Main

May 1, 2022

[ ]:
```
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)
```

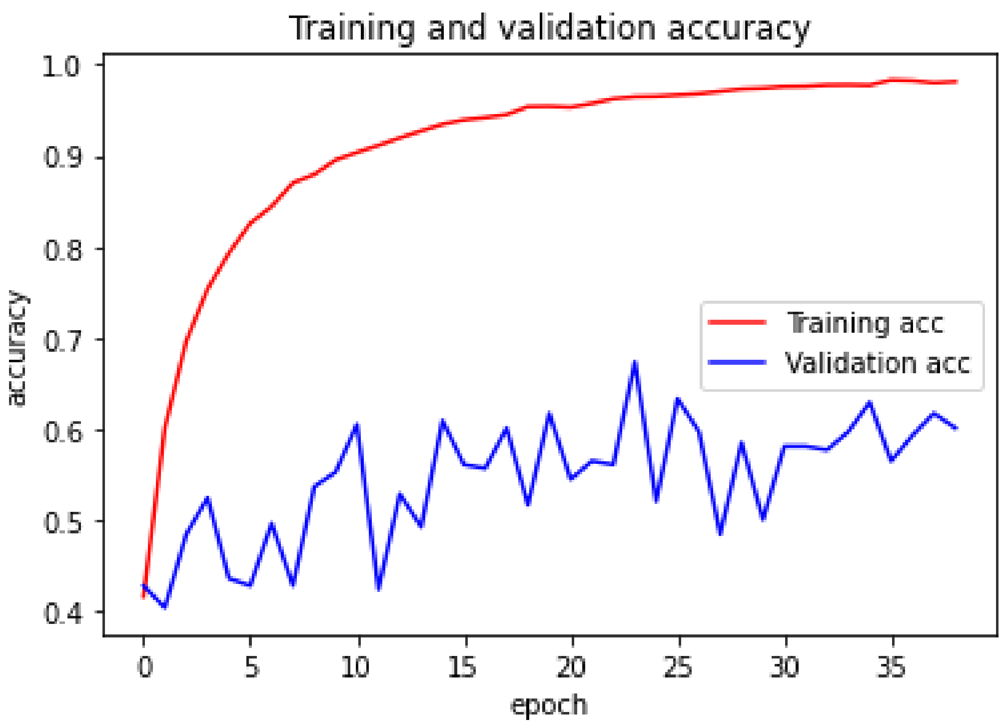
```
(eval_loss, eval_accuracy) = modelBase.evaluate(val_data, val_labels) #,␣
↪batch_size=32, verbose=1)


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()
```

