Assignment 1

by Ebba Bergman

Let's do something very similar to the lab

Hand in: This notebook, and a pdf of this notebook. No written answers to the questions are required, they are only here to help you learn

You are free to discuss the general concepts with other groups, but we encourage you not to exchange code for your own learning

A lot of the code below is inspired labs developed by Christophe Avenel at NBIS , labs and assignments made by Phil Harrison as well as by https://www.tensorflow.org/guide/keras/functional/,

```
In [6]:  ## First we need to import all of the packages we need

         import numpy as np
         import tensorflow as tf
         import pandas as pd
         from PIL import Image
         import IPython
         from tensorflow import keras
         from tensorflow.keras import layers
         import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix

         from tensorflow.keras.preprocessing.image import ImageDataGenerator

         import cnn_helper
```

Note: the cnn_helper was written by Christophe Avenel, and his code (including his lab which this one is based on), is available here: https://github.com/NBISweden/workshop-neural-nets-and-deep-learning/tree/master/session_convolutionalNeuralNetworks/Labs

In [7…

```
def plot_history(model_history, model_name):
    fig = plt.figure(figsize=(15, 5), facecolor='w')
    ax = fig.add_subplot(131)
    ax.plot(model_history.history['loss'])
    ax.plot(model_history.history['val_loss'])
    ax.set(title=model_name + ': Model loss', ylabel='Loss', xlabel='
    ax.legend(['train', 'valid'], loc='upper right')

    ax = fig.add_subplot(132)
    ax.plot(np.log(model_history.history['loss']))
    ax.plot(np.log(model_history.history['val_loss']))
    ax.set(title=model_name + ': Log model loss', ylabel='Log loss',
    ax.legend(['Train', 'Test'], loc='upper right')

    ax = fig.add_subplot(133)
    ax.plot(model_history.history['accuracy'])
    ax.plot(model_history.history['val_accuracy'])
    ax.set(title=model_name + ': Model accuracy', ylabel='Accuracy',
    ax.legend(['train', 'valid'], loc='upper right')
    plt.show()
    plt.close()

    plt.savefig("History Plot.png")
```

Set up the data, look at it

In [8]:

```
## Set up where to find our data
data_directory = "./LabData/bloodcells_small/data/"
labels_path =  "./LabData/bloodcells_small/labels.csv"
```

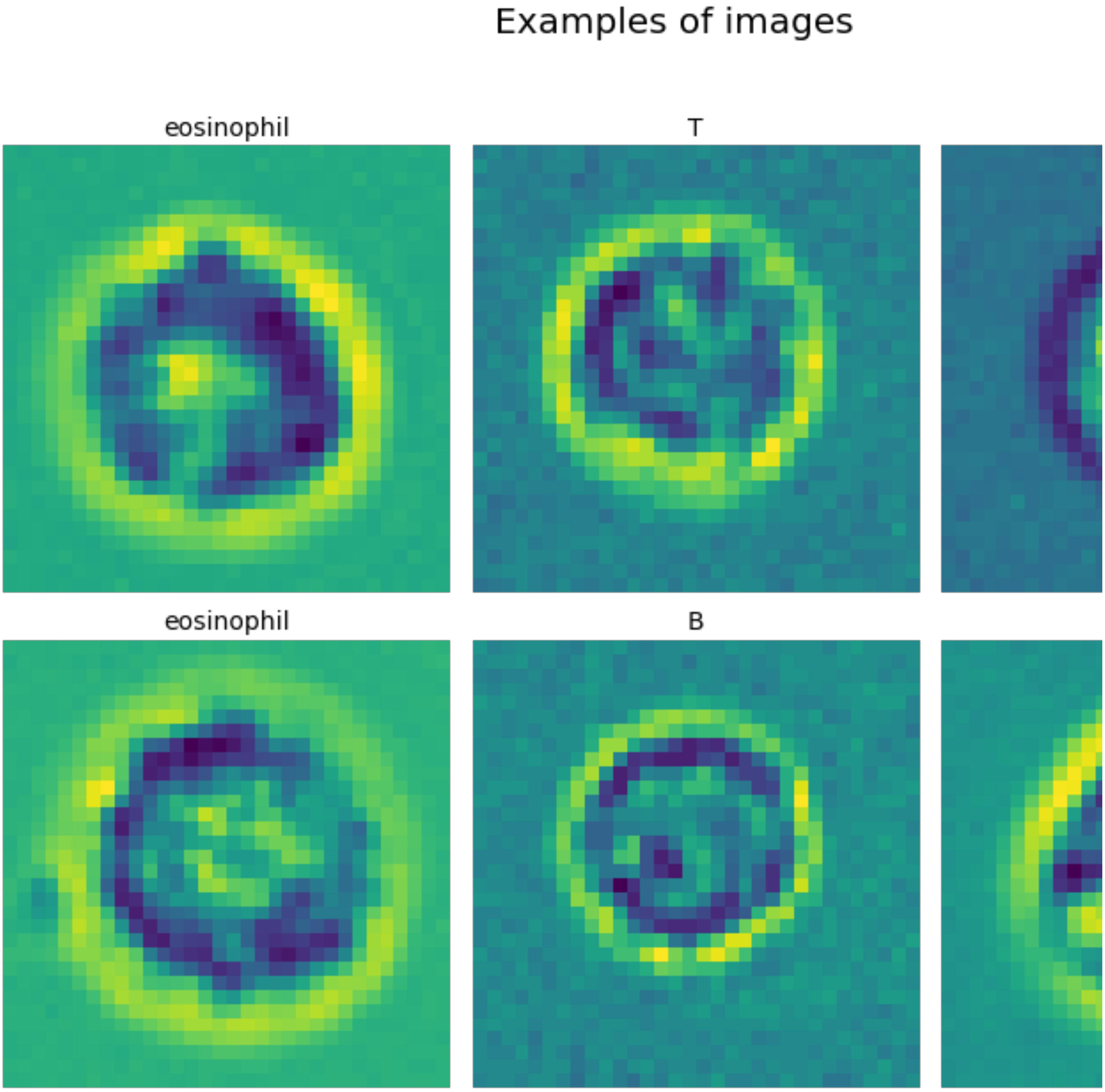
In…

```
# This is a dataframe, a way to look at data as tables.
#Google "Python pandas dataframe" to get more information, or to find n
# Anything you can do with data frames you could do with loops, but it
df_labels = pd.read_csv(labels_path)
```

Q: Look at the labels, what columns do you think contains the true label?

In [1…

```
## Let's look at the images - always a good start to the project
# Here random images will be displayed, run this several time to see

figure, ax = plt.subplots(2, 3, figsize=(14, 10))
figure.suptitle("Examples of images", fontsize=20)
axes = ax.ravel()

df_images_to_show = df_labels.sample(8)


for i in range(len(axes)):
    row = df_images_to_show.iloc[[i]]
    random_image = Image.open(data_directory + row["Filenames"].valu
    axes[i].set_title(row["Class"].values[0], fontsize=14)
    axes[i].imshow(random_image)
    axes[i].set_axis_off()

plt.subplots_adjust(wspace=0.05, hspace=0.05)
plt.show()
plt.close()
```



Q: Can you see any difference between the classes?

Q: Do you think a human being able to see the difference between classes makes it an easier or more difficult problem for a neural network?

```
In [11]: # What's the shape of the image?
         image_shape = np.array(random_image).shape
         print(image_shape)
```

```
(32, 32)
```

```
In [12]: # Let's look a little bit into the labels
         set_size = df_labels.size
         print(set_size)
         print(df_labels.head())
```

```
41578
                Filenames Class
0  CRF022_T_1_ch5_106.png     T
1  CRF022_T_1_ch5_119.png     T
2  CRF022_T_1_ch5_123.png     T
3  CRF022_T_1_ch5_128.png     T
4  CRF022_T_1_ch5_134.png     T
```

```
In [13]: df_labels['Class'].value_counts()
```

```
Out[13]: neutrophil    4500
         monocyte      4303
         T             4100
         B             4032
         eosinophil    3854
         Name: Class, dtype: int64
```

Divide the data for training, validation and test

```
…  ## Next, let's divide the filtered rows into a train, validation and a te
   class_column_header = "Class"
   df_to_use = df_labels.copy() #We're copying the df_labels so that you can

   test_set_fraction = 0.1
   validation_set_fraction = 0.2

   df_test = df_to_use.groupby(class_column_header).sample(frac = test_set_f
   df_to_use = pd.concat([df_to_use, df_test, df_test]).drop_duplicates(keep
   df_valid = df_to_use.groupby(class_column_header).sample(frac = validatio
   df_train = pd.concat([df_to_use, df_valid, df_valid]).drop_duplicates(kee
```

```
In [15]: print(df_test.head())
```

```
                   Filenames Class
19071  CRF132_B_1_ch5_73.png     B
17414  CRF034_B_2_ch5_55.png     B
19390  CRF132_B_3_ch5_15.png     B
18484  CRF074_B_3_ch5_23.png     B
17214  CRF022_B_3_ch5_30.png     B
```

In…

```
## Set up generators that specify how the images are loaded, how many at
## that the images should be shuffled etc.
batch_size = 8

filename_column = 'Filenames'
true_value = "Class"
# create a data generator

## Note: we tend to get better results if the values of the pixels are b
train_data_generator = keras.preprocessing.image.ImageDataGenerator(resc
valid_data_generator = keras.preprocessing.image.ImageDataGenerator(resc
test_data_generator = keras.preprocessing.image.ImageDataGenerator(resca

train_generator = train_data_generator.flow_from_dataframe(
    df_train, directory=data_directory, x_col=filename_column, y_col=tru
    weight_col=None, class_mode='categorical', batch_size=batch_size, ta
)

valid_generator = valid_data_generator.flow_from_dataframe(
    df_valid, directory=data_directory, x_col=filename_column, y_col=tru
    weight_col=None, class_mode='categorical', batch_size=batch_size, ta
)


test_generator = test_data_generator.flow_from_dataframe(
    df_test, directory=data_directory, x_col=filename_column, y_col=true
    weight_col=None,class_mode='categorical', batch_size=batch_size, tar
)


train_steps=train_generator.n//train_generator.batch_size if train_gener
validation_steps=valid_generator.n//valid_generator.batch_size if valid_
```

```
Found 14968 validated image filenames belonging to 5 classes.
Found 3743 validated image filenames belonging to 5 classes.
Found 2078 validated image filenames belonging to 5 classes.
```

CNN

Convolutional Neural Networks revolutionized the field of deep learning. You have seen how convolutions work in the lectures. One of the huge benefits of convolutions is that as the filters (sometimes called kernels in codes) move across the image the position of an object in an image becomes much less important than when we flattened images to use in traditional Artificial Neural Networks.

For this part of the lab you will try a couple of different architectures and hyperparameters. The **architecture** is basically the structure of the network: how many nodes, how many layers, and overall shape of these. The **hyperparamters** are most easily defined as all of the parameters changed *before* the training of the network begin, such as the number of epochs, what activation function to use in each layer, and which optimization method we use for backpropagation.

```
In [1…  ## Set up the model architecture
        # See https://www.tensorflow.org/guide/keras/functional/ if you want

        cnn_inputs = keras.Input(shape=(32,32,1))
        x = layers.Conv2D(1, kernel_size=(3, 3), strides=1,padding='same')(cr
        x = layers.MaxPooling2D(pool_size=(2, 2))(x)
        x = layers.Flatten()(x)
        cnn_outputs = layers.Dense(5, activation='softmax')(x)
```

```
In [2…  ## Define the model as a keras model
        cnn_model = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs, name
```

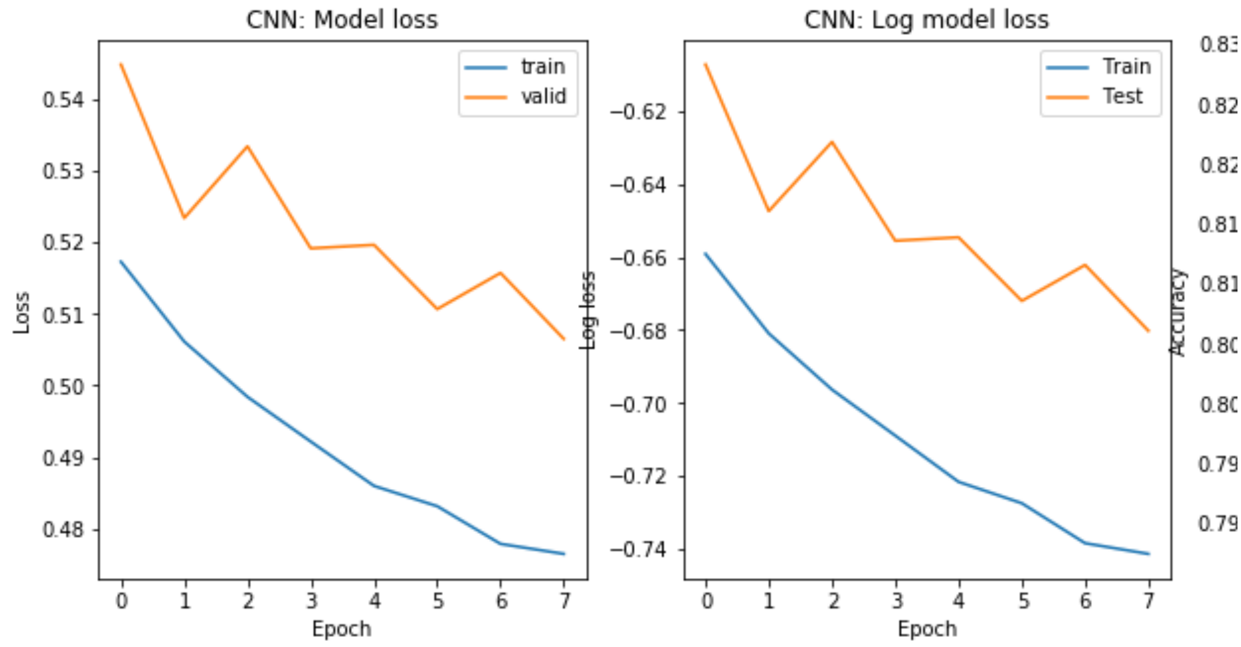
```
In [… ## We'll use the same generators as above here, so no need to redefine
      ## compile model

      cnn_model.compile(optimizer=keras.optimizers.Adam(), loss='categorical
      cnn_model.summary()
```

```
Model: "cnn_Model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 32, 32, 1)         10
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 1)         0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 5)                 1285
=================================================================
Total params: 1,295
Trainable params: 1,295
Non-trainable params: 0
_________________________________________________________________
```

```
In [26]:  ## Actually train model
          epochs = 8
          history = cnn_model.fit_generator(generator=train_generator,
                                            steps_per_epoch= train_steps,
                                            validation_data= valid_generator,
                                            validation_steps= validation_steps,
                                            epochs= epochs
                  )
```

