# Skin Cancer MNIST: HAM10000--Using Resnet 50

In [1]:

```
# Import Default Packages

import os
import shutil

import cv2
import gc
import keras
import numpy as np
import pandas as pd
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.layers import (BatchNormalization, Dense, Dropout, Flatten)
from tensorflow.keras.metrics import categorical_accuracy, top_k_categorical_accuracy
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix, f1_score
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications.resnet50 import ResNet50
```

# Data Exploration

**First check what columns are in the metadata.**

In [2]:

```
metadata = pd.read_csv("/kaggle/input/HAM10000_metadata.csv")
metadata.head()
```

In [3]:

```
# check the proportion of each label
metadata["dx"].value_counts() / metadata.shape[0]
```

In [4]:

```
image_sample = cv2.imread("/kaggle/input/ham10000_images_part_1/ISIC_0027269.jpg")
print(image_sample.shape)
```

In [5]:

```
lesion_id_cnt = metadata["lesion_id"].value_counts()
def check_duplicates(id):

    if lesion_id_cnt[id] > 1:
        return True
    else:
        return False
```

**Images are stored in 2 different folders - part_1 & part_2**

**Thus, we need to mark which folder each specific image is in.**

In [6]:

```
image_folder_1 = "/kaggle/input/ham10000_images_part_1"
image_folder_2 = "/kaggle/input/ham10000_images_part_2"
```

```
metadata["folder"] = 0
metadata.set_index("image_id", drop=False, inplace=True)

for image in os.listdir(image_folder_1):
    image_id = image.split(".")[0]
    metadata.loc[image_id, "folder"] = "1"

for image in os.listdir(image_folder_2):
    image_id = image.split(".")[0]
    metadata.loc[image_id, "folder"] = "2"
```

In [7]:

```
lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
}

metadata['cell_type'] = metadata['dx'].map(lesion_type_dict.get)
metadata['cell_type_idx'] = pd.Categorical(metadata['cell_type']).codes
```

In [8]:

```
metadata.head()
```

In [9]:

```
fig, ax1 = plt.subplots(1, 1, figsize= (10, 5))
metadata['cell_type'].value_counts().plot(kind='bar', ax=ax1)
```

**As it can be seen, the data is highly imbalanced. Thus data pre-processing is required.**

# Data Pre-processing

**Undersampling nv class**

In [10]:

```
df_nv = metadata[metadata['dx'] == 'nv']
df_not_nv = metadata[metadata['dx'] != 'nv']
```

In [11]:

```
from sklearn.utils import shuffle

df_nv = shuffle(df_nv)
```

In [12]:

```
df_nv.count()
```

In [13]:

```
df_nv = df_nv.head(1250)
```

In [14]:

```
df_nv = df_nv.reset_index(drop = True)
```

In [15]:

```
dataset_final = pd.concat([df_nv, df_not_nv])
```

In [16]:

```
dataset_final.shape
```

In [17]:

```
dataset_final['cell_type'].value_counts()
```

In [18]:

```
dataset_final.head()
```

In [19]:

```
fig, ax1 = plt.subplots(1, 1, figsize= (10, 5))
dataset_final['cell_type'].value_counts().plot(kind='bar', ax=ax1)
```

**undersampling technique is used to decrease the amount of the majority class, nv.**

# Split the Data into train / validation / test set

**Now I need to split the data into training/validation/test datasets. I split the data in a 80%-10%-10% fashion.**

In [20]:

```
X=dataset_final.drop(columns=['cell_type_idx'],axis=1)
Y_cat=dataset_final['cell_type_idx']
```

In [21]:

```
x_train, x_test, y_train, y_test = train_test_split(X, Y_cat, test_size=0.2, random_stat
e=42)
```

In [22]:

```
print("Train: " + str(x_train.shape[0] / dataset_final.shape[0]))
print("Test: " + str(x_test.shape[0] / dataset_final.shape[0]))

print("Train: " + str(y_train.shape[0] / dataset_final.shape[0]))
print("Test: " + str(y_test.shape[0] / dataset_final.shape[0]))
```