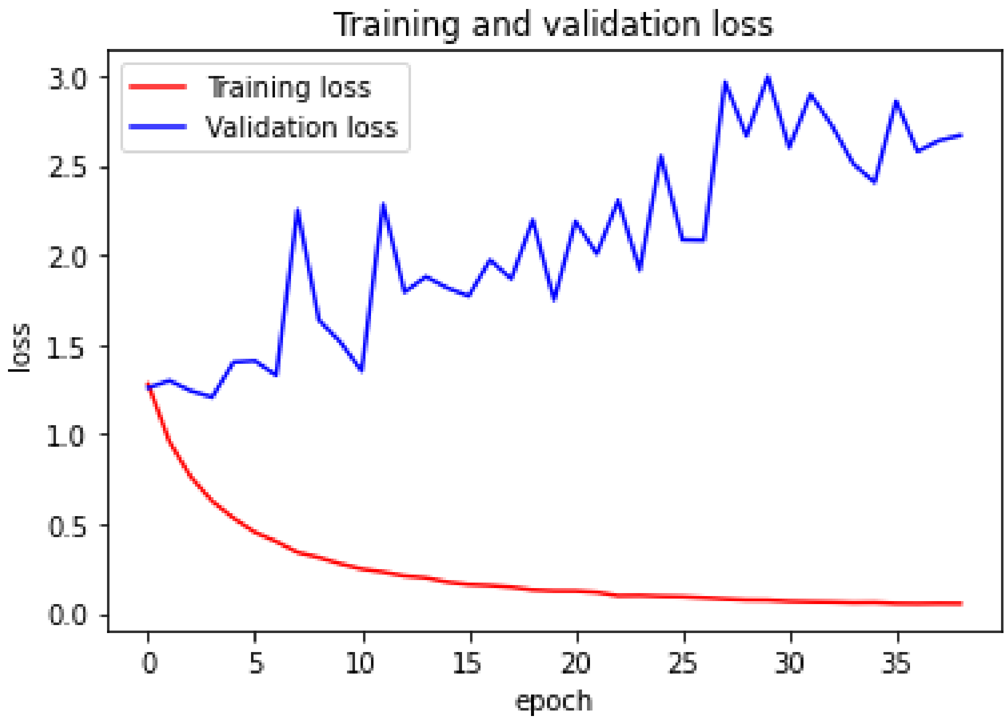
```
Epoch 1/100
312/312 [==============================] - 1s 4ms/step - loss: 1.2815 - acc:
0.4156 - val_loss: 1.2597 - val_acc: 0.4274
Epoch 2/100
312/312 [==============================] - 1s 3ms/step - loss: 0.9585 - acc:
0.6011 - val_loss: 1.2999 - val_acc: 0.4032
Epoch 3/100
312/312 [==============================] - 1s 3ms/step - loss: 0.7633 - acc:
0.6960 - val_loss: 1.2427 - val_acc: 0.4839
Epoch 4/100
```

```
312/312 [==============================] - 1s 3ms/step - loss: 0.6277 - acc:
0.7535 - val_loss: 1.2072 - val_acc: 0.5242
Epoch 5/100
312/312 [==============================] - 1s 3ms/step - loss: 0.5329 - acc:
0.7929 - val_loss: 1.4033 - val_acc: 0.4355
Epoch 6/100
312/312 [==============================] - 1s 3ms/step - loss: 0.4540 - acc:
0.8257 - val_loss: 1.4104 - val_acc: 0.4274
Epoch 7/100
312/312 [==============================] - 1s 3ms/step - loss: 0.4021 - acc:
0.8441 - val_loss: 1.3291 - val_acc: 0.4960
Epoch 8/100
312/312 [==============================] - 1s 3ms/step - loss: 0.3414 - acc:
0.8701 - val_loss: 2.2529 - val_acc: 0.4274
Epoch 9/100
312/312 [==============================] - 1s 3ms/step - loss: 0.3128 - acc:
0.8794 - val_loss: 1.6378 - val_acc: 0.5363
Epoch 10/100
312/312 [==============================] - 1s 3ms/step - loss: 0.2778 - acc:
0.8953 - val_loss: 1.5157 - val_acc: 0.5524
Epoch 11/100
312/312 [==============================] - 1s 3ms/step - loss: 0.2487 - acc:
0.9036 - val_loss: 1.3534 - val_acc: 0.6048
Epoch 12/100
312/312 [==============================] - 1s 3ms/step - loss: 0.2314 - acc:
0.9113 - val_loss: 2.2855 - val_acc: 0.4234
Epoch 13/100
312/312 [==============================] - 1s 3ms/step - loss: 0.2099 - acc:
0.9196 - val_loss: 1.7942 - val_acc: 0.5282
Epoch 14/100
312/312 [==============================] - 1s 3ms/step - loss: 0.1994 - acc:
0.9272 - val_loss: 1.8804 - val_acc: 0.4919
Epoch 15/100
312/312 [==============================] - 1s 3ms/step - loss: 0.1754 - acc:
0.9345 - val_loss: 1.8177 - val_acc: 0.6089
Epoch 16/100
312/312 [==============================] - 1s 3ms/step - loss: 0.1621 - acc:
0.9393 - val_loss: 1.7722 - val_acc: 0.5605
Epoch 17/100
312/312 [==============================] - 1s 3ms/step - loss: 0.1565 - acc:
0.9418 - val_loss: 1.9729 - val_acc: 0.5565
Epoch 18/100
312/312 [==============================] - 1s 3ms/step - loss: 0.1471 - acc:
0.9451 - val_loss: 1.8679 - val_acc: 0.6008
Epoch 19/100
312/312 [==============================] - 1s 3ms/step - loss: 0.1316 - acc:
0.9539 - val_loss: 2.1959 - val_acc: 0.5161
Epoch 20/100
```

```
312/312 [==============================] - 1s 3ms/step - loss: 0.1265 - acc:
0.9542 - val_loss: 1.7501 - val_acc: 0.6169
Epoch 21/100
312/312 [==============================] - 1s 3ms/step - loss: 0.1260 - acc:
0.9532 - val_loss: 2.1891 - val_acc: 0.5444
Epoch 22/100
312/312 [==============================] - 1s 3ms/step - loss: 0.1186 - acc:
0.9574 - val_loss: 2.0071 - val_acc: 0.5645
Epoch 23/100
312/312 [==============================] - 1s 4ms/step - loss: 0.0988 - acc:
0.9625 - val_loss: 2.3059 - val_acc: 0.5605
Epoch 24/100
312/312 [==============================] - 1s 4ms/step - loss: 0.1001 - acc:
0.9648 - val_loss: 1.9170 - val_acc: 0.6734
Epoch 25/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0959 - acc:
0.9653 - val_loss: 2.5547 - val_acc: 0.5202
Epoch 26/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0937 - acc:
0.9667 - val_loss: 2.0857 - val_acc: 0.6331
Epoch 27/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0878 - acc:
0.9682 - val_loss: 2.0829 - val_acc: 0.5968
Epoch 28/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0811 - acc:
0.9707 - val_loss: 2.9655 - val_acc: 0.4839
Epoch 29/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0758 - acc:
0.9733 - val_loss: 2.6659 - val_acc: 0.5847
Epoch 30/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0748 - acc:
0.9742 - val_loss: 2.9969 - val_acc: 0.5000
Epoch 31/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0684 - acc:
0.9759 - val_loss: 2.5995 - val_acc: 0.5806
Epoch 32/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0658 - acc:
0.9762 - val_loss: 2.8983 - val_acc: 0.5806
Epoch 33/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0640 - acc:
0.9777 - val_loss: 2.7233 - val_acc: 0.5766
Epoch 34/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0607 - acc:
0.9779 - val_loss: 2.5102 - val_acc: 0.5968
Epoch 35/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0617 - acc:
0.9775 - val_loss: 2.4059 - val_acc: 0.6290
Epoch 36/100
```

```
312/312 [==============================] - 1s 3ms/step - loss: 0.0560 - acc:
0.9831 - val_loss: 2.8598 - val_acc: 0.5645
Epoch 37/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0550 - acc:
0.9824 - val_loss: 2.5789 - val_acc: 0.5927
Epoch 38/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0562 - acc:
0.9800 - val_loss: 2.6393 - val_acc: 0.6169
Epoch 39/100
312/312 [==============================] - 1s 3ms/step - loss: 0.0555 - acc:
0.9813 - val_loss: 2.6685 - val_acc: 0.6008
8/8 [==============================] - 0s 2ms/step - loss: 2.6685 - acc: 0.6008
[INFO] accuracy: 60.08%
[INFO] Loss: 2.6685335636138916
[INFO] EarlyStopping Epoch Count: 39
Time:  0:00:39.187392
```