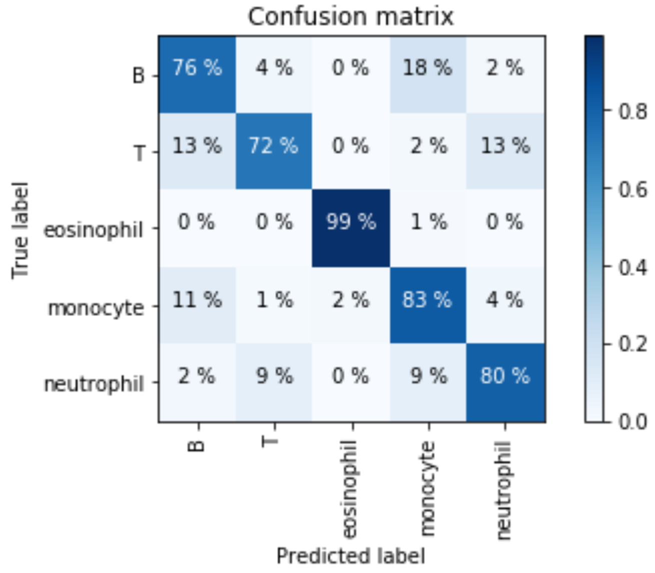
```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/8
1871/1871 [==============================] - 20s 11ms/step - loss: 0.5173
- accuracy: 0.8115 - val_loss: 0.5448 - val_accuracy: 0.7875
Epoch 2/8
1871/1871 [==============================] - 14s 8ms/step - loss: 0.5062 -
accuracy: 0.8140 - val_loss: 0.5234 - val_accuracy: 0.8116
Epoch 3/8
1871/1871 [==============================] - 15s 8ms/step - loss: 0.4984 -
accuracy: 0.8167 - val_loss: 0.5334 - val_accuracy: 0.8001
Epoch 4/8
1871/1871 [==============================] - 15s 8ms/step - loss: 0.4922 -
accuracy: 0.8183 - val_loss: 0.5192 - val_accuracy: 0.8086
Epoch 5/8
1871/1871 [==============================] - 32s 17ms/step - loss: 0.4860
- accuracy: 0.8226 - val_loss: 0.5196 - val_accuracy: 0.8065
Epoch 6/8
1871/1871 [==============================] - 14s 7ms/step - loss: 0.4832 -
accuracy: 0.8265 - val_loss: 0.5107 - val_accuracy: 0.8065
Epoch 7/8
1871/1871 [==============================] - 14s 7ms/step - loss: 0.4779 -
accuracy: 0.8283 - val_loss: 0.5157 - val_accuracy: 0.8118
Epoch 8/8
1871/1871 [==============================] - 15s 8ms/step - loss: 0.4765 -
accuracy: 0.8264 - val_loss: 0.5065 - val_accuracy: 0.8191
```

```
In [27]:  ## Plot results
          plot_history(history, "CNN")
```



```
<Figure size 432x288 with 0 Axes>
```

```
In [28…  # plot confusion matrix
         cnn_helper.plot_confusion_matrix_from_generator(cnn_model, valid_ge
Accuracy: 0.8185947101255677
```



Confusion matrix

True label \ Predicted label	B	T	eosinophil	monocyte	neutrophil
B	76 %	4 %	0 %	18 %	2 %
T	13 %	72 %	0 %	2 %	13 %
eosinophil	0 %	0 %	99 %	1 %	0 %
monocyte	11 %	1 %	2 %	83 %	4 %
neutrophil	2 %	9 %	0 %	9 %	80 %

Q: What do the curves tell you about the models?

You can see some examples of how curves can look at : https://uppsala.instructure.com/courses/23804/pages/deep-learning-plots/edit

Expanding the models

Deeper models

Sometimes a deeper model and/or a more complex model, can be helpful. Try adding some more convolution layers and pooling layers to the model. Try changing the filter sizes, and the number of filters as well. More information about the convolutional layer can be found here: https://keras.io/api/layers/convolution_layers/convolution2d/, maxpooling here: https://keras.io/api/layers/pooling_layers/max_pooling2d/, and a different kind of way of making models can be found here: https://www.tensorflow.org/tutorials/images/cnn and here https://www.tensorflow.org/tutorials/quickstart/advanced

In […

```
## Set up the model architecture
## v.1.1
# Accuracy validation: 0.9419
# Comment: Could run more than 10 epochs.
##

#Change the code below so that the new model has roughly the same num
# Hint: you can add both more Conc2D layers, and increase the kernel

cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', ac
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)

## Define the model
cnn_model = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs, name=

## Compile the model
cnn_model.compile(optimizer=keras.optimizers.Adam(), loss='categorica
cnn_model.summary()
```

```
Model: "cnn_Model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_50 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_78 (Conv2D)           (None, 32, 32, 5)         50
_________________________________________________________________
max_pooling2d_74 (MaxPooling (None, 16, 16, 5)         0
_________________________________________________________________
conv2d_79 (Conv2D)           (None, 16, 16, 5)         230
_________________________________________________________________
max_pooling2d_75 (MaxPooling (None, 8, 8, 5)           0
_________________________________________________________________
flatten_35 (Flatten)         (None, 320)               0
_________________________________________________________________
dense_38 (Dense)             (None, 5)                 1605
=================================================================
Total params: 1,885
Trainable params: 1,885
Non-trainable params: 0
_________________________________________________________________
```

In […

```python
## Set up the model architecture
# Larger kernel from 3 to 5
##
# Accuracy validation: 0.9430
# Comment: Hard to train for validationi the accuracy goes up and dow
##


#Change the code below so that the new model has roughly the same numb
# Hint: you can add both more Conc2D layers, and increase the kernel (

cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(5, 5), strides=1,padding='same', act
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(5, kernel_size=(5, 5), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)

## Define the model
cnn_model = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs, name='

## Compile the model
cnn_model.compile(optimizer=keras.optimizers.Adam(), loss='categorical
cnn_model.summary()
```

```
Model: "cnn_Model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_49 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_76 (Conv2D)           (None, 32, 32, 5)         130
_________________________________________________________________
max_pooling2d_72 (MaxPooling (None, 16, 16, 5)         0
_________________________________________________________________
conv2d_77 (Conv2D)           (None, 16, 16, 5)         630
_________________________________________________________________
max_pooling2d_73 (MaxPooling (None, 8, 8, 5)           0
_________________________________________________________________
flatten_34 (Flatten)         (None, 320)               0
_________________________________________________________________
dense_37 (Dense)             (None, 5)                 1605
=================================================================
Total params: 2,365
Trainable params: 2,365
Non-trainable params: 0
_________________________________________________________________
```

In […

```
## Set up the model architecture
# Small kernel from 3 to 1
##
# Accuracy validation: 0.9349
# Comment: Nice
##


#Change the code below so that the new model has roughly the same num
# Hint: you can add both more Conc2D layers, and increase the kernel

cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(1, 1), strides=1,padding='same', ac
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(5, kernel_size=(1, 1), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)

## Define the model
cnn_model = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs, name=

## Compile the model
cnn_model.compile(optimizer=keras.optimizers.Adam(), loss='categorica
cnn_model.summary()
```

```
Model: "cnn_Model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_43 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 32, 32, 5)         10
_________________________________________________________________
max_pooling2d_60 (MaxPooling (None, 16, 16, 5)         0
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 16, 16, 5)         30
_________________________________________________________________
max_pooling2d_61 (MaxPooling (None, 8, 8, 5)           0
_________________________________________________________________
flatten_28 (Flatten)         (None, 320)               0
_________________________________________________________________
dense_31 (Dense)             (None, 5)                 1605
=================================================================
Total params: 1,645
Trainable params: 1,645
Non-trainable params: 0
_________________________________________________________________
```

In […

```
## Set up the model architecture
# Big pool size from 2 and 2 to 8 and 4. Results in a feature maps of
##
# Accuracy validation: 0.8454
# Increased epoch to 20, still got questionable results :/
##


#Change the code below so that the new model has roughly the same num
# Hint: you can add both more Conc2D layers, and increase the kernel

cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', ac
x = layers.MaxPooling2D(pool_size=(8, 8))(x)
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(4, 4))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)

## Define the model
cnn_model = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs, name=

## Compile the model
cnn_model.compile(optimizer=keras.optimizers.Adam(), loss='categorica
cnn_model.summary()
```

```
Model: "cnn_Model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_41 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_60 (Conv2D)           (None, 32, 32, 5)         50
_________________________________________________________________
max_pooling2d_56 (MaxPooling (None, 4, 4, 5)           0
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 4, 4, 5)           230
_________________________________________________________________
max_pooling2d_57 (MaxPooling (None, 1, 1, 5)           0
_________________________________________________________________
flatten_26 (Flatten)         (None, 5)                 0
_________________________________________________________________
dense_29 (Dense)             (None, 5)                 30
=================================================================
Total params: 310
Trainable params: 310
Non-trainable params: 0
_________________________________________________________________
```

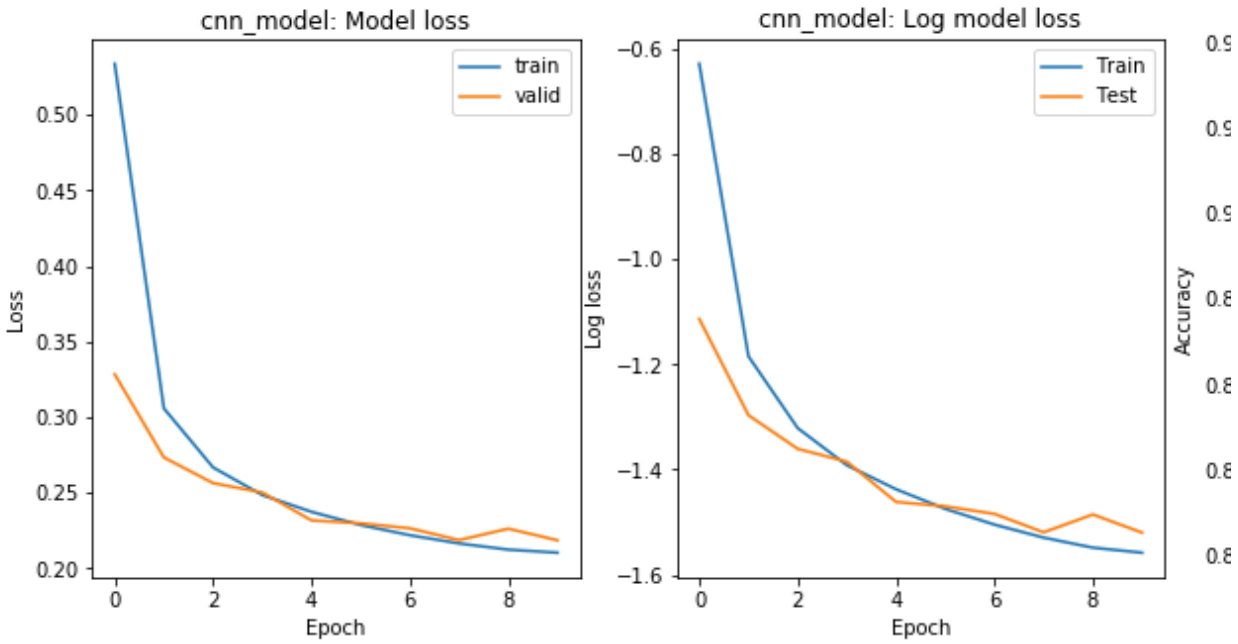
In [153]:

```
## Actually train model
epochs = 10
history = cnn_model.fit_generator(generator=train_generator,
                    steps_per_epoch= train_steps,
                    validation_data= valid_generator,
                    validation_steps= validation_steps,
                    epochs= epochs
                                 )
```