In [23]:

```
from keras.utils.np_utils import to_categorical # used for converting labels to one-hot-e
ncoding
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding

# Perform one-hot encoding on the labels
y_train = to_categorical(y_train, num_classes = 7)
y_test = to_categorical(y_test, num_classes = 7)
```

In [24]:

```
x_test, x_validate, y_test, y_validate = train_test_split(x_test, y_test, test_size = 0.
5, random_state = 2)
```

In [25]:

```
print("x Train: " + str(x_train.shape[0] / dataset_final.shape[0]))
print("x Test: " + str(x_test.shape[0] / dataset_final.shape[0]))
print("x Test: " + str(x_validate.shape[0] / dataset_final.shape[0]))

print("y Train: " + str(y_train.shape[0] / dataset_final.shape[0]))
print("y Test: " + str(y_test.shape[0] / dataset_final.shape[0]))
print("y Test: " + str(y_validate.shape[0] / dataset_final.shape[0]))
```

In [26]:

```
y_test
```

In [27]:

```
base_dir = "base_dir"
os.mkdir(base_dir)

train_dir = os.path.join(base_dir, "image_train")
os.mkdir(train_dir)

val_dir = os.path.join(base_dir, "image_val")
os.mkdir(val_dir)

test_dir = os.path.join(base_dir, "image_test")
os.mkdir(test_dir)
```

In [28]:

```
labels = list(metadata["dx"].unique())

for label in labels:
    label_path_train = os.path.join(train_dir, label)
    os.mkdir(label_path_train)
    label_path_val = os.path.join(val_dir, label)
    os.mkdir(label_path_val)
    label_path_test = os.path.join(test_dir, label)
    os.mkdir(label_path_test)
```

**Copy the images to the new directory.**

In [29]:

```
image_dir = "/kaggle/input/ham10000_images_part_"


for i in range(x_train.shape[0]):
    image_name = x_train["image_id"][i] + ".jpg"
    src_dir = os.path.join(image_dir + x_train["folder"][i], image_name)
    dst_dir = os.path.join(train_dir, x_train["dx"][i], image_name)
    shutil.copyfile(src_dir, dst_dir)

for i in range(x_validate.shape[0]):
    image_name = x_validate["image_id"][i] + ".jpg"
    src_dir = os.path.join(image_dir + x_validate["folder"][i], image_name)
    dst_dir = os.path.join(val_dir, x_validate["dx"][i], image_name)
    shutil.copyfile(src_dir, dst_dir)

for i in range(x_test.shape[0]):
    image_name = x_test["image_id"][i] + ".jpg"
    src_dir = os.path.join(image_dir + x_test["folder"][i], image_name)
    dst_dir = os.path.join(test_dir, x_test["dx"][i], image_name)
    shutil.copyfile(src_dir, dst_dir)
```

In [30]:

```
# check the amount of each label in each dataset before data augmentation
for label in labels:
    print(label + " train: " + str(len(os.listdir(os.path.join(train_dir, label)))))
print("\n")
for label in labels:
    print(label + " val: " + str(len(os.listdir(os.path.join(val_dir, label)))))
print("\n")
for label in labels:
    print(label + " test: " + str(len(os.listdir(os.path.join(test_dir, label)))))
```

In [31]:

```
# Delete the redundant data and collect the RAM.
```

```
del x_train, metadata
gc.collect()
```