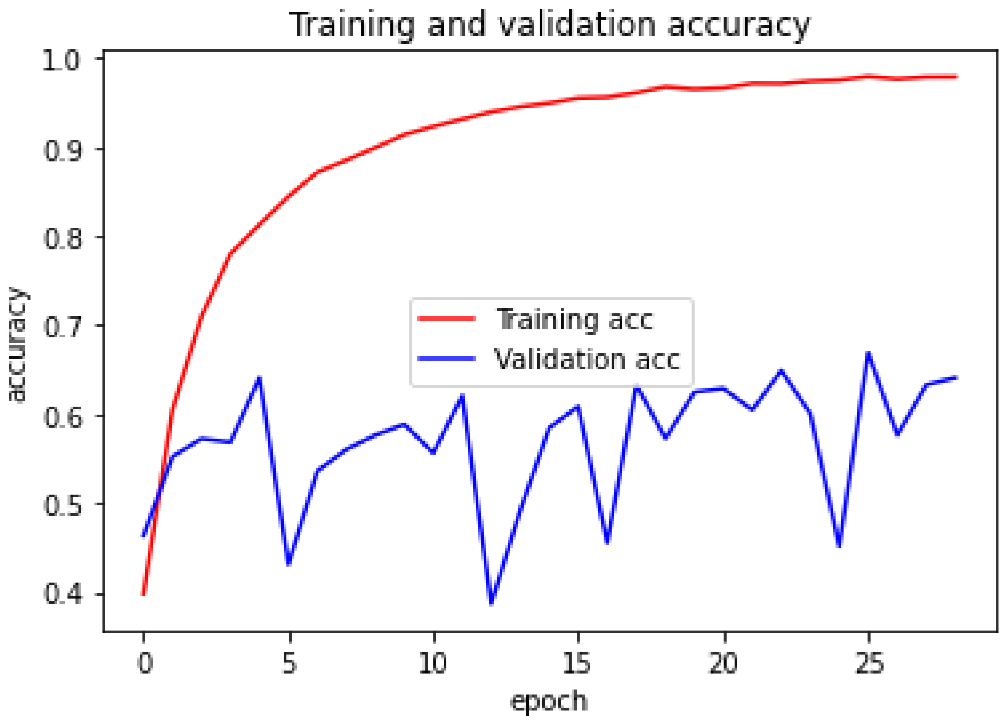
0.0.2 4 Hidden Layer CNN

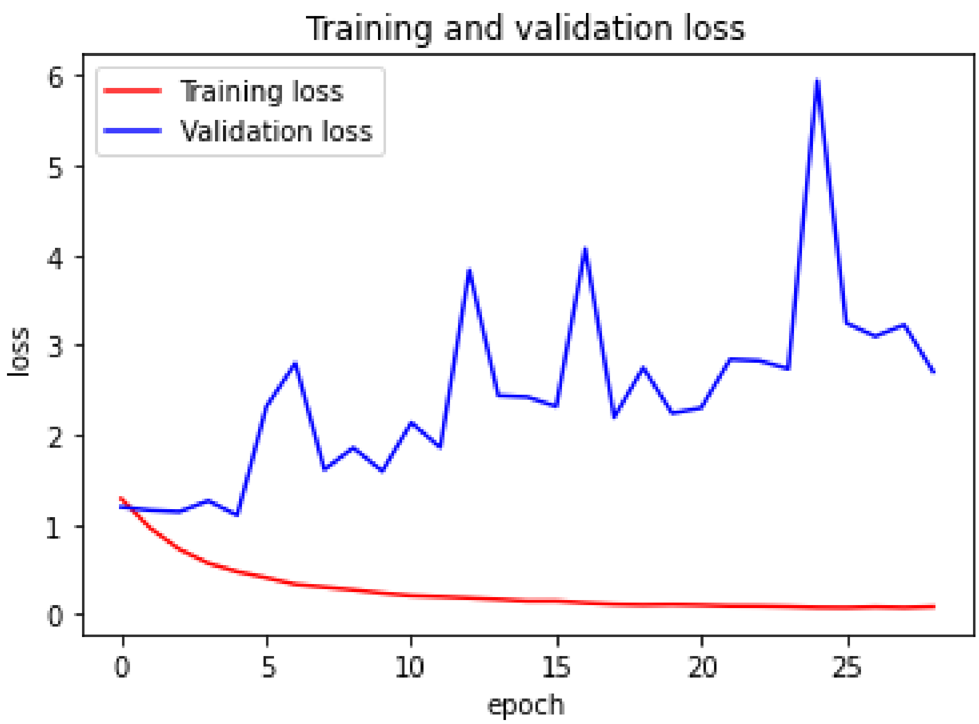
[ ]:
```
# This is a new variant of our base model! We're going to add some extra layers␣
↪here!

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()
```

```
Epoch 1/100
312/312 [==============================] - 2s 5ms/step - loss: 1.2899 - acc:
0.3982 - val_loss: 1.1973 - val_acc: 0.4637
Epoch 2/100
312/312 [==============================] - 1s 5ms/step - loss: 0.9618 - acc:
0.6053 - val_loss: 1.1617 - val_acc: 0.5524
Epoch 3/100
312/312 [==============================] - 1s 5ms/step - loss: 0.7270 - acc:
0.7102 - val_loss: 1.1445 - val_acc: 0.5726
Epoch 4/100
312/312 [==============================] - 1s 5ms/step - loss: 0.5706 - acc:
0.7802 - val_loss: 1.2667 - val_acc: 0.5685
Epoch 5/100
312/312 [==============================] - 1s 5ms/step - loss: 0.4762 - acc:
0.8129 - val_loss: 1.1048 - val_acc: 0.6411
Epoch 6/100
312/312 [==============================] - 1s 5ms/step - loss: 0.4083 - acc:
0.8445 - val_loss: 2.3142 - val_acc: 0.4315
Epoch 7/100
312/312 [==============================] - 2s 5ms/step - loss: 0.3349 - acc:
0.8718 - val_loss: 2.7984 - val_acc: 0.5363
Epoch 8/100
312/312 [==============================] - 1s 5ms/step - loss: 0.3038 - acc:
0.8854 - val_loss: 1.6142 - val_acc: 0.5605
Epoch 9/100
312/312 [==============================] - 1s 5ms/step - loss: 0.2731 - acc:
0.8995 - val_loss: 1.8550 - val_acc: 0.5766
Epoch 10/100
312/312 [==============================] - 2s 5ms/step - loss: 0.2367 - acc:
0.9141 - val_loss: 1.5970 - val_acc: 0.5887
Epoch 11/100
312/312 [==============================] - 2s 5ms/step - loss: 0.2085 - acc:
0.9233 - val_loss: 2.1355 - val_acc: 0.5565
Epoch 12/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1973 - acc:
0.9315 - val_loss: 1.8591 - val_acc: 0.6210
Epoch 13/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1829 - acc:
0.9396 - val_loss: 3.8343 - val_acc: 0.3871
Epoch 14/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1674 - acc:
0.9455 - val_loss: 2.4378 - val_acc: 0.4919
Epoch 15/100
312/312 [==============================] - 1s 5ms/step - loss: 0.1488 - acc:
0.9496 - val_loss: 2.4218 - val_acc: 0.5847
Epoch 16/100
312/312 [==============================] - 1s 5ms/step - loss: 0.1491 - acc:
0.9552 - val_loss: 2.3159 - val_acc: 0.6089
```

```
Epoch 17/100
312/312 [==============================] - 1s 5ms/step - loss: 0.1281 - acc:
0.9562 - val_loss: 4.0748 - val_acc: 0.4556
Epoch 18/100
312/312 [==============================] - 1s 5ms/step - loss: 0.1131 - acc:
0.9612 - val_loss: 2.1961 - val_acc: 0.6331
Epoch 19/100
312/312 [==============================] - 1s 5ms/step - loss: 0.1061 - acc:
0.9679 - val_loss: 2.7482 - val_acc: 0.5726
Epoch 20/100
312/312 [==============================] - 1s 5ms/step - loss: 0.1107 - acc:
0.9653 - val_loss: 2.2404 - val_acc: 0.6250
Epoch 21/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1039 - acc:
0.9667 - val_loss: 2.3000 - val_acc: 0.6290
Epoch 22/100
312/312 [==============================] - 1s 5ms/step - loss: 0.0946 - acc:
0.9715 - val_loss: 2.8417 - val_acc: 0.6048
Epoch 23/100
312/312 [==============================] - 1s 5ms/step - loss: 0.0929 - acc:
0.9713 - val_loss: 2.8256 - val_acc: 0.6492
Epoch 24/100
312/312 [==============================] - 1s 5ms/step - loss: 0.0871 - acc:
0.9742 - val_loss: 2.7365 - val_acc: 0.6008
Epoch 25/100
312/312 [==============================] - 1s 5ms/step - loss: 0.0774 - acc:
0.9754 - val_loss: 5.9460 - val_acc: 0.4516
Epoch 26/100
312/312 [==============================] - 1s 5ms/step - loss: 0.0750 - acc:
0.9794 - val_loss: 3.2466 - val_acc: 0.6694
Epoch 27/100
312/312 [==============================] - 1s 5ms/step - loss: 0.0833 - acc:
0.9766 - val_loss: 3.0971 - val_acc: 0.5766
Epoch 28/100
312/312 [==============================] - 1s 5ms/step - loss: 0.0759 - acc:
0.9788 - val_loss: 3.2262 - val_acc: 0.6331
Epoch 29/100
312/312 [==============================] - 1s 5ms/step - loss: 0.0861 - acc:
0.9789 - val_loss: 2.7028 - val_acc: 0.6411
8/8 [==============================] - 0s 2ms/step - loss: 2.7028 - acc: 0.6411
[INFO] accuracy: 64.11%
[INFO] Loss: 2.702789306640625
[INFO] EarlyStopping Epoch Count: 29
Time:  0:00:44.839195
```