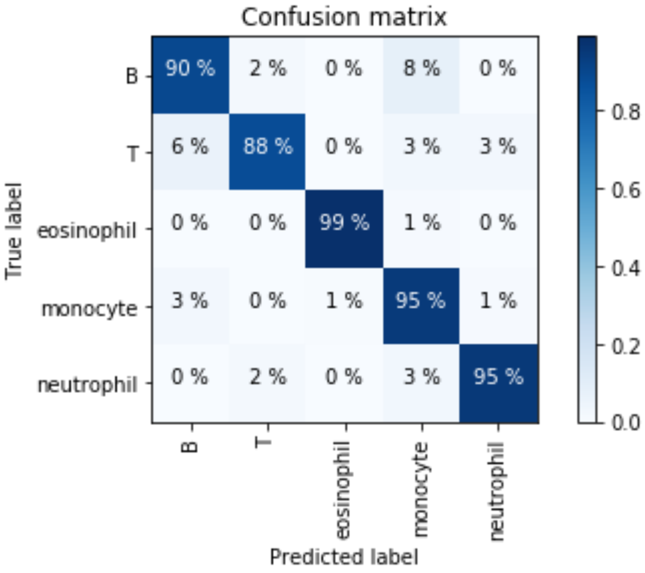
```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/10
1871/1871 [==============================] - 20s 11ms/step - loss: 0.5338
- accuracy: 0.8208 - val_loss: 0.3285 - val_accuracy: 0.8908
Epoch 2/10
1871/1871 [==============================] - 14s 7ms/step - loss: 0.3059 -
accuracy: 0.9020 - val_loss: 0.2735 - val_accuracy: 0.9111
Epoch 3/10
1871/1871 [==============================] - 14s 8ms/step - loss: 0.2669 -
accuracy: 0.9174 - val_loss: 0.2565 - val_accuracy: 0.9157
Epoch 4/10
1871/1871 [==============================] - 14s 8ms/step - loss: 0.2486 -
accuracy: 0.9224 - val_loss: 0.2503 - val_accuracy: 0.9208
Epoch 5/10
1871/1871 [==============================] - 16s 8ms/step - loss: 0.2375 -
accuracy: 0.9276 - val_loss: 0.2319 - val_accuracy: 0.9264
Epoch 6/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2288 -
accuracy: 0.9290 - val_loss: 0.2299 - val_accuracy: 0.9272
Epoch 7/10
1871/1871 [==============================] - 14s 8ms/step - loss: 0.2221 -
accuracy: 0.9323 - val_loss: 0.2266 - val_accuracy: 0.9267
Epoch 8/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2166 -
accuracy: 0.9334 - val_loss: 0.2188 - val_accuracy: 0.9299
Epoch 9/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2125 -
accuracy: 0.9339 - val_loss: 0.2264 - val_accuracy: 0.9277
Epoch 10/10
1871/1871 [==============================] - 16s 8ms/step - loss: 0.2105 -
accuracy: 0.9349 - val_loss: 0.2187 - val_accuracy: 0.9350
```

In [154…

```
## Plot results
plot_history(history, "cnn_model")

# plot confusion matrix
cnn_helper.plot_confusion_matrix_from_generator(cnn_model, valid_ge
```



```
Accuracy: 0.9345444830349987
```



Try a couple of deeper models and save your best one for further study

Add all these models beneath this heading

Data Augmentation

Let's try something else, maybe you would like to add some data augmentation? Data augmentation basically means that we randomly alter the incoming images in different ways to make sure that the network can handle those types of variations.

If you want to read more you can look at this article, especially the "Data Augmentations based on basic image manipulations Geometric transformations" is of interest here: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0

See https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator for things you can try by adding input paramters to the ImageDataGenerator().

Update the cell below to **include data augmentations, only in the training data generator then run your CNN again**

In…

```
## Set up generators
batch_size = 8

filename_column = 'Filenames'
true_value = "Class"
# create a data generator

## Note: we tend to get better results if the values of the pixels are b
train_data_generator = keras.preprocessing.image.ImageDataGenerator(resc
valid_data_generator = keras.preprocessing.image.ImageDataGenerator(resc
test_data_generator = keras.preprocessing.image.ImageDataGenerator(resca

train_generator = train_data_generator.flow_from_dataframe(
    df_train, directory=data_directory, x_col=filename_column, y_col=tru
    weight_col=None, class_mode='categorical', batch_size=batch_size, ta
)

valid_generator = valid_data_generator.flow_from_dataframe(
    df_valid, directory=data_directory, x_col=filename_column, y_col=tru
    weight_col=None, class_mode='categorical', batch_size=batch_size, ta
)


test_generator = test_data_generator.flow_from_dataframe(
    df_test, directory=data_directory, x_col=filename_column, y_col=true
    weight_col=None,class_mode='categorical', batch_size=batch_size, tar
)


train_steps=train_generator.n//train_generator.batch_size if train_gener
validation_steps=valid_generator.n//valid_generator.batch_size if valid_
```

```
Found 14968 validated image filenames belonging to 5 classes.
Found 3743 validated image filenames belonging to 5 classes.
Found 2078 validated image filenames belonging to 5 classes.
```

In [1…

```
## Set up the model architecture
### use your best model from above, and rename it here to cnn_model_a
cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', ac
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)
```

In [1…

```
## Define the model
cnn_model = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs, name
```

In […

```
## Compile the model
cnn_model.compile(optimizer=keras.optimizers.Adam(), loss='categorica
cnn_model.summary()
```

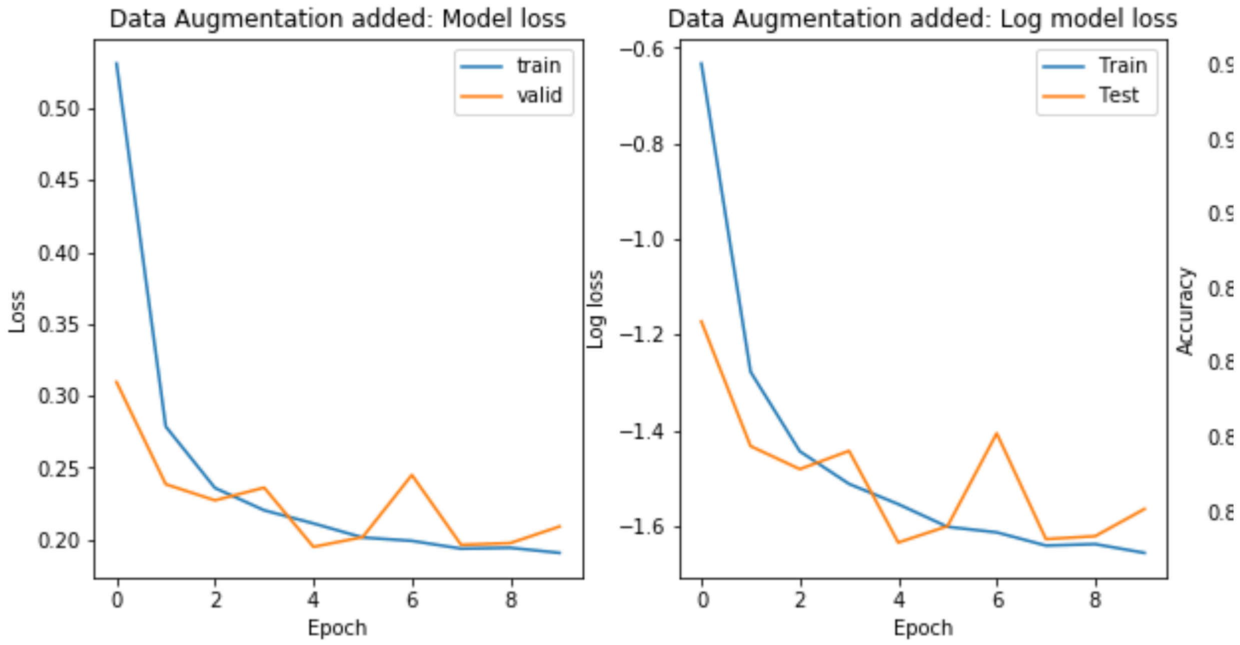
```
Model: "cnn_Model_augmented"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_44 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 32, 32, 5)         50
_________________________________________________________________
max_pooling2d_62 (MaxPooling (None, 16, 16, 5)         0
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 16, 16, 5)         230
_________________________________________________________________
max_pooling2d_63 (MaxPooling (None, 8, 8, 5)           0
_________________________________________________________________
flatten_29 (Flatten)         (None, 320)               0
_________________________________________________________________
dense_32 (Dense)             (None, 5)                 1605
=================================================================
Total params: 1,885
Trainable params: 1,885
Non-trainable params: 0
_________________________________________________________________
```

In [159]:

```
## Actually train model
epochs = 10
history = cnn_model.fit_generator(generator=train_generator,
                    steps_per_epoch= train_steps,
                    validation_data= valid_generator,
                    validation_steps= validation_steps,
                    epochs= epochs
        )
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/10
1871/1871 [==============================] - 21s 11ms/step - loss: 0.5312
- accuracy: 0.8088 - val_loss: 0.3096 - val_accuracy: 0.8967
Epoch 2/10
1871/1871 [==============================] - 21s 11ms/step - loss: 0.2787
- accuracy: 0.9088 - val_loss: 0.2385 - val_accuracy: 0.9237
Epoch 3/10
1871/1871 [==============================] - 14s 7ms/step - loss: 0.2360 -
accuracy: 0.9244 - val_loss: 0.2273 - val_accuracy: 0.9275
Epoch 4/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2204 -
accuracy: 0.9315 - val_loss: 0.2362 - val_accuracy: 0.9267
Epoch 5/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2113 -
accuracy: 0.9317 - val_loss: 0.1950 - val_accuracy: 0.9403
Epoch 6/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2015 -
accuracy: 0.9363 - val_loss: 0.2018 - val_accuracy: 0.9320
Epoch 7/10
1871/1871 [==============================] - 14s 8ms/step - loss: 0.1992 -
accuracy: 0.9369 - val_loss: 0.2450 - val_accuracy: 0.9210
Epoch 8/10
1871/1871 [==============================] - 14s 8ms/step - loss: 0.1937 -
accuracy: 0.9378 - val_loss: 0.1964 - val_accuracy: 0.9352
Epoch 9/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.1943 -
accuracy: 0.9391 - val_loss: 0.1976 - val_accuracy: 0.9371
Epoch 10/10
1871/1871 [==============================] - 16s 8ms/step - loss: 0.1908 -
accuracy: 0.9385 - val_loss: 0.2091 - val_accuracy: 0.9320
```

```
In [160]: ## Plot results
          plot_history(history, "Data Augmentation added")
```

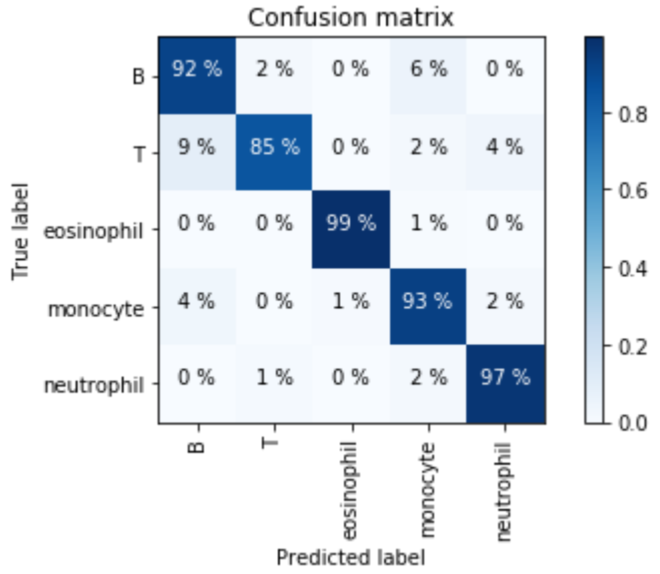


```
<Figure size 432x288 with 0 Axes>
```

In [161…

```
# plot confusion matrix
cnn_helper.plot_confusion_matrix_from_generator(cnn_model, valid_ge
```

```
Accuracy: 0.9318728292813251
```



Confusion matrix

True label \ Predicted label	B	T	eosinophil	monocyte	neutrophil
B	92 %	2 %	0 %	6 %	0 %
T	9 %	85 %	0 %	2 %	4 %
eosinophil	0 %	0 %	99 %	1 %	0 %
monocyte	4 %	0 %	1 %	93 %	2 %
neutrophil	0 %	1 %	0 %	2 %	97 %

Q: Did the data augmentation help? Why or why not? What makes this dataset more or less likely to be helped by data augmentation?

Optional hints for `question above`

1 . Are the blood cells at random places in the image? No 2 . Look at some of the images. Are the bloodcells centered? What could rotations or zooms change about this? It would change nothing 3 . Are there color changes you could compensate for?

Regularisation methods

Both BatchNormalization and DropOut are two different regularisation methods. Try adding both to the best working CNN model.

Read more about BatchNormalization here: https://keras.io/api/layers/normalization_layers/batch_normalization/ Read more about DropOut here:https://keras.io/api/layers/regularization_layers/dropout/

Q: What are the main similarities and differences between these methods?