## Data Agumentation

In [32]:

```
data_gen_param = {
    "rotation_range": 180,
    "width_shift_range": 0.1,
    "height_shift_range": 0.1,
    "zoom_range": 0.1,
    "horizontal_flip": True,
    "vertical_flip": True
}
data_generator = ImageDataGenerator(**data_gen_param)
num_images_each_label = 1000

aug_dir = os.path.join(base_dir, "aug_dir")
os.mkdir(aug_dir)

for label in labels:

    img_dir = os.path.join(aug_dir, "aug_img")
    os.mkdir(img_dir)

    src_dir_label = os.path.join(train_dir, label)
    for image_name in os.listdir(src_dir_label):
        shutil.copy(os.path.join(src_dir_label, image_name), os.path.join(img_dir, image
_name))

    batch_size = 35
    data_flow_param = {
        "directory": aug_dir,
        "color_mode": "rgb",
        "batch_size": batch_size,
        "shuffle": True,
        "save_to_dir": os.path.join(train_dir, label),
        "save_format": "jpg"
    }
    aug_data_gen = data_generator.flow_from_directory(**data_flow_param)

    num_img_aug = num_images_each_label - len(os.listdir(os.path.join(train_dir, label))
)
    num_batch = int(num_img_aug / batch_size)

    for i in range(0, num_batch):
        next(aug_data_gen)

    shutil.rmtree(img_dir)
```

**Now check if the data is balanced.**

In [33]:

```
for label in labels:
    print(label + " train: " + str(len(os.listdir(os.path.join(train_dir, label)))))
print("\n")
for label in labels:
    print(label + " val: " + str(len(os.listdir(os.path.join(val_dir, label)))))
```

In [34]:

```
# after data augmentation, the training data for each label are around 1,000
for label in labels:
    print(label + " train: " + str(len(os.listdir(os.path.join(train_dir, label)))))
```

## Define Model

## Define Model

In [35]:

```
# three models are defined
# model_64 : 64 * 64 * 3       -> model_low_resolution
# model_128 : 128 * 128 * 3    -> model_mid_resolution
# model_256 : 256 * 256 * 3    -> model_high_resolution

model_64 = ResNet50(include_top=True, weights=None, input_shape=(64, 64, 3), pooling=max
, classes=7)
model_128 = ResNet50(include_top=True, weights=None, input_shape=(128, 128, 3), pooling=
max, classes=7)
model_256 = ResNet50(include_top=True, weights=None, input_shape=(256, 256, 3), pooling=
max, classes=7)
```

In [36]:

```
model_low_resolution = Sequential()
model_low_resolution.add(model_64)


model_mid_resolution = Sequential()
model_mid_resolution.add(model_128)

model_high_resolution = Sequential()
model_high_resolution.add(model_256)
```

In [37]:

```
model_low_resolution.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=
['acc'])
model_mid_resolution.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=
['acc'])
model_high_resolution.compile(loss='categorical_crossentropy', optimizer='Adam', metrics
=['acc'])
```

# Training

In [38]:

```
x_train, x_test, y_train, y_test = train_test_split(X, Y_cat, test_size=0.2, random_stat
e=42)
x_test, x_validate, y_test, y_validate = train_test_split(x_test, y_test, test_size = 0.
5, random_state = 2)
```

In [39]:

```
# checkpoint

filepath = "model.h5"

checkpoint_param = {
    "filepath": filepath,
    "monitor": "val_categorical_accuracy",
    "verbose": 1,
    "save_best_only": True,
    "mode": "max"
}
checkpoint = ModelCheckpoint(**checkpoint_param)

lr_decay_params = {
    "monitor": "val_loss",
    "factor": 0.5,
    "patience": 2,
    "min_lr": 1e-5
}
lr_decay = ReduceLROnPlateau(**lr_decay_params)
```

```
early_stopping = EarlyStopping(monitor="val_loss", patience=4, verbose=1)
```

## Training > model_low_resolution

In [40]:

```
# generator

IMAGE_SHAPE = (64, 64, 3)
data_gen_param = {
    "samplewise_center": True,
    "samplewise_std_normalization": True,
    "rotation_range": 180,
    "width_shift_range": 0.1,
    "height_shift_range": 0.1,
    "zoom_range": 0.1,
    "horizontal_flip": True,
    "vertical_flip": True,
    "rescale": 1.0 / 255
}
data_generator = ImageDataGenerator(**data_gen_param)

train_flow_param = {
    "directory": train_dir,
    "batch_size": batch_size,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": True
}
train_flow = data_generator.flow_from_directory(**train_flow_param)

val_flow_param = {
    "directory": val_dir,
    "batch_size": batch_size,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": False
}
val_flow = data_generator.flow_from_directory(**val_flow_param)

test_flow_param = {
    "directory": test_dir,
    "batch_size": 1,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": False
}
test_flow = data_generator.flow_from_directory(**test_flow_param)
```