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


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

5 Hidden Layer CNN

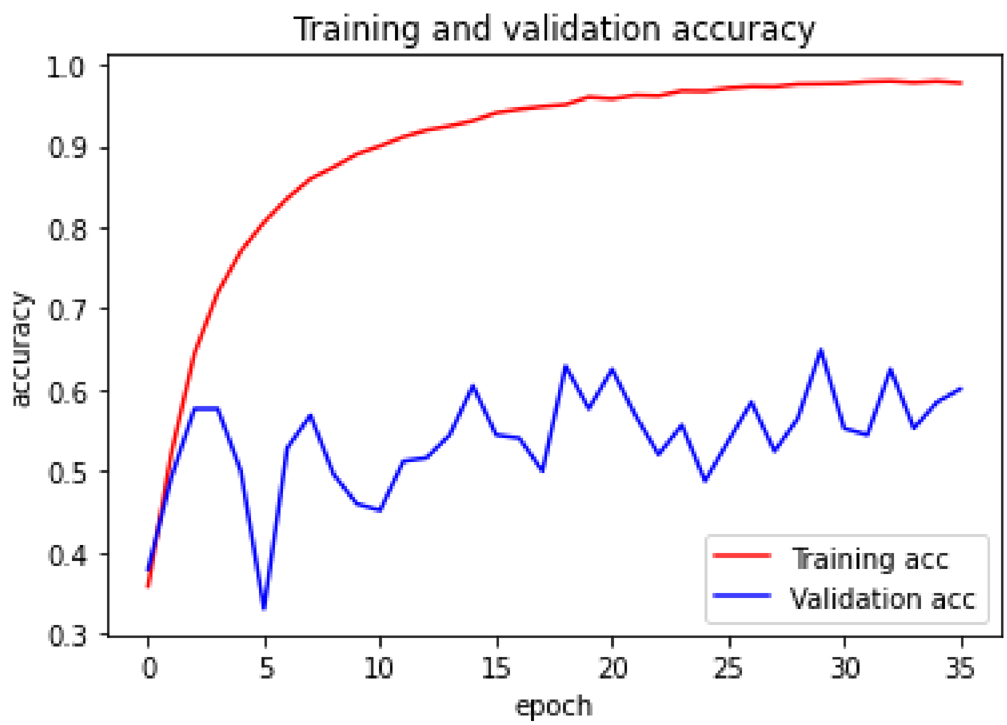
[ ]:
```
# This is a new variant of our base model! We're going to add some extra layers␣
↪here!

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)
```