In [1…

```
# Create the model here
## Set up the model architecture
### use your best model from above

cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', ac
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Dropout(.1)(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)
```

In [1…

```
cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', ac
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Dropout(.2)(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)
```

In [16…

```
## Define the model
cnn_model = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs, nam
```

In …

```
## Compile the model

cnn_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001
cnn_model.summary()
print(cnn_model.layers)
```

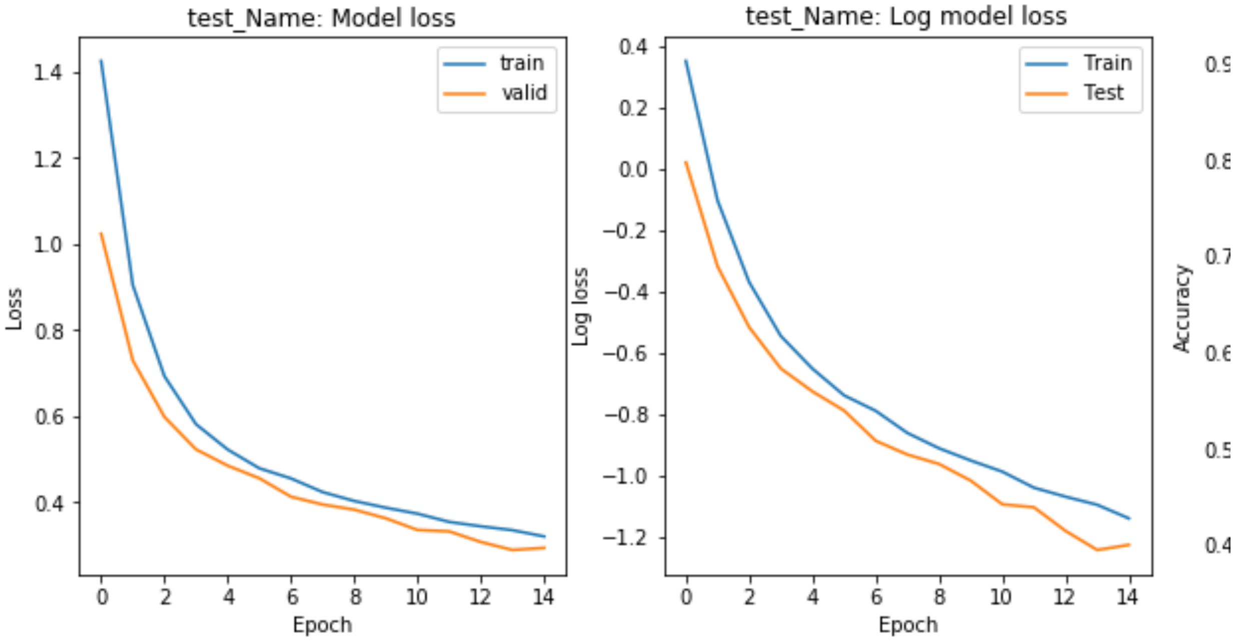
```
Model: "cnn_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_46 (InputLayer)        [(None, 32, 32, 1)]       0         
_________________________________________________________________
conv2d_70 (Conv2D)           (None, 32, 32, 5)         50        
_________________________________________________________________
max_pooling2d_66 (MaxPooling (None, 16, 16, 5)         0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 5)         0         
_________________________________________________________________
batch_normalization_15 (Batc (None, 16, 16, 5)         20        
_________________________________________________________________
conv2d_71 (Conv2D)           (None, 16, 16, 5)         230       
_________________________________________________________________
max_pooling2d_67 (MaxPooling (None, 8, 8, 5)           0         
_________________________________________________________________
flatten_31 (Flatten)         (None, 320)               0         
_________________________________________________________________
dense_34 (Dense)             (None, 5)                 1605      
=================================================================
Total params: 1,905
Trainable params: 1,895
Non-trainable params: 10
_________________________________________________________________
[<tensorflow.python.keras.engine.input_layer.InputLayer object at
0x7f40a427fb70>, <tensorflow.python.keras.layers.convolutional.Conv2D
object at 0x7f40a427f3c8>,
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at
0x7f40a427fa58>, <tensorflow.python.keras.layers.core.Dropout object at
0x7f40a43983c8>,
<tensorflow.python.keras.layers.normalization_v2.BatchNormalization object
at 0x7f40a4398828>, <tensorflow.python.keras.layers.convolutional.Conv2D
object at 0x7f40a4398358>,
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at
0x7f40a46f60f0>, <tensorflow.python.keras.layers.core.Flatten object at
0x7f407c793d30>, <tensorflow.python.keras.layers.core.Dense object at
0x7f40540d1240>]
```

```
In [166]: ## Actually train model
          epochs = 15
          history = cnn_model.fit_generator(generator=train_generator,
                              steps_per_epoch= train_steps,
                              validation_data= valid_generator,
                              validation_steps= validation_steps,
                              epochs= epochs
                  )
```

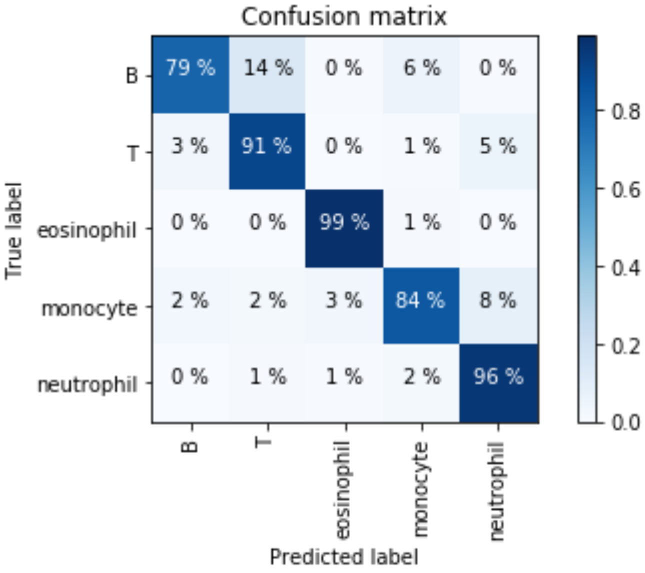
```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/15
1871/1871 [==============================] - 26s 14ms/step - loss: 1.4251
- accuracy: 0.3952 - val_loss: 1.0239 - val_accuracy: 0.6089
Epoch 2/15
1871/1871 [==============================] - 15s 8ms/step - loss: 0.9046 -
accuracy: 0.6472 - val_loss: 0.7296 - val_accuracy: 0.7184
Epoch 3/15
1871/1871 [==============================] - 15s 8ms/step - loss: 0.6930 -
accuracy: 0.7455 - val_loss: 0.5983 - val_accuracy: 0.7749
Epoch 4/15
1871/1871 [==============================] - 15s 8ms/step - loss: 0.5814 -
accuracy: 0.7925 - val_loss: 0.5229 - val_accuracy: 0.8126
Epoch 5/15
1871/1871 [==============================] - 16s 8ms/step - loss: 0.5226 -
accuracy: 0.8210 - val_loss: 0.4852 - val_accuracy: 0.8231
Epoch 6/15
1871/1871 [==============================] - 15s 8ms/step - loss: 0.4790 -
accuracy: 0.8383 - val_loss: 0.4559 - val_accuracy: 0.8346
Epoch 7/15
1871/1871 [==============================] - 14s 7ms/step - loss: 0.4555 -
accuracy: 0.8447 - val_loss: 0.4131 - val_accuracy: 0.8552
Epoch 8/15
1871/1871 [==============================] - 34s 18ms/step - loss: 0.4238
- accuracy: 0.8570 - val_loss: 0.3950 - val_accuracy: 0.8603
Epoch 9/15
1871/1871 [==============================] - 15s 8ms/step - loss: 0.4031 -
accuracy: 0.8646 - val_loss: 0.3830 - val_accuracy: 0.8643
Epoch 10/15
1871/1871 [==============================] - 14s 8ms/step - loss: 0.3874 -
accuracy: 0.8709 - val_loss: 0.3629 - val_accuracy: 0.8721
Epoch 11/15
1871/1871 [==============================] - 14s 8ms/step - loss: 0.3736 -
accuracy: 0.8777 - val_loss: 0.3358 - val_accuracy: 0.8838
Epoch 12/15
1871/1871 [==============================] - 14s 8ms/step - loss: 0.3545 -
accuracy: 0.8815 - val_loss: 0.3326 - val_accuracy: 0.8844
Epoch 13/15
1871/1871 [==============================] - 16s 8ms/step - loss: 0.3443 -
accuracy: 0.8860 - val_loss: 0.3079 - val_accuracy: 0.8964
Epoch 14/15
1871/1871 [==============================] - 16s 8ms/step - loss: 0.3353 -
accuracy: 0.8912 - val_loss: 0.2894 - val_accuracy: 0.9028
Epoch 15/15
1871/1871 [==============================] - 15s 8ms/step - loss: 0.3208 -
accuracy: 0.8958 - val_loss: 0.2942 - val_accuracy: 0.8980
```

In [167…

```python
## Plot results
plot_history(history, "test_Name")

# plot confusion matrix
cnn_helper.plot_confusion_matrix_from_generator(cnn_model, valid_ge
```



```
Accuracy: 0.8979428266096714
```



	B	T	eosinophil	monocyte	neutrophil
B	79 %	14 %	0 %	6 %	0 %
T	3 %	91 %	0 %	1 %	5 %
eosinophil	0 %	0 %	99 %	1 %	0 %
monocyte	2 %	2 %	3 %	84 %	8 %
neutrophil	0 %	1 %	1 %	2 %	96 %

Q: Is there such a thing as too much regularisation?

Visualise your best CNN

Use the code below to visualise some of the weights you have trained. Hint: Weights are present in convolutional filters and dense layers, nowhere else.

Visualize both one layer with filters, and the outputlayer

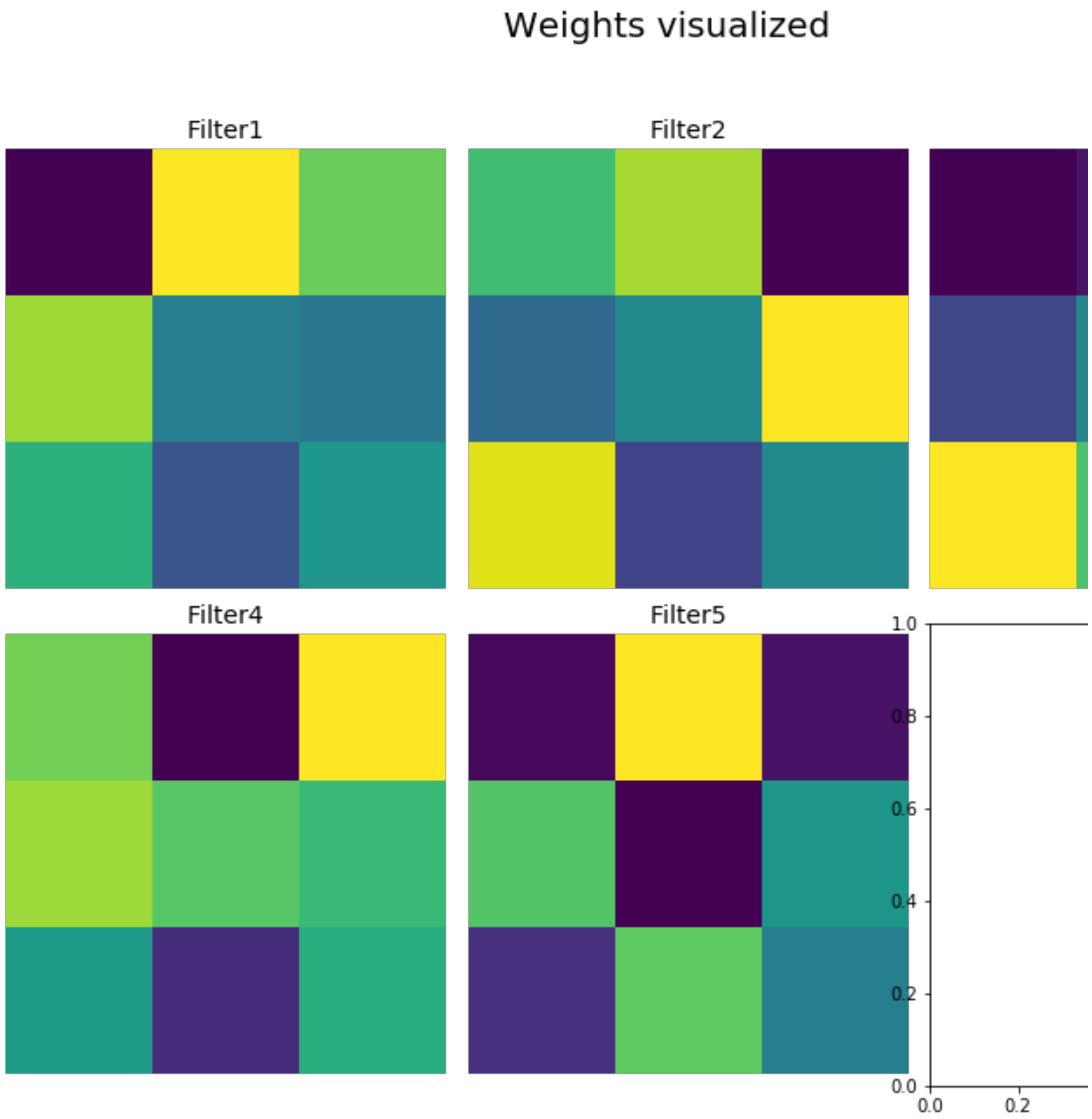
In [1…

```
# Pick the layer
print(cnn_model.layers)
cw1 = np.array(cnn_model.layers[1].get_weights()) ## Pick the layer w
print(cw1.shape) # 2 weight, 1 weight, 1 bias
print(cw1[0].shape) # Weights
print(cw1[1].shape) # Biases
matrix = cw1[0]
```

```
[<tensorflow.python.keras.engine.input_layer.InputLayer object at
0x7f40a427fb70>, <tensorflow.python.keras.layers.convolutional.Conv2D
object at 0x7f40a427f3c8>,
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at
0x7f40a427fa58>, <tensorflow.python.keras.layers.core.Dropout object at
0x7f40a43983c8>,
<tensorflow.python.keras.layers.normalization_v2.BatchNormalization object
at 0x7f40a4398828>, <tensorflow.python.keras.layers.convolutional.Conv2D
object at 0x7f40a4398358>,
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at
0x7f40a46f60f0>, <tensorflow.python.keras.layers.core.Flatten object at
0x7f407c793d30>, <tensorflow.python.keras.layers.core.Dense object at
0x7f40540d1240>]
(2,)
(3, 3, 1, 5)
(5,)
```

In [169…

```
# Plot your filters
figure, ax = plt.subplots(2, 3, figsize=(14, 10))
figure.suptitle("Weights visualized", fontsize=20)
axes = ax.ravel()

for i in range(0, 5): # Range should be 0 - the number of filters y
    image = matrix[:,:,:,i:i+1]
    image = np.reshape(image, (3, 3)) ## Reshape to the size of you
    axes[i].set_title("Filter" + str(i+1), fontsize=14)
    axes[i].imshow(image)
    axes[i].set_axis_off()

plt.subplots_adjust(wspace=0.05, hspace=0.05)
plt.show()
plt.close()
```



Using existing models

One great thing to do when making a CNN model is to use an architecture that has worked for simmilar cases. I happen to know that the existing CNN model VGG16 is a good model for these types of images, try that one next.