In [41]:

```
fit_params = {
    "generator": train_flow,
    "steps_per_epoch": x_train.shape[0] // batch_size,
    "epochs": 20,
    "verbose": 1,
    "validation_data": val_flow,
    "validation_steps": x_validate.shape[0] // batch_size,
    "callbacks": [checkpoint, lr_decay, early_stopping]
}
print("Training the model...")

history_low = model_low_resolution.fit_generator(**fit_params)
print("Done!")
```

In [42]:

```
loss = history_low.history['loss']
val_loss = history_low.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
```

```
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

In [43]:

```
acc = history_low.history['acc']
val_acc = history_low.history['val_acc']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

# Evaluation > model_low_resolution

**See the accuracies and F1 scores on validation and test sets.**

In [44]:

```
_, val_acc = model_low_resolution.evaluate_generator(val_flow, steps=len(val_flow))
y_val_true = val_flow.classes
y_val_pred = np.argmax(model_low_resolution.predict_generator(val_flow, steps=len(val_fl
ow)), axis=1)
val_f1_score = f1_score(y_val_true, y_val_pred, average="micro")

print("Validation accuracy: {:.4f}".format(val_acc))
print("Validation F1 score: {:.4f}".format(val_f1_score))
```

In [45]:

```
_, test_acc = model_low_resolution.evaluate_generator(test_flow, steps=len(test_flow))
y_test_true = test_flow.classes
y_test_pred = np.argmax(model_low_resolution.predict_generator(test_flow, steps=len(test
_flow)), axis=1)
test_f1_score = f1_score(y_test_true, y_test_pred, average="micro")

print("Test accuracy: {:.4f}".format(test_acc))
print("Test F1 score: {:.4f}".format(test_f1_score))
```

**Track the performance on each epoch.**

In [46]:

```
loss_train = history_low.history["loss"]
acc_train = history_low.history["acc"]
loss_val = history_low.history["val_loss"]
acc_val = history_low.history["val_acc"]
epochs = np.arange(1, len(loss_train) + 1)
```

In [47]:

```
plt.plot(epochs, acc_train, "bo", label="Training acc")
plt.plot(epochs, acc_val, "b", label="Validation acc")
plt.title("Accuracy for low resolution model")
plt.legend()
plt.show()
```

In [48]:

```
plt.plot(epochs, loss_train, "bo", label="Training loss")
plt.plot(epochs, loss_val, "b", label="Validation loss")
plt.title("Losses")
```

```
plt.legend()
plt.show()
```

In [49]:

```
conf_mat = confusion_matrix(y_test_true, y_test_pred)
conf_mat = conf_mat.astype('float') / conf_mat.sum(axis=1)[:, np.newaxis]
plt.imshow(conf_mat, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Image resolution 64 * 64 * 3")
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.ylabel("True label")
plt.xlabel("Prediction label")

fmt = '.2f'
thresh = 1
for i in range(conf_mat.shape[0]):
    for j in range(conf_mat.shape[1]):
        plt.text(j, i, format(conf_mat[i, j], fmt),
                 ha="center", va="center", color="white" if conf_mat[i, j] > thresh else
"black")  #horizontalalignment

plt.tight_layout()
```

**Delete the image folder so that it won't output that many files when committed.**

# ROC > model_low_resolution

In [50]:

```
predictions = model_low_resolution.predict_generator(test_flow, steps=len(x_validate), v
erbose=1)
```

In [51]:

```
predictions.shape
```

In [52]:

```
test_flow.class_indices
```

In [53]:

```
test_labels = test_flow.classes
```

In [54]:

```
test_labels.shape
```

In [55]:

```
lable_onehot = np.zeros([len(x_validate),7],dtype=np.int)
for i in range(len(test_labels)):
    for t in range(7):
        if(test_labels[i]==t):
            lable_onehot[i][t] = 1
        else:
            lable_onehot[i][t] = 0

scores_val = predictions
```

