing MONOCYTE classification

path = "C:/Users/arman/OneDrive/School/Spring 2022/ML/Final Project/CHECK/MONO.
 ↪jpeg"
test_single_image(path)
```

```
[INFO] loading and preprocessing image…
ID: 0, Label: EOSINOPHIL 56.11%
ID: 1, Label: LYMPHOCYTE 0.06%
ID: 2, Label: MONOCYTE 31.76%
ID: 3, Label: NEUTROPHIL 12.07%
Final Decision:
.
..
…
```

[ ]:

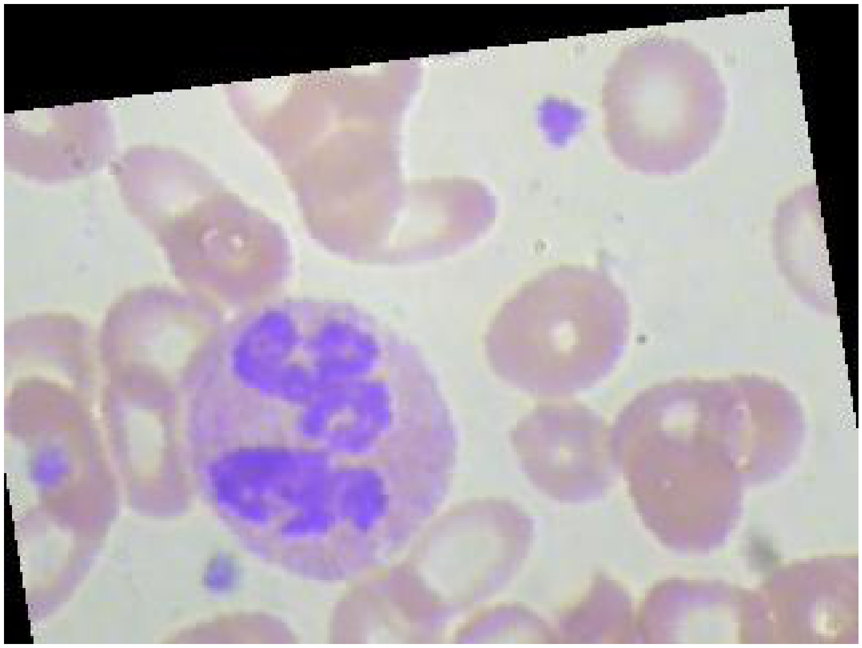


Guessed this was a monocyte with 31.76% confidence and 56.11% confidence this is an eosinphil. The model clearly has a hard time prediciting the classification of monocytes.

[ ]:
```
# Testing NEUTROPHIL classification

path = "C:/Users/arman/OneDrive/School/Spring 2022/ML/Final Project/CHECK/NEU.
↪jpeg"
test_single_image(path)
```

```
[INFO] loading and preprocessing image…
ID: 0, Label: EOSINOPHIL 32.76%
ID: 1, Label: LYMPHOCYTE 0.1%
ID: 2, Label: MONOCYTE 0.28%
ID: 3, Label: NEUTROPHIL 66.86%
Final Decision:
.
..
…
```

[ ]:



Guessed this image was a neutrophil with 66.86% confidence.