There are many way of visualising neural networks, see https://datascience.stackexchange.com/questions/12851/how-do-you-visualize-neural-network-architectures, but here is one made by Christophe Avenel

VGG 16

```
In [101]:  vgg_model = keras.applications.VGG16(
               include_top=False,
               weights=None,
               input_shape=(32, 32, 1),
               pooling=None,
           )
```

```
In [102]:  # add new classifier layers
           flat1 = layers.Flatten()(vgg_model.layers[-1].output)
           class1 = layers.Dense(1024, activation='relu')(flat1)
           output = layers.Dense(5, activation='softmax')(class1)
```

```
In [103]:  vgg_model = keras.Model(inputs=vgg_model.inputs, outputs=output)

           print (vgg_model.summary())
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_31 (InputLayer)        [(None, 32, 32, 1)]       0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        640       
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0         
_________________________________________________________________
flatten_17 (Flatten)         (None, 512)               0         
_________________________________________________________________
dense_17 (Dense)             (None, 1024)              525312    
_________________________________________________________________
dense_18 (Dense)             (None, 5)                 5125      
=================================================================
Total params: 15,243,973
Trainable params: 15,243,973
Non-trainable params: 0
_________________________________________________________________
None
```

Q: How many parameters does this model have?

In …

```
## Compile the model

vgg_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001
```

Q: Why do we need a new classification layers?

Optional hints for `question above`

1 . What is the original network classifying? 2 . What do we want to classify?

Optional hints for `The hint, if you need it`

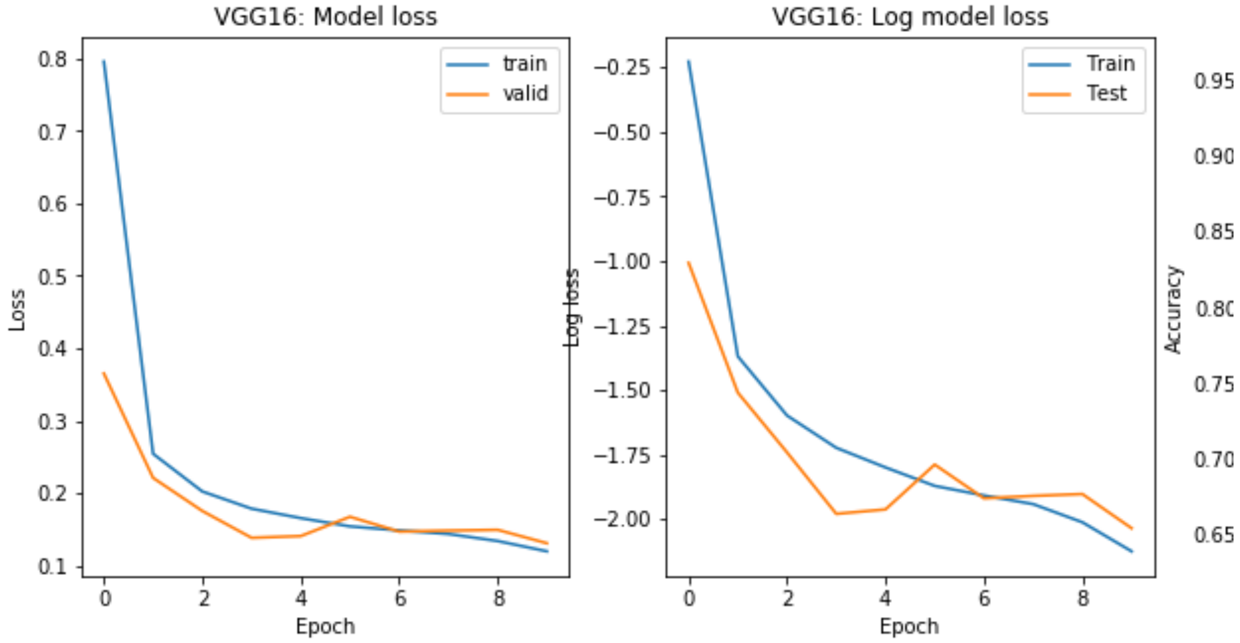
1 . So how do we remove the previous classification and make the new one? Just like the code above naturally! A flattening layer is almost always followed by a dense layer or two to expand the model, and then a final classification layer.

In [105]:

```
## Actually train model
epochs = 10
history = vgg_model.fit_generator(generator=train_generator,
                                  steps_per_epoch= train_steps,
                                  validation_data= valid_generator,
                                  validation_steps= validation_steps,
                                  epochs= epochs
                    )
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/10
1871/1871 [==============================] - 34s 18ms/step - loss: 0.7961
- accuracy: 0.6388 - val_loss: 0.3651 - val_accuracy: 0.8959
Epoch 2/10
1871/1871 [==============================] - 32s 17ms/step - loss: 0.2542
- accuracy: 0.9170 - val_loss: 0.2209 - val_accuracy: 0.9221
Epoch 3/10
1871/1871 [==============================] - 32s 17ms/step - loss: 0.2022
- accuracy: 0.9369 - val_loss: 0.1753 - val_accuracy: 0.9425
Epoch 4/10
1871/1871 [==============================] - 32s 17ms/step - loss: 0.1784
- accuracy: 0.9429 - val_loss: 0.1382 - val_accuracy: 0.9553
Epoch 5/10
1871/1871 [==============================] - 33s 18ms/step - loss: 0.1654
- accuracy: 0.9486 - val_loss: 0.1406 - val_accuracy: 0.9569
Epoch 6/10
1871/1871 [==============================] - 31s 17ms/step - loss: 0.1540
- accuracy: 0.9502 - val_loss: 0.1672 - val_accuracy: 0.9499
Epoch 7/10
1871/1871 [==============================] - 32s 17ms/step - loss: 0.1484
- accuracy: 0.9530 - val_loss: 0.1467 - val_accuracy: 0.9561
Epoch 8/10
1871/1871 [==============================] - 32s 17ms/step - loss: 0.1435
- accuracy: 0.9544 - val_loss: 0.1481 - val_accuracy: 0.9497
Epoch 9/10
1871/1871 [==============================] - 39s 21ms/step - loss: 0.1337
- accuracy: 0.9568 - val_loss: 0.1492 - val_accuracy: 0.9526
Epoch 10/10
1871/1871 [==============================] - 33s 17ms/step - loss: 0.1195
- accuracy: 0.9625 - val_loss: 0.1307 - val_accuracy: 0.9599
```

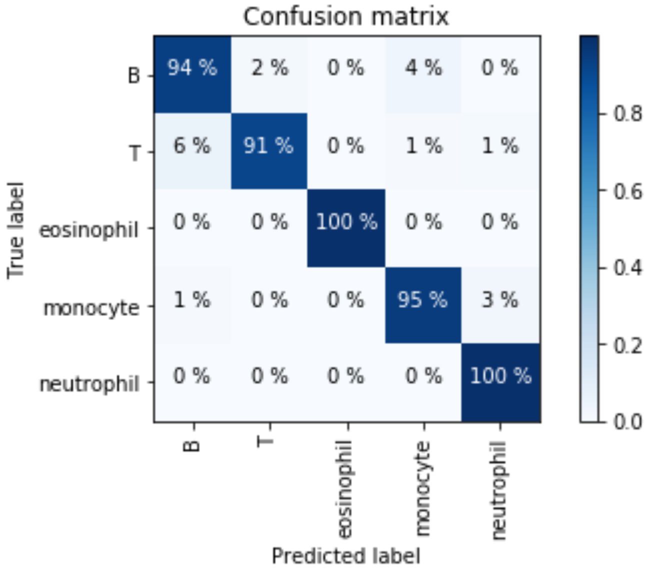
```
In [106]:  ## Plot results
            plot_history(history, "VGG16")
```



```
<Figure size 432x288 with 0 Axes>
```

```
In [107…  # plot confusion matrix
          cnn_helper.plot_confusion_matrix_from_generator(vgg_model, valid_ge
```

```
Accuracy: 0.9599251936948971
```



Q: What is your worst performing class in this classifier? Is it the same as in the other ones?

Q: How many layers with 10 filters of size $3 * 3$ would you have to add to the first CNN model we designed to achieve the same number of parameters?

Try some more models.

Try other optimizers, learning rates, batch sizes or number of epochs. Which would you like to try first and why? Show atleast 4 models

In […

```
## Set up the model architecture
##
# Model try 1
# Accuracy validation: 0.9433
# Comment: Gives good result but the model seems over fitted so for ne
##

cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', act
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)

## Define the model
cnn_model_final = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs,

## Compile the model
cnn_model_final.compile(optimizer=keras.optimizers.Adam(), loss='categ
cnn_model_final.summary()
```

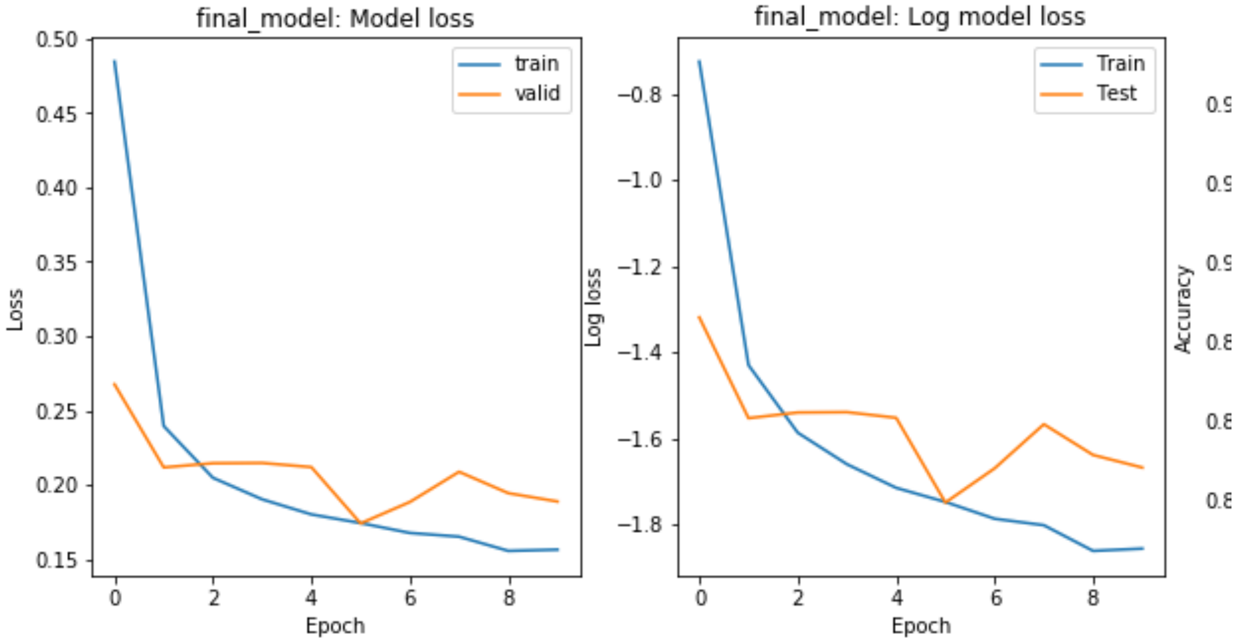
```
Model: "final_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_8 (InputLayer)         [(None, 32, 32, 1)]       0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 32, 32, 5)         50        
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 16, 16, 5)         0         
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 5)         230       
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 8, 8, 5)           0         
_________________________________________________________________
flatten_7 (Flatten)          (None, 320)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 5)                 1605      
=================================================================
Total params: 1,885
Trainable params: 1,885
Non-trainable params: 0
_________________________________________________________________
```

```
In [34]: ## Actually train model
         epochs = 10
         history = cnn_model_final.fit_generator(generator=train_generator,
                                     steps_per_epoch= train_steps,
                                     validation_data= valid_generator,
                                     validation_steps= validation_steps,
                                     epochs= epochs,
                 )
```

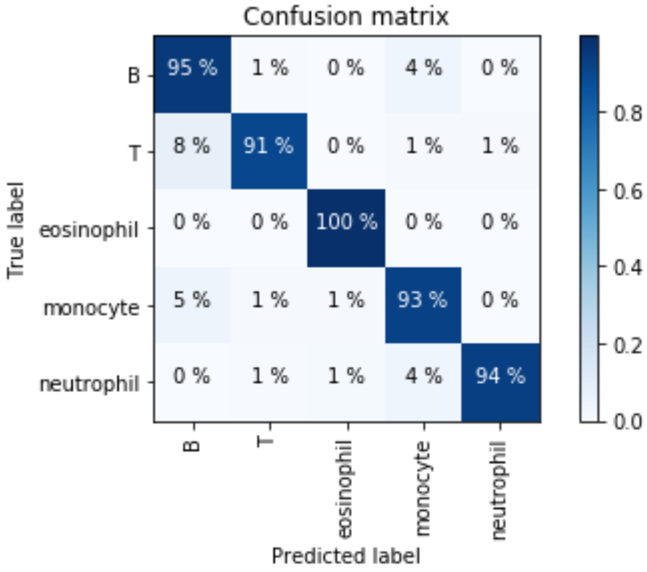
```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/10
1871/1871 [==============================] - 24s 13ms/step - loss: 0.4846
- accuracy: 0.8270 - val_loss: 0.2676 - val_accuracy: 0.9141
Epoch 2/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2395 -
accuracy: 0.9250 - val_loss: 0.2117 - val_accuracy: 0.9301
Epoch 3/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2047 -
accuracy: 0.9357 - val_loss: 0.2145 - val_accuracy: 0.9293
Epoch 4/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.1902 -
accuracy: 0.9393 - val_loss: 0.2147 - val_accuracy: 0.9296
Epoch 5/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.1801 -
accuracy: 0.9423 - val_loss: 0.2119 - val_accuracy: 0.9293
Epoch 6/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.1743 -
accuracy: 0.9430 - val_loss: 0.1740 - val_accuracy: 0.9435
Epoch 7/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.1676 -
accuracy: 0.9451 - val_loss: 0.1884 - val_accuracy: 0.9392
Epoch 8/10
1871/1871 [==============================] - 16s 8ms/step - loss: 0.1651 -
accuracy: 0.9464 - val_loss: 0.2087 - val_accuracy: 0.9296
Epoch 9/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.1555 -
accuracy: 0.9507 - val_loss: 0.1943 - val_accuracy: 0.9382
Epoch 10/10
1871/1871 [==============================] - 16s 8ms/step - loss: 0.1564 -
accuracy: 0.9495 - val_loss: 0.1887 - val_accuracy: 0.9433
```