In [56]:

```
lable_onehot
```

In [57]:

```python
import sklearn
from sklearn import metrics

fpr, tpr, thresholds = metrics.roc_curve(lable_onehot.ravel(),scores_val.ravel())
auc = metrics.auc(fpr, tpr)
plt.plot(fpr, tpr, c = 'b', lw = 2, alpha = 0.7, label = u'front view, AUC=%.5f' % auc)
plt.title('InceptionResNetV2')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
```

In [58]:

```python
#Compute ROC curve for each classes
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(7):
    fpr[i], tpr[i], _ = metrics.roc_curve(lable_onehot[:, i], scores_val[:, i])
    roc_auc[i] = metrics.auc(fpr[i], tpr[i])


# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = metrics.roc_curve(lable_onehot.ravel(), scores_val.ravel
())
roc_auc["micro"] = metrics.auc(fpr["micro"], tpr["micro"])
```

In [59]:

```python
roc_auc
```

In [60]:

```python
# Plot all ROC curves
import itertools

lw=2
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (AUC = {0:0.3f})'
               ''.format(roc_auc["micro"]),
         color='black', linestyle='--', linewidth=2)

colors = itertools.cycle(['blue', 'green', 'red','yellow', 'magenta', 'cyan','deeppink']
)
class_labels = {0:'akiec', 1:'bcc', 2:'bkl', 3:'df', 4:'mel',5:'nv', 6:'vasc'}
# colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(7), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (AUC = {1:0.3f})'
             ''.format(class_labels[i], roc_auc[i]))

#ROC curves of InceptionResNetV2 for each classes and micro-average
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Image resolution 64 * 64 * 3')
plt.legend(loc="lower right")
plt.show()
```

In [61]:

```python
# Plot all ROC curves
import itertools

lw=2
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve low model(AUC = {0:0.3f})'
               ''.format(roc_auc["micro"]),
```

```
         color='black', linewidth=2)

# for i, color in zip(range(7), colors):
#     plt.plot(fpr[i], tpr[i], color=color, lw=lw,
#              label='ROC curve of class {0} (AUC = {1:0.3f})'
#              ''.format(class_labels[i], roc_auc[i]))

#ROC curves of InceptionResNetV2 for each classes and micro-average
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Image resolution 64 * 64 * 3')
plt.legend(loc="lower right")
plt.show()
```

## Training > model_mid_resolution

In [82]:

```
# generator

IMAGE_SHAPE = (128, 128, 3)
data_gen_param = {
    "samplewise_center": True,
    "samplewise_std_normalization": True,
    "rotation_range": 180,
    "width_shift_range": 0.1,
    "height_shift_range": 0.1,
    "zoom_range": 0.1,
    "horizontal_flip": True,
    "vertical_flip": True,
    "rescale": 1.0 / 255
}
data_generator = ImageDataGenerator(**data_gen_param)

train_flow_param = {
    "directory": train_dir,
    "batch_size": batch_size,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": True
}
train_flow = data_generator.flow_from_directory(**train_flow_param)

val_flow_param = {
    "directory": val_dir,
    "batch_size": batch_size,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": False
}
val_flow = data_generator.flow_from_directory(**val_flow_param)

test_flow_param = {
    "directory": test_dir,
    "batch_size": 1,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": False
}
test_flow = data_generator.flow_from_directory(**test_flow_param)
```

In [63]:

```
fit_params = {
    "generator": train_flow,
    "steps_per_epoch": x_train.shape[0] // batch_size,
    "epochs": 20,
    "verbose": 1,
```

```
    "validation_data": val_flow,
    "validation_steps": x_validate.shape[0] // batch_size,
    "callbacks": [checkpoint, lr_decay, early_stopping]
}
print("Training the model...")

history_mid = model_mid_resolution.fit_generator(**fit_params)
print("Done!")
```