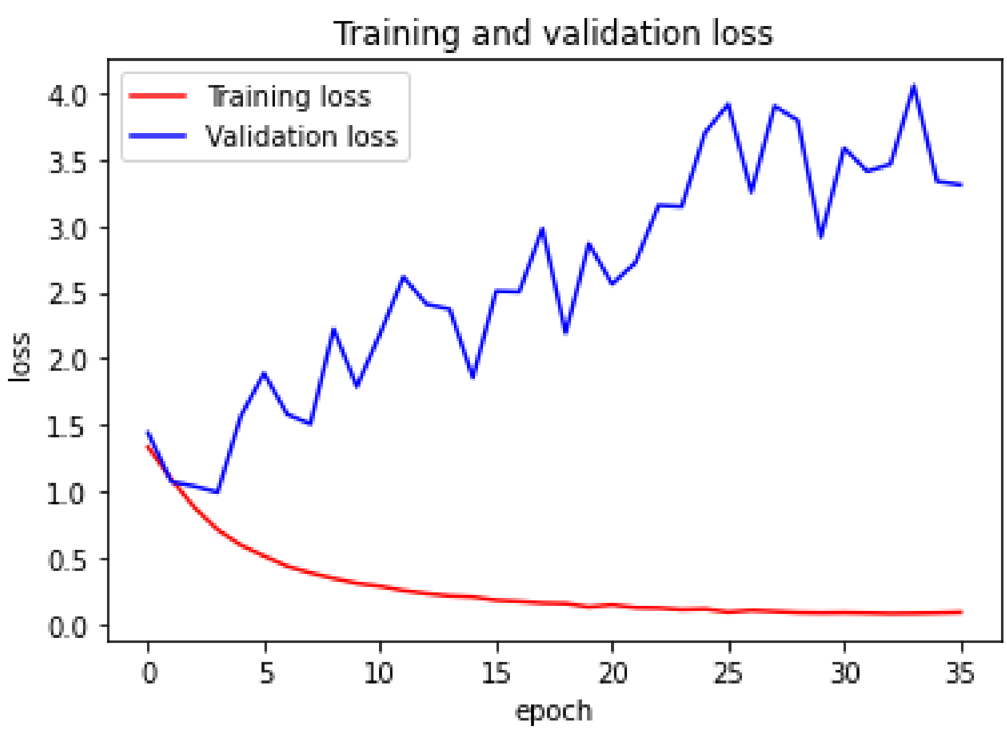
```python
# Graphing our training and validation

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()
```

```
Epoch 1/100
312/312 [==============================] - 2s 5ms/step - loss: 1.3325 - acc:
0.3583 - val_loss: 1.4402 - val_acc: 0.3790
Epoch 2/100
312/312 [==============================] - 2s 5ms/step - loss: 1.0929 - acc:
0.5213 - val_loss: 1.0737 - val_acc: 0.4919
Epoch 3/100
312/312 [==============================] - 2s 5ms/step - loss: 0.8781 - acc:
0.6447 - val_loss: 1.0387 - val_acc: 0.5766
Epoch 4/100
312/312 [==============================] - 2s 5ms/step - loss: 0.7113 - acc:
0.7195 - val_loss: 0.9940 - val_acc: 0.5766
Epoch 5/100
312/312 [==============================] - 2s 5ms/step - loss: 0.5950 - acc:
0.7703 - val_loss: 1.5677 - val_acc: 0.5000
Epoch 6/100
312/312 [==============================] - 2s 5ms/step - loss: 0.5131 - acc:
0.8061 - val_loss: 1.8898 - val_acc: 0.3306
Epoch 7/100
312/312 [==============================] - 2s 5ms/step - loss: 0.4357 - acc:
0.8354 - val_loss: 1.5801 - val_acc: 0.5282
Epoch 8/100
312/312 [==============================] - 2s 5ms/step - loss: 0.3856 - acc:
0.8594 - val_loss: 1.5075 - val_acc: 0.5685
```

```
Epoch 9/100
312/312 [==============================] - 2s 5ms/step - loss: 0.3441 - acc:
0.8739 - val_loss: 2.2232 - val_acc: 0.4960
Epoch 10/100
312/312 [==============================] - 2s 5ms/step - loss: 0.3075 - acc:
0.8898 - val_loss: 1.7877 - val_acc: 0.4597
Epoch 11/100
312/312 [==============================] - 2s 5ms/step - loss: 0.2851 - acc:
0.8999 - val_loss: 2.1882 - val_acc: 0.4516
Epoch 12/100
312/312 [==============================] - 2s 5ms/step - loss: 0.2525 - acc:
0.9112 - val_loss: 2.6157 - val_acc: 0.5121
Epoch 13/100
312/312 [==============================] - 2s 5ms/step - loss: 0.2301 - acc:
0.9194 - val_loss: 2.4104 - val_acc: 0.5161
Epoch 14/100
312/312 [==============================] - 2s 5ms/step - loss: 0.2120 - acc:
0.9245 - val_loss: 2.3729 - val_acc: 0.5444
Epoch 15/100
312/312 [==============================] - 2s 5ms/step - loss: 0.2043 - acc:
0.9309 - val_loss: 1.8559 - val_acc: 0.6048
Epoch 16/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1795 - acc:
0.9407 - val_loss: 2.5087 - val_acc: 0.5444
Epoch 17/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1705 - acc:
0.9451 - val_loss: 2.5035 - val_acc: 0.5403
Epoch 18/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1583 - acc:
0.9484 - val_loss: 2.9796 - val_acc: 0.5000
Epoch 19/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1546 - acc:
0.9508 - val_loss: 2.1934 - val_acc: 0.6290
Epoch 20/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1287 - acc:
0.9602 - val_loss: 2.8652 - val_acc: 0.5766
Epoch 21/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1450 - acc:
0.9580 - val_loss: 2.5618 - val_acc: 0.6250
Epoch 22/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1239 - acc:
0.9624 - val_loss: 2.7210 - val_acc: 0.5685
Epoch 23/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1206 - acc:
0.9616 - val_loss: 3.1552 - val_acc: 0.5202
Epoch 24/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1088 - acc:
0.9678 - val_loss: 3.1450 - val_acc: 0.5565
```

```
Epoch 25/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1135 - acc:
0.9674 - val_loss: 3.7017 - val_acc: 0.4879
Epoch 26/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0902 - acc:
0.9710 - val_loss: 3.9170 - val_acc: 0.5363
Epoch 27/100
312/312 [==============================] - 2s 5ms/step - loss: 0.1033 - acc:
0.9730 - val_loss: 3.2542 - val_acc: 0.5847
Epoch 28/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0955 - acc:
0.9727 - val_loss: 3.9031 - val_acc: 0.5242
Epoch 29/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0870 - acc:
0.9758 - val_loss: 3.7950 - val_acc: 0.5645
Epoch 30/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0842 - acc:
0.9761 - val_loss: 2.9140 - val_acc: 0.6492
Epoch 31/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0861 - acc:
0.9768 - val_loss: 3.5877 - val_acc: 0.5524
Epoch 32/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0832 - acc:
0.9789 - val_loss: 3.4110 - val_acc: 0.5444
Epoch 33/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0794 - acc:
0.9797 - val_loss: 3.4649 - val_acc: 0.6250
Epoch 34/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0810 - acc:
0.9778 - val_loss: 4.0564 - val_acc: 0.5524
Epoch 35/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0839 - acc:
0.9795 - val_loss: 3.3349 - val_acc: 0.5847
Epoch 36/100
312/312 [==============================] - 2s 5ms/step - loss: 0.0887 - acc:
0.9773 - val_loss: 3.3100 - val_acc: 0.6008
8/8 [==============================] - 0s 2ms/step - loss: 3.3100 - acc: 0.6008
[INFO] accuracy: 60.08%
[INFO] Loss: 3.310025930404663
[INFO] EarlyStopping Epoch Count: 36
Time:  0:00:58.908960
```