In [35…

```python
## Plot results
plot_history(history, "final_model")

# plot confusion matrix
cnn_helper.plot_confusion_matrix_from_generator(cnn_model_final, val
```



```
Accuracy: 0.943093775046754
```



	B	T	eosinophil	monocyte	neutrophil
B	95 %	1 %	0 %	4 %	0 %
T	8 %	91 %	0 %	1 %	1 %
eosinophil	0 %	0 %	100 %	0 %	0 %
monocyte	5 %	1 %	1 %	93 %	0 %
neutrophil	0 %	1 %	1 %	4 %	94 %

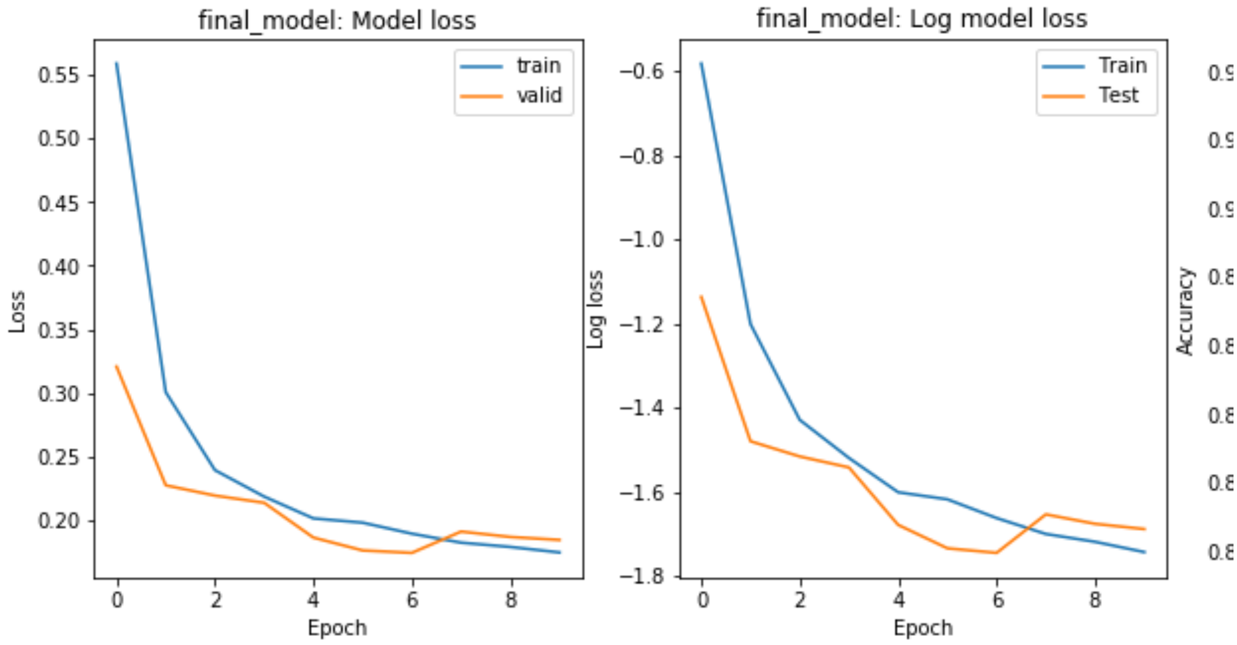
```
In […  ## Set up the model architecture
       ##
       # Model try 2
       # Added drop out layer of 0.1 rate.
       # Accuracy validation: 0.9384
       # Comment: A bit worse accuracy, but this model does not over-fit the
       ##

       cnn_inputs = keras.Input(shape=(32,32,1))
       x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', act
       x = layers.MaxPooling2D(pool_size=(2, 2))(x)
       x = layers.Dropout(.1)(x)
       x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
       x = layers.MaxPooling2D(pool_size=(2, 2))(x)
       x = layers.Flatten()(x)
       cnn_outputs = layers.Dense(5, activation='softmax')(x)

       ## Define the model
       cnn_model_final = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs,

       ## Compile the model
       cnn_model_final.compile(optimizer=keras.optimizers.Adam(), loss='categ
       cnn_model_final.summary()
```

```
Model: "final_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_15 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 32, 32, 5)         50
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 16, 16, 5)         0
_________________________________________________________________
dropout_12 (Dropout)         (None, 16, 16, 5)         0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 16, 16, 5)         230
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 8, 8, 5)           0
_________________________________________________________________
flatten_14 (Flatten)         (None, 320)               0
_________________________________________________________________
dense_14 (Dense)             (None, 5)                 1605
=================================================================
Total params: 1,885
Trainable params: 1,885
Non-trainable params: 0
_________________________________________________________________
```

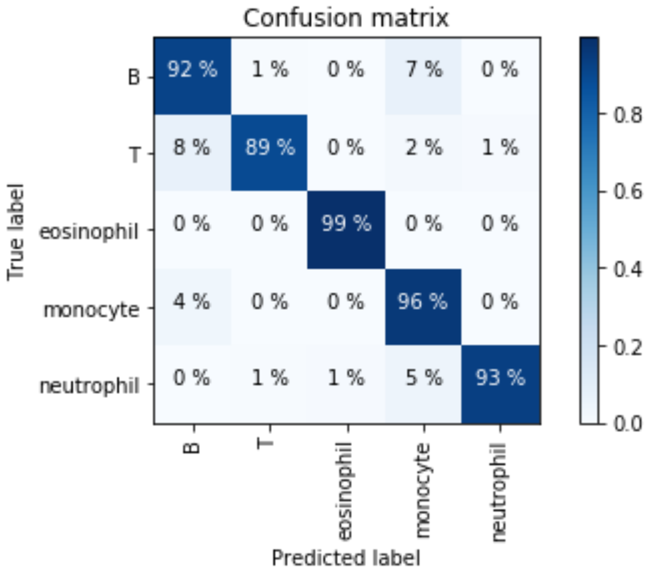
```
In [55]:  ## Actually train model
          epochs = 10
          history = cnn_model_final.fit_generator(generator=train_generator,
                                    steps_per_epoch= train_steps,
                                    validation_data= valid_generator,
                                    validation_steps= validation_steps,
                                    epochs= epochs,
                      )
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/10
1871/1871 [==============================] - 19s 10ms/step - loss: 0.5587
- accuracy: 0.7994 - val_loss: 0.3209 - val_accuracy: 0.8953
Epoch 2/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.3008 -
accuracy: 0.9033 - val_loss: 0.2277 - val_accuracy: 0.9312
Epoch 3/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2396 -
accuracy: 0.9228 - val_loss: 0.2196 - val_accuracy: 0.9264
Epoch 4/10
1871/1871 [==============================] - 16s 8ms/step - loss: 0.2188 -
accuracy: 0.9309 - val_loss: 0.2140 - val_accuracy: 0.9248
Epoch 5/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2017 -
accuracy: 0.9352 - val_loss: 0.1867 - val_accuracy: 0.9374
Epoch 6/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.1984 -
accuracy: 0.9382 - val_loss: 0.1766 - val_accuracy: 0.9408
Epoch 7/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.1897 -
accuracy: 0.9379 - val_loss: 0.1747 - val_accuracy: 0.9376
Epoch 8/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.1827 -
accuracy: 0.9410 - val_loss: 0.1914 - val_accuracy: 0.9382
Epoch 9/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.1793 -
accuracy: 0.9401 - val_loss: 0.1871 - val_accuracy: 0.9347
Epoch 10/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.1750 -
accuracy: 0.9425 - val_loss: 0.1848 - val_accuracy: 0.9384
```

```
In [56… ## Plot results
        plot_history(history, "final_model")

        # plot confusion matrix
        cnn_helper.plot_confusion_matrix_from_generator(cnn_model_final, val
```



```
Accuracy: 0.9382847982901416
```



```
In […  ## Set up the model architecture
       ##
       # Model try 3
       # Changed the Conv2D kernel to see if there is any change in performen
       # Accuracy validation: 0.9066
       # Comment: Did nit improve the performence, conclude that kernel size
       ##

       cnn_inputs = keras.Input(shape=(32,32,1))
       x = layers.Conv2D(5, kernel_size=(4, 4), strides=2,padding='same', act
       x = layers.MaxPooling2D(pool_size=(2, 2))(x)
       x = layers.Dropout(.1)(x)
       x = layers.Conv2D(5, kernel_size=(4, 4), strides=2,padding='same')(x)
       x = layers.MaxPooling2D(pool_size=(2, 2))(x)
       x = layers.Flatten()(x)
       cnn_outputs = layers.Dense(5, activation='softmax')(x)

       ## Define the model
       cnn_model_final = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs,

       ## Compile the model
       cnn_model_final.compile(optimizer=keras.optimizers.Adam(), loss='categ
       cnn_model_final.summary()
```

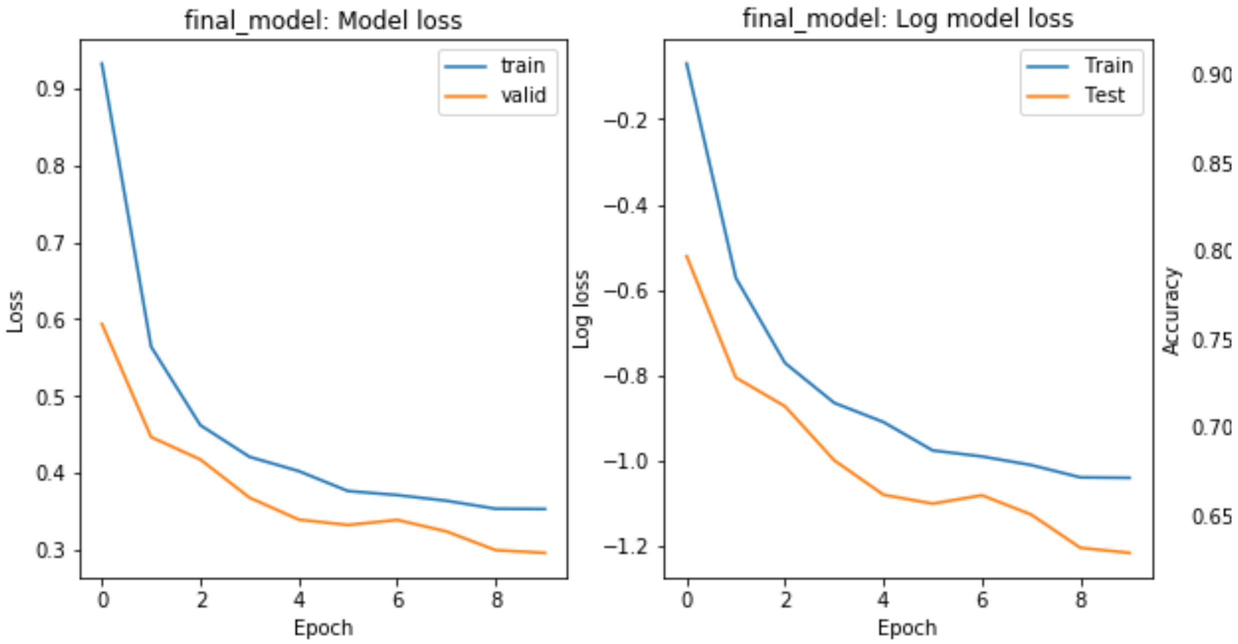
```
Model: "final_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_16 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 16, 16, 5)         85
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 8, 8, 5)           0
_________________________________________________________________
dropout_13 (Dropout)         (None, 8, 8, 5)           0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 4, 4, 5)           405
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 2, 2, 5)           0
_________________________________________________________________
flatten_15 (Flatten)         (None, 20)                0
_________________________________________________________________
dense_15 (Dense)             (None, 5)                 105
=================================================================
Total params: 595
Trainable params: 595
Non-trainable params: 0
_________________________________________________________________
```

```
In [58]:  ## Actually train model
          epochs = 10
          history = cnn_model_final.fit_generator(generator=train_generator,
                                    steps_per_epoch= train_steps,
                                    validation_data= valid_generator,
                                    validation_steps= validation_steps,
                                    epochs= epochs,
                      )
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/10
1871/1871 [==============================] - 23s 12ms/step - loss: 0.9322
- accuracy: 0.6289 - val_loss: 0.5939 - val_accuracy: 0.7968
Epoch 2/10
1871/1871 [==============================] - 14s 8ms/step - loss: 0.5645 -
accuracy: 0.7974 - val_loss: 0.4470 - val_accuracy: 0.8539
Epoch 3/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.4625 -
accuracy: 0.8429 - val_loss: 0.4179 - val_accuracy: 0.8643
Epoch 4/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.4213 -
accuracy: 0.8582 - val_loss: 0.3683 - val_accuracy: 0.8790
Epoch 5/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.4028 -
accuracy: 0.8693 - val_loss: 0.3398 - val_accuracy: 0.8967
Epoch 6/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.3769 -
accuracy: 0.8771 - val_loss: 0.3328 - val_accuracy: 0.8969
Epoch 7/10
1871/1871 [==============================] - 14s 8ms/step - loss: 0.3717 -
accuracy: 0.8769 - val_loss: 0.3394 - val_accuracy: 0.8895
Epoch 8/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.3643 -
accuracy: 0.8807 - val_loss: 0.3244 - val_accuracy: 0.8972
Epoch 9/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.3539 -
accuracy: 0.8850 - val_loss: 0.3001 - val_accuracy: 0.9066
Epoch 10/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.3535 -
accuracy: 0.8854 - val_loss: 0.2966 - val_accuracy: 0.9066
```