# Evaluation > model_mid_resolution

In [64]:

```
_, val_acc = model_mid_resolution.evaluate_generator(val_flow, steps=len(val_flow))
y_val_true = val_flow.classes
y_val_pred = np.argmax(model_mid_resolution.predict_generator(val_flow, steps=len(val_fl
ow)), axis=1)
val_f1_score = f1_score(y_val_true, y_val_pred, average="micro")

print("Validation accuracy: {:.4f}".format(val_acc))
print("Validation F1 score: {:.4f}".format(val_f1_score))
```

In [65]:

```
_, test_acc = model_mid_resolution.evaluate_generator(test_flow, steps=len(test_flow))
y_test_true = test_flow.classes
y_test_pred = np.argmax(model_mid_resolution.predict_generator(test_flow, steps=len(test
_flow)), axis=1)
test_f1_score = f1_score(y_test_true, y_test_pred, average="micro")

print("Test accuracy: {:.4f}".format(test_acc))
print("Test F1 score: {:.4f}".format(test_f1_score))
```

In [66]:

```
loss_train = history_mid.history["loss"]
acc_train = history_mid.history["acc"]
loss_val = history_mid.history["val_loss"]
acc_val = history_mid.history["val_acc"]
epochs = np.arange(1, len(loss_train) + 1)
```

In [67]:

```
plt.plot(epochs, acc_train, "bo", label="Training acc")
plt.plot(epochs, acc_val, "b", label="Validation acc")
plt.title("Accuracy for medium resolution model")
plt.legend()
plt.show()
```

In [68]:

```
plt.plot(epochs, loss_train, "bo", label="Training loss")
plt.plot(epochs, loss_val, "b", label="Validation loss")
plt.title("Losses")
plt.legend()
plt.show()
```

In [69]:

```
conf_mat = confusion_matrix(y_test_true, y_test_pred)
conf_mat = conf_mat.astype('float') / conf_mat.sum(axis=1)[:, np.newaxis]
plt.imshow(conf_mat, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Image resolution 128 * 128 * 3")
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.ylabel("True label")
plt.xlabel("Prediction label")
```

```
fmt = '.2f'
thresh = 1
for i in range(conf_mat.shape[0]):
    for j in range(conf_mat.shape[1]):
        plt.text(j, i, format(conf_mat[i, j], fmt),
                 ha="center", va="center", color="white" if conf_mat[i, j] > thresh else
"black")  #horizontalalignment

plt.tight_layout()
```

# ROC > model_mid_resolution

In [83]:

```
predictions = model_mid_resolution.predict_generator(test_flow, steps=len(x_validate), v
erbose=1)
```

In [85]:

```
lable_onehot = np.zeros([len(x_validate),7],dtype=np.int)
for i in range(len(test_labels)):
    for t in range(7):
        if(test_labels[i]==t):
            lable_onehot[i][t] = 1
        else:
            lable_onehot[i][t] = 0
# print(lable_onehot)

scores_val = predictions
```

In [86]:

```
import sklearn
from sklearn import metrics

fpr, tpr, thresholds = metrics.roc_curve(lable_onehot.ravel(),scores_val.ravel())
auc = metrics.auc(fpr, tpr)
plt.plot(fpr, tpr, c = 'b', lw = 2, alpha = 0.7, label = u'front view, AUC=%.5f' % auc)
plt.title('InceptionResNetV2')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
```

In [87]:

```
#Compute ROC curve for each classes
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(7):
    fpr[i], tpr[i], _ = metrics.roc_curve(lable_onehot[:, i], scores_val[:, i])
    roc_auc[i] = metrics.auc(fpr[i], tpr[i])


# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = metrics.roc_curve(lable_onehot.ravel(), scores_val.ravel
())
roc_auc["micro"] = metrics.auc(fpr["micro"], tpr["micro"])
```