Training and validation accuracy
accuracy
epoch
Training acc
Validation acc


Training and validation loss
loss
epoch
Training loss
Validation loss

[ ]:
```python
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)
```

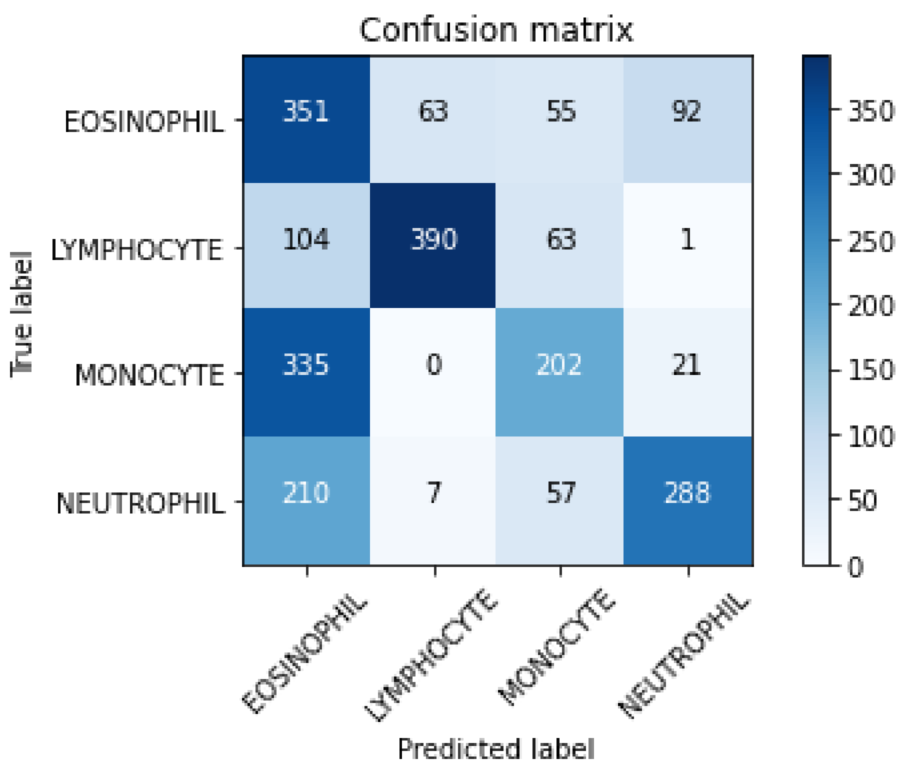
```
              precision    recall  f1-score   support

  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
```

[ ]:
```python
# 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)
```

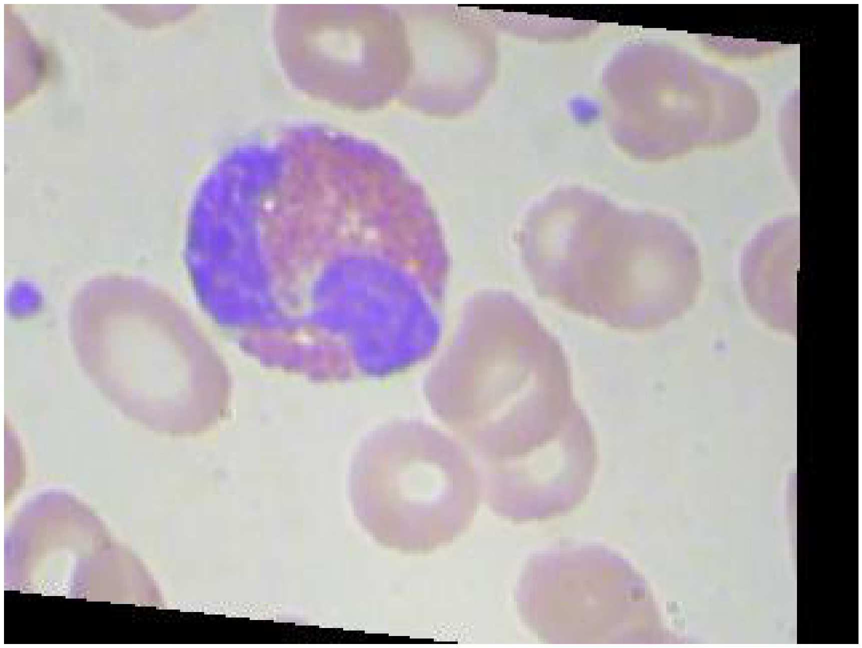
```
    image /= 255.
    return image

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:
.
..
…
```

[ ]:

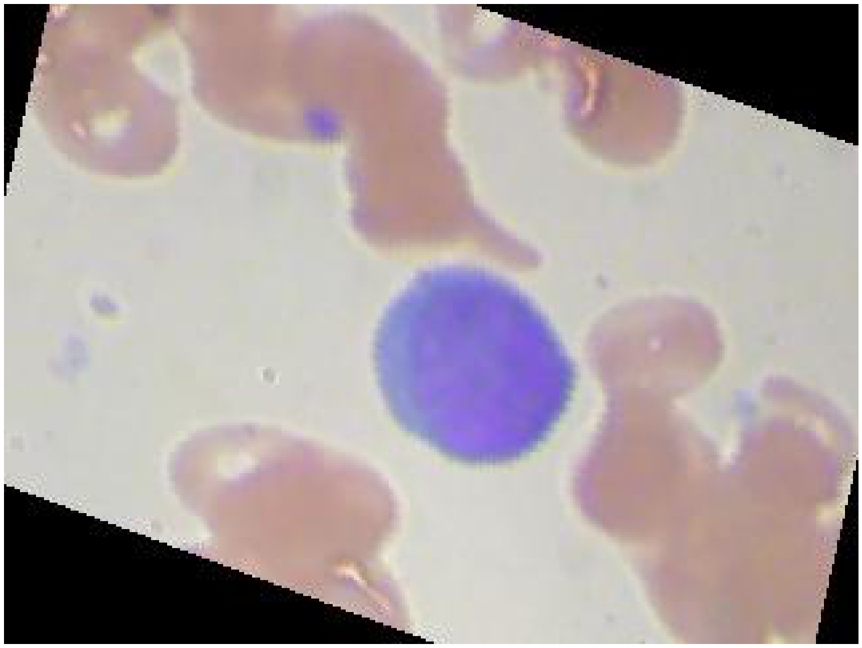


Guessed this was an eosinophile with 100.00% confidence.

```
[ ]: # 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:
.
..
…
```

```
[ ]:
```

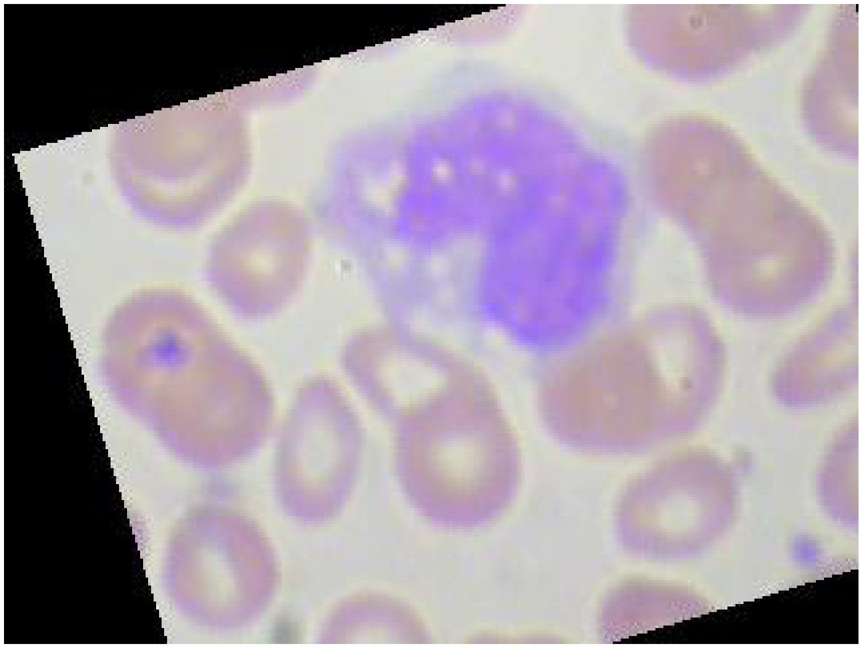


Guessed this was a lymphocyte with 100.00% confidence.

[ ]:
```
# 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:
.
..
…
```

[ ]:

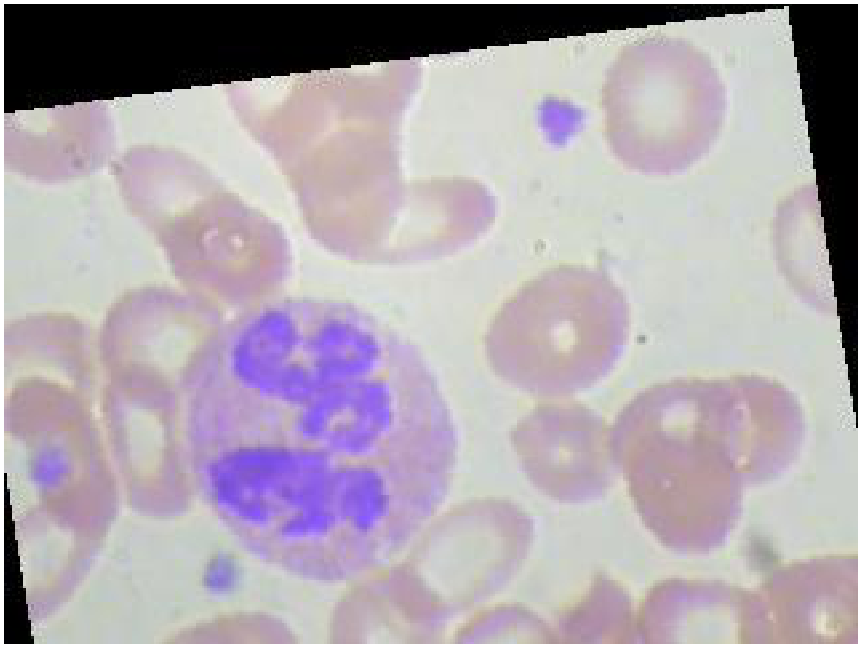


Guessed this was a monocyte with 31.76% confidence and 56.11% confidence this is an eosinphil. The model clearly has a hard time prediciting the classification of monocytes.

[ ]:
```
# 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.