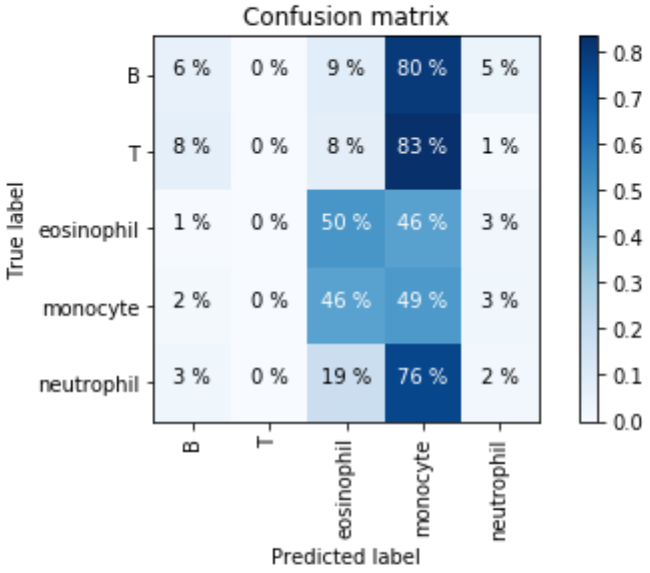
In [61…

```
## Plot results
plot_history(history, "final_model")

# plot confusion matrix
cnn_helper.plot_confusion_matrix_from_generator(cnn_model_final, val
```


final_model: Model loss

final_model: Log model loss

```
Accuracy: 0.21212930080416778
```


Confusion matrix

True label \ Predicted label	B	T	eosinophil	monocyte	neutrophil
B	6 %	0 %	9 %	80 %	5 %
T	8 %	0 %	8 %	83 %	1 %
eosinophil	1 %	0 %	50 %	46 %	3 %
monocyte	2 %	0 %	46 %	49 %	3 %
neutrophil	3 %	0 %	19 %	76 %	2 %

In […

```
## Set up the model architecture
##
# Model try 4
# Try another optimizer instead of Adam try SGD
# Accuracy validation: 0.9344
# Comment: Not a whole lot worse than model 2, but the difference is e
##

cnn_inputs = keras.Input(shape=(32,32,1))
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', act
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Dropout(.1)(x)
x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)
cnn_outputs = layers.Dense(5, activation='softmax')(x)

## Define the model
cnn_model_final = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs,

## Compile the model
cnn_model_final.compile(optimizer=keras.optimizers.SGD(), loss='catego
cnn_model_final.summary()
```

```
Model: "final_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_18 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 32, 32, 5)         50
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 16, 16, 5)         0
_________________________________________________________________
dropout_15 (Dropout)         (None, 16, 16, 5)         0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 16, 16, 5)         230
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 8, 8, 5)           0
_________________________________________________________________
flatten_17 (Flatten)         (None, 320)               0
_________________________________________________________________
dense_17 (Dense)             (None, 5)                 1605
=================================================================
Total params: 1,885
Trainable params: 1,885
Non-trainable params: 0
_________________________________________________________________
```

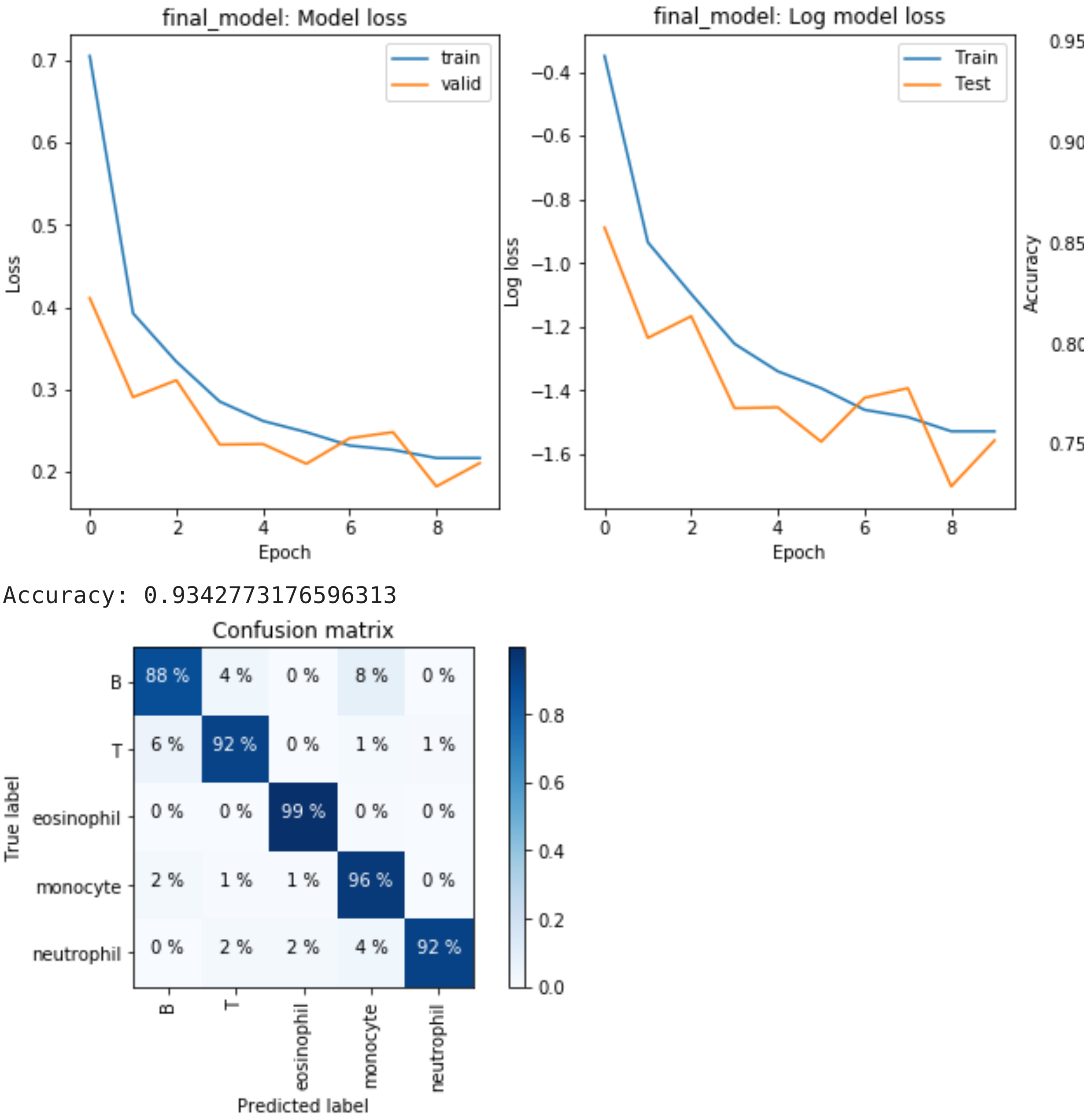
In [63]:

```
## Actually train model
epochs = 10
history = cnn_model_final.fit_generator(generator=train_generator,
                    steps_per_epoch= train_steps,
                    validation_data= valid_generator,
                    validation_steps= validation_steps,
                    epochs= epochs,
        )
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/10
1871/1871 [==============================] - 23s 12ms/step - loss: 0.7054
- accuracy: 0.7291 - val_loss: 0.4113 - val_accuracy: 0.8627
Epoch 2/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.3928 -
accuracy: 0.8626 - val_loss: 0.2909 - val_accuracy: 0.9071
Epoch 3/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.3340 -
accuracy: 0.8844 - val_loss: 0.3114 - val_accuracy: 0.8916
Epoch 4/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2856 -
accuracy: 0.9018 - val_loss: 0.2333 - val_accuracy: 0.9240
Epoch 5/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2620 -
accuracy: 0.9122 - val_loss: 0.2339 - val_accuracy: 0.9251
Epoch 6/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2484 -
accuracy: 0.9171 - val_loss: 0.2100 - val_accuracy: 0.9248
Epoch 7/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2321 -
accuracy: 0.9202 - val_loss: 0.2411 - val_accuracy: 0.9234
Epoch 8/10
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2269 -
accuracy: 0.9227 - val_loss: 0.2485 - val_accuracy: 0.9154
Epoch 9/10
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2170 -
accuracy: 0.9279 - val_loss: 0.1825 - val_accuracy: 0.9427
Epoch 10/10
1871/1871 [==============================] - 14s 8ms/step - loss: 0.2170 -
accuracy: 0.9271 - val_loss: 0.2109 - val_accuracy: 0.9344
```

In [64…

```
## Plot results
plot_history(history, "final_model")

# plot confusion matrix
cnn_helper.plot_confusion_matrix_from_generator(cnn_model_final, val
```



```
Accuracy: 0.9342773176596313
```

Confusion matrix

	B	T	eosinophil	monocyte	neutrophil
B	88 %	4 %	0 %	8 %	0 %
T	6 %	92 %	0 %	1 %	1 %
eosinophil	0 %	0 %	99 %	0 %	0 %
monocyte	2 %	1 %	1 %	96 %	0 %
neutrophil	0 %	2 %	2 %	4 %	92 %

Finally test your best model

```
In […  ## Set up the model architecture
       ##
       # Took the best model and increased the number of epochs
       # Accuracy validation: 0.9406
       ##

       cnn_inputs = keras.Input(shape=(32,32,1))
       x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same', act
       x = layers.MaxPooling2D(pool_size=(2, 2))(x)
       x = layers.Dropout(.2)(x)
       x = layers.Conv2D(5, kernel_size=(3, 3), strides=1,padding='same')(x)
       x = layers.MaxPooling2D(pool_size=(2, 2))(x)
       x = layers.Flatten()(x)
       cnn_outputs = layers.Dense(5, activation='softmax')(x)

       ## Define the model
       cnn_model_final = keras.Model(inputs=cnn_inputs, outputs=cnn_outputs,

       ## Compile the model
       cnn_model_final.compile(optimizer=keras.optimizers.Adam(), loss='categ
       cnn_model_final.summary()
```

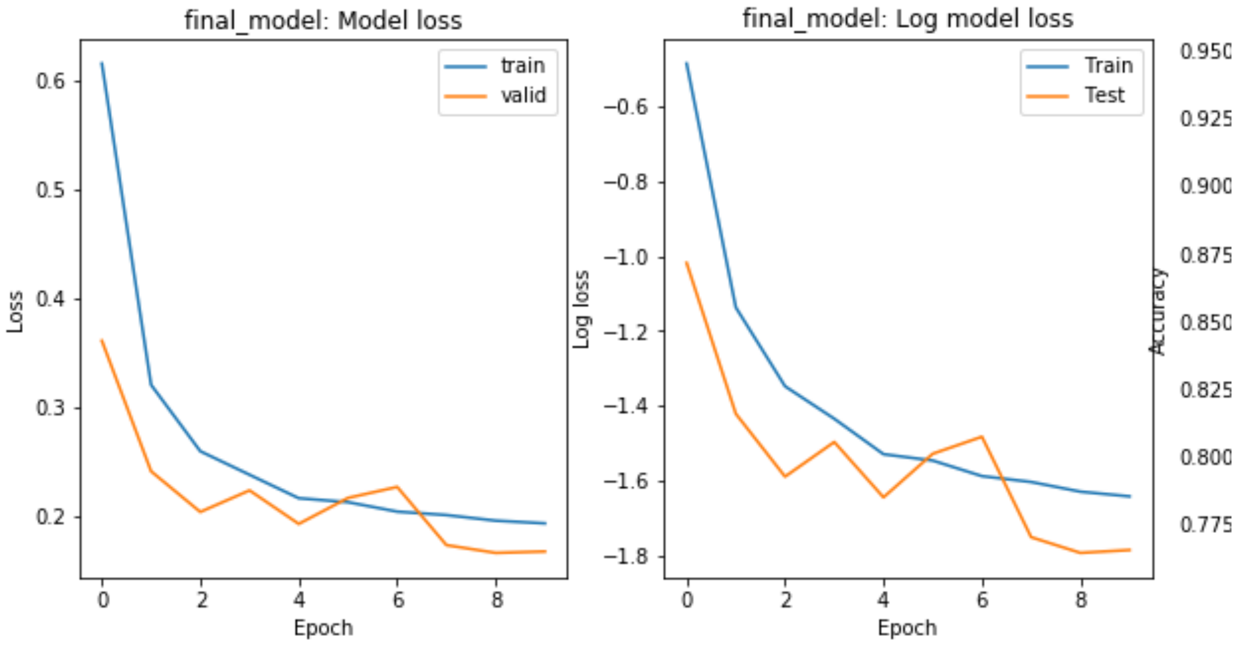
```
Model: "final_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_20 (InputLayer)        [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 32, 32, 5)         50
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 16, 16, 5)         0
_________________________________________________________________
dropout_17 (Dropout)         (None, 16, 16, 5)         0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 16, 16, 5)         230
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 8, 8, 5)           0
_________________________________________________________________
flatten_19 (Flatten)         (None, 320)               0
_________________________________________________________________
dense_19 (Dense)             (None, 5)                 1605
=================================================================
Total params: 1,885
Trainable params: 1,885
Non-trainable params: 0
_________________________________________________________________
```

```
In [69]:  ## Actually train model
          epochs = 10
          history = cnn_model_final.fit_generator(generator=train_generator,
                              steps_per_epoch= train_steps,
                              validation_data= valid_generator,
                              validation_steps= validation_steps,
                              epochs= epochs,
                  )
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.6162 -
accuracy: 0.7646 - val_loss: 0.3613 - val_accuracy: 0.8755
Epoch 2/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.3207 -
accuracy: 0.8935 - val_loss: 0.2413 - val_accuracy: 0.9200
Epoch 3/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2597 -
accuracy: 0.9133 - val_loss: 0.2039 - val_accuracy: 0.9358
Epoch 4/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2381 -
accuracy: 0.9204 - val_loss: 0.2237 - val_accuracy: 0.9221
Epoch 5/10
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2165 -
accuracy: 0.9292 - val_loss: 0.1929 - val_accuracy: 0.9363
Epoch 6/10
1871/1871 [==============================] - 18s 10ms/step - loss: 0.2129
- accuracy: 0.9290 - val_loss: 0.2168 - val_accuracy: 0.9205
Epoch 7/10
1871/1871 [==============================] - 18s 9ms/step - loss: 0.2042 -
accuracy: 0.9317 - val_loss: 0.2269 - val_accuracy: 0.9256
Epoch 8/10
1871/1871 [==============================] - 18s 10ms/step - loss: 0.2010
- accuracy: 0.9331 - val_loss: 0.1734 - val_accuracy: 0.9435
Epoch 9/10
1871/1871 [==============================] - 19s 10ms/step - loss: 0.1959
- accuracy: 0.9346 - val_loss: 0.1663 - val_accuracy: 0.9451
Epoch 10/10
1871/1871 [==============================] - 18s 10ms/step - loss: 0.1934
- accuracy: 0.9361 - val_loss: 0.1675 - val_accuracy: 0.9438
```