In [88]:

```
roc_auc
```

In [89]:

```
# Plot all ROC curves
lw=2
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
```

```
          label='micro-average ROC curve (AUC = {0:0.3f})'
                ''.format(roc_auc["micro"]),
          color='black', linestyle='--', linewidth=2)

colors = itertools.cycle(['blue', 'green', 'red','yellow', 'magenta', 'cyan','deeppink']
)
class_labels = {0:'akiec', 1:'bcc', 2:'bkl', 3:'df', 4:'mel',5:'nv', 6:'vasc'}
# colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(7), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (AUC = {1:0.3f})'
             ''.format(class_labels[i], roc_auc[i]))

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Image resolution 128 * 128 * 3')
plt.legend(loc="lower right")
plt.show()
```

In [91]:

```
# Plot all ROC curves
import itertools

lw=2
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
          label='micro-average ROC curve mideum model(AUC = {0:0.3f})'
                ''.format(roc_auc["micro"]),
          color='red', linewidth=2)

# for i, color in zip(range(7), colors):
#     plt.plot(fpr[i], tpr[i], color=color, lw=lw,
#              label='ROC curve of class {0} (AUC = {1:0.3f})'
#              ''.format(class_labels[i], roc_auc[i]))

#ROC curves of InceptionResNetV2 for each classes and micro-average
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Image resolution 128 * 128 * 3')
plt.legend(loc="lower right")
plt.show()
```

# Training > model_high_resolution

In [92]:

```
IMAGE_SHAPE = (256, 256, 3)
data_gen_param = {
    "samplewise_center": True,
    "samplewise_std_normalization": True,
    "rotation_range": 180,
    "width_shift_range": 0.1,
    "height_shift_range": 0.1,
    "zoom_range": 0.1,
    "horizontal_flip": True,
    "vertical_flip": True,
    "rescale": 1.0 / 255
}
data_generator = ImageDataGenerator(**data_gen_param)

train_flow_param = {
    "directory": train_dir,
```

```
    "batch_size": batch_size,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": True
}
train_flow = data_generator.flow_from_directory(**train_flow_param)

val_flow_param = {
    "directory": val_dir,
    "batch_size": batch_size,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": False
}
val_flow = data_generator.flow_from_directory(**val_flow_param)

test_flow_param = {
    "directory": test_dir,
    "batch_size": 1,
    "target_size": IMAGE_SHAPE[:2],
    "shuffle": False
}
test_flow = data_generator.flow_from_directory(**test_flow_param)
```

In [78]:

```
fit_params = {
    "generator": train_flow,
    "steps_per_epoch": x_train.shape[0] // batch_size,
    "epochs": 20,
    "verbose": 1,
    "validation_data": val_flow,
    "validation_steps": x_validate.shape[0] // batch_size,
    "callbacks": [checkpoint, lr_decay, early_stopping]
}
print("Training the model...")

history_high = model_high_resolution.fit_generator(**fit_params)
print("Done!")
```

# Evaluation > model_high_resolution

In [93]:

```
_, val_acc = model_high_resolution.evaluate_generator(val_flow, steps=len(val_flow))
y_val_true = val_flow.classes
y_val_pred = np.argmax(model_high_resolution.predict_generator(val_flow, steps=len(val_f
low)), axis=1)
val_f1_score = f1_score(y_val_true, y_val_pred, average="micro")

print("Validation accuracy: {:.4f}".format(val_acc))
print("Validation F1 score: {:.4f}".format(val_f1_score))
```

In [94]:

```
_, test_acc = model_high_resolution.evaluate_generator(test_flow, steps=len(test_flow))
y_test_true = test_flow.classes
y_test_pred = np.argmax(model_high_resolution.predict_generator(test_flow, steps=len(tes
t_flow)), axis=1)
test_f1_score = f1_score(y_test_true, y_test_pred, average="micro")

print("Test accuracy: {:.4f}".format(test_acc))
print("Test F1 score: {:.4f}".format(test_f1_score))
```

In [95]:

```
loss_train = history_high.history["loss"]
acc_train = history_high.history["acc"]
loss_val = history_high.history["val_loss"]
acc_val = history_high.history["val_acc"]
epochs = np.arange(1, len(loss_train) + 1)
```

In [96]:

```
plt.plot(epochs, acc_train, "bo", label="Training acc")
plt.plot(epochs, acc_val, "b", label="Validation acc")
plt.title("Accuracy for high resolution model")
plt.legend()
plt.show()
```

In [97]:

```
plt.plot(epochs, loss_train, "bo", label="Training loss")
plt.plot(epochs, loss_val, "b", label="Validation loss")
plt.title("Losses")
plt.legend()
plt.show()
```

In [98]:

```
conf_mat = confusion_matrix(y_test_true, y_test_pred)
conf_mat = conf_mat.astype('float') / conf_mat.sum(axis=1)[:, np.newaxis]
plt.imshow(conf_mat, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Image resolution 256 * 256 * 3")
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
plt.ylabel("True label")
plt.xlabel("Prediction label")

fmt = '.2f'
thresh = 1
for i in range(conf_mat.shape[0]):
    for j in range(conf_mat.shape[1]):
        plt.text(j, i, format(conf_mat[i, j], fmt),
                 ha="center", va="center", color="white" if conf_mat[i, j] > thresh else
"black")  #horizontalalignment

plt.tight_layout()
```

# ROC > model_high_resolution

In [99]:

```
predictions = model_high_resolution.predict_generator(test_flow, steps=len(x_validate),
verbose=1)
```

In [100]:

```
lable_onehot = np.zeros([len(x_validate),7],dtype=np.int)
for i in range(len(test_labels)):
    for t in range(7):
        if(test_labels[i]==t):
            lable_onehot[i][t] = 1
        else:
            lable_onehot[i][t] = 0
# print(lable_onehot)

scores_val = predictions
```

In [101]:

```
import sklearn
from sklearn import metrics

fpr, tpr, thresholds = metrics.roc_curve(lable_onehot.ravel(),scores_val.ravel())
auc = metrics.auc(fpr, tpr)
plt.plot(fpr, tpr, c = 'b', lw = 2, alpha = 0.7, label = u'front view, AUC=%.5f' % auc)
plt.title('InceptionResNetV2')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
```

In [102]:

```
#Compute ROC curve for each classes 画每一类的ROC
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(7):
    fpr[i], tpr[i], _ = metrics.roc_curve(lable_onehot[:, i], scores_val[:, i])
    roc_auc[i] = metrics.auc(fpr[i], tpr[i])


# Compute micro-average ROC curve and ROC area 计算宏观平均ROC
fpr["micro"], tpr["micro"], _ = metrics.roc_curve(lable_onehot.ravel(), scores_val.ravel
())
roc_auc["micro"] = metrics.auc(fpr["micro"], tpr["micro"])
```

In [103]:

```
roc_auc
```

In [104]:

```
# Plot all ROC curves

import itertools


lw=2
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (AUC = {0:0.3f})'
               ''.format(roc_auc["micro"]),
         color='black', linestyle='--', linewidth=2)

colors = itertools.cycle(['blue', 'green', 'red','yellow', 'magenta', 'cyan','deeppink']
)
class_labels = {0:'akiec', 1:'bcc', 2:'bkl', 3:'df', 4:'mel',5:'nv', 6:'vasc'}
# colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(7), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (AUC = {1:0.3f})'
             ''.format(class_labels[i], roc_auc[i]))

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Image resolution 256 * 256 * 3')
plt.legend(loc="lower right")
plt.show()
```

In [106]:

```
# Plot all ROC curves
import itertools

lw=2
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve high model(AUC = {0:0.3f})'
               ''.format(roc_auc["micro"]),
         color='blue', linewidth=2)

# for i, color in zip(range(7), colors):
#     plt.plot(fpr[i], tpr[i], color=color, lw=lw,
#              label='ROC curve of class {0} (AUC = {1:0.3f})'
#              ''.format(class_labels[i], roc_auc[i]))
```

```
#ROC curves of InceptionResNetV2 for each classes and micro-average
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Image resolution 256 * 256 * 3')
plt.legend(loc="lower right")
plt.show()
```