In […

```
## Plot results
plot_history(history, "final_model")

# plot confusion matrix
cnn_helper.plot_confusion_matrix_from_generator(cnn_model_final, vali
```

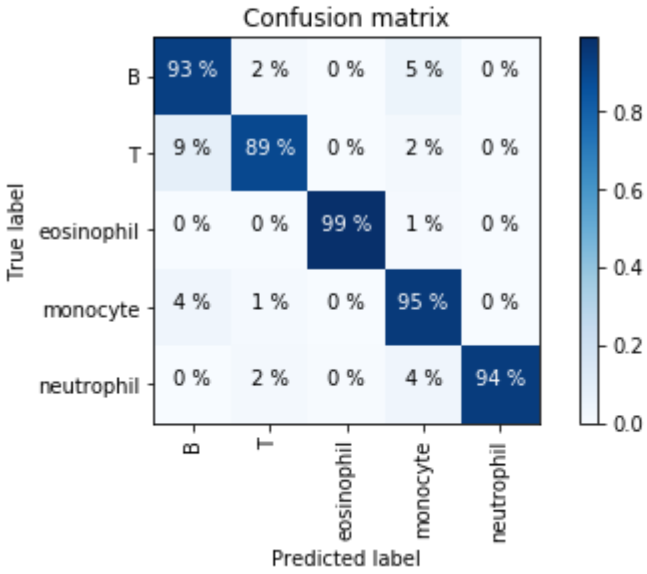


```
In [… test_steps=test_generator.n//test_generator.batch_size if test_genera

       pred=cnn_model_final.predict_generator(test_generator,
       steps=test_steps,
       verbose=1)
```

```
WARNING:tensorflow:From <ipython-input-221-c5d91644e355>:5:
Model.predict_generator (from tensorflow.python.keras.engine.training) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
259/259 [==============================] - 2s 8ms/step
```

```
In [… cnn_helper.plot_confusion_matrix_from_generator(cnn_model_final, test_
```

```
Accuracy: 0.9379210779595765
```


Confusion matrix

True label \ Predicted label	B	T	eosinophil	monocyte	neutrophil
B	93 %	2 %	0 %	5 %	0 %
T	9 %	89 %	0 %	2 %	0 %
eosinophil	0 %	0 %	99 %	1 %	0 %
monocyte	4 %	1 %	0 %	95 %	0 %
neutrophil	0 %	2 %	0 %	4 %	94 %

ANN

Make a neural network without any convolutions that achieves atleast 9 0 % on the validation test. It will be possible with the techniques you have used above.

In [94]:

```
## Set up the model architecture
inputs = keras.Input(shape= (32,32,1))
#Extend your model here (atleast)
# 32*32=1024
x = layers.Flatten()(inputs)
x = layers.Dense(70, activation=tf.nn.relu)(x)
x = layers.Dropout(.5)(x)
x = layers.Dense(20, activation=tf.nn.relu)(x)
#x = layers.Dropout(.2)(x)
outputs = layers.Dense(5, activation="softmax")(x)
```

In [95…

```
## Define the model
ann_model = keras.Model(inputs=inputs, outputs=outputs, name="ann_M
```

In […

```
## Compile the model
ann_model.compile(optimizer=keras.optimizers.Adam(), loss='categorical
ann_model.summary()
```

```
Model: "ann_Model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_16 (InputLayer)       [(None, 32, 32, 1)]       0
_________________________________________________________________
flatten_15 (Flatten)        (None, 1024)              0
_________________________________________________________________
dense_38 (Dense)            (None, 70)                71750
_________________________________________________________________
dropout_8 (Dropout)         (None, 70)                0
_________________________________________________________________
dense_39 (Dense)            (None, 20)                1420
_________________________________________________________________
dense_40 (Dense)            (None, 5)                 105
=================================================================
Total params: 73,275
Trainable params: 73,275
Non-trainable params: 0
_________________________________________________________________
```

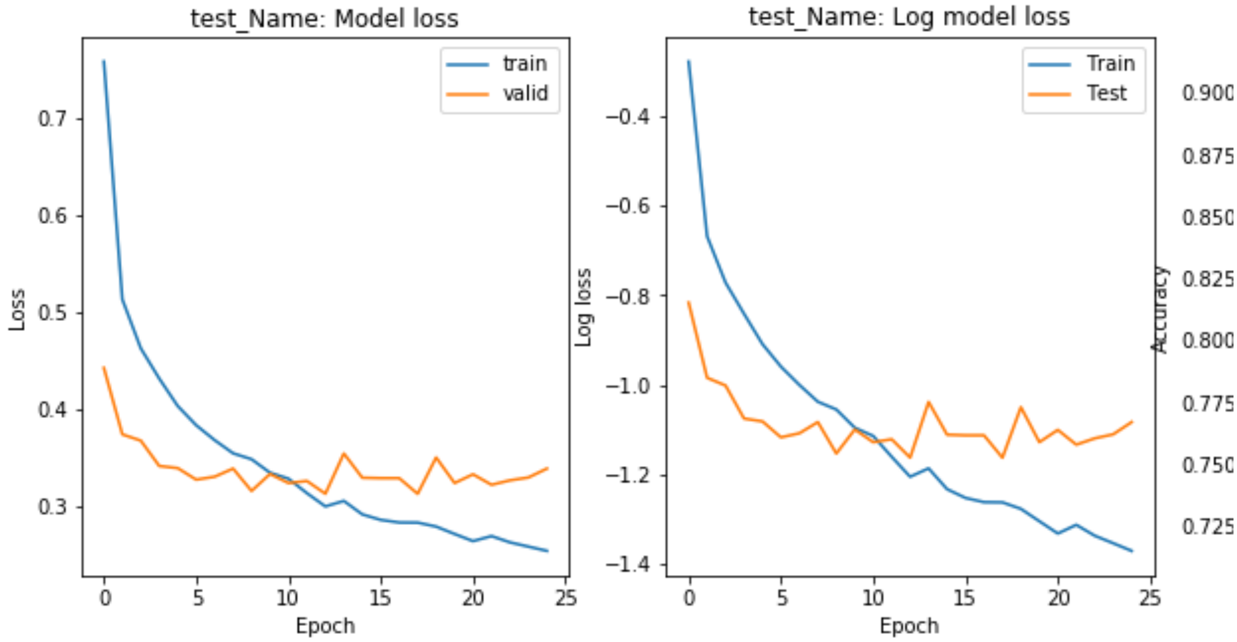
In [97]:

```
## Actually train model
epochs = 25
history = ann_model.fit_generator(generator=train_generator,
                    steps_per_epoch= train_steps,
                    validation_data= valid_generator,
                    validation_steps= validation_steps,
                    epochs= epochs
        )
```

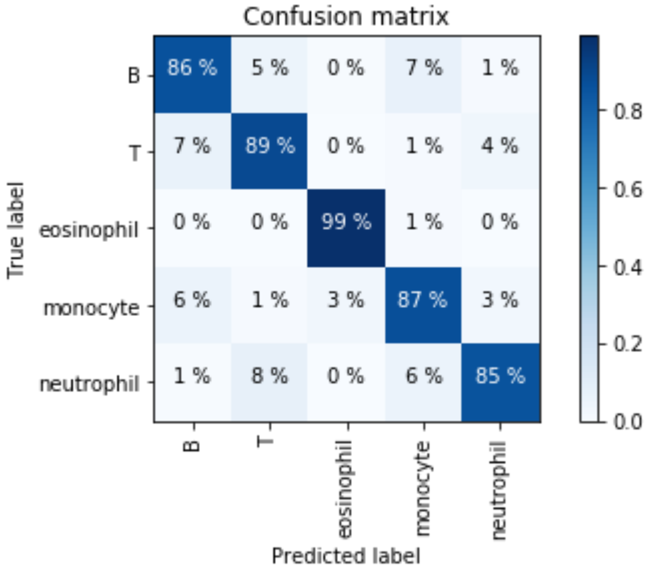
```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1871 steps, validate for 467 steps
Epoch 1/25
1871/1871 [==============================] - 25s 13ms/step - loss: 0.7574
- accuracy: 0.7149 - val_loss: 0.4422 - val_accuracy: 0.8295
Epoch 2/25
1871/1871 [==============================] - 14s 8ms/step - loss: 0.5127 -
accuracy: 0.8149 - val_loss: 0.3739 - val_accuracy: 0.8672
Epoch 3/25
1871/1871 [==============================] - 15s 8ms/step - loss: 0.4624 -
accuracy: 0.8310 - val_loss: 0.3675 - val_accuracy: 0.8710
Epoch 4/25
1871/1871 [==============================] - 15s 8ms/step - loss: 0.4313 -
accuracy: 0.8446 - val_loss: 0.3414 - val_accuracy: 0.8812
Epoch 5/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.4030 -
accuracy: 0.8568 - val_loss: 0.3392 - val_accuracy: 0.8870
Epoch 6/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.3833 -
accuracy: 0.8616 - val_loss: 0.3274 - val_accuracy: 0.8884
Epoch 7/25
1871/1871 [==============================] - 15s 8ms/step - loss: 0.3680 -
accuracy: 0.8696 - val_loss: 0.3303 - val_accuracy: 0.8897
Epoch 8/25
1871/1871 [==============================] - 15s 8ms/step - loss: 0.3544 -
accuracy: 0.8717 - val_loss: 0.3387 - val_accuracy: 0.8887
Epoch 9/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.3483 -
accuracy: 0.8735 - val_loss: 0.3157 - val_accuracy: 0.8943
Epoch 10/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.3344 -
accuracy: 0.8780 - val_loss: 0.3331 - val_accuracy: 0.8878
Epoch 11/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.3283 -
accuracy: 0.8812 - val_loss: 0.3238 - val_accuracy: 0.8935
Epoch 12/25
1871/1871 [==============================] - 16s 8ms/step - loss: 0.3134 -
accuracy: 0.8895 - val_loss: 0.3259 - val_accuracy: 0.8881
Epoch 13/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2998 -
accuracy: 0.8956 - val_loss: 0.3128 - val_accuracy: 0.8943
Epoch 14/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.3054 -
accuracy: 0.8912 - val_loss: 0.3542 - val_accuracy: 0.8771
Epoch 15/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2916 -
accuracy: 0.8979 - val_loss: 0.3293 - val_accuracy: 0.8892
Epoch 16/25
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2859 -
accuracy: 0.8960 - val_loss: 0.3289 - val_accuracy: 0.8878
Epoch 17/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2833 -
accuracy: 0.8984 - val_loss: 0.3288 - val_accuracy: 0.8937
Epoch 18/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2831 -
accuracy: 0.8985 - val_loss: 0.3128 - val_accuracy: 0.8994
Epoch 19/25
```

```
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2791 -
accuracy: 0.9015 - val_loss: 0.3501 - val_accuracy: 0.8854
Epoch 20/25
1871/1871 [==============================] - 15s 8ms/step - loss: 0.2715 -
accuracy: 0.9021 - val_loss: 0.3238 - val_accuracy: 0.8932
Epoch 21/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2641 -
accuracy: 0.9091 - val_loss: 0.3328 - val_accuracy: 0.8940
Epoch 22/25
1871/1871 [==============================] - 17s 9ms/step - loss: 0.2693 -
accuracy: 0.9058 - val_loss: 0.3220 - val_accuracy: 0.8924
Epoch 23/25
1871/1871 [==============================] - 19s 10ms/step - loss: 0.2628 -
accuracy: 0.9081 - val_loss: 0.3266 - val_accuracy: 0.8935
Epoch 24/25
1871/1871 [==============================] - 16s 8ms/step - loss: 0.2585 -
accuracy: 0.9127 - val_loss: 0.3296 - val_accuracy: 0.8932
Epoch 25/25
1871/1871 [==============================] - 16s 9ms/step - loss: 0.2541 -
accuracy: 0.9102 - val_loss: 0.3387 - val_accuracy: 0.8911
```

```
In [98…  ## Plot results
         plot_history(history, "test_Name")

         # plot confusion matrix
         cnn_helper.plot_confusion_matrix_from_generator(ann_model, valid_ge
```



```
Accuracy: 0.8907293614747529
```


Confusion matrix

True label	B	T	eosinophil	monocyte	neutrophil
B	86 %	5 %	0 %	7 %	1 %
T	7 %	89 %	0 %	1 %	4 %
eosinophil	0 %	0 %	99 %	1 %	0 %
monocyte	6 %	1 %	3 %	87 %	3 %
neutrophil	1 %	8 %	0 %	6 %	85 %

Optional

Try using different proportions for training, validation and test. How does this affect your results? Why?

```
In [ ]:
```